



Anthropic CCAR-F Practice Questions

Shared by Byrd on 14-10-2026

For More Free Questions and Preparation Resources

Check the Links on Last Page



Question 1

Question Type: MultipleChoice

You are integrating Claude Code into your Continuous Integration/Continuous Deployment (CI/CD) pipeline. The system runs automated code reviews, generates test cases, and provides feedback on pull requests. You need to design prompts that provide actionable feedback and minimize false positives.

Your automated reviewer uses a single prompt covering security issues, API design, and business-logic correctness. Your evaluation suite shows strong recall for API-design findings at 82% but poor recall for business-logic edge cases in quiz scoring at 34%. When you add few-shot examples of logic bugs to the prompt, logic recall improves to 41%, but API-design recall drops to 68%.

How should you address this trade-off to improve detection across both categories?

Options:

- A- Split the review into separate focused prompts---one for security and API design and another for business logic---each with dedicated examples, and then consolidate the findings before posting.
- B- Replace the few-shot examples with a detailed checklist of specific logic edge cases to verify, such as division by zero in score calculations and boundary conditions in grading thresholds.
- C- Upgrade to a more capable model tier because its stronger reasoning will handle both concern types in one prompt and eliminate the recall trade-off.
- D- Provide the full repository as context instead of only the changed files and surrounding code, giving the model deeper visibility into business-logic patterns.

Answer:

A

Explanation:

Option A separates competing objectives so each review call can devote its attention and examples to a coherent concern. Anthropic's Building Effective AI Agents describes sectioning as a parallelization pattern in which independent aspects are handled by separate model calls and then aggregated. It specifically notes that models generally perform better on complex tasks when each consideration receives focused attention. The evaluation results demonstrate prompt interference: optimizing business-logic detection reduces API-design recall.

A security and API-design reviewer can use examples, terminology, and evidence criteria appropriate to interfaces and vulnerabilities. A business-logic reviewer can focus on state transitions, arithmetic boundaries, invariants, and domain-specific edge cases. Their structured findings can then be deduplicated and ranked before posting. Option B may improve only the

weak category and can overfit to known quiz-scoring failures. Option C assumes that model capacity alone removes prompt interference without evaluation evidence. Option D adds substantial unrelated context and does not isolate the competing review objectives. The split prompts should be evaluated independently and together so consolidation does not reintroduce omissions or duplicate findings.

=====

Question 2

Question Type: MultipleChoice

You are building a structured data extraction system using Claude. The system extracts information from unstructured documents, validates the output using JavaScript Object Notation (JSON) schemas, and maintains high accuracy. It must handle edge cases gracefully and integrate with downstream systems.

The system routes documents with extraction confidence below 85% to human review. A quarterly audit reveals that 12% of high-confidence extractions (85%) also contain errors---cases where the model finds plausible-but-incorrect values. Error sources vary: comparison tables showing competitor specs, appendices referencing different product variants, and ambiguous phrasing the model misinterprets. You need a sustainable strategy to catch these high-confidence errors and measure whether improvements reduce the error rate over time.

What approach is most effective?

Options:

- A- Add a verification pass that re-extracts from each high-confidence document, flagging cases where the two extraction attempts produce different results.
- B- Implement heuristic rules that flag documents containing comparison tables or appendices for review regardless of confidence score.
- C- Lower the confidence threshold from 85% to 70%, routing a larger volume of extractions to human review.
- D- Implement stratified random sampling reviewing a fixed percentage of high-confidence extractions weekly, enabling error rate measurement and novel pattern detection.

Answer:

D

Explanation:

Stratified random sampling provides both an ongoing quality-control mechanism and an unbiased measurement framework. By reviewing a fixed proportion of high-confidence outputs across meaningful strata---such as document type, field, source format, and business risk---the organization can estimate the residual error rate, compare performance across releases, and discover failure patterns that were not anticipated when existing rules were designed.

