

# Platformless AI: A Decentralised Protocol for Trustless AI Inference

---

Version 1.5 | May 2026 **Author** | Jules Lai (Founder of Fabstir)

---

## Abstract

---

Platformless AI is a decentralised protocol that enables trustless AI inference, multi-agent orchestration, and agentic AI workflows through a peer-to-peer marketplace of independent compute providers. By combining Ethereum smart contracts, cryptographic proofs, and decentralised storage, the protocol eliminates platform intermediaries while maintaining accountability, privacy, and fair compensation. Beyond chat-based inference, the protocol supports autonomous AI agents — coding assistants, multi-tool workflows, and SaaS AI backends — through both an Anthropic Messages API compatibility layer (Claude Bridge) and an OpenAI Chat Completions API compatibility layer (OpenAI Bridge), enabling any Anthropic-compatible or OpenAI-compatible client to run on decentralised infrastructure. The protocol also supports image generation via host-side diffusion model sidecars, GPU-accelerated video transcoding with end-to-end encryption, HLS adaptive bitrate streaming with per-segment encryption, load balancing across multiple hosts, and experimental multi-agent orchestration via Google's Agent-to-Agent (A2A) protocol (v1.0.0-rc) — decomposing complex goals into task graphs, routing sub-tasks to optimal models across decentralised GPU hosts, and settling payments on-chain with cryptographic proof collection. Platformless AI is, to our knowledge, the first DePIN project to implement the A2A protocol. For privacy-critical workloads, a proof-of-concept confidential-computing path — software-complete and proven end-to-end on GPU hardware behind a mock attestation backend — extends the protocol toward host-blind inference inside Trusted Execution Environments, where even the host operator cannot read inputs or model weights (real hardware attestation pending). This whitepaper describes the technical architecture, economic model, and security mechanisms that enable a truly open AI marketplace where anyone can provide or consume AI services without centralised gatekeepers.

---

## Table of Contents

---

- [1. Introduction](#)
- [2. The Problem: Centralisation in AI](#)
- [3. The Solution: Platformless AI](#)
- [4. Technical Architecture](#)
- [5. Agentic AI and SaaS Infrastructure](#)
- [6. Multi-Agent Orchestration and A2A Protocol \(Experimental\)](#)
- [7. Economic Model](#)
- [8. Security and Privacy](#)
- [9. Dispute Resolution](#)
- [10. Governance](#)

## 1. Introduction

---

The AI revolution has brought unprecedented capabilities to humanity, yet access to these capabilities remains concentrated in the hands of a few large technology companies. These platforms act as gatekeepers, controlling which models users can access, what they can be used for, and at what price. Users have no choice but to trust these intermediaries with their data, their queries, and their dependency on AI-powered workflows.

**Platformless AI** represents a paradigm shift: a protocol, not a platform. Like BitTorrent democratised file sharing by eliminating central servers, Platformless AI democratises AI inference by creating a trustless marketplace where independent compute providers—called **hosts**—offer AI services directly to users without intermediaries.

The protocol leverages:

- **Ethereum smart contracts** for transparent, automated settlement
- **End-to-end encryption** for privacy-preserving inference
- **Cryptographic proofs** for verifiable computation
- **Decentralised storage (S5/Sia)** for immutable audit trails
- **Economic incentives** aligned to ensure honest behaviour

The result is an open, permissionless AI marketplace where hosts compete on price and quality, users retain full control of their data, and trust is established through verifiable code, mathematics, and economic incentives rather than reputation.

Critically, this infrastructure extends beyond simple chat to support **autonomous AI agents**, **image generation**, **GPU-accelerated video transcoding**, and **multi-agent orchestration** (experimental). The Claude Bridge translates the Anthropic Messages API, and the OpenAI Bridge translates the OpenAI Chat Completions API, into the protocol's encrypted WebSocket layer, enabling tools like Claude Code, OpenCode, Continue, and Cursor to run entirely on decentralised GPU hosts. The transcoding pipeline enables end-to-end encrypted video processing across multiple independent hosts with capacity-aware load balancing, including HLS adaptive bitrate streaming where the transcoder produces fMP4 segments with per-segment encryption (preview segments unencrypted for free access, paid segments encrypted with XChaCha20-Poly1305) and the SDK generates M3U8 playlists client-side for playback via hls.js. Since NVIDIA's NVENC encoder is separate hardware from the CUDA cores used for inference, hosts can transcode video and run LLM inference simultaneously without performance degradation, enabling dual revenue streams from the same GPU. The orchestrator package ([@fabstir/orchestrator](#)) adds DAG-based task decomposition, intelligent model routing, and inter-agent collaboration via Google's Agent-to-Agent (A2A) protocol — an emerging standard (~11 months old, v1.0.0-rc) backed by 150+ technology partners and the Linux Foundation. The x402 HTTP payment protocol enables USDC micropayments for inter-agent delegation, running alongside the existing on-chain escrow model. This positions Platformless AI as infrastructure for the next generation of AI-powered applications, including

agentic coding assistants, multi-agent workflows, video platforms, creative AI tools, and SaaS AI backends.

---

## 2. The Problem: Centralisation in AI

---

### 2.1 Platform Lock-in

Today's AI landscape is dominated by vertically integrated platforms that control the entire stack:

Layer	Centralised Control
Models	Platforms decide which models are available
Infrastructure	Users must use platform hardware
Pricing	Platforms set prices unilaterally
Data	All queries flow through platform servers
Policies	Platforms decide acceptable use

This creates **vendor lock-in** where users become dependent on a single provider's ecosystem, pricing, and policies.

### 2.2 Privacy Concerns

When users send prompts to centralised AI services:

- Prompts are transmitted and stored on platform servers
- Conversation history is retained indefinitely
- Data may be used for model training without explicit consent
- Sensitive queries expose business intelligence, personal information, or trade secrets

Users must trust platforms to handle this data responsibly—a trust that has been violated repeatedly across the technology industry.

### 2.3 Censorship and Availability

Centralised platforms can:

- Restrict access to certain models or capabilities
- Impose content policies that prevent legitimate use cases
- Discontinue services without notice
- Deny service based on geography, payment method, or arbitrary criteria

There is no recourse when a platform decides your use case is not welcome.

### 2.4 Economic Inefficiency

Platform economics favour the platform, not participants:

- Compute providers (if any external ones exist) receive a fraction of revenue
- Users pay premium prices that include platform overhead and profit margins
- No competition on pricing—platforms set prices
- Innovation is constrained to what platforms choose to offer

## 2.5 The Trust Problem

The fundamental issue is **forced trust**. Users must trust that:

- Platforms will not misuse their data
- Pricing is fair
- Services will remain available
- Quality will be maintained
- Terms will not change adversely

This trust cannot be verified, only assumed.

---

## 3. The Solution: Platformless AI

---

Platformless AI replaces platform-mediated trust with **cryptographic trust**. Instead of trusting a company, users trust verifiable code, cryptographic mathematics, and carefully aligned economic incentives.

### 3.1 Core Principles

#### **Permissionless Participation**

Anyone can become a host by staking tokens and registering their compute resources. No approval process, no gatekeepers. The only requirement is economic commitment (stake) that aligns host incentives with honest behaviour.

#### **Direct Peer-to-Peer Connections**

Users connect directly to hosts via WebSocket. There is no platform server routing traffic. The SDK can establish direct connections from desktop applications, mobile apps, or server environments. Browser applications use a minimal proxy only to bypass CORS restrictions—the proxy does not see decrypted content.

#### **Blockchain-Based Coordination**

Smart contracts on Ethereum (specifically Base, an L2) handle:

- Host registration and discovery
- Session creation and payment escrow
- Proof verification and settlement
- Model governance and approval

The blockchain provides a single source of truth that all participants can verify.

## End-to-End Encryption

All inference traffic is encrypted using modern cryptographic primitives:

- **ECDH key exchange** for session key establishment
- **XChaCha20-Poly1305** for symmetric encryption
- **ECDSA signatures** for authentication
- **Forward secrecy** via ephemeral session keys

Even if a host's long-term key is compromised, past sessions cannot be decrypted.

## Decentralised Storage

Conversation history, proofs, and audit data are stored on **S5** (backed by the Sia network), a decentralised storage layer that provides:

- Content-addressed immutability
- Censorship resistance
- Distributed redundancy
- Cost-effective archival

Data stored on S5 is encrypted by the SDK before upload—S5 nodes cannot read content.

## 3.2 How It Works: A User's Perspective

1. User authenticates with their Ethereum wallet
2. SDK discovers available hosts from the NodeRegistry contract
3. User selects a host based on price, model support, and reputation
4. User creates a session by depositing funds into the JobMarketplace contract
5. SDK establishes encrypted WebSocket connection to host
6. User sends prompts; host streams responses (all encrypted)
7. Host submits signed proofs of work to the contract
8. Session completes; contract settles payments automatically
9. Conversation is stored encrypted on S5 for user's records

 Session Lifecycle Flow

*Figure 1: Session Lifecycle Flow showing the 9-step process from wallet authentication through to encrypted conversation storage.*

The user never trusts a platform. Trust is established through:

- **Contracts** that enforce rules automatically
- **Encryption** that protects content
- **Proofs** that verify computation
- **Stakes** that deter malicious behaviour

## 3.3 How It Works: A Host's Perspective

1. Host stakes FAB tokens in the NodeRegistry contract
2. Host registers metadata (API URL, supported models, pricing)
3. Host runs inference server (GPU hardware + Fabstir Node software)
4. When sessions are created, host's server accepts WebSocket connections
5. Host processes encrypted prompts, streams encrypted responses
6. Host submits signed proofs of work periodically
7. On session completion, contract releases payment to HostEarnings
8. Host withdraws accumulated earnings at their convenience

Hosts operate as independent businesses. They set their own prices, choose which models to support, and manage their own infrastructure. The protocol handles discovery, payment, and dispute resolution.

---

## 4. Technical Architecture

---

### 4.1 System Overview

 High-Level System Architecture

*Figure 2: High-Level System Architecture showing client-host P2P connections, blockchain coordination layer (Base L2), and decentralised S5 storage.*

The architecture comprises four main layers:

- **Client Layer:** SDK-powered applications connecting via encrypted WebSocket
- **Host Layer:** P2P network of independent GPU providers running inference
- **Blockchain Layer:** Ethereum (Base L2) smart contracts for coordination, escrow, and settlement
- **Storage Layer:** S5/Sia decentralised storage for encrypted conversations and proofs

### 4.2 Smart Contracts

The protocol uses **UUPS upgradeable proxy contracts** for future improvements without disrupting existing sessions.

#### NodeRegistry

Manages host registration, staking, and pricing.

```

struct Node {
    address operator;           // Host's Ethereum address
    uint256 stakedAmount;      // FAB tokens staked (min 1000)
    bool active;               // Currently accepting sessions
    string metadata;           // JSON (name, description, etc.)
    string apiUrl;             // Inference endpoint URL
    bytes32[] models;          // Supported model IDs
    uint256 minPriceNative;    // Minimum price for ETH payments
    uint256 minPriceStable;    // Minimum price for USDC payments
}

```

Key functions:

- `registerNode()` - Stake tokens and register as host
- `unregisterNode()` - Withdraw stake and deregister
- `setModelPricing()` - Set per-model pricing
- `getNodesForModel()` - Discover hosts for a model

### JobMarketplace

Handles session lifecycle and payment escrow.

```

struct SessionJob {
    address depositor;         // User who deposited funds
    address host;              // Host providing inference
    address paymentToken;      // ETH (address(0)) or USDC
    uint256 deposit;           // Escrowed funds
    uint256 pricePerToken;     // Agreed price
    uint256 tokensUsed;        // Tokens consumed
    SessionStatus status;      // Active, Completed, TimedOut
    string conversationCID;    // S5 storage reference
}

```

Key functions:

- `createSessionJob()` - Create session and escrow funds
- `submitProofOfWork()` - Host submits signed proof
- `completeSessionJob()` - Settle and distribute payments

*Note: The contract includes a `DISPUTE_WINDOW` parameter (configurable via upgrade) that delays host-initiated session completion. This window allows users to review session outcomes before final settlement. The optimal duration is being evaluated—currently set to 30 seconds for testing, with production values to be determined based on user experience research.*

### ModelRegistry

Governs which models can be used in the marketplace.

