Anthropic.CCAR-F.v2026-09-27.q53

Exam Code:	CCAR-F
Exam Name:	Claude Certified Architect - Foundations
Certification Provider:	Anthropic
Free Question Number:	53
Version:	v2026-09-27
# of views:	185
# of Questions views:	530
https://www.exam-tests.com/CCAR-F-exam/Anthropic.CCAR-F.v2026-09-27.q53.html	

NEW QUESTION: 1
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 is configuring MCP servers in Claude Code. You want to add a shared venue lookup server that all team members should have access to, and you personally want to add an experimental music playlist server that only you are testing.
Which configuration approach correctly applies MCP server scopes?
A. Add both servers to your local ~/.claude.json .
B. Add the venue server to .mcp.json and the playlist server to ~/.claude.json .
C. Add the venue server to ~/.claude.json and the playlist server to .mcp.json .
D. Add both servers to the project-level .mcp.json file.
Answer: B (LEAVE A REPLY)
The venue server must be available to every project contributor, so it belongs at project scope. Claude Code stores project-scoped MCP configurations in the repository-root .mcp.json file, which can be committed to version control and shared with the team. The experimental playlist server should remain private. Local-scoped MCP servers are stored under the current project's entry in ~/.claude.json , while user-scoped servers are also stored in ~/.claude.json but load across all projects. Because this server is described as a personal experiment, either private scope is conceptually appropriate; among the listed choices, option B correctly separates shared project configuration from private personal configuration. (https://code.claude.com/docs/en/mcp) Option A prevents the team from receiving the venue configuration. Option C reverses the intended visibility. Option D exposes the experimental server configuration to other contributors and may commit machine- specific commands or credentials.

The preferred commands would specify --scope project for the shared venue server and --scope local for an experiment limited to the current repository. Project-scoped servers also require user approval before Claude Code activates configurations received through .mcp.json .
Official references/topics: MCP Installation Scopes; Project Scope; Local Scope; .mcp.json ; ~/.claude.json .

NEW QUESTION: 2
After deploying the automated review, you notice high precision but low recall-real bugs are slipping through undetected. Investigation reveals that your review prompt instructs Claude to "only report high- confidence issues you are certain about" and "err on the side of not commenting." Developers appreciate the low noise, but a race condition that caused a production outage was visible in a reviewed pull request and went unreported. You need to substantially improve bug detection while keeping false-positive rates manageable.
What is the most effective approach?
A. Add detailed few-shot examples demonstrating bug categories Claude should flag-race conditions, null dereferences, and error-handling gaps-while retaining the high-confidence filtering instruction.
B. Remove the conservative instructions and have Claude report every potential issue, then apply a programmatic filter that deduplicates findings and suppresses historically noisy categories.
C. Split the review into a finding stage whose objective is comprehensive coverage-reporting every potential issue with confidence and severity metadata-and a separate stage that verifies and thresholds those findings.
D. Expand the context to include related tests, recent Git history, and the module's dependency graph so Claude has richer evidence for judging severity.
Answer: C (LEAVE A REPLY)
The prompt's conservative reporting policy is directly causing the low recall. Claude may discover a legitimate race condition during analysis but suppress it because it cannot satisfy the instruction to report only issues about which it is certain. Option C separates two objectives that should not be conflated: broad defect discovery and strict acceptance filtering.
Anthropic's current code-review prompting guidance explicitly recommends reporting every issue, including uncertain or lower-severity findings, assigning confidence and severity metadata, and allowing a separate verification stage to filter them. This maximizes recall while retaining control over developer-facing noise.
Option A leaves the suppression policy intact, so additional examples cannot guarantee that discovered problems will be reported. Option B improves recall but relies on historical category-level suppression, which can discard genuine findings that happen to belong to noisy categories. Option D can improve analysis quality but does not remove the instruction responsible for withholding findings. A dedicated finding stage followed by an

independently configurable verification or thresholding stage produces measurable recall and precision controls without forcing one model call to optimize competing objectives.

NEW QUESTION: 3
You are building a structured data extraction system using Claude. The system extracts information from unstructured documents, validates the output using JSON schemas, and maintains high accuracy. It must handle edge cases gracefully and integrate with downstream systems.
Your invoice extraction uses tool use with strict JSON schemas. JSON syntax errors never occur, but 12% of extractions fail semantic validation-for example, line-item amounts do not sum to the extracted total, or vendor IDs do not match valid formats. These failures currently route to manual review.
What is the most effective approach to reduce manual-review volume while maintaining accuracy?
A. Implement post-processing logic that automatically corrects common errors, such as recalculating totals from line items when sums do not match.
B. When validation fails, make a follow-up request containing the document, extraction, and validation errors so the model can correct the result.
C. Retry the extraction up to three times when validation fails, accepting the first result that passes validation.
D. Add stricter schema constraints with detailed field descriptions to prevent the model from initially generating invalid values.
Answer: B (LEAVE A REPLY)
Option B creates a targeted correction loop using information the first extraction did not have: the validator's precise failure report. Anthropic's prompting guidance describes prompt chaining as appropriate when an application must inspect intermediate output or enforce a specific pipeline, with self-correction following the pattern generate, review against criteria, and refine. Here, deterministic validators identify arithmetic mismatches and invalid vendor-ID formats. Returning the source document, original extraction, and explicit errors lets Claude revise only the defective fields while preserving valid data. The corrected record must then be validated again before acceptance, with a bounded retry count and human-review fallback. Option A is unsafe because automatically recalculating a total may overwrite the amount actually printed on the invoice; a mismatch can originate in an OCR or line-item extraction error. Option C repeats the same extraction without explaining what failed, wasting calls and relying on chance. Option D may prevent some format violations, but JSON Schema cannot express every cross-field arithmetic relationship or external vendor-registry rule. A validator-guided second pass therefore reduces manual review without silently changing source facts, while keeping deterministic code-not the model-as the final acceptance authority.

NEW QUESTION: 4

You are building developer productivity tools using the Claude Agent SDK. The agent helps engineers explore unfamiliar codebases, understand legacy systems, generate boilerplate code, and automate repetitive tasks. It uses the built-in tools (Read, Write, Bash, Grep, Glob) and integrates with Model Context Protocol (MCP) servers.
An engineer asks the agent to find all callers of a function before removing it. The function is defined in a core library but is also exposed through wrapper modules that rename the function for domain-specific use (e.
g., calculateTax in the library becomes computeOrderTax in the orders module).
What exploration strategy will most reliably identify all callers?
A. Use Grep to find all files that import from the library or wrapper modules, then read each file to check whether it uses the function.
B. Use Grep to search for the function's original name across the codebase.
C. Read the library and wrapper modules to identify all exposed names for the function, then Grep for each name across the codebase.
D. Search for the function name in project documentation to understand intended usage patterns and navigate to documented integration points.
Answer: C (LEAVE A REPLY)
The search must first establish the function's complete naming surface. Reading the core definition and every wrapper or re-export reveals aliases such as computeOrderTax , names introduced through intermediate modules, and potentially different import paths. Once those names are known, the agent can search the entire codebase for each exposed identifier and inspect the resulting call sites.
Claude Code's Grep tool searches patterns inside file contents, whereas Read provides the surrounding implementation needed to determine whether a wrapper delegates to, renames, or conditionally invokes the original function. These tools are complementary: Read establishes semantic identity; Grep provides broad reference discovery.
(https://code.claude.com/docs/en/tools-reference?utm_source=chatgpt.com) Option B misses every caller using a wrapper alias. Option A may find importing files, but import-level discovery is indirect and can miss re-exports, namespace imports, dependency injection, or calls made through a locally renamed symbol. It also requires unnecessary manual inspection of every consumer of the module. Option D relies on documentation, which may be incomplete or stale and cannot prove that all executable references have been found.
Before removal, the agent should also inspect tests, dynamic registrations, configuration-driven references, and generated code where relevant. The defining requirement is to map the alias chain first and then search all discovered public names.
Official references/topics: Read and Grep Tool Behavior; Symbol and Alias Discovery; Codebase Reference Tracing.

NEW QUESTION: 5

You are building developer-productivity tools using the Claude Agent SDK. The agent helps engineers explore unfamiliar codebases, understand legacy systems, generate boilerplate code, and automate repetitive tasks. It uses the built-in tools-Read, Write, Bash, Grep, and Glob-and integrates with Model Context Protocol (MCP) servers.
An engineer asks the agent to understand how the caching layer works before adding a new cache-invalidation trigger. Initial Grep searches show that caching logic spans 15 files containing decorators, middleware, and service classes-approximately 8,000 lines in total.
What is the most effective next step for building understanding while managing context constraints?
A. Analyze imports and class hierarchies to identify the base cache class, read that file to understand its interface, and then trace the specific invalidation implementations.
B. Use Glob to find files matching common caching patterns such as *cache*.py or caching/ , read the largest files first, and inspect smaller files afterward.
C. Use Read to load all 15 files sequentially and build a complete understanding of the caching implementation.
D. Use Grep to search for invalidate and expire , and then read only the matching line ranges with minimal surrounding context.
Answer: A (LEAVE A REPLY)
Option A establishes the caching architecture before consuming thousands of implementation lines. The base class or protocol reveals the supported operations, lifecycle, cache-key rules, and invalidation contract.
Imports, subclasses, registrations, and call sites can then direct the agent toward only those concrete implementations relevant to the proposed trigger. Anthropic's effective context-engineering guidance explains that context is finite and that recall degrades as irrelevant tokens accumulate. It identifies Glob and Grep as mechanisms for retrieving code just in time rather than loading an entire repository upfront. Option B assumes file size or naming determines architectural importance; a small interface may define the system while a large service file contains incidental caching calls. Option C loads approximately 8,000 lines before establishing which portions matter. Option D is overly lexical and can miss invalidation implemented through event publication, key-version changes, inherited methods, or generically named hooks. Dependency-guided exploration provides a compact conceptual model first and expands the investigation only when concrete relationships show that additional files are relevant.

NEW QUESTION: 6
You are building a multi-agent research system using the Claude Agent SDK. A coordinator agent delegates to specialized subagents: one searches the web, one analyzes documents, one synthesizes findings, and one generates reports. The system researches topics and produces comprehensive, cited reports.
Production monitoring shows that the research phase takes longer than expected. Analysis reveals that the coordinator invokes the web-search subagent, waits for its response, and

then invokes the document-analysis subagent. These tasks are independent; neither requires the other's output.
How should you modify the system to run these subagents concurrently?
A. Switch both subagents to a Haiku-tier model instead of Sonnet to reduce their individual execution times.
B. Structure the coordinator to emit both Agent tool calls-formerly Task tool calls-in a single response rather than across separate conversation turns.
C. Add instructions explaining the performance benefits of parallel execution and requesting that the coordinator invoke both subagents simultaneously.
D. Create an asynchronous orchestration layer that starts separate coordinator-subagent pairs in parallel and aggregates their results.
Answer: B (LEAVE A REPLY)
Option B expresses both independent delegations in the same orchestration turn, allowing the runtime to execute them concurrently and return their results together. The current Claude Agent SDK calls the subagent- spawning capability the Agent tool; Task is its former name. Anthropic's SDK subagent documentation explicitly identifies parallel analysis as a primary subagent use case. Claude tool responses can also contain multiple tool_use blocks , enabling independent calls to be handled as one parallel group rather than as serial model turns. Option A shortens each execution but does not remove the unnecessary wait between them and may reduce analysis quality. Option C provides useful behavioral guidance, but instructions alone do not establish the required response structure; the coordinator must actually emit both calls together. Option D duplicates coordinator execution, complicates state management, and creates unnecessary aggregation work.
The correct flow is parallel fan-out from one coordinator, followed by a synchronization point that validates both results before synthesis begins. Failures should be tracked independently so that only the unsuccessful branch requires retrying.