Anthropic recommends measurable success criteria and evaluations that mirror the real-world task distribution, including edge cases. Evaluation volume and repeatability are important because improvements must be demonstrated empirically rather than inferred from isolated examples. (<https://docs.anthropic.com/en/docs/build-with-claude/develop-tests>) A weekly sampling program creates a stable benchmark and allows confidence intervals, trend analysis, regression detection, and error-taxonomy updates.

Option A detects only disagreements between stochastic extractions. Two attempts may produce the same plausible but incorrect answer, so agreement is not equivalent to factual correctness. Option B addresses known patterns but will miss new failure modes and may generate excessive false positives. Option C actually lowers the review threshold in the wrong direction: documents between 70% and 85% would be treated as automated rather than reviewed if the routing rule remains "below threshold," and changing thresholds alone does not measure high-confidence error prevalence.

Official references/topics: Evaluation Design; Representative Test Distributions; Continuous Reliability Measurement; Human-in-the-Loop Sampling.

Question 3

Question Type: MultipleChoice

You are building a structured data extraction system using Claude. The system extracts information from unstructured documents, validates the output using JavaScript Object Notation (JSON) schemas, and maintains high accuracy. It must handle edge cases gracefully and integrate with downstream systems.

Your extraction pipeline processes invoices and extracts line items, subtotals, tax amounts, and grand totals. During evaluation, you discover that in 18% of extractions, the sum of extracted line item amounts doesn't match the extracted grand total---sometimes due to OCR errors in the source document, sometimes due to extraction mistakes by the model. Downstream accounting systems reject records with mismatched totals.

What's the most effective approach to improve extraction reliability?

Options:

A- Add few-shot examples demonstrating invoices where extracted line items sum correctly to the stated total, encouraging the model to produce mathematically consistent extractions.

- B- Extract line items and totals independently, then use a separate validation model to reconcile discrepancies by determining which extracted values are most likely correct.
- C- Implement post-processing that automatically adjusts line item amounts proportionally when their sum doesn't match the stated total.
- D- Add a "calculated_total" field where the model sums extracted line items alongside a "stated_total" field. Flag records for human review when values differ.

Answer:

D

Explanation:

The pipeline must preserve source evidence while making inconsistencies explicit. Option D records the amount stated on the invoice separately from the total derived from extracted line items. A mismatch then becomes a machine-detectable validation condition rather than an invisible extraction defect.

This approach is superior because it does not silently overwrite source data or ask another model to guess which value is correct. Anthropic's evaluation guidance recommends automated, code-based grading whenever the criterion can be expressed deterministically. Arithmetic reconciliation is precisely such a criterion.

(<https://docs.anthropic.com/en/docs/build-with-claude/develop-tests>) In production, the summation should preferably be calculated by application code using normalized decimal values, even though the option describes the model populating calculated_total. The essential design principle remains the same: preserve stated_total, compute an independent total, compare them, and route discrepancies for adjudication.

Option A might improve behavior but cannot resolve genuine OCR corruption and could encourage the model to modify extracted values merely to create mathematical consistency. Option B introduces a second probabilistic judgment without new evidence. Option C is unacceptable for accounting data because it fabricates adjusted amounts and destroys fidelity to the invoice.

The schema should therefore expose both values and attach a validation status or review reason when they differ.

Official references/topics: Deterministic Validation; Human-in-the-Loop Review; Structured Output Design; Source-Fidelity Controls.

Question 4

Question Type: MultipleChoice

After the web-search agent finds 25 sources containing 120,000 tokens of raw content, the

document-analysis agent extracts 15,000 tokens of key insights, and the synthesis agent produces a coherent 3,000-token narrative draft, the coordinator must pass context to the report-generation agent for the final output with proper source citations. What context-passing strategy provides the best balance of completeness and efficiency?

