Exam

CCDV-F

Claude Certified Developer - Foundations (CCDV-F)

Free demo	17 sample questions
Full exam	83 questions with answers and explanations
Contents	Questions with answers and explanations

Exam domains

Domain	Exam weight	Questions
D1 Agents and Workflows		2
D2 Applications and Integration		3
D3 Claude Code		2
D4 Eval, Testing, and Debugging		1
D5 Model Selection and Optimization		2
D6 Prompt and Context Engineering		3
D7 Security and Safety		2
D8 Tools and MCPs		2

This is a free demo from Dumpsgate: 17 of the exam's 83 questions. **Get the full CCDV-F exam** with every question, answer and explanation.

Case study

This section contains standalone questions (no scenario).

Exam Syllabus & Objectives

Exam Overview

The Anthropic CCDV-F (Claude Certified Developer - Foundations) certification validates your ability to design, develop, and deploy applications powered by Claude. This exam tests practical knowledge across the full stack of Claude application development, from architectural decisions and API integration to security practices and cost optimization.

Knowledge Domains and Relative Weighting

● Applications and Integration (33.1%) — The largest domain, covering how to translate business

requirements into Claude-based solutions. You must understand functional and infrastructure needs, system lifecycle management, the Claude Messages API with all its features (streaming, vision, caching, batch processing), core software engineering practices, application design patterns, and configuration management strategies for Claude systems.

● Model Selection and Optimization (16.8%) — Understanding the characteristics and

performance profiles of Claude models. This includes foundational LLM concepts (tokenization, context windows, sampling behavior), when to select Opus versus Sonnet versus Haiku, how extended and adaptive thinking impact quality and cost, prompting techniques, and cost management through token budgeting and caching strategies.

● Agents and Workflows (14.7%) — Designing and building agent-based systems with Claude.

You need to understand when and why to use agents versus simpler workflows, how to architect multi-agent hierarchies, the tools and SDKs for agent construction, common agentic design patterns, and popular frameworks that support agent development.

● Prompt and Context Engineering (11.0%) — Mastering the art of writing effective prompts and

managing context efficiently. This covers prompt clarity and structure, few-shot learning, system versus user message placement, preventing context drift, validating and parsing model outputs defensively, and iterative refinement techniques.

● Tools and MCPs (10.6%) — Extending Claude's capabilities through external integrations. You

must understand how to implement and describe tools, configure them for reliable function calling, deploy MCP servers, handle tool errors gracefully, and select the right approach (built-in tools, custom tools, Skills, or MCPs) for different scenarios.

● Security and Safety (8.1%) — Deploying Claude applications responsibly. This encompasses

data privacy, prompt injection and jailbreak mitigation, safe input handling, PII protection, secure credential management, layered guardrail deployment, and using hooks to enforce safety controls.

● Claude Code (3.1%) — Working with Claude's code development environment. You need to

know how Claude Code operates, its core components (Rules, Skills, Commands, Agents, Agent Memory), session management, configuration files like CLAUDE.md and settings.json, and operations appropriately

● Build and configure agents and workflows for multi-step task execution, including hierarchical

and sub-agent patterns

● Select and configure Claude models based on quality, latency, and cost tradeoffs for specific

use cases

● Write clear, well-structured prompts and manage context effectively to avoid drift and token

waste

● Implement tools and integrate Model Context Protocol servers for external system interaction

Your team is building a multi-step document processing application that needs to extract structured data from PDFs, validate the extracted content against business rules, and then route documents to different approval workflows based on content type. You are evaluating whether to use an agent or a workflow architecture.

Which of the following factors should most strongly influence your decision toward using an agent rather than a workflow?

A. The approval routing requires dynamic decision-making based on unpredictable document variations and emerging patterns that were not pre-defined in the system

B. The extraction, validation, and routing steps are deterministic and always follow the same sequence regardless of document content

C. The application needs to maintain strict audit compliance and every action must be explicitly logged and repeatable

D. The team prefers working with declarative, step-by-step orchestration that clearly maps each stage in the document pipeline

E. The PDF processing step is computationally expensive and should be cached and reused across multiple documents

Answer: A

Agents excel at handling dynamic, unpredictable scenarios where the system must adapt its behavior based on emerging patterns and make autonomous decisions. In this scenario, the need for dynamic routing based on document variations that weren't pre-defined is a key indicator that an agent's iterative decision-making loop would be more appropriate than a rigid workflow.

The other options point toward workflow characteristics: deterministic sequences (workflows), strict audit requirements (workflows provide better determinism and traceability), declarative step-by-step orchestration (workflows by design), and caching concerns (an optimization orthogonal to the agent vs. workflow choice). Agents are more flexible but less predictable and auditable, while workflows provide explicit control and repeatability.

Question 1

Single choice · Domain: D1 Agents and Workflows › Choose between agent and workflow architectures based on process determinism and adaptability

SINGLE CHOICE

Your team is building a customer support system where requests need to be classified, routed to specialized handlers, and tracked across multiple steps. You're deciding between implementing this as an agent versus a workflow. Which of the following factors most strongly suggests using a workflow instead of an agent for this scenario?

