Anthropic
CCA-F
Claude Certified Architect Foundations (CCA-F)

Latest Version: 6.0

Question: 1

When designing an automated code review system for a large repository, you notice that developers are experiencing 'alert fatigue' because the AI flags too many non-issues and stylistic preferences. What is the most effective prompt engineering technique to reduce these false positives?

A. Instruct the model to be thorough and flag everything suspicious to ensure no bugs are missed.
B. Define explicit, measurable review criteria, such as flag functions exceeding 50 lines of code, instead of relying on vague instructions.
C. Set the model's temperature to 0.0 to ensure deterministic output and eliminate all false positives.
D. Provide at least 15 few-shot examples covering every possible coding style in the repository.

Answer: B

Explanation:
Explicit, measurable criteria produce consistent, actionable results that build developer trust and reduce false positives. Vague instructions like 'be thorough' lead to over-flagging and alert fatigue. Temperature adjustments do not fix vague criteria, and providing 15 few-shot examples bloats the prompt without proportional benefit.

Question: 2

You are building a classifier to categorize customer reviews, but the model struggles with ambiguous scenarios like sarcasm and mixed sentiments. You decide to use few-shot prompting. What is the recommended best practice for this scenario?

A. Provide exactly 1 example of a standard positive review to establish the JSON format.
B. Provide 8 to 10 examples to ensure every possible sentiment category is covered comprehensively.
C. Provide 2 to 4 examples, ensuring that at least one example specifically addresses an ambiguous edge case like sarcasm.
D. Avoid few-shot prompting and instead use a detailed system prompt explaining the exact definition of sarcasm.

Answer: C

Explanation:
The optimal number of few-shot examples for ambiguous tasks is 2-4. Providing more than 6 examples bloats the prompt without adding proportional value. It is critical that at least one of these examples covers an edge case or ambiguous scenario to establish the expected reasoning pattern.

Question: 3

A pull request modifies 14 files across a stock tracking module. When analyzing all files in a single pass, Claude provides superficial comments, misses obvious bugs, and produces contradictory feedback. How should you redesign the multi-pass review architecture to resolve this?

A. Switch to a higher-tier model with a larger context window so all 14 files receive adequate attention in one pass.
B. Require developers to manually split large PRs into smaller submissions of 3-4 files before triggering the automated review.
C. Run three independent single-pass reviews on the full PR and only flag issues that achieve consensus across at least two runs.
D. Split the review into focused passes: analyze each file individually for local issues, followed by a separate integration-focused pass for cross-file data flow.

Answer: D

Explanation:
Splitting reviews into focused passes directly addresses attention dilution when processing many files at once. Per-file analysis ensures consistent depth for local issues, while a separate integration pass catches cross-file architectural issues.

Question: 4

In a Continuous Integration (CI) pipeline, you run a script where a single Claude session generates a block of code and then immediately reviews its own changes in the same session. Why is this considered an architectural anti-pattern?

A. The reviewer retains the reasoning context from the code generation phase, creating confirmation bias and reducing the likelihood of catching subtle issues.
B. It exceeds the maximum token limit for the context window, causing the session to silently truncate the generated code.
C. Same-session reviews trigger rate limiting automatically, breaking the CI/CD pipeline.
D. The Message Batches API does not support multi-turn tool calling within a single request.

Answer: A

Explanation:
Same-session self-review is an anti-pattern because the model retains its original reasoning context, creating a blind spot. Independent review instances without prior reasoning context are much more effective at objectively evaluating code.

Question: 5

While configuring prompt engineering standards for a production application, you must decide when to implement few-shot examples. Which of the following scenarios represent the most effective use cases for few-shot prompting? Choose 2 correct answers.

A. Ambiguous classification tasks where the boundaries between categories are fuzzy or subjective.
B. Simple, well-defined data extraction tasks with clear objective criteria.
C. Tasks requiring standard output formats like valid JSON, where tool_use is already implemented.
D. Tasks requiring a custom, non-standard output format that Claude does not handle natively.

