Databricks Context-Engineer-Associate Dumps

Shared by Hernandez on 10-11-2026

For More Free Questions and Preparation Resources

Check the Links on Last Page

Question 1

Question Type: MultipleChoice

A context engineer is evaluating a Databricks customer support agent that handles complex multi-turn conversations involving subscription changes, billing adjustments, and service cancellation requests.

The agent uses Lakebase-backed session storage to maintain conversation history. MLflow 3 tracing is enabled, and each inference trace includes a session identifier.

During testing, the engineer observes the following behavior:

Turn 1: The customer requests cancellation of an existing subscription.

Turn 2: The agent correctly identifies the subscription and explains the cancellation procedure.

Turn 4: The customer specifies that cancellation must occur at the end of the current billing cycle.

Turn 6: The agent retrieves the relevant cancellation policy and correctly answers a billing question.

Turn 8: The agent generates a final confirmation stating that the subscription will be canceled immediately, overlooking the customer's earlier instruction to cancel at the end of the billing cycle.

Individual response evaluations show high correctness and relevance scores. However, the operations team suspects that the agent is failing to preserve important information across conversational turns.

The engineer needs an evaluation mechanism that detects failures involving information introduced earlier in a conversation, rather than scoring each response independently.

Which MLflow evaluation approach should the context engineer implement?

Options:

A- Run the built-in Correctness scorer independently on each response trace, increasing the number of test questions to improve statistical confidence in the results.
B- Evaluate only the final response against the latest retrieved policy document, since the agent's correctness should depend primarily on the most recent authoritative source.
C- Increase the number of MLflow traces collected per conversation and compare average token usage, tool-call count, and latency between successful and unsuccessful sessions.
D- Group traces by their conversation session identifiers and run MLflow's multi-turn KnowledgeRetention judge to evaluate whether information introduced in earlier turns is correctly maintained throughout the conversation.

Answer:

D

Explanation:

The failure concerns conversational knowledge retention rather than the correctness of an isolated response.

The agent correctly processes individual requests but fails to retain the customer's previously established cancellation timing.

Single-turn evaluation can miss this problem because it examines responses independently, potentially overlooking information that appeared several interactions earlier.

Option D uses MLflow's multi-turn evaluation capabilities to assess the conversation as a complete sequence.

Databricks documents session-level evaluation through MLflow, allowing traces associated with the same conversation identifier to be grouped and assessed together.

The KnowledgeRetention judge specifically evaluates whether an agent correctly retains information established earlier in a conversation.

In this scenario, the evaluator can determine whether the final cancellation confirmation preserves the customer's instruction to cancel at the end of the billing cycle.

Option A may improve single-turn measurement but does not directly evaluate cross-turn consistency. Option B ignores the earlier customer instruction. Option C collects useful operational metrics without measuring the missing information.

The engineer should construct evaluation scenarios containing explicit preference changes, delayed instructions, clarifications, and contradictory later responses.

Databricks documents multi-turn evaluation as an experimental capability requiring MLflow 3.10 or later and appropriately instrumented session identifiers.

Official Databricks reference: Evaluate conversations, Built-in LLM judges, and Evaluate and improve.

Question 2

Question Type: MultipleChoice

A context engineer manages a personalized travel planning agent deployed on Databricks. The agent uses Databricks Agent Memory backed by Lakebase to retain durable user preferences across independent conversations.

Memory entries are organized using actor_id, session_id, and path.

During an earlier conversation, User42 stated that they preferred economy-class flights. The application persisted this preference in the memory store using:

actor_id: User42

path: /preferences/flight-class.md

content: Prefers economy-class flights.

session_id: Unset

Three months later, User42 explicitly changes their preference to business-class flights for future travel. The agent must retain the updated preference across all subsequent sessions without creating competing entries that could cause contradictory recommendations.

The application must also ensure that User42's preference does not affect memories belonging to other users.

Which memory-management operation should the context engineer configure?

