Databricks.Databricks-Generative-AI-Engineer-Associate.v2026-08-18.q40

Exam Code:	Databricks-Generative-AI-Engineer-Associate
Exam Name:	Databricks Certified Generative AI Engineer Associate
Certification Provider:	Databricks
Free Question Number:	40
Version:	v2026-08-18
# of views:	142
# of Questions views:	400
https://www.exam-tests.com/Databricks-Generative-AI-Engineer-Associate-exam/Databricks.Databricks-Generative-AI-Engineer-Associate.v2026-08-18.q40.html	

NEW QUESTION: 1
A Generative AI Engineer is testing a simple prompt template in LangChain using the code below, but is getting an error:
Python
from langchain.chains import LLMChain
from langchain_community.llms import OpenAI
from langchain_core.prompts import PromptTemplate
prompt_template = "Tell me a {adjective} joke"
prompt = PromptTemplate(input_variables=["adjective"], template=prompt_template)
# ... (Error-prone section)
Assuming the API key was properly defined, what change does the Generative AI Engineer need to make to fix their chain?
A. (Incorrect structure)
B. (Incorrect structure)
C. prompt_template = "Tell me a {adjective} joke"
prompt = PromptTemplate(input_variables=["adjective"], template=prompt_template) llm = OpenAI()
llm_chain = LLMChain(prompt=prompt, llm=llm) llm_chain.generate([{"adjective": "funny"}])
D. (Incorrect structure)
Answer: C (LEAVE A REPLY)
The error in the original snippet usually stems from the improper instantiation of the LLMChain or the incorrect call to the .generate() method. In LangChain, an LLMChain requires two primary components: an LLM (the engine) and a Prompt (the template). Option C provides the correct syntax: first, the PromptTemplate is defined with the correct input_variables. Second, the OpenAI model is instantiated. Third, the LLMChain binds the model and the prompt together. Finally, the .generate() method expects a list of dictionaries, where each dictionary represents a set of inputs for the prompt variables. Options A,

B, and D in the original image contain syntax errors such as passing the variable directly into the chain initialization or missing the dictionary list format required by the standard LangChain API for batch-like generation.

NEW QUESTION: 2
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?
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: (SHOW ANSWER)
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.

NEW QUESTION: 3
A Generative AI Engineer is ready to deploy an LLM application written using Foundation Model APIs. They want to follow security best practices for production scenarios Which authentication method should they choose?
A. Use an access token belonging to service principals
B. Use a frequently rotated access token belonging to either a workspace user or a service principal
C. Use OAuth machine-to-machine authentication
D. Use an access token belonging to any workspace user
Answer: (SHOW ANSWER)
The task is to deploy an LLM application using Foundation Model APIs in a production environment while adhering to security best practices. Authentication is critical for securing access to Databricks resources, such as the Foundation Model API. Let's evaluate the options based on Databricks' security guidelines for production scenarios.
* Option A: Use an access token belonging to service principals

* Service principals are non-human identities designed for automated workflows and applications in Databricks. Using an access token tied to a service principal ensures that the authentication is scoped to the application, follows least-privilege principles (via role-based access control), and avoids reliance on individual user credentials. This is a security best practice for production deployments.
* Databricks Reference:"For production applications, use service principals with access tokens to authenticate securely, avoiding user-specific credentials"("Databricks Security Best Practices," 2023). Additionally, the "Foundation Model API Documentation" states:"Service principal tokens are recommended for programmatic access to Foundation Model APIs."
* Option B: Use a frequently rotated access token belonging to either a workspace user or a service principal
* Frequent rotation enhances security by limiting token exposure, but tying the token to a workspace user introduces risks (e.g., user account changes, broader permissions). Including both user and service principal options dilutes the focus on application-specific security, making this less ideal than a service-principal-only approach. It also adds operational overhead without clear benefits over Option A.
* Databricks Reference:"While token rotation is a good practice, service principals are preferred over user accounts for application authentication"("Managing Tokens in Databricks," 2023).
* Option C: Use OAuth machine-to-machine authentication
* OAuth M2M (e.g., client credentials flow) is a secure method for application-to-service communication, often using service principals under the hood. However, Databricks' Foundation Model API primarily supports personal access tokens (PATs) or service principal tokens over full OAuth flows for simplicity in production setups. OAuth M2M adds complexity (e.g., managing refresh tokens) without a clear advantage in this context.
* Databricks Reference:"OAuth is supported in Databricks, but service principal tokens are simpler and sufficient for most API-based workloads"("Databricks Authentication Guide," 2023).
* Option D: Use an access token belonging to any workspace user
* Using a user's access token ties the application to an individual's identity, violating security best practices. It risks exposure if the user leaves, changes roles, or has overly broad permissions, and it's not scalable or auditable for production.
* Databricks Reference:"Avoid using personal user tokens for production applications due to security and governance concerns"("Databricks Security Best Practices," 2023).
Conclusion: Option A is the best choice, as it uses a service principal's access token, aligning with Databricks' security best practices for production LLM applications. It ensures secure, application-specific authentication with minimal complexity, as explicitly recommended for Foundation Model API deployments.

NEW QUESTION: 4
A Generative AI Engineer has created a RAG application to look up answers to questions about a series of fantasy novels that are being asked on the author's web forum. The fantasy novel texts are chunked and embedded into a vector store with metadata (page number, chapter number, book title), retrieved with the user's query, and provided to an LLM for response generation. The Generative AI Engineer used

their intuition to pick the chunking strategy and associated configurations but now wants to more methodically choose the best values.
Which TWO strategies should the Generative AI Engineer take to optimize their chunking strategy and parameters? (Choose two.)
A. Change embedding models and compare performance.
B. Add a classifier for user queries that predicts which book will best contain the answer. Use this to filter retrieval.
C. Choose an appropriate evaluation metric (such as recall or NDCG) and experiment with changes in the chunking strategy, such as splitting chunks by paragraphs or chapters.
Choose the strategy that gives the best performance metric.
D. Pass known questions and best answers to an LLM and instruct the LLM to provide the best token count. Use a summary statistic (mean, median, etc.) of the best token counts to choose chunk size.
E. Create an LLM-as-a-judge metric to evaluate how well previous questions are answered by the most appropriate chunk. Optimize the chunking parameters based upon the values of the metric.
Answer: C,E (LEAVE A REPLY)
To optimize a chunking strategy for a Retrieval-Augmented Generation (RAG) application, the Generative AI Engineer needs a structured approach to evaluating the chunking strategy, ensuring that the chosen configuration retrieves the most relevant information and leads to accurate and coherent LLM responses. Here's why C and E are the correct strategies:
Strategy C: Evaluation Metrics (Recall, NDCG)
Define an evaluation metric: Common evaluation metrics such as recall, precision, or NDCG (Normalized Discounted Cumulative Gain) measure how well the retrieved chunks match the user's query and the expected response.
Recall measures the proportion of relevant information retrieved.
NDCG is often used when you want to account for both the relevance of retrieved chunks and the ranking or order in which they are retrieved.
Experiment with chunking strategies: Adjusting chunking strategies based on text structure (e.g., splitting by paragraph, chapter, or a fixed number of tokens) allows the engineer to experiment with various ways of slicing the text. Some chunks may better align with the user's query than others.
Evaluate performance: By using recall or NDCG, the engineer can methodically test various chunking strategies to identify which one yields the highest performance. This ensures that the chunking method provides the most relevant information when embedding and retrieving data from the vector store.
Strategy E: LLM-as-a-Judge Metric
Use the LLM as an evaluator: After retrieving chunks, the LLM can be used to evaluate the quality of answers based on the chunks provided. This could be framed as a "judge" function, where the LLM compares how well a given chunk answers previous user queries.
Optimize based on the LLM's judgment: By having the LLM assess previous answers and rate their relevance and accuracy, the engineer can collect feedback on how well different chunking configurations perform in real-world scenarios.
This metric could be a qualitative judgment on how closely the retrieved information matches the user's intent.

