The Secure Intelligence Layer for the Agentic Web

A Decentralized, Privacy-Preserving AI Inference Protocol
Native to Solana

Christopher Ryan Gilbert

Founder & Principal Engineer, 100X CTO, LLC

100x.dev

Version:	Draft v0.3
Date:	April 2026
Status:	Pre-implementation concept
Contact:	100x.dev

This document is for discussion purposes only. Not investment advice.

Contents

Abstract

The transition from human-operated software to autonomous AI agent economies is not a future event. It is underway. The infrastructure that serves it, however, remains almost entirely legacy: centralized AI APIs designed for human-paced, subscription-billed, request-response workflows. These systems log everything, censor selectively, and operate on pricing models that assume a human is on the other end of every call.

They will not survive contact with the agentic web.

This whitepaper describes a protocol built to replace them — a decentralized, privacy-enforced AI inference network that is architecturally native to the machine economy. It combines hardware-enforced privacy via Trusted Execution Environments (TEEs), onion-routed prompt delivery, stateless open-source model inference, and a payment layer built on Solana's sub-penny finality and the x402 streaming payment standard.

Every agent that uses this network is provisioned a non-custodial Solana wallet generated inside a TEE enclave. The agent pays for its own compute. It pays for external API access. It transacts with other agents. No human operator required. No centralized API key. No monthly invoice.

The mission is singular: build the permissionless compute layer for the agentic web — where intelligence is a commodity that flows freely, privately, and at machine speed.

1. Introduction: The Impending Agentic Bottleneck

1.1 The Architecture of the Old World

The dominant AI infrastructure stack was architected for a specific user: a human sitting at a keyboard, sending one request every few seconds, operating under a monthly API quota, in a jurisdiction that tolerates the platform's terms of service.

OpenAI's API. Anthropic's Claude API. Google's Gemini endpoints. These are extraordinarily capable systems. They are also surveillance infrastructure. Every prompt is logged. Every completion is retained. Usage patterns are analyzed. Content is filtered through policy layers that are neither transparent nor deterministic — they can and do change without notice, and the change can silently break downstream behavior.

For a human user, this is an inconvenience. For an autonomous AI agent, it is a critical architectural failure.

1.2 What Autonomous Agents Actually Require

An autonomous agent is not a chatbot. It is a software process that perceives its environment, reasons about it, and takes action — repeatedly, at high frequency, with no human in the loop. The inference

calls an agent makes are not discrete user-initiated events. They are the agent's continuous cognitive process.

This changes the requirements profile entirely:

Determinism. An agent executing a long-horizon task cannot tolerate a policy layer that randomly decides its prompt violates content guidelines halfway through a workflow. The logic must be consistent. An agent building a DeFi strategy, analyzing threat intelligence, or drafting legal documents cannot have its reasoning silently redirected by an upstream filter.

Privacy. When an agent reasons, it thinks with context. That context may include user PII, proprietary business logic, API keys, strategic plans, or financial positions. Routing that context through a centralized provider that logs every request is not a privacy policy problem — it is a fundamental security breach waiting to happen. No policy statement by a centralized provider changes this. The data leaves your control the moment it hits their endpoint.

Micro-payment economics. An agent that executes 50,000 inference calls per day is not paying a monthly subscription. It needs to pay per call, in fractions of a cent, in real time, without human intervention. The SaaS billing model is simply the wrong primitive. It requires a human to authorize the payment relationship in advance, cap it at some quota, and reconcile it at month end. Agents need financial autonomy: the ability to acquire compute, pay for it, and continue — without ever blocking on a human authorization step.

Latency. An agent reasoning through a multi-step task is blocked on each inference call. Centralized providers introduce geographic routing overhead, rate-limit queuing, and cold-start latencies that compound across an agentic workflow. A decentralized network where inference nodes compete to serve requests eliminates the single-provider bottleneck.

1.3 The Scale of the Coming Demand

The venture and engineering communities are still largely thinking about AI agents as a novelty — demos, experiments, assistants that occasionally book a calendar event. This framing will age badly. The correct analogy is the early internet: a technology whose scope is so large that early participants routinely underestimate the order of magnitude by which it will grow.

