databricks

Databricks Exam Guide

Databricks Certified Context Engineer Associate

Provide Exam Guide Feedback

Purpose of this Exam Guide

This exam guide gives you an overview of the exam and what it covers to help you determine your exam readiness. This document will get updated anytime there are any changes to an exam (and when those changes will take effect on an exam) so that you can be prepared. **This version covers the version available beginning July 29, 2026. Please check back two weeks before this time to make sure you have the most current version.**

Audience Description

The Databricks Certified Context Engineer Associate certification exam assesses an individual's ability to design, assemble, and govern the information that AI agent systems receive at inference time using Databricks.

This includes structuring instructions and system prompts and configuring retrieval systems such as Databricks AI Search to surface relevant knowledge at inference time. It also covers designing memory architectures using Lakebase and MLflow for state persistence across sessions, integrating agents with tools and data sources using protocols such as MCP, and managing context window constraints through compaction and trimming strategies. Governance of what enters the context relies on Unity Catalog's metadata layer, including data quality, PII handling, and policy enforcement. The exam also covers context strategies for multi-agent and long-horizon workflows, and evaluating context engineering decisions empirically to assess whether agents perform better or worse as context is tuned. Individuals who pass this certification exam can be expected to build and manage the information environment for AI agent systems on Databricks, ensuring agents receive the relevant enterprise context needed to perform their tasks reliably.

The exam covers:

1. Foundations of Context Engineering – 16%
2. System Prompt and Instruction Design – 9%
3. Knowledge Retrieval and Genie Configuration – 20%
4. Memory Architecture with Lakebase and MLflow – 18%
5. Tool Design, MCP, and Agent Context – 13%
6. Context Compression and Compaction – 11%
7. Multi-Agent and Long-Horizon Task Design – 13%

About the Exam

- Number of scored questions: approximately 45 multiple-choice or multiple-selection items*
- Time Limit: 90 minutes
- Registration fee: $200
- Delivery method: Online Proctored
- Prerequisite: None is required; related course attendance and six months of hands-on experience are highly recommended.
- Validity: 2 years.
- Recertification: Recertification is required every two years to maintain your certified status. To recertify, you must take the full exam that is currently live. Please review the "Getting Ready for the Exam" section on the exam webpage to prepare for taking the exam again.
- * Unscored Content: Exams may include unscored items to gather statistical information for future use. These items are not identified on the form and do not impact your score, and additional time is factored into account for this content

Recommended Preparation

- Self-Paced (available in Databricks Academy):
    - Generative AI Engineering with Databricks, with these courses:
        - Building Retrieval Agents On Databricks
        - Building Single-Agent Applications on Databricks
        - Generative AI Application Evaluation and Governance
        - Generative AI Application Deployment and Monitoring
        - Building Reliable Conversational Agents with Genie
    - Data Engineering with Databricks, with these courses:
        - Get started with Databricks for Data Engineering
        - Get Started with Data Governance
        - Data Governance at Scale
- Knowledge of LLM context windows and how context length affects model performance.
- Knowledge of prompt engineering, system prompt design, and few-shot example construction.
- Working knowledge of agent frameworks and the Model Context Protocol (MCP)
- Working knowledge of Databricks Agent Bricks, Semantic Search, and embedding models.
- Working knowledge of Lakebase for agent memory and MLflow 3 for evaluation.
- Working knowledge of Unity Catalog governance for agent tools and retrieval sources
- Relevant Databricks Documentation resources for Agent Bricks, Semantic Search, Lakebase, and Genie.

Exam outline

Section 1: Foundations of Context Engineering

- Given a described agent failure, identify the context management technique that would most directly address it.
- Given a Databricks agent design, identify which proactive context management strategies (e.g., minimal tool sets, just-in-time retrieval, tool result scoping) would reduce context window pressure before compaction becomes necessary.
- Diagnose context failure modes: context poisoning, context distraction, context confusion, and context clash, given an agent trace.
- Select the right tool in the Databricks product stack (Unity Catalog, Lakebase, MCP, MLflow 3) for a given scenario.
- Given a described agent configuration, identify which context elements are consuming disproportionate attention budget and select the change most likely to improve model focus.
- Given a Databricks agent task and its performance requirements, select the appropriate reasoning mode (standard, extended thinking, or reduced thinking) and justify the selection based on token budget constraints and context window impact.
- Given a Databricks agent operating over an extended interaction, identify the point at which context length is causing measurable degradation in retrieval accuracy or reasoning quality, and select the intervention that restores performance.

