

FREE PDF

Claude Certified Architect – Foundations (CCAR-F) practice questions

20 free questions with answers and explanations, written to the official exam guide.

Questions	pages 2–21
Check your answers	page 22
Answers and explanations	pages 23–36
Exam tips	page 37

Covers all 5 domains of the exam guide

Agentic Architecture & Orchestration	27%
Tool Design & MCP Integration	18%
Claude Code Configuration & Workflows	20%
Prompt Engineering & Structured Output	20%
Context Management & Reliability	15%

Question 1

SCENARIO · CUSTOMER SUPPORT RESOLUTION AGENT

Setting: an Agent SDK agent that settles customer cases from start to finish (returns, billing disputes, account problems), working through four custom MCP tools: `get_customer`, `lookup_order`, `process_refund` and `escalate_to_human`. Most cases should close in one contact; the rest go to a person.

In normal weeks 9% of `get_customer` searches by email return an empty customers list, nearly all because the customer used a different address. During two CRM outages last month, each lasting about an hour, the MCP server caught every query timeout and also returned an empty list. The agent told 640 customers it found no account and asked them to create a new one.

What change would MOST effectively fix this?

- ☐ (A) Retry the CRM query inside `get_customer` up to three times on a timeout, and return an empty customers list if every attempt fails.
- ☐ (B) Return an `isError` result whenever `get_customer` finds no account, with a message telling the agent to ask the customer for another email.
- ☐ (C) Have the agent call `escalate_to_human` whenever `get_customer` returns an empty customers list, so a person can check the account by hand.
- ☐ (D) Return CRM timeouts as `isError` results marked transient and retryable, and keep returning an empty customers list when a search finds no match.

Note your answers as you go, on paper or in your notes app, then check them all at the answer key.

Question 2

SCENARIO · CUSTOMER SUPPORT RESOLUTION AGENT

Setting: an Agent SDK agent that settles customer cases from start to finish (returns, billing disputes, account problems), working through four custom MCP tools: `get_customer`, `lookup_order`, `process_refund` and `escalate_to_human`. Most cases should close in one contact; the rest go to a person.

Refunds above \$300 need a supervisor. A `PostToolUse` hook on `process_refund` reads the refunded amount and, above \$300, adds context telling Claude to call `escalate_to_human`. An audit found 41 refunds above \$300 last month, each issued and then escalated. Each refund emails the customer at once and incurs a processor fee, and human agents use the same refund backend for larger approved refunds.

What change would MOST effectively fix this?

- (A) Have the `PostToolUse` hook call the payment system's reversal endpoint for any refund over \$300, then tell Claude to escalate the case.
- (B) Move the check into a `PreToolUse` hook on `process_refund` that denies calls over \$300, with a reason telling Claude to escalate the case.
- (C) Configure the refund backend to reject any request over \$300, returning an error result that Claude can act on by escalating the case.
- (D) State the \$300 supervisor limit in the system prompt, with examples of contacts in which the agent escalates the case instead of refunding.

Question 3

SCENARIO · MULTI-AGENT RESEARCH SYSTEM

Setting: on the Agent SDK, a coordinator splits research work among subagents for web search, document analysis, synthesis and report writing, and hands back a report with citations.

Logs show two problems. The synthesis subagent receives eight search subagents' findings pasted together in arrival order, and reports include the first and last subagents' findings in 94% of runs but the middle four subagents' findings in 51%. Separately, long sessions outgrew the context window, so the coordinator's history is now summarized every few turns; after a summary, 22% of delegations drop the user's date cutoff or required currency.

Which two changes should you make? **Choose 2**

- ☐ A Open the synthesis input with a short summary of each subagent's key findings, then give each subagent's full results under its own section header.
- ☐ B Send the coordinator's full conversation history on each request instead of summaries, so the user's date cutoff and currency are kept word for word.
- ☐ C Keep the user's stated constraints, such as date cutoffs and currency, in a block added to each coordinator request outside the summarized history.
- ☐ D Have each search subagent return the full text of the pages it read, so the synthesis subagent can check each finding against the original wording.
- ☐ E Have the coordinator summarize its history less often, every ten turns instead of every few, so the user's refinements stay verbatim for longer.

Question 4

SCENARIO · CLAUDE CODE FOR CONTINUOUS INTEGRATION

Setting: Claude Code runs unattended in a CI/CD pipeline, reviewing pull requests and proposing tests. Its comments must be specific enough to act on, with few false alarms.

The team's review criteria, covering what counts as blocking and which patterns to leave alone, sit in the tech lead's `~/.claude/CLAUDE.md`. The lead's local reviews apply them. The `claude -p` reviews on the CI runner do not, and neither do teammates' local sessions. The criteria change every few weeks, and the team wants each change reviewed in a pull request like code.

Which change would **MOST** reliably fix this?

- ☐ (A) Keep the criteria in the lead's user-level file and add an `@import` of that `~/.claude` path to the project's `CLAUDE.md` in the repository.
- ☐ (B) Move the criteria into a `CLAUDE.local.md` file at the repository root so Claude Code loads them alongside the project's other instructions.
- ☐ (C) Move the criteria into the project's `CLAUDE.md` at the repository root and commit it, so the runner's checkout and each clone load one file.
- ☐ (D) Store the criteria in a file in the repository and pass it to the CI job's `claude -p` command with `--append-system-prompt-file` on each run.

Question 5

Agentic Architecture & Orchestration

SCENARIO · DEVELOPER PRODUCTIVITY WITH CLAUDE

Setting: an Agent SDK assistant for engineers in large or legacy codebases. It reads, searches and edits code with the built-in tools (Read, Write, Bash, Grep, Glob) and connects to MCP servers.

For each bug report the team runs four stages: reproduce, locate, fix, and add a regression test. Locating ranges from one Grep to a trace across several services. A prototype with a fixed sequence of searches failed on most multi-service bugs. A second prototype, one free-form agent given the whole job, located bugs well but skipped the regression test in 20% of runs.

How should the workflow be structured?

- (A) Keep the four stages as a fixed sequence in code, and run the locate stage as an agent loop that chooses its own searches.
- (B) Keep the free-form agent and add a CLAUDE.md rule that each fix must include a regression test before the agent reports the bug as done.
- (C) Keep the fixed search sequence and add more search patterns for traces across services, so the locate stage handles the harder bugs as well.
- (D) Run the free-form agent twice for each bug and keep whichever run's output includes a regression test, discarding the other run's changes.

Question 6

SCENARIO · STRUCTURED DATA EXTRACTION

Setting: Claude pulls fields out of unstructured documents, the output is checked against a JSON schema, and the results feed downstream systems. Odd cases must not slip through.

