Exam : **Certified Context Engineer Associate**

Title : Certified Context Engineer Associate Exam

Version : DEMO

1.A context engineer is debugging a Databricks customer support agent that handles complex billing escalations.
The system prompt instructs the agent to: (1) strictly apply the company's official refund policy; (2) treat entries from the "Billing-Policies" tool as authoritative; and (3) treat FAQ snippets and prior chat history as non-binding guidance only.
In one multi-step trace, the engineer observes the following:
Step 1: The agent calls the "Billing-Policies" tool and retrieves the official refund policy, which clearly states that chargebacks filed more than 90 days after purchase are ineligible for refunds.
Step 2: The agent calls a "Customer-FAQ-Search" tool that returns a community-authored FAQ thread. A top-voted but outdated and unofficial answer incorrectly claims that chargebacks are fully refundable regardless of when they are filed, and the agent summarizes this into the running conversation as an established fact.
Step 3: In a later turn, the customer describes a chargeback dispute from 18 months ago. The agent reasons from its own earlier summary, approves a full refund, and shows no reference to the official policy section that excludes cases older than 90 days, even though that policy text is still present in the context.
Step 4: In subsequent similar cases, the agent repeatedly approves refunds for old chargebacks, each time citing its prior summary as justification.
Which context failure mode describes what is going wrong in this trace?
A. Context distraction, because the context has grown so long with policy text, FAQs, and chat history that the model loses focus on the specific 90-day rule and drifts toward irrelevant retrieved content.
B. Context clash, because the agent is simultaneously presented with two contradictory authoritative sources and produces an inconsistent response that partially applies each policy.
C. Context confusion, because the agent misidentifies which retrieved policy section governs chargeback disputes, applying refund rules intended for a different transaction category to the current case.
D. Context poisoning, because a mistaken statement about refund eligibility enters the context as ground truth and is repeatedly reused in downstream steps, overriding the correct policy that is also present.
Answer: D
Explanation:
The agent has allowed an incorrect, non-authoritative statement to enter its working context as if it were established fact. That contaminated context is then reused in later reasoning and repeatedly drives the same wrong decision, even though the authoritative policy remains available. This persistence and propagation of bad context is characteristic of context poisoning.

2.A context engineer is building a Databricks agent to triage incoming customer support tickets for a B2B software platform. Each ticket averages 200 tokens.
For each ticket, the agent must:
(1) assign a severity rating of P0/P1/P2/P3 based on whether the issue is fully blocking, degrades core functionality, affects a non-critical feature, or is informational, a judgment that requires reading the ticket body rather than matching on keywords
(2) route to one of eight engineering teams based on the technical area implied by the ticket
(3) generate a brief, context-aware acknowledgment reply.
The pipeline must process up to 500 tickets per minute with an end-to-end latency budget of under 2

seconds per ticket. The token budget per invocation is 2,048 tokens, and the operations team has flagged that inference costs must remain minimal at this throughput.
Which reasoning mode should the context engineer configure for this agent?
A. Standard mode with extended thinking disabled, because skipping all reasoning overhead minimizes both per-invocation latency and inference cost at 500 tickets per minute.
B. Extended thinking with max_tokens capped at 1,024, because limiting the response length reduces cost while preserving full reasoning depth.
C. Extended thinking with a default thinking budget, because severity classification across four tiers and routing across eight teams benefits from deep chain-of-thought reasoning to maximize accuracy.
D. Reduced thinking, because the task requires modest structured reasoning across four severity tiers, a small thinking budget fits within the 2,048-token invocation, and latency and cost stay manageable at 500 tickets per minute.
Answer: D
Explanation:
The task needs some reasoning because severity and routing depend on understanding the ticket's meaning, not just keyword matching. Reduced thinking provides enough reasoning capacity for this moderate classification-and-routing workload while keeping token usage, latency, and inference cost low enough for the stated high-throughput requirement.

3.A context engineer is building a Databricks agent that orchestrates three tools: a freight rate calculator, a shipment status tracker, and a carrier booking submitter. The system prompt consumes 1,900 tokens of a 8,192-token context window, and the engineer reserves 5,100 tokens for the user turn and response, leaving 1,192 tokens for few-shot examples. Each example costs approximately 595 tokens, so the budget allows exactly two examples. The engineer has five candidates: Example 1 shows a freight rate query where the tool returns a list of rate options and the agent outputs that list verbatim; Example 2 shows a freight rate query where the tool returns an empty list and the agent outputs the same empty list with no additional handling; Example 3 shows a shipment status lookup where the tool returns a structured JSON object and the agent extracts and summarizes two fields; Example 4 shows a carrier booking submission where the tool returns a confirmation ID and the agent outputs a single acknowledgment sentence; and Example 5 shows a carrier booking submission where the tool returns a conflict code and the agent outputs a structured explanation paired with a recommended next action.
Which two examples should the engineer include?
A. Example 3 and 4, because they cover two distinct tools with structurally different outputs, demonstrate a data-retrieval pattern and a write-confirmation pattern, and avoid redundancy with the freight rate examples in the candidate set.
B. Example 1 and 4, because they cover two tools, demonstrate a list-retrieval output and a write-confirmation output, and represent the two most common happy-path response types the agent is expected to handle.
C. Example 3 and 5, because Example 3 is the only candidate demonstrating the shipment status tool path with JSON extraction, and Example 5 is the only candidate showing a conflict-code response paired with a multi-part agent output.
D. Example 2 and 3, because Example 2 covers an empty-list edge case on the freight rate tool and Example 3 introduces a JSON extraction pattern, together spanning two tools and two structurally different agent output behaviors.