A. The system needs to handle dynamic, unpredictable user requests that may require tool decisions at runtime

B. The process follows a well-defined, deterministic sequence with clear entry and exit points that rarely change

C. The system must learn from past interactions and adapt its behavior based on user feedback

D. Individual subtasks may fail and need intelligent retry logic with context awareness

Answer: B. The process follows a well-defined, deterministic sequence with clear entry and exit points that rarely change

Explanation

This question tests whether the candidate can identify when a deterministic, predefined sequence is better suited to a workflow than to an autonomous agent. Workflows provide explicit orchestration and predictable execution, whereas agents are more appropriate when runtime decisions and adaptation are

Free demo · full exam: 83 questions

required.

✗ **A** The system needs to handle dynamic, unpredictable user requests that may require tool decisions at runtime
Incorrect. Dynamic requests and runtime tool selection are characteristics that favor an agent, because agents can determine actions and adapt their behavior during execution.
https://www.anthropic.com/research/building-effective-agents

✓ **B** The process follows a well-defined, deterministic sequence with clear entry and exit points that rarely change
Correct. A stable, deterministic sequence with explicit stages and predictable entry and exit conditions is best implemented as a workflow, which provides direct orchestration and repeatable execution.
https://www.anthropic.com/research/building-effective-agents

✗ **C** The system must learn from past interactions and adapt its behavior based on user feedback
Incorrect. Learning from interactions and adapting behavior indicates a need for flexible, agent-like behavior rather than a fixed workflow.
https://www.anthropic.com/research/building-effective-agents

✗ **D** Individual subtasks may fail and need intelligent retry logic with context awareness
Incorrect. Context-aware retries and intelligent recovery require runtime judgment and adaptation, which are more strongly associated with agents. A workflow can support predefined retries, but intelligent context-dependent retry decisions favor an agent.
https://www.anthropic.com/research/building-effective-agents

Question 2

Single choice · Domain: D1 Agents and Workflows › Design supervisory controls for multi-agent delegation, including budgets, output validation, and explicit handoff contracts

SINGLE CHOICE

A team has deployed a multi-agent system in which a primary agent decomposes user requests and delegates subtasks to three specialized subagents: one for data retrieval, one for analysis, and one for report generation. In production, the team observes that subagents are making redundant tool calls, occasionally exceeding token budgets, and sometimes producing outputs that contradict each other --- all of which the primary agent passes along without catching.

What is the most appropriate way to address these failures?

A. Add retry logic to each subagent so that when a tool call fails, the subagent retries automatically before escalating --- and configure each subagent to log its tool calls and outputs to a shared trace so the team can audit redundancy and contradictions after the fact.

B. Give each subagent read access to the other subagents' outputs so they can identify and resolve contradictions without routing back through the primary agent.

C. Strengthen the primary agent's management layer to enforce per-subagent tool budgets, validate outputs against a defined schema before passing them forward, and establish explicit handoff contracts between stages.

D. Collapse the three subagents into a single large-context model call that handles retrieval, analysis, and generation in one pass.

Answer: C. Strengthen the primary agent's management layer to enforce per-subagent tool budgets, validate outputs against a defined schema before passing them forward, and establish explicit handoff contracts between stages.

Free demo · full exam: 83 questions

Dumpsgate

Question 3

Single choice · Domain: D2 Applications and Integration › Describe and configure default retry behavior in Claude SDKs for transient API errors

SINGLE CHOICE

A teammate has asked how the Claude SDK handles transient API errors, such as a temporary network issue or a brief rate-limit response. They want to know whether the application code needs to handle every transient error or whether the SDK provides any default behavior.

How would you describe the SDK's default behavior?

A. The SDK provides default retry behavior for transient errors up to a fixed number of attempts, and this behavior is not configurable.

B. The SDK provides default retry behavior for network errors but surfaces rate-limit responses directly to the application code, which must implement its own retry logic for those cases.

C. The SDK logs transient errors to a default error stream and continues execution without retrying, leaving the application code responsible for detecting and responding to failed calls.

D. The SDK provides default retry behavior for many transient errors, and the application code can configure or extend that behavior as needed. responses are among the conditions automatically retried by the supported SDKs; the API also provides retry-after information where applicable. C is incorrect because the SDK does not merely log these failures and continue without attempting recovery. Application-level handling is still necessary after the configured retry policy is exhausted. Production code should distinguish permanent errors from transient failures, preserve request IDs for diagnostics, and decide whether higher-level fallback, queuing, or user-visible failure handling is appropriate. Therefore, D correctly characterizes the SDK abstraction: useful retry defaults combined with developer control. Relevant Study Guide topics: Claude SDKs, transient errors, HTTP 429, retries, exponential backoff, configuration, and resilient API integration.

Answer: D. The SDK provides default retry behavior for many transient errors, and the application code can configure or extend that behavior as needed. responses are among the conditions automatically retried by the supported SDKs; the API also provides retry-after information where applicable. C is incorrect because the SDK does not merely log these failures and continue without attempting recovery. Application-level handling is still necessary after the configured retry policy is exhausted. Production code should distinguish permanent errors from transient failures, preserve request IDs for diagnostics, and decide whether higher-level fallback, queuing, or user-visible failure handling is appropriate. Therefore, D correctly characterizes the SDK abstraction: useful retry defaults combined with developer control. Relevant Study Guide topics: Claude SDKs, transient errors, HTTP 429, retries, exponential backoff, configuration, and resilient API integration.