NEW QUESTION: 7
You are building developer productivity tools using the Claude Agent SDK. The agent helps engineers explore unfamiliar codebases, understand legacy systems, generate boilerplate code, and automate repetitive tasks. It uses the built-in tools (Read, Write, Bash, Grep, Glob) and integrates with Model Context Protocol (MCP) servers.
An engineer asks your agent to identify untested code paths in a legacy payment processing module spanning
45 files. After reading the first 8 source files, the agent's responses are becoming noticeably less accurate-it' s forgetting previously discussed code patterns and hasn't yet located all test files or traced critical payment flows.
What's the most effective approach to complete this investigation?
A. Spawn subagents to investigate specific questions (e.g., "find all test files for payment processing,"

"trace refund flow dependencies") while the main agent coordinates findings and preserves high-level understanding.
B. Clear context with /clear , then selectively re-read only the most critical files discovered so far, writing key findings to a scratchpad file that persists between context resets.
C. Switch to using Grep to search for specific function names instead of reading full files, reducing the content loaded into context for remaining exploration.
D. Document all current findings in a summary report, clear context completely, then use that report as the sole reference for continuing the investigation.
Answer: A (LEAVE A REPLY)
The investigation contains several bounded research questions that can be delegated independently: locating the complete test suite, tracing payment and refund flows, identifying conditional branches, and mapping external dependencies. Each subagent can read the relevant files in its own context and return a focused summary to the coordinating agent.
Anthropic recommends subagents for codebase exploration because extensive file reading rapidly consumes the main context window. Subagents isolate that volume and return only their conclusions, preserving the main conversation for synthesis and implementation.
(https://docs.anthropic.com/en/docs/claude-code
/common-workflows) Anthropic also describes parallel research as appropriate when separate investigation paths can proceed independently and the main agent can synthesize the results afterward. (https://docs.
anthropic.com/en/docs/claude-code/sub-agents)
Option B sacrifices the current conversational state and requires reconstruction after /clear . Option C may reduce token usage, but isolated text matches cannot reliably reveal full execution paths, indirect calls, or test coverage relationships. Option D converts the current analysis into a single lossy summary and risks omitting details needed later.
Option A directly addresses the demonstrated context degradation while retaining a high-level coordinating thread. The subagent prompts should be narrowly scoped and require concrete outputs such as file paths, uncovered branches, call-chain evidence, and existing tests associated with each flow.
Official references/topics: Subagent Context Isolation; Parallel Research; Context Preservation; Coordinated Codebase Analysis.

NEW QUESTION: 8
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's CLAUDE.md includes a rule: "Use 4-space indentation and always run Prettier formatting." Despite this, code reviews reveal that roughly 30% of files Claude Code generates use inconsistent formatting-sometimes 2-space indentation, sometimes missing

trailing commas. Adding emphasis ("IMPORTANT: You MUST use Prettier formatting") reduces violations to about 15%, but doesn't eliminate them.
What is the most effective way to ensure all generated code is consistently formatted?
A. Extract the formatting rules into a dedicated skill that Claude loads automatically when generating code, with more detailed examples of correct formatting.
B. Add a Stop hook with a prompt-based check that evaluates whether generated code follows formatting standards and prompts Claude to fix violations.
C. Split the formatting rules into path-scoped .claude/rules/ files that load when Claude works on matching file types.
D. Configure a PostToolUse hook with an Edit|Write matcher that automatically runs Prettier on each file Claude modifies.
Answer: D (LEAVE A REPLY)
Formatting is deterministic and should be enforced by deterministic tooling rather than stronger natural- language instructions. A PostToolUse hook executes after a successful tool operation. Matching Edit|Write restricts the hook to file modifications, allowing the edited file path to be passed directly to Prettier.
Anthropic provides this exact pattern in its official hooks guidance: configure a PostToolUse event with an Edit|Write matcher and run Prettier against the modified file. This ensures formatting occurs automatically after every relevant edit instead of depending on Claude remembering to execute a formatter. (https://code.claude.com/docs/en/hooks-guide)
Option A still relies on instruction-following and unnecessarily loads procedural material. Option B asks another probabilistic model evaluation to determine whether formatting is correct, even though Prettier can enforce the result directly. A Stop hook also runs later than necessary. Option C improves contextual relevance but does not guarantee compliance; path-scoped rules remain instructions rather than enforcement controls.
The hook should be stored in project settings so the workflow is shared by the team. The formatter's exit status should also be monitored so configuration or syntax failures are visible instead of silently ignored.
Official references/topics: Claude Code Hooks; PostToolUse; Edit|Write Matchers; Deterministic Formatting Enforcement.

NEW QUESTION: 9
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 review generates many findings per pull request, but developer feedback shows that approximately half are dismissed as "not worth addressing." Analysis reveals

that these findings are often technically accurate but involve minor style preferences or patterns that are acceptable in the project.
Before adding infrastructure complexity, what prompt-design change would most effectively reduce dismissals while maintaining detection of genuine issues?
A. Add a secondary classification model that filters findings according to predicted developer acceptance.
B. Ask Claude to rate each finding's confidence from 1 to 10 and include only findings rated 8 or higher.
C. Define explicit reporting criteria that distinguish reportable bugs and security issues from minor style preferences and accepted local patterns.
D. Add the instruction: "Only report findings you are highly confident are genuine problems."
Answer: C (LEAVE A REPLY)
Option C defines the reviewer's actual objective instead of relying on an undefined notion of confidence. The prompt should identify reportable categories such as correctness defects, security vulnerabilities, data-loss risks, broken requirements, and meaningful performance regressions. It should also specify that cosmetic preferences, lint-enforced formatting, speculative refactoring, and intentional repository conventions must not be reported unless they cause concrete incorrect behavior.
Anthropic's Claude Code best-practices guidance recommends telling a reviewer to report gaps affecting correctness or stated requirements and to treat other observations as optional. Without such criteria, a reviewer asked to "find issues" will predictably return technically defensible but operationally low-value observations. Option A adds cost and another probabilistic decision layer before correcting the primary prompt. Option B relies on self-reported confidence, which does not necessarily align with developer value or severity. Option D can suppress genuine defects that require inference or cross-file verification because
"highly confident" is undefined. Explicit scope, evidence requirements, and examples of both reportable and excluded findings directly calibrate the review toward what developers will act upon. Claude Code best practices

NEW QUESTION: 10
You are building a structured data extraction system using Claude. The system extracts information from unstructured documents, validates the output using JSON schemas, and maintains high accuracy. It must handle edge cases gracefully and integrate with downstream systems.
Your extraction uses tool use with a JSON schema in which property_type is defined as an enum: house, apartment, condo, or townhouse. After deployment, 8% of extractions fail schema validation. Investigation reveals that listings mention many uncommon property types-"studio," "loft," "duplex," "mobile home,"
"tiny house," and "converted warehouse"-and new types continue appearing regularly.

What is the most effective long-term solution?
A. Change property_type from an enum to a free-form string and implement a normalization step in post- processing.
B. Add few-shot examples demonstrating how to map unexpected property types to the closest existing enum value.
C. Continuously expand the enum to include newly observed property types and add monitoring for additional edge cases.
D. Add an other value to the enum with a separate property_type_detail string field for specifics when other is selected.
Answer: A (LEAVE A REPLY)
Option A is the most sustainable open-world design. The property-type domain is demonstrably not closed:
uncommon values already occur, and new labels continue to appear. An enum is appropriate only when the valid vocabulary is stable and exhaustive. If the source contains a legitimate value outside that vocabulary, the schema either rejects it or pressures the model to map it inaccurately. A free-form string preserves the source meaning, while deterministic post-processing can normalize "studio," "loft," "duplex," and future terms into an evolving canonical taxonomy. Anthropic's Structured Outputs documentation distinguishes structural schema conformance from the semantic decisions made within that schema; it also documents that JSON Schema constraints such as enums govern allowable output shapes and values. Few-shot mapping to the nearest existing enum, option B, discards real distinctions. Continually expanding the enum, option C, creates recurring schema changes and still lags newly observed values. An "other" bucket with detail, option D, is workable but collapses all emerging categories at the primary field and complicates analytics until downstream interpretation occurs. Capturing the raw type first and normalizing it in code provides lossless extraction, testable mappings, and forward compatibility.

NEW QUESTION: 11
The coordinator agent has AgentDefinition objects configured for all four specialized subagents, each with appropriate descriptions, prompts, and tool restrictions. During testing, you notice that the coordinator correctly reasons about when to delegate-it generates messages such as, "I'll ask the web-search agent to find sources on this topic"- but no subagent execution occurs. The coordinator then proceeds as if the delegation happened and continues with incomplete information. Logs show no errors. What is the most likely cause?
A. The AgentDefinition objects are configured correctly, but the coordinator's system prompt does not explicitly list the available subagent types.
B. The coordinator's allowedTools configuration does not include " Agent " -called " Task " in older SDK releases-so it cannot invoke the tool required to spawn subagents.

C. Subagent context isolation prevents task descriptions from reaching subagents unless explicit context forwarding is configured in ClaudeAgentOptions.
D. The coordinator's max_tokens setting is too low, causing the subagent invocation to be truncated before the agent-type parameter is specified.
Answer: B (LEAVE A REPLY)
Option B identifies the missing executable capability. Defining specialized agents makes their configurations available, but the coordinator must still be permitted to call the tool that invokes them. Without that tool, Claude can describe an intended delegation in ordinary text but cannot create a subagent execution.
The current Claude Agent SDK documentation requires " Agent " in allowedTools to auto-approve subagent invocations. The tool was renamed from " Task " to " Agent " in Claude Code 2.1.63, so the terminology in the original candidate question required correction. Older integrations may still expose " Task " in initialization or permission records.
Option A is unlikely because properly written AgentDefinition.description values already tell Claude when each agent should be used. Option C misinterprets context isolation: the parent supplies the subagent's assignment through the Agent tool's prompt, and no additional automatic forwarding setting is required.
Option D would normally produce truncation evidence or a max_tokens stop reason rather than silent absence of every invocation. The coordinator needs both agent definitions and permission to use the invocation tool.

NEW QUESTION: 12
You are building a multi-agent research system using the Claude Agent SDK. A coordinator agent delegates to specialized subagents: one searches the web, one analyzes documents, one synthesizes findings, and one generates reports. The system researches topics and produces comprehensive, cited reports.
The document-analysis agent has a single analyze_document tool that takes a document and a free-text instruction parameter. During evaluation, requests such as "extract the key financial metrics" often return narrative summaries, while "summarize the methodology" sometimes returns raw data tables. The synthesis agent reports that 35% of analysis results require new requests with clarified instructions.
What is the most effective way to improve reliability?
A. Enhance the tool description with detailed examples showing how different instruction phrasings should map to different output formats.
B. Split the generic tool into purpose-specific tools-extract_data_points, summarize_content, and verify_claim_against_source-each with defined input and output contracts.
C. Keep the single tool but add an analysis_type enum parameter requiring explicit selection between extraction, summarization, and verification modes.
D. Have the coordinator pre-classify each analysis request before passing instructions to the document- analysis agent.

Answer: B (LEAVE A REPLY)
Option B removes ambiguity at the tool-interface level. Extraction, summarization, and claim verification are different operations with different success criteria and output structures. Separate tools allow each operation to have a precise name, purpose, parameter schema, response contract, and error behavior. Anthropic's guidance on writing effective tools for agents recommends a small set of thoughtful tools targeted at specific, high-impact workflows rather than generic wrappers that force the model to infer operational meaning from loosely structured instructions. Tool descriptions and examples, option A, could improve behavior but leave the overloaded free-text contract intact. An analysis_type enum, option C, identifies the requested mode but still requires one tool to return substantially different result structures, increasing validation and downstream branching. Coordinator pre-classification, option D, adds another probabilistic decision without correcting the interface used by the analysis agent. Purpose-specific tools let the synthesis layer know exactly what output it will receive. Their schemas should require relevant fields-for example, data-point names and source locations for extraction, or claim, verdict, and evidence for verification-and should be tested against the observed failure cases.

NEW QUESTION: 13
Your CI pipeline performs security-focused code reviews on approximately 50 pull requests daily, currently costing $150 per day through the synchronous API. Reviews are non-blocking-developers merge after tests pass and address findings in follow-up commits. You are evaluating the Message Batches API because it offers a 50% cost reduction. What factor most determines whether batch processing is appropriate for this use case?
A. Whether your result-processing system can handle reviews arriving in a different order from the order in which they were submitted.
B. Whether each review can be structured as a single request without multi-turn refinement.
C. Whether review feedback arriving up to 24 hours after pull-request creation remains actionable.
D. Whether reducing per-review latency from 30-60 seconds to near-instantaneous delivery matters to your workflow.
Answer: (SHOW ANSWER)
The decisive trade-off is whether the workflow can tolerate asynchronous completion. Anthropic's Message Batches API documentation states that batch requests receive a 50% discount and that most batches finish within one hour, but processing can continue for as long as 24 hours. Therefore, the review process is suitable for batching only if potentially delayed feedback remains useful.
Option C directly tests that operational requirement. The reviews are already non-blocking, which makes batching promising, but non-blocking does not automatically mean that feedback delivered many hours later still has value. If developers have already merged

and moved to unrelated work, delayed security findings may increase remediation cost or remain unaddressed.
Option A is manageable through each request's unique custom_id, which allows results to be correlated regardless of return order. Option B is incorrect because batch requests can contain supported multi-turn conversation histories and tool-use configurations; each request is processed independently, but it need not represent a single conversational turn. Option D reverses the relevant consideration: batch processing is not intended to provide near-instantaneous responses. The real question is whether the cost saving justifies its longer and nondeterministic completion time.