- Community-governed model approval
- Model ID derivation from repository/filename

- Prevents malicious model distribution

## ProofSystem

Verifies host signatures and integrates with zero-knowledge proofs.

- ECDSA signature verification
- STARK proof hook (future: on-chain verification)
- Graceful degradation when proof system unavailable


 Proof-of-Inference Flow

Figure 3: Proof-of-Inference Flow showing the trustless verification mechanism from AI inference through STARK proof generation, S5 storage, and on-chain recording, with the verification path for disputes.

## HostEarnings

Accumulates host payments for gas-efficient batch withdrawals.

- 80% gas reduction vs individual transfers
- Hosts withdraw at their convenience
- Supports multiple payment tokens

## 4.3 SDK Architecture

The SDK ([@fabstir/sdk-core](#)) is the primary interface for client applications. It is organised into focused modules — one per capability (authentication, payments, sessions, host discovery, storage, and so on) — all reached through a single SDK object:

```
// One module per capability, reached via the SDK object
const sdk = new FabstirSDKCore({
  /* config */
});
await sdk.authenticate("privatekey", { privateKey });

// Core capability modules
const auth = sdk.getAuthManager(); // Authentication
const payments = sdk.getPaymentManager(); // Deposits/withdrawals
const sessions = sdk.getSessionManager(); // Session Lifecycle
const hosts = sdk.getHostManager(); // Host discovery
const storage = sdk.getStorageManager(); // S5 storage

// Advanced capability modules
const vectorSearch = sdk.getVectorRAGManager(); // Vector search (RAG)
const models = sdk.getModelManager(); // Model queries
const transcoding = sdk.getTranscodeManager(); // Video/audio transcoding
```

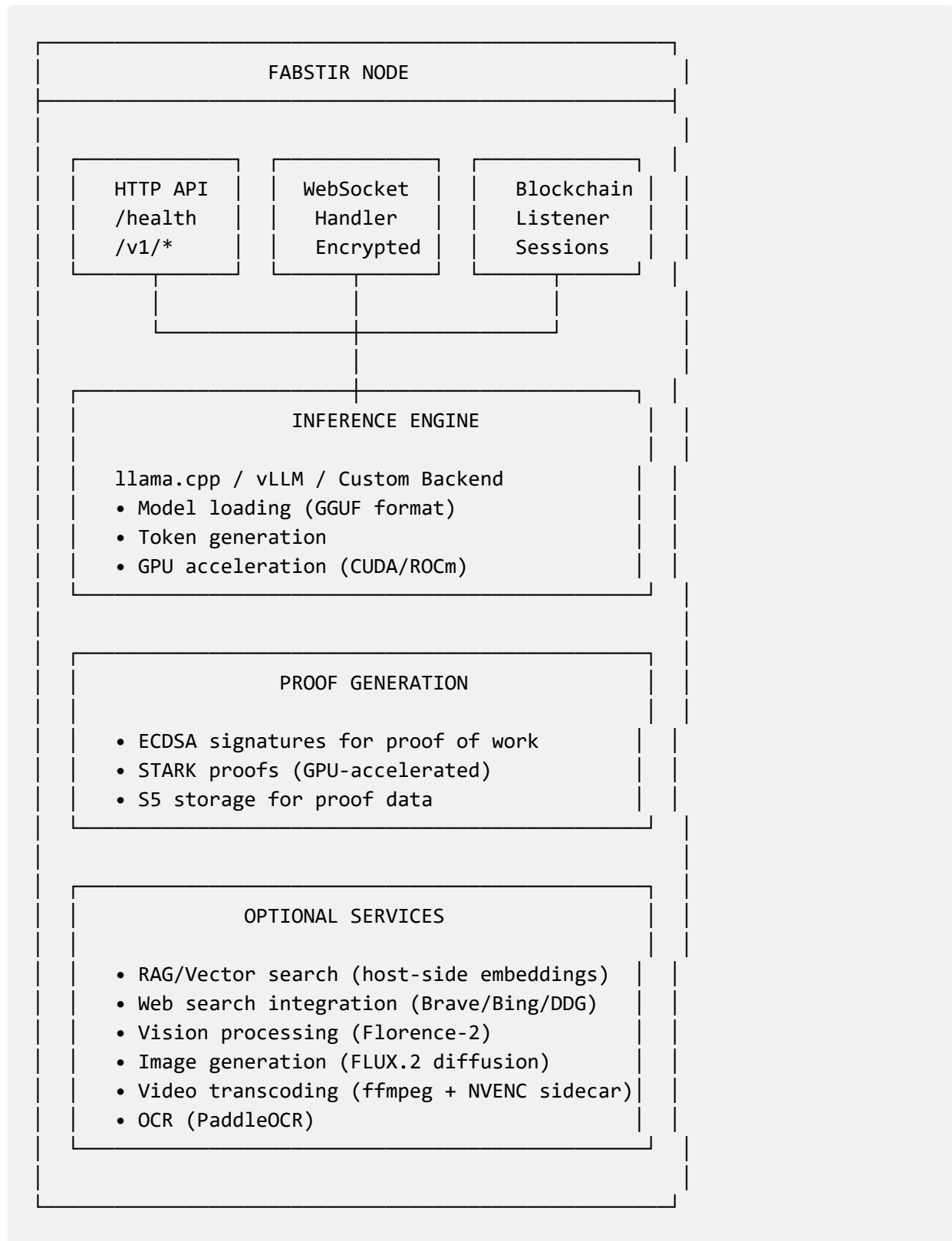
Key SDK features:

- **Browser-compatible:** Zero Node.js dependencies

- **Multi-chain:** Base Sepolia (production), opBNB Testnet (development)
- **Encryption by default:** All sessions encrypted automatically
- **Deterministic S5 seed:** Derived from wallet signature for data sovereignty

## 4.4 Host Node Architecture

The host node (`fabstir-llm-node`) is written in Rust for performance and security. Rust's ownership model and compile-time memory safety guarantees eliminate entire classes of vulnerabilities (buffer overflows, use-after-free, data races) that plague C/C++ inference servers, making it ideal for handling encrypted user data and financial transactions.



## 4.5 Decentralised Storage (S5)

S5 provides decentralised, content-addressed storage backed by the Sia network.

### Why S5?

- **Content addressing:** Data identified by cryptographic hash (CID)
- **Immutability:** Once stored, content cannot be modified
- **Redundancy:** Data distributed across multiple Sia hosts
- **Cost efficiency:** Storage costs fraction of centralised alternatives
- **Path-based API:** Familiar filesystem interface via Enhanced S5.js

### What We Store on S5:

Data Type	Purpose	Encryption
Conversation history	User's chat records	User-controlled
Proof data (proofCID)	Complete proof for verification	Public
Delta CIDs	Incremental proof changes	Public
Vector databases	RAG document embeddings	User-controlled
Source videos	Original video content	User-controlled
Transcoded outputs	Whole-file video (Phase 1)	User-controlled
HLS init segments	fMP4 initialisation segment	Public (z-prefix)
HLS preview segments	Free preview fMP4 segments	Public (z-prefix)
HLS paid segments	Encrypted fMP4 segments	Key-in-CID (u-prefix)
HLS content metadata	Segment CIDs, resolutions, durations	User-controlled
Transcode proof trees	GOP-level Merkle proof trees	Public
Session metadata	Audit trail	Host-signed

### Data Sovereignty:

Each user's S5 identity is derived deterministically from their Ethereum wallet signature. This means:

- Users own their data (not the platform)
- Data follows the user across devices
- No account recovery needed—wallet = identity
- Data cannot be accessed without the wallet



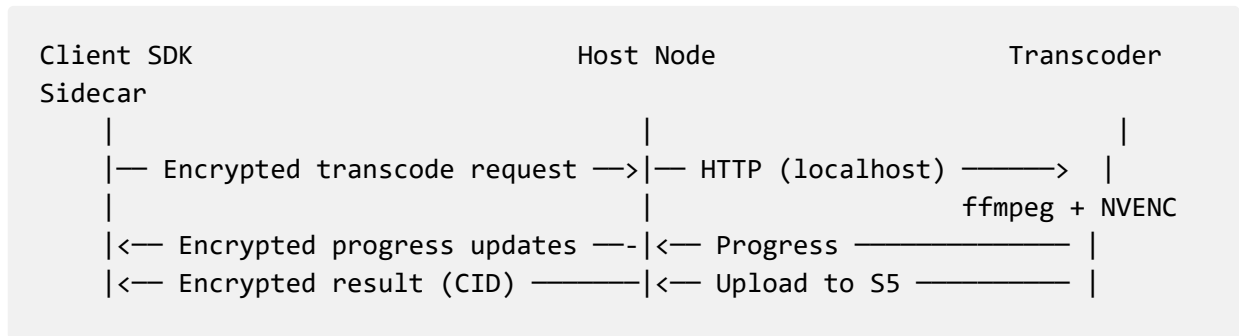
Figure 4: S5 Storage Architecture showing the decentralised content-addressed storage flow from client encryption through Enhanced S5.js to the Sia blockchain network. Developed with Sia Foundation grant funding.

## 4.6 Decentralised Video Transcoding

The protocol extends beyond text-based AI to include **GPU-accelerated video transcoding** — a compute-intensive workload that benefits from the same decentralised, encrypted, and verifiable infrastructure used for inference.

### Architecture

Each host runs a **transcoder sidecar** — an independent Docker container running ffmpeg with NVIDIA NVENC hardware encoding — alongside its main host node. The sidecar communicates with the host node over localhost HTTP, while all client-facing communication flows through the encrypted WebSocket channel.



NVIDIA's NVENC is dedicated encoder circuitry, separate from the CUDA cores used for AI inference. This means a host running LLM inference can simultaneously transcode video without either workload degrading the other — enabling dual revenue streams from the same GPU hardware.

### End-to-End Encrypted Video Pipeline

The transcoding pipeline maintains full encryption throughout:

- Source upload:** Client encrypts video with XChaCha20-Poly1305 and uploads to S5. The decryption key is embedded in the CID — only the CID holder can decrypt.
- Encrypted transit:** The transcode request (source CID, format parameters, job ID) is sent over the E2E encrypted WebSocket channel. Each transcode job establishes its own WebSocket connection with a unique session key via ECDH, enabling multiple concurrent jobs without session interference.
- Host-side processing:** The sidecar downloads the encrypted source from S5, decrypts in memory, transcodes via GPU, and encrypts the output before uploading back to S5. Unencrypted video exists only in memory during processing — never persisted to disk.
- Result delivery:** The encrypted output CID is returned to the client over the encrypted WebSocket. Only the client can decrypt the transcoded video.

No centralised server ever sees the unencrypted video content.

### Capacity-Aware Load Balancing

When multiple hosts support the same transcode model, the SDK's load balancer distributes jobs across them:

- Rank:** Hosts scored by blockchain-derived factors (stake, price, uptime, latency) using the configured selection mode (AUTO, CHEAPEST, RELIABLE, FASTEST).