Purchase orders arrive from about 400 suppliers, many sending one order a year. The prompt holds three worked examples, all from one large supplier, each showing payment terms of Net 30 and a single address. `payment_terms` is nullable, and the prompt says to return null when terms are absent. Yet 14% of orders that state no terms come back as Net 30, and orders with separate billing and shipping addresses repeat the billing address in both fields. Every output passes validation.

What change would MOST effectively fix this?

- ☐ (A) Validate each extraction in code, and retry any failing record with its specific validation errors appended to a follow-up request.
- ☐ (B) Replace the examples with three or four orders from different suppliers, including one that states no terms and one that ships to a second address.
- ☐ (C) Build a library holding one example order per supplier, and insert the matching supplier's example into each request at extraction time.
- ☐ (D) Restate the null rule for `payment_terms` in capitals at the end of the prompt, after the document, so the model reads it last.

Question 7

SCENARIO · CLAUDE CODE FOR CONTINUOUS INTEGRATION

Setting: Claude Code runs unattended in a CI/CD pipeline, reviewing pull requests and proposing tests. Its comments must be specific enough to act on, with few false alarms.

Logs show two problems. A nightly flaky-test triage step that a teammate added runs `claude "$PROMPT"` and hangs until the 60-minute CI timeout; its log ends at Claude Code's input prompt. Separately, the test-generation step runs `claude -p` with `--permission-mode dontAsk`, and 25% of its generated tests fail in CI. Its transcripts show that every `npm test` call Claude made was denied.

Which two changes should you make? **Choose 2**

- ☐ A Add `--max-turns` to the triage step's claude command, so that a session running too long stops before the CI timeout.
- ☐ B Add the `-p` flag to the triage step's claude command, so that Claude Code runs the prompt and exits when it finishes.
- ☐ C Switch the test-generation step from `dontAsk` to `acceptEdits` permission mode, so that Claude's shell commands run without approval.
- ☐ D Add a line to the project `CLAUDE.md` telling Claude to run `npm test` after writing tests and to fix any failures it sees.
- ☐ E Add `Bash(npm test *)` to the test-generation step's `--allowedTools`, so that its test runs are approved under `dontAsk`.

Question 8

SCENARIO · CODE GENERATION WITH CLAUDE CODE

Setting: a team uses Claude Code every day to write, refactor, debug and document code. They set it up with CLAUDE.md files and custom slash commands, and have to decide when to plan before editing.

To map how an order moves through a legacy fulfillment service of 60 files, Claude spawned one Explore subagent per file, each asked to summarize its file. The 60 summaries, about 400 words each, now take up much of the main context, and Claude's account of the order flow still has gaps where one file hands off to another.

What change would MOST effectively fix this?

- ☐ (A) Keep one subagent per file, but cap each summary at 100 words so that the 60 results take up far less of the main context.
- ☐ (B) Delegate by question: one subagent traces an order from intake to shipment, another maps retry paths, and each cites the files involved.
- ☐ (C) Have Claude read the 60 files itself in the main session, so that it sees how each file calls the next one while it builds the order flow.
- ☐ (D) Run /compact after each batch of 15 subagent results, asking it to keep each file's summary and drop the rest of the output.

Question 9

SCENARIO · DEVELOPER PRODUCTIVITY WITH CLAUDE

Setting: an Agent SDK assistant for engineers in large or legacy codebases. It reads, searches and edits code with the built-in tools (Read, Write, Bash, Grep, Glob) and connects to MCP servers.

Your team's legacy-systems MCP server, also used by three other teams' agents, offers `get_module_info`, which reads build manifests, and `describe_component`, which reads deployed services' runtime settings. Both descriptions say "Returns details about a legacy component," and the agent calls the wrong one for 30% of questions. Separately, `run_report` takes a free-text report name; 20% of calls pass names such as "deps" that match none of its six fixed reports, and each failure returns the valid names for a retry.

Which two changes should you make? **Choose 2**

- ☐ A Rename the two tools after what each reads, such as `get_build_dependencies` and `get_runtime_settings`, and state in each description which questions it answers.
- ☐ B Add example inputs to both tool definitions showing a typical component identifier, so the agent sees how `get_module_info` and `describe_component` are called.
- ☐ C Declare `run_report`'s report name as an enum listing the six fixed reports in its input schema, with a one-line description of each report.
- ☐ D Have `run_report` return an `isError` result listing the six valid report names whenever a call passes a report name it does not recognize.
- ☐ E Add rules to the agent's system prompt mapping dependency questions to `get_module_info` and runtime settings questions to `describe_component`.

Question 10

Claude Code Configuration & Workflows

SCENARIO · CODE GENERATION WITH CLAUDE CODE

Setting: a team uses Claude Code every day to write, refactor, debug and document code. They set it up with CLAUDE.md files and custom slash commands, and have to decide when to plan before editing.

Shoppers occasionally see an old price for several minutes after an admin edits a promotion. The price passes through three layers you have never worked in: an edge cache in front of the storefront, a Redis cache in the pricing service, and memoization in the cart module. A teammate expects the eventual fix to be a few lines, but nobody has traced which layer holds the stale value. Staging runs without the edge cache.

What is the **BEST** way to start this task with Claude Code?

- ☐ (A) Start the session in plan mode so Claude reads the three caching layers and proposes where the fix belongs before any file changes.
- ☐ (B) Use direct execution to have Claude invalidate the Redis pricing entry whenever a promotion is saved, then run the pricing tests.
- ☐ (C) Ask Claude to write an end-to-end staging test that edits a promotion and checks the cart price, then fix code until it passes.
- ☐ (D) Leave plan mode off and turn on auto-accept edits so Claude can adjust each caching layer in turn, rerunning the pricing tests.



Question 11

SCENARIO · STRUCTURED DATA EXTRACTION

Setting: Claude pulls fields out of unstructured documents, the output is checked against a JSON schema, and the results feed downstream systems. Odd cases must not slip through.

On a labelled set of 1,500 receipts, fields scored 0.9 or higher were right 99% of the time, so such fields skip review. Last week the team revised the prompt and its examples, which halved errors on handwritten receipts. Since then 30% more fields score 0.9 or higher, and the weekly 2% random sample of auto-accepted receipts shows errors rising from 1% to 4%.

What change would MOST effectively fix this?

- ☐ (A) Raise the auto-accept threshold to 0.95 so that fewer fields skip review until the sampled error rate falls back to 1%.
- ☐ (B) Score the labelled receipts with the revised prompt, and set the auto-accept threshold again from those new scores.
- ☐ (C) Roll back the prompt revision, because the earlier prompt's scores were the ones calibrated against the labelled set.
- ☐ (D) Raise the weekly random sample of auto-accepted receipts from 2% to 10% to measure the new error rate precisely.