NEW QUESTION: 14
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 CI pipeline performs security-focused code reviews on approximately 50 pull requests daily, currently costing $150 per day using the synchronous API. Reviews are non-blocking-developers merge after tests pass and address findings in follow-up commits. You are evaluating the Message Batches API for its 50% cost reduction.
What factor most determines whether batch processing is appropriate for this use case?
A. Whether reducing per-review latency from 30-60 seconds to near-instant matters for your workflow.
B. Whether your result-processing system can handle reviews arriving in a different order from the order in which they were submitted.
C. Whether review feedback arriving up to 24 hours after pull-request creation remains actionable.
D. Whether you can structure each review as a single request without multi-turn refinement.
Answer: C (LEAVE A REPLY)
Option C identifies the fundamental trade-off introduced by batch processing: lower cost in exchange for asynchronous completion and potentially substantial latency. Anthropic states that most Message Batches complete within one hour, but results may become available only when all requests finish or after 24 hours, whichever occurs first. Therefore, the workflow must remain useful even if security findings arrive considerably later than they would through synchronous requests.
The reviews are explicitly non-blocking, so batch processing can be suitable if developers can still act on delayed findings through follow-up commits. Option B describes an implementation requirement because batch results may be returned out of submission order, but Anthropic provides a deterministic solution through each request's unique custom_id. Option A incorrectly focuses on near-instant feedback, which batch processing

does not provide. Option D is not decisive because batches support independent Messages API requests containing system prompts, tool use, and multi-turn conversation content. The primary decision is therefore whether the maximum practical feedback delay is acceptable to the development workflow.
Anthropic Message Batches documentation

NEW QUESTION: 15
You are building developer-productivity tools using the Claude Agent SDK. The agent helps engineers explore unfamiliar codebases, understand legacy systems, generate boilerplate code, and automate repetitive tasks. It uses the built-in tools-Read, Write, Bash, Grep, and Glob-and integrates with Model Context Protocol (MCP) servers.
After adding an MCP server with specialized code-refactoring tools-extract_function, rename_variable, and inline_function-you notice that the agent still uses basic text manipulation through Write and Bash sed commands for refactoring tasks. The MCP server is connected and healthy. Examining the configuration, you find that each MCP tool has a minimal description such as, "extract_function: Extracts a function from code." What is the most effective way to improve adoption of the MCP refactoring tools?
A. Implement a request classifier that detects refactoring intent and automatically routes those requests to the MCP server before the agent processes them.
B. Accept this as expected behavior because simpler tools such as sed are more predictable than specialized refactoring tools.
C. Enhance the MCP tool descriptions to explain when each tool is preferable to text manipulation and clarify expected inputs and outputs.
D. Remove the Write tool from the agent's configuration for refactoring sessions so it must use the MCP tools for code modifications.
Answer: C (LEAVE A REPLY)
Option C corrects the weak selection signal presented to the model. Claude chooses among available tools using their names, descriptions, parameter schemas, and the current request. "Extracts a function from code" does not explain whether the tool understands syntax trees, preserves imports, updates call sites, validates scope, or offers advantages over Write and sed. Anthropic identifies prompt-engineering tool descriptions as one of the most effective ways to improve agent tool use. Descriptions should state what the operation performs, when it should be selected, what inputs are required, what output it returns, and any limitations.
Option A adds a separate probabilistic routing layer without improving the tool contract Claude ultimately sees. Option B ignores the server's intended value. Option D removes a broadly useful capability and may prevent unrelated edits without guaranteeing that the MCP tools are used correctly. Each refactoring tool should instead describe its semantic behavior and contrast it with plain text manipulation-for example, that rename_variable performs scope-aware symbol renaming and updates references. Clear schemas, concrete

examples, and evaluation against real refactoring tasks should accompany the improved descriptions.

NEW QUESTION: 16
You are building developer-productivity tools using the Claude Agent SDK. The agent helps engineers explore unfamiliar codebases, understand legacy systems, generate boilerplate code, and automate repetitive tasks. It uses the built-in tools-Read, Write, Bash, Grep, and Glob-and integrates with Model Context Protocol (MCP) servers.
Engineers frequently ask the agent to cross-reference code changes with Jira tickets during reviews- checking ticket descriptions, acceptance criteria, and recent comments. This currently requires manually copying and pasting content into conversations. The team wants the agent to access this standard Jira ticket data directly.
What is the most effective approach?
A. Use the Bash tool with curl to call Jira's REST API, including authentication headers and parsing JSON responses inline.
B. Build a custom MCP server wrapping Jira's API with tools designed specifically for this team's code- review workflow.
C. Export Jira tickets to Markdown files in the repository that the agent accesses using the Read tool.
D. Integrate an existing Jira MCP server that exposes tickets, comments, and metadata through discoverable tool interfaces.
Answer: D (LEAVE A REPLY)
Option D uses the established integration mechanism without creating unnecessary infrastructure. Anthropic's Claude Code MCP documentation specifically recommends connecting an MCP server when users repeatedly copy information from an external system, such as an issue tracker, into conversations. Once connected, the server exposes Jira operations through named, schema-defined tools that Claude can discover and invoke directly. This allows the agent to retrieve ticket descriptions, acceptance criteria, comments, and metadata while preserving the server's authentication and access controls. Option A exposes authentication details to shell commands and requires the agent to construct requests and parse responses repeatedly. Option B is justified only when no existing server provides the required capabilities or the workflow requires highly specialized operations. The question requires standard Jira information, so building and maintaining another server is unnecessary. Option C creates stale, manually synchronized copies that lose permissions, current comments, and authoritative ticket state. The team should therefore select a trusted Jira MCP implementation, configure appropriate read-only scopes where possible, and verify its tool descriptions and data-access boundaries before enabling it.

Valid CCAR-F Dumps shared by BraindumpsPass.com for Helping Passing CCAR-F Exam! BraindumpsPass.com now offer the **newest CCAR-F exam dumps**, the BraindumpsPass.com CCAR-F exam **questions have been updated** and **answers have been corrected** get the **newest** BraindumpsPass.com CCAR-F dumps with Test Engine here: https://www.braindumpspass.com/Anthropic/CCAR-F-practice-exam-dumps.html (**191** Q&As Dumps, **40%OFF** **Special Discount:** **Exam-Tests**)

NEW QUESTION: 17
Your multi-agent research pipeline crashed after processing 12 of 28 documents. The web-search agent had identified relevant sources, the document analyzer had partially completed extraction, and the synthesizer had begun identifying patterns. You need to resume processing without repeating work or losing fidelity in the prior findings. What state-management approach best balances information fidelity with context efficiency when restoring agent state?
A. Have each agent persist a structured export to a known location. On resumption, the coordinator loads the manifest and injects relevant state into agent prompts.
B. Have each agent maintain its own persistent state file and reload it independently at the beginning of each session.
C. Persist the coordinator's conversation log containing all task delegations and responses, and provide this log to the agents when resuming.
D. Index all agent outputs in a shared vector store. When resuming, have each agent query the store using semantic search to retrieve relevant prior findings.
Answer: A (LEAVE A REPLY)
Option A combines lossless persistence with selective restoration. Each agent should checkpoint structured records containing completed document identifiers, extracted findings, source provenance, unresolved work, verification status, and schema version. A coordinator-level manifest can record which stages completed and where their artifacts are stored. On recovery, the coordinator skips completed documents and injects only the state required by each restarted agent.
Anthropic's effective context-engineering guidance recommends structured notes stored outside the active context window and reloaded when needed. This preserves critical dependencies and progress without replaying an entire execution history.
Option B fragments recovery ownership across agents and makes cross-agent consistency, versioning, and dependency validation difficult. Option C preserves extensive logs but wastes context on redundant tool calls, obsolete instructions, and intermediate messages. Option D is useful for relevance retrieval but semantic search can omit exact details and should not serve as the authoritative recovery record. Structured checkpoints and a validated manifest provide deterministic recovery, efficient context reconstruction, auditability, and precise provenance. Writes should be atomic so partially produced exports are never mistaken for completed work.

NEW QUESTION: 18
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?
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: (SHOW ANSWER)
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.

NEW QUESTION: 19
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 system has been operating with 100% human review for 3 months. Analysis shows that extractions with model confidence #90% have 97% accuracy overall. To reduce reviewer workload, you plan to automate high- confidence extractions.

Before deploying, what validation step is most critical?
A. Analyze accuracy by document type and field to verify high-confidence extractions perform consistently across all segments, not just in aggregate.
B. Compare accuracy at different confidence thresholds (85%, 90%, 95%) to find the optimal cutoff that maximizes automation while minimizing errors.
C. Verify that 97% accuracy meets requirements for all downstream systems that consume the extracted data.
D. Run a two-week pilot routing 25% of high-confidence extractions directly to downstream systems and monitor error reports.
Answer: (SHOW ANSWER)
An aggregate accuracy value can conceal severe performance disparities. A system may achieve 97% overall accuracy while performing poorly on a low-volume document type, a critical financial field, or a specific edge case. Automating outputs solely from the aggregate figure could therefore expose downstream systems to concentrated, high-impact errors.
Anthropic's evaluation guidance states that evaluations should be task-specific, reflect the real-world task distribution, and explicitly include edge cases. It also emphasizes multidimensional success criteria rather than reliance on a single global metric. (https://docs.anthropic.com/en/docs/build-with-claude/develop-tests) Option A applies those principles by stratifying performance according to document type and field. This reveals whether confidence is calibrated consistently and whether the proposed automation threshold remains safe for every operationally significant segment.
Option B is useful only after segment-level performance has been understood. Selecting a global threshold cannot correct a subgroup where confidence is systematically overstated. Option C is necessary governance work, but it treats the overall 97% result as though errors were uniformly distributed. Option D places unvalidated outputs into downstream systems and depends on passive error reporting, which may fail to detect silent corruption. The correct deployment gate is therefore segmented validation, followed by threshold selection, downstream acceptance criteria, and a controlled pilot.
Official references/topics: Define Success Criteria; Task-Specific Evaluations; Edge-Case Coverage; Reliability Segmentation.

NEW QUESTION: 20
Users report that final reports sometimes lack depth on specific subtopics. Investigation shows that the document-analysis agent frequently identifies evidence gaps-for example, noting that "the retrieved sources discuss API authentication but lack details about token-refresh patterns." Under the current strict pipeline, this insight is not actionable because searching has already finished. What is the most effective architectural change?
A. Add a research-planning agent before the initial search phase to decompose every topic into detailed subquestions.

B. Have the synthesis agent assign confidence scores to each report section and flag insufficiently supported sections for manual review.
C. Require the analysis agent to return specific evidence gaps to the coordinator, which launches targeted searches and invokes analysis again until the defined coverage criteria are satisfied.
D. Have the coordinator look for general gap indicators in the analysis output and run additional searches without repeating the analysis stage.
Answer: (SHOW ANSWER)
Option C converts the linear pipeline into a controlled iterative research loop. The document-analysis agent is best positioned to identify precisely what the retrieved evidence fails to answer. It should return structured gaps containing the missing question, evidence already examined, preferred source characteristics, and the coverage criterion that remains unsatisfied. The coordinator can then issue focused searches and resubmit the new material for analysis.
Anthropic's multi-agent research architecture uses a lead agent to decompose work, coordinate research agents, assess returned information, and continue investigation when further evidence is required. The guidance also stresses clear objectives, output formats, source requirements, and task boundaries to prevent gaps and duplicated work.
Option A may improve initial coverage but cannot anticipate every deficiency revealed only after sources are examined. Option B documents incompleteness rather than correcting it. Option D retrieves more information but does not ensure that the new evidence is analyzed and tested against the original gap. The loop in Option C establishes an explicit completion condition and preserves specialization: analysis identifies the gap, the coordinator orchestrates, search agents retrieve evidence, and analysis verifies whether the gap has actually been closed.

NEW QUESTION: 21
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 review jobs take 18 seconds to initialize before Claude begins analyzing code. Profiling reveals that the delay results from automatically discovering hooks, MCP servers, plugins, skills, and multiple nested CLAUDE.md files throughout the monorepo. You need to reduce startup time while ensuring reviews still enforce the coding standards documented in the root-level CLAUDE.md file.
What is the most effective approach?
A. Replace the default prompt using --system-prompt-file ./CLAUDE.md, which bypasses default prompt assembly and loads only the project rules.