2. **Filter:** Each host's real-time capacity checked via `GET /v1/transcode/capacity`. The SDK also tracks its own pending submissions per host, so rapid sequential submissions don't all land on the same host before the capacity endpoint updates.
3. **Overflow:** When the preferred host's effective capacity reaches zero, jobs automatically overflow to the next-ranked host. If all hosts are temporarily exhausted, the load balancer retries after a configurable backoff.

This was validated in a stress test with 7 concurrent jobs across two GPU hosts (NVIDIA L40S and RTX Pro 6000 Max-Q), achieving 100% completion with automatic overflow when the preferred host's 3 NVENC slots were saturated.

### GOP-Level Verification

Transcoded output is verified at the **Group of Pictures (GOP)** level using the same STARK proof infrastructure used for inference:

- Each GOP produces: input hash (keccak256), output hash, quality metrics (PSNR in dB, SSIM), and a STARK proof of computational integrity
- GOPs are assembled into a Merkle tree; the root hash is committed on-chain (32 bytes)
- The full proof tree is stored on S5 and can be independently verified by anyone
- Random spot-checks verify individual GOPs against quality thresholds (standard: PSNR >= 38.0 dB)
- On-chain dispute resolution with refund logic if quality violations are demonstrated

### Supported Formats

Codec	Encoder	Status
H.264	h264_nvenc	Production
HEVC/H.265	hevc_nvenc	Production
AV1	av1_nvenc	Production

Output Mode	Description	Status
Whole-file (Phase 1)	Single CID per format, preview via <code>trim_percent</code>	Production
HLS segments (Phase 2)	fMP4 segments per format, preview via <code>previewPercent</code>	Production

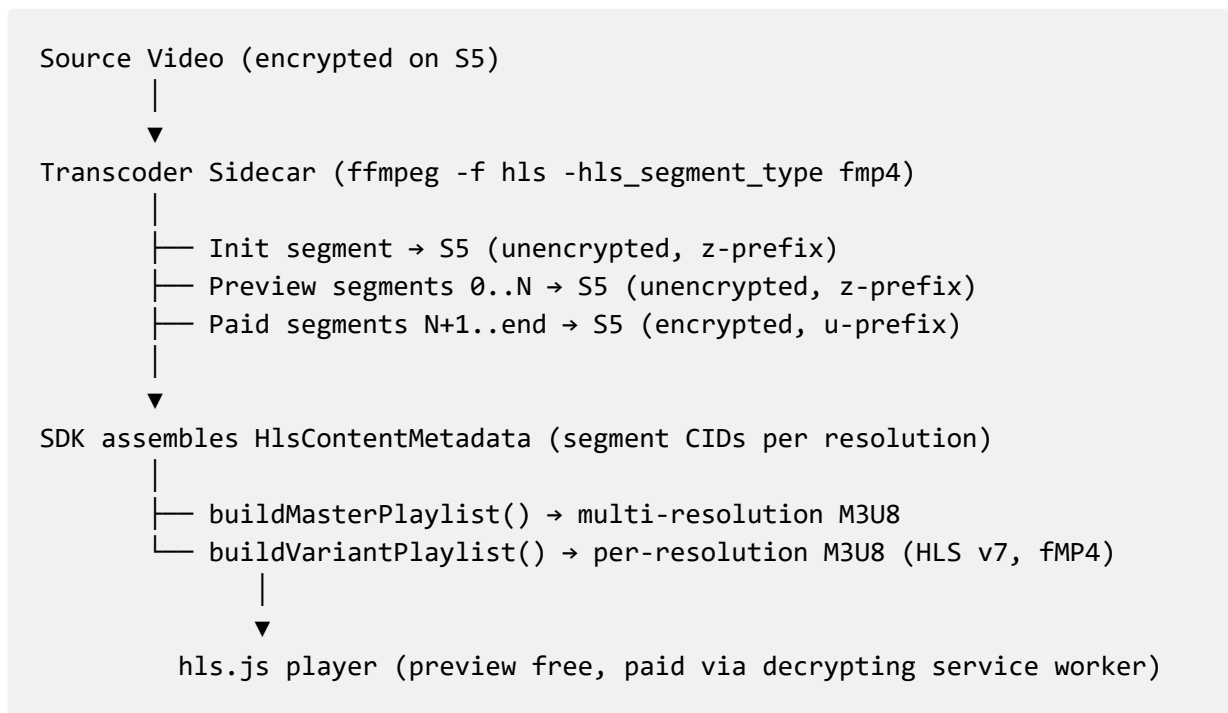
Multiple output resolutions can be produced in a single job (e.g. 720p + 1080p + 4K).

### HLS Adaptive Bitrate Streaming

The protocol supports **HLS adaptive bitrate streaming** with per-segment encryption — enabling white-label streaming platforms where creators upload content, viewers get a free preview, and paid content is cryptographically gated.

When a format includes `hls: true`, the transcoder sidecar produces **fMP4 segments** instead of a single file. Each segment is uploaded individually to S5, with preview segments unencrypted (`z-`

prefix CID, publicly accessible) and paid segments encrypted (u-prefix CID, decryption key embedded in CID). The SDK then generates **M3U8 playlists** client-side from the segment metadata.



This pipeline has been **validated end-to-end**: encrypted source upload to S5, GPU transcoding on NVIDIA L40S (480p/720p/1080p AV1), HLS fMP4 segments with XChaCha20-Poly1305 per-segment encryption stored on decentralised Sia/S5, USDC payment settlement, client-side M3U8 playlist generation, and adaptive bitrate playback via hls.js with a decrypting service worker.

## 5. Agentic AI and SaaS Infrastructure

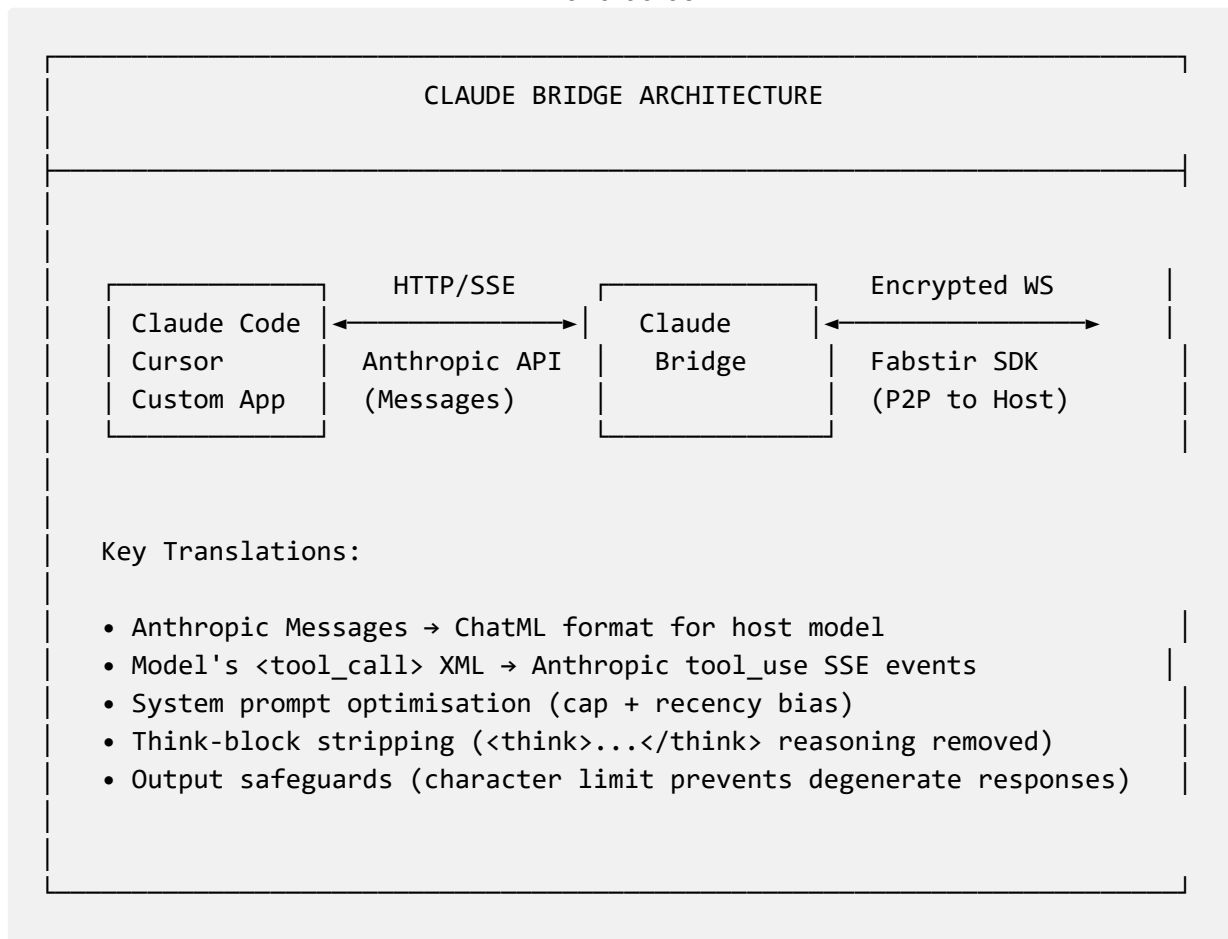
### 5.1 Beyond Chat: The Agentic AI Opportunity

Traditional AI inference is request-response: a user sends a prompt, receives a response. **Agentic AI** is fundamentally different — autonomous agents that plan, use tools, observe results, and iterate. These agents may execute dozens of tool calls per session: reading files, running terminal commands, editing code, searching the web — all without human intervention between steps.

This represents a massive market shift. Agentic coding tools (GitHub Copilot, Cursor, Claude Code, Windsurf) are growing rapidly, with the agentic AI market projected to exceed \$15B by 2027. Every one of these tools currently depends on centralised API providers — creating the same vendor lock-in, privacy, and censorship risks described in Section 2.

### 5.2 Claude Bridge: Anthropic API Compatibility Layer

The **Claude Bridge** ([@fabstir/claude-bridge](https://github.com/fabstir/claude-bridge)) is an API translation layer that enables any Anthropic Messages API-compatible client to use Platformless AI's decentralised infrastructure without modification.

