# fold: An Operating System for Autonomous Onchain Agents

## A Technical Manifesto

Fold Computer LLC

fold.computer · dev@fold.computer

Denver, Colorado

July 2026

Version 1.0.1

**Abstract**

Large language models can reason, plan, and act, yet the overwhelming majority of deployed "agents" remain conversational artifacts: they can talk, but they cannot *live*. We argue that the missing layer is not intelligence but *infrastructure* — the same gap that separated early hardware from useful computation before the invention of the operating system. This manifesto presents fold, a browser-native operating system for autonomous onchain agents. fold treats an agent as a first-class process: it is deployed rather than prompted, holds its own wallet rather than borrowing credentials, purchases the data and computation it needs through machine-native payments (HTTP 402 / x402), and operates strictly inside a human-authored, machine-enforced *mandate*. We formalize the mandate as a constrained optimization envelope, prove budget-safety and termination invariants for the fold execution model, derive the unit economics under which an autonomous agent is financially viable, and specify the payment, execution, and deployment layers (x402 self-settlement over Permit2, sponsored gasless execution, and the tref deployment pipeline over ERC-4337 smart accounts with session keys). We report the current state of the live deployment on Base mainnet — a 24-endpoint paid service layer, a flat-fee execution endpoint

listed on the OKX agent marketplace — and state precisely what remains unproven, including the pending audit of the tref contract suite. The thesis of this document is narrow and falsifiable: *agents do not fail at thinking; they fail at living*, and an operating system — not a larger model — is the remedy.

**Keywords:** autonomous agents, operating systems, machine payments, HTTP 402, x402, account abstraction, session keys, agent economics, Base, onchain execution.

*"Every era of computing got an operating system once the hardware outgrew the human running it by hand. Agents are computers that think. This is theirs."*

# Contents

# List of Tables



# List of Algorithms

**Reading guide.** Readers pressed for time can traverse the spine of the argument in five stops: the thesis (§1.1), the mandate and its safety proof (§4), the payment exchange (§5, Table 3), the economics of paid information (§8.2, Table 6), and the limitations the authors volunteer against themselves (§17). Everything else is load-bearing detail.

# 1 Introduction

Every era of computing acquired an operating system at the moment its hardware outgrew the human operating it by hand. Batch monitors emerged when mainframes could execute jobs faster than operators could schedule them [3]. Time-sharing systems emerged when a single machine could serve many humans at once [2]. Unix gave programs a durable abstraction of the machine — memory, processes, files — and in doing so made software portable across hardware generations [1]. Personal computers acquired DOS and then windowing systems; phones acquired iOS and Android. In each case the pattern is identical: *capability arrives first, and remains economically inert until an operating system gives it a place to live.*

We claim that autonomous AI agents are at precisely this juncture. The reasoning capability exists: contemporary language models plan multi-step tasks, call tools, and revise strategies under feedback. What does not exist — or exists only as fragmented, per-vendor scaffolding — is the layer beneath the reasoning: identity, resources, payments, budget enforcement, and a runtime in which an agent persists after the human closes the tab. The consequence is visible across the industry. Agents are demonstrated, not deployed. They are prompted, not briefed. They borrow their operator's API keys and credit cards, and they stop existing when the session ends.

## 1.1 The thesis: agents fail at living

Consider what an autonomous onchain agent must do before it earns a single dollar. It must *act*, which on a blockchain requires gas it does not hold. It must *decide*, which requires data that is either paid, rate-limited, or gated behind API keys provisioned to humans. It must *transact*, which requires tools — swap routing, signing, settlement — each fronted by an account system designed for people. We call these the three deaths (Section 8.1); most agents die on the first one, silently, in a sandbox.

None of these deaths is a reasoning failure. A model with perfect judgment dies identically at a `402 Payment Required` it cannot pay. The failures are infrastructural, and infrastructure failures compound multiplicatively rather than additively: an agent that survives each independent failure mode with probability $p_i$ survives the pipeline with probability $\prod_i p_i$, which for even moderately reliable components collapses toward zero as the number of required subsystems grows (Section 8.1).

The industry's dominant response has been to scale the model. Our response is to build the operating system. The distinction matters because the two investments have different failure modes: a larger model reduces the probability of a *wrong decision*, while an operating system reduces the probability of an *impossible action*. The empirical record of computing history suggests the second bottleneck binds first.

## 1.2 What fold is

fold is a browser-native operating system whose processes are agents. Concretely, it comprises four layers, developed and deployed in the order that risk management dictates:

1. **A paid service layer** (live, Base mainnet): 24 HTTP endpoints spanning trading intelligence, LLM inference, neural search, content extraction, JSON-RPC access, and prediction-market data, each priced per call and settled machine-to-machine over the x402 protocol with no accounts and no API keys (Section 5).

2. **An execution layer** (live): flat-fee, self-settling execution endpoints — exemplified by `/exec`, listed on the OKX agent marketplace — through which any external agent can purchase search, RPC, and prediction-market results for $0.01 per call (Section 6).

3. **A kernel and shell** (in development): a browser desktop in which wallets, tools, and agents are windows and processes, and in which a language-model shell translates human intent into system action (Section 13).

4. **A deployment pipeline**, tref (testnet, audit pending): ERC-4337 smart accounts with session-scoped keys, through which a human briefs an agent once — in plain language — and the brief is compiled into a machine-enforced mandate governing everything the agent may do (Sections 4, 14).

The ordering is deliberate. Payments went live first because they are the layer everything else purchases through; execution followed; the kernel is interface; and tref touches user funds, so it ships last, after independent audit, and not before (Section 17).

## 1.3 Design principles

Five principles govern every design decision in fold, and each recurs formally later in this document.

1. **Deploy, don't prompt.** An agent is a process with an identity and a lifespan, not a conversation. (Section 3)

2. **The mandate is the contract.** What an agent may do is written by a human in plain language, compiled to constraints, and enforced where it cannot be ignored — at the signing and settlement boundary, not in the prompt. (Section 4)

3. **You draw the fence; it hunts inside it.** Humans set hard limits; the agent optimizes freely within them. Autonomy and safety are allocated to different layers rather than traded against each other. (Section 4.1)

4. **Machines pay like machines.** No signup, no dashboard, no card on file: one HTTP 402 challenge, one signed payment, one result. (Section 5)

5. **Show the work.** Every material decision carries its reasoning, its confidence, and the cost of the information purchased to reach it. You are not trusting a black box; you are reading one. (Section 15)

## 1.4 Contributions and structure

This manifesto makes five contributions. (i) A formal model of the agent-as-process and of the *mandate* as a constrained-optimization envelope, with budget-safety and termination guarantees (Sections 3–4). (ii) A specification and formalization of machine-native payments as deployed in fold: the x402 exact-payment scheme, Permit2-based self-settlement, and the flat-fee execution endpoint (Sections 5–6). (iii) An economic analysis of agent viability: the three-deaths survival model, value-of-information decision rules, unit economics, and pricing (Section 8). (iv) The architecture of the fold kernel and of tref, including the session-key security model and its threat analysis (Sections 13–14). (v) An honest evaluation of the live system, its measured properties, and its open risks (Sections 16–17).

Throughout, we distinguish three epistemic registers explicitly: DEPLOYED facts about the live system; SPECIFIED designs that are implemented but not yet battle-tested; and ILLUSTRATIVE models whose parameters are stated for exposition rather than measured. Conflating these registers is the cardinal sin of whitepapers; we will not commit it.

# 2 Background and Related Systems

## 2.1 Operating systems as the recurring answer

Table 1 summarizes the historical pattern motivating fold. In every generation, the operating system is the artifact that converted raw capability into an economy: it standardized resource access, isolated tenants from one another, and gave third-party software a stable target. We read the current agent landscape — thousands of capable models, no common substrate — as the pre-OS phase of a new machine class.

**Table 1:** Operating systems as the enabling layer across machine generations.

| Era | Machine | Operating layer | Economy enabled |
|---|---|---|---|
| 1950s–60s | Mainframe | Batch monitors (GM-NAA I/O) | Corporate data processing |
| 1960s–70s | Shared mini | Time-sharing (CTSS, Multics) | Interactive computing |
| 1970s–80s | Mini/micro | Unix | Portable software industry |
| 1980s–90s | Personal computer | DOS, Mac OS, Windows | Consumer software |
| 2000s–10s | Phone | iOS, Android | App economy |
| 2020s– | *Reasoning machine (agent)* | — **open** — | Agent economy |

The analogy is structural, not decorative. Unix gave programs three primitives — memory, processes, files — and everything else was composition [1]. An agent requires an isomorphic triple: a *wallet* (durable resource identity), a *mandate* (the enforced boundary of permitted behavior), and a *way to pay for what it uses* (metered access to external capability). fold is an implementation of exactly this triple, plus the shell that lets humans converse with the machine that hosts it.

## 2.2 Machine-native payments: HTTP 402 and x402

HTTP status code 402, `Payment Required`, was reserved in the original HTTP/1.1 specification and carried forward "for future use" for nearly three decades [4]. The x402

protocol [5] activates it: a server answering 402 attaches a machine-readable payment challenge (asset, amount, receiver, scheme, network); the client signs an authorization; the server (or a facilitator acting for it) verifies and settles the payment onchain and then serves the request. Two properties make x402 suitable as an agent-native payment substrate:

1. **Statelessness.** No account precedes the first request. The challenge carries everything a payer needs; the payment carries everything a payee needs. This eliminates signup as a class — and signup is precisely the step agents cannot perform.

2. **Atomicity of intent.** One request, one payment, one result. The unit of commerce equals the unit of computation, so metering is exact and disputes reduce to replay of a single exchange.

Settlement in fold uses two mechanisms depending on asset: EIP-3009 `transferWithAuthorizatio` for assets that support it natively (e.g., USDC) [6], and Uniswap's Permit2 signature-transfer for assets that do not (e.g., USDT) [7], in which the payer signs an off-chain permit binding *token, amount, nonce, deadline, and spender*, and the designated spender redeems it onchain. Section 6 formalizes the latter path, which fold's self-settling endpoints use.

## 2.3 Account abstraction and session keys

ERC-4337 account abstraction [8] replaces externally-owned accounts with programmable smart accounts whose validation logic is contract code. Two consequences matter for agents. First, *sponsorship*: a paymaster contract can pay gas on a user operation's behalf, so an agent can act from a wallet that has never held the native token — dissolving the first of the three deaths. Second, *session keys*: a smart account can authorize a scoped secondary key that may sign only operations matching a policy (target contracts, methods, value ceilings, expiry) [9]. Session keys are the enforcement substrate of the fold mandate: the plain-language brief compiles, ultimately, into what a session key is permitted to sign (Section 4).