B. Run in --bare mode and pass --append-system-prompt-file ./CLAUDE.md to load the required project standards explicitly while skipping automatic discovery.
C. Run in --bare mode and repeat all review criteria directly in the -p prompt for every invocation.
D. Keep the default initialization and add --exclude-dynamic-system-prompt-sections to improve prompt- cache reuse across CI runners.
Answer: (SHOW ANSWER)
Option B removes the identified startup work while deliberately restoring the one source of project context the reviews require. Anthropic documents that --bare skips automatic discovery of hooks, skills, plugins, MCP servers, auto memory, and CLAUDE.md files. It is specifically intended for CI and scripted execution where fast, reproducible startup behavior is more important than loading every locally configured extension.
Because bare mode also skips the root CLAUDE.md, the pipeline must supply that content explicitly. -- append-system-prompt-file ./CLAUDE.md loads the standards while retaining Claude Code's default coding- agent behavior and tool guidance. Option A replaces the default system prompt but does not, by itself, establish the same minimal startup path as --bare; replacement also discards valuable default coding instructions. Option C can work technically but duplicates repository policy inside every pipeline invocation and creates configuration drift. Option D improves prompt-cache reuse by relocating machine-specific prompt sections, but it does not eliminate discovery of hooks, MCP servers, plugins, skills, and nested instructions.
Bare mode plus an explicitly appended standards file directly addresses both performance and policy requirements. Claude Code bare-mode documentation

NEW QUESTION: 22
You are building a customer support resolution agent using the Claude Agent SDK. The agent handles high- ambiguity requests like returns, billing disputes, and account issues. It has access to your backend systems through custom Model Context Protocol (MCP) tools (get_customer , lookup_order , process_refund , escalate_to_human). Your target is 80% + first-contact resolution while knowing when to escalate.
Production logs reveal inconsistent error handling: when lookup_order fails, the agent sometimes retries 5+ times (wasteful when the order ID doesn't exist), sometimes escalates immediately (premature for temporary network issues), and sometimes asks users for clarification (inappropriate when the issue is a backend permission error).
Investigation shows your MCP tool returns uniform error responses: { " isError " : true, " content " : [{ " type " : " text " , " text " : " Operation failed " }]} . The agent cannot distinguish between error types.
What's the most effective improvement?
A. Enhance error responses with structured metadata-include error_category (transient/validation
/permission), isRetryable boolean, and a description of what caused the failure.

B. Implement retry logic with exponential backoff in your MCP server for all errors, returning to the agent only after retries are exhausted.
C. Create an analyze_error MCP tool the agent calls after any failure to determine the error category and recommended action.
D. Add few-shot examples to the system prompt demonstrating how to interpret error message patterns and select appropriate responses for each.
Answer: A (LEAVE A REPLY)
The agent is behaving inconsistently because every failure is represented identically. "Operation failed" contains no information about permanence, user correctability, authorization, or whether another attempt is likely to succeed. Adding structured fields converts the error into an actionable interface contract.
Anthropic instructs tool implementations to mark failures using an error flag and return information Claude can use to retry with corrected input, ask for clarification, or explain a limitation. Tool design should expose sufficient detail for the model to choose the correct next action rather than forcing it to infer the cause from a generic message.
(https://platform.claude.com/docs/en/agents-and-tools/tool-use/build-a-tool-using-agent?utm_source=chatgpt.com)
With error_category , isRetryable , and a causal description, the agent can retry transient network failures, request corrected identifiers for validation errors, and escalate or report permission failures without pointless repetitions. Option B retries permanent validation and permission failures unnecessarily. Option C adds another tool call and failure point merely to classify information the original tool already possesses. Option D relies on parsing variable text and examples rather than providing explicit machine-readable semantics.
The tool should preserve isError: true , return a stable structured error object, include a safe customer-facing message where appropriate, and avoid exposing internal secrets or stack traces.
Official references/topics: MCP error contracts, actionable tool results, retryability metadata, agent- computer interface design.

NEW QUESTION: 23
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 review calls the Claude API for each pull request, using tool_use with a report_findings tool that returns a JSON array of finding objects. Each object contains file_path, line_number, severity, category, and description. During testing on a large pull request touching more than 30 files, the response reaches the max_tokens limit and is truncated in the middle of the JSON, causing your pipeline's parser to fail.
What is the most effective way to handle this?

A. Split the review into multiple API calls that each analyze a subset of the changed files, and then merge the resulting findings arrays.
B. Increase max_tokens to the model's maximum and instruct Claude to keep each finding description under 50 words.
C. Switch from tool_use to prompting Claude to return findings as a Markdown list.
D. Add retry logic that detects truncated JSON and resends the request with instructions to report only critical and high-severity findings.
Answer: (SHOW ANSWER)
Option A is correct because the failure results from excessive output volume, not from JSON or tool use.
Anthropic documents that max_tokens is a hard output ceiling and that a response stopped at this limit reports stop_reason: " max_tokens " . Its tool-streaming guidance also warns that generation can stop midway through a tool parameter, leaving partial JSON that must not be treated as complete. Splitting the pull request into bounded file groups limits the number of findings produced by each call, preserves the report_findings schema, and allows the pipeline to validate every response independently before merging and deduplicating the arrays. Shared dependency context can still be included when cross-file analysis is required.
Option B only postpones the failure because the model's output maximum remains finite.
Option C abandons the structured contract and makes deterministic parsing less reliable.
Option D changes the review requirements by suppressing potentially valuable lower-severity findings, while the retried request could still reach the same limit. Controlled partitioning followed by deterministic aggregation is therefore the most reliable and scalable design. Anthropic stop-reason guidance and tool-streaming guidance .

NEW QUESTION: 24
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 infrastructure-as-code repository includes Terraform modules (/terraform/), Kubernetes manifests (
/kubernetes/), and CI/CD pipeline scripts (/pipelines/). Each requires different conventions, but your single root CLAUDE.md has grown to 500+ lines. When developers work on Kubernetes files, Terraform-specific rules load into context unnecessarily, consuming tokens.
What is the best approach to reorganize so only relevant guidance loads when editing specific file types?
A. Create files in .claude/rules/ with YAML frontmatter path-scoping (e.g., paths: ["terraform/**/*.tf"]), loading rules only when editing matching files.

B. Restructure the root CLAUDE.md into clearly labeled sections with headers (e.g., "## Terraform Conventions"), improving organization and readability.
C. Split content into subdirectory CLAUDE.md files (/terraform/CLAUDE.md , /kubernetes/CLAUDE.md), so Claude loads directory-specific guidance.
D. Keep the root CLAUDE.md and use @path/to/import syntax to modularly include tool-specific guidance files from separate documents.
Answer: A (LEAVE A REPLY)
Path-scoped rules directly satisfy the requirement that instructions load only when Claude works with matching files. Each infrastructure domain can have a separate Markdown file under .claude/rules/ , with YAML paths frontmatter targeting Terraform files, Kubernetes manifests, or pipeline configuration.
Anthropic documents that .claude/rules/ keeps project guidance modular and supports conditional loading based on glob patterns. Rules with a paths field apply only when Claude works with matching files, reducing irrelevant context and token consumption. Rules without path frontmatter load unconditionally. (https://code.claude.com/docs/en/memory)
Option B improves readability but leaves all 500-plus lines in every session. Option C can provide directory- specific CLAUDE.md guidance, but path-scoped rules are more precise when conventions depend on extensions or patterns spanning multiple directories. Option D still requires manual imports and does not provide the documented conditional rule-loading mechanism.
A suitable design would use rules such as terraform.md scoped to terraform/**/*.tf , kubernetes.md scoped to kubernetes/**/*.{yaml,yml} , and pipelines.md scoped to the pipeline directory. Broad project-wide commands and repository etiquette can remain in the root CLAUDE.md.
Official references/topics: .claude/rules/ ; Path-Specific Rules; YAML Frontmatter; Context-Efficient Instruction Loading.

NEW QUESTION: 25
You are building a multi-agent research system using the Claude Agent SDK. A coordinator agent delegates to specialized subagents: one searches the web, one analyzes documents, one synthesizes findings, and one generates reports. The system researches topics and produces comprehensive, cited reports.
After the web-search and document-analysis subagents complete their tasks, the coordinator needs to spawn the synthesis subagent to synthesize the findings.
What is the correct approach for providing the synthesis subagent with the information it needs?
A. Provide the subagent with tool definitions that allow it to request outputs from other subagents through callbacks.

B. Include the complete findings from both subagents directly in the synthesis subagent's prompt.
C. Spawn the subagent with only a brief task description, relying on automatic context inheritance from the coordinator.
D. Pass reference identifiers and configure the subagent with read access to a shared memory store where the other subagents deposited their results.
Answer: (SHOW ANSWER)
Option B follows the Claude Agent SDK's subagent context model. A normal subagent begins with a fresh context window and does not inherit the parent agent's conversation history or prior tool results. Anthropic's Subagents in the SDK documentation states that the information passed from parent to subagent is the spawning tool's prompt string; required file paths, decisions, errors, or findings must therefore be included in that prompt. In this scenario, the coordinator should supply both agents' relevant findings, source metadata, and explicit synthesis instructions. "Complete findings" means the full required result artifacts, not every intermediate search trace. Option C incorrectly assumes automatic context inheritance. Option A introduces callbacks that are neither necessary nor the standard handoff mechanism. Option D can be a valid custom architecture when a shared store has deliberately been implemented, but the question does not establish such infrastructure, and identifiers alone do not give the subagent information. The prompt should use clear sections or structured objects to distinguish web findings, document findings, sources, unresolved conflicts, and expected output. This preserves context isolation while providing everything the synthesis task actually requires.

NEW QUESTION: 26
Your automated code review is missing genuine bugs in pull requests. Investigation reveals that the review prompt includes this instruction: "Only flag critical issues that would definitely cause production failures.
Ignore minor concerns and anything you are uncertain about." Developers confirm that some missed findings are genuine logic errors that the model investigated but chose not to report. The team requires the review output to remain structured, with every finding tagged with metadata, and actionable. Which prompt change both removes the cause of the suppressed findings and preserves structured, tagged output for downstream filtering?
A. Enable extended thinking and instruct the model to reason step by step about every code change before producing its review.
B. Instruct the model to report all findings with confidence and severity tags, deferring filtering to a downstream step.
C. Remove all severity-related instructions and allow the model to use its default judgment about which findings to report.
D. Add a second review pass that rereads the diff using the same prompt and looks for anything the first pass may have missed.
Answer: (SHOW ANSWER)

The prompt explicitly instructs Claude to suppress uncertain and lower-severity findings. Increasing reasoning depth cannot override that reporting policy: Claude may identify a real defect during analysis and still omit it from the final response. Option B separates comprehensive detection from acceptance filtering, removing the source of the false negatives while retaining metadata needed for automated decisions.
Anthropic's current code-review prompting guidance recommends reporting every issue, including uncertain or lower-severity candidates, and attaching confidence and estimated severity so a separate verification or filtering stage can rank them. A structured schema can additionally require fields such as file path, line number, category, confidence, severity, evidence, and recommended action.
Option A may improve investigation quality but leaves the suppression instruction intact. Option C removes the problematic threshold but also discards the explicit structured classification needed by downstream processing. Option D repeats the same review policy, so the second pass can suppress the same findings as the first. Option B preserves recall and machine-readable output while allowing deterministic, adjustable downstream thresholds. This is more reliable than embedding an overly restrictive acceptance decision inside the model's initial detection task.

NEW QUESTION: 27
You are building developer productivity tools using the Claude Agent SDK. The agent helps engineers explore unfamiliar codebases, understand legacy systems, generate boilerplate code, and automate repetitive tasks. It uses the built-in tools (Read, Write, Bash, Grep, Glob) and integrates with Model Context Protocol (MCP) servers.
Your agent has analyzed a complex service module-reading 23 source files, tracing request flows, and identifying error handling patterns. A developer wants to compare two testing strategies before committing to one: end-to-end tests with mocked external services vs. snapshot tests capturing expected outputs. They need to independently develop both approaches to evaluate trade-offs.
How should you manage the sessions?
A. Resume the analysis session with fork_session enabled, creating a separate branch for each testing strategy.
B. Start two fresh sessions, having each re-read the relevant source files before beginning.
C. Continue in the original session, developing end-to-end tests first, then snapshot tests sequentially.
D. Export the analysis session's key findings to a file, then create two new sessions that reference this file.
Answer: (SHOW ANSWER)
Forking the existing analysis session creates independent continuations that inherit the accumulated conversation context. Each branch begins with the same understanding of the service module, request flow, source files, and error-handling patterns, but subsequent work on one testing strategy does not alter the other branch or the original session.