Options:

A- Append the new business-class preference as another memory entry under a unique path, retaining both preferences so semantic retrieval can select whichever entry is most relevant to future travel requests.
B- Use memory_store.put() with the existing actor_id and memory path, leaving session_id unset, to replace the previously stored preference with the newly confirmed business-class preference.
C- Create a new memory entry using the latest session identifier and the existing preference path, allowing the agent to retrieve the updated value through session-scoped memory search.
D- Retain the original economy-class preference in the durable memory store and place the new business-class preference exclusively in the current conversation's short-term session context.

Answer:

B

Explanation:

The correct approach is to update the existing durable memory entry rather than introduce

additional conflicting records.

Databricks Agent Memory stores persistent information using an identity structure that includes actor_id, optional session_id, and path.

An entry is uniquely identified by the combination of these attributes.

For preferences intended to persist across conversations, the session_id should remain unset. This allows the entry to represent user-level knowledge rather than information associated with only one session.

Option B uses memory_store.put() to update the existing memory at the same actor and path.

Databricks documents that add() fails with ALREADY_EXISTS when an entry already exists at the specified location, whereas put() can create or update an entry.

Reusing /preferences/flight-class.md replaces the outdated preference with the user's newly confirmed choice.

Option A risks retaining contradictory information in separate entries. Option C creates session-specific memory rather than updating the durable cross-session preference. Option D loses the updated preference when the active session ends.

The application should also restrict memory retrieval and updates to the authenticated user's actor_id.

This design supports consistent personalization while preventing stale preferences from contaminating future recommendations.

Official Databricks reference: Agent memory --- Memory Entries, Retrieval, and the memory_store.put() operation.

Question 3

Question Type: MultipleChoice

A context engineer is building a regulatory compliance agent on Databricks that connects to an external compliance management platform through an MCP Service registered in Unity Catalog.

The MCP server exposes the following tools:

search_compliance_records: Retrieves existing compliance records without modifying them.

get_submission_status: Checks the status of previously submitted reports.

submit_regulatory_report: Sends a finalized regulatory report to an external authority.

delete_compliance_record: Permanently deletes an existing compliance record.

The agent serves compliance officers who are authorized to retrieve records and submit reports. However, organizational policy requires explicit human approval before every regulatory report submission.

The agent must continue executing read-only tools without interruption. It must also prevent unauthorized destructive operations and maintain an auditable record of tool invocations.

The engineering team has confirmed that the MCP tools function correctly. The remaining requirement is to enforce the approval and authorization rules at the platform governance layer rather than relying exclusively on instructions inside the agent's system prompt.

Which approach should the context engineer implement?

Options:

A- Add instructions to the system prompt requiring the agent to ask the user for confirmation before invoking submit_regulatory_report, and instruct the agent never to invoke delete_compliance_record.
B- Increase the model's reasoning effort so the agent evaluates the consequences of each MCP invocation before deciding whether user approval is necessary.
C- Configure the MCP Service with appropriate Unity Catalog permissions and tool restrictions, and attach an input-phase service policy that requires approval for submit_regulatory_report while allowing authorized read-only calls.
D- Remove all tools capable of modifying external systems from the MCP server and require compliance officers to submit regulatory reports through an independent manual process.

Answer:

C

Explanation:

Option C provides governance enforcement at the service boundary rather than depending on probabilistic agent behavior.

Databricks supports registering external MCP servers as Unity Catalog securable objects. This allows administrators to control which principals may access the service and which tools are exposed.

For this scenario, the engineer should grant the appropriate service permissions to authorized compliance officers and restrict the exposed tool set to approved operations.

Databricks service policies can inspect MCP requests before execution through the ON CALL

evaluation phase.

A policy can return an ASK decision, requiring user approval before the requested operation proceeds.

The engineer should configure this behavior specifically for submit_regulatory_report, while permitting authorized read-only operations without unnecessary approval prompts.