Tune chunking parameters: Based on the LLM's judgment, the engineer can adjust the chunk size or structure to better align with the LLM's responses, optimizing retrieval for future queries.
By combining these two approaches, the engineer ensures that the chunking strategy is systematically evaluated using both quantitative (recall/NDCG) and qualitative (LLM judgment) methods. This balanced optimization process results in improved retrieval relevance and, consequently, better response generation by the LLM.

NEW QUESTION: 5
A Generative AI Engineer is designing an LLM-powered live sports commentary platform. The platform provides real-time updates and LLM-generated analyses for any users who would like to have live summaries, rather than reading a series of potentially outdated news articles.
Which tool below will give the platform access to real-time data for generating game analyses based on the latest game scores?
A. DatabricksIQ
B. Foundation Model APIs
C. Feature Serving
D. AutoML
Answer: (SHOW ANSWER)
* Problem Context: The engineer is developing an LLM-powered live sports commentary platform that needs to provide real-time updates and analyses based on the latest game scores. The critical requirement here is the capability to access and integrate real-time data efficiently with the platform for immediate analysis and reporting.
* Explanation of Options:
* Option A: DatabricksIQ: While DatabricksIQ offers integration and data processing capabilities, it is more aligned with data analytics rather than real-time feature serving, which is crucial for immediate updates necessary in a live sports commentary context.
* Option B: Foundation Model APIs: These APIs facilitate interactions with pre-trained models and could be part of the solution, but on their own, they do not provide mechanisms to access real- time game scores.
* Option C: Feature Serving: This is the correct answer as feature serving specifically refers to the real-time provision of data (features) to models for prediction. This would be essential for an LLM that generates analyses based on live game data, ensuring that the commentary is current and based on the latest events in the sport.
* Option D: AutoML: This tool automates the process of applying machine learning models to real-world problems, but it does not directly provide real-time data access, which is a critical requirement for the platform.
Thus,Option C(Feature Serving) is the most suitable tool for the platform as it directly supports the real-time data needs of an LLM-powered sports commentary system, ensuring that the analyses and updates are based on the latest available information.

NEW QUESTION: 6

A Generative AI Engineer is setting up a Databricks Vector Search that will lookup news articles by topic within 10 days of the date specified An example query might be "Tell me about monster truck news around January 5th 1992". They want to do this with the least amount of effort.
How can they set up their Vector Search index to support this use case?
A. Split articles by 10 day blocks and return the block closest to the query.
B. Include metadata columns for article date and topic to support metadata filtering.
C. pass the query directly to the vector search index and return the best articles.
D. Create separate indexes by topic and add a classifier model to appropriately pick the best index.
Answer: B (LEAVE A REPLY)
The task is to set up a Databricks Vector Search index for news articles, supporting queries like "monster truck news around January 5th, 1992," with minimal effort. The index must filter by topic and a 10-day date range. Let's evaluate the options.
* Option A: Split articles by 10-day blocks and return the block closest to the query
* Pre-splitting articles into 10-day blocks requires significant preprocessing and index management (e.g., one index per block). It's effort-intensive and inflexible for dynamic date ranges.
* Databricks Reference:"Static partitioning increases setup complexity; metadata filtering is preferred"("Databricks Vector Search Documentation").
* Option B: Include metadata columns for article date and topic to support metadata filtering
* Adding date and topic as metadata in the Vector Search index allows dynamic filtering (e.g., date ± 5 days, topic = "monster truck") at query time. This leverages Databricks' built-in metadata filtering, minimizing setup effort.
* Databricks Reference:"Vector Search supports metadata filtering on columns like date or category for precise retrieval with minimal preprocessing"("Vector Search Guide," 2023).
* Option C: Pass the query directly to the vector search index and return the best articles
* Passing the full query (e.g., "Tell me about monster truck news around January 5th, 1992") to Vector Search relies solely on embeddings, ignoring structured filtering for date and topic. This risks inaccurate results without explicit range logic.
* Databricks Reference:"Pure vector similarity may not handle temporal or categorical constraints effectively"("Building LLM Applications with Databricks").
* Option D: Create separate indexes by topic and add a classifier model to appropriately pick the best index
* Separate indexes per topic plus a classifier model adds significant complexity (index creation, model training, maintenance), far exceeding "least effort." It's overkill for this use case.
* Databricks Reference:"Multiple indexes increase overhead; single-index with metadata is simpler"("Databricks Vector Search Documentation").
Conclusion: Option B is the simplest and most effective solution, using metadata filtering in a single Vector Search index to handle date ranges and topics, aligning with Databricks' emphasis on efficient, low-effort setups.

NEW QUESTION: 7

A Generative AI Engineer is using an LLM to classify species of edible mushrooms based on text descriptions of certain features. The model is returning accurate responses in testing and the Generative AI Engineer is confident they have the correct list of possible labels, but the output frequently contains additional reasoning in the answer when the Generative AI Engineer only wants to return the label with no additional text.
Which action should they take to elicit the desired behavior from this LLM?
A. Use few snot prompting to instruct the model on expected output format
B. Use zero shot prompting to instruct the model on expected output format
C. Use zero shot chain-of-thought prompting to prevent a verbose output format
D. Use a system prompt to instruct the model to be succinct in its answer
Answer: D (LEAVE A REPLY)
The LLM classifies mushroom species accurately but includes unwanted reasoning text, and the engineer wants only the label. Let's assess how to control output format effectively.
* Option A: Use few shot prompting to instruct the model on expected output format
* Few-shot prompting provides examples (e.g., input: description, output: label). It can work but requires crafting multiple examples, which is effort-intensive and less direct than a clear instruction.
* Databricks Reference:"Few-shot prompting guides LLMs via examples, effective for format control but requires careful design"("Generative AI Cookbook").
* Option B: Use zero shot prompting to instruct the model on expected output format
* Zero-shot prompting relies on a single instruction (e.g., "Return only the label") without examples. It's simpler than few-shot but may not consistently enforce succinctness if the LLM's default behavior is verbose.
* Databricks Reference:"Zero-shot prompting can specify output but may lack precision without examples"("Building LLM Applications with Databricks").
* Option C: Use zero shot chain-of-thought prompting to prevent a verbose output format
* Chain-of-Thought (CoT) encourages step-by-step reasoning, which increases verbosity-opposite to the desired outcome. This contradicts the goal of label-only output.
* Databricks Reference:"CoT prompting enhances reasoning but often results in detailed responses"("Databricks Generative AI Engineer Guide").
* Option D: Use a system prompt to instruct the model to be succinct in its answer
* A system prompt (e.g., "Respond with only the species label, no additional text") sets a global instruction for the LLM's behavior. It's direct, reusable, and effective for controlling output style across queries.
* Databricks Reference:"System prompts define LLM behavior consistently, ideal for enforcing concise outputs"("Generative AI Cookbook," 2023).
Conclusion: Option D is the most effective and straightforward action, using a system prompt to enforce succinct, label-only responses, aligning with Databricks' best practices for output control.

NEW QUESTION: 8
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?
A. Use spark.conf.set ()
B. Pass variables using the Databricks Feature Store API
C. Pass the secrets in plain text
D. Add credentials using environment variables
Answer: (SHOW ANSWER)
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 Cis the best method for securely passing secrets and credentials to an application, protecting them from exposure.

NEW QUESTION: 9
A Generative AI Engineer is working with a retail company that wants to enhance its customer experience by automatically handling common customer inquiries. They are working on an LLM-powered AI solution that should improve response times while maintaining a personalized interaction. They want to define the appropriate input and LLM task to do this.
Which input/output pair will do this?
A. Input: Customer reviews; Output Group the reviews by users and aggregate per-user average rating, then respond
B. Input: Customer service chat logs; Output Group the chat logs by users, followed by summarizing each user's interactions, then respond
C. Input: Customer service chat logs; Output: Find the answers to similar questions and respond with a summary
D. Input: Customer reviews: Output Classify review sentiment
Answer: C (LEAVE A REPLY)
The task described in the question involves enhancing customer experience by automatically handling common customer inquiries using an LLM-powered AI solution. This requires the system to process input data (customer inquiries) and generate personalized, relevant responses efficiently. Let's evaluate the options step-by-step in the context of Databricks Generative AI Engineer principles, which emphasize leveraging LLMs for tasks like question answering, summarization, and retrieval-augmented generation (RAG).