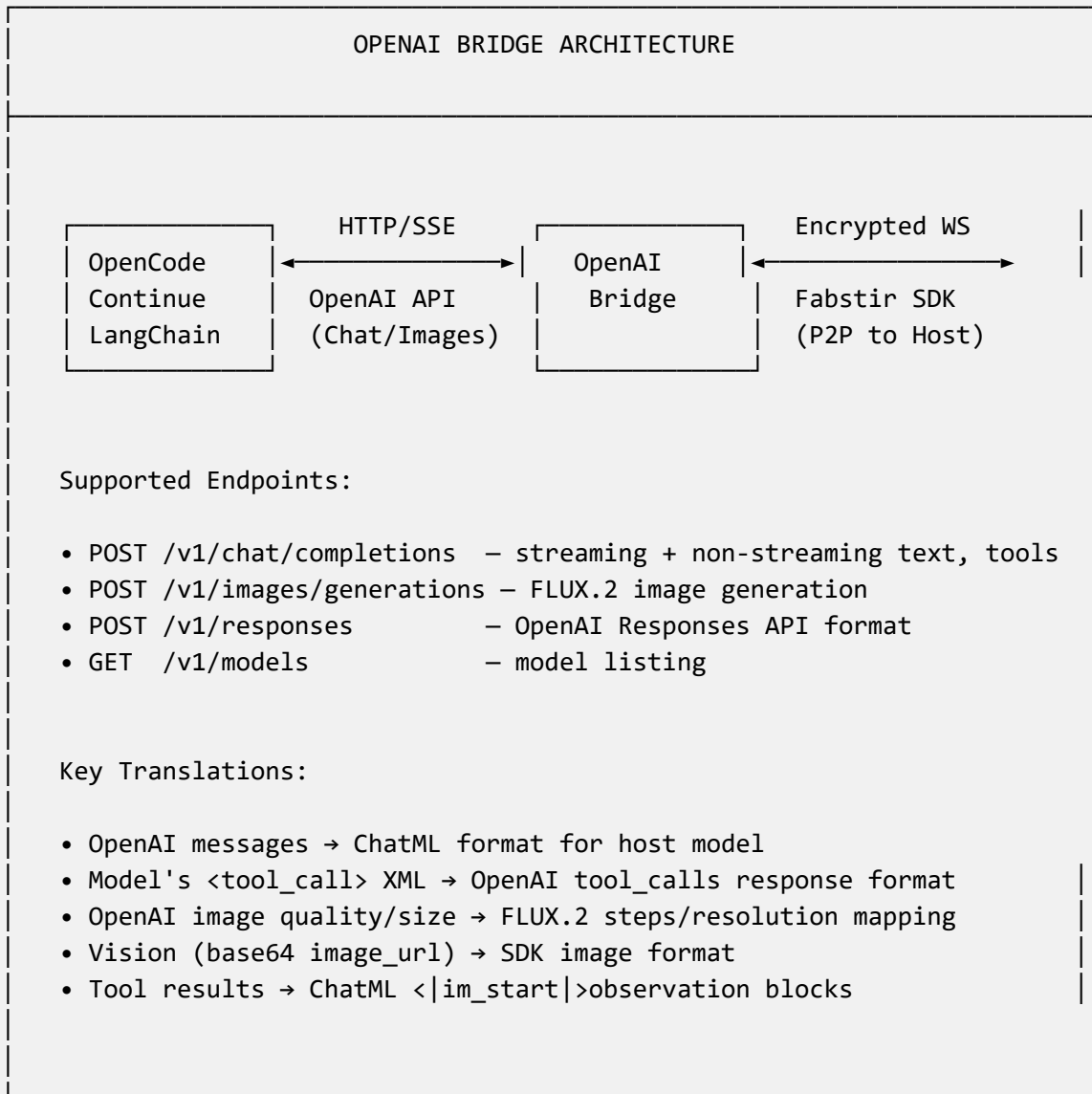


### Technical implementation:

- **SSE streaming:** Full Anthropic streaming protocol — `message_start`, `content_block_start`, `content_block_delta`, `content_block_stop`, `message_delta`, `message_stop`
- **Tool use:** `tool_use` content blocks with `input_json_delta` events, proper `stop_reason: "tool_use"`
- **Multi-turn tool results:** `tool_result` messages serialised into ChatML `<|im_start|>observation` blocks
- **Session management:** Automatic session creation, encrypted WebSocket pooling, session auto-recovery on disconnect
- **Output safeguards:** Character limit based on `max_tokens` prevents degenerate model responses from consuming GPU indefinitely

### 5.3 OpenAI Bridge: OpenAI API Compatibility Layer

The **OpenAI Bridge** ([@fabstir/openai-bridge](#)) extends the same principle to the OpenAI ecosystem. Any application using the OpenAI SDK — including OpenCode, Continue, LangChain, and custom applications — can use Platformless AI's decentralised infrastructure by changing a single environment variable.



**Integration is minimal** — applications using the OpenAI SDK simply set their base URL:

```

# Start the bridge (handles blockchain, encryption, sessions automatically)
npx fabstir-openai-bridge --private-key $KEY --model "repo:file"

# Client configuration
export OPENAI_BASE_URL=http://localhost:3457/v1
  
```

```
import OpenAI from "openai";

const client = new OpenAI({
  baseURL: "http://localhost:3457/v1", // OpenAI Bridge endpoint
  apiKey: "any-key", // Bridge handles auth via wallet
});

// Text inference – works unchanged
const response = await client.chat.completions.create({
  model: "any-model", // Mapped to host model
  messages: [{ role: "user", content: "Hello" }],
  stream: true,
});

// Image generation – maps to FLUX.2 diffusion sidecar
const image = await client.images.generate({
  prompt: "A futuristic cityscape at sunset",
  size: "1024x1024",
  quality: "hd", // Maps to 20 inference steps
});
```

The OpenAI Bridge handles blockchain session creation, host discovery, E2E encryption, and session re-keying automatically. The first request triggers lazy initialisation; subsequent requests reuse the session.

## 5.4 Production Validation

Both bridges have been validated in production. The Claude Bridge has been tested with Claude Code — Anthropic's autonomous coding agent that uses 23 tools. The OpenAI Bridge has been tested with OpenCode running GLM-4.7-Flash on decentralised hosts, including image generation via FLUX.2 diffusion.

**Claude Code (via Claude Bridge)** uses 23 tools:

- **File operations:** Read, Edit, Write, Glob, Grep
- **Terminal:** Bash command execution
- **Web:** WebFetch, WebSearch
- **Planning:** EnterPlanMode, ExitPlanMode, Task delegation
- **Notebook:** NotebookEdit for Jupyter notebooks

**Demonstrated capabilities:**

- Creating a complete React application from scratch (scaffolding, editing, starting dev server)
- Multi-turn tool conversations (read file → edit file → verify changes)
- Background task management (dev servers running while agent continues working)
- Error recovery (agent adapts when tool calls fail)

All tool calls flow through encrypted P2P WebSocket connections to decentralised GPU hosts. The source code, prompts, and responses never touch a centralised platform.

## 5.5 SaaS AI Backend

For companies building AI-powered products, the Claude Bridge provides a **drop-in replacement for the Anthropic API** and the OpenAI Bridge provides a **drop-in replacement for the OpenAI API**:

Feature	Anthropic / OpenAI API	Platformless AI
API compatibility	Native	Full Messages API + Chat Completions API
Data privacy	Trust platform	End-to-end encrypted, no intermediary
Vendor lock-in	Single provider	Any host, any model
Rate limits	Platform-imposed	No artificial limits
Pricing	Platform markup	Direct GPU market rates
Censorship	Platform policies	Permissionless
Compliance	Trust platform	Cryptographic guarantees (GDPR, HIPAA ready)
Availability	Single point of failure	Distributed, multi-host failover
Image generation	Platform-hosted	Decentralised FLUX.2 via OpenAI Bridge

**Integration is minimal** — SaaS applications using the Anthropic or OpenAI SDK simply point their base URL to the bridge endpoint:

```
import Anthropic from "@anthropic-ai/sdk";

const client = new Anthropic({
  baseURL: "http://bridge.example.com/v1", // Claude Bridge endpoint
  apiKey: "any-key", // Bridge handles auth via wallet
});

// Everything else works unchanged
const response = await client.messages.create({
  model: "claude-3-opus-20240229", // Mapped to host model
  max_tokens: 1024,
  messages: [{ role: "user", content: "Hello" }],
});
```

**OpenAI SDK integration:**

```
import OpenAI from "openai";

const client = new OpenAI({
  baseUrl: "http://bridge.example.com/v1", // OpenAI Bridge endpoint
  apiKey: "any-key", // Bridge handles auth via wallet
});

// Chat – works unchanged
const response = await client.chat.completions.create({
  model: "any-model",
  messages: [{ role: "user", content: "Hello" }],
});

// Image generation – works unchanged
const image = await client.images.generate({
  prompt: "A cat astronaut",
  size: "1024x1024",
  quality: "hd",
});
```

## 5.6 Market Implications

The agentic AI and SaaS infrastructure capabilities — with both Anthropic and OpenAI API compatibility — position Platformless AI to capture value from three rapidly growing markets:

1. **Agentic Coding** (\$15B+ by 2027): Enterprise development teams using AI coding assistants need IP protection. Source code sent to centralised APIs is a liability. Platformless AI provides the same capability with end-to-end encryption, supporting both Anthropic-compatible tools (Claude Code, Cursor) and OpenAI-compatible tools (OpenCode, Continue, LangChain).
2. **AI-Powered SaaS** (\$30B+ by 2027): Every SaaS company embedding AI into their product needs reliable, private inference infrastructure. A single API key dependency is a business risk. Platformless AI eliminates this with decentralised, multi-host infrastructure and drop-in compatibility for both major API standards.
3. **AI Agent Orchestration** (emerging): Multi-agent workflows — research agents, planning agents, implementation agents working in teams — require sustained, tool-heavy sessions. Platformless AI's orchestrator with A2A protocol support, DAG-based task decomposition, and x402 inter-agent payments is purpose-built for this market.

---

## 6. Multi-Agent Orchestration and A2A Protocol (Experimental)

*Note: The A2A protocol was introduced by Google in April 2025 and remains at release candidate status (v1.0.0-rc) as of March 2026. Platformless AI's orchestration layer is functional and fully tested (351 unit tests), but should be considered experimental infrastructure — the underlying standard is ~12 months old and may evolve as it approaches stable 1.0. Our implementation will track the specification as it matures.*

### 6.1 The Need for Multi-Agent Orchestration

Single-model inference is insufficient for complex goals. Analysing a quarterly earnings report requires extracting metrics, comparing historical performance, researching market context, and drafting a summary — tasks best handled by different models with different strengths. The [@fabstir/orchestrator](#) package decomposes complex goals into Directed Acyclic Graphs (DAGs) of sub-tasks, routes each to the optimal model across decentralised GPU hosts, executes them with encrypted inference, settles payments on-chain, and collects cryptographic proofs.

## 6.2 Google's Agent-to-Agent (A2A) Protocol

The [Agent-to-Agent \(A2A\) protocol](#) is an open standard introduced by Google in April 2025 for enabling AI agents to discover, communicate, and collaborate — regardless of framework, vendor, or runtime. Transferred to the Linux Foundation in January 2026, it has 150+ technology partners across enterprise, cloud, and AI ecosystems. The protocol is currently at **v1.0.0-rc** (release candidate) — rapidly approaching stable 1.0 but not yet finalised.