Explanation

This question tests knowledge of how Claude SDKs handle transient failures such as network errors and rate limits, including built-in retries and developer-configurable retry policies.

✗ **A** The SDK provides default retry behavior for transient errors up to a fixed number of attempts, and this behavior is not configurable.
Incorrect. Claude SDKs provide default retries, but the retry policy is configurable, including the maximum number of retries.

https://docs.anthropic.com/en/api/client-sdks

Free demo · full exam: 83 questions

✗ **B** The SDK provides default retry behavior for network errors but surfaces rate-limit responses directly to the application code, which must implement its own retry logic for those cases.
Incorrect. Rate-limit responses such as HTTP 429 errors are among the transient conditions that supported SDKs can retry automatically; the API also provides retry-related information where applicable.
https://docs.anthropic.com/en/api/errors

✗ **C** The SDK logs transient errors to a default error stream and continues execution without retrying, leaving the application code responsible for detecting and responding to failed calls.
Incorrect. The SDK does not merely log transient failures and continue; it performs retry attempts according to its retry policy and raises an error if the request still fails.
https://docs.anthropic.com/en/api/client-sdks

✓ **D** The SDK provides default retry behavior for many transient errors, and the application code can configure or extend that behavior as needed. responses are among the conditions automatically retried by the supported SDKs; the API also provides retry-after information where applicable. C is incorrect because the SDK does not merely log these failures and continue without attempting recovery. Application-level handling is still necessary after the configured retry policy is exhausted. Production code should distinguish permanent errors from transient failures, preserve request IDs for diagnostics, and decide whether higher-level fallback, queuing, or user-visible failure handling is appropriate. Therefore, D correctly characterizes the SDK abstraction: useful retry defaults combined with developer control.
Relevant Study Guide topics: Claude SDKs, transient errors, HTTP 429, retries, exponential backoff, configuration, and resilient API integration.
Correct. Claude SDKs provide default retry behavior for transient failures, while allowing developers to configure retry behavior and handle failures that remain after retries are exhausted.
https://docs.anthropic.com/en/api/client-sdks

Question 4

Single choice · Domain: D2 Applications and Integration › Manage environment-specific Claude application configuration through version-controlled configuration management

SINGLE CHOICE

Your Claude application is deployed to development, staging, and production environments.

Each environment uses a different model version, different prompt versions, and different plugin dependencies, but the configuration is currently scattered across environment variables, hardcoded values, and undocumented setup scripts.

How would you manage the configuration?

A. Use the latest available model version everywhere and stop pinning model versions, on the grounds that pinning adds maintenance overhead the team should aim to reduce.

B. Move all configuration into hardcoded application code to reduce reliance on external configuration sources that are difficult to track over time.

C. Consolidate the configuration into a version-controlled system documenting model version pinning, prompt versioning, and plugin dependencies for each environment.

D. Standardize all environments to use the same configuration values to eliminate the differences between development, staging, and production.

Answer: C. Consolidate the configuration into a version-controlled system documenting model version pinning, prompt versioning, and plugin dependencies for each environment.

Dumpsgate

Question 5

Single choice · Domain: D2 Applications and Integration › Select an appropriate Claude API integration approach that supports maintainability and future provider portability

SINGLE CHOICE application can switch providers in the future.

A. Skip the SDK and embed Claude calls in shell commands invoked from Python, so that the application runs the calls outside the main Python process.

B. Use Anthropic's supported client SDK behind a small application-owned provider interface or adapter, keeping Claude-specific request/response logic isolated so another provider can be substituted without rewriting the application's business logic

Answer: B. Use Anthropic's supported client SDK behind a small application-owned provider interface or adapter, keeping Claude-specific request/response logic isolated so another provider can be substituted without rewriting the application's business logic

Question 6

Single choice · Domain: D3 Claude Code › Configure repository-level Claude Code instructions using CLAUDE.md

SINGLE CHOICE

You are setting up Claude Code for a new project repository. Your team has shared coding standards, preferred libraries, and project-specific context that every developer working on the repository should have available when they use Claude Code.

How would you set this up?

A. Create a personal ~/.claude/CLAUDE.md file on each developer's computer and ask every team member to keep it synchronized manually.

B. Add the standards to the repository README.md and require developers to paste the relevant sections into Claude Code at the beginning of each session.

C. Create a project-level CLAUDE.md at the repository root (or .claude/CLAUDE.md) containing the repository's shared coding standards, preferred libraries, commands, architecture, and relevant context, and commit it to version control for the team.

D. Create a shell script that launches Claude Code with the team's standards embedded in command-line arguments.

Answer: C. Create a project-level CLAUDE.md at the repository root (or .claude/CLAUDE.md) containing the repository's shared coding standards, preferred libraries, commands, architecture, and relevant context, and commit it to version control for the team.

Explanation

This question tests how to make shared coding standards, preferred libraries, and repository-specific context available to Claude Code for all developers working in a project repository.

✗ **A** Create a personal ~/.claude/CLAUDE.md file on each developer's computer and ask every team member to keep it synchronized manually.

✗ **B** Add the standards to the repository README.md and require developers to paste the relevant sections into Claude Code at the beginning of each session.

