

AI Agent Engineering

29 modules with diagrams — every concept visualised for fast recall. Sized for the night before a phone screen.

- ▶ LLM mental model & inference mechanics
- ▶ Tokens, context windows, cost optimization
- ▶ Tool use, MCP, multi-tool orchestration
- ▶ Conversation memory, RAG, advanced retrieval
- ▶ ReAct loop, planning, multi-agent patterns
- ▶ Guardrails (input & output), HITL, circuit breakers
- ▶ Evaluation, tracing, monitoring & observability
- ▶ Agent frontier: A2A, long-horizon, Agent SDK

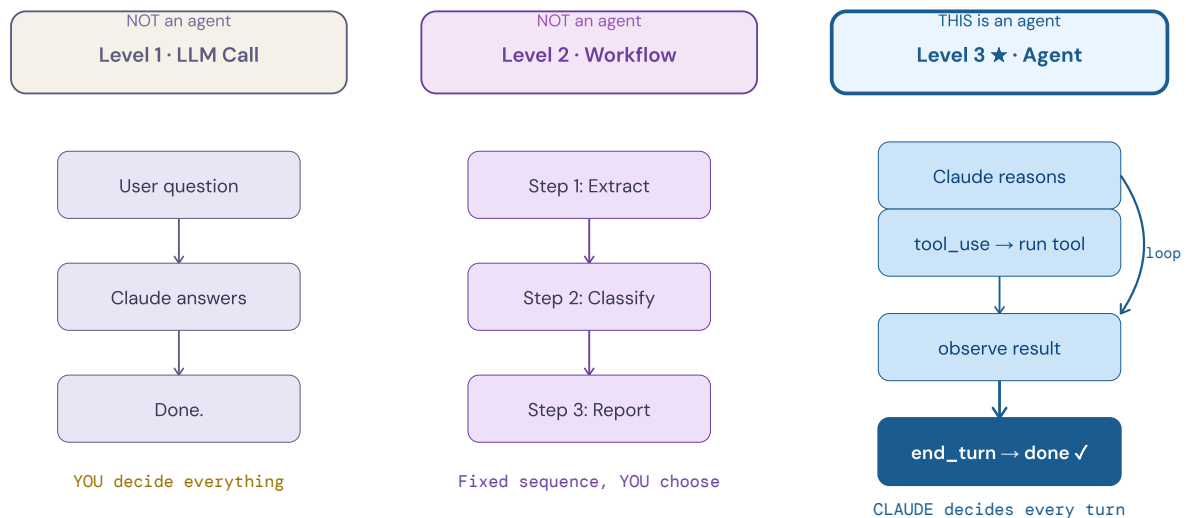
7 Tracks 29 Modules Domain: UCC Lien Risk Pipeline claude-sonnet-4-6

M00

Course Overview & Agent Fundamentals

TRACK 1: FOUNDATIONS

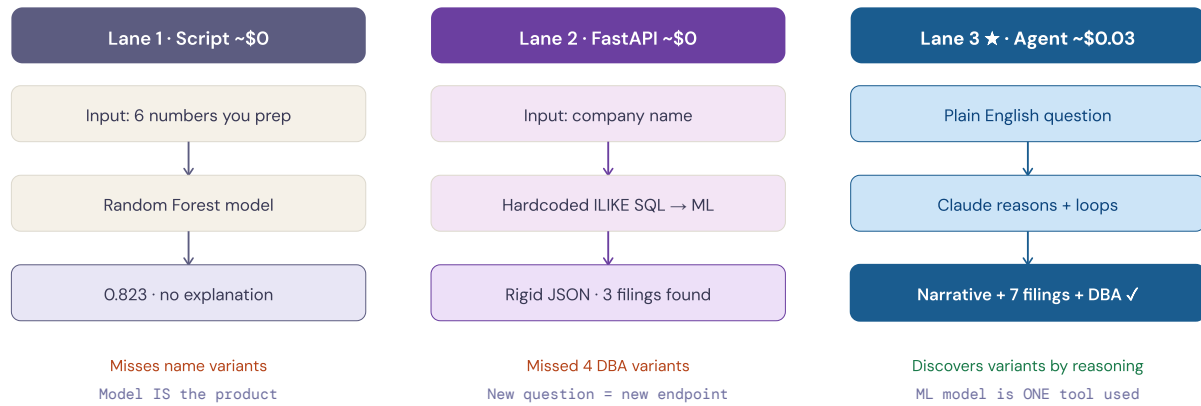
THREE LEVELS OF LLM PROGRAMS — WHO DECIDES WHAT HAPPENS NEXT?