Section 2: System Prompt and Instruction Design

- Given a business domain, select and validate the instructions, sample questions, and trusted SQL assets that together produce a production-ready Genie space.
- Given a Databricks agent, a set of candidate few-shot examples, and a token budget, justify which examples to include or exclude based on their marginal contribution to agent performance (e.g., covering untested tool paths, demonstrating output structure, handling ambiguous inputs).
- Given a miscalibrated Databricks agent system prompt and its observed failure pattern, select the targeted revision strategy that resolves the failure with the least increase in token cost and maintenance burden.
- Given experiment tracking results comparing two system prompt configurations on a Databricks agent, determine whether the higher-token configuration is justified and identify the specific prompt element driving the cost-performance tradeoff.

Section 3: Knowledge Retrieval and Genie Configuration

- Given an underperforming Databricks agent and its associated Unity Catalog metadata, identify which missing or poorly specified metadata elements are causing the accuracy gap and select the configuration change that will have the highest impact on agent performance.

- Select which Unity Catalog objects (managed tables, views, parameterized queries, SQL functions) to curate into a Genie space for a given business domain.
- Given a Databricks agent with documented retrieval quality problems, diagnose which Databricks AI Search configuration is the root cause and select the remediation that most directly improves the signal quality of retrieved context.
- Design a RAG pipeline that retrieves chunks from a Unity Catalog-governed document corpus and injects them into agent context.
- Select an appropriate chunking strategy given document structure, embedding model context length, and the types of queries the agent will face.
- Given a described agent task and available data sources, select the context elements required for the agent to correctly scope and execute the task.
- Distinguish between pre-inference retrieval (embedding-based, up-front loading) and just-in-time agentic retrieval (tool calls, dynamic Delta table queries) and select the appropriate strategy for a given use case.
- Identify which retrieval failure mode is occurring by using MLflow eval logs and UC metadata, and select the Unity Catalog governance action that most directly resolves the failure.
- Given a Databricks agent deployment that will retrieve context from a Unity Catalog environment containing both authoritative and derived data assets, design a governance strategy that constrains the agent's retrieval space to authoritative sources before deployment.

Section 4: Memory Architecture with Lakebase and MLflow

- Given a Databricks agent exhibiting degraded performance because its context is populated from an inappropriate memory source (e.g., retrieving from long-term storage when the information exists in the current session, or relying on session history when the needed context requires cross-session persistence), identify the mismatch between the memory type being used and the information need, and select the correct memory strategy.
- Identify when a Delta-backed state object is required over an in-context scratchpad for an agent operating on a multi-step task.
- Given an agent retrieving memories persisted in Lakebase, identify whether Databricks AI Search or structured query retrieval is the appropriate mechanism for a specified query type.
- Given MLflow 3 experiment results across multiple agent runs, identify which context configuration produced the most reliable outcomes on a specified task.
- Evaluate the tradeoffs of static retrieval vs. dynamic retrieval from Lakebase for a given agent architecture.
- Diagnose risks of over-retrieval (context pollution) and under-retrieval (missing relevant history) in a memory system.
- Configure persistent agent memory across sessions using Lakebase-backed durable store.

- Given a Databricks agent pipeline that produces accurate but contextually mismatched responses, identify the pipeline stage where user intent should be resolved before context retrieval.