Question 12

SCENARIO · CLAUDE CODE FOR CONTINUOUS INTEGRATION

Setting: Claude Code runs unattended in a CI/CD pipeline, reviewing pull requests and proposing tests. Its comments must be specific enough to act on, with few false alarms.

Findings are labelled blocking, major or minor, and the merge gate stops a pull request on any blocking finding. The prompt defines blocking as 'a serious problem that must be fixed before merge.' Last month 16 findings of SQL built by string concatenation in admin endpoints were split 9 blocking and 7 minor, and the split did not follow whether request input could reach the query, the line the security lead draws. Each pull request had one to three findings, and rerunning a review returns the same labels.

What change would MOST effectively fix this?

- ☐ (A) Move severity labelling into a separate second pass that receives one finding at a time with its diff and assigns blocking, major or minor.
- ☐ (B) Add an instruction to be conservative with blocking, labelling an issue blocking only when it is clearly severe and choosing major when in doubt.
- ☐ (C) Turn on extended thinking for the review step so the model reasons through each finding's risk at length before it assigns a severity label.
- ☐ (D) Rewrite the severity guidance as criteria stating what makes an issue blocking, major or minor, each with a short code example from this codebase.

Question 13

SCENARIO · CUSTOMER SUPPORT RESOLUTION AGENT

Setting: an Agent SDK agent that settles customer cases from start to finish (returns, billing disputes, account problems), working through four custom MCP tools: `get_customer`, `lookup_order`, `process_refund` and `escalate_to_human`. Most cases should close in one contact; the rest go to a person.

A front-line agent holds nine tools for account settings, shipping, promotions and escalation, and delegates billing work to a subagent that holds `get_customer`, `lookup_order` and `process_refund`, with the 30-page refund policy in its prompt. Logs show 60% of delegations are order-status questions settled by a single `lookup_order` call. Each such delegation adds about 25 seconds, of which the `lookup_order` call takes under one second.

What change would BEST reduce these delays?

- (A) Merge the billing subagent into the front-line agent, giving it `get_customer`, `lookup_order`, `process_refund` and the full refund policy in its prompt.
- (B) Route order-status questions to a new status subagent holding only `lookup_order`, and leave refund work with the billing subagent and its policy.
- (C) Give the front-line agent a read-only order-status tool, and keep `process_refund` and the refund policy with the billing subagent.
- (D) Have the MCP server cache `lookup_order` results for ten minutes, so repeated status checks on an order skip the backend query.

Question 14

SCENARIO · MULTI-AGENT RESEARCH SYSTEM

Setting: on the Agent SDK, a coordinator splits research work among subagents for web search, document analysis, synthesis and report writing, and hands back a report with citations.

Logs show two problems. A teammate added a peer tool so web-search subagents hand pages straight to the document-analysis subagent; since then it analyzes pages unrelated to its assigned subtopic, and the coordinator's logs show neither which pages arrived nor why. Separately, when one of four search subagents fails because a source site is down for hours, the coordinator abandons the whole report.

Which two changes should you make? **Choose 2**

- ☐ A Give the document-analysis subagent its own web search tool so it can fetch any page it needs without waiting on the search subagents.
- ☐ B Remove the peer tool so search subagents return pages to the coordinator, which passes the relevant ones to document analysis in its prompt.
- ☐ C Add a logging hook that records every message the peer tool carries, so that the coordinator's traces show each page exchange.
- ☐ D Raise the search subagents' timeout and retry budget so that a slow subtopic can finish before the coordinator gives up on the report.
- ☐ E Have the coordinator treat a failed subagent as a gap, re-delegate that subtopic once, then finish the report stating what remains uncovered.

Question 15

Tool Design & MCP Integration

SCENARIO · DEVELOPER PRODUCTIVITY WITH CLAUDE

Setting: an Agent SDK assistant for engineers in large or legacy codebases. It reads, searches and edits code with the built-in tools (Read, Write, Bash, Grep, Glob) and connects to MCP servers.

Every engineer's Claude Code sessions should use the internal incident-history MCP server, a local stdio server started with npx that reads a personal API token from its environment. Audit rules require each query to be attributed to the engineer who ran it, and tokens may not be committed. The server's launch arguments change often, and the team wants each change to reach everyone through the repository.

Which configuration BEST meets these requirements?

- (A) Commit the server to .mcp.json with its token entry written as `${INCIDENTS_TOKEN}`, and have each engineer export that variable in their own shell.
- (B) Have each engineer add the server at local scope with their own token, keeping the entry in their `~/.claude.json` for this project.
- (C) Commit the server to .mcp.json pointing at a team proxy that adds one shared service token, so no engineer has to handle a token.
- (D) Commit the server to .mcp.json without a token, and have each engineer sign in through `/mcp` using the server's OAuth flow.

Question 16

SCENARIO · CUSTOMER SUPPORT RESOLUTION AGENT

Setting: an Agent SDK agent that settles customer cases from start to finish (returns, billing disputes, account problems), working through four custom MCP tools: `get_customer`, `lookup_order`, `process_refund` and `escalate_to_human`. Most cases should close in one contact; the rest go to a person.

Logs from 400 contacts that raised two or more issues, such as a damaged-item return plus a duplicate charge, show the agent verifies the customer, settles the first issue, then ends its turn with a closing message. These sessions average 7 turns against a `max_turns` limit of 30, and 38% are reopened about an issue the agent never addressed.

What change would MOST effectively fix this?

- ☐ (A) Raise the agent's `max_turns` limit so that longer conversations have enough iterations to work through every issue the customer raises.
- ☐ (B) Have the agent first list each distinct issue in the message, investigate each one with tool calls sharing one customer lookup, then send one combined reply.
- ☐ (C) Add orchestration code that rejects the agent's final reply until `process_refund` or `escalate_to_human` has been called at least once in the session.
- ☐ (D) Start a separate agent session for each issue the customer raises, give each session the full message, and send the customer each reply as it finishes.

Question 17

SCENARIO · CODE GENERATION WITH CLAUDE CODE

Setting: a team uses Claude Code every day to write, refactor, debug and document code. They set it up with CLAUDE.md files and custom slash commands, and have to decide when to plan before editing.

.claude/skills/db-migrate/SKILL.md holds the team's 250-line procedure for schema migrations, including backfill and rollback steps. Its description reads 'Database helper.' When developers ask Claude to add a column or an index, Claude writes the migration by hand, without the skill, in 80% of sessions and skips the rollback script. Typing /db-migrate gives the intended result. Most sessions involve no migrations.