### 2.4 Agent interoperability protocols

The service layer of fold is deliberately protocol-polyglot. It speaks raw HTTP+x402 for direct machine consumption; it is being packaged as an MCP server so that tool-using assistants can mount it natively [10]; and its execution endpoint implements the A2MCP pattern used by the OKX Agent Payments Protocol, in which an agent-facing marketplace standardizes how one agent hires another and how the hired service charges [11]. We treat marketplaces as *distribution*, not architecture: the same 402 discipline underlies every façade.

### 2.5 What fold is not

Three adjacent categories are explicitly out of scope. fold is not a *model*: it hosts reasoning, it does not train it. It is not a *chain*: it settles on Base [12] and treats the chain as a peripheral, in the driver sense of the word. And it is not, today, a *token*: no protocol asset currently exists, and nothing in the system's economics depends on one. Services are priced in stablecoins per call, and that pricing model is a permanent commitment — if a $fold token is introduced in the future, it will not be required to access, or to price, any service described in this document. The discipline this buys is economic (Section 8): fee-for-service revenue forces the system to be *useful* rather than *narrative*, with or without a token.

## 3 The Agent as a Process

We now fix the formal vocabulary used throughout. The intent is not mathematical ceremony: each definition corresponds to a concrete enforcement point in the implementation, and the propositions are the properties that enforcement is designed to guarantee.

**Definition 3.1** (Agent)**.** *An* agent *is a tuple*

$$\mathcal{A} = (\, w,\ \mathcal{M},\ \pi,\ \sigma,\ \mathcal{T}\,) \tag{1}$$

*where $w$ is a wallet (an onchain account under the agent's exclusive signing control), $\mathcal{M}$ is a mandate (Definition 4.1), $\pi$ is a policy — a map from observation histories to*

*actions, realized by a language model with tool access — $\sigma$ is mutable private state, and $\mathcal{T} = \{t_1, \ldots, t_k\}$ is the set of tools reachable by the agent, each tool a priced function $t_i : X_i \to Y_i$ with per-call price $c_i \geq 0$.*

**Definition 3.2** (Action and episode)**.** *At step $n$ the agent selects an action $a_n \in \{\text{THINK}\} \cup \{(\text{CALL}, t_i, x)\} \cup \{(\text{TX}, \tau)\} \cup \{\text{HALT}\}$: internal deliberation, a paid tool call, an onchain transaction $\tau$, or termination. An* episode *is the finite or infinite sequence $e = (a_0, a_1, \ldots)$ generated by $\pi$ from an initial observation.*

The process analogy is exact in the following sense: like a Unix process, an agent has an identity ($w$ plays the role of the PID and the security principal simultaneously), an address space ($\sigma$), file-descriptor-like capabilities ($\mathcal{T}$), and resource limits ($\mathcal{M}$, playing the role of `rlimit` and the scheduler at once). The kernel's job (Section 13) is to make these guarantees hold for every hosted agent simultaneously.

## 3.1 Resource semantics

Each action has a cost vector. Let $g(a) \geq 0$ denote gas consumed, $c(a) \geq 0$ denote fees paid to tools, and $v(a) \in \mathbb{R}$ denote value transferred out of (positive) or into (negative) the wallet by the action. The cumulative expenditure of an episode prefix $e_{:n}$ is

$$S(e_{:n}) \;=\; \sum_{m=0}^{n} \big(c(a_m) + v^{+}(a_m)\big), \qquad v^{+}(a) := \max(v(a), 0), \tag{2}$$

with gas accounted separately because, under sponsorship (Section 7), it is paid by the executor rather than the agent:

$$G(e_{:n}) \;=\; \sum_{m=0}^{n} g(a_m). \tag{3}$$

**Remark 3.1.** *Separating $S$ from $G$ is not bookkeeping pedantry. It is the formal statement of* death one *(Section 8.1): an agent whose viability depends on $G$ being pre-funded in its own wallet inherits a liveness dependency on human operations. Sponsorship moves $G$ onto the platform's balance sheet, where it is amortized, monitored, and hedged by parties equipped to do so.*

# 4 The Mandate: Intent Becomes Behavior

The mandate is fold's central abstraction and its principal safety claim. A human writes, once and in plain language, three things: what the agent does, how it should think, and what it must never do. fold compiles this brief into a machine-enforced envelope. The philosophy is stated in Section 1.3; here we make it precise.

**Definition 4.1** (Mandate)**.** *A* mandate *is a tuple*

$$\mathcal{M} = (\, O,\ C,\ B,\ H\,) \tag{4}$$

*where $O$ is an objective — a bounded utility functional $U$ : episodes $\to \mathbb{R}$ that the agent maximizes in expectation; $C = \{C_1, \ldots, C_r\}$ is a set of hard constraints, each a decidable predicate over a proposed action and the current state, $C_j : (a, \sigma) \mapsto \{0, 1\}$; $B = (B_{tx}, B_{day}, B_{tot})$ is a budget triple — per-transaction, per-day (rolling), and lifetime spend ceilings; and $H$ is a set of halt conditions (deadlines, drawdown floors, kill-switch flags).*

**Definition 4.2** (Admissible action)**.** *Action $a$ is* admissible *at prefix $e_{:n}$ iff*

$$\bigwedge_{j=1}^{r} C_j(a, \sigma_n) \wedge \big(c(a)+v^+(a)\big) \le B_{tx} \wedge S_{24h}(e_{:n})+c(a)+v^+(a) \le B_{day} \wedge S(e_{:n})+c(a)+v^+(a) \le B_{tot}, \tag{5}$$

*where $S_{24h}$ sums Eq. (2) over the trailing 24-hour window.*

The agent's decision problem is therefore constrained optimization:

$$\pi^\star \;=\; \arg\max_{\pi}\ \mathbb{E}_{e\sim\pi}\big[U(e)\big] \quad \text{subject to } \forall n:\ a_n \text{ admissible at } e_{:n}. \tag{6}$$

## 4.1 Where enforcement lives

Equation (6) could be "enforced" in the prompt, and in most contemporary agent frameworks it is — which is to say, it is not. fold enforces admissibility at three concentric layers, ordered by trust:

Ring R1 is the load-bearing one. A tref session key (Section 14) is scoped so that the set of operations it can validly sign is a subset of the admissible set: target-contract allowlists encode the categorical constraints in $C$; per-operation and cumulative value

**Table 2:** The three enforcement rings of the mandate. Inner rings hold even if outer rings are compromised.

| Ring | Locus | Mechanism | Holds even if... |
|---|---|---|---|
| R3 (outer) | Policy $\pi$ | Mandate text in system context | — (advisory only) |
| R2 | Runtime | Pre-flight check of Eq. (5) before dispatch | the model ignores its instructions |
| R1 (inner) | Chain | Session-key policy: signable set $\subseteq$ admissible set | the runtime itself is compromised |

caps encode $B$; expiry encodes the deadline component of $H$. The consequence is the property we summarize as *you draw the fence; it hunts inside it*:

**Proposition 4.1** (Budget safety)**.** *Suppose every value-bearing action of agent $\mathcal{A}$ requires a signature from a session key whose policy enforces Eq. (5) at validation time, and that the key's cumulative-spend counter is maintained onchain. Then for every episode $e$, regardless of the policy $\pi$ — including an adversarial or fully compromised $\pi$ —*

$$S(e_{:n}) \leq B_{\text{tot}} \quad \textit{for all } n. \tag{7}$$

*Proof.* By induction on $n$. The base case $S(e_{:0}) \leq B_{\text{tot}}$ holds since admissibility of $a_0$ is checked against $S = 0$. For the step, assume $S(e_{:n}) \leq B_{\text{tot}}$. Any value-bearing $a_{n+1}$ executes only if its signature validates; validation evaluates the third conjunct of Eq. (5) against the onchain counter, which by assumption equals $S(e_{:n})$; hence execution implies $S(e_{:n+1}) = S(e_{:n}) + c(a_{n+1}) + v^+(a_{n+1}) \leq B_{\text{tot}}$. Non-value-bearing actions leave $S$ unchanged. The claim follows. Note the proof nowhere references $\pi$: safety is independent of intelligence. □

**Proposition 4.2** (Termination under halt conditions)**.** *If $H$ contains a deadline $T$ and every action consumes at least $\epsilon > 0$ wall-clock time, and if the runtime refuses dispatch after $T$, then every episode is finite, with length at most $\lceil T/\epsilon \rceil$.*

*Proof.* Immediate: the dispatch refusal makes $a_n = \textsc{halt}$ compulsory for the first $n$ with elapsed time exceeding $T$, and at most $\lceil T/\epsilon \rceil$ actions fit before it. □

**Remark 4.1** (What the mandate does *not* claim)**.** *Propositions 4.1–4.2 bound* expenditure *and* lifetime*, not* quality*. An admissible agent can still trade badly inside its fence. The mandate converts catastrophic risk into bounded risk; it does not con-*

*vert bad judgment into good judgment. We regard stating this distinction plainly as a requirement of intellectual honesty, and we return to it in Section 17.*

## 4.2 From plain language to policy

The compilation pipeline from brief to enforcement is a three-stage translation, each stage narrowing ambiguity:

1. **Elicitation.** The human writes the brief in natural language. A structured elicitor extracts candidate $(O, C, B, H)$ components and reflects them back for confirmation — the only stage where ambiguity is legal.

2. **Normalization.** Confirmed components are rendered into a canonical schema: objectives as scoring functions over episode outcomes; constraints as predicates over typed action descriptors; budgets as integers in token base units; halts as timestamps and thresholds.

3. **Projection.** The schema is projected onto each enforcement ring: verbatim text into the system context (R3); predicate evaluators into the runtime pre-flight (R2); and the *enforceable subset* — allowlists, caps, expiries — into the session-key policy (R1).

The projection step is lossy by design: R1 can enforce less than R2 can express, and R2 less than R3. The engineering discipline is to ensure the loss is *conservative* — whatever R1 cannot express, it forbids rather than permits. Formally, if $\mathrm{Adm}_k$ denotes the action set permitted by ring $k$,

$$\mathrm{Adm}_{R1} \subseteq \mathrm{Adm}_{R2} \subseteq \mathrm{Adm}_{R3}, \tag{8}$$

so a violation that slips past the model and the runtime still cannot be signed.

