Domain 2 — 18%

Tool Design & MCP Integration

Domain 2 — 18% of the CCA exam

IN THIS MODULE:

- Writing effective tool descriptions
- Structured error responses with isError and errorCategory
- tool_choice modes: auto, any, forced selection
- MCP configuration: project vs user level
- Built-in tools: Read, Write, Edit, Grep, Glob, Bash

90 PRACTICE QUESTIONS — 30 BEGINNER · 30 INTERMEDIATE · 30 ADVANCED

QUESTION BANK OVERVIEW

Level	Questions	Focus
Beginner	Q1–Q30	Core concepts, terminology, and foundational patterns
Intermediate	Q31–Q60	Application scenarios, design decisions, trade-offs
Advanced	Q61–Q90	Production architecture, edge cases, system design

HOW TO USE THIS MODULE

Work through each level in order. For Beginner questions, aim for 90%+ before moving to Intermediate. For Intermediate, 80%+ before Advanced. Each question includes the correct answer and a full explanation — read the explanation even for questions you answered correctly to understand the underlying principle. The exam tests judgment, not memorization.

BEGINNER QUESTIONS

Questions 1–30

Q1. Why are tool descriptions critical for reliable tool selection?

A) Tool descriptions are documentation for developers — Claude doesn't read them

B) Tool descriptions are the primary signal Claude uses to decide which tool to call; vague descriptions cause misrouting

C) Longer descriptions always produce better tool selection

D) Claude selects tools based on their position in the tools array, not descriptions

■ Correct Answer: B

Claude reads tool descriptions to understand what each tool does and when to use it. Minimal or overlapping descriptions cause Claude to make incorrect selections. Every description must clearly state purpose, inputs, outputs, and scope boundaries.

Q2. What information should a well-written tool description include?

A) Only the tool name and return type

B) Purpose, expected input formats, example queries, edge cases, output format, when to use it, and when NOT to use it vs. similar tools

C) The implementation details of the tool function

D) Only a one-sentence summary — brevity is best for tool descriptions

■ Correct Answer: B

A complete tool description prevents all common misrouting scenarios: clear purpose prevents confusion with similar tools, input format guidance prevents invalid calls, negative examples ('do NOT use for X') set explicit boundaries.

Q3. What does the isError flag pattern in MCP tool responses accomplish?

A) It marks tool responses as errors so Claude knows to stop the agentic loop

B) It communicates structured failure information back to the agent so it can make intelligent recovery decisions

C) It triggers automatic retry logic in the Agent SDK

D) It logs the error to Anthropic's monitoring system

■ Correct Answer: B

isError: true in a tool result signals to Claude that the tool failed, not that the task is impossible. Combined with errorCategory, isRetryable, and a human-readable message, it enables Claude to choose the right recovery: retry, escalate, or try an alternative approach.

Q4. What are the four main error categories to distinguish in MCP tool error responses?

A) low, medium, high, critical

B) transient, validation, business_rule, permission

C) network, timeout, api, database

D) warning, error, fatal, unknown

■ Correct Answer: B

Transient (retry-able: timeouts, outages), validation (bad input: fix the request), business_rule (policy violation: different action needed), permission (access denied: no retry needed). Each category implies a different recovery strategy for Claude.

Q5. What does tool_choice='auto' mean?

A) Claude must call a tool — any tool in the list

B) Claude decides whether to call a tool or respond with plain text

C) Claude selects a tool automatically based on input format

D) Tool selection is determined by the Agent SDK, not Claude

■ Correct Answer: B

'auto' is the default — Claude reasons about whether calling a tool or responding with text is more appropriate. Use 'any' when you need to guarantee a tool is called, or force a specific tool with {'type':'tool','name':'...'} for prerequisites.

Q6. What is the difference between project-level and user-level MCP server configuration?

A) Project-level supports more tools; user-level has a tool limit of 10

B) Project-level (.mcp.json) is committed to version control and shared with the team; user-level (~/.claude.json) is personal and not shared

C) Project-level requires admin approval; user-level is self-service

D) They are identical in behavior — the difference is only naming convention

■ Correct Answer: B

Scope determines sharing: .mcp.json in the repo root is checked into git → every developer who clones the repo gets the same MCP tools. ~/.claude.json is personal → only you see those servers. Use project-level for team tools, user-level for experiments.

Q7. Why should you use environment variable expansion in .mcp.json instead of hardcoding tokens?

A) Environment variables load faster than hardcoded strings

B) Hardcoding API tokens in .mcp.json commits secrets to version control — env var expansion keeps secrets in the environment, not the repo

C) .mcp.json does not support string values — only env var references work

D) Environment variables enable automatic token rotation

■ Correct Answer: B

${TOKEN_NAME} in .mcp.json expands to the actual value at runtime from the developer's environment. The file itself contains no secret — it's safe to commit. Hardcoded tokens in version control create serious security vulnerabilities.

Q8. What is the Grep built-in tool best used for?

A) Finding files by name or extension pattern (e.g., **/*.test.ts)

B) Searching file contents for patterns like function names, error messages, or import statements

C) Reading entire file contents

D) Making targeted edits to specific lines in a file

■ Correct Answer: B

Grep is for content search — 'find all files that import from this module', 'find where this function is called'. Glob is for file name pattern matching (finding files by name/extension). Read/Write/Edit are for file content operations.

Q9. What is the Glob built-in tool best used for?

A) Searching file contents for a specific string or regex pattern

B) Finding files by name or path pattern (e.g., **/*.test.tsx, src/**/index.ts)

C) Reading all files in a directory recursively

D) Watching files for changes

■ Correct Answer: B

Glob matches file paths by pattern — use it to find all test files, all TypeScript files, or all index files in a specific directory structure. It returns paths, not content. Grep searches content within files.

Q10. When should you use the Edit tool instead of Read + Write?

A) Always use Edit — it is faster than Read + Write

B) Use Edit for targeted modifications when the text to replace is unique in the file; fall back to Read + Write when Edit cannot find a unique anchor

C) Use Edit for new file creation; Read + Write for modifications

D) Edit is for binary files; Read + Write is for text files

■ Correct Answer: B

Edit requires unique text matching — it finds a specific string and replaces it. If the target text appears multiple times, Edit may fail. Read the full file, modify in memory, Write back when Edit cannot identify a unique anchor point.

Q11. What does 'scoped tool access' mean in multi-agent design?

A) Limiting which users can invoke specific tools

B) Giving each agent only the tools relevant to its specific role, preventing cross-specialization misuse

C) Rate limiting tool calls to prevent API overuse

D) Encrypting tool inputs and outputs for security

■ Correct Answer: B

Scoped access: a synthesis agent gets only summarize and verify tools — not search or file-write tools. This prevents misuse (synthesis agent searching instead of synthesizing) and improves tool selection reliability within each agent.

Q12. What happens when you expose an MCP resource instead of requiring tool calls for data access?

A) Resources are more expensive than tool calls

B) Resources give the agent visibility into available data catalogs without requiring exploratory tool calls — reducing unnecessary round-trips

C) Resources can only contain text, not structured data

D) Resources are only available in project-scoped configurations

■ Correct Answer: B

MCP resources expose content catalogs (issue summaries, doc hierarchies, schemas) that the agent can browse without calling tools. Instead of calling list_issues() to discover what's available, the agent sees a resource with all issue summaries already available.

Q13. What is the recommended approach for standard integrations like Jira or GitHub in MCP?