What change would MOST effectively fix this?

- (A) Add a paths field matching db/migrations/** to the skill's frontmatter so that Claude loads it for work on migration files.
- (B) Move the 250-line procedure into the project CLAUDE.md so that it is in context for every column or index change.
- (C) Set user-invocable to false in the skill's frontmatter so that Claude treats the skill as background knowledge it applies on its own.
- (D) Rewrite the skill's description to state what it does and which requests it covers, such as schema changes to tables.

Question 18

SCENARIO · STRUCTURED DATA EXTRACTION

Setting: Claude pulls fields out of unstructured documents, the output is checked against a JSON schema, and the results feed downstream systems. Odd cases must not slip through.

Your code validates each contract extraction and retries a failure with the error appended, up to four times, before human review. Retries now use 35% of tokens. In 500 sampled failures, 80% were date-format errors, and 94% of those passed on the first retry. When a contract never states `contract_end_date`, the first three retries fail, and in 9% of those cases the fourth returns a plausible date that passes. Billing schedules renewals from each stored end date.

What change would MOST effectively fix this?

- ☐ (A) Classify each validation failure, retrying format errors with their specific error and sending a missing `contract_end_date` straight to human review.
- ☐ (B) Send every failed extraction straight to human review without any retry, so that no tokens go to retries and no later attempt can invent a date.
- ☐ (C) Resend the full contract text with each retry, alongside the failed extraction and its error, so the model can locate `contract_end_date` before human review.
- ☐ (D) Enable strict tool use on the extraction tool, so that each response carries the required fields, including `contract_end_date`, with valid types.

Question 19

SCENARIO · MULTI-AGENT RESEARCH SYSTEM

Setting: on the Agent SDK, a coordinator splits research work among subagents for web search, document analysis, synthesis and report writing, and hands back a report with citations.

For broad topics the coordinator launches three web-search subagents in parallel, and each receives the same prompt: the user's topic and an instruction to find authoritative sources. Across 50 reports, 58% of the URLs each subagent returned were also returned by another, search tokens tripled compared with one subagent, and reports cite about as many distinct sources as before. Report latency has to stay where it is.

What change would MOST effectively fix this?

- ☐ A Replace the three parallel search subagents with one search subagent that runs the queries in turn and keeps a running list of sources already read.
- ☐ B Have the coordinator merge repeated sources before synthesis, so the synthesis subagent receives each source once along with the combined notes from each search.
- ☐ C Let the search subagents record each retrieved URL in a shared store that each subagent checks before reading a source, skipping any already taken.
- ☐ D Have the coordinator give each search subagent a distinct subtopic or source type, and state in each delegation what the other subagents are covering.

Question 20

SCENARIO · STRUCTURED DATA EXTRACTION

Setting: Claude pulls fields out of unstructured documents, the output is checked against a JSON schema, and the results feed downstream systems. Odd cases must not slip through.

Claim forms are extracted into a schema whose `policy_number` and `incident_date` fields are nullable, and the prompt says to return null when a value is absent. Every record passes validation. An audit of 600 records found 4% holding a policy number or date that appears nowhere on the form, often with two digits swapped. Rerunning a flawed form returns the same wrong value, and the model reports the same confidence on such values as on correct ones.

What change would MOST effectively catch and correct these errors?

- ☐ (A) Add a pattern constraint to the `policy_number` and `incident_date` fields so values in the wrong format fail validation and are retried with the error.
- ☐ (B) Have the model rate its confidence in each field, and send any field rated below a set threshold to a human reviewer before loading.
- ☐ (C) Require a verbatim excerpt from the form for each field, have code confirm the excerpt is on the form and holds the value, and retry each failure.
- ☐ (D) Extract each form twice in separate requests, and send any record whose two extractions disagree on a field to a human reviewer.

Check your answers

Tick each one you got right. Explanations start on the next page.

Count a multiple-response question as right only if you picked every correct letter and nothing else.

☐ Q1	D	☐ Q2	B	☐ Q3	A, C	☐ Q4	C	☐ Q5	A
☐ Q6	B	☐ Q7	B, E	☐ Q8	B	☐ Q9	A, C	☐ Q10	A
☐ Q11	B	☐ Q12	D	☐ Q13	C	☐ Q14	B, E	☐ Q15	A
☐ Q16	B	☐ Q17	D	☐ Q18	A	☐ Q19	D	☐ Q20	C

Your score ____ / 20

By domain

Agentic Architecture & Orchestration	Q2, Q5, Q14, Q16, Q19	____ / 5
Tool Design & MCP Integration	Q1, Q9, Q13, Q15	____ / 4
Claude Code Configuration & Workflows	Q4, Q7, Q10, Q17	____ / 4
Prompt Engineering & Structured Output	Q6, Q12, Q18, Q20	____ / 4
Context Management & Reliability	Q3, Q8, Q11	____ / 3

Question 1

- D** Return CRM timeouts as `isError` results marked transient and retryable, and keep returning an empty customers list when a search finds no match.

Why D is right

During the outages the server caught every timeout and sent back an empty list, the same answer a search with no match gives. An empty list from a completed search is a valid answer. A timeout is a failure, and it should come back as an error that says what went wrong. With the two kept apart, the agent asks for another address after a no-match and tells the customer about a temporary problem during an outage.

Why the other options are wrong

- A** Retry the CRM query inside `get_customer` up to three times on a timeout, and return an empty customers list if every attempt fails.
The outages lasted about an hour, so all three retries time out and the server still returns the empty list.
- B** Return an `isError` result whenever `get_customer` finds no account, with a message telling the agent to ask the customer for another email.
Timeouts and no-matches still look identical, now both as errors, so customers hit by an outage are asked for an email they don't have.
- C** Have the agent call `escalate_to_human` whenever `get_customer` returns an empty customers list, so a person can check the account by hand.
About 9% of searches are routine address mix-ups the agent can fix by asking again. Escalating all of them sends easy cases to people and cuts first-contact resolution.

Exam tip: Never let a tool turn a backend failure into an empty result. Report the failure as an error, and keep empty results for searches that found nothing.

Sources: modelcontextprotocol.io/specification/2025-11-25/server/tools; Official exam page, Anthropic Partner Academy

Question 2

- B** Move the check into a `PreToolUse` hook on `process_refund` that denies calls over \$300, with a reason telling Claude to escalate the case.

Why B is right

By the time a `PostToolUse` hook runs, the money has moved, the customer has the email and the fee is charged. A `PreToolUse` hook runs before the call reaches the backend and can deny it. Its deny reason steers Claude to `escalate_to_human`. The check sits in the agent's path, so human agents can still issue larger approved refunds through the same backend.

