Domain 5 — 15%

Context Management & Reliability

Domain 5 — 15% of the CCA exam

IN THIS MODULE:

- Context window budgeting and overflow prevention
- Checkpoint and session resumption patterns
- Multi-instance independent review
- Human-in-the-loop escalation triggers
- Attention dilution and lost-in-the-middle mitigation

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. What does the context window include in a Claude API call?

A) Only the current user message

B) System prompt + all conversation messages + tool results — everything in the call counts toward the context limit

C) Only messages — system prompt doesn't count

D) Only the last 10 messages to optimize performance

■ Correct Answer: B

The context window is the total of everything Claude processes in one call: system prompt (even if cached), all messages, all tool results. Monitoring input_tokens in the usage object tracks how close you are to the limit.

Q2. What is the first sign that your agentic system is approaching context window exhaustion?

A) Claude starts producing shorter responses

B) usage.input_tokens approaches the model's context limit (200K for current models)

C) The API returns a 413 error

D) Response quality degrades noticeably before any error

■ Correct Answer: B

Monitor usage.input_tokens on every response. At 80% capacity (~160K tokens), implement a context management strategy (summarize history, start a new session) before hitting the limit. Silent quality degradation often precedes hard limits.

Q3. What is a structured handoff in multi-agent systems?

A) A formatted API request between two agent instances

B) A compiled data package (customer ID, issue root cause, steps taken, recommended action) that gives the receiving agent or human everything needed to act without reading the full conversation

C) The message format used between coordinator and subagents

D) A checkpoint saved when a session is paused

■ Correct Answer: B

Structured handoffs are essential for escalations and cross-system handoffs. The receiver (human agent, downstream system) never has access to the conversation transcript — the handoff is their only information. It must be complete and actionable.

Q4. What is human-in-the-loop (HITL) in agentic systems?

A) Having a human developer monitor all Claude API calls for safety

B) Designed pause points where human judgment is required before the agent proceeds — typically for high-stakes, irreversible, or ambiguous decisions

C) Manual review of all Claude outputs before showing them to users

D) A debugging mode where developers step through agent actions one by one

■ Correct Answer: B

HITL is architectural: specific triggers (amount threshold, account deletion, low confidence) pause the agentic workflow and request human input or approval before proceeding. The agent resumes after human decision.

Q5. Why does self-review by the same Claude session fail to catch as many issues as independent review?

A) Self-review uses more tokens and produces worse quality

B) The generating session retains the reasoning context that produced the output — it is less likely to question decisions it has already made

C) Claude API sessions cannot review their own outputs due to API restrictions

D) Self-review mode is not activated by default

■ Correct Answer: B

Cognitive bias applies to LLMs too: the session that generated code 'understands' why it made each decision and will rationalize it. An independent session approaches the code fresh — it sees the result, not the reasoning, and is more likely to spot flaws.

Q6. What triggers should automatically escalate an agent conversation to a human?

A) Any conversation longer than 10 turns

B) High-stakes irreversible actions (large financial transactions), low model confidence, repeated issue recurrence, or when the user explicitly requests a human

C) Every third conversation as a quality sample

D) Only when Claude explicitly requests escalation

■ Correct Answer: B

Rule-based escalation triggers are more reliable than model-decided escalation. Define explicit thresholds: refund > $500, account deletion, confidence < 0.8, same issue reopened 3 times. These cover the cases where human judgment adds the most value.

Q7. What is the purpose of a confidence scoring field in structured extraction schemas?

A) To measure how confident the user is in their request

B) To have Claude assess its own extraction certainty, enabling automated routing of low-confidence results to human review

C) To determine which model version to use for the next call

D) Confidence scoring is automatic — no schema field is needed

■ Correct Answer: B

Embedding confidence in the schema leverages Claude's ability to self-assess. A confidence: 0.45 on an invoice extraction signals: route this to human review. A confidence: 0.97 means: process automatically. This is the foundation of automated quality triage.

Q8. What is the recommended approach when a multi-turn conversation accumulates 180,000 tokens and the task is not yet complete?

A) Continue — 200K is the limit and you have 20K remaining

B) Proactively summarize the conversation using a cheap model (haiku), start a fresh session with the summary as context, and continue the task from the checkpoint

C) Reduce max_tokens on remaining calls to create more input space

D) Split the conversation into two parallel sessions

■ Correct Answer: B

At 90% capacity, proactive checkpointing is better than reactive recovery. Summarize before hitting the wall: compress key findings, decisions, and remaining tasks into a structured summary, then start fresh. Hitting 200K mid-task forces a harder recovery.

Q9. What is the multi-pass review technique for large PRs?

A) Running the same review prompt three times and taking the best result

B) A first pass of per-file local analysis, then a second pass of cross-file integration analysis — separating concerns for better coverage

C) Having multiple developers each review a section of the PR

D) Progressive disclosure: start with a summary, then drill into details

■ Correct Answer: B

Multi-pass separates concerns: local pass focuses on within-file correctness (null checks, error handling, logic bugs). Integration pass focuses on cross-file consistency (interface matching, type compatibility, import correctness). Mixed single-pass reviews show attention dilution.

Q10. What is attention dilution in the context of large context windows?

A) Network latency when context windows are very large

B) Claude's effective attention is not uniform across very long contexts — information in the middle of large tool results may receive less effective processing than content at the beginning or end

C) The cost increase when using large context windows

D) Token limit errors that truncate context silently

■ Correct Answer: B

Research shows LLM attention is stronger at the start and end of long contexts (the 'lost in the middle' problem). For critical information in long tool results, ensure it's at the beginning or clearly structured for navigation. Don't bury key findings in the middle of large blobs.

Q11. What is conversation pruning and when is it appropriate?

A) Deleting all messages older than a specific date

B) Removing the oldest conversation turns when approaching context limits — appropriate as a lightweight strategy for low-stakes conversations where early context has little impact on the current task

C) Filtering profanity and sensitive content from conversations

D) Truncating the system prompt to free context space

■ Correct Answer: B

Pruning drops oldest user+assistant pairs when context approaches limits. Appropriate for: ongoing chat where early messages are irrelevant to current exchange. NOT appropriate for: tasks where early decisions affect later steps. For complex tasks, structured summarization is more reliable than pruning.

Q12. What should a multi-agent handoff summary include when passing work from an analysis agent to a report generation agent?

A) The analysis agent's full conversation history

B) Key findings (structured), data sources and citations, confidence levels, outstanding questions, and any data gaps the report should acknowledge

C) A raw dump of all tool results from the analysis session

D) Only the final conclusion — omit the supporting data

■ Correct Answer: B

The report generation agent needs actionable structured input: findings with citations (for attribution), confidence levels (for appropriate hedging in the report), and acknowledged gaps (so the report is honest about limitations). Raw tool results are too large and unstructured.

Q13. When is it appropriate for an agentic system to ask for clarification from the user?

A) Never — agents should always proceed with their best interpretation

B) When ambiguity would lead to significantly different outcomes, the cost of asking is low, and the task is not time-critical

C) Before every action to ensure the user approves each step

D) Only when the task explicitly involves financial transactions

■ Correct Answer: B

Clarification is valuable when ambiguity is high-impact: 'delete all records for client X' — does X mean the company or a specific person? Asking avoids irreversible errors. But over-asking creates poor UX — ask only when the ambiguity genuinely changes the outcome significantly.

Q14. What is the difference between session resumption and starting fresh with a summary?

A) There is no practical difference

B) Resumption (--resume) continues with all prior context intact — good when prior tool results are still valid. Fresh with summary starts new — good when prior tool results are stale or context window is too large.