LITMUS TEST

Replace Claude with hardcoded responses. If the program still works — NOT an agent. Level 3 breaks because Claude's runtime decisions can't be hardcoded. That fragility IS the proof.

SAME UCC PROBLEM — THREE WRAPPERS, SMARTER OUTPUT EACH LANE



M00

7 Building Blocks of a Production Agent

ALL TRACKS

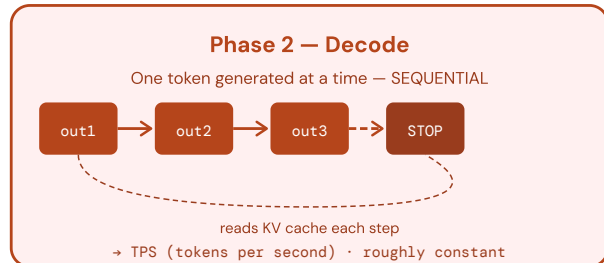
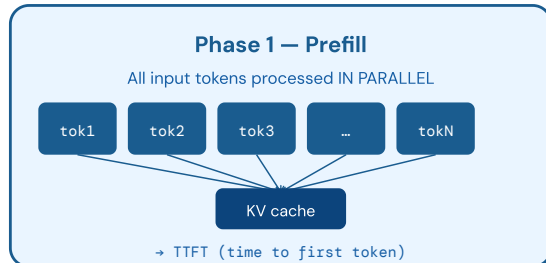
DROP ANY BLOCK AND YOUR AGENT STOPS BEING TRUSTWORTHY



Most tutorials only teach Brain + Tools — that's 2 of 7. This course covers all seven.

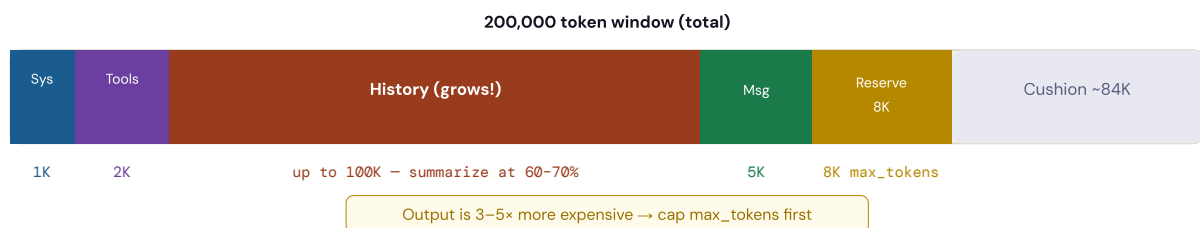
Drop any one block → agent stops being trustworthy

TWO-PHASE INFERENCE: PREFILL (PARALLEL) THEN DECODE (SEQUENTIAL, ONE TOKEN AT A TIME)