Why the other options are wrong

- A** Have the `PostToolUse` hook call the payment system's reversal endpoint for any refund over \$300, then tell Claude to escalate the case.
A reversal comes too late. The email has gone, the fee is charged and the customer sees a second transaction.

Question 2, continued

- C** Configure the refund backend to reject any request over \$300, returning an error result that Claude can act on by escalating the case.
Human agents use the same backend for larger approved refunds, so a cap there blocks legitimate work.
- D** State the \$300 supervisor limit in the system prompt, with examples of contacts in which the agent escalates the case instead of refunding.
Prompt instructions and examples are probabilistic, and a supervisor limit has to hold on every call.

Exam tip: Put any check that must stop an action in PreToolUse. Keep PostToolUse for work that is safe after the tool has run, such as logging or trimming results.

Sources: code.claude.com/docs/en/hooks; Official exam page, Anthropic Partner Academy

Question 3

- A** Open the synthesis input with a short summary of each subagent's key findings, then give each subagent's full results under its own section header.
- C** Keep the user's stated constraints, such as date cutoffs and currency, in a block added to each coordinator request outside the summarized history.

Why A and C are right

Findings from the middle four subagents reach only 51% of reports, while the first and last reach 94%. A short summary of each subagent's key findings at the top puts them where the model reads reliably, and section headers let it find the detail behind each point. Separately, summaries turn exact values such as a date cutoff or a currency into vague phrases. A block added to each coordinator request, outside the summarised history, carries those constraints into every delegation.

Why the other options are wrong

- B** Send the coordinator's full conversation history on each request instead of summaries, so the user's date cutoff and currency are kept word for word.
The sessions already outgrew the context window, which is why summarising began. Sending the full history brings that failure back.
- D** Have each search subagent return the full text of the pages it read, so the synthesis subagent can check each finding against the original wording.
Full page text makes the synthesis input far longer and pushes even more findings into the middle.
- E** Have the coordinator summarize its history less often, every ten turns instead of every few, so the user's refinements stay verbatim for longer.
Constraints from the first turns still reach a summary eventually, and longer gaps let the history grow back towards the limit.

Exam tip: When one agent aggregates many results, lead with a findings summary under clear headers, and keep the user's exact constraints outside anything that gets summarised.

Sources: Official exam page, Anthropic Partner Academy; code.claude.com/docs/en/agent-sdk/agent-loop

Question 4

- C** Move the criteria into the project's CLAUDE.md at the repository root and commit it, so the runner's checkout and each clone load one file.

Why C is right

The criteria sit in the lead's ~/.claude/CLAUDE.md, which is why only the lead's reviews follow them. The project CLAUDE.md travels with the repository, so the runner's checkout and every teammate's clone load the same file. Edits to it go through pull requests like any other change, which is what the team wants.

Why the other options are wrong

- A** Keep the criteria in the lead's user-level file and add an @import of that ~/.claude path to the project's CLAUDE.md in the repository.
The import resolves against each machine's own home directory, so the runner and teammates point at a file they don't have.
- B** Move the criteria into a CLAUDE.local.md file at the repository root so Claude Code loads them alongside the project's other instructions.
CLAUDE.local.md is for personal preferences and belongs in .gitignore, so it never reaches the runner or teammates and is never reviewed.
- D** Store the criteria in a file in the repository and pass it to the CI job's claude -p command with --append-system-prompt-file on each run.
This fixes the CI job alone. Teammates' local reviews still load nothing.

Exam tip: Put rules that must apply in CI and on every machine in the committed project CLAUDE.md. Keep user-level files for personal taste.

Sources: code.claude.com/docs/en/memory; code.claude.com/docs/en/cli-reference; [Official exam page](#), Anthropic Partner Academy

Question 5

- A** Keep the four stages as a fixed sequence in code, and run the locate stage as an agent loop that chooses its own searches.

Why A is right

Running the stages in code makes the regression test a step that happens on every bug. The locate stage can't be scripted, because multi-service bugs follow paths nobody can list in advance. Running that one stage as an agent loop lets it choose its own searches. Workflow patterns are building blocks you combine, not one choice for the whole job.

Why the other options are wrong

- B** Keep the free-form agent and add a CLAUDE.md rule that each fix must include a regression test before the agent reports the bug as done.
The agent already skipped a test it was expected to write. A prompt rule is followed most of the time, not every time.

Question 5, continued

- (C) Keep the fixed search sequence and add more search patterns for traces across services, so the locate stage handles the harder bugs as well.
Multi-service bugs follow paths nobody can list in advance. More fixed patterns add searches without letting the stage follow what it finds.
- (D) Run the free-form agent twice for each bug and keep whichever run's output includes a regression test, discarding the other run's changes.
If runs fail independently, both still skip the test on about 4% of bugs, and each bug now costs two full runs.

Exam tip: Put the stages that must happen in code. Give autonomy only to the stage whose steps you can't predict.

Sources: anthropic.com/engineering/building-effective-agents; Official exam page, Anthropic Partner Academy

Question 6

- (B) Replace the examples with three or four orders from different suppliers, including one that states no terms and one that ships to a second address.

Why B is right

Every example says Net 30 and shows a single address, so the model has learned those as defaults. The copied values are well formed, so every output still passes validation. Examples that differ from one another break the accidental pattern. One that returns null for missing terms and one with two distinct addresses show the edge cases directly.

Why the other options are wrong

- (A) Validate each extraction in code, and retry any failing record with its specific validation errors appended to a follow-up request.
Net 30 and a repeated address are well-formed values, so validation passes and nothing triggers a retry.
- (C) Build a library holding one example order per supplier, and insert the matching supplier's example into each request at extraction time.
Many suppliers send one order a year, so their examples would be few or missing, and each supplier's own example only teaches that supplier's habits, not an order with no terms.
- (D) Restate the null rule for payment_terms in capitals at the end of the prompt, after the document, so the model reads it last.
The null rule is already in the prompt and loses to three examples that all show Net 30. Repeating it does nothing for the copied address.

Exam tip: Check your few-shot examples for values they all share, and make sure at least one shows each edge case your documents contain.

Sources: platform.claude.com/docs/en/build-with-claude/prompt-engineering/claude-prom...; Official exam page, Anthropic Partner Academy

Question 7

- B** Add the `-p` flag to the triage step's claude command, so that Claude Code runs the prompt and exits when it finishes.
- E** Add `Bash(npm test *)` to the test-generation step's `--allowedTools`, so that its test runs are approved under `dontAsk`.

