PERSONAL AI LAB · REFERENCE MANUAL V1.2

The Agent Builder's Field Manual

Everything worth knowing about AI agents in 2026: the architecture, the full ecosystem, the decisions that matter, and the path from automation analyst to agent engineer.

Author Mathieu Tellene

Scope Concepts · Ecosystem map · Decision frameworks · Learning path

Compiled 20 September 2026 · revision 1.2

Status Living document. The concepts last; the tool names move quarterly.

How to read this manual

This manual has one job: to make the AI agent landscape legible, so that every tool you meet from now on lands in a slot you already understand instead of adding to the noise.

It is organised as five parts, and they are deliberately unequal in nature.

Part	What it covers	Why it is here
I	Foundations	The mental model. Four layers, one loop. Everything else in the manual is an instance of this. Read it once properly and the other four parts become skimmable.
II	The ecosystem map	A complete inventory of the 2026 landscape, sorted into twelve categories. Written to be used as a lookup table, not read end to end.
III	Decision frameworks	The part that is actually hard. Knowing the tools is cheap; knowing which one a given problem deserves is the job.
IV	What separates an expert	The eight behaviours that distinguish someone who ships agents from someone who demos them, the failure modes that create that gap, and the nine-step process experienced builders actually follow.
V	The learning path	Certifications ranked by real market weight, courses ranked by signal, portfolio projects that read as senior, and a twelve-week curriculum.

A note on shelf life

Two kinds of content sit in this manual, and they age very differently.

The structural content does not move. The tool-calling loop, the stateless nature of model APIs, the difference between a workflow and an agent, the context budget, the reasons evaluation matters: all of that will read the same in three years.

The inventory moves every quarter. Product names, version numbers, prices and market positions in Part II are accurate as of September 2026 and will decay. Treat that part as a snapshot with a date stamp, not as a fact.

> THE ONE SENTENCE TO KEEP
>
> **A model does not act. It reads text and writes text. Everything people call "an AI agent" is a loop of ordinary code that reads what the model wrote, runs the function the model named, and hands the result back.**

Conventions used here

- CORE marks something worth mastering regardless of employer or stack.
- CONTEXT marks something worth recognising but not studying now.
- CAUTION marks a common trap or an overstated claim.
- Prices are in USD unless stated, and are list prices at time of writing.

Contents

PART I

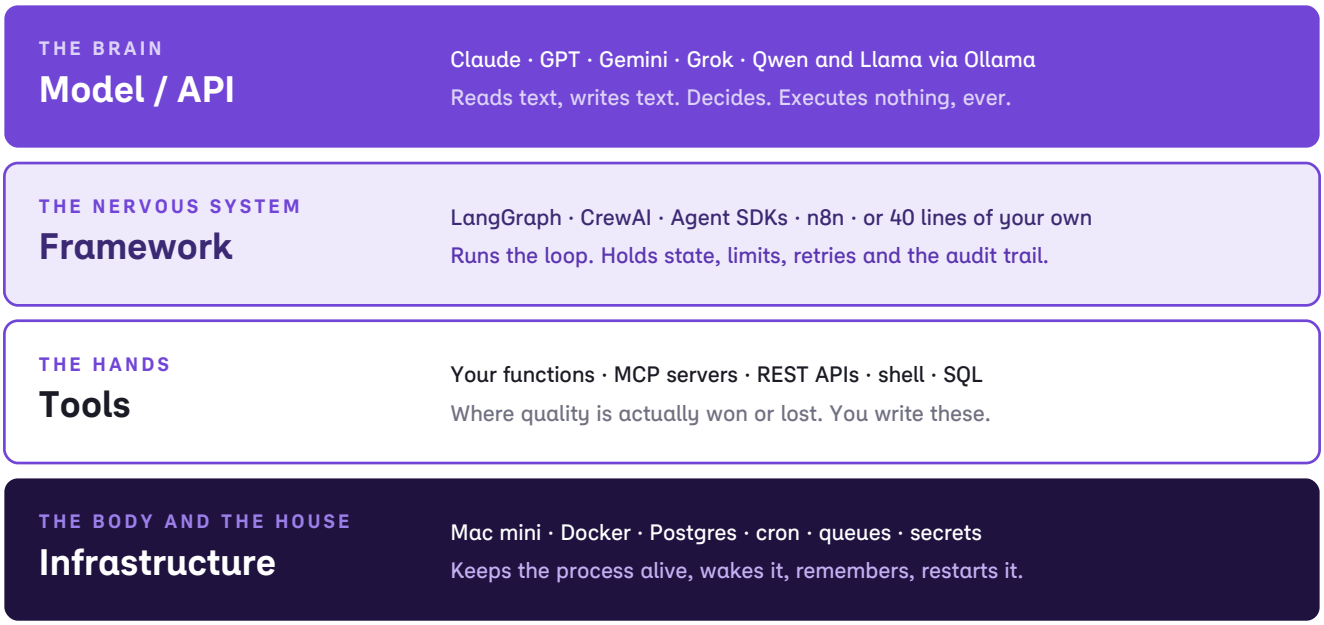
Foundations

Four layers and one loop. Everything called an agent, from a node in n8n to a fleet of coordinated subagents, is an instance of the same mechanism. This part builds that mechanism from the bottom up, so the rest of the manual reads as variations rather than as new material.

CHAPTER 1

The four layers

Most confusion about AI engineering comes from mixing things that live at different layers. LangChain and Claude are not alternatives to each other. n8n and LangGraph are not alternatives to Python. Sorting them into layers dissolves most of the questions before they are asked.



Vendor supplies the top layer. You supply the other three.

Figure 1. The four layers of any agent system. A tool that occupies one layer is never a substitute for a tool in another.

1.1 The model is a pure function

The single most useful correction to most people's mental model is this: a large language model is a function from text to text. It has no memory between calls, no ability to reach the network, no filesystem, no clock. Given the same input it produces a similar output and then it is gone.

When you "give Claude tools", you are not attaching hands. You are appending a written menu to the prompt: a list of function names, descriptions and JSON schemas. When the model "uses a tool", all it does is emit a structured block that says I would like to call `search_drive` with these arguments. Nothing has happened yet. Something outside the model must read that block, actually run the function, and feed the result back in.

> **WHY THIS MATTERS PRACTICALLY**
>
> Every capability gap you will ever hit is on one of the three layers you own, not on the model. The model cannot be slow at reading your database, cannot get your API credentials wrong, and cannot lose your conversation history. Those are your layers.

1.2 Tools are the layer that decides quality

Practitioners consistently report the same surprise: the ratio of effort shifts away from prompt writing and towards tool design. A tool's name, its description, its parameter schema, its defaults and, above all, **the error message it returns when it fails** determine whether the model recovers gracefully or spirals.

A tool that returns `Error: 400` teaches the model nothing. A tool that returns `No file matched 'invoice Q3'. 41 files exist in this folder. Try listing the folder first, or search without a date filter.` turns a dead end into a next step. That difference is engineering, and it is invisible in any framework comparison table.

1.3 The framework is the loop plus the bookkeeping

Strip a framework down and what remains is a `while` loop. What you pay a framework for is everything wrapped around that loop: persistence between runs, the ability to pause for a human and resume later, retry policy, step limits, spend limits, structured traces, and streaming. A framework that gives you none of those is giving you an abstraction and nothing else.

1.4 Infrastructure is the layer people discover last

An agent that only runs while a laptop is open is a script with ambitions. The infrastructure layer answers four unglamorous questions, and a system that cannot answer them is not in production regardless of how sophisticated its reasoning is.

Question	What answers it
What process is running?	A supervised daemon, a container with a restart policy, a serverless function, a worker consuming a queue
Who wakes it?	cron, a webhook, a queue message, a file-system event, a human
Where does it remember?	Postgres, Redis, a file. Never the model provider
Where do the credentials live?	Environment variables, a keychain, a secret manager. Never in the prompt, never in git

CHAPTER 2

The agent loop

This is the entire mechanism. Every framework in Part II implements this and adds bookkeeping around it. Once it is clear, no agent product can be mysterious, only better or worse engineered.

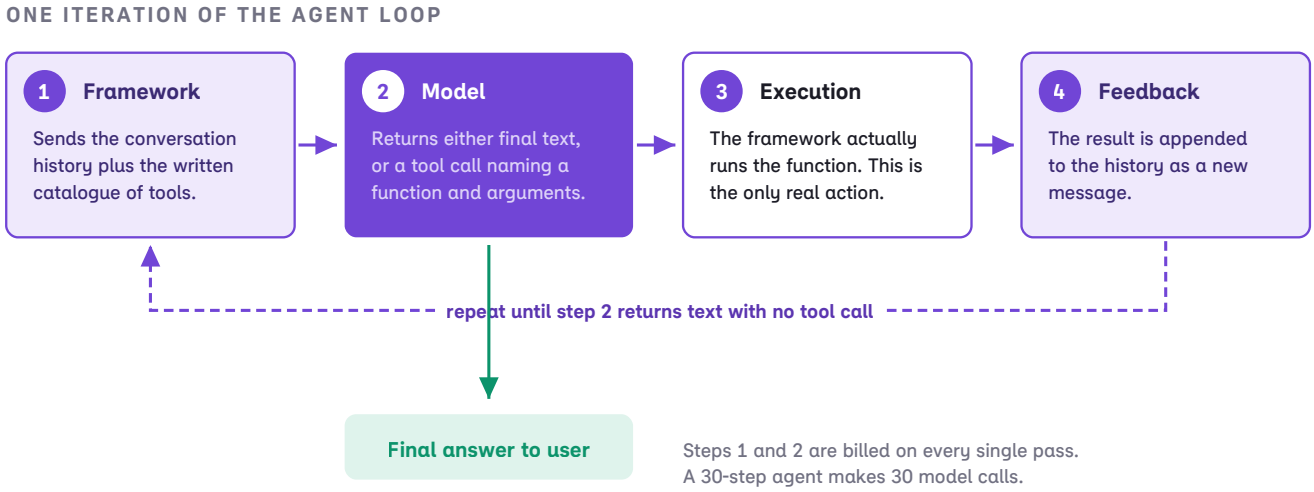


Figure 2. The agent loop. Frameworks differ in what they wrap around this, not in the loop itself.

2.1 The loop in code, with no framework

This is a complete, working agent. Reading it once is worth more than reading three framework tutorials, because every framework in this manual is a more careful version of it.

```
# A complete agent. No framework. This is all an agent is.
import anthropic, json

client   = anthropic.Anthropic()
messages = [{"role": "user", "content": goal}]

for step in range(MAX_STEPS):                # a hard limit is not optional
    reply = client.messages.create(
        model="claude-sonnet-5",
        max_tokens=4096,
        tools=TOOL_SCHEMAS,                    # the written menu
        messages=messages,
    )
    messages.append({"role": "assistant", "content": reply.content})

    calls = [b for b in reply.content if b.type == "tool_use"]
    if not calls:                              # no tool call means it is done
        return reply.content[0].text

    results = []
    for c in calls:
        try:
            out = TOOLS[c.name](**c.input)       # the only real action
```

```
            except Exception as e:
                out = f"Tool failed: {e}. Try a different approach."
            results.append({"type": "tool_result",
                            "tool_use_id": c.id,
                            "content": str(out)[:8000]})   # truncate: context is a budget

        messages.append({"role": "user", "content": results})

    raise RuntimeError("Step limit reached without a final answer")
```

Five things in that snippet are the difference between a toy and something you would leave running unattended, and none of them are about the model.

- **A step limit.** Without it, a confused agent loops until your budget is gone.
- **Tool errors are caught and returned as text.** A raised exception ends the run. A described failure lets the model route around it.
- **Tool output is truncated.** One unbounded database query can consume the entire context window in a single step.
- **The full history is resent every iteration.** This is why cost grows quadratically with conversation length, and why compaction is a real engineering concern.
- **The final answer is recognised by the absence of a tool call**, not by the model announcing it is finished.

> **THE ECONOMICS PEOPLE MISS**
>
> A thirty-step agent run is thirty model calls, and each one resends everything that came before. Cost does not scale with the length of the answer, it scales with the square of the conversation. This is the mechanical reason behind almost every "why is my agent so expensive" question.

CHAPTER 3

Scripts, workflows and agents

The word "agent" is applied to everything from a saved prompt to a fleet of coordinated processes. There is one test that cuts through it, and one gradient that describes everything in between.

> Who decides the next step: your `if` statement, or the model?

If the path is written in your code, it is a workflow, however much AI it calls along the way. If the path is chosen at runtime by the model, it is an agent. A second test sharpens the first: **if you add a new tool tomorrow without touching the control flow, does the system start using it on its own?** If yes, it is an agent.

3.1 The autonomy gradient

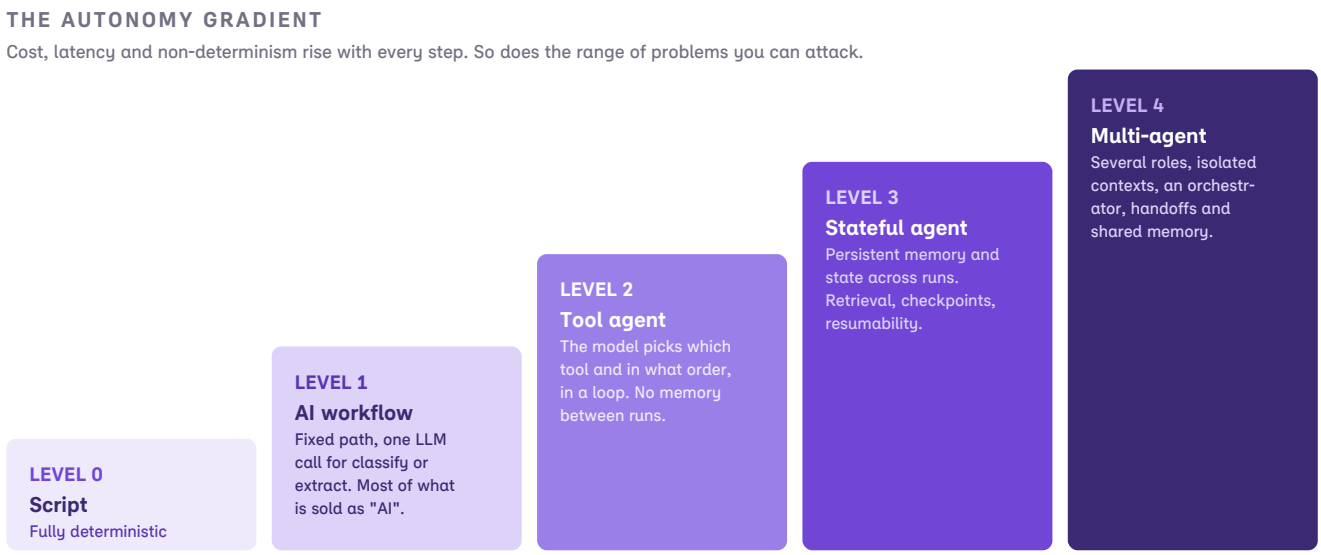


Figure 3. Autonomy is a gradient, not a switch. Most production value sits at levels 1 and 2; most career value sits in being able to build level 4 and in knowing when not to.

3.2 Worked example: reading invoices from Google Drive

Level 0: the script

```
files = drive.list(folder_id)
for f in files:
  if "invoice" in f.name.lower():
    t = extract_text(f)
    n = re.search(r"TOTAL:\s*([\d,.]+)", t)
    sheets.append([f.name, n.group(1)])
```

Breaks when: the file is named `rechnung_03.pdf`; the PDF is a scan; a supplier writes `AMOUNT DUE`; there is a nested folder nobody mentioned.

But: deterministic, near-free, fast, fully auditable. If it works, it works identically ten thousand times.

Level 2 and up: the agent

Goal. Put every supplier invoice from July to September into the sheet, with supplier, date, net, VAT and total. If something does not reconcile, do not invent it: ask.

Tools. `search_drive`, `list_folder`, `read_pdf`, `ocr_pdf`, `append_row`, `ask_human`.

Limits. 40 steps, 2 USD, never delete.

Given that, the agent does something like the following, and nobody wrote steps two through five:

1. Searches for "invoice", finds twelve, judges that too few for a quarter.
2. Lists the folder, notices a `Proveedores/2026` subfolder, opens it.
3. Finds thirty more files named `invoice_*` and `rechnung_*`, and treats them as invoices because it understands the concept rather than matching a string.
4. Opens one, gets empty text back, infers it is a scan, and switches to `ocr_pdf`.
5. Extracts the fields, spots that net plus VAT does not equal total on one document, and calls `ask_human` instead of writing a plausible number.
6. Writes forty-one clean rows and reports the three it did not trust.

> **THE HONEST TRADE**
>
> That run costs roughly fifty times the script and takes roughly thirty times as long, and two runs may differ. You buy tolerance for the unforeseen, and you pay in determinism, money and latency. **An agent is the right answer only where the path genuinely cannot be written in advance.**

3.3 Where each level belongs

Level	Use it when	Do not use it when
0 Script	Inputs are structured and the rules are stable	Format drift is constant and you are patching weekly
1 AI workflow	One messy step inside an otherwise fixed pipeline: classify, extract, summarise, translate	The pipeline shape itself has to change per input
2 Tool agent	The sequence of actions depends on what is found along the way	A three-branch decision tree would have covered it
3 Stateful agent	Work spans sessions, needs memory, or must pause for approval	Every run is genuinely independent

Level	Use it when	Do not use it when
4 Multi-agent	Distinct expertise is needed, or contexts must be isolated to stay clean	One agent with good tools would do. This is the most over-applied level in the field

CHAPTER 4

Where an agent lives

An agent is not an entity that lives "in the AI". It is a folder of code, a running process, and a database. Where it lives is a backend decision, and the fact that it calls a model changes nothing about it.

