Databricks-Generative-AI-Engineer-Associate Practice Questions

Shared by Reeves on 18-08-2026

For More Free Questions and Preparation Resources

Check the Links on Last Page

Question 1

Question Type: MultipleChoice

A Generative Al Engineer wants their (inetuned LLMs in their prod Databncks workspace available for testing in their dev workspace as well. All of their workspaces are Unity Catalog enabled and they are currently logging their models into the Model Registry in MLflow.

What is the most cost-effective and secure option for the Generative Al Engineer to accomplish their gAi?

Options:

A- Use an external model registry which can be accessed from all workspaces
B- Setup a script to export the model from prod and import it to dev.
C- Setup a duplicate training pipeline in dev, so that an identical model is available in dev.
D- Use MLflow to log the model directly into Unity Catalog, and enable READ access in the dev workspace to the model.

Answer:

D

Explanation:

The goal is to make fine-tuned LLMs from a production (prod) Databricks workspace available for testing in a development (dev) workspace, leveraging Unity Catalog and MLflow, while ensuring cost-effectiveness and security. Let's analyze the options.

Option A: Use an external model registry which can be accessed from all workspaces

An external registry adds cost (e.g., hosting fees) and complexity (e.g., integration, security configurations) outside Databricks' native ecosystem, reducing security compared to Unity Catalog's governance.

Databricks Reference: 'Unity Catalog provides a centralized, secure model registry within Databricks' ('Unity Catalog Documentation,' 2023).

Option B: Setup a script to export the model from prod and import it to dev

Export/import scripts require manual effort, storage for model artifacts, and repeated execution, increasing operational cost and risk (e.g., version mismatches, unsecured transfers). It's less efficient than a native solution.

Databricks Reference: Manual processes are discouraged when Unity Catalog offers built-in sharing: 'Avoid redundant workflows with Unity Catalog's cross-workspace access' ('MLflow with Unity Catalog').

Option C: Setup a duplicate training pipeline in dev, so that an identical model is available in dev

Duplicating the training pipeline doubles compute and storage costs, as it retrains the model from scratch. It's neither cost-effective nor necessary when the prod model can be reused securely.

Databricks Reference: 'Re-running training is resource-intensive; leverage existing models where possible' ('Generative AI Engineer Guide').

Option D: Use MLflow to log the model directly into Unity Catalog, and enable READ access in the dev workspace to the model

Unity Catalog, integrated with MLflow, allows models logged in prod to be centrally managed and accessed across workspaces with fine-grained permissions (e.g., READ for dev). This is cost-effective (no extra infrastructure or retraining) and secure (governed by Databricks' access controls).

Databricks Reference: 'Log models to Unity Catalog via MLflow, then grant access to other workspaces securely' ('MLflow Model Registry with Unity Catalog,' 2023).

Conclusion: Option D leverages Databricks' native tools (MLflow and Unity Catalog) for a seamless, cost-effective, and secure solution, avoiding external systems, manual scripts, or redundant training.

Question 2

Question Type: MultipleChoice

An AI developer team wants to fine-tune an open-weight model to have exceptional performance on a code generation use case. They are trying to choose the best model to start with. They want to minimize model hosting costs and are using Hugging Face model cards and spaces to explore models. Which TWO model attributes and metrics should the team focus on to make their selection?

Options:

A- Big Code Models Leaderboard
B- Number of model parameters
C- MTEB Leaderboard
D- Chatbot Arena Leaderboard
E- Number of model downloads last month

Answer:

A, B

Explanation:

To optimize for code generation performance and hosting costs, a Generative AI engineer must look at specific metrics.

Big Code Models Leaderboard (A): This is the industry-standard benchmark for code-specific LLMs (like StarCoder or CodeLlama). It measures performance on tasks like HumanEval and MBPP, providing a direct indicator of how well the model handles programming logic.

Number of model parameters (B): This is the primary driver of hosting costs. Larger models (e.g., 70B) require more GPU memory (VRAM) and more expensive compute instances (like A100s/H100s) than smaller models (e.g., 7B or 13B). To minimize costs, the team should look for the smallest model that achieves a high score on the Big Code Leaderboard.

Note: MTEB (C) is for embeddings, and Chatbot Arena (D) is for general-purpose chat, neither of which is the primary metric for specialized code generation fine-tuning.