Section 5: Tool Design, MCP, and Agent Context

- Apply a progressive-disclosure approach to MCP tool access (staged tool discovery with context-efficient execution) to reduce token usage compared to raw tool dumps.
- Given two MCP tool descriptions that an agent is consistently confusing, identify the overlap in their descriptions that is causing ambiguous tool selection.
- Explain how progressive disclosure can help control how much tool information enters the agent context window at each stage, and why this staged approach reduces token consumption compared to loading full tool schemas upfront.
- Given an agent with a deep message history where the context window is approaching capacity, evaluate which raw tool outputs are candidates for clearing.
- Given a described agent task and a set of tools registered in Unity Catalog (with names, descriptions, and parameter schemas), identify the most appropriate tool and justify the selection based on functional fit, input/output compatibility, and task requirements.
- Given a Databricks agent whose system prompt is overloaded with rarely-invoked capability instructions, identify which capabilities are candidates for packaging as Agent Skills and select the loading strategy that minimizes baseline context cost without harming task success rate.

Section 6: Context Compression and Compaction

- Given a long running agent task that has been compacted and is now exhibiting downstream coherence failures, identify which category of information was incorrectly discarded during compaction and select the compaction prompt revision that preserves that category without significantly increasing the token cost of the summarized context.
- Tune a compaction prompt for a Databricks agent trace: maximize recall first (capture everything relevant), then iterate to improve precision (remove superfluous outputs).
- Given a long-running Databricks agent task where context window pressure is building, evaluate whether hard-coded trimming heuristics are sufficient for the task's information relevance pattern, or if it requires a more sophisticated compaction approach.
- Given a Databricks agent trace, identify which content is safe to remove during compaction without affecting downstream task execution.
- Evaluate the tradeoffs between aggressive compaction (lower token cost, risk of losing subtle context) and conservative compaction (higher fidelity, higher cost).

Section 7: Multi-Agent and Long-Horizon Task Design

- Diagnose failure modes in multi-agent systems caused by insufficient shared context: inconsistent outputs, conflicting decisions, degraded reliability.

- Given a multi-agent system where a coordinating agent has decomposed a task and dispatched sub-agents, and the system is now exhibiting downstream failures, diagnose the root cause (i.e., sub-agents receiving individual task messages rather than full agent traces at dispatch time), and select the configuration that resolves the failure without expanding each sub-agent's context window.
- Given a multi-agent system producing conflicting outputs, identify the context propagation change that would most likely prevent the conflict.
- Given a multi-agent Databricks workflow where the orchestrating agent is experiencing context saturation, identify the sub-agent output design change that would most reduce orchestrator context load without compromising task coherence.
- Given a multi-agent system design where agent boundaries have been drawn and the resulting architecture is either exhibiting excessive handoff compression overhead or unmanageable context window growth within individual agents, diagnose which boundary placement failure is occurring.
- Given a long-running Databricks agent task with documented performance failures, identify which long-horizon strategy mismatch is causing the observed failure, select the replacement strategy most appropriate for the task's dependency structure, and justify the selection by identifying the specific characteristic that makes the original strategy insufficient.

Sample Questions

Question answers are available at the end of this document.

Question 1

Objective: Diagnose context failure modes: context poisoning, context distraction, context confusion, and context clash, given an agent trace.

A context engineer is auditing a Databricks agent built for a legal document review workflow. The system prompt instructs the agent to answer questions using only the retrieved contract clauses stored in a Unity Catalog volume. During trace inspection, the engineer observes the following steps:

Step 1: The agent calls the retrieval tool and receives Clause 4.2 (limitation of liability) from the current contract.
Step 2: The agent calls the retrieval tool again and receives Clause 7.1 (termination conditions) from the current contract.
Step 3: The agent calls the retrieval tool a third time and receives Clause 9.3 (payment obligations) from the current contract.

Step 4: The context window also contains a full conversation history from a prior session involving a software licensing agreement with different obligation structures.
Step 5: The agent's final response correctly cites Clause 4.2 but then drifts into referencing indemnification and IP ownership terms that appear nowhere in the retrieved clauses, closely mirroring the prior session's licensing agreement content instead.

Which context failure mode does this agent trace exhibit?

A. Context distraction, because the irrelevant prior session content in the context window grew so long that the model over-focused on the context, neglecting what it learned during training.
B. Context confusion, because the agent failed to reconcile which retrieval results were most important, and the superfluous content was used to generate a low-quality result.
C. Context poisoning, because the prior session content introduced factually incorrect information that corrupted the agent's grounding instructions, where it was repeatedly used.
D. Context clash, because the system prompt and the retrieved contract clauses issued contradictory instructions about which obligations to apply, leading to conflicting information in the context.