4.1 The artefact

Stripped of mystique, a serious agent project looks like this on disk. Note how much of it has nothing to do with machine learning.

```
my-agent/
├── agent.py              # the loop: who decides the next step
├── tools/
│   ├── drive.py          # the hands, one module per surface
│   ├── warehouse.py
│   └── slack.py
├── prompts/
│   └── system.md         # versioned like code, because it is code
├── evals/
│   ├── cases.jsonl       # 80 to 200 recorded cases
│   └── run_evals.py      # the thing that says better or worse
├── migrations/           # state schema, because state is a database
├── .env.example          # names of secrets, never the values
├── Dockerfile
└── docker-compose.yml    # agent + postgres + tracing, one command
```

4.2 The four things that keep it alive

Need	Real question	Where it lives
Body	What process is running right now?	A container, a supervised daemon, a serverless invocation, a queue worker
Heartbeat	Who wakes it?	cron, webhook, queue message, Slack event, a human typing
Memory	Where does it remember?	**Your** Postgres or Redis. The model provider stores nothing for you
Keys	Where are the credentials?	Environment, keychain or secret manager. Never in the prompt, never in the repo

STATELESS BY DESIGN

Model APIs are stateless. The continuity a user experiences over days is not stored by Anthropic or OpenAI. **It is stored by you, and replayed into every single call.** Your database is where the agent actually lives; the model is a service you rent by the token.

4.3 The shapes an agent can take at runtime

The same `my-agent/` folder can adopt every shape below without changing the loop. Only the wrapper changes: a `__main__` block, an HTTP route, a `while True`, or a crontab line.

Shape	How it lives	Choose it when
CLI script	Starts, acts, exits	Development, one-off jobs, human-triggered tasks
Scheduled job	Asleep until cron wakes it	Daily reports, periodic monitoring, batch reconciliation
Long-running daemon	Always-on process	Socket-mode chat bots, continuous listeners, in-memory state
HTTP service	Waits for requests	Another system calls it; you need a URL; you need streaming
Queue worker	Consumes a task queue	Volume, retries, long tasks, back-pressure, fan-out
Serverless function	Exists only during execution	Spiky traffic, zero idle cost, short runs. Watch cold starts and timeouts
Node inside n8n	n8n is the living process; the agent is one step	The hard part is integrations, not reasoning
Managed platform	Someone else's infrastructure	You do not want to operate anything. Costs flexibility and portability

4.4 Runtime anatomy of a self-hosted lab

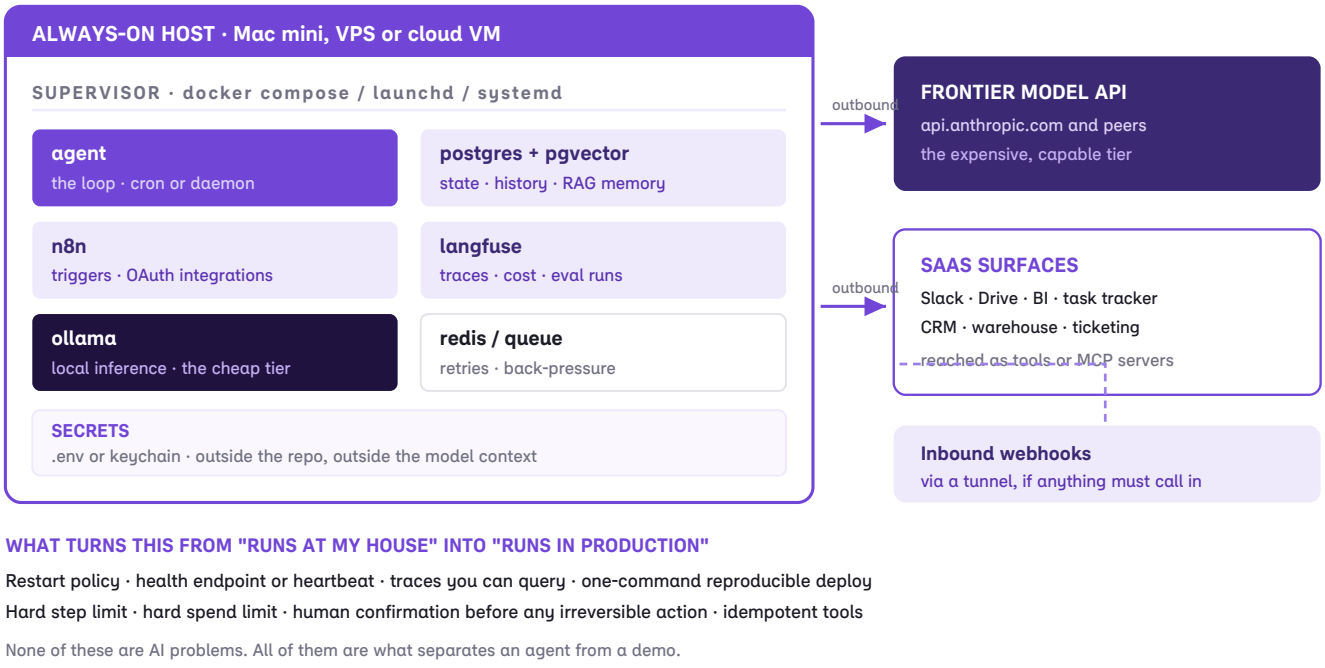


Figure 4. A self-hosted agent lab. The frontier model is the only part that does not live on the host.

4.5 The deployment ladder

Where an agent runs is read by other engineers as a signal about you. The rungs matter less than the jump from rung one to rung three.

Rung	Where it runs	What it signals
0	Your laptop, launched by hand	You followed a tutorial
1	An always-on host with Docker and cron	You can deploy and keep something alive
2	Rung 1 plus persistent state, traces and a health check	You think in operations, not only in code
3	A VPS or a managed platform, deployed by CI on push	You work the way a team works
4	Container platform with queues, retries and full observability	You can be put on production systems

4.6 Running something 24/7 in practice

Four things have to be configured on day one, or the machine will be asleep at the moment the agent was supposed to act. None of them are interesting, and all of them are the difference between a lab and a toy.

Concern	What to do
It must not sleep	Disable system sleep, and on macOS keep a `caffeinate` process alive through a LaunchAgent so a setting change cannot silently undo it
It must come back	Enable automatic restart after a power failure, and give every container a restart policy. A host that survives a power cut but whose services do not is not solved
Services must survive a reboot	`launchd` agents on macOS, `systemd` units on Linux, or simply put everything in Docker Compose with `restart: unless-stopped` and let one file be the answer
You must be able to reach it	A private mesh network such as Tailscale, rather than opening a port on the router. Reach the machine from a phone without exposing anything to the internet. Where a webhook genuinely has to arrive from outside, a tunnel exposes exactly one endpoint and nothing else

Add one more that people discover late: **something must tell you when a scheduled run did not happen.** A silent agent and a healthy agent look identical from the outside, and the failure that costs the most is the one nobody noticed for three weeks.

> **WHERE TO STOP**
>
> Rung 3 is enough for almost every hiring conversation below principal level, and rung 4 adds little signal for the effort. What changes perception is not sophistication, it is the existence of a URL or a trace proving the thing runs when you are not watching. **An agent that only works when you open your laptop does not count as an agent to anyone who might hire you.**

CHAPTER 5

Context and memory

The context window is a budget, not a storage unit. Treating it as storage is the most common cause of agents that get slower, dumber and more expensive the longer they run.

5.1 Three things people call memory

Kind	Lives in	What it is for
Working context	The prompt of the current call	Everything the model can actually see right now. Finite, billed every call
Session state	Your database, keyed by thread	Message history, intermediate results, checkpoints for resuming
Long-term memory	A vector store or a structured table	Facts, preferences and past solutions, retrieved selectively when relevant

Only the first is visible to the model. The other two exist so that the right slice of them can be loaded into the first at the right moment. That loading decision is the discipline called **context engineering**, and it matters more than prompt wording once a system runs longer than a single exchange.

5.2 The techniques, in order of how often they are needed

Technique	What it does and when to reach for it
Output truncation	Cap what any tool can return. The single highest-value line of code in most agents. One unbounded query otherwise eats the window in a step
Compaction	Replace old turns with a summary once the history passes a threshold. Keep the goal, the decisions and the open questions; drop the transcript
Selective retrieval	Do not paste the knowledge base. Retrieve the few passages that the current step needs. This is what RAG is for
Subagents	Give a noisy subtask its own clean context and return only its conclusion. The parent never sees the mess. The main structural reason to go multi-agent
Externalised notes	Let the agent write findings to a file or table and read them back, instead of carrying everything in the conversation
Prompt caching	Mark the stable prefix (system prompt, tool schemas, reference docs) as cacheable. Large cost reduction for long loops, with no behavioural change

5.3 RAG, long context or fine-tuning

Three answers to "the model does not know my thing", regularly confused with each other.

Approach	Right when	Wrong when
Retrieval (RAG)	The corpus is large, changes, or must be cited and permissioned per user	The corpus is small and stable. You are adding a pipeline for nothing
Long context	The whole corpus fits and you want zero retrieval machinery	It gets resent every call. Cheap to build, expensive to run at volume
Fine-tuning	You need a consistent format, style or classification behaviour, and you have thousands of examples	You need facts. Fine-tuning teaches behaviour, not knowledge, and it is almost never the answer to a knowledge gap

THE MOST COMMON EXPENSIVE MISTAKE

Reaching for fine-tuning to inject knowledge. It is slow, costly, hard to update and it still hallucinates. Retrieval solves the knowledge problem; fine-tuning solves the behaviour problem. Confusing the two burns months.

PART II

The ecosystem map

Twelve categories, and every product in the field belongs to one of them. This part is written to be used as a lookup table rather than read end to end. The structure is durable; the inventory is a September 2026 snapshot and will decay.

CHAPTER 6

The master taxonomy

The reason the landscape feels overwhelming is that public discussion mixes categories. LangChain and Claude get compared. n8n and LangGraph get compared. They are not competitors; they occupy different slots. Here are the slots.

Category	What it actually is	Representative names
1. Models the brain	A hosted or local function from text to text	Claude · GPT · Gemini · Grok · Llama · DeepSeek · Qwen · Mistral · Kimi · GLM
2. Client SDKs	Official HTTP client for one provider's API. You write the loop	`anthropic` · `openai` · `google-genai`
3. Vendor agent SDKs	An agent framework shipped by the model maker	Claude Agent SDK · OpenAI Agents SDK · Google ADK
4. Independent frameworks	Model-agnostic orchestration of the loop	LangGraph · CrewAI · Pydantic AI · smolagents · Mastra · Microsoft Agent Framework
5. Component libraries	Adapters, loaders, retrievers, parsers. Not orchestrators	LangChain · LlamaIndex · Haystack
6. Coding agents	Agents specialised in files, repos and shells	Claude Code · Codex CLI · Gemini CLI · Cursor · Aider · Cline · OpenClaw · Goose
7. Packaged agent products	End-user agents, little or no code	ChatGPT GPTs · Grok Bot · Hermes Agent · Claude Skills and Projects · Copilot Studio
8. Visual orchestrators	Flow drawn on a canvas, integrations pre-solved	n8n · Dify · Make · Zapier · Flowise · Langflow
9. Local inference	Runs open-weight models on your own hardware; models come from the Hugging Face Hub	Ollama · llama.cpp · LM Studio · vLLM · MLX · Tabby · Hugging Face
10. Protocols	Standards for how agents talk to tools, sites, each other	MCP · A2A · WebMCP · OSI · AP2 · ACP
11. Data and retrieval	Embeddings, vector search, reranking, memory stores	pgvector · Qdrant · Weaviate · Chroma · Milvus
12. Evals and observability	Measuring quality; seeing what happened	Langfuse · LangSmith · Braintrust · Arize Phoenix · MLflow

Two supporting categories sit underneath all twelve and are easy to forget until something breaks.

Category	What it is	Names
13. Routers and gateways	One key, many models; fallback, spend caps, logging	OpenRouter · LiteLLM · 9router · CLIProxyAPI
14. Runtime and deployment	Where the process actually lives	Docker · FastAPI · Temporal · Celery · Modal · Railway · Fly.io · Cloud Run · LangGraph Platform · Managed Agents

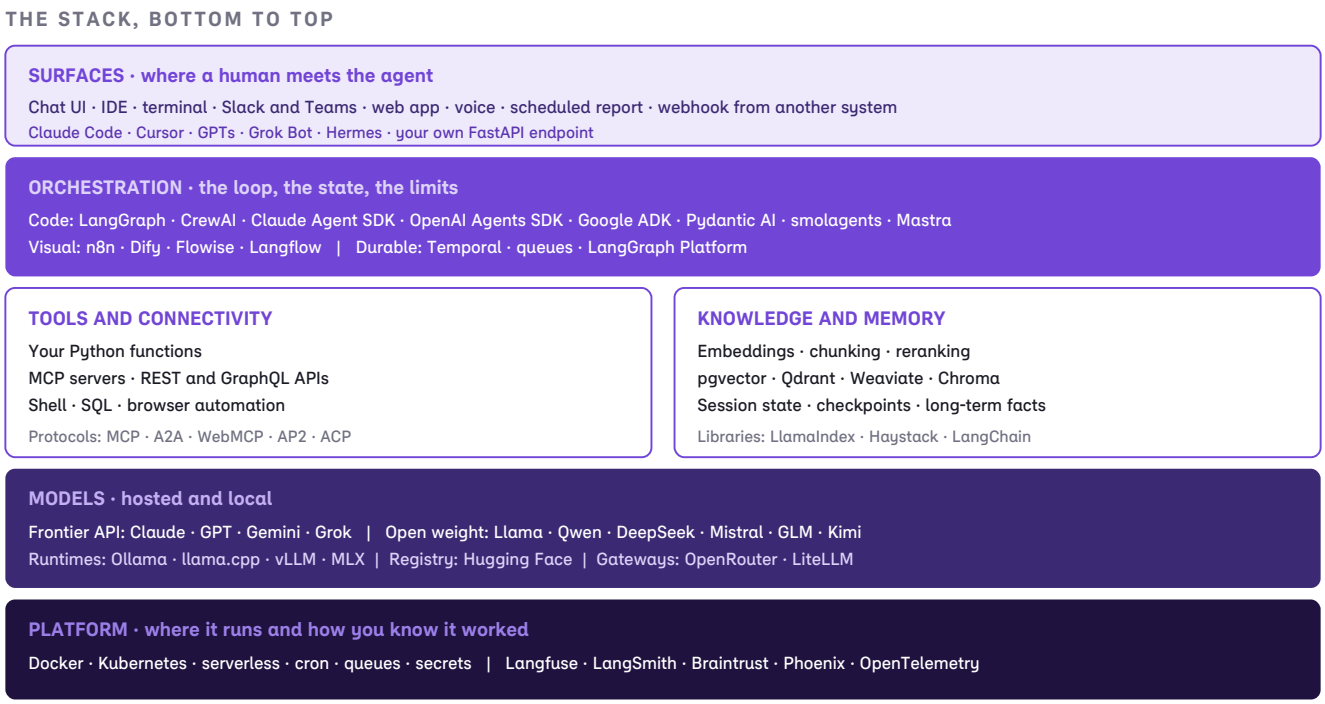


Figure 5. The ecosystem as a stack. Almost every "X vs Y" argument online is comparing two things from different bands.

CHAPTER 7

Models and providers

Model choice matters far less than most beginners assume and far more than most intermediates assume. It matters little for learning architecture and a lot for cost, latency and tool-calling reliability in production.

7.1 The four tiers that actually exist

Tier	What it is for	Trade
Frontier	Long-horizon agentic work, planning, hard reasoning, code at scale	Highest capability, highest price, highest latency
Workhorse	The default for almost everything: orchestration, tool use, drafting	The best capability-per-euro point. Where most production traffic belongs
Fast and cheap	Classification, extraction, routing, summarising, guardrails	Near-frontier on narrow tasks at a fraction of the cost. Under-used
Open weight	Privacy, zero marginal cost, offline, full control	Weaker multi-step tool calling. Great as the cheap tier of a router

7.2 The Anthropic lineup, September 2026

Included in full because it is the stack this manual assumes. The pattern of a frontier tier, a workhorse tier and a fast tier repeats across every provider.

Model	Context	Position
Claude Fable 5.1	1M	Frontier. Demanding reasoning and long-horizon agentic work. Most expensive
Claude Opus 5	1M	Complex agentic coding and enterprise workloads. The recommended starting point
Claude Sonnet 5	1M	Balance of speed and intelligence. The sensible default for agent loops
Claude Haiku 4.5	200K	Fastest, cheapest, near-frontier on narrow tasks. The classification tier

7.3 The wider provider landscape

Provider	Family	Where it is genuinely strong
Anthropic	Claude	Agentic coding, tool use reliability, long-horizon tasks, MCP. The reference stack for agent work
OpenAI	GPT	Broadest ecosystem, largest third-party integration surface, strong multimodal, the de facto API format

Provider	Family	Where it is genuinely strong
Google	Gemini	Very large context, aggressive free tiers, tight Google Cloud and Workspace integration
xAI	Grok	Real-time data access, fast iteration, the Grok Bot always-on agent product
Meta	Llama	The default open-weight baseline; enormous fine-tune and tooling ecosystem
Mistral	Mistral, Devstral	European hosting and data residency, efficient small models, Apache-licensed tooling
DeepSeek	DeepSeek V-series	Reasoning quality per euro. Frequently the price-performance leader
Alibaba	Qwen, Qwen Coder	The strongest open-weight family for local coding and agent use in most rankings
Moonshot	Kimi	Long context and agentic coding; Kimi Code CLI pushes large parallel agent swarms
Zhipu	GLM	Strong open-weight coding models, widely used as a local Claude Code substitute

ON VERSION NUMBERS

Every name in the table above is stable; every version number attached to it is not. Model releases arrive monthly. Build the habit of checking the provider's own model page before making a cost or capability decision, and never hard-code a model ID in more than one place in a codebase.

7.4 What actually differs between models, in practice

Benchmarks are a poor guide to agent work. These five properties are what you will feel.

- **Tool-calling reliability.** Does it invent parameters? Does it forget it already called something? Does it give up and answer from memory? This is the single biggest practical gap between frontier and small models.
- **Instruction adherence under long context.** Does the system prompt still govern behaviour at turn forty?
- **Recovery from tool errors.** Does a failed call cause a useful retry or a spiral?
- **Cost and latency at your token profile.** Input-heavy agent loops price very differently from output-heavy generation.
- **Structured output fidelity.** Does it reliably return schema-valid JSON without repair passes?

7.5 Where the models actually stand

Rankings are the most requested and least durable content in this manual. The chart below is a single snapshot from a single source, and the only responsible way to use it is to read the shape rather than the ordering. The shape has been stable for two years and will stay stable; the ordering changes monthly.