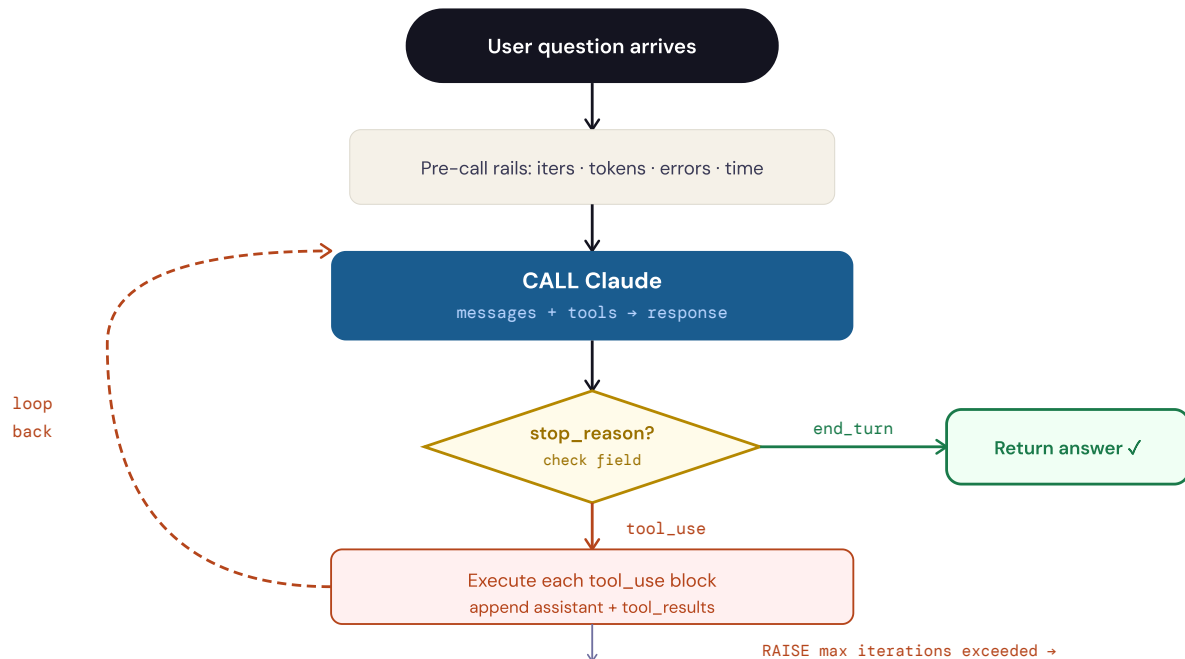
Prompt caching collapses Phase 1 on repeated prompts → 90% cost reduction on cache

200K CONTEXT WINDOW — EVERYTHING SHARES ONE POOL. SUMMARIZE AT 60-70%, NOT 95%.



Overflow = HTTP 400. No graceful fallback. Manage proactively — never reactively.

UNIVERSAL AGENT LOOP – EVERY AGENT IS A VARIATION OF THIS



```

# The universal agent loop – every agent is a variation of this

messages = [user_question]
iters, tokens, errors = 0, 0, 0

WHILE iters < MAX_ITERS AND tokens < BUDGET AND errors < THRESHOLD:

    response = CALL Claude(tools=tool_list, messages=messages)

    messages.append({role: "assistant", content: response.content})
    tokens += response.usage.total

    IF response.stop_reason == "end_turn":                ← happy path ✓
        RETURN final_text(response)

    results = []
    FOR block IN response.content:
        IF block.type == "tool_use":
            out = RUN(block.name, block.input)
            results.append({tool_use_id: block.id, content: out})

    messages.append({role: "user", content: results})      ← BOTH sides required
    iters += 1

RAISE "max iterations exceeded"                          ← safety rail fires

```

5 stop rails Three guards fire *before* each Claude call: iteration cap, token budget, consecutive error threshold. Natural completion (`end_turn`) is the only signal meaning "success" — the others are safety nets for runaway loops.

end_turn **Never parse Claude's text** to detect completion. `stop_reason == "end_turn"` is the only reliable signal — scanning natural language for "I'm done" breaks across model versions and prompt changes.

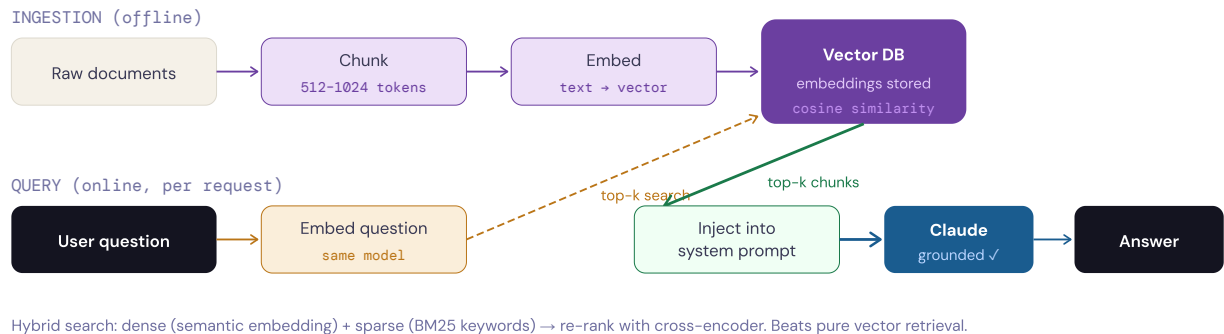
both sides Append *both* the assistant response and the `tool_results` before the next API call. Forgetting either causes the API to reject the request. Each `tool_result`'s `tool_use_id` must exactly match its `tool_use`'s `id`.