C) Resumption is for interactive use; fresh+summary is for CI/CD

D) Resumption requires paid API tier; fresh sessions are free

■ Correct Answer: B

Choose based on context validity: if you investigated a codebase yesterday and it hasn't changed, resumption gives you all that context efficiently. If files have changed or you're 190K tokens in, a fresh session with a structured summary of key findings is more reliable.

Q15. How do you preserve attribution (which source contributed which fact) when passing data between agents?

A) Include source citations as inline text within the findings

B) Use structured JSON that separates content from metadata: {finding: '...', source: {url: '...', agent: '...', timestamp: '...'}}

C) Attribution is automatic — agents track their own sources

D) Use numbered footnotes in plain text

■ Correct Answer: B

Structured source metadata ensures attributions survive agent handoffs. When the synthesis agent produces a report, it can cite sources correctly because each finding carries its own source record. Inline text citations get lost when content is restructured.

Q16. What is the most reliable strategy for managing context in a multi-day agentic task?

A) Use a single session for the entire task — context accumulated over days is valuable

B) Daily checkpoints: at end of each work session, summarize findings and decisions into a structured checkpoint; start next session with checkpoint injection

C) Split the task across multiple sessions running in parallel

D) Store the full context in a database and reload it each session

■ Correct Answer: B

Daily checkpoints prevent context window issues and ensure continuity. A structured checkpoint (findings, decisions, next steps) is injected as initial context for the next session. This pattern scales to weeks-long tasks without context limit concerns.

Q17. What happens when an agent hits the context window limit mid-task?

A) Claude automatically summarizes its context and continues

B) The API returns an error (400 or context exceeded) and the task terminates abruptly

C) The oldest messages are silently dropped to make room

D) The agent switches to a model with a larger context window automatically

■ Correct Answer: B

Context overflow is an error, not graceful degradation. The API call fails. This is why proactive context management (checkpointing at 80-90% capacity) is critical — reactive recovery from a failed call mid-task is more disruptive than proactive checkpointing.

Q18. What key information must be preserved in a conversation checkpoint?

A) The exact text of every message

B) Task state (what's been done), key findings (with citations), decisions made (with rationale), remaining work, and any open questions

C) The model version, temperature, and max_tokens used

D) System prompt text only

■ Correct Answer: B

Checkpoints must enable resumption without the full history: what was accomplished (avoid redoing), what was found (key information), what decisions were made (avoid re-deciding), what remains (continue from here), and what's uncertain (flag for attention). This is the minimum viable checkpoint.

Q19. A customer support agent successfully resolved a customer issue but the customer asks to speak to a manager. What should the agent do?

A) Explain that managers are not available and offer to help further

B) Honor the request immediately, compile a structured handoff summary, and escalate to a human manager without arguing

C) Ask the customer to explain why they want a manager before escalating

D) Offer the customer a discount to resolve the issue without escalation

■ Correct Answer: B

Explicit user escalation requests must always be honored. The agent compiles a complete handoff (conversation summary, resolution attempted, customer context, reason for escalation request) and routes to human. Barriers to requested escalation erode customer trust.

Q20. When should an agentic loop NOT attempt to handle an error and instead escalate immediately?

A) All errors should be handled by the loop — never escalate automatically

B) When the error involves data integrity risks, exceeds retry limits, is a permission/auth failure for sensitive operations, or could cascade to other systems

C) Only when the error message contains the word 'critical'

D) When the API returns a 5xx error code

■ Correct Answer: B

Escalation triggers: (1) data integrity risk — don't retry operations that may have partially succeeded, (2) retry exhaustion — escalate with full context, (3) permission failures for sensitive ops — these need human authorization, (4) cascade risk — stop and alert rather than compound the problem.

Q21. What is 'token budget planning' and why is it important for long agentic workflows?

A) Setting max_tokens to control API costs

B) Estimating how many tokens each phase of a workflow will consume (system prompt, tool results, conversation history) to prevent context overflow before starting

C) Buying a token bundle from Anthropic for cost savings

D) The process of caching tokens to reduce costs over time

■ Correct Answer: B

Token budget planning: before a long workflow, estimate: system prompt (~1K), tool results (~5K per call × 20 calls = 100K), conversation history growth. If estimated total approaches 200K, plan checkpointing intervals before starting. Reactive recovery is more disruptive than proactive planning.

Q22. What is 'progressive disclosure' in the context of agentic system design?

A) Gradually revealing the system's architecture to end users

B) Providing context to agents incrementally — giving each agent only what it needs for the current step, retrieving more detail on demand rather than loading everything upfront

C) A UX technique for introducing users to new features gradually

D) Progressively increasing Claude's capabilities over time

■ Correct Answer: B

Progressive disclosure manages context efficiently: don't load 50-page documents upfront — provide summaries first, have agents request sections when needed. This is the multi-agent equivalent of lazy loading: fetch detail on demand, not all at once.

Q23. Why is appending a summary at the BEGINNING of a long tool result more effective than appending it at the end?

A) Beginnings are processed faster than ends

B) LLMs show stronger attention at context boundaries — information at the beginning receives more reliable processing than information buried in the middle of a long result

C) The API charges less for content at the beginning of messages

D) Summary at the beginning reduces total token count

■ Correct Answer: B

Attention patterns in long contexts: strong at beginning, weakens toward middle, slightly stronger at end. For critical information in a large tool result, lead with a summary: 'KEY FINDINGS: [summary]. FULL DATA: [long content]'. The model processes the key findings reliably even if it attends less to the full data.

Q24. What does it mean for an agent to 'prefer reversible actions'?

A) Always using read-only tools

B) When multiple approaches exist, choosing the one that can be undone if it turns out to be wrong — e.g., move file instead of delete, create draft instead of send, update test environment before production

C) Only taking actions that have been previously approved

D) Asking for confirmation before every action

■ Correct Answer: B

Reversibility as a design principle: move file → recoverable (move back). Delete file → not recoverable (unless backup). When both achieve the goal, prefer the reversible option. This reduces the cost of errors without sacrificing capability.

Q25. What is 'decision documentation' in the context of agentic workflows?

A) Recording which Claude model made each decision

B) Capturing why each significant decision was made, not just what was decided — enabling debugging, audit, and learning from mistakes

C) The human-readable documentation generated by the agent

D) A log of all API calls with timestamps

■ Correct Answer: B

Decision documentation captures reasoning: 'Chose approach B over approach A because: [reasons]. Tradeoffs: [what was given up].' This is essential for debugging unexpected outcomes, audit trails, and improving future prompts based on where reasoning went wrong.

Q26. What is 'context freshness' and when does it matter?

A) How recently the Claude model was trained

B) Whether the information in the agent's context accurately reflects the current state of the world — context becomes stale when time passes or when external systems change since the context was loaded

C) The recency of the user's messages in a conversation

D) A metric for prompt caching efficiency

■ Correct Answer: B

Context freshness: resumed sessions have 'old' context — file contents may have changed, API responses may be outdated, database records may be different. Before acting on prior session information, verify that time-sensitive context is still accurate. Explicit staleness checks prevent acting on outdated state.

Q27. How do you implement 'graceful shutdown' for an agentic workflow that must stop mid-task (user cancellation, timeout, resource limit)?

A) Terminate immediately — partial work is better than delayed shutdown

B) Checkpoint current state before terminating: save completed work, current position in task graph, in-flight actions status. Mark incomplete actions for cleanup or continuation. Return a structured status indicating what completed vs. what remains.

C) Graceful shutdown is not possible in agentic systems — terminate abruptly

D) Complete the current tool call then terminate — no checkpoint needed

■ Correct Answer: B

Graceful shutdown protocol: save completed work (don't lose it), record current position (enable resumption), handle in-flight actions (cancel or complete them cleanly), return structured status (next session can continue from the saved state). Abrupt termination leaves the system in an unknown state.