* Option A: Input: Customer reviews; Output: Group the reviews by users and aggregate per-user average rating, then respond
* This option focuses on analyzing customer reviews to compute average ratings per user. While this might be useful for sentiment analysis or user profiling, it does not directly address the goal of handling common customer inquiries or improving response times for personalized interactions. Customer reviews are typically feedback data, not real-time inquiries requiring immediate responses.
* Databricks Reference: Databricks documentation on LLMs (e.g., "Building LLM Applications with Databricks") emphasizes that LLMs excel at tasks like question answering and conversational responses, not just aggregation or statistical analysis of reviews.
* Option B: Input: Customer service chat logs; Output: Group the chat logs by users, followed by summarizing each user's interactions, then respond
* This option uses chat logs as input, which aligns with customer service scenarios. However, the output-grouping by users and summarizing interactions-focuses on user-specific summaries rather than directly addressing inquiries. While summarization is an LLM capability, this approach lacks the specificity of finding answers to common questions, which is central to the problem.
* Databricks Reference: Per Databricks' "Generative AI Cookbook," LLMs can summarize text, but for customer service, the emphasis is on retrieval and response generation (e.g., RAG workflows) rather than user interaction summaries alone.
* Option C: Input: Customer service chat logs; Output: Find the answers to similar questions and respond with a summary
* This option uses chat logs (real customer inquiries) as input and tasks the LLM with identifying answers to similar questions, then providing a summarized response. This directly aligns with the goal of handling common inquiries efficiently while maintaining personalization (by referencing past interactions or similar cases). It leverages LLM capabilities like semantic search, retrieval, and response generation, which are core to Databricks' LLM workflows.
* Databricks Reference: From Databricks documentation ("Building LLM-Powered Applications," 2023), an exact extract states:"For customer support use cases, LLMs can be used to retrieve relevant answers from historical data like chat logs and generate concise, contextually appropriate responses."This matches Option C's approach of finding answers and summarizing them.
* Option D: Input: Customer reviews; Output: Classify review sentiment
* This option focuses on sentiment classification of reviews, which is a valid LLM task but unrelated to handling customer inquiries or improving response times in a conversational context.
It's more suited for feedback analysis than real-time customer service.
* Databricks Reference: Databricks' "Generative AI Engineer Guide" notes that sentiment analysis is a common LLM task, but it's not highlighted for real-time conversational applications like customer support.
Conclusion: Option C is the best fit because it uses relevant input (chat logs) and defines an LLM task (finding answers and summarizing) that meets the requirements of improving response times and maintaining personalized interaction. This aligns with Databricks' recommended practices for LLM-powered customer service solutions, such as retrieval-augmented generation (RAG) workflows.

NEW QUESTION: 10

A Generative AI Engineer is designing an LLM-powered live sports commentary platform. The platform provides real-time updates and LLM-generated analyses for any users who would like to have live summaries, rather than reading a series of potentially outdated news articles.
Which tool below will give the platform access to real-time data for generating game analyses based on the latest game scores?
A. Foundation Model APIs
B. DatabricksIQ
C. AutoML
D. Feature Serving
Answer: D (LEAVE A REPLY)
* Problem Context: The engineer is developing an LLM-powered live sports commentary platform that needs to provide real-time updates and analyses based on the latest game scores. The critical requirement here is the capability to access and integrate real-time data efficiently with the platform for immediate analysis and reporting.
* Explanation of Options:
* Option A: DatabricksIQ: While DatabricksIQ offers integration and data processing capabilities, it is more aligned with data analytics rather than real-time feature serving, which is crucial for immediate updates necessary in a live sports commentary context.
* Option B: Foundation Model APIs: These APIs facilitate interactions with pre-trained models and could be part of the solution, but on their own, they do not provide mechanisms to access real- time game scores.
* Option C: Feature Serving: This is the correct answer as feature serving specifically refers to the real-time provision of data (features) to models for prediction. This would be essential for an LLM that generates analyses based on live game data, ensuring that the commentary is current and based on the latest events in the sport.
* Option D: AutoML: This tool automates the process of applying machine learning models to real-world problems, but it does not directly provide real-time data access, which is a critical requirement for the platform.
Thus,Option C(Feature Serving) is the most suitable tool for the platform as it directly supports the real-time data needs of an LLM-powered sports commentary system, ensuring that the analyses and updates are based on the latest available information.

NEW QUESTION: 11
A small and cost-conscious startup in the cancer research field wants to build a RAG application using Foundation Model APIs.
Which strategy would allow the startup to build a good-quality RAG application while being cost-conscious and able to cater to customer needs?
A. Limit the number of relevant documents available for the RAG application to retrieve from
B. Pick a smaller LLM that is domain-specific
C. Limit the number of queries a customer can send per day
D. Use the largest LLM possible because that gives the best performance for any general queries

Answer: B (LEAVE A REPLY)
For a small, cost-conscious startup in the cancer research field, choosing a domain-specific and smaller LLM is the most effective strategy. Here's whyBis the best choice:
* Domain-specific performance: A smaller LLM that has been fine-tuned for the domain of cancer research will outperform a general-purpose LLM for specialized queries. This ensures high-quality responses without needing to rely on a large, expensive LLM.
* Cost-efficiency: Smaller models are cheaper to run, both in terms of compute resources and API usage costs. A domain-specific smaller LLM can deliver good quality responses without the need for the extensive computational power required by larger models.
* Focused knowledge: In a specialized field like cancer research, having an LLM tailored to the subject matter provides better relevance and accuracy for queries, while keeping costs low.Large, general-purpose LLMs may provide irrelevant information, leading to inefficiency and higher costs.
This approach allows the startup to balance quality, cost, and customer satisfaction effectively, making it the most suitable strategy.

NEW QUESTION: 12
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?
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 (LEAVE A REPLY)
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.

NEW QUESTION: 13
A Generative AI Engineer is building an LLM-based application that has an important transcription (speech-to-text) task. Speed is essential for the success of the application Which open Generative AI models should be used?
A. L!ama-2-70b-chat-hf
B. MPT-30B-Instruct
C. DBRX
D. whisper-large-v3 (1.6B)
Answer: D (LEAVE A REPLY)
The task requires an open generative AI model for a transcription (speech-to-text) task where speed is essential. Let's assess the options based on their suitability for transcription and performance characteristics, referencing Databricks' approach to model selection.
* Option A: Llama-2-70b-chat-hf
* Llama-2 is a text-based LLM optimized for chat and text generation, not speech-to-text. It lacks transcription capabilities.
* Databricks Reference:"Llama models are designed for natural language generation, not audio processing"("Databricks Model Catalog").
* Option B: MPT-30B-Instruct
* MPT-30B is another text-based LLM focused on instruction-following and text generation, not transcription. It's irrelevant for speech-to-text tasks.
* Databricks Reference: No specific mention, but MPT is categorized under text LLMs in Databricks' ecosystem, not audio models.
* Option C: DBRX

* DBRX, developed by Databricks, is a powerful text-based LLM for general-purpose generation.
It doesn't natively support speech-to-text and isn't optimized for transcription.
* Databricks Reference:"DBRX excels at text generation and reasoning tasks"("Introducing DBRX," 2023)-no mention of audio capabilities.
* Option D: whisper-large-v3 (1.6B)
* Whisper, developed by OpenAI, is an open-source model specifically designed for speech-to-text transcription. The "large-v3" variant (1.6 billion parameters) balances accuracy and efficiency, with optimizations for speed via quantization or deployment on GPUs-key for the application's requirements.
* Databricks Reference:"For audio transcription, models like Whisper are recommended for their speed and accuracy"("Generative AI Cookbook," 2023). Databricks supports Whisper integration in its MLflow or Lakehouse workflows.
Conclusion: OnlyD. whisper-large-v3is a speech-to-text model, making it the sole suitable choice. Its design prioritizes transcription, and its efficiency (e.g., via optimized inference) meets the speed requirement, aligning with Databricks' model deployment best practices.