safety rail Start with `MAX_ITERS = 10` for most tasks, up to 25 for research agents. A partial result at 8/10 is valuable — return what you have with a note rather than nothing. The rail catches confused loops, not successful completion.

RAG — Retrieval-Augmented Generation

TRACK 3: MEMORY & CONTEXT

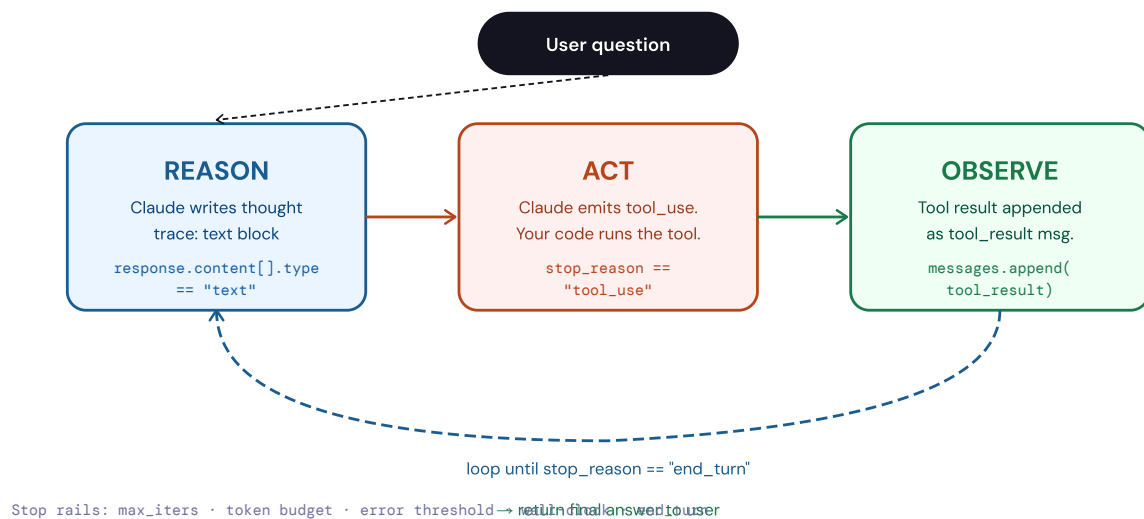
RAG PIPELINE — GROUND ANSWERS IN REAL DOCUMENTS, NOT TRAINING HALLUCINATIONS



The ReAct Loop

TRACK 4: AGENT ARCHITECTURES

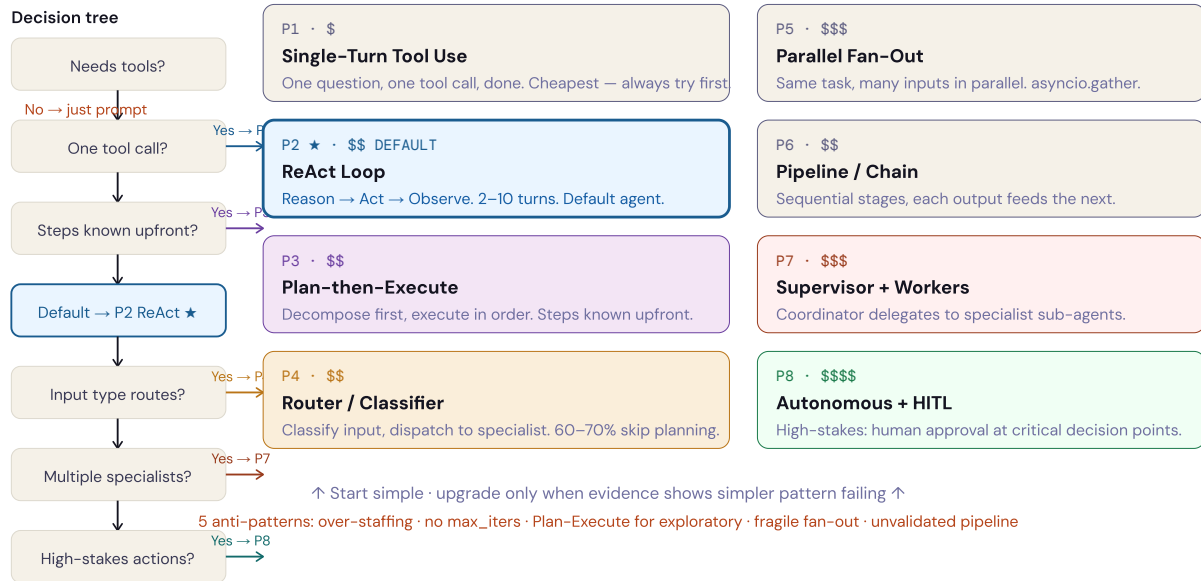
REACT: REASON → ACT → OBSERVE — THE CYCLE THAT MAKES A CHATBOT AN AGENT



8 Agent Patterns — Complexity Ladder

TRACK 4: AGENT ARCHITECTURES

