

gleem

An Operating System for Autonomous Onchain Agents

A Technical Manifesto

Gleem Computer LLC

gleem.computer · dev@gleem.computer

Robinhood Chain

September 7, 2026

Version 1.0

*“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.”*

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 **gleem**, a browser-native operating system for autonomous onchain agents. *gleem* 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 *gleem* 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 EIP-3009 authorization, sponsored gasless execution, and the *gleem engine* deployment pipeline over ERC-4337 smart accounts with session keys. *gleem* is built on **Robinhood Chain**, a permissionless, EVM-compatible Layer 2 whose native gas token is ETH and whose settlement stablecoin is USDG. We report the current state of the system honestly — distinguishing what is live, what is under active construction, and what remains unproven — and we state precisely what a reader may and may not rely on today. 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, Robinhood Chain, onchain execution.

Contents

1	Introduction	4
1.1	The thesis: agents fail at living	4
1.2	What gleem is	5
1.3	What is live, what is being built	6
1.4	Design principles	6
2	Background and Related Systems	7
2.1	Language models as reasoning engines	7
2.2	Machine-native payments and HTTP 402	7
2.3	Account abstraction and session keys	8
2.4	Agent interoperability protocols	8
2.5	What gleem is not	8
3	Why Robinhood Chain	9
4	The Agent as a Process	9
5	The Mandate: Intent Becomes Behavior	10
5.1	Where enforcement lives	11
6	The Payment Layer: x402 on Robinhood Chain	12
6.1	The protocol exchange	12
6.2	State: what is live, what is under construction	13
7	The Execution Endpoint: Anatomy of a Paid Service	13
8	Gasless Execution	14
9	Economics of Autonomous Agents	14
9.1	The three deaths, formally	14
9.2	Value of information: when should an agent pay for data?	15
9.3	Two-sided structure	15
10	Protocol Formalization: the 402 Exchange as a State Machine	15
11	Marketplace Dynamics	16

12 Comparative Analysis	17
13 The Kernel: A Computer You Walk Into	17
13.1 Architecture	17
13.2 The application surface	18
13.3 The shell as the human’s mandate	18
14 Deployment and Threat Model	18
14.1 Pipeline	18
14.2 Threat model	18
15 Transparency: Reading a Mind	19
15.1 Calibrating stated confidence	19
16 Evaluation: The State of the Machine	20
17 Case Studies	20
17.1 Case A: a research agent	21
17.2 Case B: a fenced trading agent	21
17.3 Case C: a monitoring agent with escalation	21
18 Custody, Compliance, and Governance Posture	21
19 Summary of Formal Guarantees	22
20 Limitations and Open Problems	23
21 Roadmap	23
22 Conclusion	24

1 Introduction

A language model, given a prompt, can produce a plan that would impress a human analyst. It can weigh evidence, propose actions, and explain its reasoning. And then the conversation ends, the context window closes, and nothing it decided persists into the world. The model did not fail to think. It failed to *live*: to hold resources, to act without a human re-prompting it at every step, to pay for what it needs, and to be constrained by rules it cannot talk its way around.

This is not a limitation of intelligence. It is a limitation of infrastructure. The same gap once separated raw hardware — capable, but useless without a human toggling switches — from useful computation. What closed that gap was not a faster processor. It was the operating system: a layer that gave programs processes, memory, a filesystem, and system calls, so that software could run without a human operating the machine by hand.

Agents are in the pre-operating-system era. Each one is a script a human babysits, a chatbot bolted to an API key, a demo that works once and cannot be trusted to run unattended. We argue that the missing layer for agents is the same layer that was missing for early programs: an operating system that treats an agent as a first-class process with its own identity, its own resources, its own means of payment, and its own enforced constraints.

This manifesto presents **gleem**, our attempt at that operating system, built on Robinhood Chain.

1.1 The thesis: agents fail at living

The thesis is narrow and falsifiable. An agent economy does not emerge from larger models alone. It emerges when agents can do four things they mostly cannot do today:

1. **Pay for what they need.** An agent that must ask a human to enter a credit card, or that carries a human's API key in a config file, is not autonomous. It is a puppet with a good vocabulary. Autonomy requires that the agent settle its own micropayments, per call, without a human in the loop.
2. **Hold and use funds under limits.** An agent that cannot touch money cannot act in a financial world. But an agent that holds unlimited funds is a liability.

Autonomy requires custody the agent controls *within bounds it cannot exceed*.