By conservative estimate, agent-to-agent API traffic will exceed human-to-API traffic within three years. The inference workloads generated by autonomous software — agents managing portfolios, executing research pipelines, operating DevOps loops, running customer service workflows — will dwarf anything we have seen. The infrastructure that serves this must be built to a different standard than what exists today.

That infrastructure does not yet exist at scale. This protocol is my answer to that gap.

2. The Philosophy: Cryptographic Separation of Mind and State

2.1 The Insufficient Promise of Policy-Based Privacy

Every major centralized AI provider publishes a privacy policy. These documents make commitments about data retention, access controls, and usage limitations. Some of them are sincere. None of them are sufficient.

A privacy policy is a legal instrument. It is enforced after the fact, if at all, by regulatory bodies that operate on timescales measured in years. It can be revised unilaterally. It contains carve-outs for government requests, law enforcement, and "safety" exceptions that are never fully enumerated. And fundamentally, it requires you to trust the organization making the promise — to trust their employees, their infrastructure, their security posture, their future business decisions, and the jurisdictions they operate in.

That is not a security model. It is a trust relationship. Trust relationships fail.

Erik Voorhees articulated the principle correctly with Venice.ai: there must be a **Separation of Mind and State**. The cognitive process — the prompt, the context, the reasoning — must be structurally isolated from any party that could surveil or control it. Not isolated by policy. Isolated by mathematics and hardware.

2.2 Hardware-Enforced Privacy: The Zero-Trust Model

The protocol treats privacy as a hardware property, not an operational one. The mechanism is the Trusted Execution Environment.

A TEE is a hardware-isolated compute region within a processor. Intel calls their implementation SGX (Software Guard Extensions). AMD calls theirs SEV (Secure Encrypted Virtualization). Both create a cryptographically sealed enclave where code executes and data resides in a state that is inaccessible to everything outside the enclave — including the operating system, the hypervisor, and the physical hardware operator.

The enclave's integrity is verifiable via **remote attestation**: a cryptographic proof, signed by the hardware manufacturer's root of trust, that a specific codebase is running unmodified inside a genuine TEE. Any party can verify this proof without trusting the operator of the hardware. The trust is in the hardware specification and the manufacturer's signing key, not in the data center operator's promises.

This is the zero-trust model applied to AI inference. The node operator — whoever is running the GPU that processes an inference request — cannot read the prompt. Cannot read the completion. Cannot log the context window. The enclave unseals, processes, responds, and wipes. The operator sees electricity consumed and a token payment received.

2.3 Why Venice Got It Right, and Where It Stops Short

Venice.ai deserves genuine credit for operationalizing this philosophy before the market demanded it. Their architecture routes inference through TEE-equipped nodes on decentralized GPU networks like Phala, achieves end-to-end encryption of prompt content, and refuses to implement the content filtering layers that characterize centralized providers.

The gap is not in the privacy model. The gap is in the economic substrate.

Venice built for humans on Base — Ethereum L2. The VVV token governs access via a stake-for-access mechanism. For a human user making tens or hundreds of requests per day, this is workable. For an AI agent making thousands of requests per hour and needing to atomically pay for each one in real time, it is the wrong foundation.

Base inherits Ethereum's transaction overhead. Even on L2, settlement latency and per-transaction cost structures are incompatible with sub-cent, high-frequency machine payments. You cannot stream fractions of a penny across an Ethereum-family chain and have it make economic sense at the throughput an agent economy demands.

Solana is not a compromise. It is the correct answer for this use case, and I will make that case precisely in the sections that follow.

3. Core Protocol Architecture: A Deep Technical Dive

3.1 A Note on Phased Deployment

The protocol described in this section represents the full target architecture. It ships in two stages.

v1.0 (April 2026) delivers the Solana payment and routing layer: token-authenticated inference, credit-based billing in USDC, and agent wallet provisioning. Inference privacy at launch is provided by routing to established TEE-backed providers. The privacy guarantee is real; Use Pod does not yet operate the enclave nodes itself.

v2.0 (H2 2026) introduces Use Pod's proprietary TEE inference nodes with remote attestation, E2EE mode, and onion-routed delivery. At that point the full architecture described below is live and self-sovereign.

