Skills Needed to Build an AI Agent

The real competency map for building AI agents — from Python and prompting to tools, evaluation, security, and on-chain work.

Alain AI Lab Research · Published July 15, 2026 · 10 min read

Building an AI agent is often pitched as a weekend project: wire a language model to a few tools and let it run. That framing hides how much genuine engineering sits underneath a system that behaves reliably. An agent is not a single skill but a *stack* of them — programming, model interaction, orchestration, evaluation, and security — layered so that a model can reason, call tools, and act toward a goal without a human in the loop for every step. This report maps the skills that actually matter, separates the durable ones from the hyped ones, and shows where the work gets genuinely hard. For the mechanics of what an agent is doing while it runs, our companion piece on how AI agents work covers the loop in detail; here the focus is the builder, not the build.

AT A GLANCE

PRIMARY LANGUAGE **Python (JS/TS secondary)**	THE CORE PIVOT **Tool use / function calling**	REASONING PATTERN **ReAct: reason–act–observe**
HARDEST SKILL **Evaluation (non-determinism)**	TOP SECURITY RISK **Prompt injection (OWASP LLM01)**	DESIGN MAXIM **Simplest solution first**

01

What “Building an Agent” Actually Requires

The first skill is judgment about whether you need an agent at all. Anthropic’s widely cited engineering note *Building Effective Agents* draws a sharp line between two things people lump together. A **workflow** is a system where models and tools are orchestrated through predefined code paths — the developer decides the sequence. An **agent** is a system where

the model dynamically directs its own process and tool usage, deciding how to accomplish the task. Agents buy flexibility at the cost of predictability, latency, and expense. Anthropic's guidance is deliberately conservative: find the simplest solution possible and increase complexity only when it demonstrably improves outcomes — which "might mean not building agentic systems at all." A builder who reaches for a multi-agent architecture when a single well-structured prompt would do has failed before writing a line of code. The best practitioners treat autonomy as a cost to be justified, not a default.

02

The Programming Foundation

Agents are software, and the ordinary discipline of software engineering does not become optional because a model is in the loop. **Python** is the dominant language of the field: the major frameworks — LangChain and LangGraph, CrewAI, AutoGen, LlamaIndex — are Python-first, with JavaScript and TypeScript forming a real secondary ecosystem through official ports. Beyond the language itself, the prerequisites are unglamorous and non-negotiable: comfort with **REST APIs** and HTTP, **asynchronous programming** (an agent waiting on several tool calls at once), disciplined **error handling and retries**, and version control. Agents fail in mundane ways — a timed-out API, a malformed response, a rate limit — and a builder who cannot reason about network calls and failure modes will produce something that demos well and breaks in production. The model is the interesting part; the plumbing is what keeps it standing.

03

Talking to the Model: Prompting and Structured Output

Interacting with the model itself is its own competency, though a smaller one than the 2023 hype suggested. **Prompt engineering** — writing clear instructions, showing examples, structuring context — is a genuine skill, but it is a skill *within* a broader engineering role rather than a standalone career; the market for "prompt engineer" as a job title has largely folded back into ordinary development. What endures is fluency with the **LLM API layer**: understanding context windows, the effect of temperature and sampling on output, and how tokens drive both cost and latency. Equally important is **structured output** — forcing the model to return validated JSON rather than free prose. In Python this usually means Pydantic models or JSON-schema validation, backed by provider-side features that constrain the model to a schema. Free text is where agent pipelines quietly break; a builder who can guarantee the shape of a model's output has removed an entire class of downstream

failure. The distinction worth internalising is that provider-side constraints and client-side validation are two layers, not one: the API can be asked to emit schema-conforming JSON, and the application should still validate and retry when it does not.

04

The Agent Loop: Tools, Function Calling, and ReAct

The single skill that converts a chatbot into an agent is **tool use**, also called **function calling**. The builder describes each available tool with a JSON schema — its name, purpose, and typed parameters — and the model responds not with prose but with a structured request to invoke one. The application runs the tool, returns the result, and the model decides what to do next. That cycle is the heart of agency. The canonical pattern organising it is **ReAct** — reason, act, observe — introduced by Yao and colleagues in a 2022 paper later published at ICLR 2023, in which the model interleaves explicit reasoning traces with tool actions and observations. Understanding this loop is what lets a builder debug the questions that actually arise: why an agent looped forever, why it called the wrong tool, why it declared success on bad data. The loop is simple to describe and surprisingly hard to make robust.

> *A framework will give you the agent loop in a few lines of code. It will not give you the judgment to know when the loop has gone wrong — that judgment is the skill, and it is the one no library ships.*

05

Knowledge and Memory: RAG, Embeddings, and State

Most useful agents need to know things beyond the model's training data, and the standard technique is **retrieval-augmented generation** — RAG — introduced by Lewis and colleagues in 2020. The builder converts documents into **embeddings**, stores them in a **vector database**, and retrieves the most relevant passages by similarity to ground the model's answer in real source material. Adjacent but distinct is **memory**: the ability to persist state across turns and sessions. It is useful to separate short-term memory — what fits in the current context window — from long-term memory, which stores information for later recall, though in practice long-term memory is frequently implemented with the very same retrieval machinery as RAG. The honest framing is that these techniques overlap rather than being cleanly separate; a builder should understand embeddings and similarity

search as the shared substrate, and treat “RAG” and “memory” as two applications of it rather than rival systems.