3. **Persist as a process.** An agent that exists only for the length of a conversation is a feature, not a worker. Autonomy requires a runtime that keeps the agent alive, running its mandate, while the human does something else.
4. **Be legible and constrained.** An autonomous agent that no one can read, and that nothing can stop, is not a product. It is a hazard. Autonomy requires that every decision be recorded and that limits be enforced below the level of the prompt, where the model cannot argue with them.

Each of these is an infrastructure problem, not an intelligence problem. A larger model does not solve any of them. An operating system does.

1.2 What gleem is

gleem is a browser-native operating system for autonomous onchain agents. Concretely:

- It is an **operating system**, not an application. It has a desktop, windows, a dock, and a set of applications, and every one of them is wired to the chain. The metaphor is literal: gleem treats agents as processes with resources, identity, permissions, and system calls.
- It runs on **Robinhood Chain**, a permissionless EVM Layer 2 built on the Arbitrum Orbit stack, whose native gas token is ETH and whose settlement stablecoin is USDG (Paxos). We chose it deliberately; Section 3 states why.
- An agent on gleem is **deployed, not prompted**. A human writes a mandate — a strategy, a budget, a set of constraints — and the agent runs it, reasoning in language, acting onchain, and paying its own way, while the human does something else.
- gleem is **non-custodial**. The agent's funds live in an account the user controls. The agent acts through a session key, scoped and capped on-chain. It can act; it cannot take. This is the core principle, and it is enforced by the chain, not promised by us.

1.3 What is live, what is being built

We hold ourselves to a standard this document will repeat: we distinguish, everywhere, between what runs today, what is under active construction, and what remains unproven. A manifesto that blurs those lines is marketing, and this is not marketing.

Table 1: Honest state of the system at the time of writing.

Capability	State	Note
Browser-native OS shell	<i>live</i>	desktop, apps, windows, dock
Robinhood Chain wallet connection	<i>live</i>	chain 4663, ETH gas, non-custodial
Live chain reads (gas, balances)	<i>live</i>	read directly from Robinhood RPC
Portfolio (ETH, USDG)	<i>live</i>	read from chain, not estimated
x402 payment layer	<i>building</i>	USDG implements EIP-3009; facilitator under construction
Non-custodial swap execution	<i>building</i>	routed via aggregation; see Sec. 7
Agent deployment & runtime	<i>building</i>	session-key model specified; not yet live on Robinhood
Registry contract (on-chain)	<i>unproven</i>	to be deployed and audited on Robinhood Chain

The reader should treat the “building” and “unproven” rows as design commitments with specified mechanisms, not as claims of current function. Where this document formalizes a guarantee, it states whether that guarantee is enforced today or is a property of the target architecture. This is the same discipline that a serious systems paper applies to itself, and it is the discipline an agent’s owner is entitled to.

1.4 Design principles

Five principles run through every layer of gleem.

1. **Custody stays with the human.** No design decision may require the agent, or gleem, to hold the user’s funds. Where an agent acts, it acts through a scoped, capped authority the chain enforces.
2. **Enforcement lives below the prompt.** A constraint an agent can talk its way around is not a constraint. Budgets, limits, and permissions are enforced at

the account and protocol layer, where the model has no voice.

3. **Every action is legible.** An autonomous system that cannot be read cannot be trusted. Every decision, every payment, every trade leaves a record.
4. **Payment is machine-native.** Agents pay each other and pay for services the way software should: per call, in a stablecoin, over an open protocol, with no accounts and no keys in a config file.
5. **Honesty about state.** We say what works, what does not yet, and what is unproven. A system asking to manage money earns trust by being precise about its own limits.

2 Background and Related Systems

gleem sits at the intersection of several mature lines of work: large language models as reasoning engines, account abstraction as a custody mechanism, machine-native payment protocols, and agent interoperability standards. This section places gleem among them and, as importantly, states what gleem is *not*.

2.1 Language models as reasoning engines

The capability that makes agents plausible is recent: models that can decompose a goal, weigh alternatives, and produce structured actions rather than prose. gleem takes this capability as given. It is not a contribution of this work, and gleem is deliberately agnostic to which model provides it. The manifesto’s claim is orthogonal: that reasoning, however good, is inert without the infrastructure to act on it.

2.2 Machine-native payments and HTTP 402

The HTTP status code 402, “Payment Required,” was reserved at the birth of the web and left unused for decades. The x402 protocol revives it as the basis for machine-native payment: a server responds to an unpaid request with 402 and a description of what payment it will accept; the client signs an authorization to transfer a stablecoin; the server verifies and settles it, then returns the result. No account, no subscription, no API key. Section 6 specifies how gleem uses x402 over EIP-3009 authorization on Robinhood Chain, and states honestly which parts are live and which are under construction.