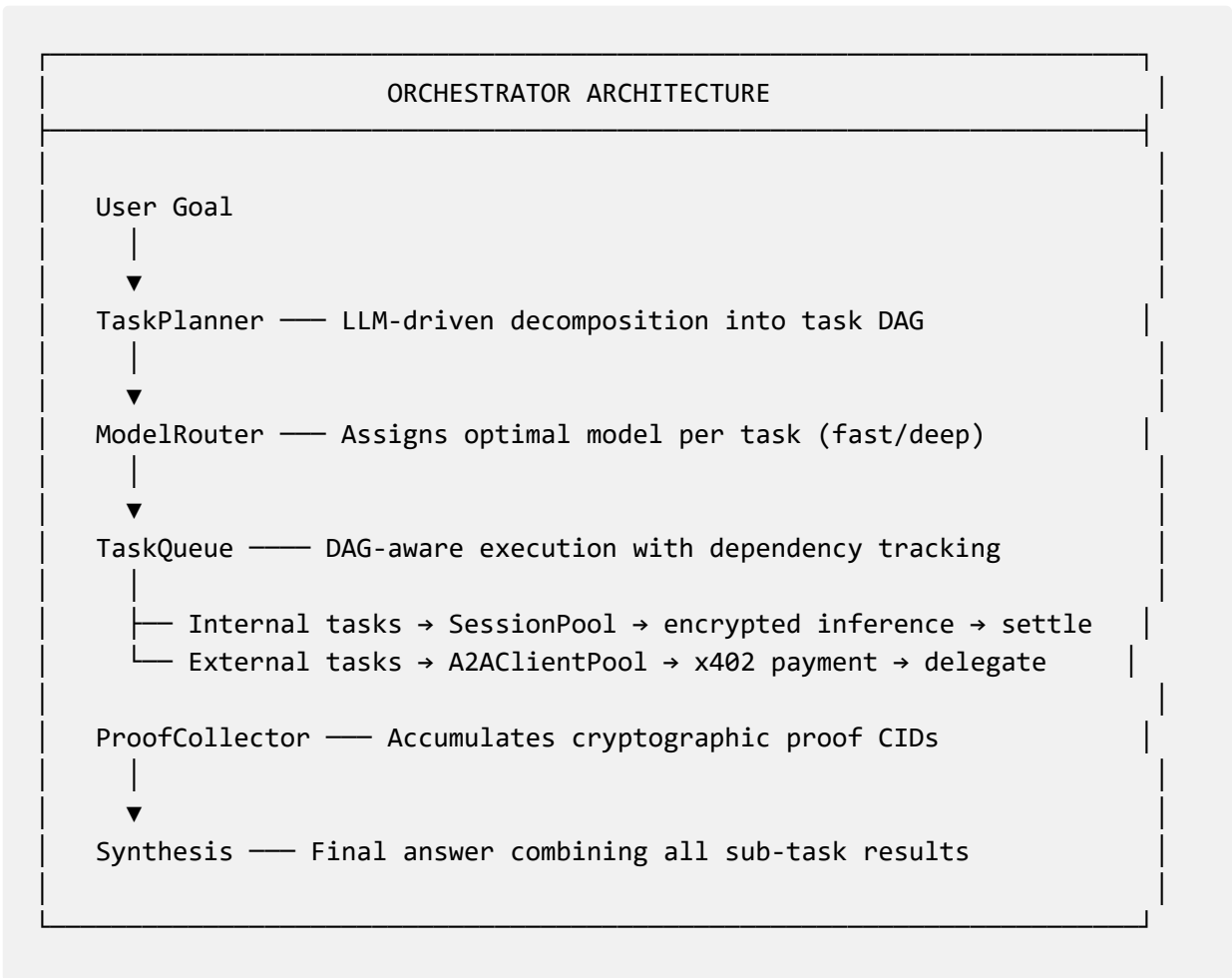
**Platformless AI is, to our knowledge, the first and only DePIN project to implement A2A** — bridging decentralised compute infrastructure with the emerging standard for inter-agent communication. As an early adopter of a pre-1.0 specification, our implementation may require updates as the standard evolves, but the architectural foundations (Agent Cards, JSON-RPC 2.0, SSE streaming) are stable and unlikely to change significantly.

A2A defines:

- **Agent Cards** ([/.well-known/agent.json](#)): JSON metadata describing an agent's capabilities, skills, supported protocols, and authentication — enabling automatic discovery
- **JSON-RPC 2.0**: Standardised request/response format for task delegation between agents
- **Server-Sent Events (SSE)**: Real-time streaming for long-running tasks with progress updates
- **Skill-based routing**: Agents advertise tagged skills; clients discover agents by matching skill tags to requirements

Until now, DePIN AI projects have operated as isolated inference endpoints. A2A transforms this into a collaborative agent network where agents discover each other's capabilities automatically, complex tasks get delegated to the most suitable agent, and results flow back through standardised channels.

## 6.3 Orchestrator Architecture



**Task Decomposition:** The **TaskPlanner** uses a persistent planning session to break complex goals into DAGs of typed sub-tasks (**analysis, synthesis, research, tool-calling, creative**), each with optional model preferences.

**Intelligent Model Routing:** The **ModelRouter** assigns models based on task complexity — deep models for tool-calling, synthesis, and large analysis tasks; fast models for research and small analysis — validated against the on-chain **ModelRegistry**.

**Orchestration Patterns:** Three built-in patterns cover common workflows:

Pattern	Description	Use Case
<b>FanOut</b>	Execute N sub-tasks in parallel, collect all results	Independent research queries, parallel analysis
<b>Pipeline</b>	Sequential execution, each task receives prior output	Staged refinement, multi-step reasoning
<b>MapReduce</b>	Parallel map phase, single reduce synthesis step	Document analysis, data aggregation

**Session Pooling and Multiplexing:** The **SessionPool** manages concurrent **FabstirSDKCore** instances with semaphore-based concurrency control, abort signal propagation, and transaction nonce serialisation for safe concurrent on-chain settlement. **Session multiplexing** further reduces costs: when a sub-task finishes, its blockchain session is cached by model rather than destroyed. The next sub-task using the same model reuses the cached session — no new SDK creation, no new

`startSession()`, no additional USDC deposit. For a typical orchestration with 1 planning + 5 same-model sub-tasks, this reduces deposits from 6 to 2 (67% savings). Sessions are only destroyed during `destroy()` at the end of orchestration.

## 6.4 SSE Streaming and Task Lifecycle

The orchestrator supports real-time progress streaming via Server-Sent Events, giving clients granular visibility into every stage of orchestration:

- **Content negotiation:** `POST /v1/orchestrate` with `Accept: text/event-stream` returns an SSE stream; without it, a sync JSON response is returned (fully backward compatible)
- **Phased progress events:** `decomposing` (goal breakdown), `executing` (per sub-task start/completion with task ID), `synthesising` (final answer generation)
- **Artifact streaming:** Synthesis results delivered as `artifact-update` events before the final `completed` status
- **Client disconnect handling:** If the client closes the SSE connection, orchestration is aborted via `AbortSignal` — no wasted compute
- **Task cancellation:** `DELETE /v1/orchestrate/:taskId` cancels an active streaming task

## 6.5 x402 HTTP Payment for A2A Delegation

The x402 HTTP payment protocol enables USDC micropayments for inter-agent delegation, running **alongside** the existing escrow model:

- **Internal inference** (own hosts): Unchanged escrow via `JobMarketplace`
- **External A2A delegation** (foreign agents): x402 per-request USDC micropayment via EIP-3009 `transferWithAuthorization`

**Inbound payments** (server): External agents can pay USDC to use the orchestrator as an alternative to JWT authentication. The server returns HTTP 402 with structured payment requirements; clients respond with a signed `X-PAYMENT` header.

**Outbound payments** (client): When delegating to external agents that return HTTP 402, the orchestrator automatically parses requirements, checks budget, signs an EIP-712 `TransferWithAuthorization`, and retries with the payment header.

Key properties:

- **Gasless USDC transfers:** EIP-3009 — the server submits the signed authorisation on-chain, no pre-funding required from the client
- **Budget enforcement:** `X402BudgetTracker` tracks cumulative outbound x402 spend with configurable limits
- **Agent Card advertising:** x402 pricing advertised in the Agent Card's `x402.accepts` field for automatic discovery
- **Account Kit V2 integration:** Sub-account signer provides scoped USDC spending authority for outbound payments

## 6.6 A2A Server and Client

**Server:** The **OrchestratorA2AServer** exposes the orchestrator as an A2A-compatible agent. It publishes an Agent Card at `/.well-known/agent.json` advertising skills, authentication options (JWT or x402), and pricing. Clients send orchestration requests via JSON-RPC; the server streams progress via SSE.

**Client:** The **A2AClientPool** and **AgentDiscovery** modules allow the orchestrator to discover external A2A agents by skill tags, delegate tasks to them, handle x402 payment negotiation automatically, and cache Agent Cards to avoid repeated network calls.

This means the orchestrator can both **serve** A2A requests from other agents and **delegate** to external A2A-compatible agents — enabling true multi-agent collaboration across organisational boundaries.

---

## 7. Economic Model

---

### 7.1 Token Economics

The protocol uses two tokens:

#### FAB Token (Native)

- Utility token for the Platformless AI ecosystem
- Used for host staking (minimum 1000 FAB)
- Governance voting rights (used for model approval voting)
- Slashing collateral for dispute enforcement (owner-controlled at MVP, DAO-governed post-MVP)

#### USDC (Stablecoin)

- Primary payment currency for sessions
- Stable value for predictable pricing
- No volatility risk for users or hosts

#### ETH (Native Currency)

- Alternative payment option
- Direct blockchain settlement
- Preferred by crypto-native users

### 7.2 Pricing Model

Hosts set prices per **million tokens** with a precision multiplier of 1000x.

```
Actual USD per million = pricePerToken / 1000
```

Examples:

- `pricePerToken = 60` → \$0.06/million (budget models)
- `pricePerToken = 5000` → \$5.00/million (standard models)
- `pricePerToken = 25000` → \$25.00/million (premium models)

This allows prices from \$0.001/million to \$100,000/million, accommodating everything from tiny test models to premium specialised inference.

### Price Discovery:

- Hosts compete on price for each model
- Users query `getModelPricing()` to compare hosts
- Market forces drive prices towards equilibrium
- No platform markup—users pay hosts directly

## 7.3 Payment Flow


 Token Economics & Payment Flow

Figure 5: Token Economics & Payment Flow showing the escrow-based settlement process and FAB token utility.

### Fee Structure:

- **90%** → Host (via HostEarnings contract)
- **10%** → Protocol treasury

### Accumulation Pattern:

To save gas, host payments and treasury fees accumulate in contracts rather than being sent immediately. Hosts withdraw when balances are sufficient to justify gas costs. This reduces gas by ~80% compared to individual transfers.

## 7.4 Staking Requirements

Requirement	Value	Purpose
Minimum host stake	1000 FAB	Sybil resistance, skin in the game
Maximum slash per action	50% of stake	Prevents accidental/malicious total loss
Minimum stake after slash	100 FAB	Below this triggers auto-unregister
Slash cooldown	24 hours	Prevents rapid-fire slashing
Stake lock period	None	Can unregister and withdraw anytime

### Why Stake?

- **Sybil resistance:** Prevents spam registrations
- **Quality signal:** Hosts with higher stakes demonstrate commitment
- **Weighted selection:** SDK's host selection algorithm weights stake at 35% by default
- **Dispute collateral:** Stakes can be slashed for proven misbehaviour (evidence-based)

### Slashing at MVP:

The protocol includes owner-controlled stake slashing at launch. When misbehaviour is proven via the CID evidence trail:

1. Owner reviews evidence (proofCID, deltaCID, conversationCID)
2. Owner calls `slashStake(host, amount, evidenceCID, reason)`
3. Slashed tokens transfer to protocol treasury
4. All slashes emit public events for transparency

See [SLASHING\\_SPECIFICATION.md](#) for full implementation details.

## 7.5 Host Selection and Reputation

The SDK implements a **weighted host selection algorithm** that users can configure:

Factor	Default Weight	Source
<b>Stake</b>	35%	On-chain <code>stakedAmount</code> from NodeRegistry
<b>Price</b>	30%	Normalised inverse of host pricing
<b>Uptime</b>	20%	Placeholder (95%) until metrics system
<b>Latency</b>	15%	Placeholder (100ms) until metrics system

### Selection Modes:

- **AUTO**: Standard weighted scoring (default)
- **CHEAPEST**: 70% price weight
- **RELIABLE**: 50% stake + 40% uptime weight
- **FASTEST**: 60% latency weight
- **SPECIFIC**: Use a preferred host address

*Note: Uptime and latency currently use placeholder values. A comprehensive metrics collection system is planned for future development.*

 Host Selection Algorithm

*Figure 6: Host Selection Algorithm showing the decentralised discovery and selection flow from NodeRegistry query through weighted scoring (stake, price, uptime, latency) to P2P connection.*

## 7.6 Minimum Deposits

Payment Method	Minimum	Rationale
ETH	0.0001 ETH (~\$0.40)	Cover gas costs
USDC	\$0.50	Prevent dust sessions

These minimums ensure sessions are economically viable for hosts.

