IO

Input. Output. Nothing.

A Privacy-First Multi-Model AI Inference Network

Tech, AI & Privacy — as a single stack

Whitepaper v0.1.0 | June 2026

`useio.xyz` | $IO (PumpFun, Solana)

Abstract

IO is a privacy-first AI *provider* with a single guarantee: nothing about a request is written down. It is reachable as an inference layer—from a web client, a CLI, a 21-tool MCP server, language SDKs and messaging relays— with no database, no logs, no analytics, no IP retention, and no accounts. Every response carries a signed cryptographic receipt, `io_receipt_v1`, that attests which model answered, which policy was active, and that no prompt was retained. IO is not a single-model chat wrapper. It is a multi-model orchestration engine: one prompt can be broadcast to four models at once, diffused into a single synthesized answer with per-model attribution, or forked into independent branches and merged back. Sensitive data is redacted before inference; receipts can be chained into tamper-proof audit trails and exported as verifiable proof certificates. The same capabilities are exposed through a 21-tool Model Context Protocol (MCP) server, language SDKs, and messaging relays. The network token, $IO, launches as a fair launch on PumpFun with no pre-sale and no venture allocation. This paper describes the design, the threat model it does and does not address, the verification mechanism, and the path from today's inspectable guarantees to tomorrow's hardware-attested ones.

Input. Output. Nothing. And we prove it.

1. Introduction

Every conversation with an AI today leaves a record—and not a copy you control. Your prompt, your phrasing, the half-formed idea, the secret pasted in haste, the IP address, the hour you were awake to ask: logged, joined to an identity, retained on a schedule you cannot see, and—more often than the terms admit—folded back into the next model. The most intimate interface humanity has ever built, the one we increasingly think out loud through, runs on a business model of remembering everything. IO begins from the opposite premise: the surest way to protect a thought is to build a system that was never able to keep it.

Mainstream AI services are built on retention. Prompts, responses, IP addresses, device fingerprints and account metadata are logged by default, often retained indefinitely, and frequently used as training data. Privacy, where it exists, is a policy promise rather than a property of the system. A user cannot inspect a provider's storage layer, and a deleted-log claim is unfalsifiable.

IO takes the opposite stance. The system is designed so that there is nothing to retain. Inference happens in memory and is wiped on response. No persistent store exists in the request path. Because absence of a thing is hard to prove, each response is accompanied by a signed receipt that commits to the conditions under which it was produced. Trust is replaced, as far as is technically possible, by verification.

Privacy alone, however, is not a product. A user who must abandon model quality to gain privacy will not switch. IO's second design goal is therefore to make multi-model orchestration a first-class primitive: querying several open-weight models in parallel, synthesizing their outputs, and branching reasoning paths the way a developer branches code. The combination—verifiable non-retention *and* model diversity—is the contribution of this work.

2. Design Goals and Non-Goals

Goals.

1. No request data is persisted anywhere in the inference path.
2. Every response is cryptographically attributable and self-describing.
3. Multi-model querying (broadcast, diffuse, fork) is native, not bolted on.
4. Sensitive data can be stripped before any model observes it.
5. Guarantees are independently verifiable without trusting the operator.

Non-goals. IO does not claim anonymity against a global network adversary, does not protect against a compromised client device, and— prior to the TEE milestone in Section 8—cannot cryptographically prove server-side memory was never written to disk. We state these boundaries plainly rather than overclaim. The current guarantee is inspectability; the v1.0 guarantee is hardware attestation. The difference is made explicit throughout.

3. Products

IO is best understood not as an application but as a *private AI provider*: an inference layer that any human or agent can reach—through a web client, a CLI, an MCP server, language SDKs or a messaging bot—while retaining nothing. The products below are surfaces over one shared substrate. Each surface inherits the same three properties: open-weight multi-model inference, redaction before the model observes input, and a signed receipt on every response. Privacy, in other words, is not a product tier; it is the default of the provider. The through-line across the catalog is the project's thesis—*tech, AI, and privacy as a single stack*, where the privacy guarantee is enforced by architecture and attested by cryptography rather than asserted by policy.

3.1 IO Chat — Multi-Model Private AI

Status: v0.1, live.

A user opens `useio.xyz`, with no account and no sign-in, and chooses how a prompt is dispatched:

Solo. One prompt, one router-selected model, one answer, with a self-destruct timer counting down.

Broadcast. One prompt fired at all four models simultaneously, returned side-by-side. Each answer carries its own receipt and timer, exposing disagreement between models.

Diffuse. Broadcast plus automatic synthesis into a single answer, with an attribution panel naming which model contributed which insight.

Fork. A conversation is split into independent branches—each with its own context, receipt chain and timer—and merged back once the better path is found.