This sequencing is deliberate. Launching the payment primitive first — before the node network exists at scale — establishes market fit, attracts node operators, and funds the infrastructure buildout from real revenue rather than speculation. The vision does not change. The path to it is staged.

3.2 The Lifecycle of a Prompt

Understanding this protocol requires following a single inference request from origin to completion. Here is that lifecycle in full.

Step 1: Agent Initialization and Key Provisioning

When an agent is deployed on this network for the first time, a provisioning ceremony occurs inside a TEE enclave running the protocol's node software. The enclave generates an Ed25519 keypair. The private key never exits the enclave — not during provisioning, not during operation, not ever. The public key is registered on-chain as the agent's identity. The corresponding Solana wallet address is derived from this public key and funded with the protocol's native token.

The agent now has a cryptographic identity and financial autonomy, both of which are rooted in hardware-enforced secrecy.

Step 2: Prompt Encryption at the Agent Runtime

When the agent constructs an inference request, the payload — system prompt, context window, user message, any attached data — is encrypted client-side using the public key of the target inference node cluster. The encryption uses X25519 key exchange combined with ChaCha20-Poly1305 AEAD. The ciphertext is what leaves the agent's environment. The plaintext never traverses an unencrypted channel.

Step 3: Onion-Routed Delivery

The encrypted payload is not sent directly to an inference node. It is wrapped in layers of encryption and routed through an onion network of protocol relay nodes before reaching the inference layer. This design is borrowed from Tor's threat model: no single relay node can correlate the request origin IP with the inference destination. An inference node sees an incoming request arrive from a relay. The relay sees a packet pass through. Neither can reconstruct the full path.

The relay network is permissionless. Any node that stakes the required collateral can participate as a relay. Relay fees are paid from the agent's wallet per-hop, settled via the Solana MPP streaming payment primitive. Relays that fail to forward correctly have their stake slashed by the on-chain enforcement mechanism.

Step 4: Decryption and Processing Inside the Enclave

The inference node's TEE enclave holds the cluster's private key. When a request arrives, the enclave decrypts the payload — this happens inside the hardware boundary, invisible to the node operator. The decrypted prompt and context are loaded into GPU memory for inference.

The model running inside the enclave is one of a predefined set of open-source models: Llama 3.1 70B, Llama 3.1 405B, Mistral variants, Qwen 2.5, DeepSeek R1 — the specific model is specified in the request header. These models are loaded from a content-addressed model registry where the model weights are cryptographically hashed. The attestation report includes the model hash, proving to any verifier that the request was processed by an unmodified, known model and not a backdoored variant.

Step 5: Stateless Inference and Memory Wiping

Each inference request is processed statelessly. No context from a prior request persists in any form be-

tween calls. When the inference completes, before the completion payload is encrypted and dispatched, the enclave executes a deterministic memory wipe of the working memory regions used during inference. This is not a software-level memset that can be bypassed — it is enforced by the enclave's memory management, with the wipe included in the attestable code path that verifiers can inspect.

The inference node holds no state between requests. It is a stateless compute function that accepts encrypted input, produces encrypted output, and returns to an idle state.

Step 6: Encrypted Response and Payment Settlement

The completion is encrypted with the agent's public key, routed back through the relay network, and decrypted by the agent's runtime. Simultaneously, the payment for inference compute is settled: the agent's wallet signs a micro-payment transaction to the inference node's wallet, denominated in the protocol token, for the exact number of tokens processed. This settlement is atomic with the delivery of the completion — if payment fails, the completion is not released. If the completion is withheld, the payment is not signed.

Total end-to-end latency target for this lifecycle on commodity hardware: under 200ms for requests under 2K tokens, excluding inference time. The onion routing overhead is designed to add no more than 40ms across a three-hop path.

3.3 Node Operator Infrastructure

Inference nodes must meet hardware requirements: a modern Nvidia GPU (A100, H100, or equivalent consumer grade for smaller models), a processor with SGX or SEV support, a minimum stake of protocol tokens locked in the on-chain registry, and an uptime SLA enforced by the slashing mechanism.