Why B and E are right

The triage step runs claude without `-p`, so it sits at the input prompt until the CI timeout. With `-p` it runs the prompt and exits. The `dontAsk` mode denies any call that would otherwise prompt, but it runs calls an `--allowedTools` entry covers. No entry covered `npm test`, so Claude never got to run its tests. Pre-approving that one command lets it run and fix them while the rest stays locked down.

Why the other options are wrong

- A** Add `--max-turns` to the triage step's claude command, so that a session running too long stops before the CI timeout.
A turn cap ends a loop that keeps working. This session sits idle at the input prompt, so the cap never fires.
- C** Switch the test-generation step from `dontAsk` to `acceptEdits` permission mode, so that Claude's shell commands run without approval.
`acceptEdits` approves file edits and a few filesystem commands. `npm test` still needs an `allow` entry, so the denials continue.
- D** Add a line to the project `CLAUDE.md` telling Claude to run `npm test` after writing tests and to fix any failures it sees.
Claude already tries `npm test`. An instruction grants no permission, so each call is still denied.

Exam tip: Start every Claude Code step in CI with `-p`, and pre-approve the exact commands it must run instead of loosening the whole permission mode.

Sources: code.claude.com/docs/en/cli-reference; code.claude.com/docs/en/agent-sdk/agent-loop; [Official exam page](#), [Anthropic Partner Academy](#)

Question 8

- B** Delegate by question: one subagent traces an order from intake to shipment, another maps retry paths, and each cites the files involved.

Why B is right

Per-file summaries leave the gaps observed, because no subagent sees where one file hands off to the next. Delegating by question lets each subagent follow the code across files. It returns one flow that cites the files involved. The main context then holds a few answers instead of 60 summaries.

Why the other options are wrong

- A** Keep one subagent per file, but cap each summary at 100 words so that the 60 results take up far less of the main context.
Shorter summaries save space, but each subagent still sees one file, so the handoffs stay unexplained.

Question 8, continued

- C** Have Claude read the 60 files itself in the main session, so that it sees how each file calls the next one while it builds the order flow.
Reading 60 files in the main session puts every line into the context the subagents were meant to protect, and it fills even faster.
- D** Run /compact after each batch of 15 subagent results, asking it to keep each file's summary and drop the rest of the output.
Compacting the summaries shrinks them further but can't add the cross-file links none of them contained.

Exam tip: Split exploration by the question you need answered, not by file, so each subagent can follow the code where it leads.

Sources: code.claude.com/docs/en/sub-agents; Official exam page, Anthropic Partner Academy

Question 9

- A** Rename the two tools after what each reads, such as `get_build_dependencies` and `get_runtime_settings`, and state in each description which questions it answers.
- C** Declare `run_report`'s report name as an enum listing the six fixed reports in its input schema, with a one-line description of each report.

Why A and C are right

The tools `get_module_info` and `describe_component` do different jobs, but their identical descriptions give the model nothing to choose by. Names and descriptions that say what each tool reads give every agent using the server a clear boundary. The report name takes one of six fixed values, so an enum in the input schema lets the agent pick a valid report on the first call instead of guessing free text.

Why the other options are wrong

- B** Add example inputs to both tool definitions showing a typical component identifier, so the agent sees how `get_module_info` and `describe_component` are called.
Both tools take the same kind of identifier, so the examples look alike and say nothing about which question each tool answers.
- D** Have `run_report` return an `isError` result listing the six valid report names whenever a call passes a report name it does not recognize.
The server already returns the valid names after a failure. An error only helps once a call has failed, so each bad name still costs a wasted round trip.
- E** Add rules to the agent's system prompt mapping dependency questions to `get_module_info` and runtime settings questions to `describe_component`.
Three other teams' agents read the same descriptions. A rule in one agent's prompt leaves their misrouting in place.

Exam tip: When you own the server, fix tool confusion in its interface, and turn any fixed set of parameter values into an enum.

Sources: anthropic.com/engineering/writing-tools-for-agents; Official exam page, Anthropic Partner Academy

Question 10

- A** Start the session in plan mode so Claude reads the three caching layers and proposes where the fix belongs before any file changes.

Why A is right

Anthropic's guidance says planning helps most when you're unsure of the approach or new to the code, and both hold here. In plan mode Claude reads the edge cache, Redis and cart code before any file changes. The fix may be a few lines, but it has to land in the layer that causes the bug. A small expected diff doesn't mean you know where it goes.

Why the other options are wrong

- B** Use direct execution to have Claude invalidate the Redis pricing entry whenever a promotion is saved, then run the pricing tests.
It commits to Redis before anyone has traced the stale value. If the edge cache or the cart memoisation serves it, shoppers still see old prices.
- C** Ask Claude to write an end-to-end staging test that edits a promotion and checks the cart price, then fix code until it passes.
Staging runs without the edge cache, so a passing staging test can't show the fix holds where shoppers see the old price.
- D** Leave plan mode off and turn on auto-accept edits so Claude can adjust each caching layer in turn, rerunning the pricing tests.
Editing each layer in turn is trial and error with no review. The pricing tests can't see a value served by the edge cache or the cart, so a green run proves nothing.

Exam tip: Choose plan mode by how well you know where the change belongs, not by how small the diff will be.

Sources: code.claude.com/docs/en/best-practices; [Official exam page](#), Anthropic Partner Academy

Question 11

- B** Score the labelled receipts with the revised prompt, and set the auto-accept threshold again from those new scores.

Why B is right

The revised prompt shifted the confidence scores, so the same 0.9 cut-off now lets weaker fields through. Scoring the labelled receipts again shows what accuracy each score now carries. The threshold can then sit where it again means 99%. The revision's gains on handwritten receipts stay.

Why the other options are wrong

- A** Raise the auto-accept threshold to 0.95 so that fewer fields skip review until the sampled error rate falls back to 1%.
0.95 is a guess on a scale that has moved. Without the labelled set, nobody knows whether it lets errors through or sends far more fields to review.

Question 11, continued

- (C) Roll back the prompt revision, because the earlier prompt's scores were the ones calibrated against the labelled set.
Rolling back gives up halving the errors on handwritten receipts, when the fault is a stale threshold, not the revision.
- (D) Raise the weekly random sample of auto-accepted receipts from 2% to 10% to measure the new error rate precisely.
A bigger sample measures the rise more precisely, but errors keep flowing downstream while the routing rule stays wrong.

Exam tip: Treat every prompt or model change as invalidating your confidence threshold until you have re-scored the labelled set.

Source: [Official exam page, Anthropic Partner Academy](#)

Question 12

- (D) Rewrite the severity guidance as criteria stating what makes an issue blocking, major or minor, each with a short code example from this codebase.

