FREE PREVIEW · ~30% OF THE FULL GUIDE

Unofficial Study Guide for the CCAR-P Exam

Claude Certified Architect — Professional · sample edition

CCAR-P | 63 Q · 120 MIN | PASS 720/1000

An independent study resource by Zehon.com · August 2026
This free preview contains complete, unabridged sections of the full guide: Day 1 of the 7-day plan, all of Domain 1, and 8 practice questions with full answer explanations.

The 7-day plan

Read a domain, get your hands dirty, checkpoint with the quiz — every day.

~2 hours a day for 7 days: each day is ~70 min guided study + ~40 min hands-on + ~10 min checkpoint. The hands-on exercises matter more than re-reading — CCAR-P tests judgement, and judgement is built by making design decisions and defending them. Keep one running document (a **decision journal**) where you write every exercise output; by Day 7 it doubles as your revision sheet.

Materials: this guide · the 40-question quiz bank · your decision journal (started Day 1, revised daily)

One running claims-assistant scenario threads through every day's hands-on work; the Day 7 quiz tells you whether to book.

Day 1 — Solution Design & Architecture, part 1 (Objectives 1–3)

Study (70 min): guide Domain 1, objectives 1–3. Memorize the pattern taxonomy cold: augmented LLM; the five workflow patterns (prompt chaining, routing, parallelization sectioning/voting, orchestrator-workers, evaluator-optimizer); agents. For each: one-line definition, when to use, main failure mode — and internalize the simplicity principle, which breaks ties throughout this guide.

Hands-on (40 min) — architecture sketch from a brief: *"An insurer wants to cut claims-intake handling time. Adjusters read emailed claim descriptions plus attached photos and police reports, extract structured facts, check policy coverage, and either fast-track or route to a senior adjuster. 4,000 claims/day; coverage decisions are regulated; wrong fast-tracking is expensive."*

Draw input → processing → output → feedback. Label every box with a pattern from the taxonomy and every arrow with what flows; mark deterministic gates vs probabilistic steps vs human gates. Then write 3 sentences: which pattern dominates and why NOT an autonomous agent.

✓ **CHECKPOINT**

Without notes, list all 5 workflow patterns with a one-line "use when." State the workflow-vs-agent decision rule. If you can't, repeat tomorrow's warm-up with Day 1 material.

🔒 **IN THE FULL EDITION — THE REST OF THE PLAN**

- Day 2 — Solution Design part 2 + Models/Prompting/Context (Objectives 4–9)
- Day 3 — Integration (Objectives 10–13) — a broad domain, give it full weight
- Day 4 — Evaluation & Testing (Objectives 14–17)
- Day 5 — Governance, Safety & Risk (Objectives 18–21)
- Day 6 — Stakeholder Communication + Lifecycle (Objectives 22–28)
- Day 7 — Full quiz + review + exam logistics

The study guide

Seven domains, twenty-eight objectives — judgement, patterns, and trade-offs rather than API mechanics.

CCAR-P tests architectural *judgement*, not API mechanics — the published objectives frame nearly everything as "given these constraints, which pattern / trade-off / control, and why." Governance is a first-class design input: a compliance constraint in a scenario is never decoration, it eliminates options. Everything below maps to the 7 domains and 28 objectives, ending with the rapid-fire self-test, glossary, and the decision tables most items reduce to.

EXAM SNAPSHOT

63 Q ANSWER EVERYTHING	**7** DOMAINS · 28 OBJECTIVES	**3** QUESTION TYPES	**120 min** CLOSED BOOK
720 PASS, 100–1000 SCALE	**$175** LISTED PRICE		

Question types: (1) standard multiple choice, (2) multiple response "select 2 of 5", (3) scenario-matching with dropdowns (match patterns/controls to scenarios). No guessing penalty — never leave a blank.

THE 7 DOMAINS · OBJECTIVE COUNT PER DOMAIN

Anthropic hasn't published official per-domain weights, so treat every domain as fair game; bars show each domain's objective count of the 28.