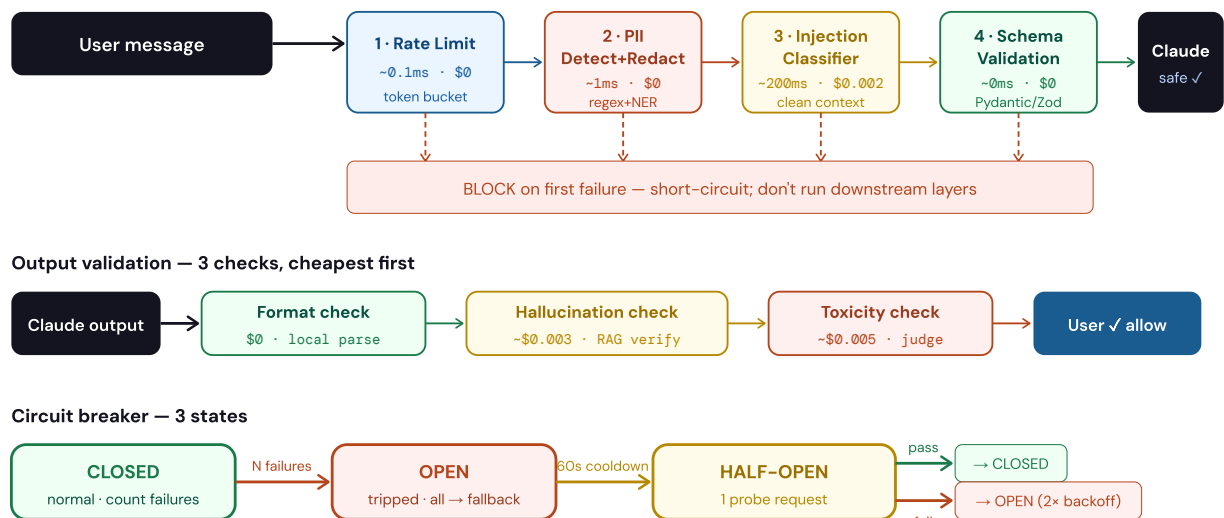
PICK THE CHEAPEST PATTERN THAT FITS THE TASK. PATTERN CHOICE SWINGS API COST 10-50x.



Guardrails — Input & Output

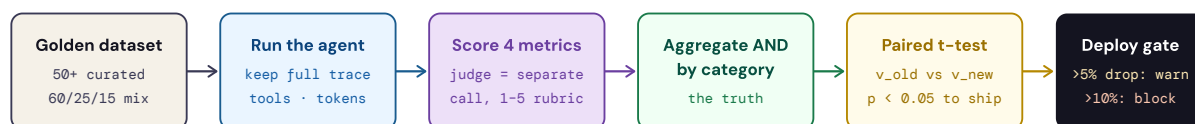
TRACK 5: GUARDRAILS & SAFETY

5-LAYER INPUT GUARDRAIL STACK — CHEAPEST FIRST, FAIL CLOSED ON ERRORS



Agents fail differently from ordinary software, so **assertEqual** can't test them. It breaks three ways: outputs are non-deterministic, the path the agent takes matters as well as the answer, and quality is a spectrum, not pass/fail. The replacement is **rubric scoring on a curated dataset, with statistical thresholds**.