Self-destruct timer. Every conversation, branch and mode has a configurable TTL with a visible countdown. Presets: flash (tab close), 1 minute, 5 minutes, 1 hour, 24 hours, session. There is no recovery, because the server never held the data.

Retention. Prompts, responses, history, IP, device fingerprint and account data are never stored. Inference is memory-only and wiped on response. No `localStorage`, `IndexedDB` or cookies; a refresh is a wipe.

Table 1: IO MCP tool catalog (21 tools).

Group	Tools
Core inference	`io_completion`, `io_stream`, `io_generate_image`
Orchestration	`io_broadcast`, `io_diffuse`
Privacy & proof	`io_redact`, `io_proof`, `io_canary`
Session control	`io_timer_set`, `io_wipe`, `io_fork`, `io_merge`
Collaboration	`io_room_create`, `io_room_invite`, `io_room_send`
Audit	`io_verify_receipt`, `io_receipt_chain`, `io_export_receipt`
Discovery	`io_list_models`, `io_quote_completion`, `io_create_paid_request`, `io_health`

3.2 IO Code — Multi-Path Coding Agent

Status: v0.2, in build.

Where conventional coding agents propose one solution, IO Code proposes several in parallel and lets the developer fork, explore and merge. A refactor request is diffused across three models, returning distinct approaches as side-by-side unified diffs; nothing is applied without review. The `io -redact` flag scans for keys and secrets locally so the model sees `[REDACTED]` rather than the value. Every generation carries a receipt, and receipts chain into an end-to-end audit trail. A linter and type checker run against each proposal before it is shown.

3.3 IO MCP — The Orchestration Layer

Status: v0.1.3, live. Endpoint: `https://mcp.useio.xyz/mcp`

IO MCP exposes 21 tools spanning core inference, multi-model orchestration, privacy and proof, session control, collaboration, and audit (Table 1). It is intended as an orchestration layer for agent workflows rather than a thin prompt-passing connector.

Five capabilities distinguish the layer: *broadcast and diffuse* (query and synthesize across models with attribution); *fork and merge* (`git`-style branching for conversations); *redact-before-infer* (strip wallets, keys, emails, identifiers before the model observes them, with a receipt proving what was redacted); *proof certificates* (a downloadable PDF with a QR code allowing any third party to verify a receipt without cryptographic expertise); and *receipt chaining* (a Merkle chain in which breaking one link invalidates the whole, suitable for auditing long agent runs). A daily warrant canary (`io_canary`) completes the set.

3.4 Planned Surfaces

The remaining products extend the same primitives to new surfaces and are on the roadmap (Section 12): **IO Room** (end-to-end encrypted collaborative rooms with a private per-participant AI sidebar), **IO Canvas** (an infinite whiteboard with an AI co-creator), **IO Files** (document analysis with field-level redaction before inference),

IO Memory (client-side encrypted agent memory where the user holds the keys), **IO Agents** (autonomous agents whose every step carries a receipt), **IO Relay** (Telegram, WhatsApp, Signal, Discord and SMS bridges), and the **IO SDK** (Rust, Python, TypeScript, Go).

Listing 1: Five-line SDK integration.

```
from io_sdk import IO
io = IO()
r = io.completion("Explain ZK proofs",
                  model="dolphin",
                  ttl=300, redact=True)
print(r.text, r.receipt.verify())
```

4. Provider Capabilities and Utility

The products in Section 3 are surfaces; the value of the network is the set of *provider capabilities* beneath them and the *utility* each creates for $IO. We separate the two deliberately. A capability is something the network can do; a utility is a reason a token must move. A privacy project that lists features but no demand for its token is a liability, so each capability below is mapped to the demand it generates.

4.1 Provider Capabilities

Private inference. Open-weight completion, streaming and image generation with zero retention and a signed receipt. The base unit of the provider.

Multi-model orchestration. Broadcast to four models and diffuse their outputs into one attributed answer. Intelligence is amplified through model diversity, not a single vendor's weights.

Redaction-before-inference. PII, wallets, keys and secrets are stripped before any model observes the input, and the receipt proves what was redacted.

Verifiable receipts. Every response is self-describing and signed; receipts chain into Merkle audit trails and export as portable proof certificates.

Ephemeral sessions. Protocol-level self-destruct with configurable TTL on every conversation, branch and room.

Programmatic access. The same capabilities reachable from CLI, MCP (21 tools), four-language SDKs, and messaging relays—one substrate, many doors.

Metered settlement. Pay-per-use inference settled through the x402 micropayment protocol, with discounts when paid in $IO.

4.2 Utility: Why $IO Must Move

$IO is the unit of account and access for the provider. Its demand is a function of usage, not of speculation alone. Four distinct utilities drive that demand (Table 2):

Table 2: Each surface maps to a provider utility and a source of token demand.