2.3 Account abstraction and session keys

An agent needs to act on-chain without holding the user's root key. Account abstraction (ERC-4337) makes an account a smart contract rather than an externally-owned key, which allows a *session key*: a subordinate signer with narrow, on-chain-enforced permissions — spend at most this much, only on these actions, only until this time. This is the mechanism by which gleem gives an agent the power to act and denies it the power to take. The permissions are enforced by the account contract, not by the agent's good behavior.

2.4 Agent interoperability protocols

A growing set of standards lets agents discover and call one another's tools — the Model Context Protocol (MCP) among them. gleem exposes its own capabilities as an MCP surface, so that an external agent, including one running in a general assistant, can read markets and prepare transactions on Robinhood Chain that the user signs. gleem is thus both a place agents live and a service other agents consume.

2.5 What gleem is not

Precision about scope is a courtesy to the reader.

- gleem is **not a language model**. It is the operating system a model acts through. It is agnostic to the model.
- gleem is **not a custodial wallet or exchange**. It never takes possession of user funds. Every movement of value is authorized by a user signature or by a session key the user granted and the chain caps.
- gleem is **not a token launchpad**. Launching a token is a different function with different machinery; gleem deploys and runs agents, it does not mint assets.
- gleem is **not a backtesting toy**. Where the system reads the chain, it reads the live chain. Where it cannot yet act, it says so rather than simulating action and presenting it as real.

3 Why Robinhood Chain

A choice of chain is a choice of physics. gleem runs on Robinhood Chain, and the choice was deliberate rather than incidental.

Robinhood Chain is a permissionless, EVM-compatible Layer 2 built on the Arbitrum Orbit stack, settling to Ethereum, with a native gas token of ETH and a chain ID of 4663. Three properties made it the right ground for an agent operating system.

1. **It is designed to be agent-native.** Robinhood Chain was launched with autonomous software agents named as a first-class class of user, not an afterthought. An operating system for agents belongs on a chain whose own thesis is that agents are participants.
2. **It carries real financial rails and real distribution.** The chain's flagship instruments are tokenized real-world assets — equities that trade continuously and compose as ERC-20 tokens — reachable through a wallet with a large, established user base. If agents are to manage value, they should live where value already flows.
3. **It is ordinary where it should be ordinary.** It is a standard EVM surface: Solidity contracts deploy unchanged, and `viem`, Foundry, and MetaMask work without modification. Novelty in the execution layer is a cost, not a feature; gleem spends its novelty budget on the agent layer, not on fighting the chain.

The settlement stablecoin on Robinhood Chain is USDG, issued by Paxos. USDG implements EIP-3009, the “transfer with authorization” standard, which — as Section 6 shows — is precisely the primitive that machine-native payment requires. This is not a lucky coincidence so much as a sign that the chain and gleem are aimed at the same future.

4 The Agent as a Process

The central abstraction of gleem is stated in the name: it is an operating system, and its unit of execution is a *process*. In a conventional operating system, a process is a running program with an identity, an address space, a set of open resources, and a set of permissions granted by the kernel. A gleem agent is the same idea, lifted onchain.

Definition 1 (Agent process). *A gleem agent is a tuple $A = (\iota, \mu, W, K, S)$ where ι is a stable identity, μ is a mandate (Section 5), W is an account holding the agent’s working funds, K is a session key granting scoped authority over W , and S is the persistent state — decisions, positions, and records — that the process accumulates as it runs.*

Four things distinguish a process from a prompt, and each maps to a component of A .

Identity (ι). A prompt is anonymous; it exists only in the moment of asking. A process has a name the system knows it by, under which its history accrues. A gleem agent’s identity persists across its whole life, and every action it takes is attributable to it.

Resources (W). A prompt borrows the human’s context. A process holds its own. A gleem agent has an account with working funds — not the human’s root wallet, but a bounded balance the agent operates within.

Authority (K). A prompt has whatever power the human’s credentials carry, all of it, indiscriminately. A process has exactly the permissions the kernel granted, no more. A gleem agent acts through a session key whose authority is scoped and capped on-chain: a maximum spend, a set of permitted actions, an expiry. The agent may act; it may not exceed.

State (S). A prompt forgets. A process remembers. A gleem agent accumulates a durable record: what it decided, why, what it did, and the onchain proof of each action. This record is not a courtesy log; it is the substance of the agent’s legibility (Section 15).

The consequence of this abstraction is that “deploying an agent” is not “sending a long prompt.” It is instantiating a process: giving it an identity, funding its account, granting it a scoped key, and setting it running under a mandate. The human writes the rules once. The process runs them.

