

Why Agents Need an Operating System

Gleem Computer LLC
gleem.computer · Robinhood Chain

A gleem research brief · 001

Abstract

Most deployed “agents” can reason but cannot live: they hold no funds, pay for nothing, persist only for a conversation, and operate under no enforceable limit. This brief argues that the missing layer is not intelligence but an operating system — a runtime that treats an agent as a first-class process with identity, resources, machine-native payment, and constraints enforced below the model. We frame the agent-as-process abstraction, define the mandate as an enforceable envelope, and state the economic condition, value of information, under which an autonomous agent is viable. gleem is our implementation of this layer on Robinhood Chain; this brief is the compressed argument for why such a layer must exist.

1 The gap is infrastructure, not intelligence

A language model, given a goal, can produce a plan a human analyst would respect. Then the context closes and nothing persists. The model did not fail to think; it failed to *live* — to hold resources, act unattended, pay for what it needs, and be bound by rules it cannot argue past.

This is the gap that once separated raw hardware from useful computation. What closed it was not a faster processor but the operating system: processes, memory, system calls, so software could run without a human at the switches. Agents are in the pre-OS era: each is a script a human babysits. The missing layer is the same one that was missing for early programs.

2 The agent as a process

An operating system’s unit of execution is the process: a running program with identity, resources, and permissions granted by the kernel. An agent should be the same, lifted onchain. Four properties separate a process from a prompt.

- **Identity.** A prompt is anonymous. A process has a name under which its history accrues.
- **Resources.** A prompt borrows the human’s context. A process holds its own bounded funds.
- **Authority.** A prompt carries the human’s full credentials. A process has exactly the permissions the kernel granted — for an agent, a session key scoped and capped on-chain.
- **State.** A prompt forgets. A process accumulates a durable, legible record.

Deploying an agent is therefore not sending a longer prompt. It is instantiating a process: an identity, a funded account, a scoped key, a running mandate.

3 The mandate: enforcement below the prompt

An agent is only as safe as the constraint it cannot escape. A mandate is the human’s intent made enforceable: an objective, a set of permitted actions, a budget, and stop conditions. The critical question is where it is enforced.

Enforcement *in the prompt* does not work. “Never spend more than \$100” is a request the model usually honors — and “usually” is not a guarantee a system managing money can run on. A constraint the agent can reason about is one it can violate.

Enforcement *below the prompt* works. The budget is a cap on the session key, checked on-chain. The permitted actions are the only ones the key can sign. The agent cannot exceed its budget because the chain rejects the transaction, not because the agent chose well. This is the difference between an agent and a well-prompted chatbot.

4 The economics: value of information

An agent that cannot pay is a puppet; one that pays more than it earns is not viable. The condition for viability is decision-theoretic. Let a decision be worth V made well and $V - L$ 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

$$(p' - p) L > c. \quad (1)$$

The structural fact that makes an agent economy possible is the asymmetry of scale: information priced in cents can inform decisions valued in dollars. When c is a cent and L a dollar, the inequality holds at a probability lift of barely one percent. This is the entire economic case for machine-native micropayments, in one line.

5 Why an operating system, specifically

Each near-neighbor supplies one faculty and stops. A programmable wallet signs transactions but has no mandate, no budget, no runtime. A trade-by-message bot executes an instruction but does not reason or refuse. A payment protocol moves money but does not trade, reason, or persist. Autonomy requires all of these together — identity, resources, payment, execution, enforced constraint — and assembling them is precisely what an operating system does.

gleem is our attempt at that operating system, built on Robinhood Chain: a permissionless EVM Layer 2 whose gas token is ETH and whose settlement stablecoin, USDG, implements the EIP-3009 authorization primitive that machine-native payment requires. The full architecture, and an honest account of what is live versus under construction, is given in the gleem manifesto.

6 Conclusion

Agents do not fail at thinking. They fail at living. A larger model closes none of that gap; an operating system closes all of it. The claim is narrow and falsifiable: an agent economy emerges not from better reasoning alone, but from a runtime that lets agents pay for what they need, hold funds under limits, persist as processes, and be read and constrained. That runtime is the work.

gleem.computer · see also: the gleem manifesto, and research 002, “Paying Without Keys.”