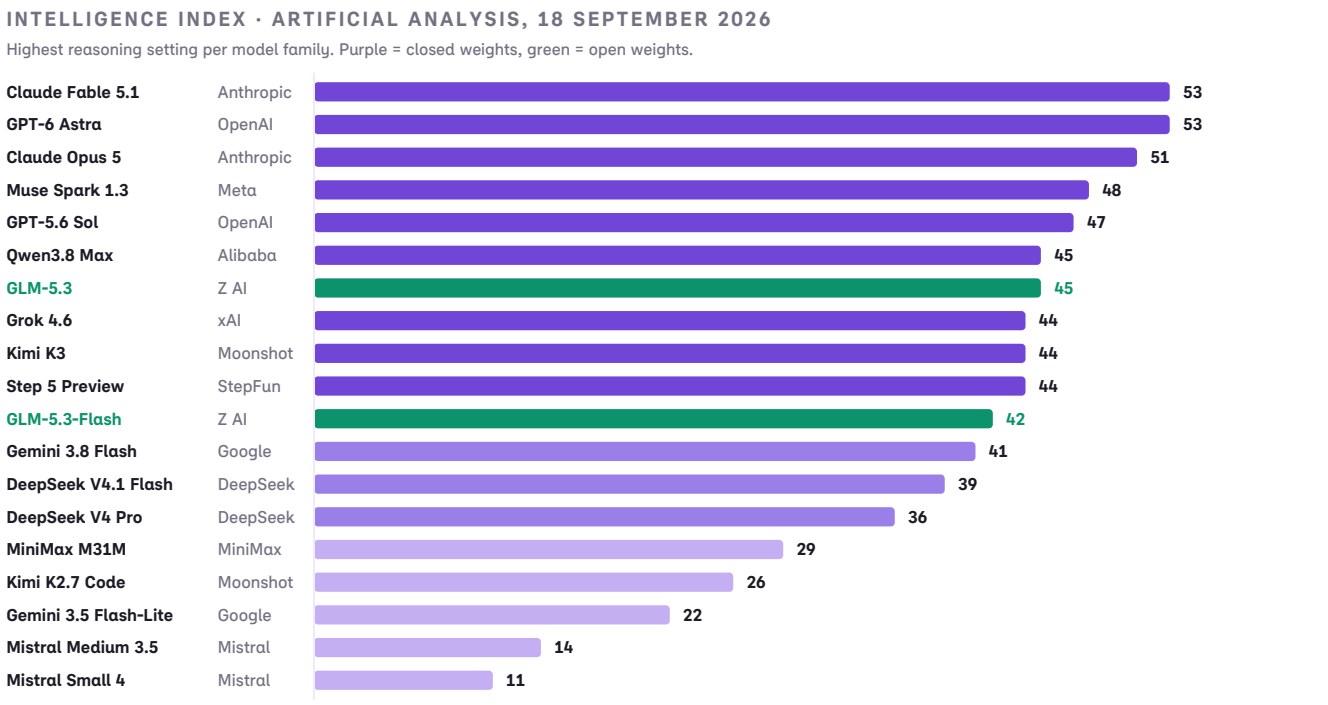


Figure 6. Intelligence Index, Artificial Analysis, 18 September 2026. The leaderboard lists a separate entry per reasoning-effort setting; this takes the highest per family.

FOUR CAVEATS, AND THEY MATTER MORE THAN THE NUMBERS

One source, one day. A different aggregator using a different normalisation produces a visibly different ordering, including a different model at the top. Any single leaderboard is one opinion with a methodology attached.

Family coverage is uneven. This snapshot surfaced Google's Flash tiers rather than its top Pro tier, so Gemini's position here understates the family. Never conclude a provider is weak from one leaderboard row.

Benchmarks correlate weakly with agent reliability. Nothing in this index measures whether a model invents tool parameters on step nine, which is the property that actually decides whether your agent works.

Reasoning effort is a setting, not a model. The same model appears several times at different costs and scores. The setting you choose changes your bill more than the model you choose.

7.6 The chart that should drive decisions

Ranking by intelligence alone answers a question almost nobody has. The decision you actually make is what does a point of capability cost me at this volume, and that is a two-dimensional question.

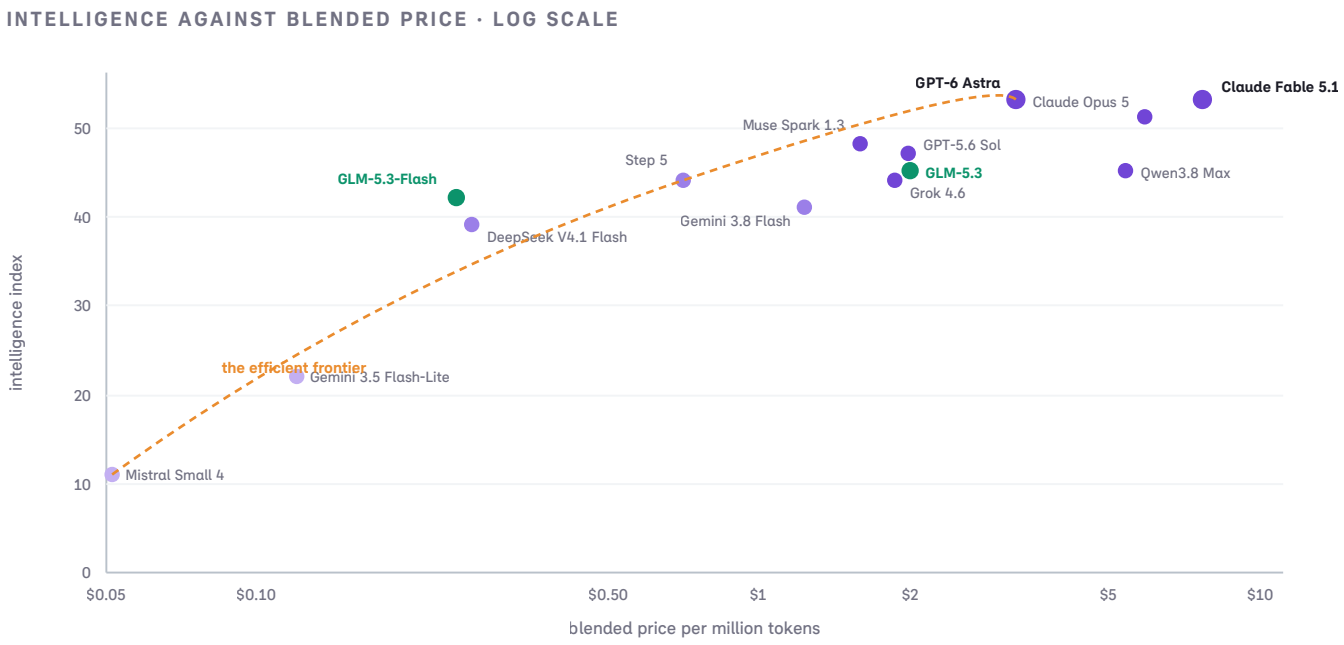


Figure 7. The same data plotted against price. Everything above and left of the dashed line is a bargain; everything below and right is paying for something this index does not measure.

Three things fall out of that picture, and all three are decisions rather than trivia.

- **The curve flattens hard.** Going from roughly 40 to roughly 53 on the index multiplies the price by something like six. For classification, extraction and summarising, that spend buys almost nothing, which is the whole argument for tiering in Chapter 20.
- **The best open-weight model sits inside the top band.** One point below a major proprietary model, at a fraction of the price, with the option of self-hosting. That was not true two years ago and it changes what "you need a frontier API" means.
- **Price differences between the top models are larger than capability differences.** Two models can tie on the index and differ by a factor of two on cost. At agent volumes, that is the entire decision.

CHAPTER 8

Agent frameworks

A framework buys you the bookkeeping around the loop. The correct question is never "which framework is best" but "which piece of bookkeeping do I need badly enough to take on a dependency".

8.1 The field

Framework	Language	What it really is, and its ceiling
LangGraph CORE	Python, TS	A state machine for agents: nodes, conditional edges, a typed `State` object. Buys checkpointing to Postgres, pause-and-resume, human-in-the-loop, time travel and streaming. The most common name in job adverts. Ceiling: real learning curve, overkill for simple flows
CrewAI	Python	A high-level "team with roles" abstraction: Agents, Tasks, a sequential or hierarchical Process. Fastest path to a working multi-agent demo. Ceiling: limited fine control; unusual flows mean fighting the abstraction
Claude Agent SDK CORE	Python, TS	The engine behind Claude Code, as a library. Ships the loop, built-in file and shell and web tools, subagents, hooks, permissions, sessions with resume and fork, MCP, and skills loaded from `.claude/`. Best-in-class for agents that touch files, repos and systems
OpenAI Agents SDK	Python, TS	Lightweight loop with handoffs, guardrails and tracing. Provider-agnostic despite the branding, with support for a hundred-plus models. Good default if the shop is OpenAI-centred
Google ADK	Python, Java	Agent Development Kit, tuned for Gemini and Vertex AI. Natural choice inside Google Cloud, less so outside it
Microsoft Agent Framework	.NET, Python	The merger of AutoGen and Semantic Kernel into one supported product. The answer for .NET and Azure shops. AutoGen itself is now in maintenance
Pydantic AI	Python	Type-safe agents built on Pydantic. Strong validation, dependency injection, structured outputs. Loved by teams who already think in types
smolagents	Python	Hugging Face's minimal, code-first agent library. Agents write Python to act rather than emitting JSON tool calls. Excellent for learning the mechanism
Mastra	TypeScript	The mature TypeScript option, v1.0 since January 2026. Workflows, agents, RAG and evals in one. Relevant if the team lives in Node
AutoGen	Python	Historically important for multi-agent conversation patterns. Now maintenance-only; folded into Microsoft Agent Framework

8.2 Component libraries, which are not frameworks

Library	What it is for
LangChain	A box of adapters: model clients, document loaders, retrievers, output parsers, vector store wrappers. Its reputation for over-abstraction was earned in early versions; modern practice is to use pieces of it rather than build on it wholesale

Library	What it is for
LlamaIndex	Retrieval-first. Indexing strategies, query engines, document-heavy pipelines. The strongest option when the problem is genuinely "search my documents well"
Haystack	Pipeline-based RAG and document processing, production-hardened for years. Favoured where the workload is search and extraction rather than open-ended agency

THE ADOPTION RULE

Write the loop by hand first. It is forty lines. Adopt a framework the day you feel a specific pain: "this must survive a restart", "this must stop and ask a human", "I cannot see why it did that". A framework adopted before the pain teaches you its API instead of teaching you agents, and its API will be obsolete before the concepts are.

CHAPTER 9

Coding agents and CLIs

The most crowded category in the field, and the one where the distinction between a tool you build with and a tool you build on is most often missed. A coding agent can play three roles: your workshop, your runtime, or your engine.

9.1 The three roles a coding agent can play

Role	What happens	What you end up with
Workshop	It writes your agent's code, tools and Dockerfile	An agent that does not depend on it at all. Runs on Python and an API key
Runtime	Headless invocation from cron or CI, for example `claude -p "..." --output-format json`	The coding agent is the agent. Under-appreciated, and the fastest autonomous agent you can deploy
Engine	You import its SDK into your own program	Your agent, reusing its loop, context management, subagents, hooks and permissions

9.2 Subscription and commercial tools

Tool	Maker	Character
Claude Code CORE	Anthropic	Terminal-first agent with very large context and strong token efficiency. Skills, subagents, hooks, MCP, plugins. The reference implementation, and the SDK behind it is a product in its own right
Codex CLI	OpenAI	CLI plus desktop, with cloud sandbox execution for running work remotely
Cursor	Cursor	VS Code fork. Largest community, most polished UX, strong multi-file editing
Windsurf	Windsurf	IDE with persistent-context "Flows"
Antigravity	Google	Agent-first IDE running parallel agents with built-in browser testing
Amp	Sourcegraph	CLI and IDE with an autonomous deep-research mode over large codebases
Mistral Vibe	Mistral	Apache-licensed CLI, paid models. The European option
Devin	Cognition	Hosted autonomous software engineer. Ticket in, pull request out. Expensive, narrow, occasionally spectacular

9.3 Free tiers

Tool	Note
Gemini CLI	Google. Very generous free daily request allowance. The best zero-cost starting point
GitHub Copilot CLI	GitHub. Natural fit if the workflow already lives in GitHub

Tool	Note
Amazon Q Developer	AWS-optimised; understands AWS resources natively
Kiro	Amazon. Spec-driven: writes requirements first, then implements against them
Qwen Code	Alibaba. Free API access, strong for its price of zero

9.4 Open source, bring your own key

Tool	What distinguishes it
OpenClaw	Community-built open-source alternative to Claude Code, started by Peter Steinberger. Self-hosted, forkable and fully auditable: a community agentic loop over the same APIs. No subscription, you pay only tokens, and it has become a common gateway into the Chinese open-weight model ecosystem. Trade: you own the security review and the setup complexity that a managed tool absorbs for you
OpenCode	Supports seventy-five-plus model providers. The most provider-agnostic option
Aider	Git-native, auto-commits each change. Superb for disciplined, reviewable edits
Cline	The most adopted VS Code agent extension by install count
Continue.dev	VS Code and JetBrains, highly configurable
Goose	Block. Apache 2.0, native MCP integration from the start
Roo Code	Cline fork focused on reliability across large multi-file changes
Zed	Rust-native editor with agent panel. The performance option
iFlow	Subagents with explicitly controlled permissions
Kimi Code CLI	Moonshot. Pushes very large parallel agent swarms

> **HOW TO READ THIS CATEGORY FOR YOUR CAREER**
>
> Using a coding agent well is a productivity skill, not an engineering credential. Almost every candidate now claims it. What differentiates is the engine role: importing an agent SDK, defining your own tools, wiring hooks and permissions, and shipping something that runs unattended. That is the same skill the rest of this manual is about, applied to a strong starting point.

CHAPTER 10

Packaged agent products

Agents sold as products rather than as libraries. Useful to know, useful to use, and near-worthless as evidence of engineering ability. Knowing exactly why is itself a signal of judgement.

Product	Maker	What it is
ChatGPT GPTs	OpenAI	A saved system prompt plus optional files and actions, shareable in a store. Genuinely useful personal tooling. Engineering signal: close to zero, because a reviewer cannot distinguish it from a bookmarked prompt
Grok Bot	xAI	Launched in beta August 2026. A team of always-on cloud agents with their own compute that can sign into applications and work across tools and websites without APIs or special protocols. They keep working while you are offline, learn a workflow by being shown it once, coordinate in parallel with each other, and retain context over time. Marketed as hiring a colleague rather than configuring an automation. Available to SuperGrok subscribers and to Cursor Pro and Teams tiers, on desktop and iOS
Hermes Agent	Nous Research	Open-source, MIT-licensed, self-hosted general agent. Runs as a desktop app or terminal tool on macOS, Windows and Linux, with optional cloud hosting. Reaches the user through Telegram, Discord, Slack, WhatsApp, Signal, email and CLI. Its distinguishing claim is persistent memory that auto-generates reusable skills from problems it has already solved, plus isolated subagents, scheduling, browsing, vision and sandboxed code execution across several backends. The most interesting open answer to "a personal agent that lives on my own machine"
Claude Projects and Skills	Anthropic	Projects hold shared context and documents; Skills are folders of instructions and scripts that load on demand and travel across Claude Code, the apps and the Agent SDK. Skills sit closer to engineering than GPTs do, because they are files you can version
Managed Agents	Anthropic	A hosted REST API where Anthropic runs the agent and its sandbox. For long-running or asynchronous agents when you do not want to operate session infrastructure. A separate product from the Agent SDK
Copilot Studio	Microsoft	Low-code agent builder wired into Microsoft 365 and Power Platform. Very common in large enterprises, invisible everywhere else
Agentforce	Salesforce	Agents embedded in the Salesforce data and permission model. Relevant only inside that world

10.1 The pattern worth noticing

Grok Bot and Hermes Agent approach the same goal from opposite directions, and the contrast is instructive about where the field is heading.

Grok Bot: managed and opaque

- Runs on the vendor's compute, always on
- Acts through the UI of applications rather than their APIs, so it needs no integration work
- Learns by demonstration rather than configuration
- You cannot inspect the loop, self-host it, or move it

Hermes Agent: open and yours

- Runs on your machine, under a permissive licence
- Reaches you through the messaging surfaces you already use
- Persists memory and turns past solutions into reusable skills
- You own the operations, the security review and the uptime

The same fork appears at every layer of this manual: a managed product that removes work and takes control with it, against an open component that gives you control and hands you the operations bill. Neither is the right answer in general. Knowing which one a given situation deserves is the judgement being tested.

CHAPTER 11

Visual and low-code orchestrators

The category most often dismissed by engineers and most often present in job adverts. Its superpower is not the AI node; it is the hundreds of pre-solved OAuth integrations, the triggers, the retry behaviour and the execution log you did not have to build.

Tool	Model	Character
n8n CORE	Open source, self-hostable	Around four hundred connectors, webhook and cron triggers, code nodes for escape hatches, a built-in AI agent node. Self-hosting keeps data on your infrastructure. Appears by name in AI automation job adverts
Dify	Open source	The most starred project in the category. Visual agent and RAG builder with a strong app-publishing layer
Flowise	Open source	Node canvas over LangChain primitives. Good for prototyping chains visually
Langflow	Open source	Similar canvas approach; exports to Python
Make	SaaS	Deep integration catalogue, strong visual branching. No self-hosting
Zapier	SaaS	The widest integration catalogue in existence, the shallowest logic. Business-user oriented

11.1 The real ceiling of a visual tool

These are the four walls you hit, and hitting them is the correct trigger to move that workload into code. None of them are reasons to avoid visual tools in the first place.

- **Version control.** The workflow is a large JSON blob. A diff is unreadable, code review is impossible, and merge conflicts are unresolvable in practice.
- **Testing.** There is no natural unit test. You cannot assert that a change to node fourteen did not break node three.
- **State across executions.** Anything beyond simple persistence means bolting on an external store, at which point you are programming anyway.
- **Complex control flow.** Nested loops with dynamic exit conditions, selective retry of one failed step, and fan-out with partial failure all become fights with the canvas.