Question 2
Objective: Given a Databricks agent operating over an extended interaction, identify the point at which context length is causing measurable degradation in retrieval accuracy or reasoning quality, and select the intervention that restores performance.

A context engineer operates a customer support agent that handles multi-turn conversations exceeding 100 tool calls per session. MLflow traces over the past week show that as conversations approach roughly 75% of the model's context window, the agent begins to mis-cite information from earlier tool results, recall the wrong policy from documents retrieved earlier in the same session, and produce reasoning that ignores constraints stated in prior turns. The agent has not yet hit the model's hard token limit, and the team wants to keep each conversation in a single uninterrupted session.

Which configuration addresses the degradation in reasoning quality without discarding previously retrieved facts or reducing the agent's retrieval capability?

A. Upgrade to a model with a larger context window so the agent can hold more turns in memory before the attention budget is depleted, restoring recall accuracy on earlier tool results.
B. Implement a sliding-window truncation strategy that drops the oldest messages once the context exceeds 75% capacity, so the active window always stays within the model's comfortable range.

C. Disable retrieval tool calls after a fixed turn count so accumulated tool outputs no longer expand the context window, accepting that no new policy lookups can occur for the remainder of the session.
D. Enable conversation compaction so the message history is summarized into a high-fidelity summary, and the agent continues with that summary plus the most recently used items.

Question 3
Objective: Given a business domain, select and validate the instructions, sample questions, and trusted SQL assets that together produce a production-ready Genie space.

A data analyst has built a Genie space for a corporate treasury team that tracks cash balances, hedging positions, and settlement obligations. The engineer has added the relevant tables with accurate column descriptions, registered trusted SQL assets for the team's regulatory reporting queries, and configured SQL expressions for key financial metrics. During validation, Genie handles quantitative questions correctly but consistently misinterprets domain-specific terminology: when analysts ask about "the rates desk," Genie searches for a column named **`rates_desk`** instead of filtering the positions table by the correct **`product_type`** and **`business_unit`** values; when they request figures "in bps," Genie does not recognize the abbreviation as basis points; and when they reference "last fixing," Genie does not understand this means the most recent ECB reference rate publication date in the rates table.

What should the engineer do to resolve these interpretation failures?

A. Register example SQL queries as trusted assets for the most common treasury questions using the team's domain-specific phrasing, so Genie returns trusted results when analysts use terms like "rates desk" or "last fixing."
B. Add SQL expressions in the knowledge store defining "rates desk," "bps," and "last fixing" as filters and measures so Genie applies the correct logic automatically when those terms appear in questions.
C. Add general instructions that define the treasury team's domain-specific vocabulary, so Genie has the business context to correctly interpret analyst questions.
D. Add sample questions written in the treasury team's preferred vocabulary so Genie learns the correct domain terminology from the example phrasing.

Question 4
Objective: Given a Databricks agent with documented retrieval quality problems, diagnose which Databricks AI Search configuration is the root cause and select the remediation that most directly improves the signal quality of retrieved context.

An AI engineer is investigating a Databricks agent deployed for a corporate legal team. The agent uses a **`DELTA_SYNC`** Databricks AI Search index with **`pipeline_type`** set to TRIGGERED to search

regulatory opinions and contract clauses. The source Delta table is refreshed each morning by an automated pipeline that ingests newly published opinions and clause amendments from the firm's document management system. During a compliance review, attorneys found that the agent cited a contract clause that had been amended three weeks earlier; the source Delta table contained the current amended text, but the agent's retrieved passage matched the original, pre-amendment language.

Which Databricks AI Search configuration change most cost-effectively resolves this retrieval discrepancy?

A. Create a Databricks job that calls sync on the index after the daily ingestion pipeline completes, so that the index reflects each day's updates.
B. Switch **`pipeline_type`** from TRIGGERED to CONTINUOUS so the index incrementally syncs with seconds of latency as the source Delta table changes.
C. Replace the **`DELTA_SYNC`** index with a **`DIRECT_ACCESS`** index and add programmatic upsert calls to the document ingestion workflow.
D. Drop and recreate the Databricks AI Search index from the current Delta table snapshot to incorporate all pending amendments.