The destructive delete_compliance_record operation should be excluded or denied according to the organization's authorization requirements.

Option A depends entirely on instruction following and cannot reliably enforce privileged operations. Option B changes reasoning behavior without adding authorization controls. Option D unnecessarily removes the legitimate report-submission capability.

Databricks also supports auditing MCP activity, enabling administrators to review tool usage and investigate unauthorized or unexpected actions.

Service policies are currently documented as a Beta capability, so the relevant preview must be enabled before this mechanism can be used.

Official Databricks reference: Service policies for AI securables, Connect agents to third-party tools with MCP Services, and AI governance guide.

Question 4

Question Type: MultipleChoice

A context engineer is developing a Databricks legal research agent for an international financial services organization. The agent uses a DELTA_SYNC Databricks AI Search index containing regulatory compliance documents for the United States, United Kingdom, and European Union.

Each document chunk includes the following metadata:

jurisdiction: The regulatory jurisdiction to which the document applies.

document_status: Whether the document is active, superseded, or archived.

effective_date: The date on which the regulatory provision became effective.

document_id: The unique identifier of the source document.

The agent uses semantic similarity search with num_results=10 to retrieve regulatory evidence.

During MLflow evaluation, the engineer discovers that questions about current UK financial reporting obligations frequently retrieve superseded UK regulations and highly relevant US regulatory documents. The resulting answers are technically plausible but cite requirements that

do not apply to the user's jurisdiction or requested reporting period.

The embedding model produces accurate semantic similarity scores, and increasing num_results does not improve regulatory correctness.

The engineer must preserve semantic retrieval while ensuring that only regulations applicable to the requested jurisdiction and reporting date are considered.

Which retrieval configuration should the context engineer implement?

Options:

A- Increase num_results from 10 to 50 and apply a reranker so that the most semantically relevant regulatory documents appear first, regardless of their original jurisdiction.
B- Apply metadata filters for the requested jurisdiction, active regulatory status, and applicable effective-date conditions as part of the AI Search query, then rank the eligible documents by semantic relevance.
C- Replace semantic retrieval with full-text search so that exact jurisdiction names and regulatory terminology determine which documents are returned.
D- Retrieve all semantically relevant documents first and instruct the LLM through the system prompt to ignore regulations belonging to other jurisdictions or historical reporting periods.

Answer:

B

Explanation:

The observed failure results from a mismatch between semantic relevance and regulatory applicability.

Semantic similarity measures how closely document content relates to a query. It does not independently establish whether a regulatory document applies to a specific jurisdiction, reporting period, or effective date.

Consequently, a highly relevant US regulation may rank above an applicable UK regulation when both discuss similar financial reporting obligations.

Option B resolves this problem through metadata-constrained retrieval.

Databricks AI Search supports query filters that restrict results according to indexed metadata fields. The engineer can filter documents by jurisdiction and applicable regulatory status before incorporating their contents into the agent's context.

Effective-date filtering should account for the requested reporting date and the organization's regulation-versioning rules. A document marked active today is not necessarily the regulation

that governed an earlier reporting period.

After applying the eligibility constraints, semantic retrieval can identify the most relevant passages from the remaining documents.

Option A increases candidate volume without correcting applicability. Option C sacrifices valuable semantic matching. Option D relies on generation-time instructions to compensate for incorrect retrieval.

The engineer should also enforce access permissions independently; metadata filters improve retrieval correctness but are not a substitute for authorization.

Official Databricks reference: AI Search filtering guide and Improve RAG chain quality --- Metadata Filtering.

P2P exams

Question 5

Question Type: MultipleChoice

A context engineer is evaluating a target-landscape research agent at a pharmaceutical R&D organization. Discovery scientists ask the agent to prepare a structured landscape memo about a proposed drug target: what has been studied, who else is in the space, at what clinical phase, and with what regulatory history.