⚖ **THE SINGLE MOST REUSABLE HEURISTIC IN THIS GUIDE**

From Anthropic's "Building Effective Agents": **start with the simplest thing that meets the requirement — a single augmented LLM call — and add workflow structure, then agency, only when simpler solutions demonstrably fall short.** When two options both work, choose the simpler, more observable, more governable one. When a stated constraint (SLA, budget, regulation, team skill) is violated by an option, that option is wrong no matter how elegant.

Domain 1 · Solution Design & Architecture OBJECTIVES 1–6

Translating business problems, end-to-end pipeline design, the pattern taxonomy (the heart of this domain), multi-agent orchestration, decomposition, and business value pillars.

1 · Translate business problems into Claude-based AI solutions

Take a fuzzy business statement ("support costs are too high", "contract review takes weeks") and turn it into a well-scoped AI solution. Identify the task type (classification, extraction, generation, summarization, agentic action), the success metric in business terms, the data available — and whether an LLM is even the right tool.

Fit test first

LLMs suit tasks that are language-heavy, tolerant of probabilistic output (or checkable), and expensive for humans at scale. Deterministic calculations, exact lookups, and hard real-time control loops belong in conventional code — an LLM can *orchestrate* them via tools but shouldn't *perform* them.

Decompose the business problem

Separate the parts that need judgement (LLM) from the parts that need correctness (code/tools/database) and the parts that need accountability (human).

Success metric translation

"Reduce handle time" → measurable proxy (first-contact resolution %, deflection rate) → eval metric (accuracy on a golden set, escalation precision). If you can't define what "good" looks like, you can't evaluate it — scope discovery until you can.

Buy / build / configure spectrum

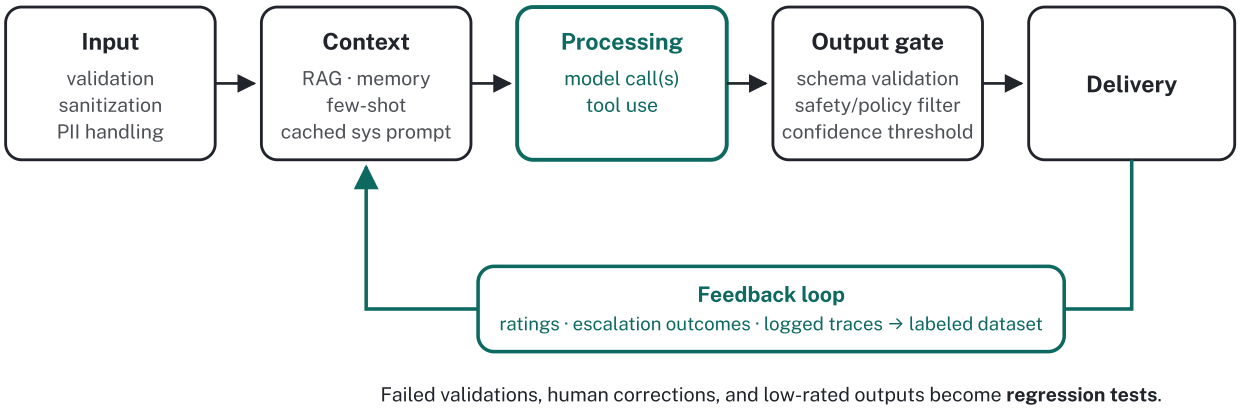
Claude.ai + Projects for knowledge-worker workflows; Claude Code for developer workflows; the API (with workflows/agents) for product features. Don't build an agent platform when a configured product solves it.

⚖ **DECISION CUES**

A task that is fundamentally arithmetic, exact retrieval, or a compliance-mandated deterministic decision → route that piece to a tool/rules engine, not to the model. If the stem gives a business KPI, the right answer traces the architecture back to that KPI, not to a technology preference.

2 · Design end-to-end architectures (input → processing → output → feedback loops)

Sketch the full pipeline, not just the model call: ingestion and preprocessing, retrieval/context assembly, the LLM step(s), output validation and formatting, delivery, and the feedback loop that improves the system over time.