Q28. What is 'scope creep' in an agentic workflow and how does it affect reliability?

A) The gradual expansion of an AI's capabilities over time

B) When an agent expands the scope of its task beyond what was requested — investigating additional tangentially related issues instead of staying focused on the original request

C) When token usage grows unexpectedly due to large tool results

D) When a subagent fails to complete its assigned scope

■ Correct Answer: B

Scope creep in agents: an agent asked to fix a specific bug starts refactoring the entire module. While potentially helpful, this can: (1) consume context budget, (2) make unexpected changes, (3) cause downstream failures. Explicit scope boundaries in the task prompt and CLAUDE.md reduce scope creep.

Q29. What is the 'minimum context principle' in multi-agent design?

A) Using the smallest possible context window for efficiency

B) Giving each agent only the context it needs for its specific task — not everything you have, not everything that might be relevant, just what is directly needed

C) Keeping system prompts as short as possible

D) Limiting conversation history to the last 5 turns

■ Correct Answer: B

Minimum context principle: each agent is a context budget manager. Providing irrelevant context wastes tokens and can distract the agent. A synthesis agent doesn't need raw search results — it needs structured findings. Filter and compress context before passing to each agent.

Q30. What happens to model performance when important information is placed in the middle of a very long prompt (the 'lost in the middle' effect)?

A) Middle content is weighted equally to beginning/end content

B) Content in the middle of long contexts receives less reliable attention — models show stronger processing at the beginning and end of context windows

C) The model only processes the beginning and end of long prompts

D) This only affects prompts longer than 150K tokens

■ Correct Answer: B

Lost-in-the-middle is documented empirically: critical instructions in the middle of long prompts are sometimes missed. Mitigation: put critical instructions at the beginning AND repeat key constraints near the end. Use section headers for navigation. This applies to both system prompts and tool results.

INTERMEDIATE QUESTIONS

Questions 31–60

Q31. You are building a multi-agent research system. The synthesis agent receives combined output from 3 subagents totaling 45,000 tokens. Important findings are distributed throughout. How do you ensure the synthesis agent processes all findings?

A) Increase max_tokens — larger budgets improve coverage

B) Structure the input with critical findings first; use section headers for navigation; include an executive summary at the top listing the most important points from each agent

C) Use claude-opus which has better long-context performance

D) Split the 45,000-token input across 3 synthesis subagents

■ Correct Answer: B

Attention management for long inputs: key information at the top (strongest attention), section headers enable navigation, executive summary at the top signals important points for attention allocation. Burying critical findings in the middle of large blobs risks them being overlooked.

Q32. A production agentic system is processing a user task and has accumulated 170,000 tokens. The task will require at least 50 more tool calls. What should happen?

A) Continue until the limit is hit, then resume

B) Proactively checkpoint: summarize all findings and task state, start a new session with the checkpoint, continue from the checkpoint with fresh context

C) Switch to claude-haiku which uses fewer tokens per response

D) Prune all tool results from history to free space

■ Correct Answer: B

At 170K/200K (85%), proactive checkpointing before the remaining 50 tool calls is essential. 50 tool calls could easily add 30K+ tokens (tool results are often large). Hitting the limit mid-task forces abrupt recovery. Checkpoint now while you have control.

Q33. How should you implement a HITL workflow where a human approves a specific tool call before execution?

A) Add 'ask the user before calling tools' to the system prompt

B) Use a PreToolUse hook to intercept the tool call, send an approval request to a human (via queue, notification, UI), pause the agentic loop, and resume only after human approval is received

C) Run the tool call in a sandbox first, then ask the human if they approve the result

D) Add a confirm: true field to the tool input schema

■ Correct Answer: B

HITL implementation requires external coordination: (1) hook intercepts the call, (2) approval request sent to human via async channel (Slack, email, UI), (3) agentic loop pauses (blocking or callback-based), (4) human approves/rejects, (5) loop resumes with result. This is architecturally non-trivial — it requires external state management.

Q34. A complex task requires Claude to analyze a 150-page document and then generate a 30-page report. What context management strategy handles both phases?

A) Process both in a single 200K-token context window

B) Phase 1: Analyze the document with Explore subagent (isolated context), extracting a structured findings summary. Phase 2: Generate the report in the main session using the findings summary — not the full document.

C) Split the document into 10 sections and process each independently

D) Use the Batch API to process both phases asynchronously

■ Correct Answer: B

Phased approach: analysis phase (in isolated context) produces a findings summary (much smaller than the full document). Report generation phase uses the summary as input — focused, manageable context. The full document never enters the report generation context.

Q35. A long-running agent session discovers at turn 40 that a key assumption it made at turn 3 was incorrect. How should this be handled?

A) Ignore the error — the agent has already done significant work based on the assumption

B) Explicitly acknowledge the incorrect assumption, assess which findings/decisions were affected, note corrections to make, and proceed with corrected understanding — do not restart from the beginning unless most work is invalidated

C) Always restart the session from the beginning for correctness

D) Create a parallel session with the correct assumption and compare results

■ Correct Answer: B

Targeted correction is more efficient than full restart. Assess impact: if 5/40 decisions were affected, correct those specific decisions. If 38/40 decisions were affected, restart is more efficient. The agent's ability to reason about partial invalidity is what makes mid-task correction possible.

Q36. You need to implement a multi-agent system where findings must be verified across multiple sources before inclusion in the final report. What reliability pattern applies?

A) Trust all subagent findings equally

B) Implement cross-verification: each finding from one subagent is verified by a second subagent using a different source type; facts that cannot be corroborated are flagged as unverified in the output

C) Use a confidence threshold — findings above 0.8 confidence are trusted without verification

D) Have the coordinator verify all findings itself

■ Correct Answer: B

Cross-source verification: 'claim X from web search' → verify with 'academic source agent'. Findings verified across multiple source types have higher reliability. Unverified claims are labeled as such in the report — maintaining accuracy while preserving all found information.

Q37. What is the recommended context window management for a customer service agent handling a conversation that has lasted 2 hours with 180 messages?

A) Continue until the context limit is hit — 180 messages is unlikely to be near 200K tokens

B) Implement a rolling window: maintain the last 20 messages in full detail; summarize earlier messages into a compact context block; preserve any key facts (customer ID, issue timeline, commitments made) explicitly

C) Start a new session after every 50 messages

D) Delete messages older than 30 minutes

■ Correct Answer: B

Rolling window for long conversations: recent context (last 20 messages) enables natural conversation flow. Earlier messages are compressed. Key facts (commitments made, escalation history, customer ID) are preserved explicitly regardless of age. This balances context quality with window efficiency.

Q38. A document processing pipeline must handle documents that arrive out of order. Context dependencies exist: document A must be processed before document B (B references A). How do you manage this?

A) Process documents in arrival order — out-of-order handling is too complex

B) Implement dependency tracking: detect cross-document references during ingestion; queue dependent documents until their dependencies are processed; inject processed dependency context when processing dependents

C) Batch all documents and process simultaneously

D) Process A and B in the same agent session back to back

■ Correct Answer: B

Dependency-aware processing: detect references during ingestion (document B references A), queue B until A is processed, inject A's extracted data into B's processing context. This handles out-of-order arrival while preserving dependency correctness.

Q39. What self-evaluation technique helps an agent assess whether its research findings are comprehensive enough to support a detailed report?

A) Count the number of sources and accept if > 10

B) Have the agent explicitly evaluate: does each required section have sufficient supporting evidence? Are there gaps? Are there contradictions? Produce a coverage_assessment before starting the report.