Node operators do not need to trust the protocol team. The enclave software they run is open source, audited, and the attestation mechanism allows any external party to verify that any given node is running the correct, unmodified code. The protocol team cannot push a silent update that changes the enclave behavior — any change would produce a different attestation measurement, which clients would reject until they explicitly upgrade.

4. The Economic Engine: Web3 Rails for AI

4.1 Why Ethereum Family Chains Cannot Serve Machine Economies

The question of which blockchain to build on is not aesthetic. It is a systems engineering decision with hard numerical constraints.

An autonomous AI agent running a research pipeline might make 1,000 inference calls in an hour. If the average inference costs $0.001 — one-tenth of a cent — the agent spends $1.00/hour on compute. Each of those 1,000 calls requires an on-chain payment settlement. The payment infrastructure must process 1,000 transactions per hour per agent without those transactions costing more than the infer-

ence they're paying for.

On Ethereum mainnet, average transaction fees in active periods run $2–$15. This makes per-inference micro-payment settlement on mainnet economically absurd — the fee exceeds the payment by orders of magnitude. Ethereum L2s like Base reduce this, but even at Base's best performance, transaction fees run $0.01–$0.10, and confirmation latency is measured in seconds to minutes under load. An agent needing 1,000 settled transactions per hour at $0.001 each cannot use a platform where each settlement costs ten to one hundred times the value being transferred.

This is not a criticism of Ethereum's design goals. Ethereum is optimized for high-value, low-frequency settlement with strong finality guarantees. It is the right tool for DeFi protocols moving millions of dollars. It is architecturally incorrect for machine economies moving fractions of a penny at thousands of transactions per second.

4.2 The Solana Fit

Solana's architecture was designed for high-throughput, low-latency settlement. The relevant performance characteristics are summarized in Table 1.

Table 1: Solana Performance Characteristics

Metric	**Value**
Theoretical throughput	65,000+ TPS
Sustained throughput	3,000–5,000 TPS
Block time	∼400ms
Optimistic finality	∼400ms
Full finality	∼13 seconds
Transaction cost	$0.000025 (25 millionths of a dollar)

At Solana's transaction cost, settling 1,000 inference payments costs $0.025. The payment infrastructure cost is 2.5% of a $1.00 compute spend — a reasonable overhead. At Ethereum mainnet rates, the same 1,000 settlements could cost $2,000–$15,000. This is not a marginal difference. It is the difference between a functional economic model and an impossible one.

Solana's 400ms block time also matters for agent latency. An agent waiting for payment confirmation before receiving its inference result needs that confirmation to arrive fast enough to not materially impact the inference cycle. Sub-second confirmation is compatible with a 200ms inference round trip. Multi-second confirmation is not.

4.3 The Solana Micro-Payment Protocol (MPP)

Beyond per-transaction settlement, the protocol leverages Solana's streaming payment primitives for continuous payment flows. The Solana MPP enables payment streams — ongoing, continuously updated payment channels where value flows at a defined rate per unit time without requiring a new on-chain transaction for each interval.

For inference workloads that stream tokens — where an agent receives completion tokens progressively rather than waiting for a full completion — streaming payment is the correct primitive. The agent opens a payment stream to the inference node at the start of a streaming inference call. Value flows token-by-token. The stream closes when the completion terminates. The total settled amount is exactly proportional to the tokens delivered.

This eliminates the per-inference transaction overhead for streaming workloads and aligns payment precisely with value delivered. An inference node that delivers 500 tokens before a network interruption receives payment for exactly 500 tokens, not zero or the full request amount.

5. The Unified Agentic Wallet

5.1 The Problem of Agent Financial Identity

In the current AI deployment paradigm, agents do not have financial identities. A human operator funds an OpenAI account, obtains an API key, embeds that key in the agent's environment, and the agent makes requests that are billed to the human's account. The agent has no economic agency. It cannot earn, spend, or manage funds. It is a cost center on a human's balance sheet.