✓ **C** Create a project-level CLAUDE.md at the repository root (or .claude/CLAUDE.md) containing the repository's shared coding standards, preferred libraries, commands, architecture, and relevant context, and commit it to version control for the team.
Project-level CLAUDE.md files provide persistent repository-specific instructions that Claude Code loads for work in that project. Keeping the project instructions with the repository makes them available consistently to collaborators.

✗ **D** Create a shell script that launches Claude Code with the team's standards embedded in command-line arguments.

Free demo · full exam: 83 questions

Question 7

Single choice · Domain: D3 Claude Code › Configure and invoke custom Skills in Claude Code

SINGLE CHOICE

A teammate has asked how to extend Claude Code with a custom Skill that the team can invoke during sessions. The Skill consists of a set of instructions and a few support scripts the team wants Claude to be able to call when the Skill is loaded. manually before each Claude Code session that needs it.

A. Store the instructions in a project README.md file and ask users to paste the relevant section into Claude Code whenever they need the Skill.

B. Put the scripts in a scripts/ directory and add their paths to the system prompt at the start of every Claude Code session.

C. Create a project Skill at .claude/skills/<skill-name>/SKILL.md (or a personal Skill at ~/.claude/skills/<skill-name>/SKILL.md), keep supporting scripts/resources in the Skill directory and reference them from SKILL.md; Claude can load the Skill when relevant or the user can invoke it with /skill-name.

D. Create a shell alias that launches Claude Code with the Skill instructions passed as command-line text each time.

Answer: C. Create a project Skill at .claude/skills/<skill-name>/SKILL.md (or a personal Skill at ~/.claude/skills/<skill-name>/SKILL.md), keep supporting scripts/resources in the Skill directory and reference them from SKILL.md; Claude can load the Skill when relevant or the user can invoke it with /skill-name.

Explanation

This question tests knowledge of extending Claude Code with custom Skills containing instructions and supporting scripts, including how Skills are made available during sessions.

✗ **A** Store the instructions in a project README.md file and ask users to paste the relevant section into Claude Code whenever they need the Skill.

✗ **B** Put the scripts in a scripts/ directory and add their paths to the system prompt at the start of every Claude Code session.

✓ **C** Create a project Skill at .claude/skills/<skill-name>/SKILL.md (or a personal Skill at ~/.claude/skills/<skill-name>/SKILL.md), keep supporting scripts/resources in the Skill directory and reference them from SKILL.md; Claude can load the Skill when relevant or the user can invoke it with /skill-name.
Claude Code Skills are directory-based packages centered on a SKILL.md file. They can include additional scripts and resources and may be discovered automatically or invoked directly, depending on their configuration and use.

✗ **D** Create a shell alias that launches Claude Code with the Skill instructions passed as command-line text each time.

Question 8

Single choice · Domain: D4 Eval, Testing, and Debugging › Design CI/CD pipelines that automate Claude application testing and security validation

SINGLE CHOICE

You are setting up a CI/CD pipeline for a new Claude application. The pipeline needs to run

A. Automated linting and security scanning, with Claude integration testing handled manually during pre- release verification by a designated reviewer.

B. Automate Claude application/evaluation regression tests in CI together with appropriate linting and security checks, and gate deployment on defined quality/security thresholds while retaining human review for cases that genuinely require judgment

Answer: B. Automate Claude application/evaluation regression tests in CI together with appropriate linting and security checks, and gate deployment on defined quality/security thresholds while retaining human review for cases that genuinely require judgment

Explanation

This question tests the ability to distinguish appropriate automated validation in a CI/CD pipeline from manual-only verification. Claude application integration tests should be repeatable and automated where feasible, with evaluation and testing integrated into the development lifecycle.

✗ **A** Automated linting and security scanning, with Claude integration testing handled manually during pre- release verification by a designated reviewer.
Incorrect. Although automated linting and security scanning are appropriate, relying exclusively on manual Claude integration testing weakens repeatability, regression detection, and deployment safety. Claude application evaluations and tests should be incorporated into an iterative testing process rather than deferred entirely to a manual pre-release review.

https://docs.anthropic.com/en/docs/test-and-evaluate/overview

✓ **B** Automate Claude application/evaluation regression tests in CI together with appropriate linting and security checks, and gate deployment on defined quality/security thresholds while retaining human review for cases that genuinely require judgment
Anthropic recommends repeatable evaluations using representative test cases and automating evaluation workflows where practical. Running those tests during CI helps detect behavioral regressions before deployment rather than depending only on a final manual integration review.

Dumpsgate

Question 9

Single choice · Domain: D5 Model Selection and Optimization › Select a Claude model based on task complexity, latency requirements, and per-request cost

SINGLE CHOICE

You are choosing a Claude model for a high-volume classification task. Each classification is straightforward, latency requirements are tight, and per-request cost matters at scale.

Which model would you choose?

A. A mid-tier Claude model selected by default, because mid-tier models balance quality and cost in a way the team can apply across most tasks.

B. Multiple Claude models in series, where each request runs through more than one model and the application combines the outputs into a final classification.

C. A smaller, faster Claude model, because the task is straightforward and the workload prioritizes latency and per-request cost at scale.