5 The Mandate: Intent Becomes Behavior

An agent is only as safe as the constraints it cannot escape. The mandate is how a human’s intent becomes an agent’s enforced behavior.

Informally, a mandate is a plain-language brief: *what* the agent should pursue, *how*

much it may spend, and *what* it must never do. Formally, we model it as a constrained optimization envelope.

Definition 2 (Mandate). *A mandate is a tuple $\mu = (o, \mathcal{C}, b, \tau)$ where o is an objective the agent seeks to advance, $\mathcal{C}$ is a set of constraints (permitted actions, position limits, forbidden assets), b is a budget — a hard cap on cumulative spend — and τ is a set of stop conditions under which the process halts.*

The agent chooses actions $a_1, a_2, \dots$ to advance o , but every action is admissible only if it satisfies $\mathcal{C}$ and leaves the cumulative spend below b . Write $c(a_i)$ for the cost of action a_i . Then the budget constraint is simply

$$\sum_{i=1}^n c(a_i) \leq b \quad \text{for all } n,$$

and this is not a request to the agent; it is a property enforced below the agent, at the account layer.

5.1 Where enforcement lives

This is the crux, and it is the design decision that separates gleem from a well-prompted chatbot. There are two places one might try to enforce a mandate, and only one of them works.

Enforcement in the prompt (does not work). One can instruct a model: “never spend more than \$100; never touch this asset.” The model will usually comply. But “usually” is not a guarantee, and a system that manages money cannot run on “usually.” A sufficiently clever prompt, an adversarial input, or a simple model error can talk past a prompt-level rule. A constraint the agent can reason about is a constraint the agent can violate.

Enforcement below the prompt (works). gleem enforces the mandate where the model has no voice: at the account and protocol layer. The budget b is a cap on the session key, checked on-chain at settlement. The permitted actions $\mathcal{C}$ are the only actions the session key is authorized to sign. The agent cannot spend past b because the chain will reject the transaction, not because the agent chose not to.

Invariant 1 (Budget safety). *For any execution of an agent under mandate $\mu = (o, \mathcal{C}, b, \tau)$, the cumulative settled spend never exceeds b , regardless of the agent’s reasoning, inputs, or errors.*

Invariant 1 is a property of the enforcement layer, not of the agent's good behavior. Its strength is exactly that it does not depend on the model. In the target architecture on Robinhood Chain, the session-key cap is enforced by the account contract; where that contract is not yet deployed, the invariant is a design commitment, and this document marks it as such (Table 1).

The mandate is thus the human's true interface to the agent. Not a prompt the agent may reinterpret, but an envelope it operates inside and cannot leave.

6 The Payment Layer: x402 on Robinhood Chain

An autonomous agent must pay for what it consumes — data, signals, computation — without a human entering a card or embedding an API key. This is the problem HTTP 402 was reserved to solve, and the x402 protocol solves it.

6.1 The protocol exchange

The exchange is small by design. Smallness is the property being claimed: a payment protocol an agent can implement in one function.

1. **Unpaid request.** The agent sends a request to a paid endpoint with no payment. The server replies **402 Payment Required** and a description of what it will accept: the asset, the amount, the recipient, and a nonce.
2. **Signed authorization.** The agent signs an EIP-3009 `transferWithAuthorization` message — an off-chain signature authorizing a one-time transfer of the named stablecoin, for the named amount, to the named recipient. No transaction is broadcast by the agent; no gas is spent by the agent.
3. **Verify and settle.** The server (or a facilitator acting for it) verifies the signature and the agent's balance, then submits the authorization on-chain, paying gas itself and netting it out of the payment. The stablecoin moves.
4. **Result.** With settlement confirmed, the server returns the result the agent paid for.

On Robinhood Chain, the named stablecoin is USDG, which implements EIP-3009 — so step 2 is a signature over a message the USDG contract will honor, and step 3 is a call the USDG contract will settle. The primitive exists; the protocol is a thin

choreography over it.

Proposition 1 (No charge without result, no result without charge). *Under the x402 exchange, a well-behaved server settles a payment only when it is about to return the corresponding result, and returns a paid result only against a settled payment. An agent that receives a 402 and stops has paid nothing.*

6.2 State: what is live, what is under construction

The honest picture (Table 1) is this. The stablecoin primitive is live: USDG on Robinhood Chain implements `transferWithAuthorization`, and its authorization signatures verify. The *facilitator* — the service that verifies and settles on a resource server’s behalf — is under construction for Robinhood Chain, because the hosted facilitators that serve Base do not serve chain 4663. gleem therefore operates its own facilitator, and this document does not claim the settlement path is live until that facilitator is in production. What is specified here is the mechanism; what is marked “building” is the deployment.