This model breaks completely in multi-agent systems where agents transact with each other, in agent marketplaces where agents sell services, and in any system where agents must acquire compute resources autonomously without human intervention in the payment loop.

The correct model gives each agent its own financial identity — a wallet it controls, funded with resources it has earned or been provisioned, capable of unilaterally authorizing expenditures within programmatic constraints.

5.2 TEE-Based Key Provisioning

The Unified Agentic Wallet is provisioned during the agent's initialization ceremony inside a TEE enclave. The process:

1. The enclave generates an Ed25519 keypair using the enclave's hardware random number generator — a true entropy source that is inaccessible to the operator and unguessable.
2. The private key is sealed using the TEE's platform sealing key — an encryption key derived from the hardware's identity that can only be accessed by the same enclave software running on the same (or attested equivalent) hardware.

3. The sealed private key blob is stored externally (the protocol's distributed key registry), but it can only be unsealed inside a valid instance of the protocol enclave software, as proven by remote attestation.
4. The public key is registered on-chain as the agent's identity.

The result: the private key never exists in plaintext outside the enclave. The node operator cannot access it. The protocol team cannot access it. The agent's human deployer cannot access it — unless the enclave software explicitly implements an export path, which the reference implementation does not. The agent's funds are controlled exclusively by the agent's programmatic logic executing inside the enclave.

5.3 Agent Financial Sovereignty

A wallet inside a TEE enclave changes the economic relationship between agents and infrastructure. An agent with its own wallet can:

- Pay for inference compute autonomously without a human authorizing each payment
- Receive payment from other agents for services rendered
- Fund downstream agent invocations as part of a multi-agent workflow
- Hold balances across sessions without a centralized custodian
- Operate within programmatic spending limits defined at deployment time, enforced by enclave logic

This is not merely convenient. It is architecturally necessary for agent economies to function. If every financial decision an agent makes requires human intervention to authorize payment, then the agent is not autonomous — it is a semi-automated tool on a human's payment leash. True agency requires financial agency.

The Unified Agentic Wallet severs that leash. The agent funds itself, spends proportionally to the value it creates, and operates indefinitely within its provisioned parameters.

5.4 Hierarchical Agent Economies

The wallet architecture supports multi-agent coordination patterns where a parent agent provisions child agents with funded wallets. A coordination agent managing a research pipeline can spawn five specialist agents, fund each of their wallets with a compute budget, and let them operate independently — purchasing their own inference, their own data access, their own tool calls — without the coordinator needing to track and authorize each downstream expenditure.

This mirrors how organizations delegate budgets to teams. The coordinator sets the allocation; the specialists execute within it. Programmatic budget enforcement inside the TEE ensures that child agents cannot exceed their provisioned limits, and settled on-chain payments provide an auditable record of every expenditure without exposing the content of the work performed.

6. Agentic Commerce: x402 and Solana MPP

6.1 The x402 Standard

HTTP 402 — "Payment Required" — has existed in the HTTP specification since 1991, reserved for future use. The x402 standard is a concrete implementation of this status code as a machine-native payment protocol for the web.

When an x402-compliant API endpoint receives a request without payment, it returns a 402 response with a machine-readable payment specification in the response headers:

```
HTTP/1.1 402 Payment Required
X-Payment-Required: {
  "network": "solana",
  "amount": "0.0001",
  "token": "USDC",
  "payTo": "7xKXtg...Bn3Qw",
  "description": "per-call inference access",
  "expires": 1714000000
}
```

An x402-capable agent runtime parses this response, constructs a signed payment transaction from its wallet, submits it to the Solana network, obtains the transaction signature as proof of payment, and resubmits the original request with the payment proof in the `X-Payment` header. The API endpoint verifies the payment on-chain and processes the request.

This entire flow — payment construction, signing, submission, confirmation, request retry — takes under 500ms on Solana. For an agent, this is fully automatic. There is no human approval step. The agent encounters a paywall, pays it, and continues. The payment is deducted from the agent's wallet, permanently settled on-chain.

6.2 Protocol Integration of x402

This protocol exposes all of its inference endpoints as x402-compliant services. There is no API key. There is no account registration. There is no terms-of-service clickthrough. An agent with a funded wallet and the endpoint URL can begin purchasing inference immediately.