Question 5
Objective: Select an appropriate chunking strategy given document structure, embedding model context length, and the types of queries the agent will face.

A context engineer is building a Databricks-hosted RAG agent to answer questions over a large corpus of internal wiki articles. The articles vary widely in length, from short 200-word FAQs to multi-section technical guides exceeding 3,000 words, and have no consistent heading structure. The embedding model has an 8,192-token context window. User queries range from broad conceptual questions like "How does our deployment pipeline work?" to narrow lookups like "What is the retry limit for the job scheduler?"

Which chunking strategy should the context engineer apply to handle this variability in document length and query type?

A. Apply fixed-size chunking at 512 tokens with no overlap, relying on the model's large context window to compensate for any mid-sentence splits.
B. Chunk each article as a single document, since the 8,192-token context window is large enough to embed entire articles without truncation.
C. Apply sentence-level chunking where each sentence is a separate chunk, maximizing granularity so narrow lookup queries always find an exact match.
D. Use a recursive character text splitter with a target chunk size of 512–1,024 tokens and 10–15% overlap to preserve context across chunk boundaries for broad and narrow queries.

Question 6
Objective: Given MLflow 3 experiment results across multiple agent runs, identify which context configuration produced the most reliable outcomes on a specified task.

A context engineer ran a multi-turn question-answering agent across four context configurations in a single MLflow 3 experiment. Each configuration varied the retrieval chunk size, the number of documents retrieved, and whether a system prompt summarizing the retrieved context was included. For every run, the engineer logged three quality metrics: **`answer_correctness`**, **`context_recall`**, and faithfulness.

The engineer now needs to identify the configuration that produced the most reliable outcomes on the question-answering task. For this task, a reliable configuration is one that scores well on all three metrics together, with no single metric lagging, rather than one that excels on one metric while underperforming on another.

Which approach should the engineer use to identify that configuration?

A. Rank the configurations by the arithmetic mean of the three metrics, and select the one with the highest mean.
B. Use the MLflow parallel coordinates chart to compare the runs and select the configuration that stays high on every metric axis.
C. Compute the spread (range or standard deviation) of the three metric scores for each configuration, and select the one with the smallest spread as the most consistent.
D. Select the configuration that achieves the single highest metric value observed across all runs, and treat it as the top performer.

Question 7
Objective: Configure persistent agent memory across sessions using Lakebase-backed durable store.

A context engineer is building a multi-turn research assistant agent using LangGraph on Databricks. The agent runs inside an async Python execution environment and must persist each conversation thread's accumulated state to a Lakebase-backed PostgreSQL store so that sessions can be resumed after interruption. The engineer has obtained a valid async database connection pool from the Lakebase instance and must now instantiate the correct checkpointer class to register with the compiled LangGraph graph.

Which approach should the engineer use to instantiate and register the checkpointer?

A. Instantiate **`AsyncPostgresSaver`** with the async connection pool, call **`checkpointer.setup()`** synchronously because the setup step must run synchronously, then pass the instance to **`graph.compile(checkpointer=checkpointer)`**.
B. Instantiate **`AsyncCheckpointer`** with the async connection pool, call **`await checkpointer.setup()`** and configure it to flush state to the Lakebase PostgreSQL endpoint by setting the **`connection_string`** parameter then passing it to **`graph.compile()`**.
C. Instantiate **`AsyncPostgresSaver`** with the async connection pool, call **`await checkpointer.setup()`** to initialize the schema, then pass the instance to **`graph.compile(checkpointer=checkpointer)`**.
D. Instantiate **`AsyncCheckpointer`** with the async connection pool and pass it directly to **`graph.compile(checkpointer=checkpointer),`** relying on auto-migration at first write to manage state schema.

Question 8
Objective: Apply a progressive-disclosure approach to MCP tool access (staged tool discovery with context-efficient execution) to reduce token usage compared to raw tool dumps.