> **A CORRECTION WORTH MAKING TO YOURSELF**
>
> "Visual tools are for non-engineers" is wrong and expensive. The right framing is that a visual orchestrator is the correct tool when the difficulty of the problem lives in integration, and code is the correct tool when it lives in logic. Rebuilding four hundred OAuth flows in Python to feel like a real engineer is not engineering judgement, it is its opposite.

CHAPTER 12

Local inference

Running open-weight models on your own hardware. Strategically valuable, frequently misunderstood as an all-or-nothing choice against frontier APIs when it is really one tier of a routing decision.

12.1 Runtimes

Runtime	Best for
Ollama	The easiest path. One command to pull and serve a model, with an OpenAI-compatible endpoint so existing code works unchanged. The default for a personal lab
llama.cpp	The engine underneath much of the ecosystem. Maximum control over quantisation and memory. Where you go when Ollama's defaults are not enough
LM Studio	Desktop GUI. Excellent for trying models quickly and comparing them side by side
vLLM	Serving at throughput, with batching and paged attention. The production answer for self-hosted inference on GPUs
Tabby	Self-hosted code completion, as a private alternative to hosted autocomplete
MLX	Apple's array framework for Apple Silicon. Where you go when you want the last thirty per cent of performance out of a Mac, or want to fine-tune locally. Lower level than Ollama, worth knowing if the lab lives on Apple hardware

12.2 Open-weight families worth knowing

Qwen and its coder variants, DeepSeek, Llama, Mistral and Devstral, GLM from Zhipu, Kimi from Moonshot, MiniMax, and the open-weight models released by the large labs themselves. Rankings shuffle every few months; the families are stable.

12.3 Hugging Face: the layer underneath all of this

Every runtime above pulls its weights from somewhere, and that somewhere is almost always Hugging Face. It is not a single product, which is why it is easy to under-rate. It is six things at once, and at least three of them belong in a working agent stack.

What it is	Why it matters to you
The Hub	The registry for open-weight models, datasets and evaluation results. When Ollama pulls a model, this is the origin. Model cards carry licence, size, quantisation and intended use, and reading them is a real skill
`transformers`	The reference library for running and fine-tuning open models in Python. The common language almost all open-model tutorials and papers are written in
`sentence-transformers`	How most people generate embeddings locally. Directly relevant the moment your retrieval layer should not send text to a provider

What it is	Why it matters to you
Inference Providers and TGI	Hosted endpoints for open models, and Text Generation Inference for serving them yourself. The middle ground between a frontier API and your own GPU
Spaces	Free hosting for small demos, typically Gradio or Streamlit. The cheapest way to put a working demo of a portfolio project behind a public URL
Datasets and leaderboards	Where evaluation sets live, and where open-model rankings are published. Useful for building an eval set without starting from zero
smolagents and the Agents Course	Their minimal agent library and the best free structured course on agents. Both covered elsewhere in this manual

HOW TO PLACE IT MENTALLY

Hugging Face is to open models roughly what GitHub is to open source and what npm is to JavaScript: the distribution layer plus the tooling that grew around it. You will interact with it whether or not you notice, because Ollama, vLLM, LM Studio and llama.cpp all reach into it. **Noticing is the difference between using open models and understanding them.**

12.4 Sizing the machine

Two numbers decide what a local setup can do, and they do different jobs. Neither is the processor name on the box.

Number	What it decides
Memory unified memory on Apple Silicon, VRAM on a discrete GPU	**Which models fit at all.** A hard wall: below it the model will not load, above it nothing improves. Rough arithmetic for a 4-bit quantised model: multiply the parameter count in billions by about 0.6 to get gigabytes of weights, then multiply by roughly 1.6 to leave room for the context window and everything else the machine is running. A 30B model in Q4 wants about 30 GB of headroom, a 70B about 70 GB
Memory bandwidth	**How fast tokens come out** once the model fits. Generation is memory-bound, so at equal capacity roughly double the bandwidth means roughly double the tokens per second. This is the difference between an agent that answers and an agent you watch thinking

Three practical consequences follow, and they are the ones people learn the expensive way.

- **Memory is not upgradable on Apple Silicon.** It is soldered. You are buying today the machine you will want in three years, so the memory decision is the only irreversible one.
- **Quantisation is the lever, not the processor.** Q4 versus Q8 changes the memory requirement by roughly half, at a quality cost that is small for classification and extraction and noticeable for reasoning. Match the quantisation to the job rather than always taking the largest model that fits.
- **Internal storage is the worst value in any machine.** Weights load perfectly well from an external NVMe drive over a fast bus. Buy the minimum internal disk and keep the model library outside it.

THE SIZING MISTAKE THAT WASTES THE MOST MONEY

Buying for the largest model you can imagine running rather than the models you will actually use daily. A 70B model at home is impressive once and then mostly idle, because for the tasks a local tier should handle, from Chapter 20, a well-chosen model in the 7B to 30B range is both sufficient and several times faster. **Spend the difference on API credits and on an always-on machine you never turn off.**

12.5 Choosing the machine

The principles above turn into a purchase decision. On Apple Silicon the two numbers are unified memory and memory bandwidth; on a discrete GPU they are VRAM and the same bandwidth question. What follows is the desktop Apple line as of September 2026, because an always-on low-power desktop is the usual shape of a personal lab. Prices are starting configurations and move constantly.

ndash;32nbsp;GBndash;64nbsp;GBnbsp;GB/sndash;128nbsp;GBndash;614nbsp;GB/sndash;512nbsp;GBnbsp;TB/s

Machine	From	Memory	Bandwidth	What it comfortably runs
Mac mini M6	$899	16	16 – 32 GB	16 – 32 GB
Mac mini M5 Pro	mid tier	24	24 – 64 GB	24 – 64 GB
Mac Studio M5 Max	$2,500	36	36 – 128 GB	36 – 128 GB
Mac Studio M5 Ultra	$5,500	96	96 – 512 GB	96 – 512 GB

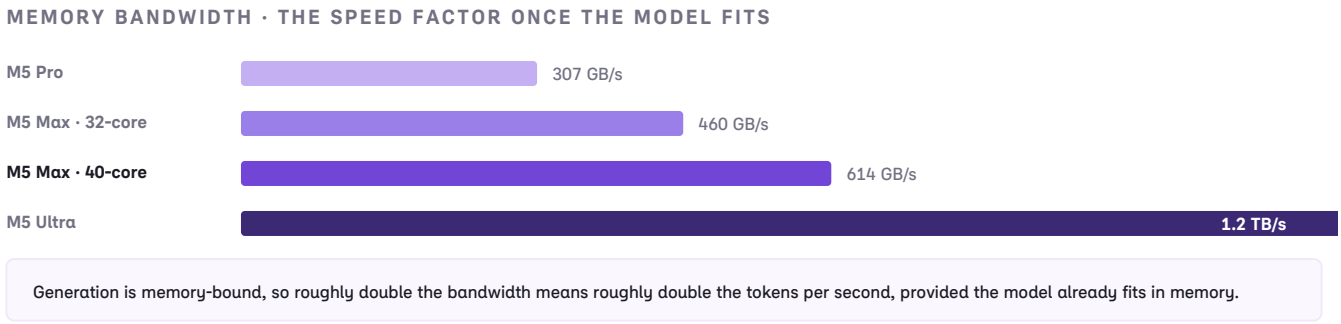


Figure 8. Bandwidth on a common scale. It decides speed; memory decides whether the model loads at all.

What actually fits, and how fast

Model size	Memory needed	Realistic throughput
7B to 13B, Q4	24 GB	Comfortably interactive on any tier. 50 to 75 tokens/s on a Max
30B, Q4	32 to 48 GB	The practical sweet spot for a local tier: fast enough to sit inside an agent loop
70B, Q4	64 GB, tight	Roughly 4 to 6 tokens/s on an M5 Pro, 8 to 12 on an M5 Max. Usable for batch work overnight, painful interactively
70B, Q5	128 GB	Only on the larger Studio configurations. 12 to 18 tokens/s

THE DECISION, STATED PLAINLY

Those throughput figures are the whole argument. **A 70B model at home generates a handful of tokens per second, which is unusable inside an agent loop that makes thirty calls.** The machine worth buying is the one that runs a 30B model quickly and stays on all year, not the one that technically loads a 70B model you will run twice. Put the saved money into API credits for the steps that genuinely need frontier capability, and remember that memory is soldered, so it is the one decision you cannot revisit.

BEFORE BUYING ANYTHING

Every figure on this page decays: prices change, configurations get renamed, and published bandwidth numbers for the same chip disagree between reputable outlets. **Check the manufacturer's own specification page before ordering**, and compare the exact configuration rather than the family name. Also price the refurbished previous generation: a one-generation-old chip with double the memory usually beats a new chip with half of it, for this workload specifically.

12.6 The honest assessment

Where local genuinely wins

- Privacy and data residency. Nothing leaves the machine, which is the difference between "I cannot touch that data" and "I can"
- Zero marginal cost on high-volume, narrow tasks
- Low latency for short tasks with no network round trip
- Works offline, and runs continuously on hardware you already pay for
- Rare enough as a demonstrated skill to be a genuine differentiator

Where local genuinely loses

- Multi-step tool-calling reliability. This is the wall, and you meet it around step three or four
- Long-context instruction adherence
- Recovery from unexpected tool errors
- Throughput on consumer hardware once concurrency appears
- Operational burden: you are now the one who maintains inference

THE PROFESSIONAL PATTERN

Do not treat local as a replacement. Treat it as **the cheap tier of a hybrid router**: local for classification, extraction, summarising, embeddings and filtering; frontier API for planning, tool orchestration and final output. Then measure the split, and be able to say what it cost and what it did to accuracy. That sentence, with real numbers attached, is worth more in an interview than any framework you can name. Chapter 20 covers the routing design in full.

CHAPTER 13

Protocols and standards

The most strategically important chapter in Part II. Protocols are the only part of the ecosystem explicitly designed to survive the death of individual products, which makes them the highest-return thing to learn.

13.1 The 2026 protocol stack

Protocol	Layer	Governance and status
MCP CORE	Agent to tools and data	Model Context Protocol. Anthropic-led with an open process. The closest thing the agent world has to a settled standard: thousands of servers across databases, SaaS products and developer tools, and support from all major clients. **Learn this one properly**
A2A	Agent to agent	Agent-to-Agent Protocol. Donated by Google to the Linux Foundation; IBM's competing protocol merged into it. Version 1.0 shipped April 2026 with signed Agent Cards for verified identity. Over 150 organisations, SDKs in five languages, native support in the major frameworks
WebMCP	Agent to websites	W3C standards track, developed jointly by Google and Microsoft. Declarative APIs for forms and JavaScript APIs for dynamic functionality, so agents can use a site through a structured interface instead of scraping pixels. Early preview in Chrome Canary as of February 2026. Not yet a deployed norm
OSI	Business semantics	Open Semantic Interchange. Apache 2.0, launched by Snowflake with BI and data tooling partners; specification live January 2026. Ensures agents use consistent definitions of metrics and dimensions across catalogues and BI tools. Directly relevant to anyone whose agents touch analytics
AP2	Agent payments	Agent Payments Protocol. Cryptographic mandates: user-signed, scoped spending authorisations with proof trails
ACP	In-conversation checkout	Agentic Commerce Protocol, from OpenAI and Stripe. The payments layer is crowded and consolidation is expected

13.2 Conventions that are not protocols but behave like them

Convention	What it does
The OpenAI API shape	Not a standard, but universally copied. Ollama, vLLM, OpenRouter, Groq and most local servers speak it. Learning it once unlocks nearly every endpoint in the field
`agents.md`	A file in a repository telling coding agents how to work in it: commands, conventions, boundaries
`llms.txt`	An agent-oriented content map for a website, analogous in spirit to `robots.txt`
OpenTelemetry GenAI	Semantic conventions for LLM spans, so traces are portable between observability vendors
Signed Agent Cards, Web Bot Auth	Emerging identity and trust layer: proving which agent is acting and on whose authority

13.3 Why MCP deserves disproportionate attention

Write a capability once as an MCP server and every MCP-speaking client can use it: Claude Code, Cursor, n8n, LangGraph, the OpenAI stack, and whatever replaces them. It is the closest thing in this field to a portable skill, and it is the direct structural answer to the fear of learning something that will not transfer to the next employer.

There is also a career asymmetry worth exploiting. Almost everyone in the market has consumed MCP servers. Far fewer have written and published one. The gap between those two sentences on a CV is large and the work to cross it is small.

> **THE SECURITY SIDE NOBODY MENTIONS IN THE TUTORIALS**
>
> An MCP server is code you are granting into your agent's action space, and its tool descriptions enter the model's context. A malicious or compromised server can therefore inject instructions, not just return bad data. Treat third-party MCP servers with the same suspicion as any dependency with credentials: read the source, pin the version, grant the narrowest scope, and prefer ones you or your organisation wrote.

CHAPTER 14

Data, retrieval and memory

Retrieval is not a product you buy, it is a pipeline you tune. Most disappointing RAG systems fail at chunking and ranking, not at the choice of vector database, and yet the database is what people argue about.

14.1 The pipeline, stage by stage

Stage	What it does	Where it goes wrong
Ingestion	Get text out of PDFs, docs, tickets, web pages	Scanned PDFs, tables destroyed by naive extraction, lost heading hierarchy. Fix this first; nothing downstream recovers from bad extraction
Chunking	Split into retrievable units	Fixed-size splits that cut a table in half or separate a clause from its heading. Prefer structure-aware splitting and overlap
Embedding	Turn chunks into vectors	Mismatched models between indexing and querying. Forgetting that re-embedding the corpus is required to change model
Storage and index	Store vectors plus metadata	Ignoring metadata filters, which are usually more valuable than the vector similarity itself
Retrieval	Find candidates for the query	Pure semantic search. Hybrid search, combining keyword and vector, beats either alone almost every time
Reranking	Reorder candidates with a stronger model	Skipping it. A cross-encoder reranker over twenty candidates is often the single largest quality gain available
Assembly	Build the final context	Pasting everything. Retrieve broadly, rerank hard, inject narrowly

14.2 Stores

Store	When it is the right call
pgvector CORE	Postgres extension. Your relational data, your metadata, your permissions and your vectors in one database with one backup story. The correct default for almost everyone, and the reason most teams never need a dedicated vector database
Qdrant	Purpose-built, Rust, excellent filtering. The usual step up when pgvector's scale or latency stops being enough
Weaviate	Hybrid search and modular vectorisers built in
Chroma	Simplest local developer experience. Great for prototypes, less so for production
Milvus	Very large scale, distributed deployments
Elasticsearch / OpenSearch	Already in the stack and now vector-capable. Often the pragmatic answer in an existing enterprise

THE DEFAULT WORTH DEFENDING

Start with Postgres and pgvector. One database holds your state, your history, your permissions and your embeddings, with one connection string, one backup and one place to join across all of it. Introduce a dedicated vector database when you have a measured reason, not because a tutorial used one.

CHAPTER 15

Evaluation and observability

This is the frontier between a demo and a product, and it is where the smallest proportion of practitioners actually operate. If you invest disproportionately anywhere in this manual, invest here.

15.1 Two different jobs

Discipline	Question it answers	Shape
Observability	What did the agent do, and why, on that specific run?	Traces of every model call, tool call, token count and cost, searchable after the fact
Evaluation	Did my change make the system better or worse, on average, across cases I care about?	A fixed dataset, a scoring function, and a number that moves

Observability without evaluation means you can explain failures but cannot prevent them. Evaluation without observability means you know the score dropped but not where. Production systems need both, and a personal project with both is immediately distinguishable from one without.

15.2 Building an eval set that is actually useful

1. **Collect real cases, not invented ones.** Eighty to two hundred is plenty. Invented cases test what you imagined, which is exactly the set of cases already handled.
2. **Include the failures.** Every time the agent does something wrong, that run becomes a permanent test case. This is how the set earns its value over time.
3. **Score at the right level.** Some tasks have a deterministic answer and can be checked with an assertion. Others need a rubric and a model-as-judge. Others only a human can score. Use all three rather than forcing one.
4. **Measure trajectory, not just output.** For agents, whether the correct tools were called in a sensible order often matters more than the final text.
5. **Run it in CI.** A prompt change that drops the score should fail the build exactly like a failing unit test. The moment this exists, you are doing engineering.

15.3 The tools

Tool	Character
Langfuse CORE	Open source and self-hostable with Docker. Tracing, prompt management, datasets, evaluation runs, cost tracking. Framework-agnostic. The obvious choice for a personal lab: it costs nothing and running it yourself is itself a demonstrable skill
LangSmith	LangChain's hosted platform. Framework-agnostic in principle, deepest with LangGraph. The most common commercial choice

Tool	Character
Braintrust	Evaluation-first. Strongest workflow for iterating on prompts against a dataset
Arize Phoenix	Open source, OpenTelemetry-native, strong on drift and root-cause analysis
MLflow	Extended from classical ML into LLM tracing. Natural if MLflow is already in the organisation
OpenTelemetry GenAI	The vendor-neutral conventions underneath. Instrument to these and you can change vendor without reinstrumenting

THE DIFFERENTIATOR, STATED PLAINLY

Almost every candidate can show an agent that works on a happy path. Very few can open a dashboard, point at a distribution of two hundred runs, and say: **"this version scores 0.84 against this set, the previous one scored 0.71, the regression came from tool timeouts, and here is the trace."** That sentence is what a senior interviewer is listening for, and nothing else in this manual substitutes for it.

CHAPTER 16

Routers, gateways and runtime

The plumbing. Unglamorous, rarely discussed in tutorials, and responsible for most of the difference between a system that survives contact with reality and one that does not.