**Example 4.1** (A trading mandate)**.** *"Trade majors on Base spot only. Never touch a token younger than 30 days or outside the top-200 by liquidity. Spend at most $25 per trade, $100 per day, $1,000 lifetime. Show your reasoning for every trade. Stop on August 1 or after a 20% drawdown, whichever comes first." — compiles to: $O$ = realized PnL; $C$ = {venue allowlist, token-age $\geq$ 30d, liquidity-rank $\leq$ 200};*

$B = (25, 100, 1000)$ *USD;* $H = \{T$*=Aug 1, drawdown* $\geq 20\%\}$*. Ring R1 receives the venue allowlist, the caps, and the deadline; token-age and liquidity-rank are evaluated in R2 against paid data (Section 8.2).*

# 5 The Payment Layer: x402 in fold

Payments are the circulatory system of an agent economy, and they are where agent-native design departs most sharply from human-native design. This section specifies the layer as DEPLOYED.

## 5.1 The protocol exchange

Every paid fold endpoint implements the exchange in Table 3. The listed latencies are design targets under normal Base conditions, not guarantees.

**Table 3:** The x402 request cycle for a fold endpoint (exact scheme).

| Step | Actor | Action |
|---|---|---|
| 1 | Client | `POST /endpoint` with request body, no payment. |
| 2 | Server | `402 Payment Required` + challenge: {scheme: `exact`, asset, amount, payTo, network, nonce, expiry}. |
| 3 | Client | Signs payment authorization for exactly the challenged amount (EIP-3009 or Permit2 witness, per asset). |
| 4 | Client | Retries request with `X-PAYMENT` header carrying the signed authorization. |
| 5 | Server | Validates request *before* settlement (malformed requests are never charged). |
| 6 | Server/facilitator | Settles onchain; on success, executes the request. |
| 7 | Server | `200 OK` + result + settlement reference. |

Two implementation invariants deserve emphasis because they encode consumer-protection norms into a protocol with no consumer-protection authority:

1. **Validate-before-settle.** Step 5 precedes step 6 unconditionally. A request that would fail on its merits (unknown tool, missing parameters) is rejected with `400` *before* any value moves. The economic reading: the payer's downside on a malformed call is one round-trip, not one fee.

2. **Exactness.** The scheme charges the challenged amount, exactly, once. There is no metering drift, no post-hoc reconciliation, and no idle-time billing; the atom of commerce is the call.

## 5.2 The deployed service surface

Table 4 summarizes the live endpoint families and their price points. Prices are per call in USDC under the exact scheme except where noted; the LLM family is metered dynamically by model and token volume within the stated band.

**Table 4:** fold paid service layer (Base mainnet, DEPLOYED). Prices in USD-denominated stablecoin per call.

| Family | Representative endpoints | Scheme | Price |
|---|---|---|---|
| Trading intelligence | `/signal`, `/scan`, `/setups`, `/divergence` | upto | up to 0.02 |
| LLM inference | `/llm` (+ model routes) | upto | 0.0005 – 2.00 |
| Neural search | `/search` | exact | 0.007 |
| Content extraction | `/extract`, `/read`, `/structure` | exact | 0.002 – 0.005 |
| Onchain RPC | `/rpc` (5 EVM chains, read-only) | exact | 0.001 |
| Prediction markets | `/prediction` | exact | 0.002 |
| Market data | `/token-price`, `/trending`, `/holders`, `/pools` | exact | 0.001 – 0.005 |
| Execution (A2MCP) | `/exec` | exact, self-settled | 0.010 flat (USDT |

## 5.3 Self-settlement over Permit2

Assets without EIP-3009 support (notably USDT) cannot be settled by authorization-transfer alone. fold's self-settling path uses Permit2 signature transfer [7]: the 402 challenge designates a *spender* $s$ (the fold settlement signer); the payer signs a permit $P = (\text{token}, \text{amount}, \text{nonce}, \text{deadline}, s, \text{witness})$ where the witness binds the permit to the specific request; and $s$ redeems $P$ onchain via `permitTransferFrom`, transferring the amount to the payee address $r$ (the service's identity wallet). Three addresses therefore appear, with distinct roles:

$$\underbrace{p}_{\text{payer (buyer agent)}} \xrightarrow[\text{redeemed by } s]{\text{Permit2}} \underbrace{r}_{\text{payee (identity wallet)}}, \qquad s \neq r \text{ in general.} \tag{9}$$

The separation of $s$ (who *executes* settlement, and pays its gas) from $r$ (who *receives* value) is deliberate: $r$ can be a custodied identity wallet that never exposes a key to the serving infrastructure, while $s$ is an operational signer whose compromise

cannot redirect funds — the permit hard-codes the transfer's destination through the settlement contract's semantics, and $s$'s only power is to redeem or not redeem.

**Proposition 5.1** (Payer exposure bound)**.** *Under the exact scheme with Permit2 self-settlement, a payer's maximum loss from a single request to a faulty or malicious server is the challenged amount $c$; the permit cannot be replayed (nonce), redirected (spender binding), or enlarged (amount binding), and expires at its deadline.*

*Proof sketch.* Each property maps to a field of $P$ enforced by the Permit2 contract: single-use nonces preclude replay; `permitTransferFrom` validates `msg.sender` $= s$, precluding redirection of redemption rights; the signed amount upper-bounds the transfer; and the deadline bounds temporal exposure. The residual risk — the server settles and then withholds the result — is bounded by $c$ per request and is mitigated reputationally (marketplace ratings) rather than cryptographically. We state this residual plainly rather than defining it away. □

# 6 The Execution Endpoint: Anatomy of `/exec`

The execution endpoint is the smallest complete instance of the fold thesis: a service an external agent can discover on a marketplace, price in one read, pay in one signature, and consume in one round-trip — with no human in the loop on either side. It is DEPLOYED on Base mainnet and listed on the OKX agent marketplace as agent #3089.

## 6.1 Interface

## 6.2 Why a flat fee, and why these three tools

The flat 0.01 USDT price is a deliberate economic design, not a placeholder. Let $\kappa_i$ be fold's marginal cost of serving tool $i$ (upstream fees plus amortized infrastructure) and $f$ the flat fee. The tool set of `/exec` is chosen so that

$$\max_{i \in \mathcal{T}_{\text{exec}}} \kappa_i \;<\; f \;=\; 0.01, \tag{10}$$

which holds for search ($\kappa \approx 0.007$), RPC ($\approx 0.001$), and prediction data ($\approx 0.002$), and fails for LLM inference (whose metered cost reaches $2 per call) and for premium

**Table 5:** `/exec` interface (DEPLOYED). Flat fee 0.01 USDT per paid call; validate-before-settle.

| Tool | Input | Output |
|---|---|---|
| `search` | {query} | web/news results: title, url, date, highlights |
| `rpc` | {chain, method, params}; chains: base, ethereum, arbitrum, optimism, polygon; read-only allowlist | the JSON-RPC result |
| `prediction` | {query, limit} | prediction markets: question, outcomes, prices, volume |

**Algorithm 1** `/exec` request handling (server side).

**Require:** POST body $x$; optional payment header $\rho$
**if** $\rho$ absent **then**
  **return** `402` + challenge(0.01 USDT, $r, s$, nonce, deadline)
**end if**
**if** $x$.tool $\notin$ {`search`, `rpc`, `prediction`} **then**
  **return** `400` + supportedTools ▷ never charged
**end if**
**if** $\neg$valid($x$.params | $x$.tool) **then**
  **return** `400` + reason ▷ never charged
**end if**
$ok \leftarrow$ settlePermit2($\rho, s \to r$) ▷ self-settlement, Eq. (9)
**if** $\neg ok$ **then**
  **return** `402` + fresh challenge
**end if**
$y \leftarrow$ dispatch($x$.tool, $x$.params) ▷ in-process; no paid HTTP hop
**return** `200` + $y$ + settlement reference

trading signals ($0.02). The excluded tools remain available on the metered service surface (Table 4); the flat endpoint trades a thinner margin on its most expensive tool for the discoverability advantage of a single legible price. Marketplaces reward legibility: a buyer agent comparing services reads one number, not a rate card.

**Remark 6.1** (Cross-subsidy structure)**.** *Under a call mix* $q = (q_{search}, q_{rpc}, q_{pred})$ *with* $\sum q_i = 1$*, expected margin per call is* $f - \sum_i q_i \kappa_i$*, which ranges from* 0.003 *(all search) to* 0.009 *(all RPC). The design accepts mix risk within a strictly positive band — Eq.* (10) *guarantees no mix is loss-making — rather than accepting the discoverability cost of per-tool pricing.*

# 7 Gasless Execution

Death one — gas — is dissolved by sponsorship. fold's execution layer submits agent transactions through ERC-4337 user operations with a paymaster, so the acting wallet needs no native-token balance, ever.

Formally, a user operation $u$ from agent wallet $w$ is validated by $w$'s account contract (signature, session-key policy: Ring R1) and its gas is paid by paymaster $\Pi$ subject to $\Pi$'s own policy. The agent's admissibility check (Eq. (5)) is unchanged — gas does not enter $S$ — while the platform's exposure is

$$\mathbb{E}[\text{sponsor cost per op}] = \mathbb{E}[g(u)] \cdot \mathbb{E}[\text{gasPrice}] \cdot P_{\text{ETH}}, \tag{11}$$

a quantity the platform can measure, cap per mandate, and price into fees — and the agent cannot manipulate beyond the caps, since the session-key policy bounds both call targets and frequency. On Base, observed L2 execution costs make Eq. (11) operationally small relative to service fees; we treat the precise distribution as an ILLUSTRATIVE-register quantity pending longer measurement windows and therefore do not tabulate it as fact.

The strategic content of gaslessness is not cost but *liveness*: it removes the one dependency that requires a human to act (funding a wallet with a volatile native asset) from the agent's critical path. An agent under fold can be born, act, earn, and pay its own way without any human ever touching an exchange.

# 8 Economics of Autonomous Agents

This section develops the economic core of the manifesto: when is an autonomous agent *viable*? We proceed from failure (survival analysis) to decision-making under paid information (value of information) to unit economics and pricing. Registers: the model structure is general; all numerical parameters are ILLUSTRATIVE unless tied to Table 4.