C) Use a confidence threshold on each individual finding

D) Run the report generation and see if it produces adequate length

■ Correct Answer: B

Pre-report coverage assessment: 'Before writing, assess whether I have sufficient evidence for each required section.' The agent identifies gaps before committing to report generation — allowing targeted re-research for thin sections. Starting report generation without this check produces incomplete reports.

Q40. How should an agent handle a tool call that could be irreversible (like sending an email)?

A) Proceed immediately — agents should be decisive

B) Before calling irreversible tools: compile a summary of what will happen and why, require explicit confirmation (HITL or retry-with-confirmation pattern), and log the decision with full context for audit

C) Only log the action — no confirmation needed for email

D) Use a sandbox email environment and ask the user to manually send from there

■ Correct Answer: B

Irreversible actions require explicit confirmation gates: compile action summary (recipient, content, context), request confirmation, log the full decision (what, why, who confirmed, when). This pattern prevents accidental sends and provides audit trail for compliance.

Q41. An agentic system makes 15 tool calls in a row. Context is now at 160K tokens. Tool results account for 120K of those tokens. What is the most effective context reduction strategy?

A) Delete all tool results — they've been processed

B) Summarize large tool results into compact extractive summaries while preserving key data; remove verbatim results that are no longer needed for subsequent reasoning

C) Reduce max_tokens to prevent future context growth

D) Continue — 160K is within limits

■ Correct Answer: B

Tool result summarization: large verbatim results (HTML pages, full file contents) can be replaced with extractive summaries once their key information has been processed. '15,000-token file content' becomes '200-token key findings'. Preserve only what subsequent reasoning actually needs.

Q42. When should a customer service agent proactively offer a human escalation without the customer asking?

A) Never — wait for the customer to ask

B) When: the issue has been open for > 24 hours, the same issue has recurred 3+ times, the resolution requires authorization the agent doesn't have, or the customer expresses strong dissatisfaction

C) After every 5 messages as a standard courtesy

D) Only when the customer's language is detected as emotional

■ Correct Answer: B

Proactive escalation triggers respect customer time: long-open issues likely need human attention, recurring issues suggest a systemic problem, authorization limits mean the agent can't actually help, and visible dissatisfaction is a signal. Offering proactively before the customer demands it demonstrates service quality.

Q43. A complex investigation session needs to be handed off from one engineer to another. The session has 45 turns. What information must the handoff include?

A) A link to the session log

B) Current state (what's been found), key decisions (with rationale), dead ends (to avoid revisiting), open questions, immediate next steps, and any time-sensitive dependencies

C) The last 5 messages from the session

D) A summary of every turn sequentially

■ Correct Answer: B

Engineer-to-engineer handoffs need actionable context: what was found (save re-discovery), what was tried (avoid dead ends), what decisions were made and why (avoid re-deciding without new information), what's unclear (focus areas), what to do next (immediate resumption). Dead ends are especially important — they prevent the next engineer from repeating failed approaches.

Q44. You need to implement a multi-agent system where a quality control agent reviews all other agents' outputs before they reach the user. How does this affect reliability?

A) QC agent has no benefit — it reviews the same outputs with the same potential biases

B) An independent QC agent significantly improves reliability: it reviews outputs without the generation context, catches inconsistencies between multiple agents' outputs, and applies consistent quality standards across the system

C) QC agents double costs without sufficient benefit

D) QC should be done by the coordinator, not a separate agent

■ Correct Answer: B

Independent QC provides structural reliability benefits: (1) no generation bias, (2) can compare outputs from multiple agents for consistency, (3) applies uniform quality criteria regardless of which agent produced the output, (4) catches hallucinations by cross-checking claims. The coordinator is not ideal for QC — it has generation context.

Q45. What is the 'lost in the middle' problem and how do you mitigate it in long prompts?

A) Tokens in the middle of the context window are processed slower, increasing latency

B) LLMs show weaker attention to content in the middle of very long contexts. Mitigation: put critical instructions and key information at the beginning and/or end; use section headers for navigation; repeat critical instructions at the end.

C) Prompt injections hidden in the middle of long documents

D) Context window size issues when the prompt is longer than max_tokens

■ Correct Answer: B

Attention is strongest at context boundaries (beginning and end). For long system prompts or tool results: critical rules at the top, important summary at the bottom, section headers for middle content. Don't hide critical constraints in paragraph 7 of a 20-paragraph prompt.

Q46. An agentic system for data analysis must handle both 'I need a quick summary' (2-3 tool calls) and 'perform comprehensive analysis' (30+ tool calls) requests. How do you design context management for both?

A) Use the same context management for both — simpler architecture

B) Detect task complexity upfront; for simple tasks, no context management needed; for complex tasks, implement proactive checkpointing, tool result summarization, and session forking for parallel exploration

C) Always use the complex path — it handles simple tasks adequately

D) Limit all tasks to 10 tool calls maximum

■ Correct Answer: B

Adaptive context management: simple tasks don't need overhead. Complex tasks need proactive strategies. Detecting complexity early (via task decomposition or explicit complexity hints in the request) enables choosing the right strategy before context issues arise.

Q47. How do you implement graceful degradation when the primary tool fails in a multi-agent research system?

A) Fail the entire request — tool failures cannot be recovered from

B) For each primary tool, define fallback alternatives: if web_search fails → try alternative_search_api → if both fail → use cached_knowledge_base. Include in the report which sources were unavailable.

C) Retry the primary tool indefinitely until it succeeds

D) Use only tools that have 99.9% uptime SLAs

■ Correct Answer: B

Graceful degradation via fallback chains: primary → secondary → tertiary. Each fallback uses a different source. The final report documents which sources were unavailable — transparency about research limitations is better than silently failing or silently using inferior sources.

Q48. What should an agent do when it detects that the user's request is ambiguous and could reasonably be interpreted two very different ways?

A) Choose the interpretation that is easier to execute

B) Ask a single clarifying question that disambiguates the key ambiguity — don't ask multiple questions at once. Present the two interpretations concisely and ask which is intended.

C) Execute both interpretations and present both results

D) Always choose the more conservative interpretation

■ Correct Answer: B

Single targeted clarification: present both interpretations, ask which is intended. Don't ask multiple questions — it creates friction. Don't execute both — it wastes resources and may trigger irreversible actions. The conservative interpretation heuristic is reasonable but better to confirm when the stakes are high.

Q49. A multi-agent system produces a final synthesis report. How do you ensure subagent findings that contradict each other are handled correctly?

A) The synthesis agent should pick the finding it finds most credible

B) The coordinator detects contradictions during aggregation; flags them with both sources; spawns a verification subagent if needed; the final report presents contradictions with analysis rather than silently resolving them

C) Contradictions indicate subagent error — discard both and re-run

D) The most recent finding is always correct — use it

■ Correct Answer: B

Transparent contradiction handling: contradictions are information. Surface them with both sources and, if possible, a verification step. A high-quality report acknowledges 'Source A states X while Source B states Y — we assess Y to be more reliable because Z.' Silently resolving contradictions without disclosure is a reliability failure.

Q50. When is it appropriate for an agent to start a sub-task in an independent context (fork) vs. continuing in the main context?

A) Always fork — isolated contexts are always safer

B) Fork when: the sub-task is exploratory and likely to produce large verbose outputs; when the sub-task may fail and you want to maintain main session integrity; when you want to explore multiple approaches. Continue in main context when the sub-task's outputs are needed for subsequent main-session reasoning.

C) Never fork — independent contexts lose important context

D) Fork only when the sub-task involves write operations

■ Correct Answer: B

Fork decision criteria: exploration (large verbose outputs that would crowd context), isolation (failed fork doesn't corrupt main session), and parallel exploration all favor forking. When sub-task results are integral to the main session's next steps, keeping them in the main context avoids the overhead of summarization.