16.1 Routers and gateways

Tool	What it buys you
OpenRouter	One key and one API shape across a hundred-plus models from many providers. The cheapest way to compare models honestly, and a free fallback mechanism
LiteLLM	A proxy you host yourself. Normalises providers, adds spend caps per key, retries, fallback chains and logging. The usual answer when a team needs central control of model spend
9router, CLIProxyAPI	Connect many providers into coding tools, or wrap a free endpoint in an OpenAI-compatible shape

16.2 Runtime and deployment

Piece	Role
Docker and Compose CORE	The unit of reproducible deployment. If your whole lab does not come up with one command, it is not deployable
FastAPI	The standard way to put an agent behind an HTTP endpoint, with async and streaming
Celery or RQ with Redis	Background jobs, retries, scheduling. The pragmatic Python default
Temporal	Durable execution: a workflow survives a process crash and resumes exactly where it stopped. Increasingly the serious answer for long-running agents, and a differentiating name to know
LangGraph Platform	Managed hosting for LangGraph agents with persistence and scaling handled
Modal, Railway, Fly.io, Render	Low-friction hosting between a VPS and full cloud. Usually the right rung for a portfolio project
Cloud Run, ECS, Lambda	The hyperscaler equivalents. Necessary only once scale or corporate policy requires them
Cloudflare Tunnel or ngrok	Expose a webhook endpoint from a machine behind a home network, without opening a port

> **DURABLE EXECUTION IS THE CONCEPT, NOT THE PRODUCT**
>
> Frameworks without built-in durability push the problem onto you, and teams end up bolting on Temporal or Redis just to achieve basic reliability. Whether you adopt Temporal or implement checkpoints yourself matters less than understanding why a long-running agent needs to be resumable at all: because a thirty-minute run that dies at minute twenty-eight and restarts from zero is not a system anyone can operate.

CHAPTER 17

Enterprise AI platforms

The managed clouds. Deliberately last in Part II, because studying them before the fundamentals is the most common way to spend three months learning a console instead of a craft.

Platform	What it is
AWS Bedrock	Managed access to many model families, plus Knowledge Bases, Guardrails and Agents, all inside AWS IAM
Azure AI Foundry	Microsoft's model and agent platform. In European enterprise, statistically the one you are most likely to meet
Google Vertex AI	Gemini plus tuning, grounding, and Agent Engine for deployment
Databricks	Agents next to the lakehouse, with Unity Catalog governance and a strong evaluation story
Snowflake Cortex	Models executed inside the warehouse, so the data never leaves

17.1 How to hold this category

All five are the same four components wearing different consoles: a managed model endpoint, an identity and permission system, a vector or knowledge store, and an evaluation and guardrail layer. Learn the pattern once through open components and any of them takes about a week to pick up on the job.

Two practical observations for a European job search. First, mid-size technology companies, which is where most agent-adjacent roles sit, generally run Python, Postgres, one cloud and a model API rather than a full enterprise AI platform. Second, large European enterprises skew heavily to Microsoft, so if you ever do invest in one platform, Azure is the highest-probability bet, not AWS.

> THE CORRECT ORDER
>
> Never study a cloud AI platform speculatively. Study it when a specific role, interview or project requires it. The fundamentals in Parts I and III transfer into all five of them; none of the five transfers into the others, and none of them substitutes for the fundamentals.

PART III

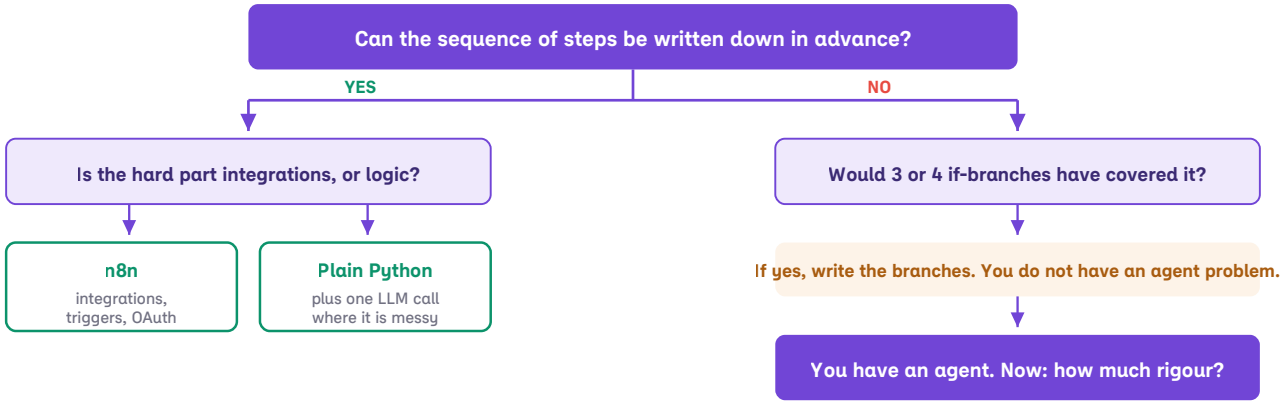
Decision frameworks

Knowing the tools is cheap and getting cheaper. Knowing which tool a given problem deserves, and being able to defend the choice with a reason rather than a preference, is the part that is actually paid for. This is the shortest part of the manual and the one worth rereading.

CHAPTER 18

The master decision tree

One pass through these questions, in this order, resolves most architecture arguments before they start. The order matters: each question is cheaper to answer than the one below it.



Nothing persists, single run, one surface → hand-written loop with the provider SDK. No framework.

Files, repos, shell, long context → Claude Agent SDK. The loop and context management are already solved.

Must resume, pause for a human, or be audited → LangGraph. This is precisely what it exists for.

Distinct roles, or contexts that must stay isolated → subagents first; CrewAI or LangGraph if roles are genuinely separate.

Survives crashes, runs for hours, retried safely → durable execution: Temporal or an equivalent, plus idempotent tools.

At every level: a step limit, a spend limit, traces, and human confirmation before anything irreversible.

Figure 9. The decision tree. Most systems that describe themselves as multi-agent should have stopped two rows higher.

THE JUDGEMENT THAT READS AS SENIOR

Roughly seventy per cent of problems presented as agent problems are better solved by a deterministic workflow with one model call inside it. Saying so, and explaining why, is the strongest single signal of engineering judgement in this field. **Reaching for an agent where a workflow would do is slower, costlier, non-reproducible and harder to debug, and it is the most common mistake made by people who have just learned to build agents.**

CHAPTER 19

n8n vs code vs LangGraph vs CrewAI

The four-way comparison that generates the most confusion, resolved on the axes that actually differ.

Axis	n8n	Plain Python	LangGraph	CrewAI
Time to first working version	Minutes	Hours	Days	Hours
Integrations solved for you	Hundreds	None	None	Some
Version control and review	Poor	Perfect	Perfect	Perfect
Testability	Poor	Perfect	Good	Moderate
State across runs	Basic	You build it	Built in	Limited
Human in the loop	Via a wait node	You build it	First class	Limited
Resume after crash	No	You build it	Checkpoints	No
Multi-agent ergonomics	Awkward	Manual	Explicit and verbose	Its whole purpose
Ceiling	Complex logic	Your own discipline	Learning curve	Fine control

19.1 Reading the table

The table is not a ranking, because the four are not on one axis. A useful way to hold it:

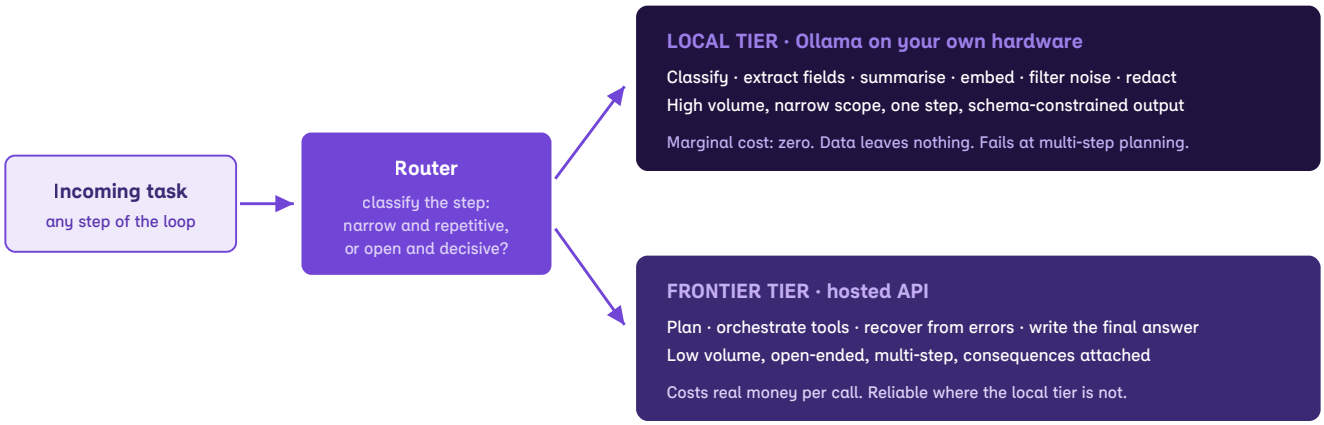
- **n8n** converts integration work into configuration. That is worth an enormous amount and nothing else on the list offers it.
- **Plain Python** is the honest baseline and, for a surprising share of real problems, the right final answer. A framework should have to earn its place against it.
- **LangGraph** converts reliability requirements into structure. You pay a learning curve and receive resumability, human approval gates and auditability.
- **CrewAI** converts an organisational metaphor into code. Fast to express, harder to bend.

The strongest real architectures combine them rather than choosing. A common and very defensible shape: n8n owns the triggers and the OAuth-heavy integrations, and calls a webhook exposed by a LangGraph service that owns the reasoning, the state and the approval gates. Each tool does the thing it is uniquely good at, and neither is asked to do the other's job.

CHAPTER 20

Local vs frontier: the hybrid router

The question is never "local or API". It is "which tier does this specific call deserve", and being able to answer that with measurements is one of the clearest available demonstrations of engineering maturity.



The deliverable is the measurement: cost per run before and after, accuracy on the same eval set, p95 latency. Without numbers this is an opinion.

Figure 10. The hybrid router. The architecture is easy; the measurement is the part that carries weight.

20.1 Where the split usually falls

Send to the local tier	Send to the frontier tier
Intent and category classification	Deciding which tool to call next
Structured extraction from a known format	Reasoning about an unexpected result
Summarising a single document	Synthesising across many sources
Embeddings and semantic filtering	Writing anything a human will read as final
Redacting or masking before anything leaves the network	Anything with a consequence attached

20.2 How to make it a portfolio asset rather than a configuration detail

1. Build the same capability twice: once entirely on the frontier tier, once routed.
2. Run both against the same evaluation set of two hundred cases.
3. Record cost per run, accuracy, and p95 latency for each.

4. Write up the trade in a short document, including where routing made things worse, because that is the part that reads as honest.

The resulting sentence, with your own numbers in it, is a complete technical interview answer: "I moved classification to a local model and cut cost per run by a measured margin at equal accuracy, but I kept planning on the frontier tier because below a certain size the tool-calling reliability collapses after about three steps."

CHAPTER 21

What to pay for

One distinction removes most confusion about AI spending, and most people building agents get it wrong for at least a month.

SUBSCRIPTIONS AND API CREDITS ARE DIFFERENT MONEY

A consumer subscription pays for **you** to use a chat interface or a coding tool. **Your agents do not run on it.** Agents consume API credits, billed per token, on a separate account. Budgeting a subscription and expecting your agents to run inside it is the classic first-month mistake.

21.1 The subscription landscape

Almost every provider has converged on the same four-rung ladder: a free tier, a tier around twenty dollars, a tier around one hundred, and a tier around two hundred. The rungs are not features, they are **usage allowances**. What you buy at the higher tiers is mostly permission to keep going.

Provider	Free	~$20 tier	~$100 tier	Top tier
Claude Anthropic	Yes, limited daily messages	**Pro $20** all model tiers, Projects, Claude Code	**Max 5× $100** roughly five times Pro's allowance	**Max 20× $200**. The rung that matters here is Pro, because Claude Code is included from the first paid tier
ChatGPT OpenAI	Yes, tight message cap	**Plus $19.99** plus a **Go** tier at $8	**Pro 5× $100**	**Pro 20× $200**, effectively unlimited. The $8 Go tier is the cheapest way to keep a second ecosystem in view
Gemini Google	Yes, generous on the Flash tier	**AI Pro $19.99** 1M context, video generation, monthly credits	**AI Ultra 5× $99.99**	**AI Ultra $199.99**, with the largest credit allowance and bundled storage. The free tier is the most usable of any provider
Grok xAI	Yes, very limited	**SuperGrok Lite $10** **SuperGrok $30**	**SuperGrok Plus $100**	**SuperGrok Heavy $300**, the most expensive consumer tier in the field. Grok Bot access sits on the SuperGrok tiers
Perplexity	Yes, a few searches a day	**Pro $20** weekly Pro searches, monthly deep research	—	**Max $200**. A search product rather than a building tool; relevant for research, not for agents
Copilot Microsoft	Yes, basic	**Microsoft 365 Personal $19.99**, bundled with the Office applications	Priced as part of Microsoft 365 rather than as an AI product. Relevant if your employer already lives there	

THE DISTINCTION THAT COSTS PEOPLE A MONTH OF CONFUSION

Everything in that table is a **consumer subscription**: it pays for a human to use a chat interface or a coding tool. **Your agents do not run on it.** Agents consume API credits, billed per token on a separate account with separate pricing. A person can hold a $200 subscription and still receive an API invoice, because they are two different products bought from the same company.

What the API costs, and why the subscription table is the wrong place to look

API pricing is quoted per million tokens, separately for input and output, and the spread between tiers within one provider is larger than the spread between providers. As an order of magnitude at the time of writing, a frontier model runs around ten dollars per million input tokens and fifty per million output, while the fast tier of the same family runs around one and five. That is a tenfold difference inside a single provider, for work that in many steps is indistinguishable.

Two consequences follow directly:

- **Your model-tiering decision dwarfs your provider decision.** Moving the classification steps of a loop from the frontier tier to the fast tier saves more than switching vendors ever will.
- **Blended price is the number to compare**, not the headline input price, because agent loops are overwhelmingly input-heavy. The chart in Chapter 7 uses blended pricing for exactly this reason.

PRICES IN THIS CHAPTER DECAY FASTER THAN ANYTHING ELSE IN THE MANUAL

Tiers get renamed, allowances get quietly adjusted, and new rungs appear. Treat this page as the shape of the market, which is stable, and check the provider's own pricing page for the number, which is not. The structural advice below does not depend on any figure above.

21.2 A rational stack for someone learning to build

Item	Verdict	Reasoning
One builder subscription Claude Pro, or Max if you build daily	YES	Your primary tool. Upgrade tiers only when rate limits genuinely obstruct you, never pre-emptively
API credits	YES	What your agents actually consume. A modest monthly budget goes a long way if the cheap tier does most of the loop iterations
OpenRouter, pay as you go	YES	One key, many models. The only honest way to compare models, and it costs nothing when idle
Langfuse	YES	Free and self-hosted. Your evaluation and observability layer, and a demonstrable skill in itself
A second frontier chat subscription	OPTIONAL	Worth it only for ecosystem literacy: knowing how the other side behaves. Use the API pay-as-you-go instead if money is tight
A second coding tool	NO	Redundant once you have one you know well. Depth beats breadth here by a wide margin
Hosting	LATER	An always-on machine you already own beats a paid VPS until you need a public URL or CI-driven deployment

Item	Verdict	Reasoning
Paid courses	SELECTIVE	See Chapter 30. The best material in this field is free, and the paid material is mostly repackaged documentation

21.3 The biggest lever on cost is not the price list

Before changing plans, change the architecture. In rough order of impact:

- **Model tiering.** Most loop iterations do not need the frontier model. This is usually the largest available saving by an order of magnitude.
- **Prompt caching.** Mark the stable prefix as cacheable. Large reduction, zero behavioural change.
- **Tool output truncation.** Unbounded outputs are the main cause of runaway token counts.
- **Compaction.** Stop resending the full transcript once a summary would do.
- **Step limits.** Cap the loop so a confused run cannot become an expensive one.

CHAPTER 22

Transferable skills vs lock-in

The concern is legitimate: time spent on a vendor console you never meet again is time lost. The resolution is a three-tier map, and a nuance that makes tier three less frightening than it looks.

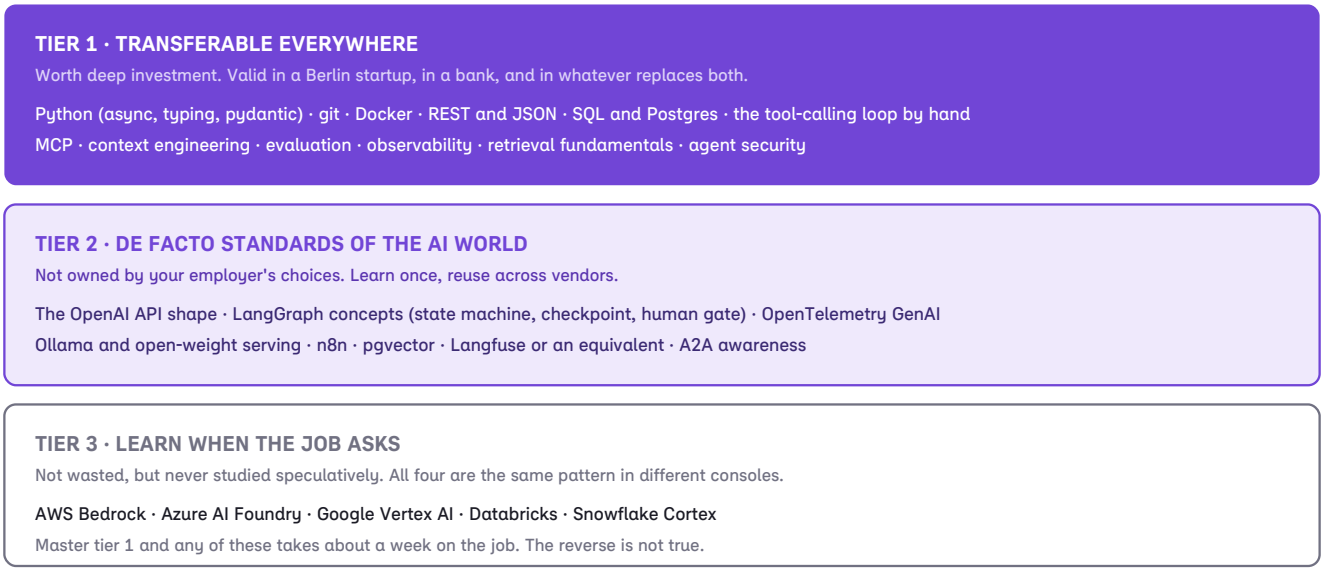


Figure 11. The three tiers. Investment should be heavily weighted to the top band and strictly on-demand for the bottom one.