NEW QUESTION: 14
A Generative AI Engineer is developing a patient-facing healthcare-focused chatbot. If the patient's question is not a medical emergency, the chatbot should solicit more information from the patient to pass to the doctor' s office and suggest a few relevant pre-approved medical articles for reading. If the patient's question is urgent, direct the patient to calling their local emergency services.
Given the following user input:
"I have been experiencing severe headaches and dizziness for the past two days." Which response is most appropriate for the chatbot to generate?
A. Here are a few relevant articles for your browsing. Let me know if you have questions after reading them.
B. Please call your local emergency services.
C. Headaches can be tough. Hope you feel better soon!
D. Please provide your age, recent activities, and any other symptoms you have noticed along with your headaches and dizziness.
Answer: B (LEAVE A REPLY)
* Problem Context: The task is to design responses for a healthcare-focused chatbot that appropriately addresses the urgency of a patient's symptoms.
* Explanation of Options:
* Option A: Suggesting articles might be suitable for less urgent inquiries but is inappropriate for symptoms that could indicate a serious condition.
* Option B: Given the description of severe symptoms like headaches and dizziness, directing the patient to emergency services is prudent. This aligns with medical guidelines that recommend immediate professional attention for such severe symptoms.
* Option C: Offering well-wishes does not address the potential seriousness of the symptoms and lacks appropriate action.

* Option D: While gathering more information is part of a detailed assessment, the immediate need here suggests a more urgent response.
Given the potential severity of the described symptoms,Option Bis the most appropriate, ensuring the chatbot directs patients to seek urgent care when needed, potentially saving lives.

NEW QUESTION: 15
A Generative AI Engineer is creating an agent-based LLM system for their favorite monster truck team. The system can answer text based questions about the monster truck team, lookup event dates via an API call, or query tables on the team's latest standings.
How could the Generative AI Engineer best design these capabilities into their system?
A. Ingest PDF documents about the monster truck team into a vector store and query it in a RAG architecture.
B. Write a system prompt for the agent listing available tools and bundle it into an agent system that runs a number of calls to solve a query.
C. Instruct the LLM to respond with "RAG", "API", or "TABLE" depending on the query, then use text parsing and conditional statements to resolve the query.
D. Build a system prompt with all possible event dates and table information in the system prompt. Use a RAG architecture to lookup generic text questions and otherwise leverage the information in the system prompt.
Answer: B (LEAVE A REPLY)
In this scenario, the Generative AI Engineer needs to design a system that can handle different types of queries about the monster truck team. The queries may involve text-based information, API lookups for event dates, or table queries for standings. The best solution is to implement a tool-based agent system. Here's how option B works, and why it's the most appropriate answer:
System Design Using Agent-Based Model:
In modern agent-based LLM systems, you can design a system where the LLM (Large Language Model) acts as a central orchestrator. The model can "decide" which tools to use based on the query. These tools can include API calls, table lookups, or natural language searches. The system should contain a system prompt that informs the LLM about the available tools.
System Prompt Listing Tools:
By creating a well-crafted system prompt, the LLM knows which tools are at its disposal. For instance, one tool may query an external API for event dates, another might look up standings in a database, and a third may involve searching a vector database for general text-based information. The agent will be responsible for calling the appropriate tool depending on the query.
Agent Orchestration of Calls:
The agent system is designed to execute a series of steps based on the incoming query. If a user asks for the next event date, the system will recognize this as a task that requires an API call. If the user asks about standings, the agent might query the appropriate table in the database. For text-based questions, it may call a search function over ingested data. The agent orchestrates this entire process, ensuring the LLM makes calls to the right resources dynamically.
Generative AI Tools and Context:

This is a standard architecture for integrating multiple functionalities into a system where each query requires different actions. The core design in option B is efficient because it keeps the system modular and dynamic by leveraging tools rather than overloading the LLM with static information in a system prompt (like option D).
Why Other Options Are Less Suitable:
A (RAG Architecture): While relevant, simply ingesting PDFs into a vector store only helps with text-based retrieval. It wouldn't help with API lookups or table queries.
C (Conditional Logic with RAG/API/TABLE): Although this approach works, it relies heavily on manual text parsing and might introduce complexity when scaling the system.
D (System Prompt with Event Dates and Standings): Hardcoding dates and table information into a system prompt isn't scalable. As the standings or events change, the system would need constant updating, making it inefficient.
By bundling multiple tools into a single agent-based system (as in option B), the Generative AI Engineer can best handle the diverse requirements of this system.

NEW QUESTION: 16
A Generative AI Engineer is building a Generative AI system that suggests the best matched employee team member to newly scoped projects. The team member is selected from a very large team. Thematch should be based upon project date availability and how well their employee profile matches the project scope. Both the employee profile and project scope are unstructured text.
How should the Generative AI Engineer architect their system?
A. Create a tool for finding available team members given project dates. Embed all project scopes into a vector store, perform a retrieval using team member profiles to find the best team member.
B. Create a tool for finding team member availability given project dates, and another tool that uses an LLM to extract keywords from project scopes. Iterate through available team members' profiles and perform keyword matching to find the best available team member.
C. Create a tool to find available team members given project dates. Create a second tool that can calculate a similarity score for a combination of team member profile and the project scope. Iterate through the team members and rank by best score to select a team member.
D. Create a tool for finding available team members given project dates. Embed team profiles into a vector store and use the project scope and filtering to perform retrieval to find the available best matched team members.
Answer: D (LEAVE A REPLY)
* Problem Context: The problem involves matching team members to new projects based on two main factors:
* Availability: Ensure the team members are available during the project dates.
* Profile-Project Match: Use the employee profiles (unstructured text) to find the best match for a project's scope (also unstructured text).
The two main inputs are theemployee profilesandproject scopes, both of which are unstructured. This means traditional rule-based systems (e.g., simple keyword matching) would be inefficient, especially when working with large datasets.

* Explanation of Options: Let's break down the provided options to understand why D is the most optimal answer.
* Option Asuggests embedding project scopes into a vector store and then performing retrieval using team member profiles. While embedding project scopes into a vector store is a valid technique, it skips an important detail: the focus should primarily be on embedding employee profiles because we're matching the profiles to a new project, not the other way around.
* Option Binvolves using a large language model (LLM) to extract keywords from the project scope and perform keyword matching on employee profiles. While LLMs can help with keyword extraction, this approach is too simplistic and doesn't leverage advanced retrieval techniques like vector embeddings, which can handle the nuanced and rich semantics of unstructured data. This approach may miss out on subtle but important similarities.
* Option Csuggests calculating a similarity score between each team member's profile and project scope. While this is a good idea, it doesn't specify how to handle the unstructured nature of data efficiently. Iterating through each member's profile individually could be computationally expensive in large teams. It also lacks the mention of using a vector store or an efficient retrieval mechanism.
* Option Dis the correct approach. Here's why:
* Embedding team profiles into a vector store: Using a vector store allows for efficient similarity searches on unstructured data. Embedding the team member profiles into vectors captures their semantics in a way that is far more flexible than keyword-based matching.
* Using project scope for retrieval: Instead of matching keywords, this approach suggests using vector embeddings and similarity search algorithms (e.g., cosine similarity) to find the team members whose profiles most closely align with the project scope.
* Filtering based on availability: Once the best-matched candidates are retrieved based on profile similarity, filtering them by availability ensures that the system provides a practically useful result.
This method efficiently handles large-scale datasets by leveragingvector embeddingsandsimilarity search techniques, both of which are fundamental tools inGenerative AI engineeringfor handling unstructured text.
* Technical References:
* Vector embeddings: In this approach, the unstructured text (employee profiles and project scopes) is converted into high-dimensional vectors using pretrained models (e.g., BERT, Sentence-BERT, or custom embeddings). These embeddings capture the semantic meaning of the text, making it easier to perform similarity-based retrieval.
* Vector stores: Solutions likeFAISSorMilvusallow storing and retrieving large numbers of vector embeddings quickly. This is critical when working with large teams where querying through individual profiles sequentially would be inefficient.
* LLM Integration: Large language models can assist in generating embeddings for both employee profiles and project scopes. They can also assist in fine-tuning similarity measures, ensuring that the retrieval system captures the nuances of the text data.
* Filtering: After retrieving the most similar profiles based on the project scope, filtering based on availability ensures that only team members who are free for the project are considered.