A) Always build custom MCP servers for full control

B) Use existing community MCP servers; reserve custom server development for team-specific, non-standard workflows

C) Avoid MCP entirely for third-party integrations — use REST APIs directly

D) Build a universal MCP server that handles all third-party integrations

■ Correct Answer: B

The MCP ecosystem has mature community servers for common tools (GitHub, Jira, Slack, databases). Using them avoids reinventing the wheel. Build custom servers only for proprietary internal systems or workflows with unique requirements.

Q14. Why does a generic 'Operation failed' error response prevent intelligent recovery?

A) Claude cannot parse generic strings — only JSON is readable

B) Without error category, retryability flag, and context, Claude cannot distinguish between 'retry in 5 seconds' and 'this will never work — escalate instead'

C) Generic errors trigger the Agent SDK's automatic error handler

D) The isError flag is not set in generic errors, so Claude doesn't see them as failures

■ Correct Answer: B

Recovery intelligence requires: what type of error (transient/permanent?), is retry appropriate (isRetryable?), and what context helps recovery. 'Operation failed' provides none of this — Claude must guess, often poorly.

Q15. What is tool_choice='any' used for?

A) Letting Claude choose the best tool from multiple options

B) Guaranteeing that Claude calls a tool (any tool in the list) rather than responding with plain text

C) Selecting all tools in parallel

D) Disabling tool use for the current turn

■ Correct Answer: B

'any' is used when you need structured output and have multiple possible extraction schemas — you don't know which document type will be submitted, but you NEED a tool call (not a text response). Claude will call whichever tool it deems appropriate.

Q16. What does providing input/output examples in tool descriptions accomplish?

A) Examples increase the token count unnecessarily

B) Examples demonstrate expected behavior for edge cases and help Claude understand format expectations, reducing misuse

C) Examples override the input_schema — only one is needed

D) Examples are only useful for debugging, not production

■ Correct Answer: B

Examples in descriptions bridge the gap between abstract specification and practical behavior. 'Input: CUST-123456 → Output: {id, name, email, status}' is clearer than prose alone. Edge case examples (null handling, format variations) are especially valuable.

Q17. What does the tool_use_id field in a tool_result block do?

A) It identifies which API call generated the tool request

B) It links the tool result back to the specific tool_use block in Claude's response, enabling correct routing when multiple tools are called

C) It is a unique identifier for audit logging purposes

D) It specifies which Claude model processed the tool call

■ Correct Answer: B

When Claude requests multiple tools in one turn, each tool_use block has a unique id. Your tool_result must include the matching tool_use_id so Claude knows which result corresponds to which tool call. Mismatched IDs cause undefined behavior.

Q18. What makes a tool description 'boundary-explicit'?

A) The description is exactly 100 words

B) The description clearly states both what the tool DOES handle and what it does NOT handle, preventing misuse on out-of-scope inputs

C) The description includes a JSON schema with strict validation rules

D) The description is formatted with bullet points and headers

■ Correct Answer: B

'Do NOT use for: local files or database records' is a boundary statement. These negative boundaries are as important as positive capabilities — they prevent Claude from calling a web-scraping tool on a file path or a database tool on a URL.

Q19. When should you enhance an MCP tool's description beyond the default?

A) Never — default descriptions from community servers are always sufficient

B) When the default description causes Claude to prefer built-in tools (like Grep) over the more capable MCP tool for the use case

C) Only when the tool is newly created

D) When the tool is used by more than 3 agents

■ Correct Answer: B

Community MCP servers often have generic descriptions. If Claude consistently reaches for the Grep built-in instead of your powerful code-search MCP tool, enhance the MCP tool's description to explicitly explain its advantages over Grep for this context.

Q20. What problem does splitting a generic 'analyze_document' tool into 'extract_data_points', 'summarize_content', and 'verify_claim_against_source' solve?

A) It reduces API latency by distributing work across smaller tools

B) It eliminates tool selection ambiguity by giving each tool a specific, unambiguous purpose with clear input/output contracts

C) It allows three agents to call different tools simultaneously

D) It reduces the total number of tokens in the tool definitions

■ Correct Answer: B

A generic analyze_document tool forces Claude to guess which type of analysis is needed. Three purpose-specific tools with clear names and descriptions make the right choice obvious from context: need to extract structured data → extract_data_points; need a summary → summarize_content.

Q21. Why should tool names use verbs that describe the action rather than nouns that describe the data?

A) Verb-based names are shorter and save tokens in the tool definition

B) Verb names communicate what the tool DOES (get_customer, update_order, delete_record) enabling immediate selection clarity; noun names (customer, order, record) don't indicate the action type

C) Anthropic's API requires verb-prefixed tool names

D) Verb names enable automatic tool discovery without descriptions

■ Correct Answer: B

Action clarity: get_customer vs update_customer vs delete_customer are immediately distinguishable — the verb makes the operation unambiguous. 'customer' as a tool name tells you nothing about the operation. Verb-first naming is a best practice that reduces selection ambiguity.

Q22. What is MCP (Model Context Protocol)?

A) A new Claude model version for context-heavy tasks

B) An open standard protocol that enables AI models to connect to external tools and data sources through a standardized interface

C) An API compression technique for reducing token costs

D) Anthropic's internal model communication format

■ Correct Answer: B

MCP is an open standard (developed by Anthropic, now industry-wide) that defines how LLMs connect to external tools, data sources, and services. MCP servers expose tools and resources; MCP clients (like Claude) discover and use them. This standardization means one MCP server works with any MCP-compatible AI.

Q23. What is the difference between an MCP tool and an MCP resource?

A) Tools are read-only; resources can modify data

B) Tools are invocable functions that perform actions or retrieve data on demand; resources are pre-existing content (documents, data catalogs) available for browsing without a function call

C) Resources are faster than tools; tools are more accurate

D) Tools are for internal data; resources are for external APIs

■ Correct Answer: B

MCP tools: invoked by Claude (search, calculate, write). MCP resources: browseable catalogs of available content (list of documents, issue summaries, schema definitions). Resources reduce tool calls by letting Claude see what's available before deciding which tool to use.

Q24. What command starts an MCP server defined in .mcp.json?

A) .mcp.json servers start automatically when Claude Code launches in that directory

B) claude mcp start [server-name]

C) npm run mcp

D) Servers must be started manually before using Claude Code

■ Correct Answer: A

MCP servers configured in .mcp.json are started automatically when Claude Code initializes in the project directory. You don't need to start them manually. The configuration defines the command and arguments Claude Code uses to start each server.

Q25. What does the 'command' field in an MCP server configuration specify?

A) The MCP API command to call when a tool is invoked

B) The executable to run to start the MCP server process (e.g., 'npx', 'node', 'python')

C) The Claude command used to authenticate with the server

D) The shutdown command when Claude Code exits

■ Correct Answer: B

The 'command' field tells Claude Code how to start the MCP server: 'npx -y @modelcontextprotocol/server-github' runs the GitHub MCP server via npx. 'python mcp_server.py' starts a custom Python MCP server. Claude Code launches this process and communicates with it via stdin/stdout.

Q26. You define a tool that accepts a customer_id parameter. Claude passes '12345' (a string) but your tool expects an integer. What should the tool do?