Anthropic's Agent SDK documentation states that sessions can be resumed with their full context and forked to explore different approaches. In the SDK, enabling fork_session while resuming causes the continuation to receive a new session identifier rather than modifying the original session. (https://docs.anthropic.com/en /docs/claude-code/sdk?utm_source=chatgpt.com)
Option B wastes time, tokens, and tool calls by requiring both new sessions to rebuild the same 23-file analysis. Option C mixes two experimental implementations into one conversation, increasing the risk that assumptions, edits, or conclusions from the first strategy influence the second. Option D preserves only a manually selected summary, which may omit details contained in the full session history.
The appropriate design is to create one fork for the end-to-end strategy and another fork for the snapshot strategy. The original analysis remains a stable parent, while each child session develops and evaluates its approach independently.
Official references/topics: Agent SDK Sessions, Session Forking, Context Preservation, Alternative- Approach Evaluation.

NEW QUESTION: 28
After investigating a billing dispute for more than 25 turns, you determine that duplicate charges resulted from a payment-gateway timeout triggering retry logic. The required refund of $847 exceeds your $500 authorization limit, so you must invoke escalate_to_human. The human agent will not have access to the conversation transcript. What context should you pass to enable effective resolution?
A. The complete conversation transcript containing every message and tool result.
B. The customer's original complaint verbatim together with excerpts from the tool results showing the duplicate transactions.
C. A structured summary containing the customer identifier, verified root cause, refund amount, relevant transaction identifiers, actions already attempted, and recommended next action.
D. Only the diagnosis and refund amount.
Answer: (SHOW ANSWER)
Option C gives the human agent the operational state required to continue without replaying a long conversation. The handoff should include verified identifiers, the duplicate transaction evidence, the diagnosed timeout-and-retry mechanism, the $847 refund requirement, the agent's $500 authorization constraint, completed verification steps, prior actions, and the recommended resolution. Any unresolved uncertainty should be labeled explicitly.
Anthropic's effective context-engineering guidance recommends preserving high-value state in structured notes while removing redundant conversational and tool-call history. Its long-running-agent guidance likewise describes structured handoffs as the mechanism for maintaining continuity across context resets or agent boundaries.

Option A maximizes raw information but forces the human to locate the relevant facts among more than 25 turns, increasing delay and error risk. Option B preserves evidence but omits the authorization constraint, completed verification, and explicit recommended action. Option D is too sparse to support validation or execution. A structured handoff balances fidelity and efficiency: it contains everything needed for the next actor to make the refund decision while excluding greetings, repeated explanations, and irrelevant intermediate tool output.

NEW QUESTION: 29
You are building a structured data extraction system using Claude. The system extracts information from unstructured documents, validates the output using JSON schemas, and maintains high accuracy. It must handle edge cases gracefully and integrate with downstream systems.
Your extraction system processes two document types: standard monthly reports, which are archived after processing, and urgent exception reports, which must trigger business alerts within 30 minutes of receipt.
Both use the same JSON schema. You want to minimize API costs while meeting the latency requirements.
How should you architect the processing pipeline?
A. Submit all documents to the Message Batches API with custom_id values for tracking. When results arrive, immediately process urgent documents and trigger delayed alerts for exceptions.
B. Route standard reports to the Message Batches API for 50% cost savings, and route urgent exception reports to the real-time Messages API.
C. Queue all documents and submit hourly batches, flagging urgent documents for expedited handling when batch results return.
D. Submit all documents to the real-time Messages API to ensure consistent processing latency across document types.
Answer: B (LEAVE A REPLY)
Option B correctly separates asynchronous bulk work from latency-sensitive operational work. Anthropic describes the Message Batches API as a cost-effective asynchronous service for high-volume requests that do not require immediate responses. Batch requests are processed independently, normally finish within an hour, may remain pending for up to 24 hours, and are billed at 50 percent of standard API prices. Those characteristics fit standard monthly reports because they are archived after processing and have no immediate business deadline. Urgent exception reports, however, must trigger an alert within 30 minutes. The batch service cannot guarantee that deadline, even if many batches finish quickly; a workload should not use typical completion time as a service-level guarantee. Routing exceptions through the synchronous Messages API provides the real-time response path needed for alerting, while the shared JSON schema keeps downstream records consistent across both routes. Option A knowingly permits delayed

alerts. Option C introduces an hourly submission delay before processing even begins. Option D meets latency requirements but wastes the batch discount on non-urgent reports. A routing layer based on document priority therefore satisfies both the cost objective and the business deadline without changing the extraction contract.

NEW QUESTION: 30
You are building developer productivity tools using the Claude Agent SDK. The agent helps engineers explore unfamiliar codebases, understand legacy systems, generate boilerplate code, and automate repetitive tasks. It uses the built-in tools (Read, Write, Bash, Grep, Glob) and integrates with Model Context Protocol (MCP) servers.
1.5An engineer asks the agent to understand how the caching layer works before adding a new cache invalidation trigger. After initial Grep searches, the agent has identified that caching logic spans 15 files including decorators, middleware, and service classes (~6,000 lines total).
What's the most effective next step for building understanding while managing context constraints?
A. Use Grep to search for "invalidate" and "expire" patterns across all files, then Read only those specific line ranges with minimal surrounding context.
B. Use the Read tool to sequentially load all 15 files, building complete understanding across the full caching implementation.
C. Use Glob to find files matching common caching patterns (cache*.py , caching/), prioritize the largest files by reading them first, then check smaller files for gaps.
D. Analyze imports and class hierarchies to identify the base cache class. Read that file to understand the interface, then trace specific invalidation implementations.
Answer: D (LEAVE A REPLY)
The correct objective is to construct an architectural map before consuming the full implementation.
Identifying the base cache abstraction, its interface, and the classes that implement or invoke it gives the agent a dependency-guided path through the code. It can then inspect only the invalidation implementations and integration points relevant to the proposed trigger.
This approach protects the context window. Anthropic states that every file read occupies context and that model performance can deteriorate as the window fills. Its Claude Code guidance warns against unbounded investigation that reads large numbers of files and recommends narrowing the exploration or delegating it. (
https://code.claude.com/docs/en/best-practices)
Option A is too lexical: searching only for invalidate or expire can miss event-driven invalidation, overridden methods, cache-key mutation, and generic interface calls. Option B loads approximately 6,000 lines without first establishing relevance. Option C assumes that filename patterns and file size correlate with architectural importance; the largest files may contain incidental code while a small interface defines the entire design.

Option D follows control and type relationships rather than arbitrary file order. After reading the base class, the agent can search for subclasses, imports, construction sites, middleware hooks, and calls to the invalidation contract, progressively expanding only where evidence requires it.
Official references/topics: Context-Efficient Exploration; Dependency-Guided Reading; Architectural Interfaces; Narrowly Scoped Investigation.

NEW QUESTION: 31
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.
Monitoring shows 12% of extractions fail Pydantic validation with specific errors like "expected float for quantity, got '2 to 3'". Retrying these requests without modification produces identical failures.
What's the most effective approach to recover from these validation failures?
A. Send a follow-up request including the validation error, asking the model to correct its output.
B. Set temperature to 0 to eliminate output variability and ensure consistent formatting.
C. Pre-process source documents to standardize problematic formats before sending them for extraction.
D. Implement a secondary pipeline using a larger model tier to reprocess documents that fail validation.
Answer: A (LEAVE A REPLY)
An unchanged retry repeats the same task specification and therefore commonly reproduces the same invalid interpretation. The validator has generated precise corrective information- quantity requires a float, but the model returned the range string 2 to 3 . Supplying that error in a follow-up turn converts a generic retry into an iterative repair operation.
Anthropic identifies iterative refinement as a method for detecting and correcting inconsistencies by feeding an earlier output back into a subsequent request with targeted instructions. (https://docs.anthropic.com/en/docs /test-and-evaluate/strengthen-guardrails/reduce-hallucinations) Option A applies that pattern directly. Claude receives the invalid output, the exact Pydantic error, and an instruction to return a schema-compliant correction. The application should cap retries, retain the original source, and escalate cases that cannot be represented without information loss.
Option B does not guarantee correct formatting; lower temperature may make the same wrong output more repeatable. Option C can be valuable for systematic OCR or source-format problems, but it is unnecessarily broad when the immediate failure is already

described by the validator. Option D increases cost and complexity without first using the actionable feedback available from the existing validation layer.
For supported models, native Structured Outputs should also be considered because they guarantee schema- conformant JSON and can prevent many Pydantic shape failures before they occur. (https://platform.claude.
com/docs/en/build-with-claude/structured-outputs)
Official references/topics: Iterative Refinement; Validation-Error Feedback; Bounded Retry Loops; Structured Outputs.

Valid CCAR-F Dumps shared by BraindumpsPass.com for Helping Passing CCAR-F Exam! BraindumpsPass.com now offer the **newest CCAR-F exam dumps**, the BraindumpsPass.com CCAR-F exam **questions have been updated** and **answers have been corrected** get the **newest** BraindumpsPass.com CCAR-F dumps with Test Engine here: https://www.braindumpspass.com/Anthropic/CCAR-F-practice-exam-dumps.html (**191** Q&As Dumps, **40%OFF** **Special Discount:** **Exam-Tests**)

NEW QUESTION: 32
You are building developer productivity tools using the Claude Agent SDK. The agent helps engineers explore unfamiliar codebases, understand legacy systems, generate boilerplate code, and automate repetitive tasks. It uses the built-in tools (Read, Write, Bash, Grep, Glob) and integrates with Model Context Protocol (MCP) servers.
Your codebase exploration tool stores session IDs to allow engineers to continue investigations across work sessions. An engineer spent an hour yesterday analyzing a legacy authentication module, building context about its architecture and dependencies. They want to continue today. The session ID is valid, but version control shows 3 of the 12 files the agent previously read were modified overnight by a teammate's merge.
What approach best balances efficiency and accuracy?
A. Start a fresh session to ensure the agent works with current codebase state without stale assumptions
B. Resume the session and inform the agent which specific files changed for targeted re-analysis
C. Resume the session and immediately have the agent re-read all 12 previously analyzed files
D. Resume the session without informing the agent about the changed files
Answer: B (LEAVE A REPLY)
Resuming preserves the prior architectural analysis, dependency mapping, and conversation history, avoiding the cost of repeating an hour of valid work. Anthropic states that resuming by session ID restores the agent's full prior context, including files read, analysis performed, and decisions made. (https://code.claude.com/docs

/en/agent-sdk/sessions)
However, the session represents historical knowledge rather than a repository snapshot. Because three files changed externally, the agent must be told which files are affected and directed to re-read them before relying on earlier conclusions. Anthropic clarifies that sessions persist the conversation, not the filesystem. (https://code.claude.com/docs/en/agent-sdk/sessions) Targeted re-analysis updates the stale portion of the agent's model while retaining the nine unchanged files' established context.
Option A discards valuable work even though most of the analyzed code remains unchanged. Option C is accurate but inefficient because it reprocesses all twelve files regardless of whether they changed. Option D is unsafe: the agent may continue reasoning from obsolete function signatures, dependencies, or control flows.
The continuation prompt should identify the changed files, summarize the merge's purpose where known, and request a focused comparison against the earlier understanding. Any architectural conclusions affected by those changes should then be revised explicitly.
Official references/topics: Session Resume; Filesystem Versus Conversation State; Targeted Re-Analysis; Repository Change Awareness.

NEW QUESTION: 33
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 test generation produces unit tests for new code, but reviews show that 55% are low-value: trivial assertions that only verify functions do not throw exceptions, tests duplicating existing coverage, or tests ignoring your team's fixture conventions.
How do you reduce the rate of low-value tests being generated in the first place?
A. Implement two-phase generation in which a second Claude call scores each test against quality criteria, filtering out low-scoring tests before presenting results to developers.
B. Add post-generation coverage analysis that automatically filters out any generated test that does not increase line coverage beyond existing tests.
C. Restrict test generation to directories where historical quality metrics show higher acceptance rates, disabling it for areas where generated tests consistently require substantial editing.
D. Document testing standards in CLAUDE.md, including valuable-test criteria, available fixtures and their intended use cases, and examples distinguishing meaningful behavioral tests from trivial assertions.
Answer: D (LEAVE A REPLY)

Option D improves the generation conditions instead of filtering defective output afterward. Anthropic recommends using CLAUDE.md for repository-wide standards and information that Claude should apply in every session. The file can define what constitutes a useful test, require assertions on observable behavior, identify approved fixture factories, prohibit duplication of existing scenarios, and provide contrasting examples of strong and trivial tests. These instructions are available before Claude decides what tests to create.
Option A may remove some weak tests, but it doubles model work and still permits the first stage to generate low-quality material. Option B treats line coverage as a quality measure even though a trivial assertion can execute new lines without validating behavior. Option C abandons difficult areas rather than improving generation. The standards should also direct Claude to inspect nearby tests and fixtures before writing new cases, identify the behavior or regression each test protects, and run the relevant suite afterward. Because CLAUDE.md consumes context in every session, the guidance should remain concise; lengthy testing procedures are better placed in a reusable skill referenced from the project instructions.