The agent runs as a single agent on Agent Bricks Custom Agents. It retrieves from a Databricks AI Search index built over PubMed, ClinicalTrials.gov, internal program memos, and FDA review documents, and produces the memo in one pass within a single context window.

Narrow queries (one target, one indication) produce accurate memos. Broader landscape queries return with competitor programs missing that the agent had named earlier in its own reasoning, quoted evidence attached to the wrong study, and cross-document fields partially filled. The MLflow trace shows window utilization rising toward the cap before drafting begins. One agent runs throughout.

What change should the AI engineer make to the agent architecture's boundaries?

Options:

A- Split the agent along the document-source axis (PubMed, ClinicalTrials.gov, internal memos, FDA reviews) and merge per-source outputs in the orchestrator.
B- Keep the single-agent boundary and lower the within-agent compaction trigger threshold so compaction fires before the memo is drafted.
C- Split the agent into sub-agents that scope exploration separately from synthesis, with

summaries (not raw evidence) crossing the boundary into the synthesis step.
D- Split the agent into a chain of single-step sub-agents (parse, retrieve, summarize, cluster, verify, cite, draft) connected by sequential handoffs.

Answer:

C

Explanation:

The architecture fails because a single agent is responsible for extensive evidence exploration, cross-document interpretation, and final report synthesis within one increasingly saturated context window.

For narrow queries, the available context remains sufficient. Broader landscape analyses introduce substantially more research findings, competitor programs, clinical development details, and regulatory evidence.

As the context approaches capacity, the agent becomes more likely to omit previously discovered entities, confuse supporting citations, and produce incomplete structured fields.

Option C introduces a functional boundary between exploration and synthesis.

Exploration sub-agents can independently investigate relevant scientific literature, clinical trials, competitor development programs, and regulatory records.

Their results should be condensed into structured summaries identifying important findings, study identifiers, clinical phases, regulatory outcomes, and source references.

The synthesis agent then operates on these focused intermediate results rather than inheriting the complete exploratory execution history.

Option A organizes agents by document source, potentially fragmenting evidence concerning the same competitor program. Option B reduces context pressure without adequately separating responsibilities. Option D introduces numerous sequential handoffs and additional coordination overhead.

Databricks supports multi-agent orchestration and structured extraction, including citations and confidence information.

For scientific reliability, the summarized handoffs must retain precise source identifiers and retrievable evidence references so the synthesis agent can verify quotations before producing the final memo.

Official Databricks reference: Build a multi-agent system on Databricks Apps, Information Extraction, AI Search retrieval quality guide, and RAG on Databricks.

Question 6

Question Type: MultipleChoice

A context engineer is investigating a multi-agent pipeline on Databricks where a coordinating agent breaks down a document analysis task and dispatches subtasks to three specialized sub-agents. After deployment, the sub-agents produce outputs that are internally consistent but contradict one another and do not reflect earlier inferences made at the pipeline level. The sub-agents have access to the same retrieval index, the embedding model is verified as current, and individual sub-agent outputs are accurate when tested in isolation with full context supplied manually.

What is a reasonable root cause of the downstream failures?

Options:

A- The coordinating agent's temperature parameter is set too high, introducing non-determinism into the subtask decomposition and producing inconsistent dispatch instructions.
B- Each sub-agent is dispatched with only its individual task message and lacks access to the coordinator's earlier reasoning, tool outputs, and decisions.
C- The retrieval index used by the sub-agents is misconfigured, so they are querying a different corpus than the coordinating agent, leading to contradictory but locally consistent outputs.
D- The embedding model used for retrieval was not trained on the document domain, causing sub-agents to retrieve irrelevant chunks and produce contradictory summaries.

Answer:

B

Explanation:

The most probable cause is incomplete transfer of relevant contextual information from the coordinating agent to the specialized sub-agents.

In a multi-agent architecture, each agent typically operates within its own execution context. Information established by the coordinator does not automatically become available to downstream agents unless the application explicitly transmits it.