A) Crash with an unhandled exception

B) Return a structured validation error: {isError: true, errorCategory: 'validation', message: 'customer_id must be an integer, received string: 12345', isRetryable: true, fix: 'Pass customer_id as an integer'}

C) Silently convert the string to integer and proceed

D) Return empty results as if no customer was found

■ Correct Answer: B

Validation errors are retryable with the right guidance. The structured error tells Claude exactly what's wrong and how to fix it — it can correct the parameter type and retry. Silent coercion is acceptable for simple conversions but can mask bugs. Crashing produces unhelpful errors.

Q27. What is the 'args' field in an MCP server configuration?

A) The arguments that Claude passes when calling tools on the server

B) The command-line arguments passed to the server's startup command

C) The authentication arguments for the server's API

D) The list of tools the server exposes

■ Correct Answer: B

'args' is the list of CLI arguments for the server startup command. For 'command': 'npx', 'args': ['-y', '@modelcontextprotocol/server-github'] runs: npx -y @modelcontextprotocol/server-github. These are startup arguments, not tool call arguments.

Q28. What does it mean when a tool returns isError: false but data: []?

A) The tool encountered an error but suppressed it

B) The tool succeeded but found no matching records — this is a valid outcome (successful empty query), not an error

C) The tool is deprecated and returning empty responses

D) The response is incomplete and more data will follow

■ Correct Answer: B

Empty results (data: []) with isError: false is a legitimate response. A search for 'customers named XYZ' finding zero matches is a successful query. Only mark isError: true for actual failures (network timeout, permission denied, invalid input). Empty results enable Claude to respond accurately: 'No matching customers found.'

Q29. Which built-in Claude Code tool is best for making targeted one-line edits to a file?

A) Bash (using sed)

B) Write (overwrite the entire file)

C) Edit (replace a specific unique string with new content)

D) Read then Write

■ Correct Answer: C

Edit is the targeted edit tool: it finds a unique string in the file and replaces it. Faster and less error-prone than Read+Write for small changes. The uniqueness requirement ensures you're editing the right location. Bash/sed works but is brittle for complex strings.

Q30. What happens when Claude tries to call a tool that is not in the tools array passed to the API?

A) The SDK automatically adds the missing tool definition

B) The API returns a validation error — Claude can only call tools explicitly provided in the tools parameter

C) Claude creates a placeholder tool call that returns empty results

D) The call succeeds but the tool returns an access denied error

■ Correct Answer: B

Tool calls are constrained to the tools array. If Claude tries to call a tool not defined in the request, it's a validation error. Claude cannot 'invent' tools. This is the safety mechanism that prevents arbitrary tool invocations.

INTERMEDIATE QUESTIONS

Questions 31–60

Q31. You have two tools with similar functionality: search_knowledge_base (searches internal docs) and search_web (searches live internet). The agent consistently calls the wrong one. What is the best fix?

A) Remove one of the tools to eliminate ambiguity

B) Rename and rewrite both descriptions to explicitly state source type: 'Searches INTERNAL knowledge base (static, no internet)' vs 'Searches LIVE web (real-time, internet)'

C) Add a disambiguation instruction to the system prompt

D) Use tool_choice='any' to let Claude choose freely

■ Correct Answer: B

The root cause is overlapping descriptions. Make source type the primary differentiator in the name AND description. 'search_internal_kb' vs 'search_live_web' is immediately clear. Including 'internal/static' vs 'live/real-time' in descriptions eliminates ambiguity.

Q32. You need to enforce that extract_metadata is always called before any enrichment tools. Which approach is most reliable?

A) Add 'always call extract_metadata first' to the system prompt

B) Use tool_choice: {'type': 'tool', 'name': 'extract_metadata'} on the first API call, then switch to 'auto' for subsequent turns

C) Remove enrichment tools from the tool list and add them back after extraction

D) Create a combined extract_and_enrich tool that does both in sequence

■ Correct Answer: B

Forced tool selection on the first call guarantees extract_metadata runs first. Once metadata is extracted and in the conversation, subsequent turns with 'auto' allow Claude to choose appropriate enrichment tools. Prompt instructions for sequencing are unreliable.

Q33. A tool returns an empty array [] because no records matched the query. How should the tool response be structured?

A) {isError: true, message: 'No records found'} — empty results are an error

B) {data: [], total: 0, query: 'original query'} with isError absent/false — empty results are a successful query outcome

C) Return null to indicate no data

D) Throw an exception so Claude knows to retry with different parameters

■ Correct Answer: B

Empty results are not errors — they are valid successful query outcomes. Marking them as errors misleads Claude into thinking the tool failed and may need retrying. Return a success response with empty data, and include context about what was queried.

Q34. Your synthesis agent needs occasional fact-verification capability (cross-role tool). How should this be implemented without giving the synthesis agent full search access?

A) Add all search tools to the synthesis agent — it will only use what it needs

B) Create a constrained verify_fact tool that accepts a claim and returns a verification result, routing complex lookups through the coordinator

C) Route all fact verification through a dedicated verification subagent via coordinator

D) Use the coordinator to perform all fact verification on the synthesis agent's behalf

■ Correct Answer: B

Scoped cross-role tools solve this elegantly: a verify_fact tool (claim → verified/refuted/uncertain + source) gives synthesis capability without granting unrestricted search access. Complex lookups that exceed verify_fact's scope route back to the coordinator.

Q35. You are configuring a team's development environment where junior developers' MCP servers should have read-only access while senior developers have full access. How do you implement this with the MCP configuration system?

A) MCP doesn't support permission levels — everyone gets the same tools

B) Use separate .mcp.json configurations per role stored in different branches

C) Configure role-specific tool lists in the server; each developer uses the same server URL but different auth tokens that map to different permission levels

D) Create separate MCP server instances per role in the shared .mcp.json

■ Correct Answer: C

Permission levels are implemented server-side via auth tokens. The .mcp.json env var expansion (${DEVELOPER_TOKEN}) means each developer's environment has a different token that the MCP server maps to appropriate permissions. The client config is identical — the server handles authorization.

Q36. When should a subagent handle an error locally vs. propagate it to the coordinator?

A) Always propagate — the coordinator should handle all errors for centralized control

B) Handle locally: transient errors (retry with backoff). Propagate: errors that cannot be resolved locally (permission denied, permanent failures), always including what was attempted and any partial results

C) Always handle locally — never bother the coordinator with error details

D) Propagate all errors and let the coordinator decide whether to retry

■ Correct Answer: B

Local handling is appropriate for self-recoverable issues (temporary outages). Propagation is needed when the coordinator must make decisions (pivot strategy, escalate, use alternative source). Always include partial results when propagating — partial data is better than no data.

Q37. A developer asks why their MCP server configuration in .mcp.json requires GITHUB_TOKEN to be set on every developer's machine. What is the correct explanation?

A) This is a bug — .mcp.json should store the actual token

B) ${GITHUB_TOKEN} expands from the developer's local environment at runtime; this keeps the actual secret out of version control while sharing the configuration structure with the team

C) The token is only needed on CI machines — local developers can skip it

D) Tokens should be stored in CLAUDE.md as comments

■ Correct Answer: B

This is the intended security model: the configuration structure (which MCP server, which command, which env var name) is shared via git. The actual secret value lives only in each developer's local environment. No secret ever enters the repo.

Q38. What is the impact of adding a keyword like 'analyze' to a system prompt instruction when you also have a tool named 'analyze_document'?