The canonical pipeline. An architecture diagram with no feedback path is incomplete — the feedback loop is part of the architecture, not an afterthought.

Deterministic gates

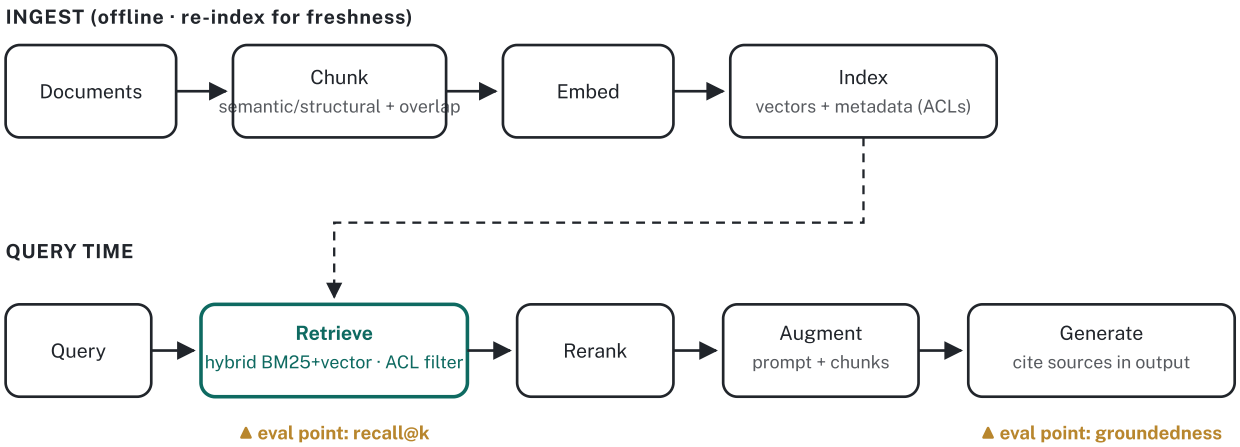
JSON schema validation, regex/allowlist checks, business-rule assertions — wherever correctness is binary. Cheap, fast, and auditable; prefer them at system boundaries.

Probabilistic gates

LLM-as-judge scoring, classifier confidence thresholds — where quality is graded rather than binary.

Where state lives

Conversation memory (in-context vs summarized vs external store), idempotency for retried tool calls, and audit logs.



Metadata filtering at query time is crucial for **access control** — otherwise the vector store bypasses permissions.

RAG when knowledge is large / changing / permissioned; long-context stuffing when the corpus is small and stable (and cacheable). Fine-tuning is rarely the right answer in these scenarios — prefer retrieval + prompting.

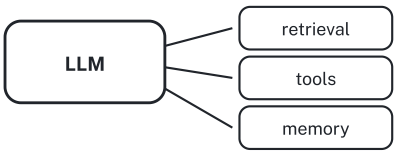
⚖ **DECISION CUES**

An option that adds output schema validation + a fallback path usually beats an option that "improves the prompt" for reliability questions. "Knowledge changes weekly/daily" or "per-user document permissions" → RAG with metadata/ACL filtering — not fine-tuning, not stuffing everything into context.

3 · Select appropriate architectural patterns (workflow, agentic, augmented LLM)

The core Anthropic taxonomy from "Building Effective Agents". Know each pattern, when it applies, and its failure modes — treat this taxonomy as the heart of the domain.

Augmented LLM
the basic building block

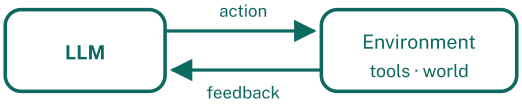


one well-prompted call —
the default when it meets the bar

Workflow
predefined code paths

predictable · testable · debuggable
cost-bounded · auditable steps

Agent
directs its own loop



open-ended · unpredictable step count
needs sandboxing, stop conditions,
checkpoints, human oversight

Escalate downward only when the simpler level **demonstrably falls short**.