Why D is right

Rerunning a review returns the same labels, so the split comes from the definition, not from chance. 'A serious problem' can be read two defensible ways, and the 9 to 7 split ignores the security lead's line, which is whether request input can reach the query. Criteria that state that line for each level, each with a short code example from this codebase, give the model a rule it applies the same way every time.

Why the other options are wrong

- (A) Move severity labelling into a separate second pass that receives one finding at a time with its diff and assigns blocking, major or minor.
These pull requests have one to three findings each, so attention isn't the problem. A second pass reads the same vague definition.
- (B) Add an instruction to be conservative with blocking, labelling an issue blocking only when it is clearly severe and choosing major when in doubt.
'Be conservative' is as vague as 'serious'. It shifts labels down without saying where the line is, so injectable queries would pass the gate as major.
- (C) Turn on extended thinking for the review step so the model reasons through each finding's risk at length before it assigns a severity label.
Longer reasoning helps when labels come from shallow analysis. Here the analysis repeats and the definition is missing, so the model reasons against the same undefined idea of serious.

Exam tip: When equivalent findings get different but repeatable severity labels, write down the boundary for each level and anchor it with a code example from your codebase.

Sources: [Official exam page, Anthropic Partner Academy](#); platform.claude.com/docs/en/build-with-claude/prompt-engineering/claude-prom...

Question 13

- C** Give the front-line agent a read-only order-status tool, and keep process_refund and the refund policy with the billing subagent.

Why C is right

The lookup_order call takes under a second, so the delay is the subagent starting from an empty context, taking its own turns and reporting back. One read-only order-status tool on the front-line agent removes that hop for 60% of delegations. It adds a single narrow choice with no side effects and no policy text behind it. The billing subagent keeps process_refund and the 30-page policy.

Why the other options are wrong

- A** Merge the billing subagent into the front-line agent, giving it get_customer, lookup_order, process_refund and the full refund policy in its prompt.
The front-line agent already holds nine tools. Adding a refund action and a 30-page policy to every chat dilutes its focus and makes tool selection less reliable.
- B** Route order-status questions to a new status subagent holding only lookup_order, and leave refund work with the billing subagent and its policy.
A leaner subagent is still a delegation, so the round trip that makes up most of the 25 seconds remains.
- D** Have the MCP server cache lookup_order results for ten minutes, so repeated status checks on an order skip the backend query.
Caching speeds up a query that already takes under a second. The time goes on the delegation around it, which still happens for every status question.

Exam tip: When one read-only need dominates the delegations, give the calling agent a narrow tool for it instead of merging agents or adding another hop.

Sources: Official exam page, Anthropic Partner Academy; code.claude.com/docs/en/sub-agents

Question 14

- B** Remove the peer tool so search subagents return pages to the coordinator, which passes the relevant ones to document analysis in its prompt.
- E** Have the coordinator treat a failed subagent as a gap, re-delegate that subtopic once, then finish the report stating what remains uncovered.

Why B and E are right

In a hub-and-spoke design the coordinator routes all traffic between subagents. Passing the relevant pages in the document-analysis prompt lets the coordinator match them to each subtopic, and every exchange lands in its trace. One failed subtopic should not throw away three good results. Re-delegating once and then marking the gap gives a usable report that is honest about its coverage.

Why the other options are wrong

- A** Give the document-analysis subagent its own web search tool so it can fetch any page it needs without waiting on the search subagents.
A second search path duplicates the search subagents' role and adds retrievals the coordinator never chose and can't see.

Question 14, continued

- C** Add a logging hook that records every message the peer tool carries, so that the coordinator's traces show each page exchange.
A log shows which pages arrived, but the coordinator still doesn't choose them, so the off-topic analysis continues.
- D** Raise the search subagents' timeout and retry budget so that a slow subtopic can finish before the coordinator gives up on the report.
Retries help only with transient errors. An outage that lasts hours outlasts the retry budget, and the all-or-nothing policy still throws away the three subtopics that succeeded.

Exam tip: Keep the coordinator as the only route between subagents, and write its failure policy so a failed subtopic becomes a stated gap, not a lost report.

Sources: [Official exam page](#), [Anthropic Partner Academy](#); anthropic.com/engineering/multi-agent-research-system

Question 15

- A** Commit the server to `.mcp.json` with its token entry written as `${INCIDENTS_TOKEN}`, and have each engineer export that variable in their own shell.

Why A is right

Project scope lives in `.mcp.json` in the repository, so each change reaches every engineer on their next pull. Claude Code expands `${INCIDENTS_TOKEN}` from each engineer's environment when it starts the server. The committed file holds no secret. Each query runs under its owner's token, which meets the audit rule.

Why the other options are wrong

- B** Have each engineer add the server at local scope with their own token, keeping the entry in their `~/.claude.json` for this project.
Local-scope entries sit in each engineer's `~/.claude.json`, outside the repository, so every argument change has to be repeated by hand on each machine.
- C** Commit the server to `.mcp.json` pointing at a team proxy that adds one shared service token, so no engineer has to handle a token.
One shared service token makes every query look like the same caller, so the audit can't tell engineers apart.
- D** Commit the server to `.mcp.json` without a token, and have each engineer sign in through `/mcp` using the server's OAuth flow.
Sign-in through `/mcp` is for remote servers that offer OAuth. This is a local stdio server that reads a token from its environment.

Exam tip: Commit shared MCP server definitions with a `${VAR}` placeholder for each credential, and list the variable names in your onboarding notes.

Sources: code.claude.com/docs/en/mcp; [Official exam page](#), [Anthropic Partner Academy](#)

Question 16

- B** Have the agent first list each distinct issue in the message, investigate each one with tool calls sharing one customer lookup, then send one combined reply.

Why B is right

Sessions average 7 turns against a limit of 30, so the cap never fires. The loop ends because Claude returns a reply with no tool calls once the first issue is settled. Listing each distinct issue first gives every issue its own investigation. All of them reuse one `get_customer` verification, and one combined reply closes the contact.

Why the other options are wrong

- A** Raise the agent's `max_turns` limit so that longer conversations have enough iterations to work through every issue the customer raises.
The cap never fires at 7 turns out of 30. The loop ends when Claude replies with no tool calls, and a higher cap doesn't change that.
- C** Add orchestration code that rejects the agent's final reply until `process_refund` or `escalate_to_human` has been called at least once in the session.
The refund for the first issue already passes this gate, so the duplicate charge is still ignored. It would also block valid replies that need no refund.
- D** Start a separate agent session for each issue the customer raises, give each session the full message, and send the customer each reply as it finishes.
No session is assigned its own issue, so each tends to settle the same first one and risks a second refund. Each also repeats the identity check, and the customer gets several disjointed replies.