NEW QUESTION: 34
You are building a customer support resolution agent using the Claude Agent SDK. The agent handles high- ambiguity requests like returns, billing disputes, and account issues. It has access to your backend systems through custom Model Context Protocol (MCP) tools (get_customer , lookup_order , process_refund , escalate_to_human). Your target is 80% + first-contact resolution while knowing when to escalate.
The agent verifies customer identity through a multi-step process before resetting passwords. During testing, you notice that after the customer answers the third verification question, the agent asks them to provide their name again, as if the earlier exchange never happened.
What's the most likely cause of this behavior?
A. The prompt lacks instructions telling Claude to remember information across multiple exchanges.
B. The conversation history isn't being passed in subsequent API requests.
C. The verification tool is clearing the agent's internal state after each successful validation step.
D. Claude's memory retention is limited to two conversational turns by default, requiring explicit configuration to extend it.
Answer: B (LEAVE A REPLY)
The Claude Messages API is stateless. Each API request must include the conversation history that Claude requires for the current response. If the application sends only the customer's third verification answer, Claude does not automatically retrieve the earlier turns containing the customer's name and previous answers.
From the model's perspective, that information is absent.

Anthropic's Messages API documentation states that applications must send the full conversational history to build a multi-turn interaction. The correct implementation appends each user message and assistant response to the messages array and resends the accumulated sequence on every subsequent request. (https://platform.claude.com/docs/en/build-with-claude/working-with-messages)
Option A cannot restore information that was omitted from the request. Instructions to "remember" do not create server-side conversational state. Option C describes application-specific behavior for which the scenario provides no evidence. Option D is factually incorrect: Claude does not have a default two-turn retention limit. Its effective conversational awareness depends on the messages provided and the model's context window.
For a verification workflow, the application should also persist structured verification state separately from prose history. That state may include completed checks, pending checks, attempt counts, and a verified customer identifier. This improves reliability while the conversational transcript preserves the natural interaction.
Official references/topics: Stateless Messages API, multi-turn history construction, structured workflow state, conversational continuity.

NEW QUESTION: 35
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 caching layer for API responses to speed up the /products endpoint. You have a rough idea-Redis with a 5-minute TTL-but you're new to production caching and aren't sure what other considerations a robust implementation requires. What's the most effective way to start your iterative workflow?
A. Ask Claude to interview you about the caching requirements before implementing, surfacing considerations like invalidation strategies, cache layers, consistency guarantees, and failure modes.
B. Use plan mode to analyze the current /products endpoint implementation, then provide your caching requirements once Claude explains how the existing code is structured.
C. Start with a minimal request: "Add Redis caching to /products with 5-minute TTL." Add features and fix issues through follow-up prompts as problems surface during testing.
D. Write a specification with your known requirements and "TBD" markers for uncertain areas, having Claude propose solutions for each TBD as it implements.
Answer: (SHOW ANSWER)
The primary risk is not implementation difficulty but incomplete requirements. Production caching introduces decisions involving invalidation, stale-data tolerance, cache keys, tenant boundaries, serialization, stampede prevention, failure behavior, observability, deployment topology, and consistency expectations. Implementing Redis with a five-

minute TTL before resolving these questions can produce a technically functional but operationally unsafe design.
Anthropic recommends having Claude interview the user before beginning a larger feature when important requirements remain uncertain. The AskUserQuestion workflow is intended to surface technical implementation concerns, edge cases, trade-offs, and assumptions the user may not have considered.
Anthropic further recommends converting the resulting answers into a self-contained specification with explicit scope and an end-to-end verification step.
(https://code.claude.com/docs/en/best-practices) Option B provides useful codebase context but postpones requirement discovery. Option C creates avoidable rework by allowing architecture to emerge from production failures. Option D documents uncertainty but delegates unresolved design choices during implementation, when they may already constrain the code.
After the interview produces a caching specification, Claude can enter plan mode to inspect the endpoint and map those requirements onto the existing architecture before implementation.
Official references/topics: Requirements Interviewing; AskUserQuestion; Specification Development; Edge- Case and Trade-Off Discovery.

NEW QUESTION: 36
You are building a customer support resolution agent using the Claude Agent SDK. The agent handles high- ambiguity requests like returns, billing disputes, and account issues. It has access to your backend systems through custom Model Context Protocol (MCP) tools (get_customer , lookup_order , process_refund , escalate_to_human). Your target is 80% + first-contact resolution while knowing when to escalate.
A customer contacts the agent about a warranty claim on a power drill. Resolving this requires multiple sequential tool calls: get_customer to look up their account, lookup_order to find the purchase details, and then either process_refund or escalate_to_human depending on warranty eligibility. You're implementing the agentic loop that orchestrates these steps using the Claude API.
What is the primary mechanism your application uses to determine whether to continue the loop or stop?
A. You check whether Claude's response contains a text content block-if text is present, the agent has produced its final answer and the loop should exit.
B. You manually set the tool_choice parameter to "none" after the final expected tool call to force Claude to stop requesting tools.
C. You check the stop_reason field in each API response-the loop continues while it equals "tool_use" and exits when it changes to "end_turn" or another terminal value.
D. You track the number of tool calls made and exit the loop once a preconfigured maximum is reached.
Answer: C (LEAVE A REPLY)

NEW QUESTION: 37
You are building a structured data extraction system using Claude. The system extracts information from unstructured documents, validates the output using JSON schemas, and maintains high accuracy. It must handle edge cases gracefully and integrate with downstream systems.
Your extraction system parses e-commerce product descriptions to extract specifications such as dimensions, weight, and materials into JSON. Despite having a well-defined schema, the model inconsistently extracts the materials field-sometimes returning "cotton blend," other times "Cotton/Polyester mix," and occasionally omitting the field when material information is clearly present in the source.
What is the most effective way to improve extraction consistency?
A. Set the temperature to 0 to eliminate randomness and ensure deterministic outputs.
B. Switch to a more capable model tier because inconsistent extraction indicates insufficient model capability.
C. Make the materials field required instead of optional in the schema to force the model to always extract a value.
D. Add few-shot examples showing two or three complete input-output pairs with standardized material- description formats.
Answer: D (LEAVE A REPLY)
Option D addresses the observed inconsistency at the model-behavior level. A schema can require that materials be a string, but it cannot teach Claude which lexical form the application considers canonical or demonstrate when a source phrase should populate the field. Anthropic identifies examples as one of the most reliable ways to steer output format, structure, and consistency, recommending several relevant and diverse examples that mirror the real task in its prompting best practices . Complete input-output pairs show both recognition and normalization: for example, a description containing "60% cotton, 40% polyester" can consistently map to the chosen "cotton/polyester blend" representation. They can also include difficult cases where material information is embedded indirectly in prose. Temperature zero reduces sampling variability but does not repair an underspecified transformation rule. A more capable model likewise lacks the missing formatting convention. Making the field required is dangerous because documents may legitimately omit materials; it can force unsupported values and increase hallucinations. Few-shot examples therefore supply the missing decision boundary while preserving truthful absence handling. The examples should be evaluated on held-out descriptions and supplemented by deterministic post-processing if downstream systems require an exact controlled vocabulary.

NEW QUESTION: 38
Production monitoring shows that follow-up queries such as "summarize what we learned about market trends" consistently take more than 40 seconds. Investigation reveals that

the coordinator spawns the synthesis subagent for every summarization request, passing more than 80,000 tokens of accumulated findings. The coordinator already has these findings in its context from orchestrating the research. What is the most effective way to improve response time for these follow-up summaries?

A. Pregenerate and cache summaries at multiple levels of detail whenever new findings accumulate.

B. Enable prompt caching for the synthesis subagent to reduce the overhead of repeatedly transferring the same research findings.

C. Have the coordinator answer straightforward summarization requests directly from its existing context, reserving subagent invocation for complex analysis.

D. Spawn the synthesis subagent with reduced context and allow it to request specific findings from the coordinator on demand.

Answer: (SHOW ANSWER)

Option C avoids an unnecessary agent boundary. The coordinator already possesses the accumulated findings and can perform a straightforward summary without serializing, transferring, and reprocessing more than
80,000 tokens in another context window. Subagents should be reserved for work that requires isolated context, specialized instructions, separate tools, or an independent analytical process.

Anthropic's current prompting guidance advises using subagents for independent workstreams and parallel or context-isolated tasks, while handling simpler tasks directly. Anthropic also notes that excessive subagent use creates unnecessary cost and latency. Option A generates multiple summaries speculatively, consuming resources even if they are never requested and creating cache-invalidation complexity whenever findings change. Option B may reduce repeated input- token cost, but it does not eliminate subagent startup, message processing, or the unnecessary orchestration round trip. Option D introduces an iterative request protocol that will likely increase latency further. Direct coordinator summarization uses information already available in active context and therefore provides the smallest architectural change, lowest token-transfer overhead, and fastest response while preserving subagent synthesis for genuinely complex comparative or cross-source analysis.

NEW QUESTION: 39

You are building a customer support resolution agent using the Claude Agent SDK. The agent handles high- ambiguity requests like returns, billing disputes, and account issues. It has access to your backend systems through custom Model Context Protocol (MCP) tools (get_customer , lookup_order , process_refund , escalate_to_human). Your target is 80% + first-contact resolution while knowing when to escalate.

When the agent calls lookup_order and receives order details showing the item was purchased 45 days ago, how does the agentic loop determine whether to call process_refund or escalate_to_human next?

A. The order details are added to the conversation and the model reasons about which action to take.
B. The orchestration layer automatically routes to the next tool based on the order's status field.
C. The agent follows a pre-configured decision tree mapping order attributes to specific tool calls.
D. The agent executes the remaining steps in a tool sequence planned at the start of the request.
Answer: (SHOW ANSWER)
In the standard Claude tool-use loop, the application executes lookup_order and sends its output back as a tool_result . That result becomes part of the conversation state available to Claude. Claude then evaluates the purchase date, refund policy, customer request, authorization constraints, and available tools before selecting the next action.
Anthropic describes client-tool orchestration as a repeated loop: Claude emits a tool_use request, the application executes it, returns a tool_result , and Claude continues reasoning from the updated conversation.
Claude, rather than the tool implementation, selects when and how to invoke the next available tool unless the application has explicitly implemented a fixed workflow.
(https://platform.claude.com/docs/en/agents-and- tools/tool-use/how-tool-use-works)
Options B and C describe possible custom orchestration architectures, but neither is stated in the scenario.
Option D is inconsistent with adaptive agent behavior because later actions depend on information that did not exist before lookup_order completed. A rigid sequence would not respond appropriately to different purchase dates, eligibility states, or order conditions.
The tool result should return high-signal fields such as purchase date, return-window status, refund eligibility, existing refund status, and stable order identifiers so Claude can make the subsequent decision accurately.
Official references/topics: Tool-result continuation, adaptive agent loops, model-directed tool selection, sequential dependency handling.

NEW QUESTION: 40
You are building a customer support resolution agent using the Claude Agent SDK. The agent handles high- ambiguity requests like returns, billing disputes, and account issues. It has access to your backend systems through custom Model Context Protocol (MCP) tools (get_customer , lookup_order , process_refund , escalate_to_human). Your target is 80% + first-contact resolution while knowing when to escalate.
A customer raises three separate issues during one session: a refund inquiry (turns 1-15), a subscription question (turns 16-30), and a payment method update (turns 31-45). At turn 48, the customer asks "What happened with my refund?" The conversation is approaching context limits.

What strategy best maintains the agent's ability to address all issues throughout the session?

A. Summarize earlier turns into a narrative description, preserving full message history only for the active issue.

B. Implement sliding window context that retains the most recent 30 turns.

C. Rely on MCP tools to re-fetch relevant information on demand when the customer references earlier issues.

D. Extract and persist structured issue data (order IDs, amounts, statuses) into a separate context layer.

Answer: D (LEAVE A REPLY)

The agent must preserve durable state for several concurrently open issues without keeping every conversational turn in the active context window. Option D creates a structured issue ledger containing identifiers, issue type, requested action, current status, completed steps, unresolved questions, and relevant tool outputs.