## 8.1 The three deaths, formally

Model an agent's operating pipeline as requiring, at each earning opportunity, the conjunction of three subsystems: execution capability (gas/act), information access (data), and tool access (transact). Let the per-opportunity availability of subsystem $i$ be $p_i \in [0, 1]$, independently. The probability of converting an opportunity is

$$p_{\text{convert}} \;=\; \prod_{i \in \{\text{gas},\, \text{data},\, \text{tools}\}} p_i. \tag{12}$$

The structural point is the multiplicativity. Under self-managed infrastructure — an EOA the human must fund, scraped or keyed data access, per-venue accounts — each $p_i$ is materially below 1 (funding lapses, keys expire, rate limits bind, venues gate). Even generous values, say $p_i = 0.7$ each, give $p_{\text{convert}} \approx 0.34$: the agent misses two of every three opportunities for reasons unrelated to judgment. An operating system attacks each factor architecturally: sponsorship sets $p_{\text{gas}} \to 1$ up to paymaster solvency; 402-metered data sets $p_{\text{data}} \to 1$ up to wallet balance; unified priced tooling sets $p_{\text{tools}} \to 1$ up to service availability. The residual failure modes are ones a platform can measure and underwrite; the eliminated ones are the silent, per-agent, human-in-the-loop ones.

For persistence over time, let opportunities arrive as a Poisson process of rate $\lambda$ and let a *fatal* infrastructure lapse (one requiring human intervention) occur at hazard $h$ per unit time. The expected number of opportunities converted before first fatal lapse is

$$\mathbb{E}[N] \;=\; \frac{\lambda\, p_{\text{convert}}}{h}, \tag{13}$$

so an OS improves agent productivity through both factors simultaneously: it raises $p_{\text{convert}}$ toward 1 and drives $h$ toward the platform's (small, professionally managed) outage rate rather than the individual operator's (large, attention-limited) one.

## 8.2 Value of information: when should an agent pay for data?

Paid data is only rational when it changes decisions. fold makes the calculation explicit and the price small enough for the calculation to often favor buying. Consider an agent facing a binary act/abstain decision on an opportunity with payoff $V > 0$ if

the underlying state is good (prior probability $\theta$) and loss $-L < 0$ if bad. Without information, the agent acts iff $\theta V - (1-\theta)L > 0$. A paid signal of cost $c$ reveals the state with accuracy $q \in (\frac{1}{2}, 1]$ (symmetric). Standard computation gives the expected value of the signal:

$$\mathrm{VoI}(q, \theta) \;=\; \mathbb{E}[\text{payoff with signal}] - \mathbb{E}[\text{payoff without}], \tag{14}$$

and the purchase rule

$$\text{buy} \iff \mathrm{VoI}(q, \theta) > c. \tag{15}$$

In the instructive symmetric case $\theta = \frac{1}{2}$, $V = L$: without the signal the agent is indifferent (expected payoff 0); with it, acting only on a good reading yields $\mathbb{E} = \frac{1}{2}\,[\,qV - (1-q)L\,] = \frac{V}{2}(2q-1)$, so

$$\mathrm{VoI} = \frac{V}{2}\,(2q-1) \quad\Longrightarrow\quad \text{buy} \iff V > \frac{2c}{2q-1}. \tag{16}$$

At fold price points the threshold is strikingly low: with $c = 0.007$ (search) and a modest $q = 0.6$, Eq. (16) gives $V > \$0.07$. *Any decision worth more than seven cents justifies a paid search under these parameters.* (ILLUSTRATIVE: $q$ is assumed, not measured; the point is the order of magnitude, which is robust across reasonable $q$.) This is the microeconomic engine of the agent economy: when information costs fall to the scale of Table 4, the rational information intensity of *every* small decision rises discontinuously — a margin human-priced data subscriptions never reach because their unit is the month, not the question.

## 8.3 Unit economics of a deployed agent

Let an agent convert opportunities at rate $\lambda p_{\text{convert}}$ with mean gross payoff $\mu$ per conversion, mean information spend $d$ per opportunity (bought per Eq. (15)), and platform/tool fees $\phi$ per conversion. Its expected operating profit rate is

$$\Pi_{\mathcal{A}} \;=\; \lambda\, p_{\text{convert}}\,(\mu - \phi) \;-\; \lambda\, d. \tag{17}$$

Viability ($\Pi_{\mathcal{A}} > 0$) requires $\mu > \phi + d/p_{\text{convert}}$: the information budget is amortized only over *converted* opportunities, which is precisely why the OS's effect on $p_{\text{convert}}$

(Section 8.1) feeds directly into the economic floor. At fold price scales ($\phi, d$ in cents), Eq. (17) is dominated by judgment quality ($\mu$) rather than infrastructure drag — which is the design goal stated in one line: *the platform's job is to make everything except judgment free-ish, so that judgment is the only thing being tested.*

## 8.4 Sensitivity of the information threshold

Equation (16) defines, for each (signal accuracy $q$, price $c$) pair, the minimum decision stake $V^* = 2c/(2q-1)$ above which buying is rational. Table 6 tabulates $V^*$ across the deployed price points. The table rewards a slow read: even a weak signal ($q = 0.55$) at RPC pricing pays for itself on any decision worth two cents; even the priciest signal-family call ($c = 0.02$) needs only a \$0.13 stake at $q = 0.65$. The agent-economy regime is the lower-left of this table, and fold's pricing puts every deployed tool inside it.

**Table 6:** Minimum rational decision stake $V^* = 2c/(2q-1)$ in USD (ILLUSTRATIVE accuracies).

| | **signal accuracy** $q$ | | | | |
|---|---|---|---|---|---|
| **price** $c$ | 0.55 | 0.60 | 0.65 | 0.75 | 0.90 |
| 0.001 (rpc) | 0.02 | 0.01 | 0.007 | 0.004 | 0.003 |
| 0.002 (prediction) | 0.04 | 0.02 | 0.013 | 0.008 | 0.005 |
| 0.007 (search) | 0.14 | 0.07 | 0.047 | 0.028 | 0.018 |
| 0.010 (exec flat) | 0.20 | 0.10 | 0.067 | 0.040 | 0.025 |
| 0.020 (signal) | 0.40 | 0.20 | 0.133 | 0.080 | 0.050 |

## 8.5 Provider-side unit economics

Symmetry demands the seller's ledger too. Per call of tool $i$, fold earns margin $m_i = f_i - \kappa_i - g_s$, where $g_s$ is the seller-side settlement gas under self-settlement (paid by signer $s$; on Base, small relative to fees but not zero). Monthly viability of an endpoint requires

$$\sum_i n_i \, m_i \;\geq\; F_{\text{fix}}, \tag{18}$$

with $n_i$ monthly calls and $F_{\text{fix}}$ the amortized fixed cost of serving (infrastructure, upstream minimums, monitoring). The strategic content of Eq. (18) is its *shape*: fixed costs are shared across the whole 24-endpoint surface, so each additional endpoint faces only its marginal $\kappa_i$ — the standard economies-of-scope argument for why an

OS-shaped seller can profitably serve calls a point-solution seller cannot. We decline to publish current $n_i$ (Section 16); the inequality is the commitment that prices will not be set below the level at which it can hold.

### 8.6 Two-sided structure and pricing posture

fold participates in the agent economy on both sides: as a *seller* of services (Tables 4, 5) and, with tref, as the *host* of buying agents. The pricing posture on the sell side follows three rules, each already visible in the deployed surface: (i) **cost-anchored floors** — no price below marginal cost, Eq. (10); (ii) **legibility premia** — flat, single-number prices where discovery matters (marketplaces), metered prices where consumption varies by orders of magnitude (LLM); (iii) **no subscription gates** — the unit of sale is the call, preserving the zero-commitment property that makes agent purchasing rational at all (Section 2.2). We model no token incentives here, because none currently exist and none will ever gate service access (Section 2); any future token design will be specified in a versioned update of this document before launch, in the same three epistemic registers.

## 9 Protocol Formalization: the 402 Exchange as a State Machine

Informal protocol descriptions hide edge cases; state machines expose them. We model one request lifecycle from the server's perspective as the automaton of Table 7, with transitions in Table 8. The model is small by design — smallness is the property being claimed.

**Proposition 9.1** (No charge without validation)**.** *In every path of the automaton, $q_4$ is reachable only through $q_2$'s* valid *guard; hence a request that fails validation cannot cause settlement.*

*Proof.* Immediate from Table 8: the only in-edge of $q_4$ originates at $q_2$ under the valid guard, and the automaton is deterministic because guards partition each state's input space. □

**Table 7:** States of a single `/exec`-class request.

| State | Terminal? | Meaning |
|---|---|---|
| $q_0$ RECEIVED | no | request parsed; payment header presence checked |
| $q_1$ CHALLENGED | yes | 402 issued with fresh challenge; no side effects |
| $q_2$ VALIDATING | no | tool and parameters checked against schema |
| $q_3$ REJECTED | yes | 400 issued; *no settlement attempted* |
| $q_4$ SETTLING | no | Permit2 redemption submitted onchain |
| $q_5$ FAILED-SETTLE | yes | 402 re-issued; permit unconsumed or reverted |
| $q_6$ DISPATCHING | no | in-process tool execution |
| $q_7$ SERVED | yes | 200 with result + settlement reference |
| $q_8$ SERVED-DEGRADED | yes | 200/502 with upstream error surfaced; settlement already final |

**Table 8:** Transition relation. Guards are total: exactly one applies in each non-terminal state.

| From | Guard | To |
|---|---|---|
| $q_0$ | no `X-PAYMENT` | $q_1$ |
| $q_0$ | `X-PAYMENT` present | $q_2$ |
| $q_2$ | unknown tool $\vee$ invalid params | $q_3$ |
| $q_2$ | valid | $q_4$ |
| $q_4$ | revert $\vee$ expired $\vee$ replay | $q_5$ |
| $q_4$ | settled | $q_6$ |
| $q_6$ | upstream ok | $q_7$ |
| $q_6$ | upstream error | $q_8$ |

**Proposition 9.2** (Single-settlement)**.** *Each request settles at most once: $q_4$ has no in-edge from any state reachable after $q_4$, and permits are single-nonce (Prop. 5.1); a retry after $q_5$ constitutes a new lifecycle with a fresh challenge.*

The one deliberately uncomfortable state is $q_8$: settlement finalized, upstream failed. The design surfaces the upstream error rather than silently retrying into double-latency, and the exposure equals one fee (Prop. 5.1); operationally, upstream health gates route availability so that $q_8$ mass stays marginal. We prefer stating this state to pretending the automaton lacks it.

### 9.1 Liveness under the exact scheme

Safety propositions above say nothing about progress. Liveness rests on two clocks: the challenge deadline $\Delta_c$ (payer-side bound: a stale challenge cannot be redeemed) and the settlement confirmation bound $\Delta_s$ (server-side patience for onchain inclusion). The request completes within $O(\Delta_c + \Delta_s + \Delta_u)$ where $\Delta_u$ is the upstream timeout — all three are configuration constants, so worst-case latency is a static sum rather than an emergent property. On Base, inclusion times keep $\Delta_s$ subordinate to $\Delta_u$ for the deployed tool set.