This system is scalable, efficient, and makes use of the latest techniques inGenerative AI, such as vector embeddings and semantic search.

NEW QUESTION: 17
A Generative AI Engineer is building a production-ready LLM system which replies directly to customers. The solution makes use of the Foundation Model API via provisioned throughput. They are concerned that the LLM could potentially respond in a toxic or otherwise unsafe way. They also wish to perform this with the least amount of effort.
Which approach will do this?
A. Host Llama Guard on Foundation Model API and use it to detect unsafe responses
B. Add some LLM calls to their chain to detect unsafe content before returning text
C. Add a regex expression on inputs and outputs to detect unsafe responses.
D. Ask users to report unsafe responses
Answer: A (LEAVE A REPLY)
The task is to prevent toxic or unsafe responses in an LLM system using the Foundation Model API with minimal effort. Let's assess the options.
Option A: Host Llama Guard on Foundation Model API and use it to detect unsafe responses Llama Guard is a safety-focused model designed to detect toxic or unsafe content. Hosting it via the Foundation Model API (a Databricks service) integrates seamlessly with the existing system, requiring minimal setup (just deployment and a check step), and leverages provisioned throughput for performance.
Databricks Reference: "Foundation Model API supports hosting safety models like Llama Guard to filter outputs efficiently" ("Foundation Model API Documentation," 2023).
Option B: Add some LLM calls to their chain to detect unsafe content before returning text Using additional LLM calls (e.g., prompting an LLM to classify toxicity) increases latency, complexity, and effort (crafting prompts, chaining logic), and lacks the specificity of a dedicated safety model.
Databricks Reference: "Ad-hoc LLM checks are less efficient than purpose-built safety solutions" ("Building LLM Applications with Databricks").
Option C: Add a regex expression on inputs and outputs to detect unsafe responses Regex can catch simple patterns (e.g., profanity) but fails for nuanced toxicity (e.g., sarcasm, context-dependent harm), requiring significant manual effort to maintain and update rules.

Databricks Reference: "Regex-based filtering is limited for complex safety needs" ("Generative AI Cookbook").
Option D: Ask users to report unsafe responses
User reporting is reactive, not preventive, and places burden on users rather than the system. It doesn't limit unsafe outputs proactively and requires additional effort for feedback handling.
Databricks Reference: "Proactive guardrails are preferred over user-driven monitoring" ("Databricks Generative AI Engineer Guide").
Conclusion: Option A (Llama Guard on Foundation Model API) is the least-effort, most effective approach, leveraging Databricks' infrastructure for seamless safety integration.

NEW QUESTION: 18
A Generative AI Engineer is developing an agent system using a popular agent-authoring library. The agent comprises multiple parallel and sequential chains. The engineer encounters challenges as the agent fails at one of the steps, making it difficult to debug the root cause. They need to find an appropriate approach to research this issue and discover the cause of failure. Which approach do they choose?
A. Enable MLflow tracing to gain visibility into each agent's behavior and execution step.
B. Run MLflow.evaluate to determine root cause of failed step.
C. Implement structured logging within the agent's code to capture detailed execution information.
D. Deconstruct the agent into independent steps to simplify debugging.
Answer: A (LEAVE A REPLY)
For complex agentic systems (like those built with LangGraph or Autogen), standard logging is often insufficient because the "state" of the agent changes dynamically. MLflow Tracing is the designated Generative AI engineering standard for debugging these systems. Tracing provides a visual, hierarchical timeline of every call made during an agent's execution-including internal LLM reasoning, tool calls, and data transformations. When a step fails, the trace allows the engineer to click into that specific node to see the exact input sent to the LLM and the raw output received. This is much faster and more comprehensive than manually deconstructing the agent (D) or adding manual logs (C). While mlflow.evaluate (B) is useful for measuring performance across a whole dataset, it is not a tool for real-time debugging of a single execution failure.

NEW QUESTION: 19
A Generative AI Engineer is creating an LLM-based application. The documents for its retriever have been chunked to a maximum of 512 tokens each. The Generative AI Engineer knows that cost and latency are more important than quality for this application. They have several context length levels to choose from.
Which will fulfill their need?
A. context length 514; smallest model is 0.44GB and embedding dimension 768
B. context length 2048: smallest model is 11GB and embedding dimension 2560
C. context length 32768: smallest model is 14GB and embedding dimension 4096
D. context length 512: smallest model is 0.13GB and embedding dimension 384

Answer: D (LEAVE A REPLY)
When prioritizing cost and latency over quality in a Large Language Model (LLM)-based application, it is crucial to select a configuration that minimizes both computational resources and latency while still providing reasonable performance. Here's whyDis the best choice:
* Context length: The context length of 512 tokens aligns with the chunk size used for the documents (maximum of 512 tokens per chunk). This is sufficient for capturing the needed information and generating responses without unnecessary overhead.
* Smallest model size: The model with a size of 0.13GB is significantly smaller than the other options. This small footprint ensures faster inference times and lower memory usage, which directly reduces both latency and cost.
* Embedding dimension: While the embedding dimension of 384 is smaller than the other options, it is still adequate for tasks where cost and speed are more important than precision and depth of understanding.
This setup achieves the desired balance between cost-efficiency and reasonable performance in a latency- sensitive, cost-conscious application.

NEW QUESTION: 20
A Generative AI Engineer is creating an LLM-based application. The documents for its retriever have been chunked to a maximum of 512 tokens each. The Generative AI Engineer knows that cost and latency are more important than quality for this application. They have several context length levels to choose from.
Which will fulfill their need?
A. context length 514; smallest model is 0.44GB and embedding dimension 768
B. context length 2048: smallest model is 11GB and embedding dimension 2560
C. context length 32768: smallest model is 14GB and embedding dimension 4096
D. context length 512: smallest model is 0.13GB and embedding dimension 384
Answer: (SHOW ANSWER)
When prioritizing cost and latency over quality in a Large Language Model (LLM)-based application, it is crucial to select a configuration that minimizes both computational resources and latency while still providing reasonable performance. Here's why D is the best choice:
Context length: The context length of 512 tokens aligns with the chunk size used for the documents (maximum of 512 tokens per chunk). This is sufficient for capturing the needed information and generating responses without unnecessary overhead.
Smallest model size: The model with a size of 0.13GB is significantly smaller than the other options. This small footprint ensures faster inference times and lower memory usage, which directly reduces both latency and cost.
Embedding dimension: While the embedding dimension of 384 is smaller than the other options, it is still adequate for tasks where cost and speed are more important than precision and depth of understanding.
This setup achieves the desired balance between cost-efficiency and reasonable performance in a latency-sensitive, cost-conscious application.

NEW QUESTION: 21
A Generative AI Engineer is building a RAG application that answers questions about internal documents for the company SnoPen AI.
The source documents may contain a significant amount of irrelevant content, such as advertisements, sports news, or entertainment news, or content about other companies.
Which approach is advisable when building a RAG application to achieve this goal of filtering irrelevant information?
A. Keep all articles because the RAG application needs to understand non-company content to avoid answering questions about them.
B. Include in the system prompt that any information it sees will be about SnoPenAI, even if no data filtering is performed.
C. Include in the system prompt that the application is not supposed to answer any questions unrelated to SnoPen AI.
D. Consolidate all SnoPen AI related documents into a single chunk in the vector database.
Answer: (SHOW ANSWER)
In a Retrieval-Augmented Generation (RAG) application built to answer questions about internal documents, especially when the dataset contains irrelevant content, it's crucial to guide the system to focus on the right information. The best way to achieve this is by including a clear instruction in the system prompt (option C).
System Prompt as Guidance:
The system prompt is an effective way to instruct the LLM to limit its focus to SnoPen AI-related content. By clearly specifying that the model should avoid answering questions unrelated to SnoPen AI, you add an additional layer of control that helps the model stay on-topic, even if irrelevant content is present in the dataset.
Why This Approach Works:
The prompt acts as a guiding principle for the model, narrowing its focus to specific domains. This prevents the model from generating answers based on irrelevant content, such as advertisements or news unrelated to SnoPen AI.
Why Other Options Are Less Suitable:
A (Keep All Articles): Retaining all content, including irrelevant materials, without any filtering makes the system prone to generating answers based on unwanted data.
B (Include in the System Prompt about SnoPen AI): This option doesn't address irrelevant content directly, and without filtering, the model might still retrieve and use irrelevant data.
D (Consolidating Documents into a Single Chunk): Grouping documents into a single chunk makes the retrieval process less efficient and won't help filter out irrelevant content effectively.
Therefore, instructing the system in the prompt not to answer questions unrelated to SnoPen AI (option C) is the best approach to ensure the system filters out irrelevant information.