The scenario provides a decisive diagnostic observation: individual sub-agents produce accurate results when they are manually supplied with the complete context.

This indicates that the agents possess the necessary capabilities but are operating with incomplete information during normal orchestration.

Option B therefore identifies the central failure.

The coordinator should provide each sub-agent with a structured context package containing the assigned task, relevant prior conclusions, applicable constraints, selected evidence, and important tool results.

This ensures that specialized agents interpret their assignments consistently with the broader workflow.

Option A suggests model randomness, but the scenario does not identify unstable dispatch instructions. Options C and D conflict with the confirmed retrieval-index configuration and successful isolated tests.

Databricks supports orchestrated multi-agent systems in which specialized agents are invoked through a coordinating component.

MLflow Tracing can record the inputs and outputs of each agent invocation, allowing engineers to inspect precisely which context was transferred and identify missing information at agent boundaries.

Official Databricks reference: Build a multi-agent system on Databricks Apps, Core concepts for agent observability, and Tracing AG2.

Question 7

Question Type: MultipleChoice

A context engineer is developing a multi-agent research pipeline on Databricks. A coordinator agent divides an incoming scientific literature query into two parallel subtasks: a citation-retrieval agent and a methodology-analysis agent. Both agents independently receive only the original user query, produce their outputs, and return them to the coordinator, which then invokes a synthesis agent. The synthesis agent produces a final report that contradicts the citation-retrieval findings because it relies solely on the methodology-analysis output and has no visibility into retrieved citations.

Which two context propagation changes would prevent the conflicting outputs in the synthesis agent?

Choose two

Options:

A- Configure the coordinator agent to merge outputs from the citation-retrieval and methodology-

analysis agents into a single context payload before invoking the synthesis agent.
B- Replace the parallel dispatch pattern with a sequential chain so the methodology-analysis agent runs first, and the citation-retrieval agent runs afterward, without changing what context is forwarded to the synthesis agent.
C- Configure the coordinator to forward only a high-level methodology summary into the synthesis agent's context while omitting the underlying citation-retrieval data, to keep the context window small.
D- Broadcast every agent's full raw interaction history, including intermediate prompts and tool calls, directly to the synthesis agent, allowing the LLM to self-structure citation results from methodology analysis.
E- Update the synthesis agent's input schema to accept a structured context object containing both citation-retrieval results and methodology-analysis results as named fields.

Answer:

A, E

Explanation:

The failure results from incomplete context propagation between parallel sub-agents and the final synthesis stage.

The citation-retrieval agent and methodology-analysis agent produce complementary information. However, the synthesis agent receives only the methodology analysis, preventing it from evaluating whether the conclusions are supported by the retrieved literature.

Option A addresses this deficiency by requiring the coordinator to combine both upstream results before invoking synthesis.

The merged payload should include methodological findings, relevant citations, source identifiers, and any important discrepancies between evidence and interpretation.

Option E establishes a structured input contract containing separate fields for citation-retrieval results and methodology-analysis results.

This structure allows the synthesis agent to distinguish source evidence from analytical interpretations and verify that conclusions are supported by the appropriate references.

Databricks multi-agent orchestration supports specialized agents coordinated through a central orchestrator. The orchestrator is responsible for managing how information moves between specialized components.

Option B changes execution order without correcting the missing context. Option C deliberately removes essential citation evidence. Option D forwards excessive raw information, increasing token consumption and potentially obscuring relevant findings.

Combining merged context with explicit input schemas provides more reliable synthesis,

improves traceability, and makes context-propagation errors easier to diagnose through MLflow traces.

Official Databricks reference: Build a multi-agent system on Databricks Apps, RAG on Databricks, and Tracing overview.

To Get Premium Files for Context-Engineer-Associate Visit
https://www.p2pexams.com/products/context-engineer-associate

For More Free Questions Visit
https://www.p2pexams.com/databricks/pdf/context-engineer-associate