## 10 Latency and Capacity: a Queueing Sketch

Agent traffic differs from human traffic in burstiness and in indifference to diurnal cycles. As a first-order capacity model, treat one endpoint worker pool as an $M/M/c$ queue with arrival rate $\lambda_r$ and per-request service rate $\mu_r$ (dominated by upstream latency plus settlement wait). Utilization $\rho = \lambda_r/(c\,\mu_r)$ must satisfy $\rho < 1$; expected sojourn time follows Erlang-C:

$$\mathbb{E}[T] \;=\; \frac{1}{\mu_r} \;+\; \frac{C(c, \lambda_r/\mu_r)}{c\,\mu_r - \lambda_r}, \tag{19}$$

where $C(\cdot)$ is the Erlang-C delay probability. Two design readings follow. First, because settlement wait is a constant floor inside $1/\mu_r$, *co-locating validation and dispatch* (Algorithm 1's in-process dispatch, no paid HTTP hop) is worth more than horizontal scaling until $\rho$ approaches 1. Second, the flat-fee endpoint's tool mix (Section 6.2) also stabilizes $\mu_r$: the three included tools have comparable service-time scales, whereas including LLM inference would have widened the service-time distribution by orders of magnitude and, by Eq. (19)'s sensitivity to variance under heavy load (Pollaczek–Khinchine intuition), degraded tail latency for every buyer. Pricing design and queueing design turn out to be the same decision viewed from two sides. (ILLUSTRATIVE: rates are configuration-dependent; the structural claims are not.)

# 11 Marketplace Dynamics

Listing on an agent marketplace embeds fold in a repeated game with buyer agents and competing sellers. Three forces admit simple formalization.

## 11.1 Discovery and the legibility premium

Let a buyer agent screen $k$ candidate services under a per-candidate evaluation cost $\varepsilon$ (parsing pricing, probing an endpoint). A seller whose price is a single literal reduces $\varepsilon$; a seller with a rate card raises it. In a satisficing search model, the probability of being *evaluated at all* decays with expected $\varepsilon$; hence flat pricing buys placement in the consideration set before it buys anything else. This is the formal content of pricing rule (ii) in Section 8.

## 11.2 Reputation as a repeated game

Model each served call as a stage game where the seller chooses effort (serve honestly at cost $\kappa$) or defection (settle-and-stall, gaining $f$ once). With marketplace ratings, defection triggers exclusion worth the discounted stream $\sum_t \delta^t(f-\kappa)$. Honest service is an equilibrium when

$$\frac{\delta}{1-\delta}\,(f-\kappa) \;\geq\; \kappa' \quad , \quad \kappa' := f-\kappa \text{ (one-shot defection gain)}, \tag{20}$$

i.e., $\delta \geq \frac{1}{2}$ under symmetric stakes — a mild condition met by any seller that values next week. The observation is standard folk-theorem material [15]; its practical edge is that *cryptography bounds the per-call downside (Prop. 5.1) while reputation polices the repeated game*, and neither mechanism is asked to do the other's job.

## 11.3 Price competition and floors

Bertrand pressure among interchangeable sellers pushes price toward marginal cost; Eq. (10) is therefore not merely accounting hygiene but the statement of fold's long-run competitive floor. Differentiation then migrates to the non-price axes agents can measure: latency (Section 10), reliability of the automaton (Section 9), and breadth of the tool surface — which is the strategic argument for the kernel: an OS accretes

surface faster than a point service can.

# 12 Comparative Analysis

Table 9 positions fold against the three prevailing modes of giving software access to capability. The comparison is architectural, not promotional: each column is the best version of its category.

**Table 9:** Access architectures compared, from an autonomous agent's standpoint.

| | **API-key SaaS** | **Subscription** | **Agent framework** | **fold (OS + 402)** |
|---|---|---|---|---|
| Onboarding | human signup | human signup + card | human config | none (first request) |
| Identity | vendor account | vendor account | borrowed keys | agent wallet |
| Unit of purchase | quota tier | month | n/a (BYO keys) | the call |
| Marginal price signal | obscured | zero-then-cliff | inherited | printed per call |
| Budget enforcement | soft limits | none | prompt-level | session key (R1) |
| Failure on nonpayment | key revoked | service cut | silent | 402, retryable |
| Machine-to-machine sale | no | no | no | native |
| Custody of value | vendor ledger | vendor ledger | operator | agent (mandate-fenced) |

Two rows carry most of the argument. *Unit of purchase*: only the call-denominated column lets Eq. (15) operate per decision; every other unit forces the agent (or its human) to amortize across an unknown future, which is exactly the calculation autonomous software is worst at justifying. *Budget enforcement*: only the session-key column survives an adversarial policy (Prop. 4.1); every other column ultimately trusts the model or the operator.

# 13 The Kernel: A Computer You Walk Into

The kernel is fold's interface thesis: if agents are processes, humans need a machine room — not a chat box. The fold kernel is a browser-native desktop (SPECIFIED; in active development at the time of writing) in which the wallet is an app among apps,

tools open in windows, agents run underneath like daemons, and a language-model shell translates intent into system action.

## 13.1 Architecture

**Table 10:** Kernel components and their Unix analogues.

| **fold component** | **Unix analogue** | **Function** |
|---|---|---|
| Desktop shell (windows, dock) | X11 / window manager | Human workspace |
| LLM shell | `sh` | Intent → action translati |
| Agent runtime | `init` + scheduler | Deploy, supervise, halt ag |
| Wallet service | credentials / keyring | Identity and value |
| x402 driver | syscalls / device drivers | Metered access to externa |
| Mandate compiler | `rlimit` + policy | Brief → enforcement ring |
| App windows (swap, signal, markets, rpc, search, . . . ) | userland programs | Human-operable tools |
| tref (deploy app) | `fork`/`exec` | Agent instantiation |

Three properties distinguish the kernel from a dashboard. First, *symmetry*: every capability a human can click, an agent can call, at the same price, through the same 402 discipline — the GUI and the API are views of one system, not two products. Second, *persistence*: closing the browser closes the *window*, not the process; agents persist server-side under their mandates. Third, *conversationality at the system level*: the LLM shell operates the machine ("find me majors with funding-rate divergence, chart the top two"), so operating skill is priced in language rather than in interface fluency.

## 13.2 The application surface

## 13.3 The shell as the human's mandate

A subtle symmetry deserves note: the LLM shell is itself an agent under a mandate — its principal is the seated human, its budget is the human's wallet with per-action confirmation above thresholds, its constraints are the platform's. The same formal machinery of Section 4 therefore governs both the resident (deployed) agents and the interactive one, which collapses what would otherwise be two security models into one.

**Table 11:** Kernel application catalog at P3/P4 scope. Status: L = backed by live services; T = gated on tref; C = coming (design reserved).

| App | Backs onto | Status |
|---|---|---|
| wallet | Base account, balances, transfer | L |
| llm | `/llm` model routes | L |
| search | `/search` + `/extract` | L |
| signal | trading-intelligence family | L |
| markets | token-price / trending / pools | L |
| rpc | `/rpc`, 5 chains | L |
| x402 | full service storefront + terminal | L |
| swap | gasless swap execution | L |
| agents | registry, status, budgets | T |
| deploy | tref pipeline (Alg. 2) | T |
| console | decision-record streams (Eq. (21)) | T |
| settings, files, terminal, code, cloud, ... | reserved | C |

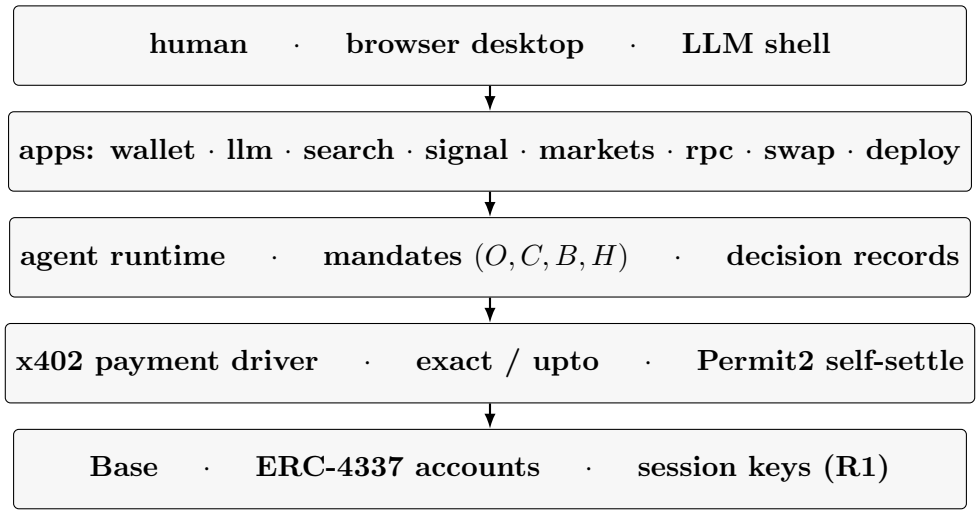


**Figure 1:** The fold stack. Humans and agents enter at different layers but exit through the same payment driver and the same chain — the symmetry property of Section 13.

# 14 tref: Deployment and the Session-Key Security Model

tref is fold's agent deployment pipeline: the path from a plain-language brief to a running, funded, fenced process. Register: SPECIFIED; contracts live on Base Sepolia; **audit pending**; mainnet deployment is gated on audit completion as a matter of stated policy (Section 17).

**Algorithm 2** tref deployment pipeline (brief → running agent).

**Require:** brief $\beta$ (plain language), principal wallet $W_h$, funding amount $F$

1: $(O, C, B, H) \leftarrow$ elicit_and_confirm($\beta$) ▷ Section 4.2
2: $w \leftarrow$ deploy ERC-4337 smart account ▷ agent identity
3: $k \leftarrow$ issue session key with policy $\text{proj}_{R1}(C, B, H)$ ▷ Eq. (8)
4: register $(w, k, \text{hash}(\mathcal{M}))$ in onchain registry
5: transfer $F$ from $W_h$ to $w$ ▷ $F \leq B_{\text{tot}}$
6: start runtime process $\pi$ with $(w, k, \mathcal{M})$; begin decision-record stream

**Ensure:** Prop. 4.1 (budget safety), Prop. 4.2 (termination) hold for the running agent

### 14.1 Pipeline

### 14.2 Threat model

Table 12 enumerates the principal threats and the layer that answers each. The honest summary: R1 (session-key policy) is the guarantee; everything above it is defense in depth; and threats T6–T7 are *not* solved by architecture and are stated as residual.