D. The largest, highest-capability Claude model, to maximize quality on every classification the application produces during normal operation across all requests.

Answer: C. A smaller, faster Claude model, because the task is straightforward and the workload prioritizes latency and per-request cost at scale.

Explanation

This question tests the ability to match model capability to a straightforward, high-volume classification workload while prioritizing low latency and cost efficiency. The appropriate choice is a smaller, faster model rather than defaulting to a larger model for every request.

✗ **A** A mid-tier Claude model selected by default, because mid-tier models balance quality and cost in a way the team can apply across most tasks.
Incorrect. Choosing a mid-tier model by default ignores the specific workload characteristics. For straightforward classification with tight latency and high request volume, a smaller model is generally more appropriate when it satisfies the required quality.

https://docs.anthropic.com/en/docs/about-claude/models/overview

✗ **B** Multiple Claude models in series, where each request runs through more than one model and the application combines the outputs into a final classification.
Incorrect. Running multiple models in series would add latency and increase cost, which conflicts with the stated requirements for tight latency and low per-request cost.

https://docs.anthropic.com/en/docs/about-claude/models/overview

✓ **C** A smaller, faster Claude model, because the task is straightforward and the workload prioritizes latency and per-request cost at scale.
Correct. A smaller, faster model is the best fit for a simple classification task when the application prioritizes low latency and cost efficiency across a high request volume, assuming the model meets the required accuracy.

https://docs.anthropic.com/en/docs/about-claude/models/overview

✗ **D** The largest, highest-capability Claude model, to maximize quality on every classification the application produces during normal operation across all requests.
Incorrect. Using the largest model for every request would likely increase cost and latency without providing proportional value for a straightforward classification task. Model selection should account for the required quality, latency, and cost tradeoffs.

https://docs.anthropic.com/en/docs/about-claude/models/overview

Question 10

Single choice · Domain: D5 Model Selection and Optimization › Identify the need to clarify quality, latency, and cost requirements before selecting a Claude model

SINGLE CHOICE

The product team has asked you to choose a Claude model for a new feature. The team has provided functional requirements but has not specified performance, cost, or quality targets. The team's product manager says, "Use whatever model gives us the best results." How would you respond?

A. First define measurable quality, capability, latency, and cost requirements, then benchmark candidate Claude models on representative prompts and data and select the model and reasoning configuration that meets those targets

B. The extraction, validation, and routing steps are deterministic and always follow the same sequence regardless of document content

C. The application needs to maintain strict audit compliance and every action must be explicitly logged and repeatable

D. The team prefers working with declarative, step-by-step orchestration that clearly maps each stage in the document pipeline

E. The PDF processing step is computationally expensive and should be cached and reused across multiple documents

Answer: A. First define measurable quality, capability, latency, and cost requirements, then benchmark candidate Claude models on representative prompts and data and select the model and reasoning configuration that meets those targets

Explanation

This question tests whether the candidate recognizes that model selection requires explicit evaluation criteria rather than an ambiguous request to use the model with the best results. Functional requirements alone are insufficient to determine the appropriate quality, performance, and cost tradeoff.

✓ **A** First define measurable quality, capability, latency, and cost requirements, then benchmark candidate Claude models on representative prompts and data and select the model and reasoning configuration that meets those targets
Anthropic's model-selection guidance recommends beginning with the application's actual requirements and benchmarking candidate models against relevant criteria such as capability, speed, cost, and required reasoning effort.

✗ **B** The extraction, validation, and routing steps are deterministic and always follow the same sequence regardless of document content
Incorrect for the model-selection question. This is a characteristic of workflow architecture and does not establish which Claude model should be selected.

https://docs.anthropic.com/en/docs/about-claude/models/overview

✗ **C** The application needs to maintain strict audit compliance and every action must be explicitly logged and repeatable
Incorrect for the model-selection question. Auditability and repeatability are architectural and operational requirements, not sufficient criteria for choosing among Claude models.

https://docs.anthropic.com/en/docs/about-claude/models/overview

✗ **D** The team prefers working with declarative, step-by-step orchestration that clearly maps each stage in

the document pipeline
Incorrect for the model-selection question. This preference supports a workflow design but does not answer how to handle an unspecified model-quality request.
https://docs.anthropic.com/en/docs/about-claude/models/overview

✗ **E** The PDF processing step is computationally expensive and should be cached and reused across multiple documents
Incorrect for the model-selection question. Caching is an optimization concern and does not determine which model provides the best result for the product's actual quality, latency, and cost requirements.
https://docs.anthropic.com/en/docs/about-claude/models/overview

---

Question 11

Single choice · Domain: D6 Prompt and Context Engineering › Apply context engineering techniques to preserve task state and reduce context growth in long-running agentic workflows

SINGLE CHOICE

Your Claude application runs long agentic workflows where the agent makes many tool calls, and the conversation history grows quickly. After about 20 tool calls, you notice the agent's responses become less focused and sometimes ignore earlier task constraints.

How would you address this?

A. Remove tool calling from the workflow entirely so the agent operates as a single text-generation step with no tool outputs accumulating in the context window.

B. Apply context engineering techniques such as tool output pruning or compaction to keep the active task state visible while reducing the volume of older content.