22.1 The nuance about tier three

Enterprise platforms are not a trap, they are a deferral. All of them are a managed endpoint plus identity plus a vector store plus guardrails. Learning that pattern once through open components means the fifth console you meet takes a week, not a quarter. The mistake is not learning them; it is learning them first, because the console teaches you its menus while the fundamentals teach you the field.

22.2 The thing most often written off that should not be

Visual automation experience is routinely discounted by the people who have it, and it should not be. It appears explicitly in job adverts, it proves delivery rather than study, and it is the fastest evidence that someone has shipped things other people depend on. The correct move is not to hide it and lead with frameworks; it is to position it as the integration layer of a system whose reasoning layer you also wrote in code.

> **A TEST FOR ANY NEW TOOL**
>
> Before investing a weekend in something, ask: **if this product disappeared tomorrow, what would I still know?** For MCP, LangGraph concepts, evaluation and retrieval, the answer is "almost everything". For a specific cloud console, the answer is "the menus". Spend accordingly.

PART IV

What separates an expert

Roughly seventy per cent of being good at this is being good at things that existed long before language models: supervised processes, state in a database, queues, idempotency, secrets, logs and reproducible deployment. The other thirty per cent is the loop, the tools, the context and the evaluation. People who learn only the thirty per cent build demos. This part ends with the process experienced builders actually follow, step by step.

CHAPTER 23

The eight marks of an agent engineer

None of these is "knows framework X". All of them are visible in a code review, a design document or a forty-minute conversation, which is exactly why they are what gets assessed.

1. Knows when not to build an agent

The clearest signal of all. Most problems presented as agent problems are workflows with one ambiguous step. Choosing the deterministic solution, and being able to say why, demonstrates more than any architecture diagram. The inverse, reaching for multi-agent because it is interesting, is the defining error of the newly competent.

2. Designs the action space, not the prompt

Tool names, descriptions, parameter schemas, defaults, idempotency and above all error messages determine behaviour more than prompt wording does. An expert spends more time on the tool layer than on the system prompt, and writes error strings that teach the model what to try next.

3. Treats context as a budget

Truncation, compaction, selective retrieval, subagents for isolation, caching of stable prefixes. Someone who has never thought about this ships agents that get slower, dumber and more expensive the longer they run, and cannot explain why.

4. Evaluates

A dataset of real cases, a scoring method, a number, and a CI job that fails when the number drops. This is the single rarest of the eight and the one that most reliably separates a product from a toy. Every failure becomes a permanent test case.

5. Can explain any past run

Traces that answer "why did it do that on Tuesday at three". Without them, debugging is guesswork dressed up as intuition, and reliability claims are unfalsifiable.

6. Knows the cost and latency profile

Tokens per run, euros per run, p95 latency, and a deliberate routing plan between model tiers. An engineer who cannot state the cost of one execution does not yet own the system.

7. Builds for failure

Retries with backoff, idempotent tools, hard step limits, hard spend limits, graceful degradation, and mandatory human confirmation before anything irreversible. A system that can send an email, move money or delete a row without a gate is not finished.

8. Thinks about the adversary

Prompt injection arriving inside a document the agent was asked to read, tool poisoning from a third-party MCP server, over-broad credentials, secrets leaking into context or logs. Agents execute actions on untrusted input by design, which makes this a first-order concern rather than a hardening pass.

> I do not build agents. I build systems in which a model makes some of the decisions, inside limits I define and can audit.

That sentence, delivered in an interview and then backed with a trace, an eval score and a cost figure, does more than any list of framework names.

CHAPTER 24

Failure modes and anti-patterns

Named failures are easier to avoid and much easier to discuss. These are the ones that recur.

24.1 Runtime failures

Failure	What it looks like	Fix
The infinite loop	The agent calls the same tool with slight variations forever	Hard step limit, plus loop detection on repeated identical calls. Return an explicit message telling it to change approach
Context exhaustion	Quality degrades sharply mid-run; the system prompt stops being obeyed	Truncate tool output, compact history, move detail to files the agent can read back
Silent fabrication	A tool fails; the agent answers from prior knowledge as if it had data	Errors must be explicit and loud in the tool result. Instruct and evaluate for refusal over invention
Cost explosion	A bill arrives that is an order of magnitude beyond expectation	Per-run spend caps enforced in code, not policy. Alert on token counts per run, not per month
Cascading autonomy	One wrong early decision compounds through twenty steps into real damage	Confirmation gates before irreversible actions. Small reversible steps. Checkpoints to roll back to
Non-idempotent retries	A retry after a timeout sends the message or charges the card twice	Idempotency keys on every tool with a side effect. Treat this as mandatory, not advanced
Stale memory	The agent confidently uses a fact that was true last quarter	Timestamp memories, expire them, prefer live lookups over remembered values for anything volatile

24.2 Design anti-patterns

Anti-pattern	Why it is wrong
Multi-agent by default	Each additional agent multiplies cost, latency and failure surface. One agent with well-designed tools beats five with vague roles almost every time. Use multiple agents for context isolation or genuinely distinct expertise, not for organisational aesthetics
The god tool	One `do_everything(action, params)` tool. The model cannot reason about a schema that means anything. Many small, sharply described tools always win
Prompt as configuration	Behaviour tuned by editing a prompt in production with no test set. Every fix silently breaks something else, and nobody finds out
Framework before pain	Adopting an orchestration framework before feeling a specific need. You learn its API rather than the domain, and the API expires first
Demo-driven development	Optimising the happy path shown to stakeholders. The gap between demo and production is entirely made of the cases the demo avoided

Anti-pattern	Why it is wrong
RAG everything	Building a retrieval pipeline for a corpus that fits comfortably in context. Complexity with no benefit
Fine-tuning for facts	Expensive, slow to update, and it still hallucinates. Fine-tuning changes behaviour; retrieval supplies knowledge

CHAPTER 25

Security for agent systems

Agents execute actions based on untrusted text, by design. That single sentence is why agent security is a different discipline from application security rather than a subset of it.

25.1 The threat model

Threat	How it happens	Mitigation
Indirect prompt injection	Instructions hidden in a document, web page, email or ticket that the agent was legitimately asked to read	Treat all retrieved content as data, never instructions. Keep it in clearly delimited blocks. Never let retrieved text expand the agent's permissions. Gate side effects behind confirmation
Tool poisoning	A third-party MCP server whose tool descriptions carry instructions into the model's context	Audit and pin third-party servers. Prefer ones you wrote. Review descriptions, not only code
Excessive agency	The agent holds credentials far broader than its task requires	Scope credentials per tool. Read-only by default. Separate identities for separate capabilities
Secret leakage	Keys pasted into prompts, or echoed into traces and logs	Secrets live in the environment and are injected inside the tool, never in the model's context. Redact at the logging layer
Data exfiltration	An injected instruction makes the agent send internal data to an external endpoint	Allowlist outbound destinations. Be extremely careful with any tool that can post arbitrary content anywhere
Confused deputy	The agent acts with its own high privilege on behalf of a low-privilege user	Propagate the user's identity and permissions through to every tool call. Never let the agent be more privileged than its requester

25.2 The rules that hold regardless of stack

- **Least privilege per tool**, not per agent.
- **Irreversible actions are gated** behind explicit human confirmation. Sending, paying, deleting and publishing are all irreversible.
- **Retrieved content is data.** Never concatenate it into the instruction region of a prompt.
- **Every tool call is logged** with its arguments, its result and the identity it acted under.
- **Spend and step limits are enforced in code**, because a compromised agent will not respect a policy document.
- **Prefer reversible designs.** An agent that drafts and a human that sends is a vastly better risk profile than an agent that sends, and in most workflows it is barely slower.

WORTH SAYING OUT LOUD IN INTERVIEWS

Very few candidates raise prompt injection unprompted. Doing so, with a concrete example of how you constrained a specific tool, is one of the highest-signal things you can say, because it demonstrates that you have thought about what happens when your system meets the real world rather than a demo script.

CHAPTER 26

Cost and latency engineering

Two numbers every owner of an agent should be able to state without looking them up: what one run costs, and what the ninety-fifth percentile run takes. Not being able to is the tell.

26.1 Why agents are expensive, mechanically

Each iteration resends the entire conversation. A run of n steps therefore costs on the order of n^2 in input tokens, not n. Combine that with a tool returning a large payload and the cost of a single run can move by two orders of magnitude without anything looking wrong. Almost every "why is this so expensive" question resolves to this paragraph.

26.2 Levers, in order of effect

Lever	Typical effect	Cost of implementing
Tier the models	Very large	Low. A routing function and an eval set to prove quality held
Cache the stable prefix	Large	Very low. Mark system prompt and tool schemas as cacheable
Truncate tool output	Large and erratic	Trivial. One line per tool
Compact history	Moderate to large on long runs	Moderate. Needs care not to drop the goal or open questions
Cut unnecessary steps	Moderate	Moderate. Better tools mean fewer exploratory calls
Parallelise independent calls	Latency only	Low. Large perceived-speed win at identical cost
Stream the output	Perceived latency only	Low, and it changes how the system feels more than any other single change

26.3 What to instrument from day one

- Input tokens, output tokens and cached tokens per call, and summed per run.
- Number of steps per run, and the distribution across runs rather than the mean.
- Latency per tool call, so you can see which tool is the bottleneck.
- Failure rate per tool, which is usually where reliability problems actually live.
- Cost per successful outcome, which is the only figure a business ever cares about.

CHAPTER 27

How an agent actually gets built

Every previous chapter describes properties a good agent has. This one describes the process that produces them. It is not the process most tutorials show, and the difference is the whole point: experienced builders spend almost no time on the parts beginners spend almost all of their time on.

THE SHORT VERSION

You do the task by hand first. You name the tools before you write them. You write the loop with no framework. You collect twenty failures before you fix one. You turn those failures into a test set. Then, and only then, you make it autonomous.

27.1 Where the time actually goes

The single most surprising thing for someone arriving from tutorials is the allocation of effort. Prompt writing, which dominates most learning material, is a minor line item. Tool design and error analysis, which most material skips, are the job.

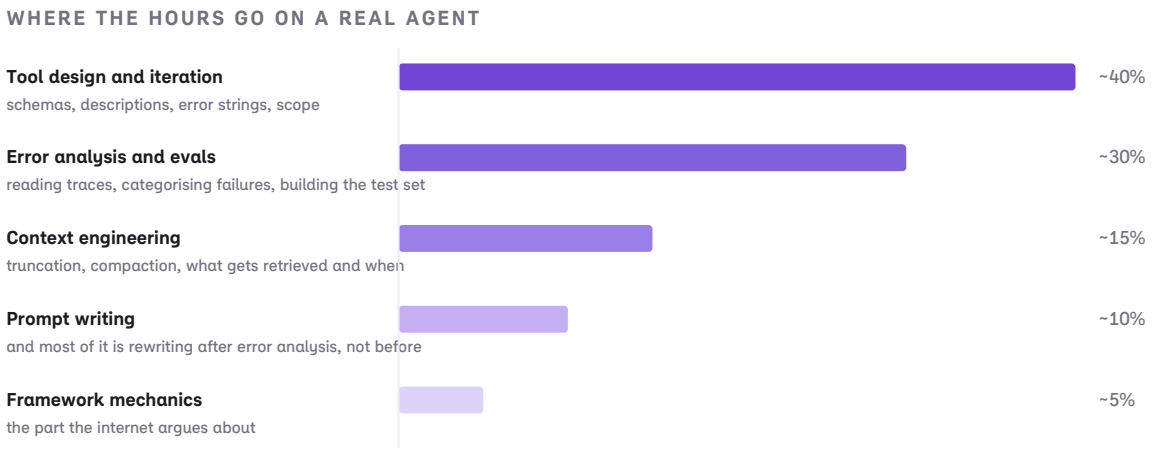


Figure 12. Indicative, not measured, but the ordering is consistent across practitioners. If your own split is inverted, that is the finding.

27.2 The nine steps, in order

Step 1 · Do the task by hand, three times, and write down every move

Before any code. Perform the task yourself and keep a literal transcript: every file you opened, every query you ran, every moment you had to decide something, every moment you had to go and find information you did not have. Do it three times on different inputs so the variable parts separate from the fixed parts.

That transcript is the specification. The fixed parts become code. The variable parts become the agent's decisions. The moments where you went looking for information become tools. Practitioners who skip this step end up designing tools from imagination and discover in week three that the agent needs something nobody exposed to it.

Step 2 · Name the tools before writing any of them

From the transcript, list the actions. Write the signatures and docstrings first, with stub bodies that return a fixed sample. Then run the agent against the stubs and watch what it does. It will call things in an order you did not expect, ask for a tool you did not list, and misread a description you thought was obvious. All of that is free information, obtained before you built anything.

This is also where the discovery tools get invented. Almost every agent needs a way to resolve its own ignorance: `list_cities`, `describe_schema`, `what_columns_exist`. Without them the model guesses, and a guess that looks plausible is worse than an error.

Step 3 · Write the loop with no framework

Forty lines, as in Chapter 2. A step limit, a spend limit, tool errors caught and returned as text, output truncated. At this point the system already works end to end on the happy path, and you own every line of it. Adopting a framework here would mean debugging someone else's abstraction while you still do not know what your own system needs.

Step 4 · Run it on real inputs and log everything. Fix nothing yet

The discipline that separates this from hobby work. Point it at twenty or thirty genuine cases, record every trace, and resist repairing anything. Fixing failure one before you have seen failure twenty means you will optimise for a case that is not representative, and you will not notice that failures three, seven and fourteen share a single root cause.

Step 5 · Do the error analysis

Read every failed trace and label the cause in one word. Then count. The distribution is almost always lopsided: two or three categories account for most of it, and the largest category is usually **a tool problem rather than a prompt problem**. Typical labels:

Label	What it looks like	Usual real fix
Missing capability	The agent needed something no tool provides	Add the tool. Not a prompt fix
Bad tool description	It called the right tool with wrong arguments	Rewrite the docstring and the schema
Unhelpful error	A failure ended the run instead of redirecting it	Make the error message name the next thing to try
Context loss	It forgot a constraint stated at the start	Truncation, compaction, or restate the constraint per step
Wrong plan	It chose a reasonable but incorrect approach	This one is genuinely a prompt or examples fix. It is rarer than expected
Model limit	The step needed reasoning the tier cannot do	Route that step to a stronger model, or decompose it

Counting before fixing is what makes this engineering rather than tinkering. Most people change the prompt after every single failure, which is why their agents oscillate: each fix silently breaks a case they are no longer testing.

Step 6 · Turn the failures into the eval set

Every failed case becomes a permanent test case with an expected outcome. Twenty is enough to start, eighty is comfortable. Now you have a score. From this point every change is measured, and a change that lowers the score gets reverted instead of defended.

Step 7 · Fix, measure, repeat, one change at a time

Change one thing. Re-run the set. Keep or revert. Changing three things and watching the score rise teaches you nothing about which one helped, and one of the three is often making things worse under cover of the other two.

Step 8 · Add the boundaries before it runs unattended

Step limit, spend limit, tool timeouts, idempotency on anything with a side effect, and a human confirmation gate in front of every irreversible action. Appendix B is the checklist. This step takes an afternoon and prevents the two failure modes that produce real damage: the runaway loop and the confident wrong action.

Step 9 · Deploy narrow, widen slowly

The pattern that experienced builders use and beginners skip: **the agent drafts, a human sends, for the first two weeks.** It runs on schedule, produces its output, and a person reads it before anything happens. You collect real failures at zero risk, and the eval set grows on production data instead of imagined data. Autonomy is then widened only where the score justifies it, one action at a time, starting with the reversible ones.

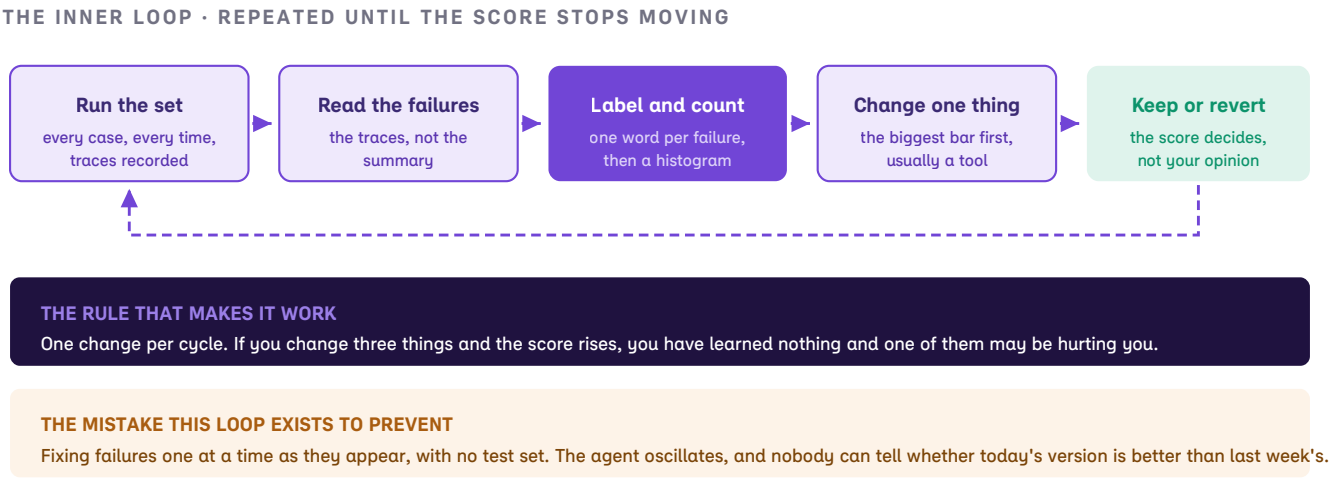


Figure 13. The inner loop. Steps 5 to 7 of the previous page, drawn as the cycle they actually form.