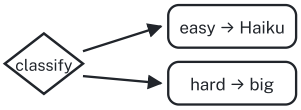
What each level adds. Agents cost higher latency, higher and less predictable spend, and compounding errors — so they require guardrails and human oversight for consequential actions.

The five named workflow patterns — LLMs and tools orchestrated through predefined code paths:

1 · Prompt chaining

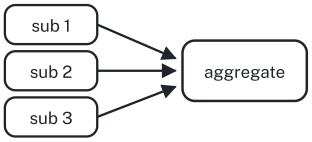
fixed sequence + gates between steps
trades latency for per-step accuracy

2 · Routing



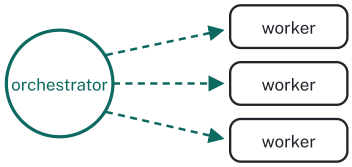
distinct input categories,
specialized prompt/model per lane

3 · Parallelization



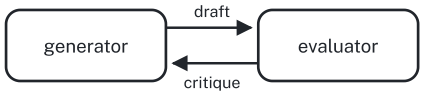
sectioning cuts latency ·
voting buys confidence at N× cost

4 · Orchestrator-workers



lead LLM decomposes dynamically —
use when subtasks can't be predicted

5 · Evaluator-optimizer



clear criteria + iteration measurably helps —
bound the loop (max iterations) to cap cost

The five workflow patterns. Orchestrator-workers differs from parallelization in that decomposition is dynamic, not fixed in advance.

PATTERN	USE WHEN	WATCH OUT
Prompt chaining — fixed sequence of calls, each consuming the previous output, optionally with programmatic checks ("gates") between steps	Task decomposes cleanly into fixed subtasks	Trades latency for accuracy per step
Routing — classify the input, dispatch to a specialized prompt/model/path	Inputs fall into distinct categories better handled separately; enables cheap-model fast paths (Haiku for easy cases, bigger model for hard ones)	Classifier quality bounds the system
Parallelization — *sectioning* (independent subtasks concurrently, results aggregated) and *voting* (same task multiple times, results compared/majority)	Sectioning: independent subtasks + latency pressure. Voting: confidence on one high-stakes output	Voting multiplies cost
Orchestrator-workers — a central LLM dynamically decomposes the task, delegates to workers, then synthesizes	Subtasks *can't be predicted in advance* (differs from parallelization, where decomposition is fixed)	Delegation quality is the failure point
Evaluator-optimizer — generator LLM + evaluator LLM in a refine loop	*Clear evaluation criteria* exist and iteration measurably helps (translation, search-and-refine)	Bound the loop (max iterations) to cap cost

⚖ **DECISION RULE**

Predictable steps → workflow. Unpredictable steps but bounded, checkable goal → agent with guardrails. One step suffices → augmented LLM. Always the **simplest** pattern that meets the requirement.

✕ **CUE PHRASES — PATTERN MATCHING**

"Fixed, well-defined steps / needs predictability / audit every step" → workflow (usually chaining or routing). "Can't predict how many steps / open-ended research / varies per request" → agent or orchestrator-workers. "Distinct input categories" → routing. "Independent subtasks + latency pressure" → parallel sectioning. "Clear rubric + iterative polish" → evaluator-optimizer. An option proposing an autonomous agent for a task with three fixed steps is **over-engineering — wrong**.

4 · Design multi-agent systems and orchestration strategies

When one agent isn't enough: how to split work across agents, coordinate them, and keep the system debuggable.