C. Increase the model's context window so the agent can hold every tool output at full detail across the entire workflow no matter how many tool calls it accumulates.

D. Restart the agent every five tool calls to prevent any drift, with the agent losing all task state at each restart point during the workflow. with heavy tool usage. Once Claude has processed an older result, verbose file contents, search output, or API responses may no longer need to remain in full. Older results can therefore be removed while recent and important state remains available. Anthropic also documents compaction, where accumulated history is summarized and replaced with a smaller representation that preserves important task state. A removes an important agent capability rather than solving context growth. C retains all accumulated noise and merely gives it more space. D repeatedly destroys valuable task state and creates artificial workflow boundaries. Therefore, B preserves goals, decisions, unresolved constraints, and necessary results while reducing irrelevant historical content. Relevant Study Guide topics: context engineering, tool-result clearing, compaction, long-running agents, context quality, and task-state preservation.

Answer: B. Apply context engineering techniques such as tool output pruning or compaction to keep the active task state visible while reducing the volume of older content.

Explanation

This question tests the ability to manage rapidly expanding conversation history from repeated tool calls. The relevant techniques include pruning obsolete tool results and compacting accumulated history while retaining goals, constraints, decisions, and necessary outputs.

✗ **A** Remove tool calling from the workflow entirely so the agent operates as a single text-generation step with no tool outputs accumulating in the context window.
Incorrect. Removing tool calling eliminates a core agent capability instead of managing the context generated by tool use. Context engineering should reduce or summarize obsolete content while preserving the tools required for the task.

https://docs.anthropic.com/en/docs/build-with-claude/context-windows

✓ **B** Apply context engineering techniques such as tool output pruning or compaction to keep the active task state visible while reducing the volume of older content.
Correct. Pruning verbose or no-longer-needed tool results and compacting conversation history can reduce context noise while retaining important goals, constraints, decisions, and recent results. This directly addresses loss of focus caused by context growth.

https://docs.anthropic.com/en/docs/build-with-claude/context-windows

✗ **C** Increase the model's context window so the agent can hold every tool output at full detail across the entire workflow no matter how many tool calls it accumulates.
Incorrect. A larger context window does not prevent irrelevant or verbose historical content from degrading context quality, and context windows remain finite. The better approach is to manage and compact accumulated context.

https://docs.anthropic.com/en/docs/build-with-claude/context-windows

✗ **D** Restart the agent every five tool calls to prevent any drift, with the agent losing all task state at each restart point during the workflow. with heavy tool usage. Once Claude has processed an older result, verbose file contents, search output, or API responses may no longer need to remain in full. Older results can therefore be removed while recent and important state remains available. Anthropic also documents compaction, where accumulated history is summarized and replaced with a smaller representation that preserves important task state. A removes an important agent capability rather than solving context growth. C retains all accumulated noise and merely gives it more space. D repeatedly destroys valuable task state and creates artificial workflow boundaries. Therefore, B preserves goals, decisions, unresolved constraints, and necessary results while reducing irrelevant historical content. Relevant Study Guide topics: context engineering, tool-result clearing, compaction, long-running agents, context quality, and task-state preservation.
Incorrect. Repeatedly restarting the agent discards useful task state and can cause loss of goals, constraints, prior decisions, and necessary results. Context compaction or selective removal preserves continuity while controlling context size.

https://docs.anthropic.com/en/docs/build-with-claude/context-windows

Question 12

Single choice · Domain: D6 Prompt and Context Engineering › Design validation and confidence-calibration measures that prevent unverified Claude outputs from being treated as factual ground truth

SINGLE CHOICE

Your Claude application returns confident-sounding answers, but occasionally those answers contain factual errors that downstream systems treat as ground truth. The team is concerned about the application's confidence-versus-accuracy gap.

How would you address the gap?

A. Lower the model's temperature so the model's responses sound less confident and downstream systems are less likely to treat the responses as ground truth in normal operation.

B. Apply skepticism toward confident output by adding validation steps, sourcing requirements, or confidence calibration before treating outputs as ground truth.

C. Reject every response the application produces until a manual accuracy review is conducted on each response by a human reviewer before any downstream system uses it.

D. Add a disclaimer to every output telling users to verify the accuracy of the output and treat the disclaimer as the primary mechanism for managing the confidence-versus-accuracy gap.

Answer: B. Apply skepticism toward confident output by adding validation steps, sourcing requirements, or confidence calibration before treating outputs as ground truth.

Explanation

This question tests defensive handling of model outputs when fluent or confident responses may contain factual errors. The appropriate approach is to add verification, sourcing, and calibration steps before downstream systems rely on the output.