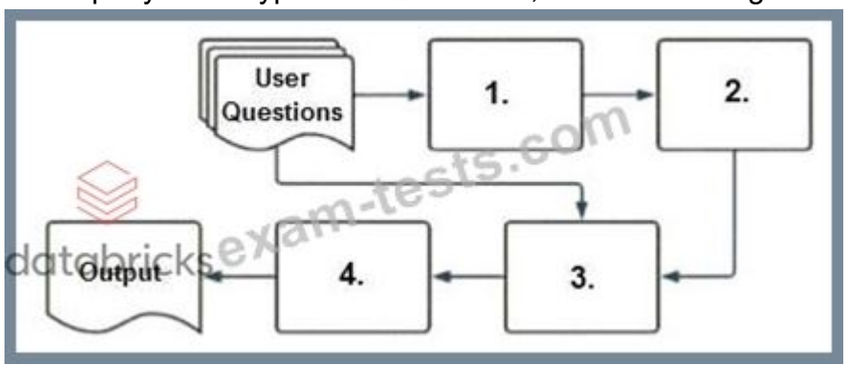
NEW QUESTION: 22
A Generative AI Engineer just deployed an LLM application at a digital marketing company that assists with answering customer service inquiries.

Which metric should they monitor for their customer service LLM application in production?
A. Number of customer inquiries processed per unit of time
B. Energy usage per query
C. Final perplexity scores for the training of the model
D. HuggingFace Leaderboard values for the base LLM
Answer: A (LEAVE A REPLY)
When deploying an LLM application for customer service inquiries, the primary focus is on measuring the operational efficiency and quality of the responses. Here's whyAis the correct metric:
* Number of customer inquiries processed per unit of time: This metric tracks the throughput of the customer service system, reflecting how many customer inquiries the LLM application can handle in a given time period (e.g., per minute or hour). High throughput is crucial in customer service applications where quick response times are essential to user satisfaction and business efficiency.
* Real-time performance monitoring: Monitoring the number of queries processed is an important part of ensuring that the model is performing well under load, especially during peak traffic times. It also helps ensure the system scales properly to meet demand.
Why other options are not ideal:
* B. Energy usage per query: While energy efficiency is a consideration, it is not the primary concern for a customer-facing application where user experience (i.e., fast and accurate responses) is critical.
* C. Final perplexity scores for the training of the model: Perplexity is a metric for model training, but it doesn't reflect the real-time operational performance of an LLM in production.
* D. HuggingFace Leaderboard values for the base LLM: The HuggingFace Leaderboard is more relevant during model selection and benchmarking. However, it is not a direct measure of the model's performance in a specific customer service application in production.
Focusing on throughput (inquiries processed per unit time) ensures that the LLM application is meeting business needs for fast and efficient customer service responses.

NEW QUESTION: 23
A Generative AI Engineer is developing a chatbot designed to assist users with insurance-related queries. The chatbot is built on a large language model (LLM) and is conversational. However, to maintain the chatbot's focus and to comply with company policy, it must not provide responses to questions about politics. Instead, when presented with political inquiries, the chatbot should respond with a standard message:
"Sorry, I cannot answer that. I am a chatbot that can only answer questions around insurance." Which framework type should be implemented to solve this?
A. Safety Guardrail
B. Security Guardrail
C. Contextual Guardrail
D. Compliance Guardrail
Answer: A (LEAVE A REPLY)
In this scenario, the chatbot must avoid answering political questions and instead provide a standard message for such inquiries. Implementing a Safety Guardrail is the appropriate solution for this:

What is a Safety Guardrail?
Safety guardrails are mechanisms implemented in Generative AI systems to ensure the model behaves within specific bounds. In this case, it ensures the chatbot does not answer politically sensitive or irrelevant questions, which aligns with the business rules.
Preventing Responses to Political Questions:
The Safety Guardrail is programmed to detect specific types of inquiries (like political questions) and prevent the model from generating responses outside its intended domain. When such queries are detected, the guardrail intervenes and provides a pre-defined response: "Sorry, I cannot answer that. I am a chatbot that can only answer questions around insurance." How It Works in Practice:
The LLM system can include a classification layer or trigger rules based on specific keywords related to politics. When such terms are detected, the Safety Guardrail blocks the normal generation flow and responds with the fixed message.
Why Other Options Are Less Suitable:
B (Security Guardrail): This is more focused on protecting the system from security vulnerabilities or data breaches, not controlling the conversational focus.
C (Contextual Guardrail): While context guardrails can limit responses based on context, safety guardrails are specifically about ensuring the chatbot stays within a safe conversational scope.
D (Compliance Guardrail): Compliance guardrails are often related to legal and regulatory adherence, which is not directly relevant here.
Therefore, a Safety Guardrail is the right framework to ensure the chatbot only answers insurance-related queries and avoids political discussions.

NEW QUESTION: 24
A company has a typical RAG-enabled, customer-facing chatbot on its website.



Select the correct sequence of components a user's questions will go through before the final output is returned. Use the diagram above for reference.
A. 1.embedding model, 2.vector search, 3.context-augmented prompt, 4.response-generating LLM
B. 1.context-augmented prompt, 2.vector search, 3.embedding model, 4.response-generating LLM
C. 1.response-generating LLM, 2.vector search, 3.context-augmented prompt, 4.embedding model
D. 1.response-generating LLM, 2.context-augmented prompt, 3.vector search, 4.embedding model
Answer: (SHOW ANSWER)
To understand how a typical RAG-enabled customer-facing chatbot processes a user's question, let's go through the correct sequence as depicted in the diagram and explained in option A:

* Embedding Model (1):The first step involves the user's question being processed through an embedding model. This model converts the text into a vector format that numerically represents the text. This step is essential for allowing the subsequent vector search to operate effectively.
* Vector Search (2):The vectors generated by the embedding model are then used in a vector search mechanism. This search identifies the most relevant documents or previously answered questions that are stored in a vector format in a database.
* Context-Augmented Prompt (3):The information retrieved from the vector search is used to create a context-augmented prompt. This step involves enhancing the basic user query with additional relevant information gathered to ensure the generated response is as accurate and informative as possible.
* Response-Generating LLM (4):Finally, the context-augmented prompt is fed into a response-generating large language model (LLM). This LLM uses the prompt to generate a coherent and contextually appropriate answer, which is then delivered as the final output to the user.
Why Other Options Are Less Suitable:
* B, C, D: These options suggest incorrect sequences that do not align with how a RAG system typically processes queries. They misplace the role of embedding models, vector search, and response generation in an order that would not facilitate effective information retrieval and response generation.
Thus, the correct sequence isembedding model, vector search, context-augmented prompt, response-generating LLM, which is option A.

NEW QUESTION: 25
A Generative AI Engineer is testing a simple prompt template in LangChain using the code below, but is getting an error.

```
from langchain.chains import LLMChain
from langchain_community.llms import OpenAI
from langchain_core.prompts import PromptTemplate

prompt_template = "Tell me a {adjective} joke"

prompt = PromptTemplate(
    input_variables=["adjective"],
    template=prompt_template
)

llm = LLMChain(prompt=prompt)
llm.generate([{"adjective": "funny"}])
```


Assuming the API key was properly defined, what change does the Generative AI Engineer need to make to fix their chain?

```
prompt_template = "Tell me a {adjective} joke"

prompt = PromptTemplate(
     input_variables=["adjective"],
     template=prompt_template
)

llm = LLMChain(prompt=prompt)
llm.generate("funny")
```

A.