- **When multi-agent is justified:** the task exceeds one context window; subtasks need genuinely different tool sets/permissions/system prompts; parallel exploration materially helps (research, breadth-first search). Anthropic's multi-agent research system uses a **lead/orchestrator agent + parallel subagents**, each with its own context window, returning condensed findings with citations.
- **Coordination patterns:** orchestrator-workers (lead decomposes, workers execute, lead synthesizes); pipeline hand-offs (agent A's artifact is agent B's input); blackboard/shared-memory (agents read/write a common store). Prefer explicit hand-offs with structured artifacts — easier to trace than free-form chatter between agents.
- **Context isolation is the point:** each subagent gets only what it needs — least context means fewer tokens, fewer distractions, better security.
- **Subagents in Claude Code** mirror this: specialized subagents with scoped tools and their own context, invoked by a main session.

✕ **COSTS AND THE #1 FAILURE MODE**

Token multiplication (multi-agent research burns ~15× the tokens of single-chat), coordination failures, duplicated work, inconsistent conclusions. **Vague delegation is the #1 multi-agent failure mode** — the orchestrator's job includes task-description quality. Mitigate with clear task boundaries, structured output contracts between agents, budget caps per subagent, and tracing that stitches the whole run together.

⚖ **DECISION CUES**

"Research across many sources / breadth" → orchestrator + parallel subagents with citations. Cost sensitivity in the stem → the answer acknowledges multi-agent token multiplication and caps or narrows scope — a "spawn more agents" option under a tight budget is wrong. Structured hand-off contracts beat "agents converse freely."

5 · Apply decomposition techniques for complex problem solving

Breaking a big problem into LLM-sized pieces — the design skill underlying chaining, routing, and orchestration.

By stage

Draft → critique → revise.

By category

Route by input type.

By section

Independent chunks in parallel.

By capability

Extraction vs reasoning vs formatting — assign each to the cheapest adequate model.

By risk

Low-risk automated; high-risk gated to humans.

- **Interface contracts between steps:** each step emits structured output (JSON schema) that the next step validates — this converts a fuzzy pipeline into checkable units and localizes failures.
- **Checkable intermediate states:** decompose so intermediate outputs are *verifiable* — a list of extracted clauses can be spot-checked; a monolithic "review this contract" answer cannot.
- **Map-reduce for scale:** chunk → per-chunk extraction (cheap model, parallel) → aggregate/synthesize (stronger model). The standard answer for "documents too large for context" or "10,000 documents nightly."

✕ **KNOW WHEN NOT TO DECOMPOSE**

Each step adds a call (latency + cost) and a lossy boundary. If a single call with good prompting hits the quality bar, decomposition is over-engineering.

⚖ **DECISION CUES**

"Accuracy on step X is poor inside a monolithic prompt" → split X into its own focused call with validation. "Volume/scale" → map-reduce with a small model for the map. If the stem says a single prompt already meets the quality bar, the answer defending the status quo (don't decompose) is right.

6 · Align solutions to business value pillars

Every architecture choice should be justified against a value pillar, and pillars can conflict — practice optimizing for the pillar *the scenario names*.

Efficiency

Same output, less resource.

Productivity

More output per person.

Transformation

New capabilities and business models.

Cost

Unit economics: cost per ticket, per document.

SLAs

Latency, availability, quality floors as contractual commitments.

- **Unit economics thinking:** compute cost per transaction (tokens × price, both directions, cache-adjusted) and compare against the value of the transaction. A $0.30 model call replacing a $6 human touch is a win; the same call on a $0.01-margin transaction is not.
- **SLA-driven design:** a latency SLO dictates model tier, streaming, caching, and pattern choice (parallel over serial chains); an availability SLA dictates fallbacks (secondary model/provider, graceful degradation to "escalate to human"); a quality SLA dictates eval gates before release.
- **Phasing for value:** ship the highest-value/lowest-risk slice first (assist mode → suggest mode → auto mode); "transformation" plays earn investment via measured productivity wins.

⚖ **DECISION CUES**

When options differ in cost and quality, pick the one satisfying the *stated* SLA/budget — not the highest quality absolutely. "Justify to the CFO" → answer in unit economics (cost per outcome), not tokens or model names.