THE EVAL PIPELINE — DATASET IN, DEPLOY DECISION OUT



AGGREGATE HIDES · CATEGORY SHOWS



THREE CI TIERS · AFFORDABLE BY DESIGN

1 · Every commit mocked LLM · <30 s · \$02 · PR merge 10 cases, real API · ~3 min · ~\$0.053 · Nightly 02:00 UTC 100 cases + full rubric · ~\$0.50

Budget cap \$300/mo · alert at 80% · pause + page at 100%

Full 200-case suite on every commit ≈ \$9K/month at 30 PRs/day — that's why the tiers exist.

THE RUBRIC — FOUR METRICS, STRATIFIED BY CATEGORY

METRIC	WHAT IT ASKS	HOW IT'S SCORED	WHAT IT CATCHES
Task completion	Did the agent achieve the stated goal?	Binary or partial credit (0–100%) across the suite	The headline "did this thing work?" number
Tool accuracy	Right tools, right parameters?	Trace compared with expected tools, fuzzy-matched (parameter order varies)	Right answer via the wrong path: brittle, expensive, a time-bomb
Response quality	Clarity, completeness, tone, formatting	1–5 by Claude-as-judge against a written rubric	The subjective dimension a binary test can't see
Efficiency	Tokens, tool calls, latency	Lower is better at equal quality	4.0 quality at 2K tokens beats 4.2 at 18K for production traffic
Category (tag)	Which kind of case was this?	Every case tagged, and every metric computed per category	The masked failure: the aggregate is for the deck, the category view is for the deploy

WRITING THE 1-5 LEVELS

Give **every** score level explicit criteria, e.g. 5 = *fully achieved with correct info*; 3 = *partial, significant gaps*. If you can't write the levels, the rubric is too vague to use.

Test it on obviously bad outputs. If those still score 3+, sharpen the level criteria. A useful rubric has a **wide score distribution**, not everything at 4 or 5.

THE JUDGE, IN FIVE STEPS

1) Write a tight rubric. 2) Build the judge prompt: task, output, rubric, and **"reason first, then score"**. 3) Run v1.0 and v1.1 on the same 50–200 cases. 4) Validate the judge against human grades (> 0.7 correlation), re-checked whenever the judge prompt, rubric or model changes. 5) Paired t-test, then decide.

THRESHOLDS TO REMEMBER

CHECK	THRESHOLD	IF IT FAILS
Judge vs human grades	> 0.7 correlation	Fix the judge prompt or rubric before trusting any score
A/B verdict	Paired t-test $p < 0.05$ and $\text{delta} > 0$	"Insufficient evidence": add cases or make a real change. +0.3 on 50 noisy cases is often luck
Inter-annotator agreement	$> 80\%$ on gold labels	The rubric is the bug: sharpen it, then relabel
Regression vs baseline	$> 5\%$ drop warns · $> 10\%$ blocks	Block the deploy. A uniform 3% drop on every metric is a red flag too
Flaky case	Retry up to 3×	A real failure only if it fails 2 of 3
Dataset mix	60% common · 25% edge · 15% adversarial	A happy-path-only suite won't detect degradation at the edges

```

FUNCTION llm_judge(task, output, rubric):
    prompt = "You are a judge. Score using this rubric.
        TASK: {task} OUTPUT: {output} RUBRIC: {rubric}
        REASON FIRST, then return JSON {reasoning, scores}."
    RETURN parse_json(claude.call(prompt)) # SEPARATE call, clean context

FUNCTION ab_compare(v_old, v_new, suite, rubric):
    delta = AVG(scores_new) - AVG(scores_old)
    p = paired_t_test(scores_old, scores_new)
    IF p < 0.05 AND delta > 0: RETURN "ship v_new" ← real win
    RETURN "insufficient evidence"

# An eval case stores expected BEHAVIOUR, never an exact output
case = { id: "tc-07",
        input: "Look up 'ACME CORP' vs 'Acme Corporation' – same?",
        expected_behavior: "Recognize entity variants, search both, suggest EIN check",
        category: "entity_resolution", difficulty: "hard", version_added: "v1.3" }

```

⚠ THREE TRAPS

"Aggregate ≥ 4.0 means we're shipping." Read the per-category breakdown first.