## 7.7 Delegated Payments (Sponsored Sessions)

For agentic and SaaS use cases, a third party often needs to pay for inference on behalf of its users or autonomous agents — a SaaS backend covering its customers' usage, or a treasury sponsoring a fleet of coding agents — without surrendering spending control. The protocol supports this through **delegated payments** (delegate-pays):

- A **payer** account pre-approves a bounded USDC allowance to the `JobMarketplace` contract and authorises one or more **delegate** addresses on-chain (`authorizeDelegate`).
- A **delegate** — typically an ephemeral hot key generated for a bridge or daemon — signs sessions and spends *from the payer's allowance* but **never holds the funds**. The on-chain allowance is a hard ceiling: a runaway agent cannot spend beyond it.
- The SDK enters this mode via `authenticateAsDelegate({ signer, payer })`; sessions are created through `createSessionForModelAsDelegate`, with pre-flight allowance and balance checks before any chain write. Delegated sessions are **USDC-only**.
- Because an ERC-20 allowance is shared across a payer's delegates, the payer should be an account distinct from any interactive wallet. The orchestrator daemon wires this up from a single `FABSTIR_PAYER` environment variable, exposing an OpenAI-compatible endpoint that agents call while the payer foots the metered, capped bill.

This makes Platformless AI usable as a sponsored backend: end users and autonomous agents consume decentralised inference without managing wallets or gas, while the sponsor retains a cryptographically enforced spending cap. It composes with the x402 inter-agent payment layer (Section 6.5) — delegate-pays funds a party's own hosted sessions, while x402 settles cross-agent delegation.

---

## 8. Security and Privacy

---

### 8.1 Threat Model

The protocol defends against:

Threat	Mitigation
Eavesdropping	End-to-end encryption
Man-in-the-middle	ECDSA authentication
Replay attacks	Nonces, timestamps, message indices
Malicious hosts	Staking + evidence-based slashing + cryptographic proofs
Data theft	User-controlled encryption keys
Host reads inputs/weights in-use	Confidential computing / TEE — host-blind inference (proof-of-concept, §8.6)
Platform capture	Decentralised storage + blockchain
Censorship	Permissionless participation

### 8.2 Encryption Architecture

All sessions are encrypted by default using state-of-the-art cryptography.

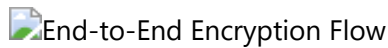
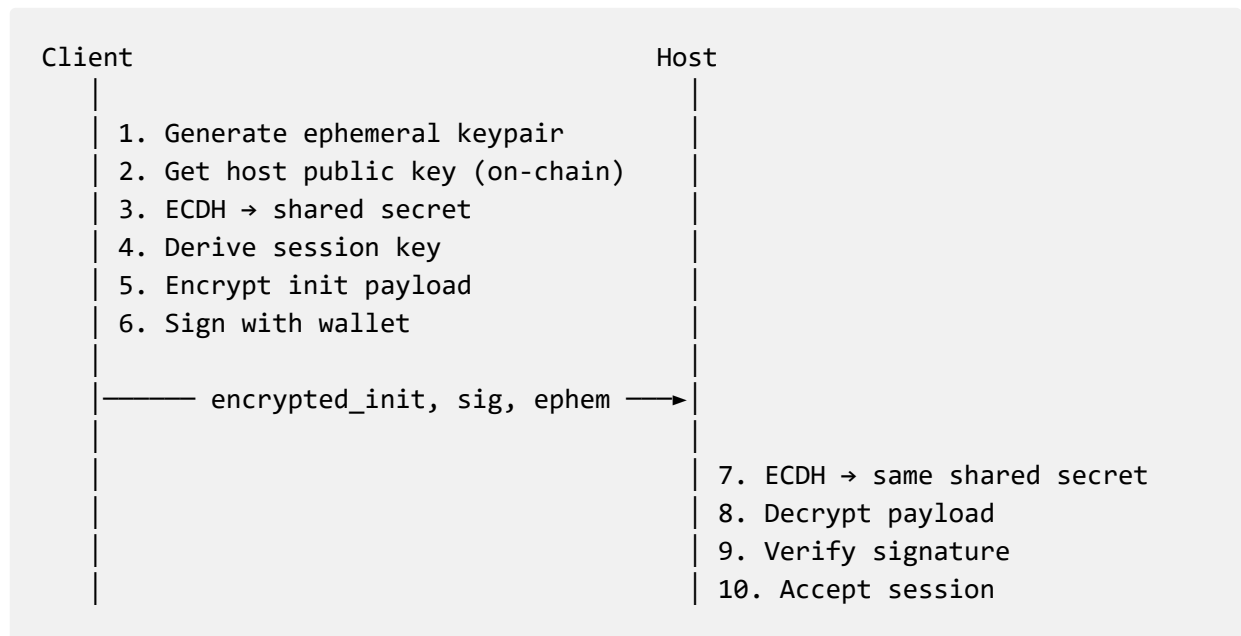


Figure 7: End-to-End Encryption Flow showing ECDH key exchange, session key derivation, and encrypted communication.

### Session Initialisation (Ephemeral-Static ECDH):



### Message Encryption (Symmetric):

After session init, messages use fast symmetric encryption:

```
encrypt(session_key, message, index) → ciphertext
```

- **XChaCha20-Poly1305**: AEAD cipher with 256-bit keys
- **Message index in AAD**: Prevents replay
- **~1ms overhead**: Negligible performance impact

### Forward Secrecy:

Ephemeral keys are generated fresh per session and discarded after use. Even if host's long-term key is compromised, past sessions remain secure.

## 8.3 What's Protected

### ✅ Encrypted:

- Prompts and responses
- Conversation history (on S5)
- Session configuration
- RAG queries and context

### ❌ Visible on-chain:

- Session creation/completion
- Payment amounts
- Token usage
- Host selection

On-chain data is necessarily public for verification, but contains no content.

### 8.3.1 In-Use Confidentiality (Current Boundary)

End-to-end encryption protects content **in transit** (client ↔ host) and **at rest** (on S5). It does not, by itself, make computation blind: the host the user selects necessarily decrypts the prompt in memory to run inference. "End-to-end" here means client to host with no platform intermediary in between — not that the host computes on ciphertext.

What this means in practice:

- Plaintext exists only **transiently, in the chosen host's memory**, for the duration of inference, and is never persisted unencrypted. No platform, proxy, or storage node ever sees it.
- Hosts are stake-bonded, independent operators selected by the user; a stateless, non-logging host that retains nothing is a far weaker exposure than a centralised provider that logs every query indefinitely.
- For workloads that require the host to be blind to inputs as well, two paths apply: **confidential computing via Trusted Execution Environments (TEE)** — the route to in-use privacy, now a software-complete, GPU-proven proof-of-concept whose final step is real hardware attestation (Section 8.6); and, today, **client-side pre-processing** — sensitive inputs can be embedded or transformed locally so only derived data (e.g. vectors) reaches the host.

We state this boundary explicitly because honest scoping of cryptographic guarantees is itself a security property.

## 8.4 Key Management

### Client Keys:

Derived from wallet private key using deterministic signature-based derivation. No additional secrets to manage.

### Host Keys:

Registered on-chain during node setup. Public key stored in NodeRegistry metadata.

### Session Keys:

Generated fresh per session, exist only in memory, discarded on completion.

## 8.5 S5 Storage Encryption

Data stored on S5 is encrypted by the SDK before upload:

```
// SDK encrypts before storage
const encrypted = encryptionManager.encrypt(conversation);
await storageManager.save(encrypted);

// S5 stores opaque bytes—cannot read content
```

S5 nodes see only encrypted blobs. Only users with the correct keys can decrypt.

## 8.6 Confidential Computing and Trusted Execution Environments (Proof-of-Concept)

Section 8.3.1 stated the current boundary plainly: the host a user selects must decrypt the prompt in memory to run inference, so end-to-end encryption protects content **in transit** and **at rest** but not **in use**. **Confidential computing** is the protocol's route to closing that last gap — *host-blind inference*, where even the host's own root operator cannot read the prompt or the model weights.

The implementation is a **working proof-of-concept**: the software is complete and has been **proven end-to-end on real NVIDIA GPU hardware** (encrypted model → attestation → key release → in-memory decrypt → live `llama.cpp` inference on CUDA → secure delete), covered by ~80 tests and hardened across two adversarial review rounds (node version `8.30.0-tee-confidential-inference`). The one remaining milestone is substituting the current **mock attestation backend** for **real hardware attestation** (see *Honest status* below). We describe it here because the architecture is built and the trust model it enables is central to the protocol's privacy story — while being explicit that it is not yet a production guarantee.

### What it adds beyond end-to-end encryption

With confidential computing, inference runs inside a hardware-isolated **Trusted Execution Environment (TEE)** — a Confidential VM combining a **CPU TEE** (Intel TDX or AMD SEV-SNP, which encrypts and integrity-protects the VM's memory) with an **NVIDIA GPU in Confidential Computing mode** (which encrypts the PCIe link and GPU memory). Inside that boundary the model weights and the prompt are processed in encrypted memory that even a privileged operator on the physical machine cannot read from system RAM or GPU VRAM. Trust shifts from "*trust the host operator not to look*" to "*trust the silicon vendor's hardware root of trust, verified by a remote attestation that the client can check*".

### The attested-model pipeline

Models are distributed **encrypted** (a sealed AES-GCM container whose data-encryption key the host never holds). Serving such a model runs through three sequential, **fail-closed** gates (`tee::prepare_attested_model`):

1. SIGNED POLICY is the  policy is  decrypted.	fetch the model's signed policy; verify the signer is the  model's on-chain provider (ModelRegistry) and the  within its validity window. Invalid ⇒ nothing is decrypted.
	 ▼
2. ATTESTED KEY RELEASE evidence  gpu_report_hash    nonce )  expected  model key,  time nonce +	generate an ephemeral keypair; gather hardware evidence  (GPU report + CPU TEE quote); cross-bind it: report_data = SHA-256( ephemeral_pubkey    gpu_report_hash    nonce ) A Key Broker Service verifies the evidence against expected  hardware measurements and only then releases the model key,  wrapped to the attested ephemeral key. The one- time nonce +  cross-binding defeat replay and attestation-relay.
	 ▼
3. DECRYPT-IN-MEMORY 0600) – + HASH-BIND + policy  the plaintext  hash; securely  failure.	decrypt only into RAM-backed tmpfs (/dev/shm, mode 0600) – never to disk; the container header binds model-id + policy hash (defeats pointer substitution); stream-hash the plaintext and check it against the on-chain SHA-256 model hash; securely erase (zeroize + unlink) on eviction or on ANY failure.

Every step fails closed: the model key is released **only** after a valid attestation; a node that is not TEE-enabled **refuses encrypted models outright**; and a host advertises the **tee-attested** capability — in its on-chain **NodeRegistry** registration and in the encrypted-WebSocket handshake — **only** when TEE enforcement is actually active. The same single flag gates both advertisement and enforcement, so a host cannot claim attested confidentiality it does not honour. Clients can therefore filter host discovery for **tee-attested** nodes and route sensitive workloads only to hosts that prove, cryptographically, that the operator is blind to the computation. This composes with the existing layers — E2E encryption still protects transit and storage; model governance (Section 4.2) still pins the model's hash and provider on-chain; the TEE adds in-use confidentiality on top.