Anthropic recommends structured note-taking for long-horizon agents. In this pattern, critical state is written outside the context window and retrieved later, allowing the agent to preserve dependencies and progress across dozens of tool calls. Anthropic also describes compaction as retaining critical details while discarding redundant raw messages and historical tool output. (https://www.anthropic.com/engineering/effective-context-engineering-for-ai-agents) Option A is better than retaining the entire transcript, but a narrative summary can merge separate issues or obscure exact identifiers and statuses. Option B will discard the refund conversation because it occurred outside the most recent 30 turns. Option C may recover current backend facts, but it cannot reconstruct conversational commitments, prior explanations, or why a particular action remains pending.

A structured context layer should maintain one record per issue and be updated after every material action.

The active prompt can include compact summaries of all open issues and expanded details only for the issue currently being discussed.

Official references/topics: Structured note-taking, external agent memory, context compaction, multi-issue state management.

NEW QUESTION: 41

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 pipeline uses a tool called extract_metadata with a JSON schema for paper details. You've also defined lookup_citations and verify_doi tools for enrichment. During testing, you notice that when users include requests like "extract the metadata and tell me how

cited it is," Claude sometimes calls lookup_citations first, which fails because it needs the DOI that extract_metadata would provide.
What's the most effective way to ensure structured metadata extraction happens first?
A. Set tool_choice to {"type": "tool", "name": "extract_metadata"} and process the enrichment requests in subsequent turns after receiving the extracted metadata.
B. Set tool_choice to "auto" and reorder the tool definitions so extract_metadata appears first in the tools array, since Claude prioritizes earlier-listed tools.
C. Set tool_choice to {"type": "tool", "name": "extract_metadata"} for every API call in the pipeline, ensuring Claude always extracts metadata before any enrichment can occur.
D. Set tool_choice to "any" so Claude must use a tool, combined with system prompt instructions prioritizing extract_metadata .
Answer: (SHOW ANSWER)
The dependency must be enforced by orchestration rather than left to probabilistic tool selection. Anthropic documents that tool_choice: {"type": "tool", "name": "..."} forces Claude to invoke the specified tool. By contrast, auto allows Claude to decide whether and which tool to call, while any requires some tool but does not force a particular one.
(https://platform.claude.com/docs/en/agents-and-tools/tool-use/define-tools) Option A therefore establishes a deterministic two-stage workflow. The first API turn forces extract_metadata
, producing the DOI and other structured paper details. The application validates and stores that result. A subsequent turn then exposes or permits verify_doi and lookup_citations , passing the extracted DOI as explicit state. This design converts an implicit tool dependency into an application-controlled execution graph.
Option B is incorrect because array order is not a documented precedence mechanism and cannot guarantee selection. Option C forces extract_metadata on every call, including turns where enrichment should occur, potentially creating an infinite or non-progressing workflow. Option D guarantees only that one available tool is called; Claude could still select lookup_citations before the DOI exists.
For stronger input integrity, the tools can also use strict schemas so their arguments conform to the declared JSON Schema. The sequencing requirement, however, remains the responsibility of the orchestration layer.
Official references/topics: Tool Choice; Forced Tool Invocation; Multi-Turn Tool Orchestration; Tool Dependency Management.

NEW QUESTION: 42
You are building a multi-agent research system using the Claude Agent SDK. A coordinator agent delegates to specialized subagents: one searches the web, one analyzes documents, one synthesizes findings, and one generates reports. The system researches topics and produces comprehensive, cited reports.
You have configured the system so that all four subagents have access to the complete set of 18 tools. During testing, agents frequently call tools outside their specialization-the

synthesis agent attempts web searches, and the report generator tries to analyze documents.
What is the primary cause of this poor tool-selection behavior?
A. The tool definitions consume too much context-window space, leaving insufficient room for task content.
B. Choosing from 18 tools instead of four or five relevant tools increases decision complexity beyond reliable selection thresholds.
C. The agents' role descriptions in their system prompts conflict with having access to tools outside those roles.
D. The coordinator cannot track which capabilities each subagent has, leading to misrouted tasks.
Answer: C (LEAVE A REPLY)
Option C identifies the architectural contradiction. A specialized subagent's role prompt says what work it should perform, but granting it every tool simultaneously signals that unrelated actions are available and potentially legitimate. The synthesis agent can therefore interpret web search as a permissible way to fill gaps, while the report generator may invoke document-analysis tools even though those responsibilities belong elsewhere. Anthropic's Agent SDK documentation describes subagents as focused workers with tailored instructions and explicitly supports restricting each subagent to specific tools; it notes that tool restrictions reduce unintended actions. There is no documented universal threshold at which choosing among 18 tools becomes unreliable, so option B invents a "four or five tools" limit. Context consumption in option A may affect efficiency, but it does not directly explain the role-specific cross-calling pattern. Option D is also incorrect because the observed behavior occurs after delegation, inside agents that already possess the tools.
The remedy is least-capability configuration: give the web agent search tools, the document agent document tools, the synthesis agent only the inputs and utilities required to combine findings, and the report agent formatting or file-output tools. Aligning role, description, and tool allowlist removes the conflicting affordances.

NEW QUESTION: 43
The synthesis agent receives summarized findings from the web-search and document-analysis agents, then passes a consolidated summary to the report generator. During testing, you discover that the generated reports make factual claims without proper citations-the report generator cannot attribute statements to their original sources because that metadata was lost during the summarization steps. What is the most effective approach to ensure proper source attribution in the final reports?
A. Have the report generator query the web-search agent to relocate sources for claims in the final report.
B. Have each agent output structured data that separates content summaries from source metadata, including URLs, document names, and page numbers.

C. Skip summarization and pass the complete raw outputs from the web-search and document-analysis agents directly to the report generator.
D. Instruct the synthesis agent to embed source references inline within its summary text using a consistent citation format.
Answer: B (LEAVE A REPLY)
Option B preserves provenance as structured data throughout every handoff. Each finding should include the summarized claim and an associated source record containing its URL or document identifier, page or section location, relevant excerpt, and possibly retrieval date and confidence. The synthesis agent can combine claims without destroying this relationship, and the report generator can produce citations deterministically.
Anthropic's multi-agent research system retains findings until a dedicated citation stage identifies the precise source locations supporting report claims. This demonstrates that citation provenance must remain available beyond initial retrieval and summarization.
Option A attempts to reconstruct provenance after it has been discarded, which is expensive and may associate a claim with a different or weaker source. Option C preserves evidence but wastes context by transferring approximately all raw search and document content. Option D is better than losing references entirely, but embedding citations only in prose makes programmatic validation, deduplication, and claim-to- source mapping less reliable. Structured content and source fields maintain both context efficiency and auditable attribution while allowing different agents to transform the narrative safely.

NEW QUESTION: 44
During testing, when a customer says, "I need a refund for my recent purchase," the agent immediately invokes process_refund but populates the required order_id parameter with a plausible-looking fabricated value instead of first calling lookup_order. The refund fails because the invented order identifier does not exist. Which change directly addresses the root cause of the fabricated order_id?
A. Update the process_refund tool description to state explicitly that order_id must come from a successful lookup_order result and must never be assumed, inferred, or invented.
B. Change tool_choice from auto to any so Claude must call a tool on every turn.
C. Add server-side validation that checks whether order_id exists before attempting the refund and returns an error when it does not.
D. Preprocess customer messages to extract any mentioned order identifiers and inject them into the conversation before sending the request to Claude.
Answer: (SHOW ANSWER)
Option A corrects the missing tool contract that permits Claude to guess a required parameter. The process_refund description should state its prerequisite explicitly: invoke lookup_order, use only the exact order_id returned by that successful call, and ask for additional identifying information when lookup cannot resolve an order. An input example can demonstrate the correct two-tool sequence.

Anthropic's tool-definition guidance identifies detailed descriptions as the most important factor in tool performance. Descriptions should explain when a tool should and should not be used, what every parameter means, and all important limitations. The tool-use overview confirms that Claude selects and populates tools based on the request and their descriptions.
Option B forces some tool call but does not tell Claude which tool must run first or where the identifier originates. Option C is an essential defence-in-depth control, but it detects the fabricated value after Claude has already made the incorrect decision. Option D works only when the customer supplies an identifier and does not cover "recent purchase." Explicit prerequisite and provenance instructions directly prevent the model from inventing the required argument.

NEW QUESTION: 45
When implementing your lookup_order MCP tool, the backend sometimes returns errors-for example,
"Order not found" or temporary database failures. What is the correct pattern for communicating these errors back to the agent?
A. Return the error message in the tool-result content with the isError flag set to true.
B. Return a successful response with a status field indicating the error type.
C. Log the error server-side and return an empty result to avoid confusing the model.
D. Throw an exception from the tool handler so the agent framework can catch and log it.
Answer: A (LEAVE A REPLY)
Option A follows the MCP error contract. An error originating during execution of a valid tool call-such as an API failure, input-validation problem, or business-logic failure-should be returned inside the tool result with isError: true. The content should explain what happened and, where possible, provide actionable recovery guidance.
The official MCP tools specification distinguishes tool-execution errors from protocol-level errors and specifies that execution failures are reported through tool results using isError: true. Anthropic's tool-using agent tutorial provides the equivalent Claude API pattern using is_error: true, allowing Claude to retry with corrected input, request clarification, or explain the limitation.
Option B falsely represents a failed operation as successful, weakening reliable failure detection. Option C deprives the agent of the information required to determine an appropriate recovery action. Option D may be converted into a tool error by some SDK runners, but uncaught exceptions are not the MCP-level response contract. The strongest implementation returns a concise, sanitized, actionable error result and separately records detailed diagnostics in server logs.

NEW QUESTION: 46
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?
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 (LEAVE A REPLY)
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.

Valid CCAR-F Dumps shared by BraindumpsPass.com for Helping Passing CCAR-F Exam! BraindumpsPass.com now offer the **newest CCAR-F exam dumps**, the BraindumpsPass.com CCAR-F exam **questions have been updated** and **answers have been corrected** get the **newest** BraindumpsPass.com CCAR-F dumps with Test Engine here: https://www.braindumpspass.com/Anthropic/CCAR-F-practice-exam-dumps.html (**191** Q&As Dumps, **40%OFF** **Special Discount:** **Exam-Tests**)

NEW QUESTION: 47
Your automated review calls the Claude API for each pull request, using tool_use with a report_findings tool that returns a JSON array of finding objects. Each object contains file_path, line_number, severity, category, and description. During testing on a large pull request touching more than 30 files, the response reaches the max_tokens limit and is truncated in the middle of the JSON, causing your pipeline's parser to fail. What is the most effective way to handle this?
A. Split the review into multiple API calls that each analyze a subset of the changed files, and then merge the resulting findings arrays.
B. Increase max_tokens to the model's maximum and instruct Claude to keep each finding description under 50 words.
C. Switch from tool_use to prompting Claude to return findings as a Markdown list.
D. Add retry logic that detects truncated JSON and resends the request with instructions to report only critical- and high-severity findings.
Answer: (SHOW ANSWER)
Option A reduces the maximum output required from any single response while preserving the structured schema and complete severity range. The pipeline can partition files into coherent groups, execute bounded reviews, validate each returned array, and merge and deduplicate findings using stable fields such as file path, line number, category, and description.
Anthropic's stop-reason documentation confirms that max_tokens means generation reached the configured output limit and the response must be treated as incomplete. Structured output constraints can guarantee schema-valid generation when completion succeeds, but they cannot create unlimited output capacity. A large findings array can still exceed the available token budget.
Option B may postpone the failure but provides no durable guarantee for still-larger pull requests, and aggressively shortening descriptions may eliminate necessary evidence. Option C abandons machine- validated structure without reducing the amount of generated content. Option D deliberately suppresses medium- or low-severity findings and repeats an oversized request rather than addressing its scope.
Partitioning establishes predictable output bounds, supports targeted retries, retains every required finding category, and prevents a single truncated response from invalidating the complete review.

