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 beta version to be offered onsite at Databricks Data + AI Summit, June 16–18, 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 Vector 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 90 multiple-choice or multiple-selection items*
- Time Limit: 120 minutes
- Registration fee: ~~$200~~ One free attempt will be offered onsite at Summit.
- Delivery method: Live Proctored
- Beta Note: No result will be available at the end of the exam during Summit. Test takers will be notified of results approximately six weeks later.
- 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.
- Beta question count: This beta contains more questions than the final exam, which is normal. Additional time is factored in to account for the larger question count.

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 set of candidate few-shot examples for a Databricks agent, evaluate each against canonical coverage criteria and select the minimal set that improves agent performance without inflating token cost.

- 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 Vector 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 that is exhibiting degraded performance due to memory type mismatch, identify the mismatch and select the storage configuration that correctly aligns memory type with scope, retrieval pattern, and persistence requirements.
- 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 Vector 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 the Databricks layered MCP architecture (discovery → planning → 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 the Databricks MCP layered architecture uses progressive disclosure to 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, identify which tool in a Unity Catalog-registered tool registry is most appropriate based on semantic similarity to the task description.
- 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.