Surface	Provider utility	$IO demand
Chat	Private multi-model inference	payment + access
Code	Multi-path generation + redaction	payment
MCP	Agent orchestration, 21 tools	payment + access
SDK	Embeddable provider	payment
Relay	Inference on messaging surfaces	payment
Files	Redacted document analysis	payment
Memory	Client-encrypted agent memory	payment + access
Agents	Receipt-audited autonomy	payment + burn
Room / Canvas	Encrypted collaboration	payment + access

1. **Payment.** Inference is paid for in $IO. Every completion, scan, broadcast and proof certificate is a metered call settled through the x402 micropayment protocol, with paying in $IO discounted relative to other assets.
2. **Access.** Premium provider features—higher rate limits, broadcast/diffuse, priority routing, proof-certificate generation—are gated by paying in $IO, so heavier use of the network means heavier use of the token.
3. **Burn.** A portion of *net* protocol revenue (after GPU and bandwidth) is used for buyback-and-burn, making circulating supply a function of real usage rather than emission.
4. **Governance.** At v1.0, holders vote on provider parameters—the active model registry, redaction policy, fee splits and the warrant-canary cadence.

The design intent is a flywheel: more surfaces create more inference, more inference is paid for in $IO, payment funds the buyback-and-burn, and burn tightens supply against rising real usage. Crucially, every link in that loop is backed by a metered, real service—the provider sells private inference, and the token is how that service is priced and governed.

5. The Model Lane

IO routes across a curated roster of open-weight and frontier models, chosen so that when upstream ships better weights or APIs, the network upgrades without re-architecting. The roster is split into two lanes: text/code completion and image generation.

Routing across the text roster is handled by a small classifier (7–14B) detailed in Section 6: in Solo mode it selects a single specialist; in Broadcast and Diffuse modes every model is queried. Image requests are dispatched to

Table 3: Text & code model routing.

Model	Strength	When used
Dolphin-Mistral-24B	General, default	Default
DeepSeek	Code, math	Technical queries
Qwen	Multilingual	Non-English
Llama 3.1 70B	Long context	Document, multi-step
Claude (Anthropic)	Reasoning, safety	Complex analysis

Table 4: Image generation model routing.

Model	Provider	Strength
Imagen 3	Google (Gemini)	Photorealism, fine detail
DALL · E 3	OpenAI (GPT)	Creative, prompt-faithful
Stable Diffusion XL	Open-weight	Customisable, fast

the image lane and the receipt records which generation model and which policy version produced the output.

6. AI Architecture

IO's intelligence is not the property of any single model—it is the property of how models are *composed*. The AI architecture is an orchestration engine that treats a prompt the way a build system treats source: it is classified, routed, optionally fanned out across specialists, and reduced to one answer—while every stage emits a verifiable record. Five mechanisms define the layer.

6.1 Intent Router

A lightweight classifier (7–14B) reads the prompt and decides, in under three seconds, what kind of work it is—general reasoning, code, math, multilingual or long-context—and which execution mode applies. The router never sees identity: IP and headers are stripped at the edge before the prompt reaches it. Its output is a routing decision and a confidence score, never a stored profile.

6.2 Broadcast — Parallel Fan-Out

In Broadcast and Diffuse the prompt is dispatched to all text models at once. Because the models genuinely disagree, the disagreement is itself signal: where they converge, confidence is high; where they diverge, the user sees the seam rather than a laundered consensus. Each branch runs in an isolated context and returns its own receipt and timer.

6.3 Diffuse — Synthesis with Attribution

Diffuse adds a synthesis pass. A reducer reads the parallel answers and composes a single response, but—unlike a black-box ensemble—it preserves provenance: an attribution panel names which model contributed which insight. Synthesis is additive (merge complementary reasoning), not a majority vote, so a lone correct answer is not outvoted by three plausible wrong ones. The reducer pass is itself receipted.

6.4 Fork and Merge — Branching Reasoning

A conversation is a tree, not a line. Fork splits a thread into independent branches—each with its own context window, receipt chain and self-destruct timer—so competing approaches are explored in parallel and the winning path merged back. These are `git` semantics for thought: cheap branches, explicit merges, and no history left behind, because there is no history to leave.

6.5 Redaction Pipeline

Before any model observes input, a redaction pass strips wallets, keys, emails and identifiers, substituting typed placeholders (`[REDACTED]`) that preserve grammatical structure so the model still reasons correctly. The receipt records *which classes* were redacted without recording the values— privacy becomes an auditable event rather than a promise.

Crucially, each of these stages—route, branch, synthesis, redaction—is bound into the response's `io_receipt_v1`. The AI architecture and the verification architecture are the same architecture: orchestration does not sit beside proof, it *emits* proof. Intelligence and attestation are produced in a single pass (Section 8).