Options:

- A- Pass a condensed summary of all prior stages that preserves the main findings and attributes them to sources by name only.
- B- Pass the synthesis draft together with a structured source index that maps key claims to their source URLs and relevant excerpts.
- C- Pass only the synthesis draft and use a separate post-processing pipeline to match claims to sources and insert citations after report generation.
- D- Pass the complete accumulated context from all prior agents.

Answer:

B

Explanation:

Option B gives the report generator the two forms of information it actually needs: a compact narrative containing the synthesized conclusions and precise evidence records supporting those conclusions. The source index should map claim identifiers to URLs, document locations, excerpts, and other provenance fields so citations can be generated and verified without reloading 120,000 tokens of raw material.

Anthropic's context-engineering guidance recommends distilling extensive subagent activity into high-signal summaries while preserving critical context and dependencies. Its research architecture similarly treats subagents as intelligent filters that compress large evidence collections before returning relevant findings to the coordinator.

Option A lacks sufficient location-level evidence for reliable citation verification. Option C postpones attribution until after drafting, when claim wording may no longer map cleanly to the original evidence. Option D maximizes theoretical completeness but introduces extensive irrelevant material, higher token costs, slower processing, and greater context pollution. The synthesis draft plus a structured claim-to-source index maintains evidentiary completeness where it matters while eliminating intermediate searches, duplicate passages, and other low-value context.

=====

Question 5

Question Type: MultipleChoice

The synthesis agent completes its initial pass but flags that three key research questions remain unanswered because the web-search and document-analysis agents did not find relevant information on those specific subtopics. The coordinator currently proceeds directly to report generation, producing reports with incomplete coverage. What change would most effectively improve research completeness?

Options:

- A- Have the coordinator evaluate the synthesis output for gaps, then redelegate targeted queries to the web-search and document-analysis agents **before** invoking synthesis again.
- B- Have the report-generation agent identify unanswered research questions so users understand the limitations of the final output.
- C- Increase the initial breadth of queries sent to the web-search and document-analysis agents to reduce the probability of missing relevant information.
- D- Give the synthesis agent direct access to web-search tools so it can autonomously fill knowledge gaps without returning control to the coordinator.

Answer:

A

Explanation:

Option A implements an iterative orchestrator-worker loop in which synthesis is treated as an evaluation checkpoint rather than an irreversible transition to report generation. When synthesis identifies missing evidence, the coordinator can formulate focused follow-up tasks, send them to the agents with the appropriate tools, and repeat synthesis after receiving the additional findings.

Anthropic's multi-agent research architecture follows this pattern: the lead agent synthesizes subagent findings and determines **whether further research is required**. If gaps remain, it creates additional subagents or **refines the research strategy**. This preserves specialization and centralized control.

Option B makes the incompleteness visible but does not improve research coverage. Option C may increase initial cost without guaranteeing that the specific gaps discovered during synthesis will be addressed. Option D weakens separation of concerns by giving the synthesis agent search capabilities that belong to the research specialists. Targeted re delegation is more efficient because the second research round is informed by concrete deficiencies rather than speculative breadth. It also allows the coordinator to track completeness explicitly before authorizing final report generation.

=====

Question 6

Question Type: MultipleChoice

You are using Claude Code to accelerate software development. Your team uses it for code generation, refactoring, debugging, and documentation. You need to integrate it into your development workflow with custom slash commands, CLAUDE.md configurations, and understand when to use plan mode vs direct execution.

Your team wants Claude to follow a detailed code review checklist (8 items covering API changes, test coverage, documentation, security, etc.) when reviewing pull requests. The team also uses Claude extensively for other tasks: writing new features, debugging production issues, and generating documentation. Currently, developers paste the checklist at the start of each review session.

Which approach best addresses this workflow need?

Options:

- A- Create a /review slash command containing the checklist, invoked when starting reviews.
- B- Create a dedicated review subagent with the checklist embedded in its configuration.
- C- Add the checklist to the project's CLAUDE.md file under a "Code Review" section.
- D- Configure plan mode as the default for code review sessions.