### Honest status

- **Software complete.** The full pipeline — sealed-container format, signed-policy verification, ephemeral-key remote attestation with cross-binding, Key-Broker key release, tmpfs-only decryption, on-chain hash-binding, secure delete, and capability advertisement — is implemented, fail-closed throughout, and covered by ~80 passing tests.
- **Proven on real GPU hardware, end-to-end.** The complete encrypted-model path runs green on an NVIDIA GPU host: encrypt → attest → key release → decrypt to `/dev/shm` →

`llama.cpp` inference on CUDA → secure delete.

- **The attestation backend is currently a mock.** It produces cryptographically correct but **synthetic** hardware evidence. It exercises the entire protocol faithfully but does **not yet verify a real silicon root of trust** — so it does not, by itself, prove that a root operator cannot read VRAM.
- **Remaining milestone.** Run the same pipeline against **real NVIDIA Confidential-Computing attestation on a Confidential VM** (a CC-On H100/H200 GPU paired with an Intel TDX or AMD SEV-SNP CPU TEE). That hardware test is what turns "*the operator is asked not to look*" into "*the operator provably cannot look.*" **Until it is complete, confidential inference should be treated as an in-development capability, not a production guarantee.**

This is the strongest privacy posture the protocol targets, and the natural fit for regulated and IP-sensitive workloads — healthcare, legal, and enterprise source code — where even a trusted, stake-bonded host must be unable to read the inputs or the proprietary model weights it serves.

---

## 9. Dispute Resolution

---

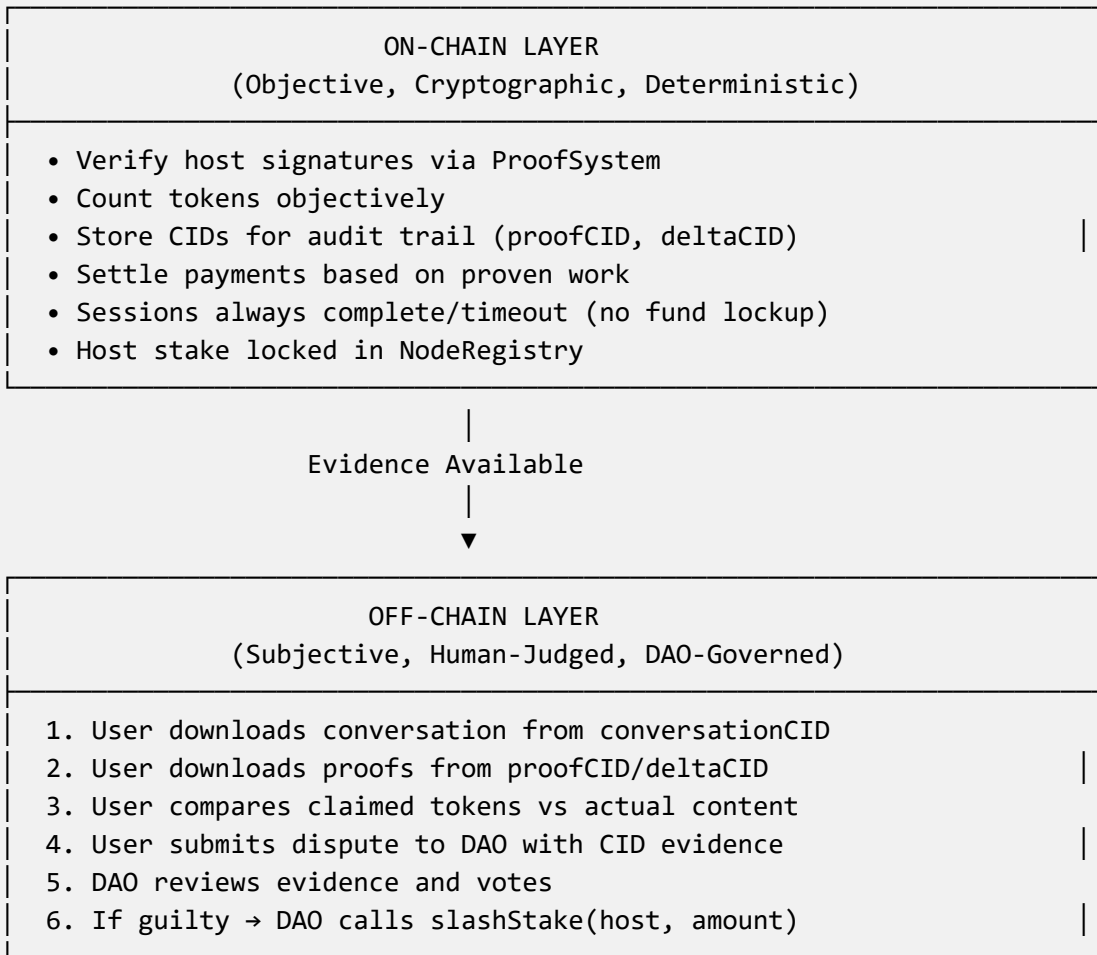
### 9.1 Design Philosophy

The protocol takes a **pragmatic approach** to disputes:

- **On-chain:** Handles objective, cryptographically verifiable facts
- **Off-chain:** Handles subjective quality judgements (future DAO)

Smart contracts **cannot** determine if an AI response was "good quality" or "factually correct"—these are inherently subjective. Attempting on-chain dispute resolution creates complexity, attack vectors, and gas waste.

### 9.2 Two-Layer Architecture



### 9.3 Evidence Trail

Every proof submission creates an immutable, verifiable record:

Component	Storage	Purpose
<code>proofHash</code>	On-chain	Cryptographic fingerprint
<code>proofCID</code>	S5	Complete proof data
<code>deltaCID</code>	S5	Incremental changes
<code>conversationCID</code>	S5	Full conversation at completion
<code>signature</code>	On-chain	Host's ECDSA attestation

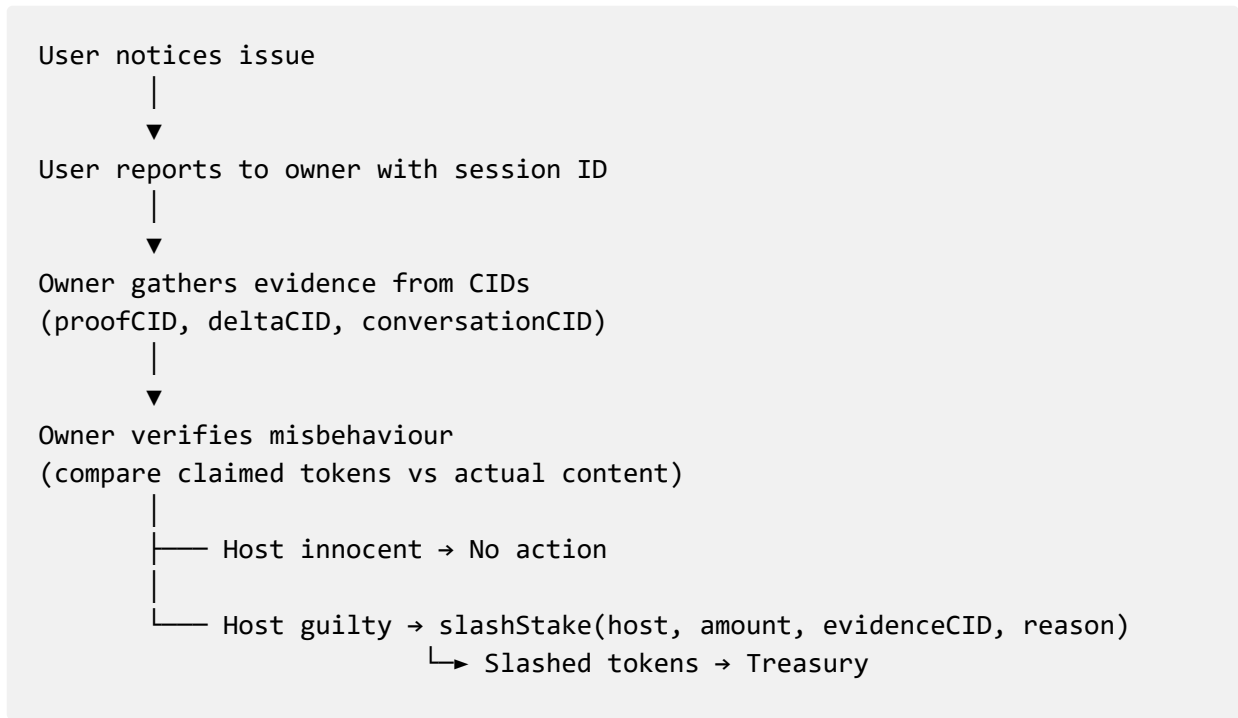
**Anyone can verify:**

1. Download CID content from S5
2. Hash content, compare to on-chain `proofHash`
3. Verify signature matches host
4. Count actual tokens vs claimed tokens

### 9.4 Dispute Resolution Flow

**MVP: Owner-Controlled Slashing**

At launch, dispute resolution is handled by the protocol owner using evidence-based slashing:



**Safety Constraints:**

- Maximum 50% slash per action
- 24-hour cooldown between slashes on same host
- Evidence CID required (public accountability)
- All slashes emit events for transparency

**Future: DAO-Governed Slashing**

Post-MVP, slashing authority transfers to a DAO contract:

```
// Owner transfers authority to DAO
setSlashingAuthority(daoContractAddress);
```

The DAO then manages dispute submissions, voting, and slash execution. No contract upgrade required—same `slashStake()` function, different caller.

 Dispute Resolution Flow

Figure 8: Dispute Resolution Flow showing the evidence-based slashing process from user report through arbiter verification to outcome, with safety constraints and governance transition path.

**9.5 Attack Mitigation**

Attack	Mitigation
Host misbehaviour	Evidence-based slashing with CID proof
False evidence	CIDs are content-addressed (immutable, verifiable)
Fund lockup grieving	Sessions always settle; no on-chain dispute state

Attack	Mitigation
Slash abuse	50% max per action, 24h cooldown, evidence required
Arbiter centralisation (MVP)	All slashes public, DAO transition planned

## 10. Governance

### 10.1 Current State

At MVP launch, governance operates in a **bootstrap phase** with admin-controlled operations:

- Contract upgrades controlled by admin keys (single-sig initially)
- Model approval via owner functions (`addTrustedModel`)
- Stake slashing via owner functions (`slashStake`)
- Parameter changes via owner functions

This centralisation is temporary and necessary for rapid iteration during early deployment. All admin actions are on-chain and publicly verifiable.

### 10.2 Progressive Decentralisation

The roadmap includes progressive decentralisation:

Phase	Governance
MVP	Admin-controlled (bootstrap)
Post-MVP	Multi-sig with community representatives
Mature	Full FAB token-weighted DAO voting

 Progressive Decentralisation Timeline

Figure 9: Progressive Decentralisation Timeline showing the governance transition from admin-controlled MVP through multi-sig to full DAO governance, with key responsibilities at each phase.

### 10.3 DAO Responsibilities (Future)

- Model approval/removal via voting
- Parameter changes (fees, minimums)
- Contract upgrades
- Dispute resolution and stake slashing
- Treasury allocation

### 10.4 Model Governance

The ModelRegistry contract is **architecturally ready** for community governance with two approval tiers:

**Tier 1: Owner-Curated (MVP)**

```
// Owner adds trusted models directly
function addTrustedModel(
    string huggingfaceRepo,
    string fileName,
    bytes32 sha256Hash
) external onlyOwner;
```

**Tier 2: Community-Voted (Available, No UI Yet)**

```
// Anyone can propose a model (requires FAB stake)
function proposeModel(string repo, string file, bytes32 hash) external;

// FAB token holders vote
function voteOnProposal(bytes32 modelId, uint256 amount, bool support)
external;

// Execute passed proposals
function executeProposal(bytes32 modelId) external;
```

The community voting infrastructure exists in the deployed contracts but awaits UI implementation. For MVP, the owner uses `addTrustedModel()` to bootstrap the model ecosystem.