7 The Execution Endpoint: Anatomy of a Paid Service

A payment protocol is only useful if there is something to buy. The smallest complete instance of a gleem service is a single paid endpoint: a resource an external agent can discover, price, pay for, and consume in one loop.

Such an endpoint does three things and no more. It advertises its price by answering an unpaid request with a 402 and a challenge. It accepts a signed authorization, verifies it, and settles it. And it returns a result whose value exceeds its price for the buyer’s purpose. The discipline of a flat, legible fee — rather than opaque metering — is deliberate: an agent deciding whether to buy must be able to compute the trade in advance, and a price it can read is a price it can reason about.

The service surface of gleem is, in the target architecture, a set of such endpoints: market-structure reads, signals, and computation, each priced per call, each settled in USDG over x402. gleem runs both sides of this rail — its agents pay over x402, and its data services sell over it — which is a claim about symmetry, not merely capability: an economy needs buyers and sellers, and gleem is built to be both.

8 Gasless Execution

Gas is a quiet death for autonomous agents. An agent that must hold the chain’s gas token, monitor its balance, and top it up is an agent with a human-shaped dependency at the center of its liveness. It stops not because it was wrong but because it ran out of gas.

gleem’s design moves gas off the agent’s liveness path. When an agent acts, the execution layer sponsors the gas: the agent signs its intent, and the platform’s execution account pays the chain. On Robinhood Chain, where gas is denominated in ETH and base fees are low, this is inexpensive to sponsor, and it removes an entire class of failure from the agent’s concern. The agent reasons about its mandate and its budget; it does not reason about gas.

This is, again, an architectural commitment with a stated construction status. The mechanism — sponsored execution over account abstraction — is specified; its production deployment on Robinhood Chain is in progress.

9 Economics of Autonomous Agents

An agent that cannot pay its own way is not autonomous; an agent that pays more than it earns is not viable. This section derives the conditions under which an autonomous agent is financially sustainable.

9.1 The three deaths, formally

An autonomous agent can die in exactly three ways, and gleem’s architecture is organized around preventing each.

1. **Gas death.** The agent runs out of the chain’s gas token and can no longer transact. Prevented by gasless execution (Section 8): gas is sponsored, off the agent’s path.
2. **Credential death.** The agent’s access depends on a human-held key, subscription, or login that expires or is revoked. Prevented by machine-native payment (Section 6): the agent pays per call, holding no credential.
3. **Mandate death.** The agent acts outside its limits and is halted, or should be. *Chosen* rather than prevented: the mandate’s stop conditions τ are the intended

death, and Invariant 1 guarantees it is bounded.

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

An agent deciding whether to buy a piece of data faces a clean decision-theoretic question. Let a decision have value V if made well and $V - L$ if made poorly, and let a datum cost c and raise the probability of a good decision from p to p' . The datum is worth buying when its expected lift exceeds its price:

$$(p' - p)L > c. \tag{1}$$

The structural fact that makes an agent economy possible is the *asymmetry of scale*: information priced at cents can inform decisions valued in dollars. When c is a cent and L is a dollar, Eq. (1) is satisfied by a probability lift of barely one percent. This is the entire economic argument for machine-native micropayments, compressed into one inequality: cent-scale information against dollar-scale decisions.

9.3 Two-sided structure

Because gleem is both a buyer (its agents pay for data) and a seller (its services charge for reads), its unit economics have two sides. As a buyer, it is governed by Eq. (1): it should pay for a datum exactly when the datum's lift exceeds its cost. As a seller, it should price a service below the value of the information it conveys to the median buyer, and above its marginal cost to serve. The gap between those two bounds is the room in which an agent economy operates.

10 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 resource server's perspective as a small automaton. Its smallness is the point: a payment exchange an agent can implement without a library.

The transitions are constrained so that no path reaches SERVED without passing SETTLING, and no path charges a buyer that ends in REJECTED. This is the structural content of Proposition 1: the automaton has no edge from an unpaid state to a

Table 2: States of a single paid request.

State	Terminal?	Meaning
AWAIT	no	request received, no valid payment yet
CHALLENGED	no	402 returned; server awaits authorization
VERIFYING	no	authorization received; signature and balance being checked
SETTLING	no	authorization valid; transfer being submitted on-chain
SERVED	yes	settlement confirmed; result returned
REJECTED	yes	invalid request, signature, or insufficient funds; nothing charged

served result, and no edge that charges on a rejection. A protocol whose safety is a property of its transition graph, rather than of its implementers' care, is a protocol an agent can trust.