✗ **A** Lower the model's temperature so the model's responses sound less confident and downstream systems are less likely to treat the responses as ground truth in normal operation.
Incorrect. Lowering temperature can reduce sampling variability, but it does not establish factual accuracy or reliably prevent confident hallucinations. Outputs still require verification and appropriate guardrails.

https://docs.anthropic.com/en/docs/test-and-evaluate/strengthen-guardrails/reduce-hallucinations

✓ **B** Apply skepticism toward confident output by adding validation steps, sourcing requirements, or confidence calibration before treating outputs as ground truth.
Correct. Confidence in wording is not evidence of correctness. Validation, requiring supporting sources, and calibrating or reviewing confidence before accepting an answer are appropriate ways to reduce the confidence-versus-accuracy gap.

https://docs.anthropic.com/en/docs/test-and-evaluate/strengthen-guardrails/reduce-hallucinations

✗ **C** Reject every response the application produces until a manual accuracy review is conducted on each response by a human reviewer before any downstream system uses it.
Incorrect. Human review may be appropriate for high-risk or uncertain cases, but requiring manual review of every response is unnecessarily rigid and may make the application impractical. Risk-based automated validation and escalation are generally more suitable.

https://docs.anthropic.com/en/docs/test-and-evaluate/strengthen-guardrails/reduce-hallucinations

✗ **D** Add a disclaimer to every output telling users to verify the accuracy of the output and treat the disclaimer as the primary mechanism for managing the confidence-versus-accuracy gap.
Incorrect. A disclaimer may communicate limitations, but it does not validate content or prevent downstream systems from consuming incorrect answers as ground truth. Verification and guardrails

must be implemented in the application flow.
https://docs.anthropic.com/en/docs/test-and-evaluate/strengthen-guardrails/reduce-hallucinations

Question 13

Single choice · Domain: D6 Prompt and Context Engineering › Mitigate context drift by managing and prioritizing conversation context

SINGLE CHOICE

Your Claude application has been running for several conversation turns, and you notice the model occasionally references information that was discussed many turns ago but is no longer relevant. You suspect context drift is causing the model to weight stale content too heavily.

How would you address the drift?

A. Summarize or prune stale conversation turns, retain only relevant context, and explicitly restate the current task and priorities in the prompt

Answer: A. Summarize or prune stale conversation turns, retain only relevant context, and explicitly restate the current task and priorities in the prompt

Explanation

This question tests the ability to prevent stale conversation content from disproportionately influencing Claude by summarizing, pruning, or restructuring the context and clearly emphasizing current requirements.

✓ **A** Summarize or prune stale conversation turns, retain only relevant context, and explicitly restate the current task and priorities in the prompt
Correct. Context drift can be reduced by removing or compressing obsolete information and explicitly reinforcing the current objective so relevant context receives greater emphasis.
https://docs.anthropic.com/en/docs/build-with-claude/prompt-engineering/overview

Question 14

Single choice · Domain: D7 Security and Safety › Handle untrusted external input safely by validating, sanitizing, and separating data from instructions

SINGLE CHOICE

Your Claude application receives untrusted input from external sources. The team is establishing how the application should treat this untrusted input.

Untrusted input would be...

A. Validated and sanitized before being incorporated into prompts, and treated as data the model should not interpret as instructions.

B. Treated identically to trusted input from internal sources, on the grounds that all input the application receives can be handled the same way.

C. Routed through a separate Claude application with more lax security controls so the original application does not handle the untrusted input directly.

D. Blocked entirely so the application accepts only trusted input, with no path for legitimate external input the application might process.

Answer: A. Validated and sanitized before being incorporated into prompts, and treated as data the model should not interpret as instructions.

Explanation

This question tests secure input-handling practices for preventing untrusted content from influencing Claude as instructions. The correct approach treats external content as data and applies validation and sanitization before incorporating it into prompts.

✓ **A** Validated and sanitized before being incorporated into prompts, and treated as data the model should not interpret as instructions.
Correct. Untrusted input should be validated and sanitized, and the application should clearly distinguish external data from trusted instructions to reduce prompt-injection risk.

https://docs.anthropic.com/en/docs/test-and-evaluate/strengthen-guardrails/mitigate-prompt-injections

✗ **B** Treated identically to trusted input from internal sources, on the grounds that all input the application receives can be handled the same way.
Incorrect. External input may contain malicious or instruction-like content and should not automatically receive the same trust level as controlled internal input.

https://docs.anthropic.com/en/docs/test-and-evaluate/strengthen-guardrails/mitigate-prompt-injections

✗ **C** Routed through a separate Claude application with more lax security controls so the original application does not handle the untrusted input directly.
Incorrect. Moving untrusted input to an application with weaker controls increases risk rather than addressing it. Untrusted content still requires validation, sanitization, and safeguards wherever it is processed.

https://docs.anthropic.com/en/docs/test-and-evaluate/strengthen-guardrails/mitigate-prompt-injections

✗ **D** Blocked entirely so the application accepts only trusted input, with no path for legitimate external input the application might process.
Incorrect. Completely blocking external input is not generally required when legitimate external content is part of the application's purpose. The appropriate approach is to accept it with suitable validation, sanitization, and prompt-injection defenses.

https://docs.anthropic.com/en/docs/test-and-evaluate/strengthen-guardrails/mitigate-prompt-injections

Question 15

Single choice · Domain: D7 Security and Safety › Evaluate deployment architecture against data residency and compliance requirements

SINGLE CHOICE

You are deciding between deploying a Claude-powered agent on Anthropic's hosted infrastructure or self-hosting under a "bring your own cloud" model in your own AWS account.

The agent processes customer data subject to your enterprise's data residency policies, but the

A. Deploying on Anthropic-hosted infrastructure to meet the team's shipping timeline, and flagging the data residency requirement for a follow-up compliance review after launch.