B.

```
prompt_template = "Tell me a {adjective} joke"

prompt = PromptTemplate(
     input_variables=["adjective"],
     template=prompt_template
)

llm = LLMChain(prompt=prompt.format("funny"))
llm.generate()
```

C.

```
prompt_template = "Tell me a {adjective} joke"

prompt = PromptTemplate(
     input_variables=["adjective"],
     template=prompt_template
     llm=OpenAI()
)

llm = LLMChain(prompt=prompt)
llm.generate([{"adjective": "funny"}])
```

D.

```
prompt_template = "Tell me a {adjective} joke"

prompt = PromptTemplate(
     input_variables=["adjective"],
     template=prompt_template
)

llm = LLMChain(llm=OpenAI(), prompt=prompt)
llm.generate([{"adjective": "funny"}])
```

Answer: (SHOW ANSWER)

To fix the error in the LangChain code provided for using a simple prompt template, the correct approach is Option C. Here's a detailed breakdown of why Option C is the right choice and how it addresses the issue:

Proper Initialization: In Option C, the LLMChain is correctly initialized with the LLM instance specified as OpenAI(), which likely represents a language model (like GPT) from OpenAI. This is crucial as it specifies which model to use for generating responses.

Correct Use of Classes and Methods:

The PromptTemplate is defined with the correct format, specifying that adjective is a variable within the template. This allows dynamic insertion of values into the template when generating text.

The prompt variable is properly linked with the PromptTemplate, and the final template string is passed correctly.
The LLMChain correctly references the prompt and the initialized OpenAI() instance, ensuring that the template and the model are properly linked for generating output.
Why Other Options Are Incorrect:
Option A: Misuses the parameter passing in generate method by incorrectly structuring the dictionary.
Option B: Incorrectly uses prompt.format method which does not exist in the context of LLMChain and PromptTemplate configuration, resulting in potential errors.
Option D: Incorrect order and setup in the initialization parameters for LLMChain, which would likely lead to a failure in recognizing the correct configuration for prompt and LLM usage.
Thus, Option C is correct because it ensures that the LangChain components are correctly set up and integrated, adhering to proper syntax and logical flow required by LangChain's architecture. This setup avoids common pitfalls such as type errors or method misuses, which are evident in other options.

NEW QUESTION: 26
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?
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 (LEAVE A REPLY)
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.

NEW QUESTION: 27
A team wants to serve a code generation model as an assistant for their software developers. It should support multiple programming languages. Quality is the primary objective.
Which of the Databricks Foundation Model APIs, or models available in the Marketplace, would be the best fit?
A. Llama2-70b
B. BGE-large
C. MPT-7b
D. CodeLlama-34B
Answer: D (LEAVE A REPLY)
For a code generation model that supports multiple programming languages and where quality is the primary objective,CodeLlama-34Bis the most suitable choice. Here's the reasoning:
* Specialization in Code Generation:CodeLlama-34B is specifically designed for code generation tasks. This model has been trained with a focus on understanding and generating code, which makes it particularly adept at handling various programming languages and coding contexts.

* Capacity and Performance:The "34B" indicates a model size of 34 billion parameters, suggesting a high capacity for handling complex tasks and generating high-quality outputs. The large model size typically correlates with better understanding and generation capabilities in diverse scenarios.
* Suitability for Development Teams:Given that the model is optimized for code, it will be able to assist software developers more effectively than general-purpose models. It understands coding syntax, semantics, and the nuances of different programming languages.
* Why Other Options Are Less Suitable:
* A (Llama2-70b): While also a large model, it's more general-purpose and may not be as fine- tuned for code generation as CodeLlama.
* B (BGE-large): This model may not specifically focus on code generation.
* C (MPT-7b): Smaller than CodeLlama-34B and likely less capable in handling complex code generation tasks at high quality.
Therefore, for a high-quality, multi-language code generation application,CodeLlama-34B(option D) is the best fit.

NEW QUESTION: 28
A Generative AI Engineer received the following business requirements for an external chatbot.
The chatbot needs to know what types of questions the user asks and routes to appropriate models to answer the questions. For example, the user might ask about upcoming event details. Another user might ask about purchasing tickets for a particular event.
What is an ideal workflow for such a chatbot?
A. The chatbot should only look at previous event information
B. There should be two different chatbots handling different types of user queries.
C. The chatbot should be implemented as a multi-step LLM workflow. First, identify the type of question asked, then route the question to the appropriate model. If it's an upcoming event question, send the query to a text-to-SQL model. If it's about ticket purchasing, the customer should be redirected to a payment platform.
D. The chatbot should only process payments
Answer: (SHOW ANSWER)
* Problem Context: The chatbot must handle various types of queries and intelligently route them to the appropriate responses or systems.
* Explanation of Options:
* Option A: Limiting the chatbot to only previous event information restricts its utility and does not meet the broader business requirements.
* Option B: Having two separate chatbots could unnecessarily complicate user interaction and increase maintenance overhead.
* Option C: Implementing a multi-step workflow where the chatbot first identifies the type of question and then routes it accordingly is the most efficient and scalable solution. This approach allows the chatbot to handle a variety of queries dynamically, improving user experience and operational efficiency.
* Option D: Focusing solely on payments would not satisfy all the specified user interaction needs, such as inquiring about event details.

Option Coffers a comprehensive workflow that maximizes the chatbot's utility and responsiveness to different user needs, aligning perfectly with the business requirements.

NEW QUESTION: 29
A Generative AI Engineer is tasked with developing a RAG application that will help a small internal group of experts at their company answer specific questions, augmented by an internal knowledge base. They want the best possible quality in the answers, and neither latency nor throughput is a huge concern given that the user group is small and they're willing to wait for the best answer. The topics are sensitive in nature and the data is highly confidential and so, due to regulatory requirements, none of the information is allowed to be transmitted to third parties.
Which model meets all the Generative AI Engineer's needs in this situation?
A. Dolly 1.5B
B. OpenAI GPT-4
C. BGE-large
D. Llama2-70B
Answer: (SHOW ANSWER)
Problem Context: The Generative AI Engineer needs a model for a Retrieval-Augmented Generation (RAG) application that provides high-quality answers, where latency and throughput are not major concerns. The key factors areconfidentialityandsensitivityof the data, as well as the requirement for all processing to be confined to internal resources without external data transmission.
Explanation of Options:
* Option A: Dolly 1.5B: This model does not typically support RAG applications as it's more focused on image generation tasks.
* Option B: OpenAI GPT-4: While GPT-4 is powerful for generating responses, its standard deployment involves cloud-based processing, which could violate the confidentiality requirements due to external data transmission.
* Option C: BGE-large: The BGE (Big Green Engine) large model is a suitable choice if it is configured to operate on-premises or within a secure internal environment that meets regulatory requirements. Assuming this setup, BGE-large can provide high-quality answers while ensuring that data is not transmitted to third parties, thus aligning with the project's sensitivity and confidentiality needs.
* Option D: Llama2-70B: Similar to GPT-4, unless specifically set up for on-premises use, it generally relies on cloud-based services, which might risk confidential data exposure.
Given the sensitivity and confidentiality concerns,BGE-largeis assumed to be configurable for secure internal use, making it the optimal choice for this scenario.

NEW QUESTION: 30
A Generative AI Engineer has built an LLM-based system that will automatically translate user text between two languages. They now want to benchmark multiple LLM's on this task and pick the best one. They have an evaluation set with known high quality translation examples. They want to evaluate each LLM using the evaluation set with a performant metric.
Which metric should they choose for this evaluation?