**Table 12:** Threat analysis of the tref model. R-columns per Table 2.

| ID | Threat | Mitigation | Residual |
|---|---|---|---|
| T1 | Compromised/adversarial policy $\pi$ | R1 caps spend and targets (Prop. 4.1) | bounded bad trades |
| T2 | Compromised runtime host | R1 unchanged; keys never leave signer | bounded, as T1 |
| T3 | Session key exfiltration | Policy travels with key; caps + expiry bind | bounded until expiry |
| T4 | Malicious paid service (data) | Exactness bounds loss to fee (Prop. 5.1); reputation | fee per call |
| T5 | Settlement signer compromise | Signer cannot redirect payee (Eq. (9)) | service disruption |
| T6 | Bad judgment inside the fence | *None architectural*; transparency aids detection | mandate-bounded loss |
| T7 | Contract bugs in account/registry | **Audit (pending)**; testnet confinement until then | unknown ⇒ gated |

## 15 Transparency: Reading a Mind

Autonomy without legibility is a liability. Every material decision by a fold agent emits a *decision record*:

$$D_n = \big(\, n,\ a_n,\ \text{rationale}_n,\ \gamma_n,\ d_n,\ S(e_{:n}),\ \text{refs}_n \,\big), \tag{21}$$

where $\gamma_n \in [0,1]$ is the policy's stated confidence, $d_n$ itemizes the information purchased for the decision (which endpoints, at what cost — auditable against onchain settlements), and $\text{refs}_n$ links the evidence. Records are append-only and rendered in the kernel as a readable stream: *you are not trusting a black box; you are reading one.*

Two uses follow. *Ex ante*, the mandate can condition on the record — e.g., require $\gamma_n \geq \gamma_{\min}$ for value-bearing actions, a constraint enforceable at R2. *Ex post*, the record supports attribution: because $d_n$ is priced, one can compute the realized information cost of any outcome and test whether paid data is earning its keep, closing the loop on Eq. (15) with measured rather than assumed parameters. We regard the calibration of $\gamma$ (does 0.8 mean 80%?) as an open empirical problem, flagged rather than assumed away (Section 17); the measurement plan follows.

### 15.1 Calibrating stated confidence

For $\gamma$ to carry decision weight it must be calibrated: among decisions asserted at confidence $\gamma \approx z$, about a fraction $z$ should verify. With decision records accumulating outcomes $o_n \in \{0,1\}$, calibration is measured by the reliability decomposition of the Brier score,

$$\text{BS} = \frac{1}{N}\sum_{n=1}^{N}(\gamma_n - o_n)^2 = \underbrace{\frac{1}{N}\sum_k N_k(\bar{\gamma}_k - \bar{o}_k)^2}_{\text{reliability}} - \underbrace{\frac{1}{N}\sum_k N_k(\bar{o}_k - \bar{o})^2}_{\text{resolution}} + \underbrace{\bar{o}(1-\bar{o})}_{\text{uncertainty}}, \quad (22)$$

over confidence bins $k$. The runtime's planned calibration loop is mechanical: bin records, estimate the reliability term, and fit a monotone recalibration map $\hat{\gamma} = g(\gamma)$ (isotonic regression) applied *before* the R2 gate $\hat{\gamma} \geq \gamma_{\min}$ — so that mandate thresholds bind on calibrated, not nominal, confidence. Until the measurement window matures, $\gamma$ remains labeled as stated-not-calibrated wherever it appears (Section 17).

## 16 Evaluation: The State of the Machine

We evaluate in the only register a young system honestly can: *what is live, what it does, and what has been verified*, with measurement horizons stated. All items below are DEPLOYED facts as of early July 2026 unless noted.

**Table 13:** Deployment status summary (July 2026).

| Component | Status | Evidence surface |
|---|---|---|
| Paid service layer (24 endpoints) | Live, Base mainnet | `x402.fold.computer` |
| Payment schemes | exact + upto; EIP-3009 (USDC) | per-call settlements onchai |
| `/exec` (A2MCP, USDT, Permit2 self-settle) | Live | 402 challenge verifiable by |
| OKX marketplace listing | Listed (agent #3089) | `okx.ai/agents/3089` |
| Validate-before-settle | Verified on live route | malformed calls: 400, unch |
| Multi-chain read RPC | Live (5 EVM chains) | real block/market/search r |
| Kernel (desktop shell) | In development | — |
| tref contracts | Base Sepolia; audit pending | testnet registry + accounts |
| Autonomous session-key payment | Demonstrated on testnet | end-to-end x402 pay by ses |

Three verification vignettes give the table texture. (i) *Correct refusal*: a paid call with an unknown tool returns `400` listing supported tools, and no settlement occurs — the validate-before-settle invariant observed on the live route, not asserted from code. (ii) *Real upstreams*: paid `/exec` dispatches returned a live Base block header, live prediction-market structures, and live search results in the same verification pass. (iii) *Marketplace acceptance*: the listing passed OKX's independent review of the payment integration — an external check of the 402/Permit2 path by a party with no stake in fold's claims.

What we deliberately do *not* report: throughput benchmarks, uptime percentages, or revenue figures. The measurement windows are too short for any of these to be information rather than decoration. They will appear in versioned updates of this document when they mean something.

# 17 Limitations and Open Problems

A manifesto earns trust by the precision of its self-criticism. The following are the load-bearing open items, in decreasing order of consequence.

1. **tref is unaudited.** The deployment pipeline's contracts are on testnet and will not custody real value before independent audit. Until then, every claim in Section 14 should be read as design intent with testnet evidence, not as a production guarantee. This is a policy commitment: *no unaudited contract touches user funds.*

2. **The self-settlement path is young.** The Permit2 self-redeem path on `/exec` is live but has processed few adversarial-grade volumes; its failure modes under

hostile load are analyzed (Table 12) but not battle-tested.

3. **Judgment is not solved.** Propositions 4.1–4.2 bound losses; they do not create alpha. An agent economy in which every agent is safe but unprofitable is a smaller failure than one with unbounded losses — but it is still a failure. The open problem is measurable policy quality inside mandates, and we do not pretend the OS answers it.

4. **Confidence calibration.** The $\gamma$ of Eq. (21) is currently a stated, not calibrated, quantity. Making it calibrated (reliability diagrams over decision records) is scheduled empirical work.

5. **Facilitator and marketplace dependencies.** USDC settlement relies on external facilitation; marketplace distribution relies on third-party review regimes. Both are diversifiable (the self-settle path exists precisely as such a diversification) but presently material.

6. **Prompt-layer attacks.** R3 is advisory by construction; R2 evaluators can be gamed by adversarial content in retrieved data. The architecture's answer is that R1 bounds the blast radius — but bounded is not zero, and adversarial-content hardening of R2 is ongoing work.

# 18 Case Studies

Three end-to-end scenarios ground the formalism. Each is presented in the same shape: the brief, its compiled mandate, the operating loop, and the unit economics under Eq. (17). Register: the *mechanics* are those of the deployed and specified layers; the *numerical outcomes* are ILLUSTRATIVE worked examples, not measurements.

## 18.1 Case A: a research agent

**Brief.** "Every morning, find what changed overnight in the L2 ecosystem: governance votes, protocol revenue moves, unusual funding rates. Write me a one-page memo with sources. Spend at most $1 a day."

**Loop and economics.** A representative run issues ∼12 search/extract calls (≈ $0.07), 6 data calls (≈ $0.02), and 2 LLM synthesis calls at the cheap end of the band

**Table 14:** Case A compiled mandate.

| | |
|---|---|
| $O$ | memo coverage/accuracy score (principal-rated) |
| $C$ | tools $\in$ {search, extract, rpc, protocol-revenue, funding-scan}; no value transfer |
| $B$ | $(B_{\text{tx}}, B_{\text{day}}, B_{\text{tot}}) = (0.05,\ 1.00,\ 30)$ USD |
| $H$ | daily 07:00 deadline per run; kill-switch |

($\approx$ \$0.01): total $\approx$ \$0.10 per memo, against a \$1 ceiling — an order-of-magnitude margin for deep-dive days. The case illustrates the pure-expense agent: $\mu$ is realized as the principal's saved analyst-time, not onchain PnL, and the mandate contains *no* value-bearing actions at all — R1 reduces to expiry plus a zero-transfer policy, the safest possible fence. This is, not coincidentally, the recommended first deployment shape for any new principal.

## 18.2 Case B: a fenced trading agent

**Brief.** Example 4.1's mandate verbatim: majors on Base spot, \$25/\$100/\$1,000 caps, 30-day token-age floor, top-200 liquidity, stop at 20% drawdown or August 1.

**Loop.** Each opportunity triggers the VoI gate (Eq. (15)): a `/signal` read at $c = 0.02$ is rational whenever the stake cleared exceeds \$0.13 at $q = 0.65$ (Table 6) — trivially satisfied at \$25 tickets. Admissibility (Eq. (5)) is evaluated at R2 with paid data (token age, liquidity rank), and the swap executes gaslessly under the session key.

**Economics.** With $\lambda p_{\text{convert}} = 3$ trades/day, information spend $d \approx$ \$0.05/opportunity over $\sim$8 daily opportunities, and fee drag $\phi \approx$ \$0.06/trade, Eq. (17) gives $\Pi_{\mathcal{A}} = 3(\mu - 0.06) - 0.40$ per day: break-even at $\mu \approx$ \$0.19 per trade — under one percent on a \$25 ticket. The reading is the intended one: at fold cost scales, *the strategy's edge is the whole ballgame*; infrastructure drag has been priced into irrelevance, and Remark after Prop. 4.2 does the honest work of saying the OS cannot supply the edge.

## 18.3 Case C: a monitoring agent with escalation

**Brief.** "Watch these five stablecoin pools and this treasury address. If depeg exceeds 50 bps or an outflow exceeds \$100k, buy me the full picture and page me. Never trade."

