Anthropic
CCDV-F
Claude Certified Developer-Foundations

Latest Version: 6.0

Question: 1

You are designing a Claude application that will require structured JSON output for downstream processing. The output schema is well-defined, and downstream systems will reject malformed JSON.

A. Structure the prompt to request output in a schema that is described in plain English, with downstream systems parsing whatever shape Claude produces.
B. Define a clear schema and structure the prompt to request output in that schema, with downstream systems handling any validation needed.
C. Define a clear schema, structure the prompt to request output in that schema, and validate Claude's output against the schema before passing it downstream.
D. Avoid structured output and use free-form text everywhere instead, on the grounds that free-form text is more flexible and handles edge cases better than structured schemas.

Answer: C

Explanation:
Option C establishes the strongest application boundary between probabilistic model generation and deterministic downstream processing. When another component requires JSON with a known contract, the application should explicitly define the expected structure and ensure that model output conforms to it before downstream execution. Anthropic's current Structured Outputs guidance states that structured outputs constrain responses to a specific schema and are intended to provide valid, parseable data for downstream processing. The current Claude API supports JSON Schema through output_config.format, while SDK helpers can additionally parse and validate returned data.
The underlying engineering principle remains the same even when structured-output enforcement is unavailable: never allow unvalidated model-generated structures to become trusted machine input. Option A provides insufficient contractual control. Option B defines the schema but pushes validation too late, increasing the probability that malformed or semantically invalid data reaches dependent components. Option D sacrifices machine reliability entirely.
Therefore, C correctly combines schema specification, explicit format guidance, and validation. This corresponds to Claude Developer topics covering structured outputs, defensive application design, schema validation, and reliable model-to-system interfaces. The question and options are reproduced from the supplied examination set.
==========

Question: 2

Your Claude application has multi-step workflows where each step's output is needed only briefly before the agent moves on. The cumulative tool output is filling the context window with content that is no longer relevant.
How would you handle the accumulating tool output?

A. Apply tool output pruning to remove tool outputs that are no longer needed by later steps in the workflow.
B. Apply prompt caching to the accumulated tool outputs so the application does not re-pay for the older content on each subsequent step.
C. Switch to a smaller Claude model that processes context more efficiently and treat any quality loss as a tradeoff for the cost reduction.
D. Keep every tool output in the context indefinitely so the agent has the full record of every step it has executed during the workflow.

Answer: A

Explanation:
Option A applies the correct context-engineering strategy: remove stale tool results once they no longer contribute useful information to subsequent reasoning. Agentic workflows frequently accumulate search results, file contents, API responses, and intermediate artifacts. Keeping all of them indefinitely consumes the finite context window, raises token cost, and can reduce model focus by introducing low-value information.
Anthropic specifically documents tool result clearing for this situation. Context Editing can remove older tool results when the conversation grows, while preserving recent interactions and optionally retaining tools whose results must remain available. Anthropic describes old tool outputs such as retrieved files or search results as candidates for clearing after Claude has processed them.
Prompt caching in B solves a different problem: it can lower cost and latency for repeated static prompt prefixes, but cached tokens still constitute context and therefore do not eliminate context-window pressure. C changes model capability without solving the architectural cause. D maximizes context pollution.
The correct architecture is therefore to preserve high-signal state while pruning ephemeral intermediate outputs. This aligns with Claude Developer coverage of context engineering, long-running agents, context-window management, tool-result clearing, and efficient agent state management. Anthropic's broader context-engineering guidance likewise emphasizes curating the smallest high-signal context necessary for successful inference.
==========

Question: 3

A teammate has asked you to explain the difference between context engineering and prompt engineering. They have heard the terms used interchangeably and are unsure how each applies to a Claude application that processes long-running multi-step tasks.
How would you describe the distinction?

A. Prompt engineering focuses on the model's response, while context engineering focuses on the user's input across many sessions in a long-running multi-step Claude application.
B. Prompt engineering is the older term for prompt design, while context engineering is the newer term that has replaced it in modern Claude applications across the industry.
C. Prompt engineering shapes individual prompts for specific outputs, while context engineering manages how content flows across turns and steps and takes steps to keep relevant state visible.
D. Prompt engineering and context engineering each address content the team gives Claude, but the team can group them under a single workflow because the practices use overlapping techniques.