A context engineer is building a customer support agent on Databricks using a progressive disclosure approach to MCP tool access. The MCP server registers 60 tools, including four tools with nearly identical names and overlapping descriptions: **`fetch_order_status`**, **`get_order_status`**, **`retrieve_order_info`**, and **`lookup_order_details`**. When a user asks about a delayed shipment, all four ambiguous tool schemas are surfaced to the model for selection, causing the LLM to consume excessive tokens evaluating redundant candidates and producing inconsistent tool selections across identical queries.

Which approach should the context engineer implement to address this problem?

A. Enrich each tool's metadata with distinct capability descriptors in the MCP tool registry, then have the tool-discovery step score candidates against the user's intent signal and surface only the single best-matching tool schema for selection.
B. Apply semantic similarity thresholding during tool discovery to cluster the four tools by embedding distance, then surface the centroid-representative schema from the highest-scoring cluster for selection.
C. Consolidate the four overlapping tools under a single registry namespace entry, exposing one canonical schema with a parameter that specifies the lookup variant, so the model always receives one schema for order queries.
D. Add a tool-ranking step that scores all four schemas against the user's intent using a lightweight classifier, but only after all four have already been loaded into context for selection, then passes only the top-ranked schema to execution.

Question 9

Objective: Given a long running agent task that has been compacted and is now exhibiting downstream coherence failures, identify which category of information was incorrectly discarded during compaction and select the compaction prompt revision that preserves that category without significantly increasing the token cost of the summarized context.

A context engineer is overseeing a long-running multi-step research agent that processes regulatory compliance documents across dozens of sequential tool calls. After compaction, the agent begins producing recommendations that contradict decisions made earlier in the session. Inspection of the compacted context shows that the summarization prompt collapsed all prior conversational turns into a single factual summary of findings. The compaction prompt currently reads: **`Summarize the key facts and findings discovered so far.`**

Which revised compaction prompt addresses the root cause of the coherence failures without significantly increasing token cost?

A. Summarize key findings and append a structured log of each user message from prior turns, preserving original wording.
B. Summarize key findings and include the outputs of all tool calls made during the session.
C. Summarize key findings and include a record of the agent's internal chain-of-thought reasoning from each prior step.
D. Summarize key findings and include a deduplicated list of constraints, exclusions, and preferences the user has stated.

Question 10

Objective: Given a multi-agent system where a coordinating agent has decomposed a task and dispatched sub-agents, and the system is now exhibiting downstream failures, diagnose the root cause (i.e., sub-agents receiving individual task messages rather than full agent traces at dispatch time), and select the configuration that resolves the failure without expanding each sub-agent's context window.

A context engineer is investigating a multi-agent pipeline on Databricks where a coordinating agent decomposes a supply chain demand forecasting task and dispatches work to five sub-agents, each responsible for a different product category. After a major inventory dataset update is ingested, the sub-agents begin producing forecasts that are mutually consistent but systematically lower than expected. The coordinating agent's reasoning log confirms it incorporated the updated figures, yet each sub-agent's retrieved context references the same set of supplier lead times and stock quantities that were valid before the update. The embedding model version and schema are unchanged.

Which root cause explains the sub-agents' behavior and which configuration change resolves it?

A. The coordinating agent dispatches only individual task messages without attaching its updated reasoning trace, so sub-agents lack visibility into the revised figures; attaching a compressed agent trace summary to each dispatched message resolves the gap.
B. The vector search index was not refreshed after the inventory update, so sub-agents are still retrieving pre-update documents; running an index sync against the updated Delta table ensures they retrieve the current figures.
C. The embedding model produces representations that are insensitive to numerical magnitude changes in inventory fields, so retrieved documents are semantically stale; reindexing after fine-tuning the embedding model on updated inventory records corrects retrieval accuracy.
D. The inventory update populated a new Delta table that the sub-agents are not configured to query, causing them to keep reading from the pre-update table; updating each sub-agent's data source configuration to point at the new table restores correct output.

Answers

1. B
2. D
3. C
4. A
5. D
6. B
7. C
8. A
9. D
10. B