Exam tip: When an agent drops part of a request, check whether the loop ended by choice or hit a cap. That tells you which layer to fix.

Sources: Official exam page, Anthropic Partner Academy; code.claude.com/docs/en/agent-sdk/agent-loop

Question 17

- D** Rewrite the skill's description to state what it does and which requests it covers, such as schema changes to tables.

Why D is right

Skill descriptions sit in context in every session, and Claude matches requests against them. A request to add a column never matches 'Database helper', so Claude writes the migration by hand. Typing `/db-migrate` works because invoking by name skips that match. A clear description lets Claude load the full procedure when it is needed, while the 250 lines stay out of sessions without migrations.

Why the other options are wrong

- A** Add a `paths` field matching `db/migrations/**` to the skill's frontmatter so that Claude loads it for work on migration files.
The `paths` field limits when Claude may load the skill on its own. It doesn't make Claude pick it, and no migration file has been touched when the request arrives.

Question 17, continued

- (B) Move the 250-line procedure into the project CLAUDE.md so that it is in context for every column or index change.
That puts 250 lines into every session, most of which never touch a migration, and the extra bulk makes the rest of CLAUDE.md less reliably followed.
- (C) Set user-invocable to false in the skill's frontmatter so that Claude treats the skill as background knowledge it applies on its own.
That hides the skill from the / menu and stops /db-migrate working, while Claude still matches requests against 'Database helper'.

Exam tip: Write a skill's description in the words developers use when they ask for the task, then test it with a few realistic requests.

Source: code.claude.com/docs/en/skills

Question 18

- (A) Classify each validation failure, retrying format errors with their specific error and sending a missing contract_end_date straight to human review.

Why A is right

Format errors clear on the first retry because the right value is in the contract. A contract that never states contract_end_date gives a retry nothing to find. Each retry is wasted, and the repeated pressure eventually produces a plausible date that billing would renew on. Classifying failures keeps the cheap fixes automatic and sends absent dates to human review.

Why the other options are wrong

- (B) Send every failed extraction straight to human review without any retry, so that no tokens go to retries and no later attempt can invent a date.
It stops invented dates, but reviewers also get the 80% of failures that one retry fixes 94% of the time.
- (C) Resend the full contract text with each retry, alongside the failed extraction and its error, so the model can locate contract_end_date before human review.
The contract never states the end date, so more of its text can't reveal one. It only makes each futile retry cost more.
- (D) Enable strict tool use on the extraction tool, so that each response carries the required fields, including contract_end_date, with valid types.
Strict tool use stops malformed types, and here that is the problem: with the end date required, a contract that has none must still come back with a date, which passes validation and goes straight to billing.

Exam tip: Sort validation failures by cause before tuning retries. Let retries fix what the document holds, and send missing information to a person.

Sources: [Official exam page](#), [Anthropic Partner Academy](#); platform.claude.com/docs/en/build-with-claude/structured-outputs

Question 19

- D** Have the coordinator give each search subagent a distinct subtopic or source type, and state in each delegation what the other subagents are covering.

Why D is right

The coordinator hands three subagents an identical task, so 58% of the URLs overlap and tokens triple with no gain in distinct sources. Separate subtopics or source types send the searches in different directions. Stating each subagent's boundaries in its delegation stops the overlap at the source. The three still run in parallel, so latency holds.

Why the other options are wrong

- A** Replace the three parallel search subagents with one search subagent that runs the queries in turn and keeps a running list of sources already read.
One subagent running the queries in turn removes the overlap but breaks the latency requirement.
- B** Have the coordinator merge repeated sources before synthesis, so the synthesis subagent receives each source once along with the combined notes from each search.
Merging after the searches trims what synthesis reads, but the tokens are already spent and the subagents still search for the same things.
- C** Let the search subagents record each retrieved URL in a shared store that each subagent checks before reading a source, skipping any already taken.
A shared URL list stops repeat reads but doesn't change what each subagent searches for, so the three still cover the same ground. It also has the subagents coordinate with each other instead of through the coordinator.

Exam tip: When parallel subagents return overlapping results, look first at how the coordinator divided the work. Each delegation should state its objective and its boundaries.

Sources: [Official exam page](#), Anthropic Partner Academy; anthropic.com/engineering/multi-agent-research-system

Question 20

- C** Require a verbatim excerpt from the form for each field, have code confirm the excerpt is on the form and holds the value, and retry each failure.

Why C is right

Swapped digits in a policy number still pass validation, and the model reports the same confidence on them. Requiring a verbatim excerpt for each value gives code evidence to check. An excerpt that isn't on the form, or doesn't hold the value, is a failure with a specific error. A retry can then correct it, because the right text is on the form.

Why the other options are wrong

- A** Add a pattern constraint to the policy_number and incident_date fields so values in the wrong format fail validation and are retried with the error.
The wrong values already match the expected format, so a pattern check passes them too.

Question 20, continued

- (B) Have the model rate its confidence in each field, and send any field rated below a set threshold to a human reviewer before loading.
The model reports the same confidence on these values as on correct ones, so a threshold catches almost none of them.
- (D) Extract each form twice in separate requests, and send any record whose two extractions disagree on a field to a human reviewer.
Rerunning a flawed form returns the same wrong value, so both extractions agree and nothing gets flagged.

Exam tip: When extracted values look right but are wrong, make the model show where each value came from, and let code test that evidence.

Sources: platform.claude.com/docs/en/test-and-evaluate/strengthen-guardrails/reduce-h...; Official exam page, Anthropic Partner Academy

SAVE THIS PAGE

CCAR-F exam tips

- 1 Put the stages that must happen in code. Give autonomy only to the stage whose steps you can't predict.
- 2 When parallel subagents return overlapping results, look first at how the coordinator divided the work. Each delegation should state its objective and its boundaries.
- 3 Put any check that must stop an action in PreToolUse. Keep PostToolUse for work that is safe after the tool has run, such as logging or trimming results.
- 4 Never let a tool turn a backend failure into an empty result. Report the failure as an error, and keep empty results for searches that found nothing.
- 5 Put rules that must apply in CI and on every machine in the committed project CLAUDE.md. Keep user-level files for personal taste.
- 6 Start every Claude Code step in CI with -p, and pre-approve the exact commands it must run instead of loosening the whole permission mode.
- 7 Check your few-shot examples for values they all share, and make sure at least one shows each edge case your documents contain.
- 8 When extracted values look right but are wrong, make the model show where each value came from, and let code test that evidence.



Good luck on exam day

Keep practising with our CCAR-F question set. There are more free questions on CertMadeEasy.com and in the app.

[More free CCAR-F questions →](#)