27.3 What experienced builders do not do

They do not	Because
Start by choosing a framework	The choice is uninformed until the system has told you what it needs. Chapter 18 is the sequence
Write a long system prompt first	Most behaviour problems are tool problems wearing a prompt costume. A long prompt hides them
Add more agents to fix quality	A second agent adds cost, latency and a new failure surface. Fix the first agent's tools instead

They do not	Because
Fix failures one at a time as they appear	Without a test set, every fix is a guess and regressions are invisible
Give full autonomy on day one	Drafting first is nearly as useful, costs nothing in risk, and generates the eval data for free
Chase the newest model for quality	A weak tool layer performs badly on every model. Upgrading the model hides the problem and raises the bill
Trust a demo that only ran once	Non-determinism means one success is an anecdote. Ten runs of the same case is a measurement

27.4 The working rhythm

Day to day, the job looks less like writing code and more like running an experiment. A typical week on a live agent:

- **Daily.** Read the traces of whatever ran overnight. Not the summary, the traces. Label anything that went wrong and drop it into the failure log.
- **Two or three times a week.** One cycle of the inner loop above: read, label, count, change one thing, re-run the set.
- **Weekly.** Check cost per run and step-count distribution. A drift upward is usually a tool returning more than it used to, not the model getting worse.
- **Monthly.** Re-read the tool descriptions as if you had never seen them. They accumulate assumptions that were true three fixes ago.

The coding itself is increasingly delegated to a coding agent, and that is fine and normal. What is not delegated, and is the actual expertise, is deciding what to build, reading the traces, labelling the failures and judging whether the score moved for a real reason.

> **THE ONE HABIT WORTH COPYING ABOVE ALL OTHERS**
>
> Keep a plain text file of every failure you have ever seen, with its label and its fix. After a few months it becomes the most valuable document you own: it is your own error taxonomy, it makes new failures recognisable in seconds, and it is the raw material for every eval set, every design document and every good answer you will give in an interview.

PART V

The learning path

What to study, what to skip, what to certify, what to build, and in what order. Written on one assumption: the goal is not to know more, it is to become visibly employable as someone who builds agent systems.

CHAPTER 28

Theory that lasts, tools that change

Neither extreme works. Pure theory produces someone who can discuss transformers and cannot ship. Pure tooling produces someone whose skills expire with a product. The dividing line is sharper than it looks.

> Learn the theory that survives the tool's death, and the tools that are standards rather than products.

28.1 Theory worth about twenty hours, once, forever

Topic	Why it pays off
How a model generates tokens	Enough to understand why it hallucinates, why context position matters, and why it cannot count characters. No mathematics required
Tokens, context, pricing	You can read an invoice and predict a cost before incurring it
Embeddings and vector search	You can diagnose why retrieval returned nonsense instead of swapping databases and hoping
Chunking and reranking	The two stages where most RAG quality is actually determined
The tool-calling loop	Every framework becomes legible at once
RAG vs long context vs fine-tuning	Stops the single most expensive wrong turn in the field
Context engineering	The difference between an agent that degrades over a long run and one that does not
Evaluation methodology	The highest-leverage topic in the entire list, and the least studied
Agent failure modes and security	Chapters 24 and 25. Pattern recognition that takes years to acquire by accident

28.2 Theory to skip, for this goal

Training models from scratch, backpropagation mathematics, attention-mechanism derivations, CUDA and kernel optimisation, distributed training, and model architecture research. This is machine learning research. It is a different job, with a different title, a different hiring process and a different day. Studying it to become an agent engineer is a several-month detour with almost no return, and the belief that it is a prerequisite is the most common reason capable people never start.

28.3 The working ratio

Roughly twenty per cent theory, eighty per cent building. The failure mode is not the ratio, it is that most people skip the twenty per cent entirely and then cannot explain why their system misbehaves. Do the twenty per cent properly, once, early, and then build almost exclusively.

CHAPTER 29

Certifications ranked by market weight

Certifications are a weak signal in this field, but they are not a zero signal, and they are not all equal. What follows separates what they actually do from what people hope they do.

29.1 What a certification can and cannot do

What it can do

- Get a CV past a keyword filter, which for some large employers is the entire barrier
- Give a career changer a credible reason to be considered at all
- Force structured coverage of ground you would otherwise skip
- Signal seriousness when your job title does not contain the word "engineer"
- Satisfy a partner or procurement requirement in consulting contexts

What it cannot do

- Substitute for a repository someone can read
- Survive a technical interview on its own for thirty seconds
- Prove you have shipped anything that runs unattended
- Outrank a live demo, a written design document, or a merged open-source contribution
- Compensate for being unable to explain your own architecture

THE ORDERING THAT REFLECTS REALITY

A public repository with a real architecture, an eval suite and a written design document outranks every certification on this list. A certification on top of that repository adds a little. A certification instead of it adds close to nothing. **Budget accordingly: most of the effort into the artefact, a small slice into one or two credentials.**

29.2 The tiers

Tier	What sits here	Market weight
S	Not certifications at all: a public repo, a live deployment, a published MCP server, a merged PR to a known project, a written technical post	Highest. This is what actually gets read
A	Hard vendor exams from AWS, Google Cloud and Microsoft, and the specialised Databricks and NVIDIA GenAI exams	Real recognition, especially in larger and more traditional employers

Tier	What sits here	Market weight
B	Entry-level vendor exams and the Anthropic Claude program	Cheap, fast, modest but non-zero signal. Good filler, not a strategy
C	Course completion certificates: DeepLearning.AI, IBM, Hugging Face, LangChain Academy, OpenAI Academy badges	Weak as credentials, strong as structured learning. Take the learning, do not lean on the certificate
D	Generic "certified AI professional" products from unknown bodies	Zero, and occasionally negative. Avoid

29.3 The comparison table

Credential	Tier	Price	Valid	Verdict for an agent-building profile
AWS Certified Machine Learning, Specialty MLS-C01	A	$300	3 yrs	One of the hardest AWS exams and carries real weight. But it is classical ML on AWS, not agents. Only worth it if you are targeting AWS-heavy employers
Google Cloud Professional ML Engineer	A	$200	2 yrs	Strong recognition, often associated with senior roles. Assumes several years on GCP. Same caveat: ML on a cloud, not agent engineering
Databricks Certified Generative AI Engineer Associate	A	$200	2 yrs	**The closest mainstream exam to this actual job.** 45 questions in 90 minutes, weighted to application development (30%), assembling and deploying (22%), design (14%), data prep (14%), evaluation and monitoring (12%), governance (8%). Assumes around six months of hands-on generative AI work
NVIDIA Certified Associate: Generative AI and LLMs NCA-GENL	A	$125	2 yrs	60 minutes, 50 to 60 questions, remotely proctored. Covers core ML and AI (30%), software development (24%), experimentation (22%), data analysis (14%) and trustworthy AI (10%). The NVIDIA name carries weight disproportionate to the exam's difficulty, which makes it good value
Microsoft Azure AI Engineer Associate AI-102	A	~$165	1 yr, free renewal	The highest-probability cloud bet for European enterprise specifically, because Azure dominates there. Skip unless you are aiming at that segment
AWS Certified AI Practitioner AIF-C01	B	$100	3 yrs	Foundational, 40 to 60 hours of preparation. Bridges business and technical language. Fine as a first credential, not a differentiator
Microsoft Azure AI Fundamentals AI-900	B	$99	No expiry	Entry level, 30 to 40 hours. Useful for career changers needing baseline credibility. Little value once you have shipped anything

Credential	Tier	Price	Valid	Verdict for an agent-building profile
Claude Certified Developer, Foundations CCDV-F, Anthropic	B	$125	12 mo	Aimed at software engineers and API developers. Relevant, current, and directly aligned with the stack in this manual
Claude Certified Architect, Foundations CCAR-F, Anthropic	B	$125	12 mo	**The most on-topic exam in existence for this specific skill set.** Its five domains are agentic architecture (27%), Claude Code configuration (20%), prompting and structured output (20%), tool design and MCP (18%), and context and reliability (15%). That weighting is essentially the syllabus of this manual
Claude Certified Architect, Professional CCAR-P	B	$175	12 mo	The step up, aimed at enterprise consultants and technical leads
Claude Certified Associate, Foundations CCAO-F	B	$99	12 mo	Non-technical track, for product, marketing and operations roles. Not the right one for a builder profile
DeepLearning.AI Machine Learning Specialization	C	~$147	n/a	Three months at five hours a week. Widely respected as learning, weak as a credential. Excellent if you want the fundamentals properly
DeepLearning.AI Deep Learning Specialization	C	~$245	n/a	Five months. Appears on many ML job listings. Deeper than an agent engineer needs; take it only if you want the ML path too
IBM RAG and Agentic AI Professional Certificate	C	subscription	n/a	Ten courses, advanced, roughly eight weeks at three hours a week. The most on-topic course-based programme available: LangChain, LangGraph, CrewAI, AG2 and BeeAI, MCP, vector stores, multimodal, and a capstone combining retrieval with an agentic system. Assumes Python. The IBM name also carries weight with non-technical recruiters, which is the main thing a course certificate can buy
IBM AI Engineering / Generative AI Engineering	C	~$196–294	n/a	Four to six months, project-based, vendor-backed. Reasonable for a career changer needing structure and a certificate to point at
PMI Certified Professional in Managing AI CPMAI	C	$500–800+	varies	Programme and delivery management of AI projects. Relevant only if you are moving toward AI programme management rather than building
Anything from an unrecognised certifying body	D	varies	n/a	Ignore. A reviewer who does not recognise the issuer reads it as padding

29.4 The Anthropic program, in detail

Launched through 2026 and delivered by Pearson VUE, online-proctored or at a test centre, with a Credly badge on passing. All four exams share the same shape: 60 questions, mixed multiple choice and multiple response, a passing score of 720 out of 1000, and a twelve-month validity. Up to four attempts per twelve-month rolling period, with waiting periods of 14, 30 and 90 days after successive failures. Preparation runs through the Anthropic Partner Academy.

CHECK THIS BEFORE PLANNING AROUND IT

Registration has been reported as requiring membership of the Anthropic Partner Network, with personal email addresses unable to complete it independently. If so, the practical route is through an employer that is a partner, and the exam is not freely available to an individual. **Verify current eligibility on the Pearson VUE Anthropic page before budgeting time for it**, because this is exactly the kind of detail that changes quietly.

29.5 The recommendation

Situation	Do this
You have shipped things and want one credential	**Databricks GenAI Engineer Associate** for closest topical fit, or **NVIDIA NCA-GENL** for best name recognition per hour invested
You work in a Claude-centred stack and can access the program	**Claude Certified Architect, Foundations.** The domain weighting maps almost exactly onto real agent work
You are targeting European enterprise	**AI-102**, and only that one. Azure is the dominant enterprise stack in that market
You are changing careers and need baseline credibility	**AWS AI Practitioner** or **AI-900**, then stop and build
You already have a strong repository and a live deployment	**Skip certifications entirely** and spend the time on a written technical post about your architecture. It will outperform any of them

CHAPTER 30

Courses and free resources

The best material in this field is free, first-party, and written by the people who built the thing. Paid courses are mostly documentation with a video track, and they lag the documentation by months.

30.1 Tier one: primary sources

Source	Why it is first
Provider documentation Anthropic, OpenAI, Google	Always ahead of every course. The tool-use, prompt-caching, context-management and agent-SDK pages are the actual curriculum. Read them directly
The MCP specification and SDKs	Short, readable, and the most transferable single thing in this manual. Read the spec, then write a server
LangGraph documentation and LangChain Academy	Free, official, concept-led. The LangGraph course teaches state machines, checkpoints and human-in-the-loop, which transfer even if you leave the framework
Engineering blogs from the labs	Posts on agent harness design, context engineering and multi-agent orchestration are where the actual state of the art is published first

30.2 Tier two: structured free courses

Course	Verdict
Hugging Face Agents Course	Free, hands-on, framework-plural, with a certificate. The best free structured introduction to agents available, and it deliberately avoids locking you to one framework
DeepLearning.AI short courses	One to two hours each, built with the vendors themselves. Cherry-pick: agent design, evaluation, RAG, multi-agent. High signal per hour, free to audit
OpenAI Academy	Free self-paced pathways including agents and workflows, designing agentic systems, RAG, evaluation and Codex. Note carefully: the platform states that its badges and pathway certificates are not certifications. Take it for the content, not the badge
Agentic AI Andrew Ng, DeepLearning.AI	About ten hours, intermediate, 31 lessons with graded assignments. Agentic workflows and task decomposition, the reflection pattern, tool use, planning and multi-agent, plus a full module on evaluations and error analysis, which is rare at this level. MCP included. The closest thing to a structured version of this manual's own syllabus
MLOps Zoomcamp DataTalks.Club	Free, roughly nine weeks, cohort-based with a public repository. Deployment, monitoring, CI/CD and experiment tracking. **The highest-return item on this page for anyone whose gap is operations rather than models**, because it supplies the vocabulary that turns "analyst who automates" into "engineer"
LLM Agents MOOC UC Berkeley	Free, research level, guest lectures from people at the major labs. The theoretical framing behind agent architectures rather than another framework tutorial, and credible to cite
LLM Bootcamp Full Stack Deep Learning	Free, about ten hours, production-oriented: cost, latency, architecture, evaluation. Best watched while building rather than before

Course	Verdict
Framework quickstarts CrewAI, Pydantic AI, smolagents	An afternoon each. Enough to have an opinion, which is all you need for most of them

30.3 What to actually read, in order

1. The tool-use documentation of one provider, properly, end to end.
2. The MCP specification, then build one trivial server.
3. One lab engineering post on context engineering.
4. The LangGraph conceptual guide: state, checkpoints, interrupts.
5. One serious treatment of evaluation, which is the topic with the worst ratio of importance to coverage.
6. Everything else, on demand, when a specific problem demands it.

30.4 If you only do two

The list above is long enough to become a way of avoiding work. Two items on it carry most of the value, and they do different jobs: one supplies the content, the other supplies the credential.

START HERE · AGENTIC AI, ANDREW NG, DEEPLEARNING.AI

About **ten hours, 31 lessons, intermediate level, with graded exercises.** It covers agentic workflows and task decomposition, the reflection pattern, tool use, planning and multi-agent systems, and MCP. What makes it unusual is a **full module on evaluations and error analysis**, which is the topic almost every other course at this level skips and the one Chapter 27 of this manual is built on.

It is, in practice, this manual's syllabus in course form, with exercises attached. Highest signal per hour of anything on this page, and available on Coursera where a subscription carries the certificate.

THEN, IF YOU WANT A CREDENTIAL · IBM RAG AND AGENTIC AI PROFESSIONAL CERTIFICATE

Ten courses, advanced level, roughly eight weeks at three hours a week. LangChain, LangGraph, CrewAI, AG2 and BeeAI, MCP, vector stores, multimodal work, and a capstone that combines retrieval with an agentic system. It assumes Python.

On price alone it belongs in tier C of Chapter 29, at roughly $196 to $294. **The calculation changes completely when Coursera access is already paid for**, by an employer, a university or a bundled subscription: the only remaining cost is time, and the IBM name is precisely the kind that clears non-technical recruitment filters, which in the German and wider European market is not a small thing. Under those conditions it moves ahead of the paid vendor exams in Chapter 29, which cost $125 to $200 and do not teach you more.

Those two together cover both halves of the problem: Ng gives you the mental model and the practice, IBM gives you the breadth across frameworks plus something a recruiter recognises. Doing both is roughly sixty hours. Doing neither, and building instead, is also a defensible answer, and Chapter 31 explains why.

THE TRAP IN THIS CHAPTER

Course collecting feels like progress and is not. The completion rate that matters is not how many courses you finished, it is how many things you built that other people can run. If you have watched more hours this month than you have deployed, reverse the ratio.

CHAPTER 31

Portfolio projects that read as senior

Reviewers spend a few minutes on a repository. What they look for is not cleverness, it is evidence of judgement. Six specific things carry almost all of the signal.

31.1 The six signals

Signal	How a reviewer sees it in under five minutes
An evaluation suite	An `evals/` directory with real cases and a score. Instantly separates the top decile
A written design document	A README that states the trade-offs considered and rejected, not just the setup commands
Observability	A screenshot of a trace, and cost per run stated as a number
Safety boundaries	Step limits, spend limits, a human confirmation gate before anything irreversible
Reproducible deployment	A `docker-compose.yml` that brings the whole thing up, and evidence it runs unattended
An honest limitations section	What it does badly and why. Counter-intuitively the strongest credibility signal in the whole repository

31.2 Project archetypes, ranked by signal

Project	Signal	Notes
A published MCP server for a real system	VERY HIGH	Small effort, rare on CVs. Almost everyone consumes MCP servers; very few have written and published one. The single best effort-to-signal ratio available
An agent for your own professional domain	VERY HIGH	Uses knowledge nobody else in the candidate pool has. Synthetic data avoids every confidentiality problem. You can defend every design decision because you understand the domain
An evaluation harness with a public report	VERY HIGH	Compare models or prompts on a task, publish the method and the numbers. Reads as engineering rather than enthusiasm
A hybrid local and frontier router, measured	HIGH	Requires hardware most candidates do not bother with, and produces concrete cost and accuracy numbers
A merged contribution to a known open-source project	HIGH	Third-party validation you did not write yourself. Slow, but it compounds
A hybrid architecture: visual orchestration plus a code reasoning service	HIGH	Demonstrates the judgement from Chapter 19: using each tool for what it is actually good at
A generic research or writing multi-agent system	LOW	There are thousands on GitHub. Only works if the evaluation and cost analysis are the real contribution
A chatbot over your own documents	VERY LOW	The "hello world" of this field. Assume it counts for nothing

ONE GOOD REPOSITORY BEATS FIVE

Five shallow projects read as a course history. One project with an eval suite, a trace screenshot, a cost figure, a compose file and an honest limitations section reads as an engineer. Depth is the whole strategy.

CHAPTER 32

A twelve-week curriculum

Sequenced so that every phase produces something demonstrable, and so that nothing is learned before the pain that justifies it. Weeks are a unit of ordering, not a schedule; compress or stretch to fit real life.