Question 3

Question Type: MultipleChoice

A Generative AI Engineer is experimenting with using parameters to configure an agent in Mosaic Agent Framework. However, they are struggling to get the agent to respond with relevant information with this configuration:

config = {"prompt_template": "You are a trivia bot. Generate a question based on the user's input: {user_input}", "input_vars": ["user_input"], "parameters": {"temperature": 0.01, "max_tokens": 500}}

Which error is causing the problem?

Options:

A- The prompt does not parse the user's input vars
B- The prompt does not set the retriever schema
C- The prompt does not list available agents for the LLM to call
D- The prompt is not wrapped in ChatModel

Answer:

A

Explanation:

In the Mosaic AI Agent Framework and underlying LangChain-based configurations, the 'input_vars' or 'input_variables' must be correctly mapped and referenced within the template. If the configuration dictionary identifies user_input as the variable but the logic executing the chain does not correctly 'inject' the runtime value into the {user_input} placeholder, the LLM will receive a literal string (or an empty value) rather than the user's actual question. This results in the model failing to provide relevant information because it essentially doesn't know what the user asked. Engineering standards require ensuring that the key used in the input_vars list matches the key in the JSON payload sent to the model serving endpoint. If there is a mismatch or a failure to parse, the prompt remains static, leading to generic or irrelevant responses.

Question 4

Question Type: MultipleChoice

A Generative Al Engineer is tasked with improving the RAG quality by addressing its inflammatory outputs.

Which action would be most effective in mitigating the problem of offensive text outputs?

Options:

A- Increase the frequency of upstream data updates
B- Inform the user of the expected RAG behavior
C- Restrict access to the data sources to a limited number of users
D- Curate upstream data properly that includes manual review before it is fed into the RAG system

Answer:

D

Explanation:

Addressing offensive or inflammatory outputs in a Retrieval-Augmented Generation (RAG) system is critical for improving user experience and ensuring ethical AI deployment. Here's why D is the most effective approach:

Manual data curation: The root cause of offensive outputs often comes from the underlying data used to train the model or populate the retrieval system. By manually curating the upstream

data and conducting thorough reviews before the data is fed into the RAG system, the engineer can filter out harmful, offensive, or inappropriate content.

Improving data quality: Curating data ensures the system retrieves and generates responses from a high-quality, well-vetted dataset. This directly impacts the relevance and appropriateness of the outputs from the RAG system, preventing inflammatory content from being included in responses.

Effectiveness: This strategy directly tackles the problem at its source (the data) rather than just mitigating the consequences (such as informing users or restricting access). It ensures that the system consistently provides non-offensive, relevant information.

Other options, such as increasing the frequency of data updates or informing users about behavior expectations, may not directly mitigate the generation of inflammatory outputs.

Question 5

Question Type: MultipleChoice

A Generative AI Engineer is deploying a customer-facing, fine-tuned LLM on their public website. Given the large investment the company put into fine-tuning this model, and the proprietary nature of the tuning data, they are concerned about model inversion attacks. Which of the following Databricks AI Security Framework (DASF) risk mitigation strategies are most relevant to this use case?

Options:

A- Implement AI guardrails to allow users to configure and enforce compliance
B- Leverage Databricks access control lists (ACLs) to configure permissions for accessing models
C- Use secure model features with Databricks Feature Store
D- Apply attribute-based access controls (ABAC) to limit unauthorized access

Answer:

A

Explanation:

Model inversion attacks occur when an attacker uses the model's outputs to reconstruct the sensitive training data used during the fine-tuning process. To mitigate this in a public-facing application, implementing AI Guardrails is the most relevant strategy. Guardrails act as a programmable 'filter' between the LLM and the end-user. They can be configured to detect if a

model's response contains patterns that look like proprietary training data or PII (Personally Identifiable Information). While ACLs (B) and ABAC (D) protect the model's infrastructure (who can invoke the API), they do not inspect the content of the output, which is where the inversion attack actually manifests. Databricks provides integrated guardrail capabilities (via Mosaic AI Gateway) specifically to enforce compliance and prevent the leakage of sensitive internal knowledge that may have been baked into the model weights during fine-tuning.

Question 6

Question Type: MultipleChoice

A Generative Al Engineer is developing a RAG application and would like to experiment with different embedding models to improve the application performance.