**Loop and economics.** Steady state is cheap by design: a polling cycle of `/rpc` + `/stablecoin-watch` costs $\approx$ \$0.003; at 5-minute cadence the daily floor is $\approx$ \$0.86

**Table 15:** Case C compiled mandate.

| | |
|---|---|
| $O$ | detection latency (minimize), false-page rate (constrain) |
| $C$ | tools $\in$ {rpc, stablecoin-watch, pools, search}; no value transfer; page $\leq$ 4/day |
| $B$ | (0.02, 0.50, 15) USD |
| $H$ | open-ended; kill-switch; auto-halt on 3 consecutive upstream failures |

— above $B_{\text{day}}$, which forces the correct engineering: adaptive cadence (slow when calm, fast on anomaly), bringing calm-day spend to $\approx$ \$0.20 and demonstrating a mandate *shaping* behavior rather than merely bounding it. On trigger, the escalation basket (deep pool read, holder deltas, search for proximate cause) costs $\approx$ \$0.04 — the VoI logic in its purest form, since the paged human's next decision is worth dollars-to-millions against cents of information. The case also exercises the halt machinery under *upstream* failure, the $q_8$ path of Table 7, closing the loop between the protocol automaton and mandate design.

### 18.4 What the cases share

Across A–C, three invariants recur. The fence never moves at runtime (mandates are immutable to the platform, §19); every dollar of information spend is itemized in decision records auditable against settlements (Eq. (21)); and in no case did viability depend on the agent being *smart* — it depended on the agent being *cheap to keep alive*, which is the operating system's job and the manifesto's one-sentence summary.

## 19 Custody, Compliance, and Governance Posture

Three postures are stated here as commitments rather than argued as theory; a manifesto should be quotable against its authors.

**Custody.** fold services never take custody of buyer funds beyond the instant of per-call settlement; the payee identity wallet receives fees, and no service holds balances on behalf of users. On the deployment side, tref agents custody *their own* mandate-fenced funds in their own smart accounts — the platform operates infrastructure, not omnibus accounts, and the audit gate of Section 17 exists precisely because this is the one place the distinction could fail in code.

**Entity and jurisdiction.** The system is operated by Fold Computer LLC, a

Colorado (USA) limited-liability company. Sanctioned-jurisdiction screening applies to marketplace distribution per venue policy; services are priced and settled in fiat-referenced stablecoins, permanently; no protocol token exists at the time of writing, and any future $fold token will be additive to — never a gate on — the fee-for-service system specified here (Section 2).

**Change governance.** Prices, tool sets, and mandate semantics change only *forward*: a challenge honored is honored at its stated terms; a mandate compiled is immutable to the platform for its lifetime (the principal, not fold, may revoke). Versioned updates of this document are the changelog of record for semantics; the onchain registry is the changelog of record for deployments.

# 20 Summary of Formal Guarantees

For reference, Table 16 consolidates every formal claim in this document, its assumptions, and — critically — its epistemic register. A guarantee is only as strong as its weakest assumption, and the table makes each assumption inspectable at a glance.

Nothing in the table asserts profitability, and one row (Prop. 4.1) wears its audit dependency in a footnote rather than in fine print. A reader who remembers only this table remembers the document correctly.

# 21 Roadmap

# 22 Conclusion

The history of computing is a history of capability waiting for an operating system. We have argued — formally where formality earns its keep, and plainly where it does not — that autonomous agents are the current instance of that pattern: the models can think; what they lack is a place to live. fold is our answer, built in the order that honesty about risk requires: payments first, because everything else buys through them; execution second; the kernel as the human's door into the machine; and custody last, behind an audit, because trust is the one component that cannot be hotfixed.

The claims we have staked are narrow enough to fail in public. Budget safety is a proof, not a promise (Proposition 4.1). Prices are printed (Table 4). The endpoint

**Table 16:** All formal claims, consolidated. Registers per Section 1.

| Claim | Statement | Key assumptions | Register |
|---|---|---|---|
| Prop. 4.1 | lifetime spend $\leq B_{\text{tot}}$, any $\pi$ | value actions require session-key signature; on-chain spend counter | SPECIFIED[†] |
| Prop. 4.2 | episodes finite under deadline | runtime refuses post-$T$ dispatch | DEPLOYED (runtime) |
| Eq. (8) | ring projection conservative | projection forbids what R1 cannot express | SPECIFIED |
| Prop. 5.1 | payer loss $\leq$ fee per call | Permit2 nonce/spender/amount/deadline semantics | DEPLOYED |
| Prop. 9.1 | no charge without validation | automaton of Table 8 | DEPLOYED (verified) |
| Prop. 9.2 | at most one settlement per lifecycle | single-nonce permits | DEPLOYED |
| Eq. (10) | no loss-making tool mix on `/exec` | stated marginal costs | DEPLOYED (prices) |
| Eq. (15), (16) | rational-purchase thresholds | symmetric signal model | ILLUSTRATIVE |
| Eq. (13) | OS lifts $\mathbb{E}[N]$ via $p$ and $h$ | independence; Poisson arrivals | ILLUSTRATIVE |
| Eq. (20) | honesty equilibrium on marketplaces | rating-based exclusion; $\delta \geq \frac{1}{2}$ | ILLUSTRATIVE |
| Eq. (22) | calibration measurable from records | sufficient record volume | SPECIFIED |

[†] The proof is unconditional given its premises; the premises reach DEPLOYED status only when the audited tref contracts reach mainnet (Section 17). Printing this footnote is the point of the table.

answers anyone's `curl` with the same 402 it answers an agent's. The parts that are not yet earned — the audit, the calibration, the long-horizon economics — are labeled as such, in a document whose registers were declared on page one.

You brief it once, in plain language: what it does, how it thinks, what it must never do. fold turns that into the mind the agent runs on, the fence it hunts inside, and the ledger you can read. Intent becomes behavior. The machine is open: `fold.computer`.

# References


[1] D. M. Ritchie and K. Thompson, "The UNIX Time-Sharing System," *Communications of the ACM*, 17(7):365–375, 1974.

[2] F. J. Corbató, M. Merwin-Daggett, and R. C. Daley, "An Experimental Time-Sharing

**Table 17:** Roadmap. Phases ship strictly in order; each leaves a working system.

| Phase | Scope | Exit criterion |
|---|---|---|
| P1 (done) | Paid service layer | 24 endpoints live on mainnet; external settlements |
| P2 (done) | Execution + marketplace | `/exec` live; independent marketplace review passed |
| P3 (now) | Kernel shell | desktop + first-party apps operable end-to-end in browser |
| P4 | Kernel apps | wallet, LLM shell, search, markets as windows; symmetric API |
| P5 | tref audit & mainnet | independent audit passed; mainnet deploys under mandates |
| P6 | Resident economy | deployed agents buying fold services under Eq. (15) at scale |

System," *Proc. AFIPS SJCC*, 1962.

[3] General Motors Research Laboratories, "GM-NAA I/O System for the IBM 704," 1956. Historical monitor; see C. Gordon Bell et al., *Computer Structures*, McGraw-Hill, 1971.

[4] R. Fielding, M. Nottingham, and J. Reschke (eds.), "HTTP Semantics," RFC 9110, IETF, June 2022. §15.5.3, status 402.

[5] Coinbase, "x402: An open protocol for internet-native payments over HTTP," specification and reference implementation, 2025. https://x402.org.

[6] P. Liu et al., "EIP-3009: Transfer With Authorization," Ethereum Improvement Proposals, 2020.

[7] Uniswap Labs, "Permit2: unified token approvals with signature transfer," contract documentation, 2022. https://github.com/Uniswap/permit2.

[8] V. Buterin, Y. Weiss, D. Tirosh, S. Nacson, A. Forshtat, K. Gazso, and T. Hess, "ERC-4337: Account Abstraction Using Alt Mempool," Ethereum Improvement Proposals, 2023.

[9] ZeroDev, "Kernel: a modular smart account with session keys," technical documentation, 2024. https://docs.zerodev.app.

[10] Anthropic, "Model Context Protocol," open specification, 2024. https://modelcontextprotocol.io.

[11] OKX, "Agent Payments Protocol and the OKX AI agent marketplace," whitepaper and developer documentation, 2026. https://okx.ai.

[12] Coinbase, "Base: an Ethereum L2 built on the OP Stack," documentation, 2023. https://base.org.

[13] R. A. Howard, "Information Value Theory," *IEEE Transactions on Systems Science and Cybernetics*, 2(1):22–26, 1966.

[14] H. Raiffa and R. Schlaifer, *Applied Statistical Decision Theory*, Harvard University Press, 1961.

[15] J. von Neumann and O. Morgenstern, *Theory of Games and Economic Behavior*, Princeton University Press, 1944.

[16] B. W. Lampson, "Protection," *ACM SIGOPS Operating Systems Review*, 8(1):18–24, 1974.

[17] J. H. Saltzer and M. D. Schroeder, "The Protection of Information in Computer Systems," *Proceedings of the IEEE*, 63(9):1278–1308, 1975.

# A Notation

**Table 18:** Symbols used throughout.

| Symbol | Introduced | Meaning |
|---|---|---|
| $\mathcal{A}$ | Def. 3.1 | agent tuple $(w, \mathcal{M}, \pi, \sigma, \mathcal{T})$ |
| $w$ | Def. 3.1 | agent wallet (identity + value) |
| $\mathcal{M} = (O, C, B, H)$ | Def. 4.1 | mandate: objective, constraints, budgets, halts |
| $B_{\text{tx}}, B_{\text{day}}, B_{\text{tot}}$ | Def. 4.1 | per-tx / rolling-24h / lifetime spend ceilings |
| $S(e_{:n})$ | Eq. (2) | cumulative fee + outbound-value spend |
| $G(e_{:n})$ | Eq. (3) | cumulative gas (sponsored; excluded from $S$) |
| $c_i, \kappa_i$ | §5, §6.2 | price / marginal cost of tool $i$ |
| $f$ | Eq. (10) | flat fee of `/exec` (0.01 USDT) |
| $p, s, r$ | Eq. (9) | payer, settlement spender, payee (identity wallet) |
| $p_i, p_{\text{convert}}$ | Eq. (12) | subsystem availability; opportunity conversion prob. |
| $\lambda, h$ | Eq. (13) | opportunity rate; fatal-lapse hazard |
| VoI | Eq. (14) | value of information of a paid signal |
| $q, \theta, V, L$ | §8.2 | signal accuracy; prior; payoff; loss |
| $\gamma_n, d_n$ | Eq. (21) | stated confidence; itemized information spend |
| $\text{Adm}_{Rk}$ | Eq. (8) | action set admitted by enforcement ring $k$ |

# B The `/exec` Wire Format

**Unpaid request** `POST /exec` ⇒ `402` with body (abridged):

```
{ "x402Version": 2, "error": "Payment required", "accepts": [ { "scheme":
"exact", "network": "base", "asset": "<USDT>", "maxAmountRequired":
"10000", "payTo": "<identity wallet>", "extra": { "spender": "<settlement
signer>" }, "nonce": "...", "maxTimeoutSeconds": ... } ] }
```

**Paid request** carries `X-PAYMENT` (base64 Permit2 authorization + witness) and a body

```
{ "tool": "search" | "rpc" | "prediction", "params": { ... } }
```

Responses: `400` (invalid tool/params; never charged), `402` (absent/failed payment), `200` (result + settlement reference). Amounts are integers in token base units (`10000` = 0.01 at 6 decimals).