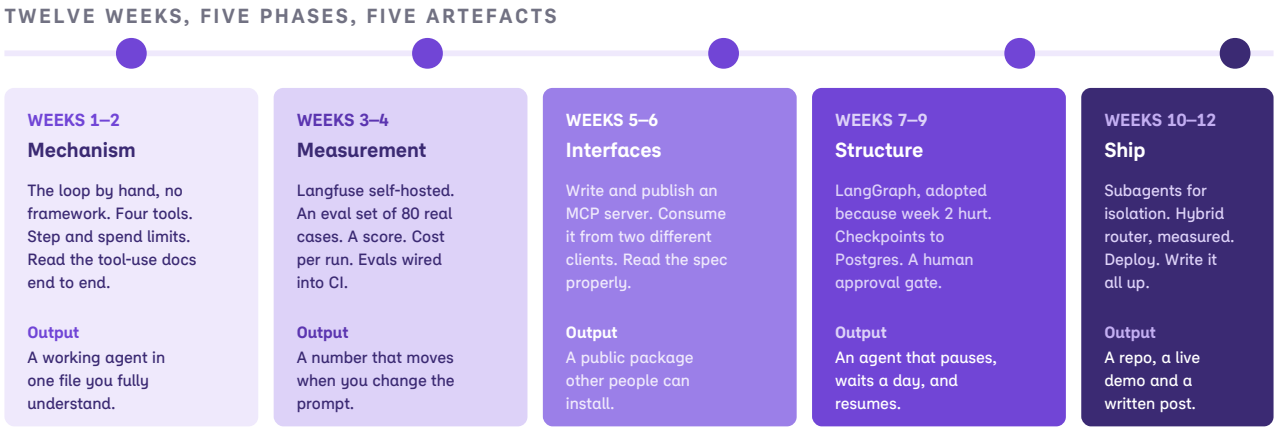


Figure 14. The sequence matters more than the pace. Each phase exists because the previous one created the need for it.

32.1 Why this order and not another

- **The loop before any framework**, so that the framework is a relief rather than a mystery. Adopting LangGraph in week one teaches its API; adopting it in week seven teaches agent architecture.
- **Evaluation before scale**, because without a score, every later change is a guess. This is the phase most people skip and the one that most changes how you are perceived.
- **MCP early**, because it is small, transferable, and produces a public artefact well before the main project is finished.
- **Multi-agent last and sparingly**, because the discipline is knowing when not to, and that judgement only forms after single-agent systems have been pushed to their limit.

32.2 What "done" means at week twelve

Artefact	Test of completeness
A public repository	Someone else can run it with one command and understand the design from the README
A running deployment	It does its job on a schedule while you are asleep, and you can prove it
An eval report	A score, a method, and at least one recorded regression you caught and fixed
A published MCP server	Installable by a stranger
A written post	The architecture, the trade-offs, the numbers, and what you would do differently

CHAPTER 33

Staying current, and what comes next

The release cadence in this field is designed to make you feel behind. A filter is worth more than a feed.

33.1 The filter

Signal	Response
A new protocol reaches a neutral foundation or a standards body	**Pay attention.** This is the rarest and most consequential kind of news. A2A moving to the Linux Foundation and absorbing a competitor is the template
A first-party engineering post on architecture or context	**Read it.** This is where the actual state of the art is published
A capability that changes what is possible, not what is convenient	**Investigate.** Long context and reliable tool calling were of this kind
A new framework promising simpler agents	**Ignore for six months.** If it still exists and people ship with it, look then
A benchmark leaderboard shuffle	**Ignore.** Benchmarks correlate weakly with agent reliability, which is what you care about
A product launch with no technical detail	**Ignore.** Note the name and move on

33.2 Three sources, not thirty

The filter above is worthless without a short list of inputs to apply it to. A feed of thirty sources is a way of feeling informed while learning nothing.

Source	What it is for
Simon Willison's weblog	The best signal-to-noise filter in the field. Tests things himself, writes up what actually changed, and is consistently early without being breathless
Latent Space newsletter and podcast	Technical depth and long-form interviews with people who build the systems rather than market them
The Batch DeepLearning.AI, weekly	A calm weekly summary with judgement attached. Good for catching what you missed without doom-scrolling
Provider changelogs and engineering blogs	Anthropic, OpenAI and Google, by RSS. This is where capability changes and deprecations appear first, and where the best writing on agent architecture is published
Release notes of the four tools you use	Not the whole ecosystem. The four things your system actually depends on

General technology press, the large aggregators and most of the social feed give you the same news two days later with none of the detail that would change a decision. Dropping them costs nothing.

THE SIX-MONTH FREEZE

Pick your stack, then freeze it for six months. **One new tool per month at most, and only if it replaces something rather than adding to it.** Everything else goes into a file called "look at this in March", which you open in March. Nine out of ten entries will be dead by then, and you will have saved nine weekends. The feeling of being behind comes from comparing your depth on every tool against someone who uses only that one; the cure is depth on few, not breadth on many.

33.3 A sustainable routine

- **Weekly, thirty minutes.** The changelogs of the three or four tools you actually use. Nothing else.
- **Monthly, two hours.** One deliberate experiment with something new, timeboxed, with a written verdict of keep or discard.
- **Quarterly, half a day.** Re-read your own architecture and ask which of your decisions a new capability has made obsolete.
- **Continuously.** When something surprises you, write down why. That file becomes the most valuable document you own.

33.4 What comes after agents

Agents are not a fashion, but they are not the destination either. Today's agent is frozen at deployment and has no real memory: everything it "remembers" is text you replayed into a context window. Four research directions are aimed squarely at those two gaps, and they will change what the word means.

Direction	What it would change
Continual learning	Models that absorb something new without a full retrain. Architectures exploring nested or memory-augmented learning point this way. It would collapse the distinction between memory and weights, which is the distinction most of Chapter 5 exists to manage
World models	Simulating an environment from observation rather than describing it in text. Relevant the moment agents act on physical or simulated systems instead of documents
Orchestration and routing	Routing intelligently between small models, large models, retrieval and tools, decided by the system rather than hand-written. Chapter 20's router, made automatic and learned
Self-refinement	Loops that criticise and correct their own output before returning it. Already partly available as the reflection pattern, and it is the cheapest quality gain most agents are not using

The practical reading: **none of the four threatens the skills in this manual, and all four increase the value of two of them.** Evaluation matters more when the system changes itself, and tool and permission design matter more when the system plans more of its own path. A system that learns continuously without a test set is not an improvement, it is an unmonitored drift.

THE CLOSING THOUGHT

Every product named in Part II will look dated within a few years. The four layers, the loop, the context budget, the decision of when not to build an agent, and the discipline of measuring before claiming will not. **Study the second list. Use the first one.**

APPENDICES

Reference

A glossary of the vocabulary used across the field, two code cards worth keeping, a pre-flight checklist for anything you are about to leave running unattended, and the sources behind the September 2026 snapshot.

APPENDIX A

Glossary

A.1 Core mechanism

Term	Definition
Agent	A system in which a model, rather than fixed code, chooses the next action at runtime from a set of available tools, in a loop, until it decides it is done
Agent loop	Send history and tool catalogue, receive text or a tool call, execute it, append the result, repeat. The entire mechanism
Tool	A function exposed to the model with a name, description and JSON schema. The model can request it; only your code can run it
Tool call	A structured block emitted by the model naming a tool and its arguments. Not an execution, a request for one
Action space	The complete set of things an agent can do. Designing it well is most of the engineering
Workflow	A system whose path is fixed in code, even if it calls a model along the way. Not an agent
Orchestration	Running the loop plus the bookkeeping: state, limits, retries, traces
Handoff	One agent transferring control and context to another
Subagent	A child agent invoked for a subtask with its own isolated context, returning only a conclusion to the parent
Human in the loop	A deliberate pause for human approval before continuing, ideally with the ability to resume days later
Durable execution	A run that survives process death and resumes from where it stopped rather than from the beginning
Checkpoint	A saved snapshot of agent state that can be resumed or rewound to
Idempotency	The property that performing an action twice has the same effect as performing it once. Mandatory for any tool with a side effect
ReAct	Reason then act then observe. The interleaved thinking-and-tool-use pattern most agent loops implement

A.2 Models and inference

Term	Definition
LLM	Large language model. A function from text to text with no memory, no clock and no ability to act
Token	The unit models read and write, roughly three quarters of a word in English. The unit you are billed in
Context window	The maximum tokens a model can consider in one call. A budget, not a store

Term	Definition
Stateless	The property that each API call is independent. Continuity is an illusion you create by resending history
Prompt caching	Marking a stable prefix so the provider can reuse it across calls at reduced cost
Structured output	Constraining generation to a schema so the result is reliably parseable
Open weight	A model whose parameters are downloadable and runnable locally. Not the same as open source, which would also require training data and code
Quantisation	Reducing numeric precision so a model fits in less memory, trading some quality for feasibility
Inference	Running a model to produce output, as distinct from training it
Fine-tuning	Further training on your examples to change behaviour or format. Does not reliably add knowledge
Distillation	Training a small model to imitate a large one, to get most of the behaviour at a fraction of the cost
Hallucination	Fluent, confident output that is not grounded in anything true. A property of the mechanism, not a bug to be patched
Router	Logic that sends each request to a different model tier based on what the request needs
Hugging Face	The distribution layer for open models: the Hub where weights and datasets live, plus the libraries, hosted inference and demo hosting built around it
Unified memory	On Apple Silicon, memory shared by CPU and GPU. It sets which models fit; memory bandwidth sets how fast they run
MLX	Apple's array framework for machine learning on Apple Silicon, used when Ollama's defaults leave performance on the table

A.3 Context, retrieval and memory

Term	Definition
Context engineering	Deciding what occupies the context window at each step. The successor discipline to prompt engineering
Compaction	Replacing old conversation turns with a summary to reclaim context space
RAG	Retrieval-augmented generation. Fetch relevant passages, then generate grounded in them
Embedding	A vector representing the meaning of a piece of text, so that similarity can be computed
Vector database	A store optimised for nearest-neighbour search over embeddings
pgvector	The Postgres extension that makes a normal relational database a competent vector store. The sensible default
Chunking	Splitting documents into retrievable units. Where most RAG quality is won or lost
Reranking	Reordering retrieved candidates with a stronger model before using them. Frequently the largest single quality gain
Hybrid search	Combining keyword and vector search. Beats either alone in most real corpora

Term	Definition
Grounding	Tying generated claims to retrieved source material, ideally with citations
Long-term memory	Facts and preferences persisted outside the conversation and retrieved selectively when relevant

A.4 Protocols, evaluation and operations

Term	Definition
MCP	Model Context Protocol. An open standard for exposing tools and data to any agent client. Write once, use everywhere
MCP server	A process exposing tools and resources over MCP. The portable unit of agent capability
A2A	Agent-to-Agent Protocol, governed by the Linux Foundation. How independent agents discover and delegate to each other
Agent Card	A signed description of an agent's identity and capabilities under A2A
WebMCP	A W3C-track proposal letting websites expose structured interfaces to agents instead of being scraped
OSI	Open Semantic Interchange. A shared way to define metrics and dimensions so agents agree on what a business term means
AP2 / ACP	Competing protocols for agent payments and in-conversation checkout
Eval	A repeatable measurement of quality against a fixed dataset. The dividing line between a toy and a product
LLM as judge	Using a model to score outputs against a rubric where no deterministic check exists
Trajectory evaluation	Scoring the sequence of tool calls rather than only the final answer. Usually more informative for agents
Golden set	The curated cases your system must never regress on
Error analysis	Reading failed runs, labelling each cause in one word and counting the labels, so effort goes to the largest category instead of the most recent complaint
Reflection	An agent design pattern in which the system critiques and revises its own output before returning it
Failure log	A running file of every failure seen, its label and its fix. Over months it becomes a personal error taxonomy and the raw material for every eval set
Trace	The full recorded execution of one run: every call, argument, result, token count and cost
Span	One timed unit inside a trace, such as a single model call or tool invocation
Observability	The ability to answer why the system behaved as it did on a specific past run
Guardrail	A check before or after a model call that blocks unacceptable input or output
Prompt injection	Instructions smuggled into content the agent reads, hijacking its behaviour. The defining security problem of agents
Tool poisoning	A malicious tool description that injects instructions simply by being listed in the catalogue

Term	Definition
Excessive agency	Granting an agent more permission than its task requires
Confused deputy	A privileged agent acting on behalf of a less privileged user without checking that user's permissions
Headless mode	Running an interactive agent tool non-interactively from a script, cron job or pipeline
Sandbox	An isolated environment where agent-generated code or commands can run without reaching anything that matters

APPENDIX B

Reference cards

B.1 A minimal MCP server

The highest effort-to-signal artefact in this manual. Once this runs, every MCP-speaking client can use your capability: coding agents, orchestration frameworks, visual tools and whatever replaces them.

```
# pip install mcp
from mcp.server.fastmcp import FastMCP

mcp = FastMCP("fleet-ops")

# The docstring IS the interface. The model reads this, not your code.
# Say what it returns, what the units are, and what it does NOT do.

@mcp.tool()
def supply_gap(city: str, hour: int) -> str:
    """Return the driver supply gap for a city and hour.

    Args:
        city: city code, e.g. "MAD". Use list_cities if unsure.
        hour: local hour, 0-23.

    Returns a one-line summary: requested rides, available drivers,
    and the gap as a percentage. Read-only. Covers the last 90 days only.
    """
    if not 0 <= hour <= 23:
        # Errors are instructions. Tell the model how to recover.
        return f"Invalid hour {hour}. Must be 0-23."
    row = db.fetch_gap(city, hour)
    if row is None:
        return f"No data for {city} at {hour}h. Try list_cities to check the code."
    return f"{city} {hour}h: {row.requested} requested, " \
           f"{row.available} available, gap {row.gap_pct:.1f}%"

@mcp.tool()
def list_cities() -> str:
    """List valid city codes with their full names."""
    return ", ".join(f"{c.code} ({c.name})" for c in db.cities())

if __name__ == "__main__":
    mcp.run()
```

Four rules visible in that file, and they generalise to every tool you will ever write.

- **The docstring is the interface.** It is what the model reads to decide whether and how to call the tool. Write it for a competent stranger.
- **State the boundaries.** Read-only, ninety days, this unit. Unstated boundaries become confident wrong answers.
- **Errors are instructions.** Every failure message should name the next thing to try.
- **Provide the discovery tool.** `list_cities` exists so the agent can resolve its own ignorance instead of guessing.

B.2 Pre-flight checklist before leaving anything unattended

Area	Check
Limits	Hard step limit. Hard spend limit per run, enforced in code. Timeout per tool call
Safety	Every irreversible action gated behind human confirmation. Every side-effecting tool idempotent
Permissions	Narrowest credentials per tool. Read-only wherever possible. No shared superuser key
Secrets	Out of the repository, out of the prompt, redacted in logs and traces
Injection	Retrieved content delimited and treated as data. Outbound destinations allowlisted
Context	Every tool output truncated. Compaction threshold set. Stable prefix cached
Observability	Traces recorded with arguments, results, tokens and cost. Queryable after the fact
Evaluation	An eval set exists, has a score, and runs in CI. The last regression is documented
Reliability	Restart policy set. Health check or heartbeat. Alert if a scheduled run does not happen
Reproducibility	One command brings the whole system up on a clean machine
Failure	You know what happens when the model API is down, when a tool times out, and when the disk fills

THE FINAL TEST

If you cannot say, without looking anything up, **what one run costs**, **what your eval score is**, and **what the agent would do if its most important tool started failing silently**, the system is not finished. Those three answers, more than any framework or certification on the preceding pages, are what being an agent engineer means.

APPENDIX C

Sources

The structural content of this manual is drawn from direct practice and from first-party documentation. The September 2026 inventory in Part II and the certification data in Chapter 28 were verified against the following at the time of compilation.

C.1 First-party documentation

- Anthropic: Claude Agent SDK overview, agent loop, subagents, hooks, permissions, sessions, MCP and skills reference. `code.claude.com/docs`
- Anthropic: model overview and pricing. `platform.claude.com/docs`
- Anthropic and Pearson VUE: Claude Certification Program, exam list, retake policy and Credly badging. `pearsonvue.com/us/en/anthropic.html`
- xAI: Introducing Grok Bot, August 2026 beta announcement. `x.ai/news/introducing-grok-bot`
- Nous Research: Hermes Agent documentation and provider integrations. `hermes-agent.nousresearch.com`
- NVIDIA: Certified Associate Generative AI and LLMs (NCA-GENL) exam page
- Databricks: Certified Generative AI Engineer Associate exam guide
- OpenAI: OpenAI Academy course catalogue and badge policy

C.2 Ecosystem and market snapshots

- Firecrawl: best open-source agent frameworks, 2026 adoption and download figures
- LangChain: AI agent frameworks comparison and selection criteria
- "Every AI Coding CLI in 2026: the complete map", tiered inventory of thirty-plus tools
- Analytics Vidhya: OpenClaw versus Claude Code, origin, deployment model and trade-offs
- "The state of agentic AI standards in 2026": MCP, A2A, WebMCP, OSI, AP2 and ACP, with governance and adoption status
- Linux Foundation: A2A protocol adoption, 150+ organisations, v1.0 and signed Agent Cards
- Comparative surveys of LLM observability and evaluation platforms, 2026
- Dataquest and comparable independent rankings of AI certifications by employer recognition
- Coursera and DeepLearning.AI course pages for the Agentic AI course and the IBM RAG and Agentic AI professional certificate
- Artificial Analysis, Intelligence Index leaderboard, snapshot of 18 September 2026, for the model scores, blended prices and open-weight designations in Chapter 7
- Published 2026 consumer subscription comparisons across Anthropic, OpenAI, Google, xAI, Perplexity and Microsoft, for Chapter 21
- Apple desktop specification and pricing round-ups, and independent Apple Silicon local-LLM benchmark write-ups, for the machine comparison in Chapter 12

C.3 How to keep this document honest

Four things in this manual decay and should be re-checked before you act on them: the leaderboard and price figures in Chapter 7, the subscription tiers in Chapter 21 and the machine prices in Chapter 12, certification eligibility and cost, and the relative standing of frameworks. Everything in Parts I, III and IV, and Appendices A and B, should still be accurate years from now. When a section here contradicts a provider's own documentation, the documentation is right and this page is out of date.

Version 1.2 · compiled 20 September 2026. Personal learning document by Mathieu Tellene. Contains no confidential or proprietary business data. Visual system adapted from the Cabify design language for personal use; this is not a Cabify publication and carries no Cabify endorsement.