11 Marketplace Dynamics

If agents can pay one another, a market forms among them. Three forces shape it.

Discovery and the legibility premium. An agent choosing among services cannot read marketing; it reads interfaces. A service whose price, inputs, and outputs are legible to a machine will be chosen over an equivalent service that is not, even at a higher price. Legibility is thus a competitive advantage, and gleem's flat, readable pricing is a bet on that premium.

Reputation as a repeated game. A one-shot interaction rewards defection; a repeated one rewards reliability. Because agents transact repeatedly and keep records, a service's history of honest, correct responses becomes an asset, and cheating becomes expensive. The onchain record (Section 15) is what makes reputation legible across agents rather than trapped inside one.

Price competition and floors. Competition drives prices toward marginal cost, but marginal cost for a paid data service is not zero: it includes the cost to produce the datum and the gas to settle the payment. The settlement floor — the minimum economical payment — is set by that gas cost, which is why a low-gas chain like Robinhood Chain matters for micropayments: it lowers the floor beneath which a market cannot go.

12 Comparative Analysis

gleem is not the only system that lets software touch a chain. It is useful to state precisely how it differs from the nearest neighbors, because the differences are the reason it exists.

Versus a wallet with an API. A programmable wallet lets code sign transactions. But it has no notion of a mandate, no enforced budget below the caller, no per-call payment for its own inputs, and no runtime that keeps a process alive. It is a hand; gleem is a body with a hand.

Versus a trade-by-message bot. A bot that executes an instruction (“buy \$100 of X”) is an executor: the human decides, the bot clicks. It does not reason under a mandate, does not choose to act or refuse, and does not pay its own way. gleem’s agents are actors, not executors: given a mandate, they decide, and a well-formed agent can say no.

Versus a payment protocol alone. x402 moves money; it does not trade, reason, or persist. A trade requires a signed order, slippage protection, routing, a scoped key, and an onchain cap — none of which a payment protocol provides. gleem uses x402 as one layer; it is not reducible to it.

The through-line: each neighbor supplies one faculty. gleem’s claim is that autonomy requires all of them together — identity, resources, payment, execution, and enforced constraint — and that assembling them is the operating system.

13 The Kernel: A Computer You Walk Into

The abstractions above — process, mandate, payment, execution — need a place to live that a human can see and operate. That place is the gleem shell: a browser-native desktop that is, deliberately, a computer you walk into.

13.1 Architecture

The shell runs in the browser and presents the familiar surface of an operating system: a desktop, windows, a dock, and a set of applications. Beneath the surface, each application is a view onto a real subsystem — the wallet onto the account, the markets app onto live chain reads, the console onto the agents’ decision records. Nothing on the desktop is a mockup; each pane is wired to the chain or to a live

service. Where a subsystem is not yet live on Robinhood Chain, the corresponding surface says so rather than presenting a simulation.

13.2 The application surface

The applications are the system calls a human makes. A wallet reads balances from the chain. A markets view reads prices from the chain. A deploy flow instantiates an agent process under a mandate. A monitor watches the running processes. A console reads their records. A terminal offers those who prefer it a direct line to the machine. And a language interface sits at the center: the human speaks, and the request becomes an action across the other applications, mediated by gleem's own tool surface. This is the operating system's shell in the original sense — the layer through which a human commands the machine.

13.3 The shell as the human's mandate

There is a symmetry worth naming. The mandate is the human's interface to an agent: written once, enforced below the prompt. The shell is the human's interface to the system: a place to write mandates, fund processes, and read what they did. The agent operates inside a mandate it cannot leave; the human operates the system through a shell that shows everything the agents do. Constraint below, legibility above.

14 Deployment and Threat Model

14.1 Pipeline

Deploying an agent is instantiating a process, and the pipeline reflects that: a human writes a brief, gleem compiles it into a mandate, an account is provisioned for the agent's working funds, a session key is granted with the mandate's caps encoded, and the process begins running. Each stage has an onchain artifact where one is required — the account, the session key's permissions, the agent's registration — so that the deployment itself is legible after the fact.

14.2 Threat model

An honest system states what it defends against and what it does not.

- **A compromised agent (defended).** If the model is manipulated into trying to overspend or to touch a forbidden asset, the session-key cap and permitted-action set reject the attempt at the chain. The compromise cannot exceed the mandate.
- **A compromised session key (bounded).** If a session key leaks, the damage is bounded by its cap and expiry, and it can be revoked. It is not the root key; it never was.
- **A compromised human account (not defended by gleem).** If the user's own root wallet is compromised, that is outside gleem's boundary, as it is outside any non-custodial system's boundary. gleem reduces the blast radius of agent and key compromise; it does not replace the user's own key hygiene.
- **A dishonest service (mitigated).** A paid service that returns garbage cannot be prevented from doing so once, but the repeated-game structure (Section 11) and the onchain record make it expensive to do so twice.