Answer: A,D

Explanation:
Few-shot prompting is most valuable when the task has ambiguous boundaries (like detecting sarcasm) or requires custom output formats that aren't standard. It is generally unnecessary for simple, well-defined tasks or standard formats like JSON which are better handled natively or via tool_use.

Question: 6

During a long-running customer support session, the agent repeatedly summarizes the conversation history to save context space. By turn 10, the agent forgets the customer's specific account number and the exact promotional price they were quoted. How should you redesign the context management to prevent this?

A. Increase the frequency of progressive summarization to occur every 2 turns instead of 5.
B. Store the conversation in an external database and run a lexical search query on every turn.
C. Extract transactional facts into an immutable 'case facts' block positioned at the start of the context that is never summarized.
D. Restart the session from scratch after 5 turns and ask the customer to repeat their details.

Answer: C

Explanation:
Progressive summarization silently destroys critical details over time. The solution is to use 'case facts' blocks-immutable, structured reference sections placed at the high-recall beginning of the context window that preserve critical information without being compressed.

Question: 7

A developer is using a multi-agent system to analyze a massive legacy codebase. The coordinator agent is running out of context window space because the search tool returns verbose file contents and exploration noise. What is the best pattern to mitigate this context degradation?

A. Share the full coordinator conversation history with all subagents to ensure maximum context propagation.
B. Delegate the verbose file-by-file exploration to a subagent, which then returns only a synthesized summary of its findings back to the coordinator's context.
C. Force the coordinator agent to use the Bash tool to run grep manually instead of reading files.
D. Switch from dynamic adaptive decomposition to static prompt chaining to force a linear review path.

Answer: B

Explanation:
Subagent delegation is a core mitigation for context degradation. By delegating verbose exploration tasks to subagents, the subagent's isolated context absorbs the exploration noise, and only the clean, synthesized results are returned to the coordinator.

Question: 8

When building an agent that queries a backend order system, the lookup_order tool returns a JSON object with 40+ fields, but the age only needs the order status, shipping date, and tracking number. What context optimization technique should be applied?

A. Implement a PostToolUse hook to trim the verbose tool outputs, returning only the relevant fields to the agent's context.
B. Prompt the model to ignore the irrelevant fields after the full JSON object is appended to the conversation history.
C. Utilize the /compact command immediately after every tool call.
D. Use a PreToolUse hook to block the tool call entirely and escalate to a human.

Answer: A

Explanation:
Tool results accumulate in context and consume tokens disproportionately. Trimming verbose tool outputs to only relevant fields before they accumulate in the context window prevents context exhaustion and improves reasoning focus.

Question: 9

In a long-running codebase debugging session, the agent makes important intermediate discoveries but starts forgetting earlier constraints as the context window fills. How can you ensure these intermediate discoveries survive context compression boundaries?

A. Declare global variables in the system prompt to hold state.
B. Use the - -resume flag to append the new context to the old session.
C. Persist critical intermediate state to extemal scratchpad files, which survive session boundaries and context resets.
D. Apply the 'lost in the middle' effect by placing the discoveries in the exact center of the prompt.

Answer: C

Explanation:

Scratchpad files are used to persist important intermediate state to external files. Because they are external, they survive context compression (like the /compact command) and allow the agent to re-read its progress when needed.

Question: 10

You are managing context in a long-running Claude Code session that has accumulated a large amount of exploratory tool calls. You want to free up context space without completely losing the memory of what was previously worked on. Which techniques are recommended? Choose 2 correct answers.

A. Use the /compact command to summarize conversation history and reclaim context space.
B. Use the /clear command to erase all history and rely on the model's pre-trained knowledge.
C. Save key findings to a scratchpad file and delegate verbose exploration to subagents.
D. Delete older messages from the conversation array manually to ensure hard truncation.

Answer: A,C

Explanation:
The recommended mitigation strategies for context degradation in extended sessions include using the Icompact command to compress history, persisting key state to scratchpad files, and using subagent delegation to keep the coordinator context clean. [clear removes all memory, which contradicts the goal of retaining previous work memory.

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