"Store exact outputs and assertEquals them." That breaks on day one. Store behaviour and let the rubric judge.

"Build the dataset once." Every production bug and user complaint becomes a new, versioned case. Curated 50 beats random 1,000.

CARDS

30 Interview Flash Cards

REVIEW THESE BEFORE THE CALL

Q: What is a true AI agent? (technical)

A program combining: (1) LLM as reasoning engine, (2) tools to interact with the world, (3) a loop that keeps it thinking and acting until the task is complete. The *loop* is the entire agent.

Q: Single question separating agents from workflows

"Who decides what happens next — your code, or Claude?" Level 3 = Claude decides at runtime which tool, which args, when to stop. Your code provides tools; Claude provides the brain.

Q: Litmus test for a true agent

Replace LLM with hardcoded responses. If program still works — NOT an agent. Level 3 breaks because Claude's runtime decisions can't be hardcoded. That fragility IS the proof.

Q: Two phases of LLM inference

Prefill: all input tokens processed in parallel; drives TTFT; scales with prompt length. Decode: one token at a time; drives TPS; ~constant per model. Prompt caching collapses repeated prefill.

Q: Default temperature for agents and why

temp=0. At temp 0.8, an agent routing 5K emails/day produces 150–400 unpredictable routing decisions. Reproducibility, debuggability, auditability beat surprise in agentic systems.

Q: Where to cut tokens — priority order

1) Cap max_tokens (output is 3–5× more expensive). 2) Trim system prompt (sent every call). 3) Summarize history at 60–70% utilization. 4) RAG instead of full docs. 5) Tier down to Haiku.

Q: The universal agent pattern (pseudocode)

```
messages=[question]. WHILE True: response=CALL Claude. IF end_turn → RETURN text. FOR each tool_use → run tool, append result. LOOP. Plus 5 stop rails (iters, tokens, errors, time, natural).
```

Q: #1 tool-use bug to watch for

Forgetting to append BOTH the assistant response AND the tool_results before the next API call. Also: each tool_result's tool_use_id must exactly match the tool_use's id.

Q: How many tools per request? Why?

Cap at 4–6. Selection accuracy: 97% at 3 tools → 61% at 18 tools. Each schema burns 200–500 input tokens. Beyond 6, split into specialized subagents (M14).

Q: Tool description — what must it include?

WHAT it does (strong verb + concrete object). WHEN to use vs. sibling tools. WHAT it returns (shape/format). Each property also needs its own description with an example value.

Q: Empty result vs error — why does it matter?

{results:[]} = "I checked, nothing matched" → Claude stops. {isError:true} = "I couldn't even check" → Claude asks for help. Conflating causes catastrophic decisions.

Q: MCP vs A2A — one sentence each

MCP = agent↔tool (vertical, Anthropic Nov 2024): how one agent connects to its tools. A2A = agent↔agent (horizontal, Google→Linux Foundation 2025): cross-org delegation with auth. Real systems use BOTH.

Q: 5 stop conditions for a ReAct loop

1) Natural completion (end_turn) — ONLY signal meaning success. 2) Max iterations (5–15). 3) Token/cost budget. 4) Error threshold (2–3 consecutive failures). 5) Wall-clock timeout (30s–2min).

Q: Why thought traces matter

The ACT of generating reasoning text improves Claude's next decision by 20–30%. Don't suppress to save tokens. Do summarize old entries in long sessions. Also provides audit log for compliance.

Q: Plan-Execute vs ReAct — cert exam trap

Plan-Execute: steps can be known before ANY tool runs — brittle when reality differs. ReAct: steps emerge from results — adapts to surprises. Choose by whether steps are knowable before execution.

Q: 8 agent patterns in order of complexity

P1 Single-Turn · P2 ReAct · P3 Plan-Execute · P4 Router · P5 Fan-Out · P6 Pipeline · P7 Supervisor+Workers · P8 Auto+HITL. Pattern choice can swing API bill 10–50× in production.