B. Before launch, choose and verify a supported deployment/configuration whose inference and data-processing geography satisfies the enterprise's residency requirements—for example, an applicable regional Amazon Bedrock endpoint or Anthropic geography control—and validate the specific residency guarantees against the organization's policy

Answer: B. Before launch, choose and verify a supported deployment/configuration whose inference and data-processing geography satisfies the enterprise's residency requirements—for example, an applicable regional Amazon Bedrock endpoint or Anthropic geography control—and validate the specific residency guarantees against the organization's policy

Explanation

This question tests whether the developer prioritizes mandatory data residency requirements when choosing between Anthropic-hosted and self-hosted deployment. Compliance constraints must be resolved before processing regulated customer data, rather than deferred until after launch.

✗ **A** Deploying on Anthropic-hosted infrastructure to meet the team's shipping timeline, and flagging the data residency requirement for a follow-up compliance review after launch.
Incorrect. Shipping speed does not override an explicitly stated enterprise data residency requirement. Launching first and reviewing compliance later could expose regulated customer data in an unauthorized location; the deployment model and applicable residency controls must be validated before launch.

https://docs.anthropic.com/en/docs/build-with-claude/data-residency

✓ **B** Before launch, choose and verify a supported deployment/configuration whose inference and data-processing geography satisfies the enterprise's residency requirements—for example, an applicable regional Amazon Bedrock endpoint or Anthropic geography control—and validate the specific residency guarantees against the organization's policy
Anthropic distinguishes inference geography and other data-residency controls, and Claude through Amazon Bedrock can use regional endpoints where applicable. Deployment architecture therefore needs to be checked against the actual enterprise residency requirement before customer data is processed.

Free demo · full exam: 83 questions

Question 16

Single choice · Domain: D8 Tools and MCPs › Handle malformed tool-call arguments by validating inputs against the tool schema and returning a controlled error

SINGLE CHOICE

Your Claude application uses tool calling to fetch patient data and generate summary reports.

The flow occasionally fails because the model returns a tool_use block that references arguments not present in the schema, and your application code does not handle this case gracefully.

How would you address this?

A. Validate the tool_use arguments against the declared schema before executing the tool, catch validation failures, and return a structured error or retry request instead of allowing the application to fail.

Answer: A. Validate the tool_use arguments against the declared schema before executing the tool, catch validation failures, and return a structured error or retry request instead of allowing the application to fail.

Explanation

This question tests reliable tool-calling implementation, specifically how to respond when Claude produces tool_use arguments that do not conform to the declared schema. The application should validate arguments before execution and handle validation failures gracefully.

✓ **A** Validate the tool_use arguments against the declared schema before executing the tool, catch validation failures, and return a structured error or retry request instead of allowing the application to fail.
Correct. Tool inputs should be validated before execution, and the application should handle invalid arguments explicitly so malformed requests do not cause an unhandled failure.

https://docs.anthropic.com/en/docs/agents-and-tools/tool-use/implement-tool-use

Free demo · full exam: 83 questions

Question 17

Single choice · Domain: D8 Tools and MCPs › Define custom Claude tools with precise schemas, usage descriptions, and explicit error handling

SINGLE CHOICE

You are implementing a custom tool for your Claude agent. The tool needs to interact with an external pricing service that returns product data.

Which of the following best practices would you apply as you develop this tool?

A. Omit the tool description and let the model infer when to use the tool based on the tool's name and the rest of the prompt context.

B. Define the tool with a loose schema and let the model interpret the inputs flexibly on each call the agent makes.

C. Implement the tool with no error handling and let the agent loop catch failures whenever the pricing service returns an error during operation.

D. Define the tool with a clear schema, write a precise description for when to call it, and handle pricing service errors explicitly.

Answer: D. Define the tool with a clear schema, write a precise description for when to call it, and handle pricing service errors explicitly.

Explanation

This question tests the ability to implement reliable custom tools for Claude, including describing when a tool should be used, constraining inputs with a schema, and handling external-service failures explicitly.

✗ **A** Omit the tool description and let the model infer when to use the tool based on the tool's name and the rest of the prompt context.
Incorrect. A precise tool description helps Claude understand the tool's purpose and when it should be called; relying only on the name and surrounding context makes tool selection less reliable.

https://docs.anthropic.com/en/docs/agents-and-tools/tool-use/overview

✗ **B** Define the tool with a loose schema and let the model interpret the inputs flexibly on each call the agent makes.
Incorrect. A clear input schema constrains arguments and improves reliable tool calls, validation, and integration with the external service.

https://docs.anthropic.com/en/docs/agents-and-tools/tool-use/overview

✗ **C** Implement the tool with no error handling and let the agent loop catch failures whenever the pricing service returns an error during operation.
Incorrect. The custom tool should explicitly handle external-service errors and return useful, structured results so the application or agent can respond appropriately; failures should not be left entirely to an agent loop.

https://docs.anthropic.com/en/docs/agents-and-tools/tool-use/overview

✓ **D** Define the tool with a clear schema, write a precise description for when to call it, and handle pricing service errors explicitly.
Correct. Claude tools should provide a clear description and structured input schema, while the implementation should validate inputs and handle external-service failures explicitly.

https://docs.anthropic.com/en/docs/agents-and-tools/tool-use/overview