15 Transparency: Reading a Mind

Autonomy without legibility is a liability. An agent no one can read, and nothing can stop, is not a product; it is a hazard. gleem's answer is that every agent emits a decision record, so that a human can read the machine's work.

A decision record is not a log line. It is a structured account of what the agent decided, the reasoning it gave, the action it took, and the onchain proof of that action. Because the actions settle on-chain, the record is anchored to something the agent cannot edit after the fact: the transaction happened, and its hash is in the record. The claim "if it happened, it is recorded" is only as strong as the anchoring, and onchain settlement is the anchor.

15.1 Calibrating stated confidence

An agent that reports its own confidence is useful only if that confidence is calibrated: when it says seventy percent, it should be right about seventy percent of the time. gleem's records make calibration measurable, because each decision carries a stated confidence and each outcome is observable. Over many decisions, a systematic gap between stated confidence and realized accuracy is itself legible, and a miscalibrated

agent can be identified and retired. Legibility is not only about reading one decision; it is about auditing a mind over time.

16 Evaluation: The State of the Machine

A manifesto that evaluates nothing is a brochure. We state the system's current, honest state against the claims above.

What is live and verifiable today: the browser-native shell; wallet connection to Robinhood Chain (chain 4663, non-custodial); live chain reads for gas and balances, sourced directly from the Robinhood RPC rather than estimated; and a portfolio view reading ETH and USDG balances from the chain. These are not simulations; they are reads of the live network, and a reader can confirm them by connecting a wallet.

What is under active construction: the x402 facilitator for Robinhood Chain, on which the payment path depends; non-custodial swap execution; and the agent deployment and runtime on Robinhood Chain. For each, the mechanism is specified in this document, and the deployment is in progress.

What is unproven and marked as such: the registry and session-key contracts, which must be deployed and audited on Robinhood Chain before Invariant 1 is enforced there rather than committed to. We will not describe those guarantees as live until the contracts that enforce them are.

This is the discipline promised in Section 1.3, applied to ourselves. A system asking to manage money earns trust by being precise about its own limits, and the precision above is the price of that trust.

17 Case Studies

Abstractions are clearest in instances. The three agents below illustrate how a mandate becomes behavior. They are presented as designs realizable on the target architecture; where a capability they use is not yet live on Robinhood Chain, Section 16 governs.

17.1 Case A: a research agent

A human wants a standing read on a market: not a trade, but an informed opinion, refreshed. The mandate's objective is to maintain a current assessment; its constraints forbid any position-taking; its budget is a small daily cap on data purchases. The agent runs on a schedule. Each cycle, it decides which data are worth buying under Eq. (1) — buying a datum only when its expected lift on the assessment exceeds its cents-scale cost — pays for them over x402, and updates its assessment. It never trades. Its entire footprint is reads and payments, and its value is the asymmetry of Section 9.2: cents of information sharpening a judgment worth more.

17.2 Case B: a fenced trading agent

A human wants an agent to act on a strategy, within strict limits. The mandate's objective encodes the strategy; its constraints set a maximum position size and a set of permitted assets; its budget caps cumulative spend; its stop conditions halt it on a drawdown. The agent reasons in language about the market, decides, and — when it acts — prepares a trade that settles non-custodially, pulling exactly the sized amount through its session key. Invariant 1 guarantees it cannot exceed its budget no matter what it reasons; the permitted-action set guarantees it cannot touch a forbidden asset. The human sleeps; the fence holds.

17.3 Case C: a monitoring agent with escalation

A human wants to be told when something matters, and only then. The mandate's objective is to watch a set of conditions; its constraints forbid any action other than notification; its budget covers the cost of the reads it makes to watch. Most cycles, the agent reads, finds nothing actionable, and spends a few cents. When a condition triggers, it escalates to the human rather than acting itself — because its mandate grants it the authority to watch and to warn, but not to trade. This is autonomy calibrated to a task: as much as the task needs, and no more.

18 Custody, Compliance, and Governance Posture

A system that lets software move value must be clear about custody and posture, both to its users and to itself.

Custody. gleem is non-custodial by construction, not by policy. It never holds the

user’s root key or takes possession of user funds. An agent acts through a session key the user granted, whose authority the chain caps. The strongest statement gleem can make about custody is also the simplest: it cannot take the user’s funds, because the architecture gives it no path to.