🔒 **IN THE FULL EDITION — SIX MORE SECTIONS OF THIS DEPTH**

- Domain 2 · Models, Prompting & Context Engineering
- Domain 3 · Integration
- Domain 4 · Evaluation & Testing
- Domain 5 · Governance, Safety & Risk
- Domain 6 · Stakeholder Communication
- Domain 7 · Lifecycle Management
- Rapid-fire self-test
- Glossary of exam terms
- Quick-reference decision tables

Practice bank — 40 questions

40 original, exam-style practice questions with full explanations — written to the published objectives, not taken from any exam.

Work them by domain as checkpoints (the 7-day plan says when), then re-run your misses on day 7. The answer key starts on its own page after the last question — answers follow on the next page.

Solution Design & Architecture

Q1 MEDIUM

A legal-tech firm processes intake contracts through exactly three steps: extract key clauses, check each clause against a playbook, and generate a summary memo. Compliance requires that every intermediate result be auditable, and the steps never vary. Which architecture best fits?

A. An autonomous agent with clause-extraction, playbook, and drafting tools that decides its own steps

B. A prompt-chaining workflow with schema validation gates between the three fixed steps

C. An orchestrator-workers system where a lead model dynamically decomposes each contract

D. A single large-context call that performs all three steps in one prompt

Q2 MEDIUM

A support automation system receives password resets (60% of volume, trivial), billing questions (30%, moderate), and complex technical escalations (10%). Leadership wants lower cost without degrading quality on hard cases. Which pattern applies?

A. Run every request through the strongest available model to guarantee quality

B. Use a small model for everything and accept lower quality on escalations

C. Classify each request first and route it: small fast model for trivial categories, stronger model for complex ones

D. Run all three category prompts in parallel and merge the answers

Q3 HARD

A research assistant must answer questions like 'compare the regulatory exposure of these five acquisition targets,' where the sources, number of lookups, and lines of inquiry differ unpredictably per request. Latency of a few minutes is acceptable. Which pattern fits best?

A. Prompt chaining with a fixed sequence of search-then-summarize steps

B. A routing layer that sends each question to one of five pre-written research prompts

C. Evaluator-optimizer: one model drafts the answer while another critiques it until quality converges

D. Orchestrator-workers: a lead model decomposes each question and delegates research subtasks to parallel worker agents that return cited findings

Q4 EASY

An internal Q&A assistant must answer from a policy corpus that changes weekly, and each employee may only see documents their role permits. Which knowledge strategy is most appropriate?

A. Fine-tune a model on the current policy corpus and retrain weekly

B. Paste the entire corpus into every prompt using a long-context model

C. RAG: index the corpus, retrieve per query with metadata filters that enforce the requesting user's access permissions

D. Have the model answer from general knowledge and flag uncertain answers

Q5 EASY

The CFO asks whether the proposed Claude-based claims triage system is worth funding. Which framing best supports the decision?

A. Token throughput and cache hit rates projected for the first quarter

B. Cost per claim processed today versus projected cost per claim with the system, including human review time for the cases it escalates

C. The model family's benchmark scores compared with competitors

D. The number of architectural patterns the design incorporates

Q6 MEDIUM

A team proposes a five-agent pipeline with an evaluator-optimizer loop to generate one-paragraph product descriptions from structured attribute data. A prototype using a single well-prompted model call with three few-shot examples already meets the quality bar in evals. What should the architect recommend?

A. Ship the single augmented-LLM call; add structure only if evals show the simpler design falling short

B. Build the five-agent pipeline anyway to leave headroom for future requirements

C. Add just the evaluator-optimizer loop as a safety margin on quality

D. Replace the few-shot examples with a fine-tuned model for robustness

Q7 HARD SELECT TWO

A multi-agent research feature works well but burns roughly 15x the tokens of a single-chat interaction, and finance has flagged the cost. Select TWO changes that best control cost while preserving the multi-agent design's value.