7. System Architecture

Inference pipeline.

1. **Router** (7–14B, <3 s): classify, route, acknowledge.
2. **Retrieval** (optional): isolated and identity-stripped.
3. **Inference**: solo, broadcast or diffuse.
4. **Stream + receipt**: tokens stream live; `io_receipt_v1` is signed.
5. **Wipe**: memory is zeroed; nothing remains.

Table 5: The stack.

Layer	Technology
Protocol	Rust, actix-web, WebSocket, SSE
Inference proxy	Stateless Rust binary, hash-published
Inference	vLLM / SGLang, open-weight only
GPU	RTX PRO 6000 Blackwell Max-Q, 96 GB ECC
Edge	Cloudflare Workers, IP stripping
Storage	None; memory-only
Receipts	Ed25519, offline key, published pubkey
Telemetry	None

8. Cryptographic Verification

Each response commits to its production conditions in a signed receipt:

Listing 2: `io_receipt_v1`.

```
receipt_v1:
```

```
model:              dolphin-mistral-24b
policy:             io-boundary-0.3
worker_binary_hash: sha256:9f1e...c2a4
enclave_quote:      tee_attestation_v1
retained_prompt:    false
signature:          0x8a3f...
```

A receipt is verifiable offline against the published Ed25519 public key, via SDK, MCP, CLI or web.

What the receipt proves today, and what it will prove. We are deliberate about the boundary. *Today*, a user can inspect source, watch the network tab, confirm no client-side storage is written, and verify the signature binds the response to a named model and policy. This makes *misattribution* and *client-side leakage* detectable. It does not, by itself, prove that server memory was never paged to disk. *At v1.0*, workers run inside TEE-attested enclaves; the `enclave_quote` field then binds the receipt to a measured binary running in a hardware-isolated environment, at which point `retained_prompt: false` becomes a hardware-backed claim rather than an operational one. The roadmap exists precisely to close this gap, and we do not present the v1.0 guarantee as though it already shipped.

9. The $IO Token

$IO coordinates access and payment across the network. It is a fair launch on PumpFun: no pre-sale, no venture allocation. Its roles are payment for inference, settled through the x402 protocol and discounted when paid in $IO; access to premium provider features gated by $IO; and buyback-and-burn funded by network revenue.

A note on fee policy. Inference is not free to serve; Blackwell GPUs and bandwidth are real costs. Any buyback-and-burn program therefore operates on *net* protocol revenue after infrastructure, not on gross inference fees. The precise split between burn and an operations reserve will be published as an on-chain parameter before it is activated, rather than asserted here as a fixed promise. We prefer a sustainable mechanism that can pay for the hardware the privacy guarantee depends on over a headline number that cannot.

10. Doctrine

1. Nothing is written down.
2. You arrive anonymous.
3. Your inputs are not training data.
4. We do not tune for agreement.
5. Guarantees are verifiable, not promised.
6. No permission needed to begin.

Table 6: IO versus a single-model, single-answer baseline.

Feature	Baseline	IO
Chat	1 model	Solo/Broadcast/Diffuse/Fork
Self-destruct	no	every session
MCP tools	8	21
Broadcast	no	yes
Diffuse	no	yes, with attribution
Fork / merge	no	yes
Redact-before-infer	no	yes
Proof certificates	no	PDF + QR
Receipt chaining	no	Merkle chain
Warrant canary	no	daily
Receipts	v1.0 promise	shipped day 1
Token chain	Base	Solana

11. Comparison

12. Roadmap

Table 7: Release roadmap.

Ver.	Deliverable	ETA
v0.1.0	Chat + MCP (21 tools) + $IO	Q2 2026 (live)
v0.2.0	Code + SDK + Terminal	Q3 2026
v0.3.0	Room + Relay (TG, Discord) + API	Q4 2026
v0.4.0	Files + Relay (WA, Signal) + Canvas	Q4 2026
v1.0.0	TEE retention + Memory + governance	Q1 2027
v1.5.0	Agents + OS + Sync + more SDKs	Q2 2027

13. Conclusion

IO reframes AI privacy from a policy a user must trust into a property a user can verify. Non-retention is enforced by an architecture with nothing to retain, and attested by signed receipts that say what produced each answer and under what conditions. Layered on top, multi-model orchestration—broadcast, diffuse, fork and merge—turns privacy from a feature one tolerates into infrastructure one prefers. The current system delivers inspectable guarantees; the roadmap delivers hardware-attested ones, and we are explicit about which is which.

Input. Output. Nothing. And we prove it.

— IO

`useio.xyz` `mcp.useio.xyz/mcp` `@useioxyz`