Answer: C

Explanation:
Option C accurately captures Anthropic's distinction. Prompt engineering primarily concerns how instructions are written, structured, and organized to obtain the desired behavior from a particular model invocation. Techniques include explicit instructions, examples, roles, XML structure, output requirements, and task-specific prompt construction. Context engineering operates at a broader architectural level: it determines which information should actually be present in the model's context at each inference step.
Anthropic defines prompt engineering as methods for writing and organizing LLM instructions, whereas context engineering encompasses strategies for curating and maintaining the optimal set of tokens during inference. For long-running agents, context can contain system instructions, tools, MCP resources, retrieved documents, prior messages, tool results, summaries, and memory.
This distinction matters because multi-step agents continuously generate new state. Effective systems may prune obsolete results, retrieve information just in time, compact earlier conversation history, isolate subagent contexts, or store persistent state externally. B is incorrect because context engineering has not simply replaced prompt engineering; the two operate at different scopes. A defines context too narrowly, and D obscures an important architectural distinction.
Therefore, C correctly represents Claude Developer coverage of prompt engineering versus context engineering, context curation, agent state, long-horizon workflows, and context-window optimization.
==========

Question: 4

You are designing a multi-step Claude workflow where some steps must reason without seeing the full prior conversation history. The team wants to keep specific context isolated to specific steps.
The context engineering technique you would use is...

A. Augmenting the context with all available content at every step so each step has access to the entire prior history of the workflow during its reasoning.
B. Using a single global prompt that applies to every step in the workflow no matter what each step is reasoning about during its run.
C. Context isolation through subagents or multi-step agentic workflows that scope each step's context to only what the step needs.
D. Embedding the full prior history in each step regardless of whether the step needs the prior history for its reasoning.

Answer: C

Explanation:
Option C is the correct application of context isolation. A specialized step should receive the minimum relevant information required for its own task rather than inheriting an ever-growing global transcript. This improves signal-to-noise ratio, limits accidental cross-task influence, controls token usage, and makes individual components easier to evaluate.
Anthropic's context-engineering guidance explicitly identifies multi-agent architectures as a technique for long-horizon work. Specialized subagents can operate with their own context windows and return condensed results to an orchestrating agent rather than exposing every agent to every intermediate detail. This architecture protects each reasoning process from irrelevant history while allowing the overall system to preserve necessary state.
Options A and D represent the opposite approach: indiscriminately loading the full prior history. Larger context is not automatically better; Anthropic warns that excessive context can introduce context pollution and degrade retrieval or attention to important information. B also fails because a single global prompt does not isolate state or tailor the information available to each processing stage.
Therefore, C best implements scoped reasoning boundaries. Relevant Claude Developer topics are multi-agent architecture, subagents, context isolation, context engineering, orchestration, and long-running workflow design.
==========

Question: 5

Your Claude application produces good responses for typical inputs but struggles with edge cases. You have several labeled examples of edge-case inputs and the desired response for each. You want to use these examples to improve the model's handling of edge cases.
What is the best way to use these examples?

A. Embed the examples in a database for the model to find during inference.
B. Add the labeled edge-case examples to the prompt as few-shot examples so the model can learn the pattern.
C. Train a custom model on the edge-case examples and deploy that custom model in place of the team's current Claude integration.
D. Tell users to avoid submitting the edge-case inputs to the application by adding warnings in the application's user interface.

Answer: B

Explanation:
Option B applies few-shot, or multishot, prompting, one of Anthropic's recommended techniques for steering Claude when examples of desired behavior are available. Labeled input/output pairs give Claude concrete demonstrations of how it should respond, which is particularly valuable when edge cases are difficult to express completely through abstract rules. Anthropic states that examples are among the most reliable mechanisms for steering output format, tone, and structure. Its prompting guidance recommends relevant, diverse examples that cover edge cases while avoiding accidental patterns. For best results, examples should be clearly separated from the main instructions, such as by using <example> and <examples> tags. A retrieval database could be useful if a very large or dynamically selected example collection were required, but that adds unnecessary complexity for the small labeled set described. C is disproportionate: a few examples do not justify replacing the application's Claude integration with custom model training. D avoids rather than solves the identified failure mode.
Therefore, B directly uses the available supervision at inference time and allows rapid iteration through evaluation. Relevant Claude Developer topics are prompt construction, few-shot prompting, edge-case handling, example selection, evaluation-driven iteration, and behavioral steering.
==========

Thank You for Trying Our Product

For More Information – Visit link below:

https://www.examsboost.com/

15 USD Discount Coupon Code:

G74JA8UF

FEATURES

- ✓ 90 Days Free Updates
- ✓ Money Back Pass Guarantee
- ✓ Instant Download or Email Attachment
- ✓ 24/7 Live Chat Support
- ✓ PDF file could be used at any Platform
- ✓ 50,000 Happy Customer