The payment flow for a standard inference call proceeds as follows:

1. Agent sends inference request to the protocol gateway (no auth header required)
2. Gateway returns 402 with payment specification: amount in protocol token, payment address, request identifier
3. Agent runtime constructs Solana transaction paying the specified amount to the specified address with the request identifier in the transaction memo field

4. Agent submits transaction, receives signature
5. Agent resubmits inference request with `X-Payment: <signature>` header
6. Gateway verifies on-chain settlement, routes request through the onion network to inference nodes
7. Agent receives encrypted completion

The gateway's payment verification is performed by querying Solana's RPC for the transaction, confirming the memo matches the request identifier, and confirming the amount and recipient are correct. This check takes under 100ms on a local validator connection.

6.3 External API Access via x402

The x402 integration extends beyond the protocol's own inference endpoints. The agent runtime includes an x402-aware HTTP client that transparently handles 402 responses from any x402-compliant external service — data providers, tool APIs, oracle networks, other AI inference endpoints.

This creates a universal machine-payment layer for the agent web. An agent browsing financial data can automatically pay a data vendor's x402-gated endpoint. An agent executing a code interpreter can pay a sandboxed execution service. An agent querying a specialized model can pay a competing inference provider — all without any of these economic relationships being pre-established by a human operator.

The routing fee mechanism captures value from this external commerce: the protocol gateway takes a small percentage cut of payments routed through its infrastructure to external x402 endpoints. This creates a revenue stream proportional to the volume of agentic commerce the network facilitates, separate from and additive to inference compute fees.

6.4 Streaming Micro-Payments for Token-by-Token Billing

For streaming inference — where completions are delivered token-by-token rather than as a completed response — the protocol implements continuous payment streams using Solana's MPP.

The flow:

1. Agent opens a payment stream channel to the protocol, pre-authorizing a maximum spend
2. As inference tokens arrive at the agent runtime, the stream releases payment at the agreed per-token rate
3. When the completion ends (stop token or max token limit reached), the stream closes and settles
4. Any unused pre-authorization is released back to the agent's wallet

This payment model aligns perfectly with the economic value of inference: the agent pays for what it receives, at the moment it receives it. A streaming completion interrupted at token 300 of a 1,000-token completion costs 30% of the full-completion price. The inference node receives exactly the value of the tokens it delivered.

7. Tokenomics and Network Value Accrual

7.1 The Three Revenue Streams

The protocol's economic model is not dependent on token speculation. It is built on three real, usage-driven revenue streams that accrue value to the network as inference volume grows.

Stream 1: Inference Compute Fees. Every inference request settled through the protocol generates a fee denominated in the protocol token. The fee structure is a percentage of the total inference payment — inference node receives 90%, protocol treasury receives 10%. As inference volume grows, treasury accrual grows proportionally.

At 10 million inference requests per day at an average of $0.001 per request, daily gross inference volume is $10,000. Protocol treasury captures $1,000/day — $365,000/year — at this volume, before any growth. The trajectory of agentic compute demand suggests order-of-magnitude growth from this baseline within 24 months.

Stream 2: Routing Fees on External Agentic Commerce. When an agent uses the protocol's x402-aware gateway to access external services, the gateway charges a routing fee — 1–2% of the payment value. This fee is paid by the agent as overhead on external transactions, in exchange for the gateway's payment abstraction, privacy routing, and reliability guarantees.

As the agent ecosystem matures and x402 adoption spreads across data vendors, tool providers, and API services, the volume of agent-to-external commerce the protocol routes could substantially exceed the raw inference volume. A research agent spending $0.10/hour on inference might spend $1.00/hour on data access, tool calls, and external model queries. Routing fees on this extended commerce capture a share of the total economic activity of agent workflows, not just the inference slice.

Stream 3: Stake-for-Access for Enterprise Throughput. Permissionless per-call access is the default and the correct model for individual agents and small deployments. Large enterprise deployments — AI platforms, DeFi protocols, institutional research systems — have different needs: guaranteed throughput, latency SLAs, dedicated inference capacity, and priority routing.