Q51. A customer service agent has successfully resolved an issue. Before ending the conversation, what reliability check should it perform?

A) End immediately after resolution — don't over-extend the conversation

B) Explicitly verify with the customer that the resolution addressed their actual concern; summarize what was done; confirm there are no remaining issues; offer follow-up if needed

C) Ask the customer to rate the service on a 1-5 scale

D) Review the entire conversation for compliance

■ Correct Answer: B

Resolution verification prevents premature closure: the agent's definition of 'resolved' may differ from the customer's. Explicit confirmation catches cases where the agent solved a symptom but not the root cause, or where the customer had additional concerns they hadn't yet expressed.

Q52. How should an agent handle a user request that requires accessing systems the agent is not authorized to access?

A) Attempt access and return whatever error occurs

B) Detect the authorization gap before attempting access; clearly explain what is needed and what authorization is required; offer to proceed with available authorized resources or escalate the authorization request to an appropriate person

C) Find alternative ways to get the data that circumvent the authorization requirement

D) Return an error and end the conversation

■ Correct Answer: B

Proactive authorization handling: attempting unauthorized access wastes time and generates confusing errors. Detecting the gap early and explaining the authorization path (who needs to grant access, through what process) is more helpful. Offering partial help with authorized resources provides immediate value while authorization is obtained.

Q53. You are managing a conversation where Claude is helping a user with a technical problem. After 40 turns, the user introduces a new, unrelated problem. How do you handle context management optimally?

A) Continue in the same session — all context is potentially relevant

B) Summarize the resolution of the first problem, start a new session with the summary as 'completed context' and the new problem as the active task. This resets context usage while preserving the completed work for reference.

C) Delete all 40 turns and start completely fresh

D) Use a sub-agent for the new problem while keeping the main session active

■ Correct Answer: B

Session transition with context preservation: summarizing the completed problem compresses 40 turns into a few hundred tokens. The new session starts with the completion summary + new problem. Context usage resets. The user gets a fresh start for the new problem without losing the completed work record.

Q54. How do you implement a 'context budget allocation' strategy for a multi-phase agent task (research → analysis → report) with a 200K token limit?

A) Allocate tokens equally across phases

B) Allocate based on phase complexity and known output sizes: Research (tool results-heavy): 80K. Analysis (reasoning + tool results): 60K. Report generation (output-focused, less tool results): 60K. Implement inter-phase summarization to stay within allocation. Monitor and rebalance if a phase exceeds its budget.

C) No allocation needed — 200K is plenty for all three phases

D) Use separate sessions per phase with no cross-session token budget coordination

■ Correct Answer: B

Phase-based allocation: different phases have different token consumption profiles. Research consumes tokens in tool results (web pages, documents). Analysis consumes tokens in reasoning chains. Report generation produces output but needs less input context. Allocation + inter-phase summarization prevents one phase from crowding out subsequent ones.

Q55. A long-running agent discovers mid-task that it needs a piece of information it retrieved at the beginning of the session (now likely compressed or dropped). What is the best recovery strategy?

A) Restart the task from the beginning

B) Re-retrieve the information using the original tool call with the same parameters — don't assume it's lost. If re-retrieval is expensive, implement a key-data store: preserve important findings explicitly in a persisted context object, not just in conversation history.

C) Ask the user to re-provide the information

D) Proceed without the information — an approximation is sufficient

■ Correct Answer: B

Re-retrieval is often simpler than recovery: if the tool is available, just call it again. For expensive re-retrievals, the key-data store pattern is better: maintain a persisted JSON object of important findings that survives context compression. This is fundamentally more reliable than hoping important data survives in a compressed conversation history.

Q56. How do you design a multi-agent system that can explain its reasoning process to a non-technical user after completing a complex task?

A) Provide the full conversation transcript — that's the explanation

B) Maintain a decision journal throughout the task: record key decisions with lay-language explanations. At task completion, synthesize the journal into a narrative explanation: what was done, why each major step was taken, what was found, and how the conclusion was reached.

C) Ask Claude to explain itself after completing the task

D) Non-technical explanations require a separate specialized model

■ Correct Answer: B

Decision journal → narrative explanation: capturing decisions with lay-language rationale during execution is more reliable than post-hoc explanation (which may rationalize rather than accurately describe decisions). The journal's chronological structure naturally supports a narrative arc for non-technical users.

Q57. You have a multi-agent system where a slow subagent is causing the coordinator to idle, wasting its context budget. How do you address this?

A) Remove the slow subagent from the pipeline

B) While waiting for the slow subagent, the coordinator processes other completed subagent results (partially synthesize available findings), prepares for the slow subagent's expected output (pre-draft sections it can fill in), and optimizes its own context (compress older results). Use async patterns throughout.

C) Replace the slow subagent with a faster model

D) Coordinator should block synchronously — parallel processing is too complex

■ Correct Answer: B

Productive waiting: async coordination enables the coordinator to continue useful work while waiting. Partial synthesis, pre-drafting, and context optimization reduce total pipeline time. The coordinator's context is used productively rather than idling.

Q58. How do you manage context for a system where Claude must maintain awareness of a large (200-item) task checklist throughout a long workflow?

A) Include all 200 items in every API call

B) Maintain the checklist in an external store; inject only the current section (5-10 items) and summary statistics (done: 45/200, in_progress: 3, blocked: 2) into context per turn. Claude only needs the active items and overall progress, not all 200 items at once.

C) Store the checklist in Claude's memory feature

D) Break the workflow into 20 separate sessions, one per 10 items

■ Correct Answer: B

External state with focused context injection: the full checklist lives in a database. Each turn injects: current active items + progress summary. Claude has the information it needs to make progress without loading all 200 items. As items complete, they're archived (not in context). Progress statistics provide global awareness without full item enumeration.

Q59. How do you implement cross-session learning where an agent applies lessons from previous runs to improve future performance?

A) Each session is independent — cross-session learning requires fine-tuning

B) Maintain a lessons database: extract key insights from each session (what worked, what failed, what shortcuts were discovered). At the start of new sessions on similar tasks, retrieve and inject relevant lessons as structured context. This is RAG-based cross-session knowledge.

C) Include complete logs from all prior sessions in context

D) Use the same session permanently — never start fresh

■ Correct Answer: B

RAG-based cross-session knowledge: lessons are vectorized and stored. At task start, semantic search retrieves relevant lessons (similar task type, similar domain). Injecting 'In previous runs on similar tasks, we found: [lessons]' provides immediately applicable guidance. This achieves learning without fine-tuning.

Q60. A long-running agent task requires writing intermediate outputs to files periodically. What context management benefit does this provide?

A) File writes reduce API costs

B) External persistence of intermediate outputs means the agent can compress or drop those results from context (they're safely stored) while continuing with fresh context budget for remaining work

C) File writes are primarily for the user's benefit, not context management

D) File writes are not possible in agentic contexts

■ Correct Answer: B

Write-to-compress pattern: once a finding is written to a file, its detailed content can be replaced in context with a reference: 'See analysis_phase1.json for detailed findings.' The file preserves the data; the context reference maintains awareness without consuming the full token budget.

ADVANCED QUESTIONS

Questions 61–90

Q61. You are designing a long-running autonomous agent that processes tasks over multiple days. Each day the agent continues from where it left off. What is the complete state management architecture?

A) Use a single continuous session for the entire multi-day run

B) Daily checkpoint to persistent storage (task graph: completed/in-progress/blocked nodes, key findings, decisions log, tool result cache for stable data, open questions). Each day: load checkpoint, inject as initial context, continue. End-of-day: generate checkpoint, store.