A) No impact — system prompt keywords don't influence tool selection

B) Keyword-sensitive instructions in the system prompt can create unintended tool associations — the word 'analyze' may bias Claude toward calling analyze_document even when inappropriate

C) This improves tool selection by providing additional context

D) Tool names are case-insensitive so keyword matching doesn't apply

■ Correct Answer: B

System prompt language influences Claude's tool selection. If your prompt says 'always analyze the document structure first' and you have a tool called analyze_document, Claude may invoke it when the instruction was about a different type of analysis. Review system prompts for unintended keyword-tool associations.

Q39. You are designing tool interfaces for a financial data pipeline with 8 tools. Which grouping best follows the principle of scoped tool access?

A) Give all 8 tools to a single agent — simplicity over specialization

B) Data ingestion agent: fetch_raw_data, validate_schema. Analysis agent: calculate_metrics, detect_anomalies. Reporting agent: format_report, send_email, archive_results

C) Split tools evenly: 4 tools per agent regardless of functional relationship

D) Give each tool to its own dedicated agent

■ Correct Answer: B

Functional cohesion: each agent gets tools that work together for its specific role. Ingestion doesn't need reporting tools. Reporting doesn't need data fetching tools. This grouping minimizes cross-role misuse and makes each agent's purpose clear.

Q40. How should a tool return a partial success when it completes 7 of 10 requested operations before hitting a rate limit?

A) Return isError: true — partial completion is a failure

B) Return {partial_success: true, completed: [7 items], failed: [3 items with error details], retry_after: 30, message: 'Rate limited after 7 operations'}

C) Return only the 7 successful results — the agent will retry for the rest

D) Return isError: true only if 0 operations completed; success if any completed

■ Correct Answer: B

Partial success is real information. The agent needs to know: what succeeded (don't re-do), what failed (retry only these), and when to retry (retry_after). Returning only the 7 results causes the agent to think the task is complete and miss the 3.

Q41. When MCP tools are added to a Claude Code session, when are they discovered and made available?

A) On-demand, when Claude first decides to call a tool in that category

B) At connection time when the MCP servers start — all tools from all configured servers are discovered simultaneously and available throughout the session

C) They must be explicitly enabled per session with a /tools command

D) Tools are discovered lazily on first call and cached for subsequent calls

■ Correct Answer: B

MCP tool discovery happens at session start. All configured servers connect, all tools are enumerated, and they become part of the agent's available tool set. There is no per-call discovery overhead.

Q42. Your agent has a tool that fetches customer data and occasionally returns PII. You want to ensure PII never appears in Claude's context. What is the correct approach?

A) Add 'never store PII in memory' to the system prompt

B) Implement a PostToolUse hook that detects and redacts PII patterns before the result reaches Claude

C) Use a separate privacy-aware Claude model

D) Instruct the tool to omit PII fields before returning

■ Correct Answer: B

PostToolUse hooks provide deterministic PII redaction — they run on every tool result before Claude sees it. System prompt instructions are probabilistic. The tool itself may be an external system you can't modify. Hook-based redaction is the only reliable architectural solution.

Q43. What is the recommended strategy for building codebase understanding incrementally using built-in tools?

A) Read all files upfront to build complete context before starting analysis

B) Start with Grep to find entry points and key definitions, then use Read to follow imports and trace specific execution paths — discovering depth on demand

C) Use Glob to find all files, then Read each one in alphabetical order

D) Start with a Bash command to print the entire codebase

■ Correct Answer: B

Incremental discovery: Grep reveals structure (where is the main function? what imports X?), Read follows specific paths of interest. This is more efficient than reading all files upfront and avoids context window exhaustion on large codebases.

Q44. You notice that Claude is using a built-in Grep tool instead of your MCP-based semantic code search tool, even though the MCP tool produces much better results. What is the likely cause and fix?

A) Built-in tools are always preferred over MCP tools by design

B) The MCP tool's description doesn't clearly explain its advantage over Grep for code search; enhance the description to explain semantic search capabilities and when it outperforms Grep

C) Increase the weight of the MCP tool in the tool configuration

D) Remove the Grep built-in from allowedTools

■ Correct Answer: B

Tool selection is description-driven. If Grep's description says 'search file contents for patterns' and your MCP tool says 'search tool for code', Claude reasonably prefers the more specific description. Make your MCP description explicitly better: 'Semantic code search — understands intent, not just literal matches; use instead of Grep for conceptual searches'.

Q45. A tool that deletes records must require confirmation before executing in production. How should this be implemented architecturally?

A) Add 'always confirm before deleting' to the tool description

B) Implement a PreToolUse hook that intercepts delete_record tool calls and requires a human_approval field to be true before allowing execution

C) Create a confirm_delete tool that Claude must call before delete_record

D) Use a two-step system prompt that asks Claude to verify before deleting

■ Correct Answer: B

Programmatic enforcement via hook: block delete_record unless human_approval=true in the tool inputs. This cannot be bypassed by prompt injection or model non-compliance. Option C (confirm_delete tool) is also good but Claude could theoretically skip it; a hook cannot be skipped.

Q46. How should you structure MCP tool error responses for a permission denied error?

A) {isError: true, message: 'Access denied'} — brief is best

B) {isError: true, errorCategory: 'permission', isRetryable: false, message: 'Agent lacks write access to this resource', requiredPermission: 'billing:write', escalationPath: 'request access via IT portal'}

C) Throw an exception with the error code

D) Return empty data with a comment field noting the permission issue

■ Correct Answer: B

Rich permission errors enable intelligent handling: isRetryable: false (don't waste time retrying), requiredPermission tells Claude exactly what's needed, escalationPath provides an actionable next step. Claude can then explain to the user what access needs to be requested.

Q47. What is the purpose of the allowedTools restriction in subagent configuration?

A) To limit API costs by reducing the number of available tools

B) To ensure each subagent has access only to tools relevant to its specific role, preventing accidental or intentional misuse of out-of-scope tools

C) To comply with Anthropic's API terms of service

D) To prevent tool name collisions between different MCP servers

■ Correct Answer: B

allowedTools enforces the principle of least privilege for agents. A web search subagent with database write tools is a risk — it might accidentally (or via prompt injection) modify data. Restricting tools to role-relevant ones minimizes blast radius and improves reliability.

Q48. Your team has both a Jira MCP server (community) and a custom internal ticket system MCP server. How should these be configured?

A) Use only one ticket system — two creates confusion

B) Configure both in project-level .mcp.json; write clear tool descriptions that distinguish when to use each (e.g., 'use Jira for customer-facing tickets, use internal system for engineering tasks')

C) Put the community Jira server in user-level config and the custom server in project-level

D) Combine them into a single MCP proxy server

■ Correct Answer: B

Both can coexist in .mcp.json. The key is disambiguation via descriptions: Claude needs to understand when to use each. Clear 'use for' criteria in tool descriptions prevent Claude from guessing which ticket system to use.

Q49. What is the correct approach when the Edit tool fails to find a unique match in a file?

A) Use a more specific pattern in the Edit call

B) Read the full file content, apply the modification programmatically in your code, then Write the modified content back

C) Skip the modification — if Edit fails, the file cannot be modified

D) Use Bash with sed to make the edit instead

■ Correct Answer: B

Read + modify + Write is the reliable fallback when Edit cannot find a unique anchor. Read gives you the full file content, you apply the change, Write replaces the file. This is always reliable regardless of uniqueness constraints.

Q50. An MCP tool result contains a list of 500 items that exceeds practical context limits. What is the best tool design response?