Answer: C
Explanation:
With room for only two examples, the strongest choices are the ones that teach the most distinctive and reusable behaviors. One demonstrates structured field extraction and summarization from a tool response, while the other demonstrates non-happy-path handling with a structured explanation and recommended recovery action. Together they provide broader behavioral coverage than simple verbatim or acknowledgment examples.

4.A context engineer is building a Databricks agent for a retail inventory management system. The system prompt and tool descriptions consume 1,800 tokens of a 4,096-token context window, and the engineer reserves 1,000 tokens for the user turn and model response. The remaining tokens are used for few-shot examples, each costing approximately 250 tokens, so the engineer includes exactly five examples.
The agent uses three tools: inventory_lookup, reorder_check, and supplier_lead_time.
Six candidate examples are available: Example 1: A warehouse manager asks whether a depleted SKU needs reordering; the agent calls inventory_lookup, confirms depletion, calls reorder_check, and returns a binary reorder decision.Example 2: A procurement officer asks when a low-stock item can be replenished; the agent calls reorder_check, calls supplier_lead_time, and returns a replenishment recommendation.Example 3: A store associate asks for the current stock count of a single product; the agent calls inventory_lookup and returns a scalar quantity.Example 4: A store associate asks for the current stock count of a single product; the agent calls inventory_lookup and returns a structured status object containing the quantity and a restocking flag.Example 5: A store associate asks whether a forecasted shortfall requires immediate action; the agent calls inventory_lookup, compares against minimums, calls reorder_check, and returns a binary reorder decision.Example 6: A logistics coordinator asks how long a specific supplier takes to fulfill an order; the agent calls supplier_lead_time and returns a scalar lead-time value.
Which example should the engineer exclude to preserve the token budget while maximizing behavioral coverage?
A. Example 1, because the same tools are exercised across Examples 2 and 5, making its specific decision path redundant.
B. Example 6, because supplier_lead_time is already invoked in Example 2, and a standalone lead-time lookup does not add new tools to the demonstrated call space.
C. Example 3, because another example in the set covers the same user role and tool path, but the output of the other example is richer.
D. Example 5, because its forecasting logic duplicates the threshold-check pattern already established in Example 1, and its user role (store associate) appears elsewhere in the set.
Answer: C
Explanation:
The best example to exclude is the one that duplicates an existing tool path and user intent while demonstrating a less informative output pattern. The richer stock-lookup example already covers the same core interaction and additionally teaches the agent how to return a structured status object, so retaining it provides greater behavioral coverage within the fixed example budget.

5.A context engineer is debugging a Databricks agent that staffs an internal IT help desk. The agent is

already connected to a retrieval pipeline over a live, continuously updated knowledge base, and relevant KB passages are retrieved and placed in its context on every request. Its system prompt has three parts: a 400-token persona preamble, a 600-token list of every supported software title, and a 200-token closing instruction block – none of which says how to weigh retrieved passages against the model's own training knowledge.

Observed behavior: the agent identifies the user's intent correctly, and the correct KB passages are present in context, but for version-specific questions, it answers from its training knowledge and recommends releases that the knowledge base marks as deprecated. The team will accept only a change that does not rewrite the full prompt and does not add or modify retrieval infrastructure.

Which system-prompt revision corrects the version-accuracy behavior?

A. Expand the persona preamble to state the model's training-data cutoff date, so users understand version details may be out of date.

B. Add a sentence to the closing instruction block telling the agent that, when retrieved passages and its own training knowledge conflict, the retrieved passages take priority.

C. Move the 600-token software-title list to the very end of the prompt, after the closing block, so the supported-software context is the most recent thing the model reads before answering.

D. Add a directive to the closing block requiring the agent to present every version number in a consistent format, such as semantic versioning (MAJOR.MINOR.PATCH).

Answer: B

Explanation:

The retrieval pipeline is already supplying the correct, current evidence, so the remaining issue is instruction priority. Adding an explicit rule that retrieved knowledge-base content overrides conflicting model training knowledge tells the agent which source is authoritative for version-specific answers without changing the retrieval system or rewriting the full prompt.