Answer:

A

Explanation:

The checklist is a reusable, task-specific procedure that should load only when a pull-request review is being performed. A /review command provides an explicit invocation mechanism, prevents repeated copy-and-paste, and keeps the checklist out of unrelated coding, debugging, and documentation sessions.

In current Claude Code terminology, custom commands have been consolidated into skills. A .claude/skills/review/SKILL.md file creates a /review command, while the legacy .claude/commands/review.md format remains supported. Anthropic recommends skills when users repeatedly supply the same checklist or multi-step procedure; unlike CLAUDE.md content, the skill body is loaded only when invoked or determined relevant. (<https://code.claude.com/docs/en/skills>)

Option B introduces an isolated agent context when the requirement is primarily to inject a repeatable review procedure. A subagent could be useful for independent verification, but it is not necessary merely to avoid repasting the checklist. Option C bloats every session with instructions that apply only to code review. Option D controls editing permissions and planning

behavior; it does not encode review criteria.

The review skill can also pre-authorize read-only tools and accept a pull-request identifier or branch as an argument.

Official references/topics: Claude Code Skills; Custom Slash Commands; On-Demand Instructions; Reusable Review Workflows.

Question 7

Question Type: MultipleChoice

You are using Claude Code to accelerate software development. Your team uses it for code generation, refactoring, debugging, and documentation. You need to integrate it into your development workflow with custom slash commands, CLAUDE.md configurations, and understand when to use plan mode vs direct execution.

You're implementing a complex graph traversal algorithm with specific performance requirements and edge cases to handle (disconnected nodes, cycles, weighted edges). You want to structure your workflow for efficient iterative refinement with Claude.

What approach will most effectively enable progressive improvement across multiple iterations?

Options:

- A- Have Claude extensively research the algorithm and create a detailed implementation plan using extended thinking, then implement the complete solution based on that plan.
- B- Provide Claude with a reference implementation from documentation, then ask it to rewrite the code to match your codebase style and add the required edge case handling, comparing outputs against the reference.
- C- Write a test suite covering expected behavior, edge cases, and performance requirements before implementation. Ask Claude to write code that passes the tests, then iterate by sharing test failures with each refinement request.
- D- Provide Claude with a detailed natural language specification of the algorithm, including all requirements and edge cases. Review each output manually and provide descriptive feedback on what behavior needs to change.

Answer:

C

Explanation:

Option C creates an objective verification loop. The tests encode expected traversal behavior for disconnected graphs, cycle handling, weighted edges, invalid inputs, and performance constraints. Claude can implement the algorithm, execute the suite, inspect concrete failures, and refine the implementation until the measurable conditions pass.

Anthropic emphasizes giving Claude a verification mechanism such as tests, builds, linters, or fixture comparisons. Without an executable pass-or-fail check, Claude can only determine that an implementation appears complete. With tests, it can perform work, evaluate the result, and iterate using evidence rather than subjective judgment. Anthropic also recommends reproducing defects with failing tests before applying corrections.
(<https://code.claude.com/docs/en/best-practices>)

Option A may produce a thoughtful initial design but does not guarantee progressive improvement after implementation. Option B risks inheriting assumptions or deficiencies from a reference that may not match the project's constraints. Option D depends on manual review and converts the developer into the primary verification system.

The test suite should include correctness fixtures, boundary cases, complexity-sensitive workloads, and regression tests added whenever a new failure is discovered. This makes every iteration cumulative: a correction must satisfy the new case without breaking previously validated behavior.

Official references/topics: Executable Verification; Test-Driven Iteration; Feedback Loops; Regression Testing.



To Get Premium Files for CCAR-F Visit

<https://www.p2pexams.com/products/ccar-f>

For More Free Questions Visit

<https://www.p2pexams.com/anthropic/pdf/ccar-f>

20%
DISCOUNT

P2P
exams