A) Return all 500 items — context limits are the caller's problem

B) Return a paginated response with a cursor: {data: [first 20], total: 500, cursor: 'abc123', has_more: true} and a corresponding fetch_next tool

C) Return only the top 10 items silently

D) Return an error indicating the result is too large

■ Correct Answer: B

Pagination at the tool level is the right design: return manageable chunks, signal there's more (has_more + cursor), and provide a way to get the next page. Silent truncation is deceptive; erroring on large results is unhelpful. Pagination enables incremental consumption.

Q51. A tool receives an input that passes schema validation but will cause the underlying system to fail (e.g., a customer_id that doesn't exist in the database). How should this be handled?

A) The schema should be extended to validate all business rules

B) Return a structured not_found error: {isError: true, errorCategory: 'not_found', message: 'Customer CUST-999 does not exist', isRetryable: false, suggestion: 'Verify the customer ID or use search_customer to find the correct ID'}

C) Return an empty result as if the customer exists but has no data

D) Throw a validation exception before attempting the database lookup

■ Correct Answer: B

Business logic errors beyond schema validation need structured error responses. not_found is a distinct category from validation or transient errors — it's not retryable as-is, but it suggests a concrete alternative action (search to find the correct ID). This enables intelligent Claude recovery.

Q52. What is the benefit of exposing MCP resources (like a list of available reports) versus requiring tool calls to discover available data?

A) Resources are faster than tool calls due to caching

B) Resources give the agent a browsable catalog of what's available without consuming tool call budget or adding latency — the agent can plan efficiently before fetching

C) Resources have higher API rate limits than tool calls

D) Resources are only readable, preventing accidental writes

■ Correct Answer: B

Discoverability without tool call cost: an MCP resource listing 'Available reports: Q1-2024-revenue, Q2-2024-revenue...' lets the agent understand what's available before deciding what to fetch. Without this, the agent must call list_reports() (a tool call) just to discover options.

Q53. You have a tool that returns a deeply nested JSON object (8 levels deep) with 200 fields. Claude frequently references the wrong fields. How do you fix this?

A) Flatten the JSON to a single level before returning

B) Add a summary field at the top level: {summary: {key_field_1, key_field_2, key_field_3}, full_data: {...nested...}} — Claude reads summary first for the most important data

C) Return only the most important 10 fields

D) Document the structure in the tool description

■ Correct Answer: B

Attention structure: Claude's attention is strongest on data it encounters first. A top-level summary with the most important fields ensures reliable access to critical data. The full nested structure is preserved for cases where Claude needs to drill deeper. Summary-first design beats documentation alone.

Q54. A tool call to an external payment API hangs for 30 seconds before timing out. How should the tool handle and communicate this?

A) Wait indefinitely — never timeout tool calls

B) Implement a client-side timeout (e.g., 10 seconds); return {isError: true, errorCategory: 'transient', isRetryable: true, message: 'Payment API timeout after 10s', retry_after: 30, attempted_amount: amount}

C) Retry 3 times before returning an error

D) Return a success response with pending status

■ Correct Answer: B

Client-side timeouts prevent hung tool calls from blocking the agentic loop indefinitely. Structured timeout errors with retry_after give Claude the information to make a retry decision. The attempted_amount field prevents Claude from losing track of what was being processed during recovery.

Q55. You need to design a tool that processes images for a Claude agent. What is the correct way to return image analysis results?

A) Return the raw image bytes as base64 in the tool result

B) Return structured analysis data: {objects_detected: [...], text_extracted: '...', dimensions: {w, h}, confidence_scores: {...}} — Claude receives structured data, not raw image bytes

C) Return a URL where Claude can fetch the analyzed image

D) Return a human-readable description paragraph

■ Correct Answer: B

Tools should return structured data that Claude can reason about programmatically. Raw image bytes in a tool result serve no purpose — Claude cannot process them differently than the original. Structured analysis (detected objects, extracted text, metadata) is actionable and can be schema-validated.

Q56. What is the relationship between tool input_schema and the quality of Claude's tool calls?

A) Schema is only for validation — it doesn't affect the quality of Claude's inputs

B) Detailed input_schema with property descriptions, examples, format patterns, and constraints directly guides Claude's input construction, reducing invalid or suboptimal tool calls

C) Simpler schemas produce better tool calls — complexity confuses the model

D) Schema properties are developer documentation only — Claude doesn't read them

■ Correct Answer: B

Claude reads and uses property descriptions in input_schema: a property with description: 'Customer ID in format CUST-XXXXXX, e.g., CUST-ABC123' produces correctly formatted IDs. A property with description: 'id' produces guessed formats. Schema quality directly affects tool call quality.

Q57. You discover that Claude sometimes calls your search tool with natural language queries when the tool expects keyword-based queries. How do you fix this?

A) Update the model to handle natural language

B) Add explicit input guidance to the tool description and property description: 'Accepts keyword-based queries only. NOT natural language. Good: "python async error handling". Bad: "How do I handle async errors in Python?"'

C) Pre-process Claude's tool inputs to convert natural language to keywords

D) Use a different tool naming convention

■ Correct Answer: B

Input format guidance with good/bad examples directly in the description eliminates the natural language vs keyword confusion. The contrast example ('Good: ... Bad: ...') is especially effective — it shows the difference concretely rather than describing it abstractly.

Q58. You have a tool that queries a database and sometimes returns sensitive customer PII. Not all callers of the tool should receive PII. How do you design this?

A) Create two separate tools: one with PII, one without

B) The tool accepts a requesting_agent_id; a PostToolUse hook checks the agent's authorization level and redacts PII fields not permitted for that agent before returning results

C) Add a include_pii: boolean parameter that defaults to false

D) Only allow PII-authorized agents to have the tool in their allowedTools

■ Correct Answer: B

Authorization-aware hooks enable fine-grained access control within a single tool. The tool returns full results; the hook redacts based on caller authorization. Option D (tool scoping) also works but creates tool proliferation. The hook approach centralizes PII policy in one place.

Q59. How should you version MCP tool APIs when changes are breaking (parameter renaming, schema changes) to avoid breaking deployed agents?

A) Update all agents simultaneously when changing the API

B) Use versioned tool names (get_customer_v2) or a version parameter; deprecate old versions with a deprecation notice in the description; maintain both versions during transition; remove old version after all agents migrate

C) Breaking changes require re-deploying all agents immediately

D) MCP tools cannot be versioned — design schemas carefully upfront

■ Correct Answer: B

Versioning strategy: new version alongside old enables gradual migration. Deprecation notices in tool descriptions signal agents to update. Maintaining both versions prevents production outages during rollout. Forced simultaneous migration breaks all agents on deployment day.

Q60. A tool that wraps a third-party API needs to handle the case where the API returns rate limit errors (429). What is the correct tool behavior?

A) Immediately return isError: true to the calling agent

B) Implement internal exponential backoff within the tool (1-3 retries); if all retries fail, return {isError: true, errorCategory: 'transient', isRetryable: true, retry_after: 30, message: 'Rate limited after 3 attempts'}

C) Queue the request and wait indefinitely until the rate limit resets

D) Switch to a backup API endpoint automatically

■ Correct Answer: B

Internal retry handles most transient rate limits transparently. If internal retries exhaust, return structured error with retry_after — the agent knows to wait before its own retry. This encapsulates rate limit handling within the tool, simplifying agent logic.

ADVANCED QUESTIONS

Questions 61–90

Q61. You are designing tool interfaces for a healthcare data system. Which approach best balances Claude's tool selection reliability with security requirements?