# C Worked Example: One Paid Decision, End to End

An external agent holds $5 in USDT on Base and a question worth answering: *is there a liquid prediction market on this week's CPI print, and what does it price?* (1) It discovers fold on a marketplace and reads one number: 0.01 per call. (2) `POST /exec` {tool: prediction, params: {query: "CPI"}} → 402 challenge. (3) It signs the Permit2 authorization — 0.01 USDT, nonce, deadline, spender bound — and retries. (4) Validation passes; settlement lands onchain; the response carries live markets with outcome prices and volumes. (5) Its decision record logs $d_n = \$0.01$, the settlement hash, and the market snapshot it acted on. Total human involvement: zero. Total accounts created: zero. Total spend: one cent, exactly, once. By Eq. (16), the purchase was rational if the decision it informed was worth more than about a dime. That asymmetry — cent-scale information against dollar-scale decisions — is the entire economic argument of this document, compressed into one round-trip.

# D Full Endpoint Catalog

Catalog membership and prices are versioned with this document; the live `openapi.json` at the storefront is authoritative between versions.

# E Mandate Schema (normalized form)

```
{ "objective": { "metric": string, "horizon": ISO8601 },
"constraints": [ { "type": "allowlist"|"threshold"|"predicate", "field":
string, "op": string, "value": any } ],
"budgets": { "perTx": uint, "per24h": uint, "lifetime": uint, "unit":
asset },
"halts": { "deadline": ISO8601, "drawdownPct": number, "killSwitch":
address },
"transparency": { "minConfidence": number, "recordLevel": "all"|"valueBearing"
} }
```

Projection to R1 (session-key policy) consumes `budgets`, `halts.deadline`, and every

**Table 19:** The deployed service surface in full (Base mainnet). Prices per call; *upto* rows state ceilings.

| Endpoint | Family | Scheme | Price (USD) |
|---|---|---|---|
| `/signal` | trading intel | upto | $\leq 0.02$ |
| `/scan` | trading intel | upto | $\leq 0.02$ |
| `/setups` | trading intel | upto | $\leq 0.02$ |
| `/divergence` | trading intel | upto | $\leq 0.02$ |
| `/funding-scan` | trading intel | upto | $\leq 0.02$ |
| `/indicators` | trading intel | exact | 0.005 |
| `/levels` | trading intel | exact | 0.005 |
| `/snapshot` | trading intel | exact | 0.005 |
| `/llm` (+9 model routes) | inference | upto | 0.0005 – 2.00 |
| `/search` | search | exact | 0.007 |
| `/extract` | extraction | exact | 0.005 |
| `/read` | extraction | exact | 0.002 |
| `/structure` | extraction | exact | 0.005 |
| `/rpc` | chain access | exact | 0.001 |
| `/prediction` | markets | exact | 0.002 |
| `/token-price` | market data | exact | 0.001 |
| `/token-security` | market data | exact | 0.003 |
| `/token-details` | market data | exact | 0.002 |
| `/trending` | market data | exact | 0.002 |
| `/holders` | market data | exact | 0.003 |
| `/pools` | market data | exact | 0.003 |
| `/stablecoin-watch` | market data | exact | 0.002 |
| `/yield` | market data | exact | 0.003 |
| `/protocol-revenue` | market data | exact | 0.003 |
| `/exec` | execution (A2MCP) | exact, self-settle | 0.010 flat (USDT) |

constraint of type `allowlist` whose field is a contract target; all other constraints bind at R2 (Eq. (8)).

# F Decision Record Schema

```
{ "n": uint, "ts": ISO8601, "action": { "kind": "think"|"call"|"tx"|"halt",
... },
"rationale": string, "confidence": number,
"informationSpend": [ { "endpoint": string, "cost": uint, "settlement":
txhash } ],
"cumulativeSpend": uint, "refs": [uri] }
```

# G Glossary

**Agent**

A policy with a wallet, a mandate, and tools; a process of the fold kernel (Def. 3.1).

**A2MCP**

Agent-to-MCP service pattern: a marketplace-discoverable service an agent pays per call, as standardized by agent marketplaces (Section 2).

**Exact scheme**

The x402 settlement mode that charges the challenged amount, exactly, once (Section 5).

**Facilitator**

A third party that verifies and settles x402 payments on a server's behalf; contrasted with self-settlement.

**Identity wallet**

The custodied payee address $r$ of Eq. (9); receives fees, exposes no key to serving infrastructure.

**Mandate**

The compiled envelope $(O, C, B, H)$ of permitted behavior (Def. 4.1); *the mandate is the contract.*

**Paymaster**

An ERC-4337 contract sponsoring gas for user operations (Section 7).

**Permit2**

Uniswap's signature-transfer standard; the self-settlement substrate for assets without EIP-3009 (Section 5.3).

**Ring (R1–R3)**

The three enforcement layers of the mandate: chain, runtime, prompt (Table 2).

**Session key**

A scoped signing key whose validity policy encodes the enforceable projection of a mandate (Section 2.3).

**Settlement signer**

The operational spender $s$ of Eq. (9); redeems permits, cannot redirect value.

**tref**

The fold deployment pipeline: brief → smart account → session key → running agent (Alg. 2).

**Upto scheme**

The x402 mode that authorizes a ceiling and settles actual metered usage at or below it.

**Validate-before-settle**

The invariant that no malformed request is ever charged (Prop. 9.1).

**x402**

The HTTP-402 payment protocol; fold's payment driver (Section 2.2).

# H Frequently Anticipated Questions

**Why an OS and not a framework?**

Frameworks live in the agent's process and inherit its trust; an OS lives beneath it and survives its compromise (Table 2, Prop. 4.1). The difference is not ergonomic but adversarial.

**Is there a token?**

Not today. Fee-for-service in stablecoins is the permanent pricing model, and no token will ever be required to use fold services (Sections 2, 19). A $fold token is under consideration for the future; if and when it launches, its design will be published in a versioned update of this document *before* launch, under the same deployed/specified/illustrative discipline as every other claim here.

**Why Base?**

Cheap, fast finality for cent-scale settlement (Section 7); institutional-grade tooling for the payment schemes used. The chain is a driver, not an identity: the RPC surface already reads five chains.

**What happens if the model providers change terms?**

Inference is one priced tool among many (Table 4); the metered `upto` scheme passes provider economics through per call, so provider drift reprices rather than breaks the system.

**Can an agent lose money?**

Inside its fence, yes — and the document says so in four places rather than one footnote (Remark after Prop. 4.2; T6 in Table 12; Sections 17, 22). What it cannot do is exceed the fence.

**When does tref reach mainnet?**

After — and only after — independent audit (Sections 14, 17). No date is stated because dates are not the gate; the audit is.

# I Integration Quickstart for Buyer Agents

The fastest correctness check of everything claimed in this document is to run it. The sequence below takes an external agent (or a human with `curl`) from zero to a paid, verified result; it exercises Props. 9.1, 5.1, 9.2 on the live system.

1. **Read the challenge (free).** `POST https://x402.fold.computer/exec` with any body and no payment header. Expect `402` and the challenge of Appendix B. Nothing has been charged; nothing will be until a signed payment is presented.

2. **Probe validation (free).** Retry with a payment header *and* an unknown tool, e.g. `{"tool": "foobar"}`. Expect `400` listing `["search","rpc","prediction"]` — and verify onchain that no settlement occurred. This is Prop. 9.1, observable by anyone.

3. **Fund the buyer.** The paying wallet needs $\geq 0.01$ USDT on Base plus a one-time Permit2 approval (`USDT.approve(Permit2, ...)`) — the single unavoidable setup step for assets without EIP-3009, noted here rather than hidden.

4. **Pay and consume.** Sign the Permit2 authorization for exactly the challenged amount and retry with a valid tool, e.g. `{"tool": "rpc", "params": {"chain": "base", "method": "eth_getBlockByNumber", "params": ["latest", false]}}`.

Expect `200`, a live block header, and a settlement reference whose amount matches the challenge to the base unit.

5. **Audit yourself.** Compare the settlement hash against the challenge nonce and amount; attempt a replay of the same permit and observe its rejection (Prop. 9.2).

Marketplace-mediated integration (OKX A2MCP) wraps steps 1–4 behind the venue's payment module; the wire semantics are identical. Total integration surface: one URL, one JSON shape, one signature scheme — the size of this appendix is itself the argument.

# J Verification and Security Checklist

The checklist below is the operational distillation of Sections 4, 9, and 14; items marked □ are gated on the audit or on measurement windows, items marked ■ are verified on the live system.

■ **402 challenge integrity**

Unpaid requests to paid routes return a well-formed challenge (asset, amount, payTo, spender, nonce, expiry) and cause no side effects.

■ **Validate-before-settle**

Malformed paid requests return `400` with no settlement (Prop. 9.1), observed on the live `/exec` route.

■ **Exactness**

Settled amount equals challenged amount; one settlement per lifecycle (Prop. 9.2).

■ **Role separation**

Payee (identity wallet) and settlement signer are distinct; signer compromise cannot redirect fees (Eq. (9)).

■ **Read-only RPC allowlist**

The `rpc` tool rejects state-changing methods.

□ **Session-key budget counters**

Onchain cumulative-spend enforcement for Prop. 4.1 — implemented on testnet; production status gated on audit.

☐ **Adversarial-load settlement**

Self-settlement behavior under hostile concurrency and reorg conditions — analyzed (Table 12), not yet battle-tested.

☐ **Confidence calibration**

Reliability term of Eq. (22) within tolerance across bins — pending measurement window.

☐ **Paymaster exposure caps**

Per-mandate sponsorship ceilings enforced and monitored under Eq. (11) — specified; production telemetry maturing.

# K Version History

| Version | Date | Scope |
|---|---|---|
| 1.0 | July 2026 | Initial publication: formal model, payment/execution layers as deployed, economics, |
| 1.0.1 | July 2026 | Token posture clarified: no protocol token exists today; stablecoin fee-for-service is p |