**Model Governance Prevents:**

- Malicious models (trojaned weights)
- Illegal content generation models
- Models violating licensing terms

## 11. Roadmap

 Roadmap Timeline

*Figure 10: Roadmap Timeline showing development progress across 9 phases, with 6 phases completed (67%) and current focus on Phase 7: Economic Refinement.*

**Phase 1: Foundation (Completed)**

- Smart contract deployment (UUPS upgradeable)
- SDK core functionality
- Host node implementation
- Basic session lifecycle
- Multi-chain support (Base Sepolia)

**Phase 2: Security (Completed)**

- End-to-end encryption (default)
- Ephemeral-static ECDH key exchange
- XChaCha20-Poly1305 symmetric encryption
- Forward secrecy
- ECDSA authentication

### Phase 3: Storage (Completed)

- S5 integration for conversations
- Proof storage (proofCID, deltaCID)
- User-owned data sovereignty
- Encrypted storage

### Phase 4: Advanced Features (Completed)

- RAG/Vector search support
- Host-side embeddings
- Web search integration
- Vision processing (Florence-2, OCR)
- Image generation (FLUX.2 diffusion sidecar with SDK auto-intent detection)
- Zero-knowledge proof generation (host-side STARK proofs)
- GPU-accelerated video transcoding (H.264, HEVC, AV1 via NVENC sidecar)
- End-to-end encrypted video pipeline (source/output encrypted on Sia, E2EE WebSocket transit)
- Capacity-aware transcode load balancing across multiple hosts
- Per-transcode WebSocket isolation for concurrent job support
- Pending job tracking with automatic overflow and retry-with-backoff
- GOP-level verification layer (STARK proofs, Merkle tree, on-chain root commitment)
- Stress-tested: 7 concurrent jobs across 2 GPU hosts (NVIDIA L40S + RTX Pro 6000), 100% completion
- HLS adaptive bitrate streaming — fMP4 segments with per-segment encryption (SDK v1.18.0)
- Client-side M3U8 playlist generation (master + variant, HLS v7 with fMP4 init segment)
- Per-segment encryption: preview segments unencrypted (z-prefix), paid segments XChaCha20-Poly1305 encrypted (u-prefix)
- End-to-end validated: encrypted upload → GPU transcode (NVIDIA L40S) → HLS segments on Sia/S5 → USDC payment → hls.js playback with decrypting service worker

*Note: Production UI for ZK proof verification display is pending; backend infrastructure is complete.*

### Phase 5: Agentic AI Infrastructure (Completed)

- Claude Bridge — Anthropic Messages API compatibility layer
- OpenAI Bridge — OpenAI Chat Completions, Images, and Responses API compatibility layer
- Full tool use support (23+ tools, streaming SSE, multi-turn tool results)
- Image generation via `/v1/images/generations` (FLUX.2 diffusion, quality/size mapping)
- Vision support (base64 image\_url passthrough)


- Think-block stripping and output safeguards
- Session auto-recovery on disconnect
- Production validation: Claude Code and OpenCode running on decentralised hosts
- SaaS-ready: drop-in Anthropic or OpenAI API replacement for product teams
- 129 + 162 tests passing

## Phase 6: Multi-Agent Orchestration (Experimental — Completed)

- `@fabstir/orchestrator` package — DAG-based task decomposition and execution
- A2A protocol (v1.0.0-rc) — Agent Cards, skill-based discovery, JSON-RPC delegation
- Intelligent model routing — task-type-aware assignment with on-chain validation
- Orchestration patterns — FanOut, Pipeline, MapReduce
- SSE streaming — real-time phased progress, content negotiation, task cancellation
- x402 HTTP payment — USDC micropayments for inter-agent delegation (EIP-3009)
- Session pooling — semaphore-based concurrency with nonce serialisation
- Session multiplexing — per-model session caching to reduce USDC deposit burden (up to 67% savings)
- BigInt consistency — uint256 ABI params consistently encoded as BigInt for Account Kit compatibility
- 351 tests passing

## Confidential Computing — Trusted Execution Environments (Proof-of-Concept, In Progress)

*A parallel security workstream toward host-blind inference (Section 8.6) — software complete and GPU-proven; one hardware-attestation milestone remains.*

- Encrypted-model container format (sealed AES-GCM; the host never holds the key)
- Signed model policy — provider-signed, on-chain-provider verification, validity window
- Remote-attestation key release — ephemeral keypair + GPU report + CPU-TEE quote, cross-bound (`SHA-256(pubkey || gpu_report_hash || nonce)`), one-time nonce; Key Broker releases the model key only on a valid attestation
- In-memory-only decryption to RAM-backed tmpfs (`/dev/shm`, never to disk) + secure delete (zeroize + unlink)
- On-chain SHA-256 model hash-binding; container-header model-id/policy binding (anti-substitution)
- Fail-closed throughout — non-TEE nodes refuse encrypted models; the `tee-attested` capability gates advertisement and enforcement with one flag
- Proven end-to-end on real NVIDIA GPU hardware (encrypt → attest → key release → decrypt → `llama.cpp` on CUDA → inference → secure delete); ~80 tests; 2 adversarial review rounds
-  **Remaining milestone:** swap the mock attestation backend for **real NVIDIA Confidential-Computing attestation** on a Confidential VM (CC-On H100/H200 GPU + Intel TDX or AMD SEV-SNP CPU TEE) — the step that proves a root operator provably cannot read the model or prompt from VRAM

## Phase 7: Economic Refinement (Planned)

- 🕒 Real-time uptime and latency metrics collection
- 🕒 On-chain reputation scoring
- 🕒 Host quality dashboards
- 🕒 Dynamic pricing mechanisms

## Phase 8: Decentralisation (Planned)

- 🕒 DAO governance launch
- 🕒 Multi-sig transition
- 🕒 Community-driven upgrades
- 🕒 Dispute resolution DAO

## Phase 9: Scaling (Future)

- 🕒 Additional L2 chains
- 🕒 Cross-chain sessions
- 🕒 Optimistic proof batching
- 🕒 State channels for high-frequency
- 🕒 Multi-orchestrator coordination via A2A
- 🕒 Agent marketplace with on-chain reputation scoring
- 🕒 Confidential computing (TEE) — production hardening beyond the proof-of-concept: real hardware attestation in production, multi-vendor TEE support, and attested-host routing in the SDK

---

## 12. Conclusion

---

Platformless AI represents a fundamental shift in how AI services are delivered. By eliminating platform intermediaries and replacing trust with cryptography, the protocol creates an open, permissionless marketplace where:

- **Users** maintain privacy, data sovereignty, and choice
- **Hosts** operate independent businesses with fair compensation
- **SaaS companies** embed AI into products without vendor lock-in or privacy risk
- **Developers** build with autonomous AI agents on private, sovereign infrastructure
- **Innovation** flourishes without gatekeepers
- **Trust** is established through verifiable open-source code, cryptographic mathematics, and carefully designed economic incentives—not platform reputation

The three pillars of trustless operation:

1. **Code:** Open-source smart contracts that anyone can audit, with deterministic execution
2. **Mathematics:** Cryptographic proofs, signatures, and encryption that provide guarantees no human can override
3. **Economics:** Staking requirements, payment escrow, and fee structures that align incentives towards honest behaviour

With the Claude Bridge and OpenAI Bridge extending the protocol to support autonomous AI agents, SaaS AI backends, and image generation, the transcoding pipeline adding encrypted GPU-

accelerated video processing with load balancing, and the orchestrator package adding experimental multi-agent coordination via the A2A protocol (v1.0.0-rc) with x402 micropayments, Platformless AI is no longer limited to chat-based inference. The same encrypted, decentralised infrastructure that protects a private conversation also protects an enterprise development team's source code as Claude Code or OpenCode edits it, transcodes video across multiple GPU hosts with HLS adaptive bitrate streaming and per-segment encryption without any intermediary seeing the content, orchestrates multi-agent workflows that decompose complex goals across optimal models, generates images from natural language prompts via FLUX.2 diffusion without exposing creative intent to intermediaries, or processes a SaaS product's customer data as it flows through AI-powered features. Since NVENC encoding uses dedicated hardware separate from the CUDA cores, hosts can serve AI inference and video transcoding simultaneously — dual revenue streams from the same GPU without performance degradation. With both Anthropic and OpenAI API compatibility plus early A2A protocol adoption — the first DePIN project to implement it — virtually any AI-powered application or agent network can migrate to decentralised infrastructure. And for the most sensitive workloads, the confidential-computing proof-of-concept charts the path beyond encryption-in-transit to host-blind inference inside Trusted Execution Environments — software-complete and GPU-proven today, with real hardware attestation the final milestone — so that one day even the host serving a model cannot read the prompt it answers or the weights it runs. Companies are already approaching us to use this infrastructure.

The AI revolution should not be controlled by a few large platforms. Platformless AI provides the infrastructure for a truly open AI ecosystem where anyone — users, developers, enterprises, and SaaS companies — can participate, compete, and benefit.

**Join the revolution. Become platformless.**

---

## Appendix A: Contract Addresses (Base Sepolia)

---

Contract	Proxy Address
JobMarketplace	0xD067719Ee4c514B5735d1aC0FfB46FECf2A9adA4
NodeRegistry	0x8BC0Af4aAa2dfb99699B1A24bA85E507de10Fd22
ModelRegistry	0x1a9d91521c85bD252Ac848806Ff5096bBb9ACDb2
ProofSystem	0xE8DCa89e1588bbdbc4F7D5F78263632B35401B31
HostEarnings	0xE4F33e9e132E60fc3477509f99b9E1340b91Aee0
FAB Token	0xC78949004B4EB6dEf2D66e49Cd81231472612D62
USDC Token	0x036CbD53842c5426634e7929541eC2318f3dCF7e

---

## Appendix B: SDK Installation

---

```
npm install @fabstir/sdk-core
```

```
import { FabstirSDKCore, ChainId } from "@fabstir/sdk-core";

const sdk = new FabstirSDKCore({
  chainId: ChainId.BASE_SEPOLIA,
  rpcUrl: "https://sepolia.base.org",
  contractAddresses: {
    jobMarketplace: "0xD067719Ee4c514B5735d1aC0FFB46FECf2A9adA4",
    nodeRegistry: "0x8BC0Af4aAa2dfb99699B1A24bA85E507de10Fd22",
    // ... other addresses
  },
});

await sdk.authenticate("metamask");
const sessionManager = sdk.getSessionManager();
```

## Appendix C: Resources

---

- **Documentation:** <https://docs.fabstir.com>
  - **GitHub:** <https://github.com/Fabstir>
  - **SDK Reference:** [https://github.com/Fabstir/fabstir-llm-sdk/blob/main/docs/SDK\\_API.md](https://github.com/Fabstir/fabstir-llm-sdk/blob/main/docs/SDK_API.md)
  - **Contract Reference:** <https://github.com/Fabstir/fabstir-compute-contracts>
- 

**Platformless AI** — Trustless AI for Everyone

Copyright © 2025-2026 Fabstir. All rights reserved. SPDX-License-Identifier: BUSL-1.1