A. Set explicit token/turn budgets per subagent and cap the number of subagents spawned per query

B. Let subagents converse freely with each other so they can self-organize more efficiently

C. Require the orchestrator to delegate with tightly scoped task descriptions and have workers return condensed, structured findings instead of full transcripts

D. Upgrade every subagent to the largest model so each finishes in fewer turns

E. Remove the orchestrator and let all subagents run unconditionally on every query

Integration

Q8 MEDIUM

A customer-service agent uses a get_orders tool backed by a service account that can read every customer's orders. The system prompt says: 'Only ever show users their own orders.' A penetration test coaxes the agent into revealing another customer's data. What is the correct architectural fix?

A. Strengthen the system prompt with firmer, repeated instructions about data access

B. Add an output filter that scans responses for other customers' names

C. Propagate the authenticated end user's identity to the tool and enforce authorization in the order service itself, so the tool can only return that user's records

D. Fine-tune the model to refuse cross-customer requests

Answer key & explanations

Read every explanation, even for questions you got right — the distractor rationales are training too.

Q1 → B The steps are fixed and known in advance, and auditability of intermediate results is required — the definition of a prompt-chaining workflow with deterministic gates. An agent or orchestrator adds unpredictability with no benefit (the subtasks ARE predictable), and a single monolithic call produces no auditable intermediate states. Rule: predictable steps plus audit requirements point to workflows, not agents.

Q2 → C Distinct input categories with different difficulty is the textbook routing scenario: a cheap classifier dispatches to specialized paths, so 60% of traffic runs on a small model while hard cases keep the strong one. Option A violates the cost constraint, B violates the quality constraint, and D (parallelization) multiplies cost for no benefit since the categories are alternatives, not independent subtasks.

Q3 → D The subtasks cannot be predicted in advance — they depend on each question — which is precisely what distinguishes orchestrator-workers from fixed chaining or routing. Parallel subagents with their own context windows returning condensed, cited findings is Anthropic's published multi-agent research design. Evaluator-optimizer polishes a draft against clear criteria; it doesn't solve dynamic decomposition.

Q4 → C Changing knowledge plus per-user permissions is the canonical RAG case: re-indexing handles freshness, and ACL metadata filtering at query time enforces authorization. Fine-tuning is slow, expensive, and cannot enforce per-user access; full-corpus stuffing ignores permissions and wastes tokens; answering from general knowledge guarantees staleness and hallucination on internal policy.

Q5 → B Executives fund outcomes: unit economics (cost per claim, before vs after, including the human fallback cost) ties the architecture directly to a business value pillar. Tokens, benchmarks, and pattern counts are engineering inputs, not business justification — translating between those vocabularies is a tested skill.

Q6 → A Anthropic's core guidance: use the simplest solution that meets the requirement, adding multi-step agentic complexity only when simpler approaches demonstrably fall short — and here the evals show the simple call already passes. Every added stage adds cost, latency, and failure surface. 'Headroom' and 'safety margin' without an identified gap are over-engineering, a recurring wrong-answer pattern.

Q7 → A and C Budget caps bound worst-case spend, and tight task scoping with condensed structured hand-offs attacks the actual driver of multi-agent cost: redundant exploration and full-context shuttling between agents. Free-form inter-agent chatter and unconditional spawning increase tokens; upgrading every worker to the largest model raises price per token without bounding volume.

Q8 → C This is the confused deputy problem: the tool's broad service-account privilege is the vulnerability, and prompt instructions are advisory, not a security control — a sufficiently crafted injection will always eventually bypass them. Enforcing authorization in the tool/downstream layer with the end user's identity makes the leak structurally impossible. Output filtering and fine-tuning reduce, but cannot guarantee, containment.

Ready for the rest?

You've just read about 30% of the full guide — at full depth, nothing dumbed down.

The full Unofficial CCAR-P Study Guide includes

- Every domain at the same depth as Domain 1 you just read — diagrams, decision tables, trap pairs, and cue boxes throughout
- The complete 7-day plan: 7 days of guided study, hands-on drills, and checkpoints
- All 40 original practice questions with full answer-key explanations — including the distractor rationales
- The quick-reference sheets and night-before cram material

Get the full edition at `zehon.com`