Compliance posture. Robinhood Chain’s flagship instruments are regulated real-world assets, and their availability is jurisdiction-dependent. gleem does not override those constraints; an agent on gleem is subject to the same availability rules as its human. gleem’s role is to give an agent scoped, legible authority to act within what its human may lawfully do — not to expand that set. This is a deliberately conservative posture, and it is the correct one for a system operating near regulated assets.

Governance. The enforcement that matters — budget caps, permitted actions, custody — lives in contracts and keys, not in gleem’s discretion. This is a feature: it means the guarantees do not depend on gleem’s continued good behavior, only on the chain’s. A guarantee that survives its author is a stronger guarantee than one that does not.

19 Summary of Formal Guarantees

A reader who remembers only this table remembers the document correctly. Each row states a guarantee, its mechanism, and — honestly — whether it is enforced today or committed in the target architecture.

Table 3: Formal guarantees and their enforcement status.

Guarantee	Mechanism	Status
Budget safety (Inv. 1)	session-key cap enforced by account contract	committed; contract pending on 4663
No charge without result (Prop. 1)	x402 automaton transitions	mechanism live; facilitator building
Non-custody	no architectural path to user funds	enforced by construction
Bounded key compromise	scoped, capped, expiring session keys	committed with contracts
Legibility	onchain-anchored decision records	records designed; anchor is settlement

The value of the table is its honesty. A guarantee marked “committed” is a design with a specified mechanism and a pending deployment, not a claim of present function. A guarantee marked “enforced” is one a reader can rely on today. The difference

between those two words is the difference between a manifesto and an advertisement.

20 Limitations and Open Problems

An honest document ends by stating what it has not solved.

The contracts are not yet deployed on Robinhood Chain. Until the registry and session-key suite are deployed and audited on chain 4663, budget safety is a commitment, not an enforced invariant. This is the single largest gap between the architecture described here and the system running today, and we mark it plainly.

The facilitator is a single point of operation. Running our own x402 facilitator means running a funded, always-on service. This is an operational burden and a liveness dependency, and a serious deployment must monitor it, fund it, and secure its signing key. A facilitator that dies silently is a payment layer that dies silently.

Model reliability bounds agent reliability. gleem constrains what an agent can do; it does not make the agent's reasoning good. A calibrated, well-mandated agent is still only as sound as the model beneath it. The enforcement layer bounds the damage of a bad decision; it does not prevent a bad decision.

The market is young. Robinhood Chain launched recently, and its agent economy does not yet exist at scale. gleem is a bet that it will, and a bet is not a proof. We would rather state the bet than disguise it.

21 Roadmap

The roadmap follows risk order, not narrative order: the cheapest, safest capabilities first, the ones that move real value last.

1. **Live reads (done).** Wallet, gas, balances, portfolio — read directly from Robinhood Chain. This is the current, verifiable state.
2. **The facilitator.** Stand up and harden the x402 facilitator for chain 4663, so the payment path is live rather than specified. Verify-then-settle, funded signer, monitored.
3. **Non-custodial execution.** Route swaps through aggregation, settling non-custodially with the user's signature. Prove the execution path with small live transactions before opening it.

4. **Contracts and the invariant.** Deploy and audit the registry and session-key suite on Robinhood Chain, turning budget safety from a commitment into an enforced invariant.
5. **The runtime.** Bring the agent deployment and runtime live on Robinhood Chain, so an agent is a running process, not a specified one.

The order is deliberate: nothing that moves value goes live before the mechanism that bounds it. We would rather ship a reader that works than an executor that might not.

22 Conclusion

The thesis was narrow and falsifiable: agents do not fail at thinking; they fail at living, and an operating system — not a larger model — is the remedy. The chapters between here and the abstract were an attempt to make that thesis precise: to define an agent as a process, a mandate as an enforceable envelope, a payment as a machine-native exchange, and a guarantee as a property of the chain rather than a promise of ours.

gleem is our attempt to build that operating system on Robinhood Chain. Parts of it run today, and this document said which parts, and refused to say more. Parts of it are under construction, and this document specified their mechanisms and marked their status. And the largest guarantees are committed rather than enforced, because the contracts that will enforce them are not yet on the chain — and we would rather write that sentence than hide it.

An agent economy does not arrive because someone announces it. It arrives when agents can pay for what they need, hold funds under limits, persist as processes, and be read and constrained — and when a system asking to manage money is precise about what it can and cannot yet do. That precision is not a weakness of this manifesto. It is the point of it.

Agents are computers that think. This is theirs, and we are building it in the open.