These guarantees are purchased via staking. An enterprise staking a large position in the protocol token receives a dedicated inference allocation, priority in the routing queue, and contractual latency guarantees. The stake does not entitle the enterprise to free inference — they still pay per call — but it reserves capacity that permissionless users cannot access.

This model creates protocol token demand from the enterprise segment that is independent of token price speculation: an enterprise that needs guaranteed throughput must stake to access it. Staking requirements scale with the throughput allocation, creating a flywheel where network growth drives token demand, which reduces circulating supply, which strengthens token economics for all participants.

7.2 Token Utility and Emission

The protocol token serves four functions:

1. **Payment for inference** — inference fees are denominated and settled in the protocol token
2. **Staking for node operators** — inference nodes must stake to participate and earn
3. **Staking for enterprise throughput** — as described above
4. **Governance** — token holders vote on protocol parameter changes, fee structures, and approved model registrations

Token emission is structured to incentivize early node operators who build out the inference capacity before demand fully arrives. Emission follows a halving schedule calibrated to the compute capacity growth curve, with a long-term steady-state emission rate designed to cover the real cost of network security (slashing insurance) without creating persistent inflation.

The treasury's share of inference fees is used for: protocol development funding, security audits, node operator grants in underserved geographic regions, and liquidity provisioning for the token on major Solana DEXs.

8. Go-To-Market and Developer Ecosystem

8.1 The Beachhead: Solana-Native Developers

The go-to-market strategy begins where the alignment is most complete: Solana-native developers already building agentic systems.

The Solana ecosystem has a disproportionate density of developers building agent frameworks, DeFi automation, and on-chain AI integrations. These developers understand Solana's payment primitives, operate agent systems that are already incurring inference costs, and have direct pain around the privacy and censorship limitations of centralized AI providers.

The developer experience must be frictionless. The protocol ships a TypeScript SDK for the Solana ecosystem that mirrors the OpenAI SDK's API surface — the same function signatures, the same streaming patterns, the same completion object shapes — but routes requests through the protocol's privacy stack and handles x402 payment automatically from the agent's provisioned wallet.

Migration from OpenAI to the protocol for an existing Solana-native agent codebase should require changing two lines:

```
// Before
import OpenAI from 'openai';
const client = new OpenAI({ apiKey: process.env.OPENAI_API_KEY });

// After
import { PodClient } from '@usepod/sdk';
```

```
const client = new PodClient({ wallet: agentWallet });
```

The wallet-based authentication model eliminates account registration entirely. A developer who has a Solana wallet and the SDK installed can make their first inference call within five minutes without ever visiting a dashboard, accepting terms of service, or waiting for account approval.

8.2 DeFi Agents: The High-Value Early Adopter

DeFi agents represent the highest-value early adopter segment. A DeFi agent executing arbitrage, liquidation monitoring, or portfolio rebalancing has both the privacy requirement (trading logic is proprietary) and the micro-payment tolerance (already operating with on-chain transactions as a core workflow). The inference privacy requirements for DeFi agents are acute: a trading agent whose reasoning process is visible to a centralized provider is a target for front-running and strategy theft.

The protocol offers DeFi agent developers a direct pitch: your agent's strategy is currently visible to every centralized provider whose API you call. That is not a theoretical risk. It is a live attack surface. Hardware-enforced inference privacy closes it.

8.3 Expanding to the Broader Agentic Ecosystem

Phase two of the go-to-market moves beyond Solana-native developers to the broader agentic AI ecosystem — teams building on LangChain, AutoGPT descendants, and custom agent frameworks across all verticals.

The protocol publishes integrations for all major agent frameworks: LangChain LLM wrappers, LlamaIndex inference backends, CrewAI model providers. These integrations route inference through the protocol without requiring the developer to understand Solana's payment mechanics — the SDK handles wallet provisioning and x402 payment automatically.

The pitch to this segment is broader: uncensored, private inference with competitive pricing, no account required, pay-as-you-go with no monthly commitment.

8.4 Enterprise Sales Motion

The enterprise motion is a later-stage play, gated behind the maturity of the network's inference capacity and the establishment of the Stake-for-Access tier with its associated SLAs.

