How Do AI Agents Work?

A mechanics walkthrough of the machine underneath the buzzword — the loop it runs, how it decides, how it calls tools and remembers, and how a crypto agent actually signs a transaction and pays.

Alain AI Lab Research · Published July 13, 2026 · 10 min read

It is easy to say an AI agent "thinks for itself." It is more useful to open the box and watch the parts move. Under the branding, an agent is a fairly mechanical arrangement: a language model that decides one step at a time, a program that carries those steps out, a memory that feeds context back in, and — in crypto — a wallet that lets the whole thing pay for what it needs. None of it is magic, and understanding the plumbing is the difference between trusting a system and being fooled by one. This report follows a single question through the machine: when you hand an agent a goal, what actually happens next? If you want the plain definition first, start with our explainer on what an AI agent in crypto is; here we go under the hood.

AT A GLANCE

CORE LOOP **Perceive → reason → act**	THE ENGINE **A stateless LLM**	DECIDING **ReAct + planning**
ACTING **Structured tool calls**	REMEMBERING **Context window + RAG**	CRYPTO STEP **Sign tx · pay via x402**

01

The Loop at the Center

Every agent runs a loop, and the loop is the whole idea. Handed a goal, the agent *perceives* its current situation — the instruction, plus whatever information it has gathered so far — then *reasons* about what to do next, *acts* by using a tool, *observes* the result, and repeats. It keeps cycling until the goal is met or it decides to stop. That perceive–reason–act–observe cycle, rooted in classical agent theory, is what separates an agent from a one-shot program:

nothing is fixed in advance except the objective. A script that always runs the same three commands is not looping in this sense; an agent chooses each next move based on what just happened. The exact wording varies by vendor, and you should treat it as a conceptual model rather than a rigid specification — but the shape holds across nearly every real system. Hold onto it, because everything that follows is a detail of one of these four beats.

02

The Engine That Forgets Everything

At the heart of the loop sits a large language model, and the single most important mechanical fact about it is one marketing rarely mentions: the model is *stateless*. Each time it is called, it sees only the text placed in front of it in that moment — its context window — and it retains nothing afterward. It does not "remember" the last step the way a person recalls a conversation; every apparent memory is simply prior text being re-fed into the next prompt by the surrounding program. Nor does the model learn while it works: its internal weights are frozen at inference, so an agent runs the *same* model on every step, changing only what it is shown. When a product claims its agent "learns and improves," it almost always means the software is storing more context to feed back in, not that the model itself is changing. Grasping this reframes the entire system: the intelligence is real, but the continuity is an illusion maintained by the code around the model.

03

How It Decides What to Do

If the model is stateless, how does an agent reason through a multi-step problem? Through a handful of well-studied prompting patterns. *Chain-of-thought* prompting has the model write out intermediate steps before committing to an answer, which improves its handling of complex tasks. The dominant agent pattern, *ReAct* — short for Reasoning and Acting — interleaves those written thoughts with actions: the model reasons a little, takes a tool action, reads the result, then reasons again with that new information in view. A related family, *plan-and-execute*, has the model first draft a full multi-step plan and then work through it, re-planning when a step fails. One honest caveat runs under all of this: the "reasoning" you see is generated text, and it need not faithfully mirror the computation actually happening inside the model. It is a powerful and useful pattern — not a window into a mind. What matters mechanically is that the model keeps emitting a next-step decision, and the loop keeps feeding it forward.

04

How It Reaches Into the World

An agent that could only talk would be a chatbot. What makes it an agent is *tool use*, also called function calling, and the mechanism is precise. The program hands the model a schema — a list of available tools, each with a name and the arguments it accepts. When the model wants to act, it does not run anything itself; it emits a structured request, typically a tool name plus arguments in JSON. The surrounding program — the harness — parses that request, executes the real function, and feeds the result back into the model as a fresh observation. This is the crucial and often-missed point: the model only ever *proposes*; the code disposes. Tools can be almost anything with an interface — a web search, a code sandbox, an API call, a database query, and, in crypto, a call to a smart contract. Both major model providers expose this tool-calling loop as a first-class feature, and it is the exact seam where an agent's decisions become effects in the world.

05

How It Remembers Across Steps

Because the model forgets between calls, memory has to be built around it, and it comes in two layers. *Short-term memory* is just the context window — the recent conversation and scratchpad the model can see right now. It is fast but finite; once the window fills, older material must be dropped or summarized. *Long-term memory* is where the interesting engineering lives. The common approach is retrieval-augmented generation, or RAG: documents are converted into *embeddings* — numeric vectors that capture meaning — and stored in a vector database. When the agent needs to recall something, its query is embedded too, the database returns the most similar entries by distance, and those snippets are injected into the prompt. The model then answers as if it "knew" them, though it only read them a moment ago. RAG is a retrieval technique rather than true memory, and it is not the only method — summaries and key-value stores are also used — but it is the workhorse that lets an agent draw on far more than fits in a single window.