NEW QUESTION: 48
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 needs to extract candidate information (name, contact details, skills, work experience, education) from uploaded resumes. The extracted data must strictly conform to a predefined JSON schema, as missing required fields or incorrect data types will cause downstream validation failures.
What is the most reliable approach to ensure Claude's output consistently matches the schema?
A. Parse Claude's text response with regex patterns to extract JSON objects, using retry logic for malformed responses.
B. Include detailed JSON formatting instructions and a template example in the system prompt, asking Claude to output only valid JSON.
C. Make two separate API calls-first extracting information as text, then asking Claude to format that text as JSON.
D. Define a tool with an input schema matching your required JSON structure and extract the data from Claude's tool_use response.
Answer: D (LEAVE A REPLY)
A schema-defined tool provides a structured interface between Claude and the application. The tool's input_schema can declare required properties, nested work-experience and education objects, arrays of skills, and exact data types. Claude then supplies the extracted resume information as tool-call arguments, which the application can read directly from the tool_use response.
Anthropic's tool documentation specifies that a custom tool definition includes a JSON Schema object describing its expected parameters. Current strict tool use can additionally enforce that generated tool arguments match the declared schema exactly. This removes the need to locate JSON inside free-form text or depend on post-generation repair.
(https://docs.anthropic.com/en/docs/agents-and-tools/tool-use/implement- tool-use)
Option A is fragile because regular expressions do not provide a robust parser for arbitrary nested JSON and cannot correct missing properties or incorrect types. Option B improves formatting behavior but remains probabilistic. Option C doubles latency and creates two opportunities for information loss: first during extraction and again during reformatting.
In a production design, fields that may genuinely be absent from a resume should be optional or nullable rather than forcing invented values. The tool schema should also use strict: true where supported. Among the available choices, D provides the strongest structural guarantee and the cleanest downstream integration.
Official references/topics: Tool Use Responses, Input Schemas, Strict Tool Use, Nested Structured Extraction.

NEW QUESTION: 49
You are building a structured data extraction system using Claude. The system extracts information from unstructured documents, validates the output using JSON schemas, and maintains high accuracy. It must handle edge cases gracefully and integrate with downstream systems.
After your daily batch of 10,000 documents completes, 300 documents (3%) fail with context_length_exceeded errors. The results file identifies each failure by custom_id.
What is the most cost-effective approach to process these failures?
A. Resubmit the entire 10,000-document batch using a model tier with a larger context window.
B. Reprocess the entire batch with prompt caching enabled to reduce the cost of retrying requests with identical system prompts.
C. Increase the max_tokens parameter for the 300 failed documents and resubmit them in a new batch.
D. Resubmit only the 300 failed documents after chunking them into smaller pieces, and then combine the partial extractions.
Answer: D (LEAVE A REPLY)
Option D fixes the actual failure while avoiding needless reprocessing. Anthropic's Message Batches API treats every request independently, records the result against its unique custom_id, and explicitly states that one failed request does not affect the others. The 9,700 successful documents therefore require no retry. The failed requests exceeded the available context; increasing max_tokens changes the permitted output budget, not the size of the input context, so option C does not correct the cause. Resubmitting all 10,000 requests would repeat paid work, and prompt caching would only reduce some repeated-input cost without repairing oversized requests. Chunking each failed document reduces the input presented to Claude so every replacement request fits within the model's context window . The application can then merge the extracted partial structures deterministically, retaining the original document identifier and chunk ordering. Anthropic recommends retry logic for failed batch requests and using custom_id to match results because batch results can arrive out of order. Consequently, selective retry plus chunking is both the technically correct recovery path and the lowest-cost option.

NEW QUESTION: 50
You are building a multi-agent research system using the Claude Agent SDK. A coordinator agent delegates to specialized subagents: one searches the web, one analyzes documents, one synthesizes findings, and one generates reports. The system researches topics and produces comprehensive, cited reports.
Production reviews reveal inconsistent handling of uncertainty in final reports. Sometimes conflicting subagent findings are synthesized into a single confident statement, losing important nuance, while other reports use excessive qualifications and become unhelpful.

The web-search agent returns, "Industry analysts estimate a $50 billion market size, although methodologies vary." The document-analysis agent returns, "A peer-reviewed study estimates $35 billion, with a ±$7 billion 95% confidence interval." The coordinator either selects one estimate arbitrarily or produces a vague $35-$50 billion range.
What systematic approach best addresses this?
A. Implement a confidence-calibration layer that normalizes subagent uncertainty expressions to probability scores between 0.0 and 1.0, and then calculate a confidence-weighted average.
B. Configure subagents to report only findings meeting a high-confidence threshold, filtering uncertain information before it reaches the coordinator.
C. Add a verification subagent that passes claims to synthesis only when they are corroborated by at least two independent sources.
D. Instruct the synthesis agent to distinguish well-established findings from contested findings explicitly, preserving each source's original uncertainty, methodology, and supporting evidence.
Answer: (SHOW ANSWER)
Option D preserves the evidence instead of manufacturing certainty. The two estimates are not directly interchangeable: one is an industry estimate with unspecified methodology, while the other is a peer-reviewed estimate with an explicit confidence interval. Converting both into model-generated confidence scores and calculating a weighted average would create a new figure that neither source reported and that may have no statistical validity. Anthropic's hallucination-reduction guidance recommends making claims auditable through quotations, citations, and supporting evidence rather than presenting unsupported synthesis as fact. Its Citations documentation similarly emphasizes retaining the exact source passages supporting individual claims. Filtering uncertain findings, option B, would remove decision-relevant information. Requiring two- source corroboration, option C, could also discard credible evidence concerning emerging or specialized subjects. The synthesis agent should report the estimates separately, explain their methodological differences, identify which findings are strongly supported or disputed, and state what evidence would resolve the disagreement. This produces calibrated, useful reporting without arbitrary selection, excessive hedging, or false precision.

NEW QUESTION: 51
You are building a customer support resolution agent using the Claude Agent SDK. The agent handles high- ambiguity requests like returns, billing disputes, and account issues. It has access to your backend systems through custom Model Context Protocol (MCP) tools (get_customer, lookup_order, process_refund, escalate_to_human). Your target is 80%+ first-contact resolution while knowing when to escalate.
A customer returns 4 hours after their initial session about the same billing dispute. The previous 32-turn session contains lookup_order results showing "Status: PENDING, Expected resolution: 24-48 hours." In testing, you observe that when resuming sessions

with stale tool results, the agent often references the outdated data in responses (e.g., "I see your refund is still being processed") even after subsequent fresh tool calls return different information.
What approach most reliably handles returning customers?
A. Resume with full history and configure the agent to automatically re-call all previously used tools at session start to ensure data freshness.
B. Resume with full history and add a system prompt instruction telling the agent to always prefer the most recent tool results when multiple calls to the same tool exist in context.
C. Resume with full history but filter out previous tool_result messages before resuming, keeping only the human/assistant turns so the agent must re-fetch needed data.
D. Start a new session, inject a structured summary of the previous interaction (issue type, actions taken, resolution status), then make fresh tool calls before engaging.
Answer: D (LEAVE A REPLY)
Option D separates durable case history from volatile operational data. The new session receives a compact, structured summary describing the billing dispute, the customer's objective, actions previously taken, and the unresolved status. It does not inherit outdated backend observations as though they were still authoritative.
Fresh tool calls then retrieve the current refund or order state before the agent responds.
Agent SDK sessions preserve conversation history, including earlier tool calls and tool results. Resuming the complete transcript therefore reintroduces stale system data into the active context, even though the external backend may have changed substantially during the four-hour gap. Conversation persistence must not be confused with persistence of external-system truth.
Option A performs unnecessary calls to every previously used tool, including tools unrelated to the returning customer's current question. Option B relies on prompt compliance while retaining contradictory historical evidence in context. Option C manually removes tool results from an existing transcript and may damage the logical relationship between prior tool_use and tool_result blocks while still retaining a long, unstructured conversation.
A structured summary should preserve stable identifiers, previous actions, customer commitments, and unresolved issues. Time-sensitive fields such as refund status, delivery state, account balance, or expected resolution should always be refreshed through authoritative tools.
Official references/topics: Session persistence, stale tool-result management, context compaction, fresh-data retrieval.

NEW QUESTION: 52
You are building a customer support resolution agent using the Claude Agent SDK. The agent handles high- ambiguity requests like returns, billing disputes, and account issues. It has access to your backend systems through custom Model Context Protocol (MCP) tools

(get_customer, lookup_order, process_refund, escalate_to_human). Your target is 80%+ first-contact resolution while knowing when to escalate.
Production logs show that when the agent handles complex billing disputes requiring 6+ tool calls, it sometimes exhausts its max_turns limit after gathering data but before completing resolution or escalating.
The team's goal is to guarantee that every customer interaction ends with either a completed resolution or a human handoff, regardless of how the agent loop terminates.
Which approach achieves this guarantee?
A. Implement a pre-tool-use hook that counts tool invocations and terminates the loop with an automatic escalation once the agent reaches 80% of its max_turns limit.
B. Split the workflow into two sequential agent invocations-a first agent gathers information via get_customer and lookup_order, then a second agent receives that data and handles process_refund or escalate_to_human, each with separate turn budgets.
C. Add orchestration-layer code that checks the agent's outcome after each loop termination-if the loop ended without a completed resolution or escalation, programmatically call escalate_to_human with the accumulated conversation context and tool results.
D. Add system prompt instructions telling the agent to call escalate_to_human with a summary of its findings whenever it determines it cannot complete resolution within its remaining actions.
Answer: (SHOW ANSWER)
Option C is the only approach that guarantees the required terminal condition independently of the model's behavior. When the agent loop stops, the orchestration layer evaluates the recorded outcome. If neither a successful resolution nor a confirmed human escalation occurred, deterministic application code invokes escalate_to_human using the information accumulated before termination.
The max_turns limit is a host-controlled safety boundary. Once that boundary is reached, the model may no longer have an opportunity to follow a prompt instruction or make another tool call. Therefore, the required fallback cannot depend entirely on Claude deciding to escalate before its remaining turns are exhausted.
Option A escalates based on an arbitrary 80% threshold and may prematurely terminate cases that could have been resolved successfully within the remaining budget. Option B provides additional turn capacity but does not guarantee that the second agent will complete its work or escalate before its own limit is reached. Option D remains probabilistic and cannot operate after an unexpected termination has already occurred.
The orchestration fallback should be idempotent, ensuring that repeated completion checks cannot create duplicate escalation records. It should also include verified customer information, relevant order data, completed actions, failure reasons, and the unresolved request in the handoff payload.
Official references/topics: Agent-loop termination, deterministic fallback orchestration, terminal-state validation, human handoff guarantees.

NEW QUESTION: 53
The coordinator provides detailed step-by-step instructions to the web-search subagent, specifying exact search queries, source priorities, and date filters. Production monitoring reveals three issues: (1) the subagent reports "insufficient results" instead of trying alternative approaches when the specified searches fail, (2) research quality drops for emerging topics that do not match expected patterns, and (3) the subagent rarely surfaces valuable tangential sources. What is the most effective way to improve subagent adaptability?
A. Specify research objectives and quality criteria-such as coverage breadth, source diversity, and recency-rather than prescribing procedural steps, allowing the subagent to determine its search strategy.
B. Remove procedural details entirely and delegate using simple goals such as "research this topic thoroughly," relying on the subagent's general capabilities.
C. Add fallback directives requiring alternative query formulations whenever the specified searches produce fewer than a predetermined number of results.
D. Classify each topic as either "well-defined" or "exploratory" and use a different instruction style for each category.
Answer: A (LEAVE A REPLY)
Option A establishes a clear destination without forcing the subagent to follow a brittle route. The prompt should define the research question, boundaries, expected deliverable, acceptable source types, recency requirements, coverage expectations, and verification criteria. The subagent can then broaden, narrow, reformulate, or follow emerging leads according to observed search results.
Anthropic's research-agent guidance emphasizes giving each subagent an objective, output format, source guidance, and clear task boundaries. It also describes research as dynamic and path-dependent, requiring agents to adapt their strategy as new information emerges.
Option B removes too much structure and may cause duplicated work, incomplete coverage, or inconsistent output. Option C handles only one known failure condition and still restricts the agent to a predetermined procedure. Option D adds classification complexity and creates an artificial boundary between topic types that may change during investigation. Objective-based delegation retains accountability because results can be evaluated against explicit quality criteria, while preserving the agent's ability to discover alternative terminology, unexpected sources, and useful adjacent evidence. This is controlled autonomy rather than either procedural micromanagement or unrestricted exploration.

Valid CCAR-F Dumps shared by BraindumpsPass.com for Helping Passing CCAR-F Exam! BraindumpsPass.com now offer the **newest CCAR-F exam dumps**, the BraindumpsPass.com CCAR-F exam **questions have been updated** and **answers have been corrected** get the **newest** BraindumpsPass.com CCAR-F dumps with Test Engine here: https://www.braindumpspass.com/Anthropic/CCAR-F-practice-exam-dumps.html (**191** Q&As Dumps, **40%OFF** **Special Discount:** **Exam-Tests**)