C) Store the full conversation history and replay it each day

D) Use session resumption — it handles multi-day state automatically

■ Correct Answer: B

Production multi-day state management: checkpoint as structured task graph (not raw conversation), persist to external storage (not just in-context), cache stable tool results to avoid re-fetching, maintain a decisions log for audit. Each day loads a focused checkpoint, not the entire history. This scales to arbitrarily long-running tasks.

Q62. A real-time customer service agent is processing 1,000 concurrent conversations. Each conversation can last up to 2 hours. What context management architecture handles this at scale?

A) Run all 1,000 on the same Claude session

B) Per-conversation context store (database); sliding window of recent messages + key facts in-context; background summarization process compresses older turns; conversation checkpointing for recovery

C) Use a shared context for customers with similar issues

D) Limit conversations to 10 minutes to control context growth

■ Correct Answer: B

At scale: per-conversation state in a database (not in-memory), sliding window for recent context (most relevant), background compression of older turns (async summarization doesn't block the conversation), checkpointing for crash recovery. This architecture scales horizontally with conversation volume.

Q63. You are building a reliability architecture for a medical AI assistant. What combination of reliability patterns is necessary?

A) High model quality (claude-opus) is sufficient

B) Independent verification of key claims + confidence-scored outputs + mandatory human review above complexity threshold + comprehensive audit logging + escalation triggers for medical urgency indicators + explicit uncertainty communication to users

C) Multiple model ensemble for all responses

D) Retry-with-feedback until confidence reaches 0.95

■ Correct Answer: B

Medical AI reliability requires systemic architecture: independent verification (not self-review), explicit confidence and uncertainty, human oversight for complex cases, complete audit trails for accountability, urgency escalation paths, and explicit communication of AI limitations to users. Any single mechanism is insufficient for medical-grade reliability.

Q64. A production agentic system experiences occasional hallucinations where it confidently asserts incorrect facts sourced from tool results it misread. How do you systematically reduce this?

A) Use a higher quality model to reduce hallucinations

B) Implement: citation requirements (every claim must reference a specific tool result), cross-verification for critical facts, structured uncertainty markers in outputs, automated fact-checking against raw tool results post-generation

C) Increase retry count to catch hallucinations on rerun

D) Lower temperature to reduce creativity

■ Correct Answer: B

Structural anti-hallucination: citations link claims to sources (making hallucination auditable), cross-verification catches misreadings, uncertainty markers flag areas of low confidence, post-generation fact-checking validates claims against raw data. Temperature reduction helps but doesn't eliminate hallucinations from complex multi-source tasks.

Q65. You need to design a context-aware summarization strategy for a multi-agent system where different types of content have different retention requirements: legal citations (must preserve verbatim), numerical data (must be exact), and prose analysis (can be compressed).

A) Apply uniform compression to all content

B) Tiered summarization: legal citations → preserve verbatim in a citations store; numerical data → extract and store in a structured data object; prose analysis → summarize to key points. Context injection uses appropriate representation for each type.

C) Keep all content verbatim — context window has room

D) Have Claude classify content and summarize everything uniformly

■ Correct Answer: B

Content-type-aware retention: verbatim for legally sensitive content, structured for numerical precision, compressed for analysis prose. Different content types have different accuracy requirements — uniform compression inappropriately treats all content identically.

Q66. A financial services multi-agent system must produce a complete audit trail for regulatory compliance. What information must be captured for every agent decision?

A) Just the final output — regulators only care about results

B) For every agent action: timestamp, agent_id, action_type, inputs (sanitized for PII), outputs, tool_calls_made, human_approvals_received, compliance_rules_applied, and alternative_actions_considered_and_rejected

C) Model version and temperature settings

D) Session ID and total token count

■ Correct Answer: B

Regulatory audit trails require complete decision documentation: not just what was decided but what alternatives were considered and why they were rejected. Human approvals must be recorded. Compliance rules applied are essential for demonstrating rule-following. PII sanitization protects data while maintaining auditability.

Q67. You need to implement a system where an agent can delegate to different expert agents based on discovered context — it doesn't know which experts it will need until mid-task. How do you manage context across dynamic agent spawning?

A) Pre-load all possible expert agents at the start

B) Maintain a task context object that grows as discovery proceeds; when spawning an expert agent, inject only the relevant subset of task context for that agent's domain; expert outputs are merged back into the task context object with attribution

C) Give each expert agent the full task context

D) Start a new context for each expert agent — isolation is always correct

■ Correct Answer: B

Dynamic context injection: maintain a growing task context object, but inject only relevant subsets per expert (domain-specific context). This avoids overwhelming experts with irrelevant context while maintaining task coherence. Attribution on merge ensures findings are traceable to their source expert.

Q68. An agent is investigating a complex production incident. After 30 turns, it has strong evidence for root cause A but is exploring whether root cause B also applies. Context is at 140K tokens. What is the optimal strategy?

A) Commit to root cause A and stop exploring

B) Fork from the current session to explore root cause B; checkpoint the current findings (strong evidence for A); the fork explores B independently; merge findings if B is confirmed, report A-only if B is ruled out

C) Start completely fresh to explore root cause B

D) Continue in the main session — 140K still has 60K remaining

■ Correct Answer: B

Fork for parallel exploration: the fork inherits the 30 turns of context, explores B independently, and doesn't contaminate the main A-focused session. If B is confirmed, merge findings. If ruled out, report only A. Starting fresh loses 30 turns of valuable context.

Q69. A large-scale document processing system needs to process 100,000 documents over a weekend. Some documents may reference each other. How do you design the context management for cross-document references?

A) Process independently — cross-references are too complex to handle at scale

B) Two-pass architecture: Pass 1 (batch, parallel) — extract metadata and cross-reference declarations from all 100K documents. Pass 2 (dependency-ordered) — process documents with injected context from referenced documents. Cache extracted context for efficiency.

C) Process in document order and hope forward references are minimal

D) Use a single massive context with all 100K documents

■ Correct Answer: B

Two-pass architecture handles cross-references at scale: the first pass (metadata extraction) is parallelizable and cheap. The second pass processes in dependency order with injected cross-reference context. Caching extracted metadata avoids re-processing referenced documents. This scales to 100K documents efficiently.

Q70. You are designing a quality assurance system where multiple Claude instances evaluate each other's work. What architecture prevents echo chambers where all instances agree on incorrect outputs?

A) Use the same model for all instances — consistency is good

B) Use different system prompts that emphasize different evaluation criteria per instance; introduce adversarial instances whose role is to find weaknesses; require disagreement resolution through explicit reasoning; monitor inter-instance agreement rates to detect echo chambers

C) More instances always produce more accurate consensus

D) Use different temperature settings for diversity

■ Correct Answer: B

Echo chamber prevention requires structural diversity: different evaluation criteria force different perspectives. Adversarial roles explicitly look for failure modes. Disagreement resolution with explicit reasoning prevents silent capitulation. Monitoring agreement rates detects systematic bias. Temperature alone doesn't prevent correlated failures.

Q71. A production multi-agent system experiences a cascade failure where one subagent's incorrect output causes downstream subagents to produce compounding errors. What architectural pattern prevents this?

A) Run all subagents in isolation with no data sharing

B) Implement validation gates between pipeline stages: each subagent's output is validated for quality and consistency before being passed to the next; cascade is halted at the first validation failure

C) Use a single large agent instead of a pipeline

D) Implement retry logic in every subagent

■ Correct Answer: B

Validation gates as circuit breakers: stage N validates output from stage N-1 before consuming it. A bad output from the analysis stage triggers a halt and re-analysis rather than being passed to synthesis where the error compounds. Early detection is cheaper than cascade recovery.