06

How the Pieces Get Wired Together

Model, tools, and memory do not assemble themselves; that is the job of an *orchestration framework*. These libraries manage the loop — deciding when to call the model, when to run a tool, how to thread memory through — so a developer does not rebuild the machinery each time. LangGraph models an agent as a stateful graph of steps; CrewAI arranges specialized agents into role-based "crews"; AutoGen pioneered conversation-driven coordination,

though Microsoft has since folded it into a broader agent framework. In crypto, *elizaOS* — the open-source framework that grew from the ai16z project and rebranded in early 2025 — fills the same role. Two ideas travel with these tools. The first is *multi-agent systems*: rather than one do-everything agent, several narrow agents are coordinated by an orchestrator or by passing messages between them. The second is the Model Context Protocol, an open standard from Anthropic introduced in late 2024 for connecting models to tools and data in a uniform way; adoption is growing but not yet universal. Our deeper look at AI agent frameworks like LangGraph, CrewAI and AutoGen takes these apart in full.

07

How a Crypto Agent Signs and Pays

Everything so far applies to any AI agent. The crypto version adds one mechanically distinct tool: a wallet the agent can operate in code. To act on-chain, the agent constructs a transaction and *signs* it programmatically — no human clicking "confirm." Because handing raw private keys to autonomous software is dangerous, this is usually done through *smart accounts* built on account abstraction (the ERC-4337 standard), where the wallet is itself a contract that can enforce rules. *Session keys* — scoped, temporary signers with preset spending limits — let an agent transact within bounds without exposing a master key, while *MPC* systems split a key into shares so no single party ever holds it whole. Payment then rides a purpose-built rail: *x402*, introduced by Coinbase in 2025, which revives the HTTP status code 402, "Payment Required." The handshake is clean: the agent requests a resource, the server answers 402 with payment terms, the agent's wallet signs and sends a stablecoin payment such as USDC, a facilitator verifies and settles it on-chain, and the server returns the resource. Software pays software over the ordinary web. To see this assembled end-to-end, read our guide on how to build an AI crypto research agent.

> **The model runs nothing:** the LLM never executes a tool or moves a coin — it only emits a decision. Every action, every signature, every payment is carried out by the surrounding code. When something goes wrong, the failure is almost always in what the harness was allowed to do, not in the model "deciding" to misbehave.

08

Where the Machine Breaks

Knowing how an agent works also reveals how it fails. The sharpest weakness is *prompt injection*, ranked the number-one risk in the security community's catalogue of language-

model vulnerabilities. Because the agent reads untrusted content — web pages, tool outputs, on-chain text — a malicious instruction hidden in that content can hijack the loop, redirecting the agent to act against its owner. When the agent holds a funded wallet, that hijack becomes an irreversible transfer, because on-chain settlement is final: confirmed transactions have no chargeback and no undo. Two more failure modes follow from the mechanics we have traced. The model can *hallucinate* — produce fluent, confident, wrong output — and autonomy removes the human who would have caught it before money moved. And the harness itself can be over-permissioned, granting the agent tools or spending scope it never needed. A note on the lurid headlines: specific, dated, nine-figure "AI agent hack" figures circulate widely and are typically uncorroborated; documented losses remain application- and key-management failures, not breaks in a blockchain's core.

"Through wisdom is an house builded; and by understanding it is established."

PROVERBS 24:3

METHODOLOGY & SOURCES

This report was compiled from primary technical documentation and corroborating industry sources, cross-checked by a multi-agent research review. The perceive–reason–act loop reflects classical agent theory; ReAct (reasoning-and-acting) and chain-of-thought are established prompting techniques from the research literature (Yao et al.; Wei et al.), and the "reasoning" they surface is generated text, not a guaranteed account of internal computation. The stateless-model and frozen-weights points are standard properties of transformer inference. Tool/function calling — the model emitting a structured call that a harness executes — is documented by both OpenAI and Anthropic. RAG (embeddings in a vector database, retrieved by similarity) is the common long-term-memory approach, not the only one. Orchestration frameworks (LangGraph, CrewAI, AutoGen — since folded into Microsoft's broader agent framework — and elizaOS, rebranded from ai16z in early 2025) and the Model Context Protocol (Anthropic, late 2024) exist as described; MCP adoption is growing, not universal. Crypto execution mechanics — programmatic signing, ERC-4337 account abstraction, session keys, MPC key management, and the x402 402-Payment-Required handshake settling USDC (initially on Base) — are drawn from Coinbase and Ethereum standards documentation. Prompt injection is ranked first in the OWASP Top 10 for LLM Applications (2025). Specific dated nine-figure "AI agent hack" figures were treated as uncorroborated and excluded. Sources: openai.com, anthropic.com (incl. Model Context Protocol), arxiv.org, coinbase.com/developer-platform, eips.ethereum.org, owasp.org. Educational only; not financial advice.

Alain AI Lab — independent crypto & AI research.
intelligencecrypto.org · This document is for educational purposes only and is not financial advice.