Which strategy for picking an embedding model should they Select?

Options:

A- Pick an embedding model trained on related domain knowledge
B- Pick the most recent and most performant open LLM released at the time
C- pick the embedding model ranked highest on the Massive Text Embedding Benchmark (MTEB) leaderboard hosted by HuggingFace
D- Pick an embedding model with multilingual support to support potential multilingual user questions

Answer:

A

Explanation:

The task involves improving a Retrieval-Augmented Generation (RAG) application's performance by experimenting with embedding models. The choice of embedding model impacts retrieval accuracy, which is critical for RAG systems. Let's evaluate the options based on Databricks Generative AI Engineer best practices.

Option A: Pick an embedding model trained on related domain knowledge

Embedding models trained on domain-specific data (e.g., industry-specific corpora) produce vectors that better capture the semantics of the application's context, improving retrieval relevance. For RAG, this is a key strategy to enhance performance.

Databricks Reference: 'For optimal retrieval in RAG systems, select embedding models aligned

with the domain of your data' ('Building LLM Applications with Databricks,' 2023).

Option B: Pick the most recent and most performant open LLM released at the time

LLMs are not embedding models; they generate text, not embeddings for retrieval. While recent LLMs may be performant for generation, this doesn't address the embedding step in RAG. This option misunderstands the component being selected.

Databricks Reference: Embedding models and LLMs are distinct in RAG workflows: 'Embedding models convert text to vectors, while LLMs generate responses' ('Generative AI Cookbook').

Option C: Pick the embedding model ranked highest on the Massive Text Embedding Benchmark (MTEB) leaderboard hosted by HuggingFace

The MTEB leaderboard ranks models across general tasks, but high overall performance doesn't guarantee suitability for a specific domain. A top-ranked model might excel in generic contexts but underperform on the engineer's unique data.

Databricks Reference: General performance is less critical than domain fit: 'Benchmark rankings provide a starting point, but domain-specific evaluation is recommended' ('Databricks Generative AI Engineer Guide').

Option D: Pick an embedding model with multilingual support to support potential multilingual user questions

Multilingual support is useful only if the application explicitly requires it. Without evidence of multilingual needs, this adds complexity without guaranteed performance gains for the current use case.

Databricks Reference: 'Choose features like multilingual support based on application requirements' ('Building LLM-Powered Applications').

Conclusion: Option A is the best strategy because it prioritizes domain relevance, directly improving retrieval accuracy in a RAG system---aligning with Databricks' emphasis on tailoring models to specific use cases.

Question 7

Question Type: MultipleChoice

A Generative AI Engineer is tasked with deploying an application that takes advantage of a custom MLflow Pyfunc model to return some interim results.

How should they configure the endpoint to pass the secrets and credentials?

Options:

A- Use spark.conf.set ()
B- Pass variables using the Databricks Feature Store API
C- Add credentials using environment variables
D- Pass the secrets in plain text

Answer:

C

Explanation:

Context: Deploying an application that uses an MLflow Pyfunc model involves managing sensitive information such as secrets and credentials securely.

Explanation of Options:

Option A: Use spark.conf.set(): While this method can pass configurations within Spark jobs, using it for secrets is not recommended because it may expose them in logs or Spark UI.

Option B: Pass variables using the Databricks Feature Store API: The Feature Store API is designed for managing features for machine learning, not for handling secrets or credentials.

Option C: Add credentials using environment variables: This is a common practice for managing credentials in a secure manner, as environment variables can be accessed securely by applications without exposing them in the codebase.

Option D: Pass the secrets in plain text: This is highly insecure and not recommended, as it exposes sensitive information directly in the code.

Therefore, Option C is the best method for securely passing secrets and credentials to an application, protecting them from exposure.

Question 8

Question Type: MultipleChoice

A Generative Al Engineer is helping a cinema extend its website's chat bot to be able to respond to questions about specific showtimes for movies currently playing at their local theater. They already have the location of the user provided by location services to their agent, and a Delta table which is continually updated with the latest showtime information by location. They want to implement this new capability In their RAG application.

Which option will do this with the least effort and in the most performant way?