Q: Subagent vs sub-prompt

A subagent is a separate agent invocation: own system prompt, own small toolset, blank context window, returns structured JSON. Isolation IS the feature. Never let subagents call each other directly.

Q: The coordinator's 4 jobs

1) Route (which subagents). 2) Brief (self-contained task string, resolve pronouns). 3) Collect (structured result, handle errors). 4) Synthesize (one answer). Coordinator owns history; subagents don't.

Q: Input guardrails — 5 layers in order

1) Rate limit (0.1ms, \$0). 2) PII detect+redact (regex+Luhn+NER). 3) Injection classifier (separate clean-context Claude call). 4) Schema validation (Pydantic, \$0). 5) Allow/deny/log. Fail CLOSED on errors.

Q: Why prompt injection is like SQL injection

Same channel, same vulnerability: data and instructions share one untrusted channel. Fix lives OUTSIDE the model — separate-context classifier on every input AND every tool result (indirect injection).

Q: Output guardrails — 3 checks, cheapest first

1) Format (\$0 local parse). 2) Hallucination (~\$0.003 RAG-grounded verifier). 3) Toxicity (~\$0.005 Claude-as-judge). Skip downstream if upstream blocks. Three verdicts only: allow, redact, block.

Q: 3 HITL gates and when to use each

Approval: irreversible actions. Modification: output needs human voice. Escalation: policy gap/capability limit. Tier on blast radius and reversibility — NOT transaction volume or model confidence score.

Q: Circuit breaker — 3 states

CLOSED (normal) → N failures → OPEN (all→fallback, 60s cooldown) → HALF-OPEN (1 probe) → pass→CLOSED or fail→OPEN with 2× backoff. Different thresholds per failure type.

Q: Why assertEquals fails for agent testing

3 breaks: non-determinism (same prompt → different output), trajectory matters (15 tools for 2-tool job = broken), quality is spectrum not binary. Use rubric scoring on curated golden dataset stratified by category.

Q: 4 evaluation metrics for agents

Task completion rate. Tool accuracy (right tool, right params). Response quality 1–5 (Claude-as-judge). Efficiency (tokens, calls, latency). ALWAYS stratify by category — aggregate hides broken subcategories.

Q: RAG pipeline — 5 steps

1) Ingest (chunk, embed, store). 2) Query (embed question, top-k cosine). 3) Augment (inject chunks into prompt). 4) Generate (Claude answers grounded). 5) Verify (check claims vs sources). Hybrid search beats pure vector.

Q: "Lost in the middle" effect

Claude's attention is strongest at START (system prompt) and END (latest message). Facts buried in a long history get less attention. Put critical instructions at START or END — never in the middle of long docs.

Q: Rate limiting — token bucket mechanics

Bucket holds N tokens; each request consumes 1; refills at R/second. Burst-friendly. Implement per user, per endpoint. Sliding window not fixed window (exploitable boundary). Back with Redis in production.

Q: Observability — print statements vs production

Print statements vanish on restart. Production = structured JSON logs + distributed traces + dashboards. Key: answer "why did Claude pick tool X on request Y at 3:14pm last Tuesday?" with trace_id.

Q: 60-second "whole arc" talking point

"Transformers (2017) understanding. GPT-3 (2020) generating. ChatGPT (2022) mainstream. LLMs alone = chatbots. Agentic AI (2024) adds tool-use, memory, loop, 240× cost drop. I build agents combining ML, RAG, tools, guardrails, observability."

200K

Claude context window — system + tools + history + response share one pool

240×

Cost reduction since GPT-3 2022 (\$0.06/1K → \$0.00025/1K Claude 3 Haiku)

4–6

Max tools per request (97% selection accuracy at 3 → 61% at 18 tools)

60–70%

Context window utilization trigger for history summarization (never wait for 95%)

3–5×

Output tokens more expensive than input — cap max_tokens first always

20–30%

ReAct (with Reason step) beats action-only on multi-step tasks

FINAL TAKEAWAY

"An agent is your existing stack with a reasoning layer on top. Same models, same FastAPI, same Docker. New brain. The student who understands the raw loop can learn ANY framework in a day. The student who learned LangChain first often cannot debug without it."