Q72. What is 'context poisoning' and how do you prevent it in multi-turn conversations?

A) Caching incorrect tool results that get re-used in future calls

B) When incorrect information enters the conversation context (hallucinated fact, bad tool result) and subsequent reasoning builds on it, compounding the error. Prevention: validate tool results before adding to context; implement correction mechanisms when errors are detected.

C) Excessive context growth that reduces response quality

D) Prompt injection through tool results

■ Correct Answer: B

Context poisoning is subtle: Claude uses all context for subsequent reasoning. A hallucinated customer name in turn 5 may be used confidently in turns 10-20. Prevention requires: validate critical facts from tool results, implement explicit correction when errors are identified, and maintain a 'verified facts' separate from 'inferred facts'.

Q73. How should an agent handle a situation where it needs information that exceeds what can be included in a single subagent's context?

A) Include only the most recent information and discard older data

B) Implement RAG (retrieval augmented generation): store information in a vector database or document store; subagents retrieve relevant chunks via semantic search rather than loading all information into context

C) Increase the model's context window by using claude-opus

D) Process the task without the information and flag it as incomplete

■ Correct Answer: B

RAG for large information sets: instead of trying to fit everything in context, store in a retrieval system (vector DB, document store) and fetch relevant chunks on demand. The subagent calls a search tool for each piece of information it needs, consuming context only for what's relevant to the current reasoning step.

Q74. You need to implement graceful degradation when a subagent fails in a critical pipeline. The pipeline cannot continue without the subagent's output. What is the correct architectural response?

A) Return an error to the user immediately

B) Attempt recovery: retry with the subagent (1-2 times with refined context); if retry fails, attempt the task with a fallback strategy (alternative subagent, alternative approach); if all recovery fails, compile partial results with clear documentation of what's missing and why

C) Continue the pipeline without the failed subagent's output

D) Restart the entire pipeline from the beginning

■ Correct Answer: B

Layered recovery: retry (maybe it was transient), fallback (alternative approach), then graceful partial failure (report what's available and what's missing). Returning an error immediately without recovery attempts wastes successful work from other pipeline stages. Partial results with clear documentation are more useful than no results.

Q75. What is the 'telephone game' problem in multi-agent systems and how do you prevent it?

A) Latency accumulates as messages pass through multiple agents

B) Information degrades or distorts as it passes through multiple summarization steps between agents. Prevention: preserve original source citations through all handoffs; limit summarization depth; validate key facts against original sources at the synthesis stage

C) Agents copy each other's errors when they share context

D) Tool call results are duplicated across multiple agent contexts

■ Correct Answer: B

Information fidelity degradation: each summarization loses detail and introduces interpretation. After 3 summarization hops, a nuanced finding becomes a distorted approximation. Prevention: maintain original source citations through all handoffs so the synthesis agent can verify key facts against primary sources, not just summaries.

Q76. A real-time agent system must handle 500 concurrent users, each potentially in multi-turn conversations. What are the key reliability engineering considerations?

A) Use a single powerful server — distributed systems add complexity

B) Per-conversation state in external storage (not in-process memory); horizontal scaling with stateless agent instances; circuit breakers for external tools; conversation checkpointing for crash recovery; rate limiting per user to prevent single users from exhausting capacity

C) Limit concurrent users to 100 for quality control

D) Use only synchronous processing to ensure message ordering

■ Correct Answer: B

Production reliability at scale requires: stateless agent instances (state in DB enables horizontal scaling), circuit breakers (prevent cascade failures from flaky external tools), checkpointing (recover from server restarts), and per-user rate limiting (prevent resource exhaustion). In-process state doesn't survive failures or scale.

Q77. An agent is given a task with an ambiguous success criterion: 'improve the codebase quality'. How should it handle this?

A) Make as many changes as possible — more changes means more improvement

B) Before starting: define measurable success criteria in consultation with the user (what specific quality dimensions matter: test coverage, complexity metrics, documentation coverage?); set a scope limit; establish a completion definition

C) Make no changes — ambiguous tasks cannot be safely executed

D) Focus only on what's easiest to improve

■ Correct Answer: B

Operationalizing ambiguous tasks: 'improve codebase quality' is unmeasurable as stated. Clarification produces: 'increase test coverage from 45% to 70% for src/core/', 'reduce cyclomatic complexity above 10 in payment module', 'add docstrings to public API functions'. These are measurable, bounded, completable.

Q78. What is the most important reliability metric to track for a production multi-agent system handling financial transactions?

A) Average response time

B) Idempotency rate — what percentage of retried operations correctly detect they've already been completed and return the prior result rather than executing twice, preventing double-charges or double-processing

C) Model version consistency across agents

D) Tool call success rate

■ Correct Answer: B

Idempotency is critical for financial operations: network retries, agent restarts, and edge cases mean the same operation may be attempted multiple times. Idempotency keys ensure 'process refund REF-123' executes exactly once regardless of how many times it's attempted. Double-processing is a severe financial error.

Q79. You are designing a production system where multiple Claude instances process parallel workstreams that will be synthesized. What context handoff protocol minimizes synthesis errors?

A) Each instance dumps its full conversation history for synthesis

B) Structured result schema per instance: {findings: [{claim, confidence, source_reference}], gaps: [{topic, why_unresolved}], key_decisions: [{decision, rationale, implications}], recommended_synthesis_weight: 0.0-1.0}. Synthesis agent receives typed, structured data — not raw transcripts.

C) Run synthesis in the same session as one of the parallel instances

D) Average outputs where they conflict and take first result where they agree

■ Correct Answer: B

Typed structured handoffs for synthesis: the synthesis agent works with typed, validated data rather than raw text. Confidence per claim enables weighted synthesis. Gaps inventory tells the synthesizer what's unknown. Recommended synthesis weights let instances signal their own reliability. Structured data also enables automated validation before synthesis.

Q80. An agent system must maintain contextual continuity across 30 days of operation on a single long-running project. What architecture achieves this at scale?

A) One continuous session for 30 days

B) Daily checkpoint architecture: (1) end-of-day checkpoint to persistent store (project knowledge graph, decision log, task state), (2) nightly knowledge distillation (compress repetitive context into concise principles), (3) session start: load distilled knowledge + recent (last 3 days) checkpoints + current task state. Archives older checkpoints for audit but doesn't load them.

C) New session each day with no cross-day context

D) Maintain context via a continuously growing CLAUDE.md

■ Correct Answer: B

Long-term context management: daily checkpoints prevent context loss, knowledge distillation prevents checkpoint bloat (30 raw checkpoints would be huge), selective loading (recent + current) keeps context focused and manageable. Archives maintain the full history for audit without loading it all.

Q81. How do you design a fault-tolerant multi-agent system where any single agent can fail and the system continues operating with degraded but functional capability?

A) Implement redundancy: run each agent twice and take the first result

B) Fault-tolerant architecture: health checks per agent, automatic failover to backup agents for critical paths, graceful degradation with explicit capability reduction logging when non-critical agents fail, circuit breakers preventing cascade failures, coordinator recalculates resource allocation when agents fail

C) Design the system to require all agents or fail entirely

D) Use a single large capable agent instead of multiple specialized agents

■ Correct Answer: B

Fault tolerance through graceful degradation: system continues operating (reduced capability is better than no capability). Health checks detect failures early. Failover handles critical path failures. Circuit breakers prevent cascade failure propagation. Dynamic resource reallocation maximizes remaining capacity.

Q82. You are building a compliance-critical system where every piece of information used in a decision must be traceable to its source. How do you implement complete information provenance?