06

Frameworks and Orchestration

A builder should be fluent in at least one agent framework like LangGraph, CrewAI, and AutoGen — alongside newer entrants such as OpenAI’s Agents SDK and Google’s ADK. It is a mistake to hunt for the “best” one; they largely wrap the same underlying **orchestration patterns** in different vocabularies. The recurring primitives are a supervisor delegating to workers, graphs or state machines that route control between steps, and sequential pipelines that hand a task down a chain. What matters is understanding those patterns well enough to recognise them beneath whichever framework’s branding you happen to be using — OpenAI calls it a handoff, Google nests a root agent over sub-agents, AutoGen runs a group chat, and all three are the same delegation idea. A related skill is knowing the interoperability layer: the **Model Context Protocol** (MCP), introduced by Anthropic in November 2024 and moved under an Agentic AI Foundation — a directed fund of the Linux Foundation — in late 2025, has become one widely adopted standard for connecting agents to tools and data, though it is one option among several rather than the only one.

07

The Hard Part: Evaluation, Observability, and Cost

The skills that separate a demo from a product are operational, and **evaluation** is the hardest of them. Agents are *non-deterministic* — even at temperature zero, run-to-run variation persists for reasons rooted in floating-point and hardware behaviour, not just sampling. Outputs are open-ended, errors compound across multi-step trajectories, and there is rarely a single ground truth, so ordinary pass/fail unit testing does not apply. Builders rely instead on curated evaluation datasets, human review, and **LLM-as-judge** scoring — using a model to grade another model’s output, a scalable method that carries documented biases (toward verbose, self-favouring, or well-positioned answers) and cannot stand alone. Alongside evaluation sits **observability**: tracing tools such as LangSmith and its peers that record an agent’s full multi-step trajectory, because a failure often looks like a successful tool call returning wrong data rather than a crash. Finally, **cost and latency management** — agent loops make many model calls, and budgets, caching, and step limits are what keep a working agent economically viable. This emerging operational discipline, sometimes labelled AgentOps, is an extension of MLOps rather than a settled field.

08

Security and the On-Chain Frontier

Because agents act on data from untrusted sources, **security** is a core skill, not an afterthought. **Prompt injection** — hostile instructions smuggled into content the agent reads — ranks as LLM01, the top entry in the OWASP Top 10 for LLM Applications (2025), with indirect injection through webpages, documents, or tickets as its most insidious subtype. The defensive competencies are recognisably those of application security applied to a new surface: least privilege, input and output validation, allowlists, sandboxing, and human-in-the-loop approval gates for high-risk actions. Building agents that touch **crypto** raises the stakes further, because on-chain transactions are largely irreversible. Such work demands all the general skills plus blockchain-specific ones — interacting with RPC nodes and indexers, handling wallets and transaction signing, reading smart contracts and ABIs, and managing gas. The governing rule is that an agent should never directly hold private keys; signing belongs behind key-management services, spending caps, and allowlists. Emerging standards such as account abstraction and machine-payment protocols are worth watching here, but they remain illustrative building blocks rather than settled requirements, and a builder should treat the security posture — not any single tool — as the fixed point. For a concrete, applied walkthrough of assembling these pieces into a working system, see our build guide on how to build an AI crypto research agent. The soft skills sit above all of it: the field moves quickly, and the durable meta-skill is the willingness to iterate, read model behaviour closely, and keep learning as the tools shift underneath you.

> *"Through wisdom is an house builded; and by understanding it is established: and by knowledge shall the chambers be filled with all precious and pleasant riches."*
>
> PROVERBS 24:3–4

Methodology & Sources

This report was produced with a parallel multi-agent verification process, each claim checked and labelled verified, partial, or uncertain before inclusion. Figures are kept qualitative where precise public numbers are contested or unconfirmed against a primary source; no salary figures, job-market percentages, benchmark scores, framework market shares, or adoption counts were invented, and several unverified third-party statistics encountered during research — including circulated salary and job-loss numbers for prompt engineering — were deliberately excluded.

Primary references include Anthropic's engineering guidance *Building Effective Agents* (December 2024) for the workflow-versus-agent distinction and its simplicity principle; the OWASP Top 10 for LLM Applications (2025) for the ranking of prompt injection as LLM01 and its indirect-injection subtype; the ReAct paper by Yao et al. (arXiv 2022, ICLR 2023) for the reason–act–observe loop; Lewis et al. (2020) for retrieval-augmented generation; Anthropic's Model Context Protocol documentation and the December 2025 announcement of its move under the Agentic AI Foundation, a directed fund of the Linux Foundation; and the public documentation of the LangGraph, CrewAI, AutoGen, LlamaIndex, OpenAI Agents SDK, and Google ADK frameworks. Crypto and on-chain claims are framed as best-practice and enumerated skill categories rather than settled requirements. Educational content only; not financial advice.

Alain AI Lab — independent crypto & AI research.
intelligencecrypto.org · Research is educational and analytical, not financial advice.