A) Give Claude all 25 tools and rely on the system prompt for access control

B) Implement role-based tool sets with a maximum of 5-7 tools per agent role; enforce data access controls in PreToolUse hooks; never rely on prompt instructions for PHI access control

C) Use a single universal tool with a permission parameter

D) Separate tools for reading vs. writing are sufficient security

■ Correct Answer: B

Healthcare requires defense in depth: (1) scoped tool sets improve selection reliability and reduce attack surface, (2) PreToolUse hooks provide programmatic PHI access control that cannot be bypassed, (3) never rely on prompts for HIPAA compliance. This is the pattern the exam rewards.

Q62. You discover that a malformed input to one of your MCP tools causes it to expose internal system paths in error messages. How should you fix this?

A) Add input validation instructions to the tool description

B) Implement input sanitization in the tool itself and ensure error messages never contain internal paths, stack traces, or system details — return generic error categories instead

C) Use a PreToolUse hook to validate inputs before calling the tool

D) Ask Claude not to pass malformed inputs in the system prompt

■ Correct Answer: B

Security must be implemented at the tool layer: sanitize inputs before processing, and ensure errors never leak internal details. PreToolUse hooks can add validation, but the tool itself must also be hardened since it may be called from contexts outside your agent.

Q63. You need to implement a tool that calls an external API with rate limits of 100 requests per minute. Your agentic system can generate up to 500 tool calls per minute. What is the correct architectural approach?

A) Let tool calls fail with rate limit errors and rely on Claude to retry

B) Implement a token bucket rate limiter in the tool wrapper; return {isError: true, errorCategory: 'transient', retry_after: seconds} when rate limited; batch multiple tool inputs where the API supports it

C) Add 'do not call this tool more than 100 times per minute' to the tool description

D) Route tool calls through a queue with no back-pressure mechanism

■ Correct Answer: B

Token bucket rate limiting at the tool layer provides accurate rate control. Returning structured retry_after enables Claude to time retries correctly. Batching reduces individual API calls. Prompt-based rate limiting is unreliable — Claude doesn't track real-time call frequency.

Q64. A multi-tenant system has customers A and B whose data must never mix. Both use the same agent system with the same tools. How do you implement data isolation?

A) Separate the tool code for each customer

B) Pass tenant_id in all tool calls; implement tenant isolation in the MCP server layer with row-level security or separate databases; validate tenant_id in PreToolUse hooks against the authenticated session

C) Use separate Claude sessions and trust that sessions are isolated

D) Add 'only access data for the current customer' to the system prompt

■ Correct Answer: B

Multi-tenant isolation requires: (1) tenant context in every tool call, (2) server-side enforcement (RLS or separate schemas), (3) PreToolUse hook validation that Claude cannot access another tenant's data. Prompt-only isolation is a security vulnerability.

Q65. You need to design a tool ecosystem for an agent that processes legal documents. The tools include: extract_entities, verify_citations, check_jurisdiction, calculate_deadlines, file_document, send_notification, archive_document, update_case. How should these be distributed across agents?

A) One agent with all 8 tools

B) Analysis agent (extract_entities, verify_citations, check_jurisdiction, calculate_deadlines); Filing agent (file_document, send_notification, archive_document, update_case) — with coordinator managing handoff and requiring human approval before filing

C) One tool per agent (8 agents)

D) Read tools in one agent, write tools in another

■ Correct Answer: B

Functional cohesion + safety: analysis tools belong together (they work on document content), filing tools belong together (they act on external systems). Separating analysis from filing creates a natural approval gate — the coordinator reviews analysis results before delegating to the filing agent.

Q66. You are building a tool that executes code snippets. What security controls must be implemented at the tool layer regardless of Claude's instructions?

A) Add 'only execute safe code' to the tool description

B) Implement: sandboxed execution environment (no network access, no file system access outside designated directory), resource limits (CPU time, memory), timeout enforcement, and output sanitization before returning to Claude

C) Validate code syntax before execution

D) Use a separate Claude instance to review code before execution

■ Correct Answer: B

Code execution tools require hardened sandboxing at the infrastructure level. Prompt instructions cannot prevent malicious code execution. Sandboxing, resource limits, and timeouts must be enforced by the execution environment itself, independent of any LLM instructions.

Q67. Your MCP server needs to expose a large corporate knowledge base (10,000 documents) to Claude agents. How should you design the interface for maximum efficiency?

A) Return all 10,000 documents as a single tool response

B) Expose a semantic_search tool for targeted queries, an MCP resource listing document categories/summaries, and paginated fetch tools — enabling agents to discover and retrieve only relevant content

C) Create one tool per document

D) Compress all documents into a single 200K-token context injection

■ Correct Answer: B

Multi-tier access: MCP resources provide structure visibility (what's available), semantic_search enables targeted retrieval, paginated fetch handles large results. This avoids overwhelming Claude with irrelevant content while making the full knowledge base accessible on demand.

Q68. A PostToolUse hook must handle results from tools that can return up to 50MB of data. Some agents need full data; others need summaries. How should the hook be designed?

A) Always return full data — let each agent decide what to use

B) The hook receives the calling agent's context; for agents with 'summary_mode' configuration, apply compression/summarization before returning; for full-access agents, return complete data

C) Always return summaries — 50MB is too large for any context

D) Implement separate tool endpoints for summary vs. full access

■ Correct Answer: B

Context-aware hooks adapt transformation to the consumer. Knowing which agent is calling (from the hook's execution context) enables appropriate data transformation. Summary agents get compressed data; analysis agents get full data. One hook handles both cases.

Q69. You are designing the tool interface for an agent that needs to interact with a database. The database has 50 tables. What is the most reliable tool design?

A) One universal query_database tool that accepts raw SQL

B) Role-specific semantic tools: get_customer(id), get_orders(customer_id, date_range), update_order_status(order_id, status) — named by business intent, not technical implementation

C) One tool per database table

D) Two tools: read_database and write_database with SQL parameters

■ Correct Answer: B

Semantic tools beat raw SQL: (1) Claude doesn't need to know the schema, (2) business-intent names eliminate ambiguity, (3) specific parameter schemas prevent SQL injection, (4) you control exactly what operations are possible. Raw SQL is a security risk and selection reliability nightmare.

Q70. What is the most robust way to ensure tool call ordering when you have three tools that must execute in sequence: authenticate → fetch_data → transform_data?

A) List them in the correct order in the tools array

B) Combine multiple enforcement layers: forced tool_choice for authenticate, a PreToolUse hook that blocks fetch_data until authentication token exists in context, and another that blocks transform_data until raw data is present

C) Add sequencing instructions to each tool's description

D) Use a single combined authenticate_fetch_transform tool

■ Correct Answer: B

Multi-layer enforcement: forced first step ensures the sequence starts correctly, prerequisite hooks ensure each step only runs after dependencies are satisfied. This is defense in depth for workflow ordering — each layer adds reliability.

Q71. When multiple agents share the same MCP server but need different levels of access, what is the correct pattern?

A) Configure separate MCP server instances for each access level

B) Use authentication tokens that map to permission levels in the MCP server; pass the appropriate token per agent context; the server enforces access control at the tool call level

C) Define separate tool sets for each agent in the MCP server

D) Use a proxy layer that filters tool calls based on agent identity

■ Correct Answer: B

Token-based authorization at the MCP server level is the correct pattern: one server instance, multiple permission levels via auth tokens. The coordinator provides each subagent's context with the appropriate token. This is scalable (one server, many permission levels) and the server enforces access control, not the agent.