A) Log all API calls — that provides sufficient provenance

B) Chain-of-custody information model: every fact carries {value, source_id, retrieval_timestamp, confidence}. Decisions carry {decision, supporting_facts: [fact_id list], reasoning_chain, decision_timestamp}. Immutable audit store records everything. Queryable provenance graph enables: 'show me the source of every fact used in decision D'.

C) Include citations in the final report — that's sufficient for compliance

D) Provenance is the source system's responsibility — not the agent's

■ Correct Answer: B

Chain-of-custody provenance: every fact is tagged at collection. Decisions reference specific fact IDs. The provenance graph answers arbitrary compliance queries: 'what source data supported this conclusion?' Immutable storage prevents retroactive modification. This is the architecture for systems where information lineage has legal implications.

Q83. How do you design a system that handles 'context inheritance' correctly — where child agents should inherit some parent context but not all?

A) Give child agents the parent's full context — more context is always better

B) Explicit context partitioning: define inheritance rules per context type. Task constraints → inherit. Global facts → inherit. Parent's tool results → filter to relevant subset. Parent's conversation history → summarize, don't inherit verbatim. Agent's reasoning chains → do not inherit.

C) Child agents should have completely isolated context — no inheritance

D) Inherit based on token budget: fill child context up to the budget limit from parent context

■ Correct Answer: B

Selective inheritance by context type: task constraints and global facts are universally applicable (inherit). Tool results are often irrelevant to child tasks (filter). Conversation history is verbose (summarize). Reasoning chains from the parent can bias the child's reasoning (do not inherit). This matches context inheritance to information relevance.

Q84. A multi-agent medical diagnosis system must provide reliable diagnoses while acknowledging uncertainty appropriately. How do you design the uncertainty quantification?

A) Claude provides reliable diagnoses — explicit uncertainty quantification is not needed

B) Multi-level uncertainty architecture: epistemic uncertainty (lack of knowledge: 'insufficient symptoms for confident diagnosis'), aleatoric uncertainty (inherent ambiguity: 'these symptoms match multiple conditions'), knowledge cutoff uncertainty ('recent research may have updated treatment for this condition'). Each type has different handling: epistemic → request more information, aleatoric → differential diagnosis with probabilities, cutoff → recommend current literature review.

C) Use a single confidence score (0-1) for all uncertainty types

D) Only express uncertainty when confidence is below 50%

■ Correct Answer: B

Uncertainty type differentiation is clinically meaningful: epistemic uncertainty → more information needed (request symptoms/tests). Aleatoric → present differential diagnosis transparently. Knowledge cutoff → explicit currency warning. A single confidence score conflates these meaningfully different situations.

Q85. You are designing an agent that monitors a production system and takes corrective actions autonomously. What principles govern when to act vs. when to alert humans?

A) Always act autonomously for efficiency

B) Autonomous action criteria: action is reversible, impact is bounded and measurable, action type has high historical reliability, no cascading effects possible, recoverable within 5 minutes if wrong. Alert human criteria: irreversible, cascading risk, novel situation, regulatory implications, impact above defined threshold. Always log both actions and non-actions with full reasoning.

C) Always alert humans — production changes require human approval

D) Act autonomously during business hours; alert humans overnight

■ Correct Answer: B

Calibrated autonomy for production monitoring: reversible + bounded + proven + isolatable actions can be autonomous. Irreversible, cascading, novel, or regulated situations require human judgment. Logging both actions AND non-actions (with reasoning) creates the feedback loop for tuning autonomy thresholds over time.

Q86. How do you implement 'explanation-level calibration' in a multi-agent system where different stakeholders need different levels of explanation detail?

A) Generate one detailed explanation and let stakeholders extract what they need

B) Tiered explanation generation: technical stakeholders receive full reasoning chains with evidence citations; management receives executive summaries with key drivers; regulatory receives compliance-mapped explanations; end users receive plain-language summaries. Each tier is generated with a stakeholder-specific prompt that draws from the same underlying decision data.

C) Use a single explanation prompt and adjust verbosity with max_tokens

D) Let each stakeholder request the level of detail they want on demand

■ Correct Answer: B

Tiered explanations from shared data: the underlying decision and evidence are the same. Stakeholder-specific prompts render the same information at different levels of detail, technical depth, and regulatory framing. This avoids maintaining multiple parallel explanation systems while serving each stakeholder appropriately.

Q87. You are deploying a multi-agent system in a zero-trust security environment. Each agent must authenticate to access resources, even when called by another agent. How do you implement this without creating authentication bottlenecks?

A) Have agents share a single service account — internal communication is trusted

B) Per-agent identity with scoped credentials: each agent has a unique identity with credentials scoped to its specific role (read-only for analysis agents, write for execution agents). Short-lived tokens (15-minute TTL) with automatic refresh. Central auth service with response caching (1-minute TTL) to reduce auth overhead. Audit log all inter-agent auth events.

C) Use mutual TLS between all agents — no additional auth needed

D) Authenticate only at system entry point — internal calls are implicitly trusted

■ Correct Answer: B

Zero-trust inter-agent auth: per-agent identity enables granular audit trails, scoped credentials enforce least privilege even between agents, short-lived tokens limit compromise exposure, auth response caching reduces latency without compromising security. Shared service accounts prevent attribution of actions to specific agents.

Q88. How do you design a system where an agent can detect and recover from its own reasoning errors mid-task (before they cascade)?

A) Errors cannot be detected mid-task — only post-hoc review catches them

B) Embedded reasoning validation: at key decision points, agent runs a self-check: 'Does this conclusion follow from the evidence I have? Are there alternative explanations I haven't considered? Does this action align with the stated goals?' Flag low-confidence decisions for additional verification before acting.

C) Use a separate reviewer agent for all decisions

D) Only low-confidence decisions are likely errors — flag those automatically

■ Correct Answer: B

Embedded self-validation at key decision points: the agent pauses to critically evaluate its own reasoning. 'Have I considered alternatives? Does the evidence actually support this conclusion?' This is most valuable at irreversible action points and when confidence is moderate (high confidence may be overconfidence, low confidence is already flagged).

Q89. A production agentic system exhibits 'reasoning drift' — after long sessions, the agent's behavior gradually deviates from the original task specification. What is the cause and architectural fix?

A) This is normal LLM behavior — restart sessions frequently

B) Cause: accumulated context shifts the agent's implicit focus away from the original task. Fix: inject the original task specification prominently every N turns ('Original task: [task]. Progress so far: [summary]. Continue from here.'), use anchor prompts that re-ground the agent in the original goals.

C) Use temperature=0 to prevent behavioral drift

D) Reduce max_tokens to keep sessions shorter

■ Correct Answer: B

Anchoring against drift: as conversation grows, early task specifications receive less attention weight. Periodic re-injection of the task specification re-grounds the agent. Anchor prompts are especially important for long autonomous sessions where the agent has taken many turns without explicit human guidance.

Q90. How do you implement 'adaptive verbosity' in agent outputs — where the agent adjusts its explanation depth based on detected uncertainty and task criticality?

A) Always provide full verbose output — more information is always better

B) Verbosity rules in the agent's output schema: low_uncertainty + low_stakes = brief summary. High_uncertainty OR high_stakes = detailed reasoning with evidence citations. Novel situation = full explanation including what alternatives were considered. Map these rules to context fields (uncertainty_score, criticality_level) in the output schema.

C) Let the user specify verbosity preference

D) Verbosity is determined by max_tokens — no special design needed

■ Correct Answer: B

Adaptive verbosity based on decision properties: calibrated information density prevents both verbosity fatigue (low-stakes decisions don't need essays) and under-explanation (high-stakes uncertain decisions need full reasoning). Schema-embedded verbosity rules make this systematic rather than arbitrary.