Options:

A- Create a Feature Serving Endpoint from a FeatureSpec that references an online store synced from the Delta table. Query the Feature Serving Endpoint as part of the agent logic / tool implementation.
B- Query the Delta table directly via a SQL query constructed from the user's input using a text-to-SQL LLM in the agent logic / tool
C- implementation. Write the Delta table contents to a text column.then embed those texts using an embedding model and store these in the vector index Look
up the information based on the embedding as part of the agent logic / tool implementation.
D- Set up a task in Databricks Workflows to write the information in the Delta table periodically to an external database such as MySQL and query the information from there as part of the agent logic / tool implementation.

Answer:

A

Explanation:

The task is to extend a cinema chatbot to provide movie showtime information using a RAG application, leveraging user location and a continuously updated Delta table, with minimal effort and high performance. Let's evaluate the options.

Option A: Create a Feature Serving Endpoint from a FeatureSpec that references an online store synced from the Delta table. Query the Feature Serving Endpoint as part of the agent logic / tool implementation

Databricks Feature Serving provides low-latency access to real-time data from Delta tables via an online store. Syncing the Delta table to a Feature Serving Endpoint allows the chatbot to query showtimes efficiently, integrating seamlessly into the RAG agent's tool logic. This leverages Databricks' native infrastructure, minimizing effort and ensuring performance.

Databricks Reference: 'Feature Serving Endpoints provide real-time access to Delta table data with low latency, ideal for production systems' ('Databricks Feature Engineering Guide,' 2023).

Option B: Query the Delta table directly via a SQL query constructed from the user's input using a text-to-SQL LLM in the agent logic / tool

Using a text-to-SQL LLM to generate queries adds complexity (e.g., ensuring accurate SQL generation) and latency (LLM inference + SQL execution). While feasible, it's less performant and requires more effort than a pre-built serving solution.

Databricks Reference: 'Direct SQL queries are flexible but may introduce overhead in real-time applications' ('Building LLM Applications with Databricks').

Option C: Write the Delta table contents to a text column, then embed those texts using an embedding model and store these in the vector index. Look up the information based on the embedding as part of the agent logic / tool implementation

Converting structured Delta table data (e.g., showtimes) into text, embedding it, and using vector search is inefficient for structured lookups. It's effort-intensive (preprocessing, embedding) and less precise than direct queries, undermining performance.

Databricks Reference: 'Vector search excels for unstructured data, not structured tabular lookups' ('Databricks Vector Search Documentation').

Option D: Set up a task in Databricks Workflows to write the information in the Delta table periodically to an external database such as MySQL and query the information from there as part of the agent logic / tool implementation

Exporting to an external database (e.g., MySQL) adds setup effort (workflow, external DB management) and latency (periodic updates vs. real-time). It's less performant and more complex than using Databricks' native tools.

Databricks Reference: 'Avoid external systems when Delta tables provide real-time data natively' ('Databricks Workflows Guide').

Conclusion: Option A minimizes effort by using Databricks Feature Serving for real-time, low-latency access to the Delta table, ensuring high performance in a production-ready RAG chatbot.

Question 9

Question Type: MultipleChoice

A team uses Mosaic AI Vector Search to retrieve documents for their Retrieval-Augmented Generation (RAG) pipeline. The search query returns five relevant documents, and the first three are added to the prompt as context. Performance evaluation with Agent Evaluation shows that some lower-ranked retrieved documents have higher context relevancy scores than higher-ranked documents. Which option should the team consider to optimize this workflow?

Options:

A- Use a reranker to order the documents based on the relevance scores.
B- Modify the prompt to instruct the LLM to order the documents based on the relevance scores.
C- Use a different embedding model for computing document embeddings.
D- Increase the number of documents added to the prompt to improve context relevance.

Answer:

A

Explanation:

The scenario describes a common 'retrieval gap' where the initial bi-encoder (embedding model) used for vector search identifies relevant documents but does not rank them perfectly. This happens because embedding models represent entire documents as a single vector, which can lose nuance. The standard engineering solution is to implement a Reranker (Cross-Encoder). Unlike embedding models, a reranker processes the query and a candidate document simultaneously, allowing it to capture deep semantic interactions between the two. In a Mosaic AI workflow, after the vector search retrieves the top k documents, the reranker evaluates those specific k documents to produce a more accurate relevance score. This ensures that the most contextually relevant documents are placed at the top of the list (and thus the top of the LLM prompt), which is crucial because LLMs are sensitive to document order and often prioritize information found at the beginning of the context.

P2P exams