Q72. A tool that writes files accepts a path parameter. What validation should the tool perform to prevent path traversal attacks?

A) Add path: string to the input schema — schema validation is sufficient

B) Server-side validation: normalize the path, verify it resolves within the allowed directory, reject any path containing '../', absolute paths outside allowed directories, or symlinks pointing outside

C) Add 'only write to the project directory' to the tool description

D) Use a sandbox environment and allow any path

■ Correct Answer: B

Path traversal is a security vulnerability where '../../../etc/passwd' escapes the intended directory. Schema validation only checks type, not content security. Server-side path normalization and directory boundary checking is the correct security control. Prompt instructions cannot prevent this attack.

Q73. You are designing MCP tooling for an enterprise with strict data residency requirements (EU data must never leave EU infrastructure). How do you enforce this at the MCP layer?

A) Add 'respect data residency' to all tool descriptions

B) Implement geography-aware MCP routing: each data operation carries a data_residency tag; a routing layer directs tool calls to region-specific MCP server instances; PreToolUse hooks validate that the requested data source matches the authorized region for the requesting tenant

C) Use only EU-hosted MCP servers for all tenants

D) Data residency is an infrastructure concern — not an MCP design concern

■ Correct Answer: B

MCP-layer residency enforcement: data_residency metadata in tool calls enables routing to correct regional servers. PreToolUse hooks provide a programmatic enforcement gate — they can reject calls that would violate residency requirements before execution. This makes residency a first-class concern at the tool interface layer.

Q74. You need to build an MCP server that provides AI agents access to a large graph database. Queries can explore complex relationship networks. What tool design pattern is optimal?

A) One raw_cypher_query tool that accepts any Cypher/SPARQL query

B) Semantic traversal tools: find_connected_entities(entity_id, relationship_type, depth), get_entity_details(entity_id), find_paths(from_entity, to_entity, max_depth). Each tool encapsulates graph traversal logic behind a semantic interface.

C) One tool per entity type in the graph

D) Export the entire graph as JSON and pass it to the agent as context

■ Correct Answer: B

Semantic graph tools: agents express graph exploration intent, not Cypher syntax. find_connected_entities is unambiguous; raw Cypher requires the agent to know the graph schema and query syntax. Semantic tools also enable server-side optimization (query planning, caching common traversals).

Q75. A financial services firm needs MCP tools that interact with multiple core banking systems, each requiring different authentication mechanisms (OAuth2, API key, certificate). How do you design a unified tool interface?

A) Create separate tool sets for each banking system with system-specific authentication

B) Abstract authentication behind a credential manager: tools use semantic names (get_account_balance, process_transaction), a credential manager handles auth per system transparently, agents never see authentication details

C) Use a single shared service account for all banking systems

D) Require the agent to specify the authentication method per tool call

■ Correct Answer: B

Auth abstraction: agents operate at business semantics (get_account_balance), the credential manager handles auth complexity transparently. This separates concerns cleanly: agents don't need to manage auth, auth can be rotated without changing tool interfaces, audit logs at the credential manager provide unified access records.

Q76. You need MCP tools for an agent that orchestrates complex multi-system workflows where each step may create resources that must be cleaned up if a subsequent step fails. What pattern enables reliable cleanup?

A) Add cleanup instructions to the agent's system prompt

B) Implement saga pattern: each tool operation registers a compensating transaction (rollback action); if any step fails, execute compensating transactions in reverse order; the MCP layer tracks the saga state

C) Use idempotent tools that can be safely re-run without side effects

D) Wrap all operations in a database transaction

■ Correct Answer: B

Saga pattern for distributed rollback: create_vm → registers delete_vm as compensation. On failure, execute compensating transactions in reverse: undo step N, N-1, N-2... This handles cleanup across systems that don't share transactions. The MCP layer as saga coordinator centralizes orchestration logic.

Q77. You are designing MCP tooling for an agent that needs to interact with a regulatory compliance database with complex access rules: some users can see all data, others only their jurisdiction, others only non-confidential records. How do you implement this efficiently?

A) Create three separate MCP servers with different data subsets

B) Row-level security at the MCP server: tools accept user context with permissions; server-side query filtering applies permissions automatically; tool descriptions document what data is available at each permission level; no multiple server instances needed

C) Have agents self-filter results based on their role

D) Return all data and rely on PostToolUse hooks to redact

■ Correct Answer: B

RLS at the server layer is more secure and efficient than client-side filtering: unauthorized data never leaves the database, query performance is optimized (DB-level filtering is faster than loading and filtering in Python), and access control is centralized. Agent self-filtering is insecure — agents can make mistakes.

Q78. An MCP tool that calls an external API starts returning different response schemas after the API provider's version upgrade. This breaks downstream agents. How do you design resilience against API schema changes?

A) Pin to a specific API version permanently

B) Implement schema normalization in the MCP server: detect response schema version from API headers or field presence; normalize to the canonical internal schema before returning; alert and log schema drift; gradually migrate when stable

C) Return raw API responses and update all agents simultaneously

D) Block all calls until schema compatibility is restored

■ Correct Answer: B

Schema normalization at the MCP boundary: the server detects API version and normalizes to a stable internal schema. Agents never see API provider schema changes. Schema drift alerting enables proactive migration. This decouples agent development from external API version lifecycles.

Q79. You need to expose a machine learning model's predictions as an MCP tool. The model produces confidence scores and feature importances. How do you design the output to maximize agent usefulness?

A) Return only the prediction label — confidence scores add noise

B) Return structured output: prediction, confidence, top_features_driving_prediction, uncertainty_type (epistemic/aleatoric), and a plain_language_explanation. Include a human_review_recommended boolean when confidence < threshold.

C) Return a probability distribution over all possible classes

D) Return the raw model output tensor and let the agent interpret it

■ Correct Answer: B

ML tool output for agents: prediction alone is insufficient. Confidence enables routing decisions. Feature importances enable explanations. Uncertainty type distinguishes 'not enough data' from 'genuinely ambiguous'. human_review_recommended triggers automatic escalation. Plain language explanation enables user-facing messaging.

Q80. How do you design MCP tool observability for diagnosing why an agent is making unexpected tool selections in production?

A) Log all tool calls with timestamps — that's sufficient

B) Implement structured tool telemetry: per-call logging of (tool_name, input_parameters, response_summary, latency, error_if_any, calling_agent_id, session_id, correlation_id). Build a tool selection pattern dashboard. Add semantic search over tool call logs to find similar past calls.

C) Monitor only failed tool calls

D) Add a reasoning field to tool inputs where Claude explains its selection

■ Correct Answer: B

Tool observability for debugging unexpected selections: structured logs enable pattern analysis ('when input contains X, agent always calls wrong tool Y'). Correlation IDs link tool calls to the agent session for full trace reconstruction. Semantic search over logs finds historical cases of the same selection bug for root cause analysis.

Q81. A compliance team requires that every MCP tool call be approved by a policy engine before execution. The policy engine has 200ms average latency. How do you implement this without making the agent unbearably slow?