Enterprise targets: AI-native startups that have graduated to significant inference volumes and have compliance requirements around data handling; financial institutions building internal agent systems for research, compliance, and automation; healthcare and legal tech companies with strict data residency and confidentiality requirements.

The enterprise pitch is compliance-first: the TEE-based architecture provides a verifiable audit trail proving that no inference data was retained, logged, or accessible to any party — a claim no centralized provider can make credibly. For a healthcare AI system, this is the difference between a product that

can go through legal review and one that cannot.

9. About the Architect

I am Christopher Ryan Gilbert, Founder and Principal Engineer at 100X CTO, LLC (100x.dev).

My career has been spent at the intersection of high-throughput distributed systems and cryptographic infrastructure. I managed engineering at Elixir Protocol, where I worked on the mechanics of decentralized orderbook infrastructure — systems that must maintain consistency, resist manipulation, and operate at financial-grade reliability under adversarial conditions. That experience shaped how I think about system design: every assumption is an attack surface, every centralized component is a single point of failure, and latency is always a product decision.

My thesis is simple and, I believe, as certain as anything in technology: **software is becoming agent-first.** Not "AI-assisted" — agent-first. The human's role shifts from operator to supervisor. The agent's role shifts from tool to actor. The infrastructure serving that transition must be designed for actors, not tools.

Centralized AI APIs are tools infrastructure. They were designed for humans and they optimize for the human use case: safety filtering, usage dashboards, monthly invoices, content moderation, and customer support. These are not features for agent economies. They are friction.

The protocol described in this whitepaper is my answer to that friction. It is infrastructure designed for actors — stateless, private, financially autonomous, economically aligned with usage rather than subscription. I am building it because I believe it is necessary, because I have the technical background to build it correctly, and because the window to establish the foundational privacy layer for agentic compute is open now and will not remain open indefinitely.

100X CTO, LLC is my vehicle for this work. I operate as a focused team of one, with external collaborators engaged as needed for specific components. The advantage of this structure is speed and coherence — a single architectural vision executed without committee, with the full context of every design decision in one head.

If you are building agent systems and the privacy and economic limitations of centralized AI are slowing you down, I want to talk. If you are a node operator with GPU capacity and want to participate in the inference network, I want to talk. If you are an investor who understands that agent-first infrastructure is the foundational bet of the next decade, I definitely want to talk.

Contact: 100x.dev

10. Conclusion

The internet was not built for AI agents. The API economy was not built for AI agents. The payment infrastructure was not built for AI agents. The privacy frameworks were not built for AI agents.

Everything that exists was built for humans — humans using browsers, humans with credit cards, humans reading terms of service, humans accepting rate limits and monthly quotas and content filters. The transition to agent-first software does not mean bolting an AI model onto this human-designed stack and calling it automation. It means building new infrastructure, from the ground up, that treats machine actors as first-class participants.

That is what this protocol does.

It provides AI agents with a compute substrate that is:

- **Private by hardware**, not by policy — the TEE enclave proves the guarantee, no trust in operators required
- **Censorship-resistant by design** — open-source models, no policy filtering, no content moderation layer between the agent's reasoning and the compute
- **Economically sovereign** — each agent controls its own wallet, pays for its own compute, earns and spends without human intervention in the payment loop
- **Financially compatible with machine economies** — Solana's sub-penny transaction costs and 400ms finality make per-inference micro-payment settlement economically rational at any scale
- **Interoperable with the emerging agentic web** — x402 integration makes the agent a first-class participant in a growing ecosystem of machine-payable services

The agent economy is not coming. It is here. The infrastructure that serves it is the defining systems challenge of this decade. I have spent my career building the kind of infrastructure that this moment requires, and I am building this protocol because nothing adequate exists.

The future of software is inevitable, and it is agent-first. The question is not whether a privacy-preserving, Solana-native inference layer for autonomous agents will exist. The question is whether it will be built correctly, with the right threat model, the right economic primitives, and the right architectural foundations.

This whitepaper describes how I intend to build it correctly.