A) Remove the policy engine requirement — 200ms per tool call is too slow

B) Implement async pre-approval: for each tool call, initiate policy check asynchronously; if policy check resolves before tool execution starts, proceed/block accordingly; implement result caching for repeated identical calls; batch policy checks for predictable tool sequences

C) Pre-approve all tools at session start instead of per-call

D) Only check policy for tools marked as sensitive

■ Correct Answer: B

Async policy checking minimizes latency impact: fire policy check and (for low-risk tools) begin tool preparation in parallel. Result caching avoids repeated checks for identical calls (same tool, same parameters). Batch checking for predictable sequences reduces round-trips. 200ms sequential per call on 20 tool calls = 4s pure overhead — async + caching reduces this to near-zero for cached calls.

Q82. You are building an MCP tool ecosystem for an agent that performs security penetration testing. The same tools could be used for legitimate testing or malicious activity. How do you design appropriate safeguards?

A) Don't build these tools — security testing should be done manually

B) Implement strict authorization: all tools require a signed scope document (authorized target hosts, authorized test types, time window); PreToolUse hooks validate scope before every call; all actions logged immutably; out-of-scope actions blocked at the tool layer

C) Add 'only use on systems you are authorized to test' to the tool descriptions

D) Limit tools to read-only scanning — no active exploitation tools

■ Correct Answer: B

Authorization-controlled security tools: signed scope documents bound what can be tested. Hook-enforced validation prevents out-of-scope actions even if the agent misbehaves. Immutable logging provides accountability. Tool descriptions alone are prompt-based and insufficient for controlling potentially harmful capabilities.

Q83. How do you design MCP tools for an agent operating in a hybrid cloud environment where some operations must be processed on-premises (for data compliance) and others can use cloud APIs?

A) Create separate on-premises and cloud tool sets and let agents choose

B) Implement data classification tagging: each tool call carries data classification; a routing layer sends classified data to on-premises MCP servers and non-classified data to cloud MCP servers; agents use a single unified tool interface regardless of processing location

C) Process all data on-premises for maximum safety

D) Process all data in the cloud for maximum availability

■ Correct Answer: B

Transparent routing via data classification: agents use unified tool interfaces without knowing processing location. The routing layer enforces compliance by directing data to the appropriate processing environment. This enables cloud efficiency for non-classified data while maintaining compliance for regulated data.

Q84. An MCP server wrapping a document database needs to handle the case where an agent's search returns 10,000 matching documents. What is the correct design?

A) Return all 10,000 documents — the agent should filter

B) Return paginated results (default page_size: 20), include result count and total_pages, provide a semantic_summary of the full result set, and a refine_query tool that helps the agent narrow results before pagination

C) Return only the top 10 by relevance score

D) Return an error: too many results — refine your query

■ Correct Answer: B

Large result sets need multiple design elements: pagination for manageable consumption, semantic summary for fast overview, refine_query guidance for narrowing. Returning only top-10 silently loses potentially relevant documents. Error response is unhelpful — provide guidance for refinement instead.

Q85. You need to design an MCP tool for an agent that performs code refactoring. The refactoring involves reading source code and writing modified versions. What safety mechanisms must be built into the tool design?

A) Trust the agent — it has good code generation capabilities

B) Tool-level safety: backup creation before modification, diff generation showing exactly what changed, dry_run mode for preview, syntax validation before writing, and rollback tool for undoing changes. All actions logged with original and modified content.

C) Only allow read operations — writes are too risky

D) Require the agent to show the diff in its response before calling the write tool

■ Correct Answer: B

Code modification safety: automated backup prevents data loss, diffs make changes reviewable, dry_run enables previewing, syntax validation prevents writing broken code, rollback enables recovery. Requiring agents to show diffs in responses adds a review step but doesn't enforce anything — tools must be independently safe.

Q86. How should you design MCP tools for an agent working with real-time streaming data (stock prices, sensor readings, event streams)?

A) Snapshot tools only — streaming is not supported in MCP

B) Provide both snapshot tools (get_current_price) and subscription management tools (subscribe_to_feed, poll_feed, unsubscribe_from_feed); subscriptions persist between tool calls; the agent polls for updates rather than blocking

C) Stream all data via the tool result as a continuous response

D) Cache streaming data and refresh every 60 seconds

■ Correct Answer: B

Subscription model for streaming data: agents subscribe, then poll for accumulated updates. This decouples the agent's interaction frequency from the data stream rate. Subscription persistence between calls avoids re-connection overhead. Blocking streaming is incompatible with the tool call/result pattern.

Q87. You are building MCP tooling for agents that need to interact with a legacy SOAP/XML API. How do you expose this cleanly to Claude agents?

A) Pass SOAP XML directly to agents and let them construct calls

B) Build an MCP adapter layer: convert semantic tool calls (get_customer_orders(customer_id, date_range)) to SOAP XML requests transparently; parse SOAP XML responses into clean JSON objects; agents never see SOAP protocol details

C) Use a regex tool to help Claude parse SOAP XML

D) Document the SOAP schema in tool descriptions for Claude to learn

■ Correct Answer: B

Protocol abstraction at the MCP layer: agents get clean JSON interfaces; the adapter handles all SOAP complexity. This separates agent logic from legacy protocol details, enables future API migration without agent changes, and prevents the model from needing to understand SOAP envelope structure.

Q88. How do you design MCP tools that need to handle the 'fan-out read' pattern — where one logical request requires reading from 5 different systems and merging results?

A) Make 5 separate tool calls and have the agent merge results

B) Implement a composite tool that performs fan-out and merge server-side: the tool accepts the logical query, queries all 5 systems in parallel, merges and deduplicates results, and returns a unified result set. Faster and uses one tool call slot.

C) Have the agent decide which of the 5 systems to query based on context

D) Create a data warehouse that pre-merges the 5 systems' data

■ Correct Answer: B

Server-side fan-out is more efficient: one tool call triggers 5 parallel upstream queries, merging happens at network speed server-side. Exposing 5 separate tools and having the agent orchestrate adds 4 unnecessary round-trips and requires the agent to understand the merging logic.

Q89. You need MCP tools for an agent that processes financial transactions where some must complete within 100ms (trading) and others have 5-second SLAs (settlement). How do you design for mixed latency requirements?

A) All tools must meet the strictest (100ms) requirement

B) Tool-level SLA metadata: each tool declares its expected_latency_ms and guaranteed_sla_ms in its definition; a routing layer can select appropriate backend implementations (fast-path vs. standard-path) based on SLA requirements at tool invocation time

C) Use the Batch API for settlement, synchronous for trading

D) Create separate MCP servers for trading and settlement tools

■ Correct Answer: B

SLA-aware tool design: tool metadata enables intelligent routing. The trading execution tool routes to a pre-warmed, co-located backend (sub-100ms). The settlement tool routes to a standard backend. Agents declare latency requirements via tool selection; routing handles fulfillment. Separate servers add operational complexity without benefit.

Q90. What is the correct pattern for implementing 'tool result caching' in an MCP server to reduce redundant external API calls without serving stale data?

A) Cache all tool results permanently — external systems rarely change

B) Cache with TTL matching the data's natural change frequency: stock prices (1s), exchange rates (60s), static reference data (24hrs). Include cache metadata in tool results (cache_hit, cache_age, expires_at). Support explicit cache invalidation for event-driven updates.

C) Never cache tool results — freshness is always required

D) Cache based on result size — cache small results, skip large ones

■ Correct Answer: B

TTL-based caching aligned with data freshness requirements: not all data changes at the same rate. Cache metadata enables agents to assess data freshness. Event-driven invalidation handles cases where TTL alone is too coarse (a price change should invalidate the cache immediately, not wait for TTL).