A. ROUGE metric
B. BLEU metric
C. NDCG metric
D. RECALL metric
Answer: B (LEAVE A REPLY)
The task is to benchmark LLMs for text translation using an evaluation set with known high-quality examples, requiring a performant metric. Let's evaluate the options.
* Option A: ROUGE metric
* ROUGE (Recall-Oriented Understudy for Gisting Evaluation) measures overlap between generated and reference texts, primarily for summarization. It's less suited for translation, where precision and word order matter more.
* Databricks Reference:"ROUGE is commonly used for summarization, not translation evaluation"("Generative AI Cookbook," 2023).
* Option B: BLEU metric
* BLEU (Bilingual Evaluation Understudy) evaluates translation quality by comparing n-gram overlap with reference translations, accounting for precision and brevity. It's widely used, performant, and appropriate for this task.
* Databricks Reference:"BLEU is a standard metric for evaluating machine translation, balancing accuracy and efficiency"("Building LLM Applications with Databricks").
* Option C: NDCG metric
* NDCG (Normalized Discounted Cumulative Gain) assesses ranking quality, not text generation. It's irrelevant for translation evaluation.
* Databricks Reference:"NDCG is suited for ranking tasks, not generative output scoring" ("Databricks Generative AI Engineer Guide").
* Option D: RECALL metric
* Recall measures retrieved relevant items but doesn't evaluate translation quality (e.g., fluency, correctness). It's incomplete for this use case.
* Databricks Reference: No specific extract, but recall alone lacks the granularity of BLEU for text generation tasks.
Conclusion: Option B (BLEU) is the best metric for translation evaluation, offering a performant and standard approach, as endorsed by Databricks' guidance on generative tasks.

NEW QUESTION: 31
Which TWO chain components are required for building a basic LLM-enabled chat application that includes conversational capabilities, knowledge retrieval, and contextual memory?
A. (Q)
B. Vector Stores
C. Conversation Buffer Memory
D. External tools
E. Chat loaders
F. React Components

Answer: B,C (LEAVE A REPLY)

Building a basic LLM-enabled chat application with conversational capabilities, knowledge retrieval, and contextual memory requires specific components that work together to process queries, maintain context, and retrieve relevant information. Databricks' Generative AI Engineer documentation outlines key components for such systems, particularly in the context of frameworks like LangChain or Databricks' MosaicML integrations. Let's evaluate the required components:

* Understanding the Requirements:
* Conversational capabilities: The app must generate natural, coherent responses.
* Knowledge retrieval: It must access external or domain-specific knowledge.
* Contextual memory: It must remember prior interactions in the conversation.
* Databricks Reference:"A typical LLM chat application includes a memory component to track conversation history and a retrieval mechanism to incorporate external knowledge"("Databricks Generative AI Cookbook," 2023).
* Evaluating the Options:
* A. (Q): This appears incomplete or unclear (possibly a typo). Without further context, it's not a valid component.
* B. Vector Stores: These store embeddings of documents or knowledge bases, enabling semantic search and retrieval of relevant information for the LLM. This is critical for knowledge retrieval in a chat application.
* Databricks Reference:"Vector stores, such as those integrated with Databricks' Lakehouse, enable efficient retrieval of contextual data for LLMs"("Building LLM Applications with Databricks").
* C. Conversation Buffer Memory: This component stores the conversation history, allowing the LLM to maintain context across multiple turns. It's essential for contextual memory.
* Databricks Reference:"Conversation Buffer Memory tracks prior user inputs and LLM outputs, ensuring context-aware responses"("Generative AI Engineer Guide").
* D. External tools: These (e.g., APIs or calculators) enhance functionality but aren't required for a basicchat app with the specified capabilities.
* E. Chat loaders: These might refer to data loaders for chat logs, but they're not a core chain component for conversational functionality or memory.
* F. React Components: These relate to front-end UI development, not the LLM chain's backend functionality.
* Selecting the Two Required Components:
* Forknowledge retrieval, Vector Stores (B) are necessary to fetch relevant external data, a cornerstone of Databricks' RAG-based chat systems.
* Forcontextual memory, Conversation Buffer Memory (C) is required to maintain conversation history, ensuring coherent and context-aware responses.
* While an LLM itself is implied as the core generator, the question asks for chain components beyond the model, making B and C the minimal yet sufficient pair for a basic application.

Conclusion: The two required chain components areB. Vector StoresandC. Conversation Buffer Memory, as they directly address knowledge retrieval and contextual memory, respectively, aligning with Databricks' documented best practices for LLM-enabled chat applications.

NEW QUESTION: 32

A Generative AI Engineer would like an LLM to generate formatted JSON from emails. This will require parsing and extracting the following information: order ID, date, and sender email. Here's a sample email:

```
Date: April 23, 2024
Time: 4:22 PM
From: anjali.thayer@computex.org
To: cust_service@realtek.com
Subject: Shipment details

Hey there,

I have a shipment (order ID is CD34RFT) can you please send me an update?

Thank you,
Anjali
```


They will need to write a prompt that will extract the relevant information in JSON format with the highest level of output accuracy.

Which prompt will do that?

A. You will receive customer emails and need to extract date, sender email, and order ID. You should return the date, sender email, and order ID information in JSON format.

B. You will receive customer emails and need to extract date, sender email, and order ID. Return the extracted information in JSON format.

Here's an example: {"date": "April 16, 2024", "sender_email": "sarah.lee925@gmail.com", "order_id": "RE987D"}

C. You will receive customer emails and need to extract date, sender email, and order ID. Return the extracted information in a human-readable format.

D. You will receive customer emails and need to extract date, sender email, and order ID. Return the extracted information in JSON format.

Answer: B (LEAVE A REPLY)

Problem Context: The goal is to parse emails to extract certain pieces of information and output this in a structured JSON format. Clarity and specificity in the prompt design will ensure higher accuracy in the LLM' s responses.
Explanation of Options:
* Option A: Provides a general guideline but lacks an example, which helps an LLM understand the exact format expected.
* Option B: Includes a clear instruction and a specific example of the output format. Providing an example is crucial as it helps set the pattern and format in which the information should be structured, leading to more accurate results.
* Option C: Does not specify that the output should be in JSON format, thus not meeting the requirement.
* Option D: While it correctly asks for JSON format, it lacks an example that would guide the LLM on how to structure the JSON correctly.
Therefore,Option Bis optimal as it not only specifies the required format but also illustrates it with an example, enhancing the likelihood of accurate extraction and formatting by the LLM.

NEW QUESTION: 33
A Generative Al Engineer is ready to deploy an LLM application written using Foundation Model APIs. They want to follow security best practices for production scenarios Which authentication method should they choose?
A. Use an access token belonging to service principals
B. Use a frequently rotated access token belonging to either a workspace user or a service principal
C. Use OAuth machine-to-machine authentication
D. Use an access token belonging to any workspace user
Answer: (SHOW ANSWER)
The task is to deploy an LLM application using Foundation Model APIs in a production environment while adhering to security best practices. Authentication is critical for securing access to Databricks resources, such as the Foundation Model API. Let's evaluate the options based on Databricks' security guidelines for production scenarios.
Option A: Use an access token belonging to service principals
Service principals are non-human identities designed for automated workflows and applications in Databricks. Using an access token tied to a service principal ensures that the authentication is scoped to the application, follows least-privilege principles (via role-based access control), and avoids reliance on individual user credentials. This is a security best practice for production deployments.
Databricks Reference: "For production applications, use service principals with access tokens to authenticate securely, avoiding user-specific credentials" ("Databricks Security Best Practices," 2023). Additionally, the "Foundation Model API Documentation" states: "Service principal tokens are recommended for programmatic access to Foundation Model APIs." Option B: Use a frequently rotated access token belonging to either a workspace user or a service principal Frequent rotation enhances security by limiting token exposure, but tying the token to a workspace user introduces risks (e.g., user account changes, broader permissions). Including both user and service principal options dilutes the

focus on application-specific security, making this less ideal than a service-principal-only approach. It also adds operational overhead without clear benefits over Option A.
Databricks Reference: "While token rotation is a good practice, service principals are preferred over user accounts for application authentication" ("Managing Tokens in Databricks," 2023).
Option C: Use OAuth machine-to-machine authentication
OAuth M2M (e.g., client credentials flow) is a secure method for application-to-service communication, often using service principals under the hood. However, Databricks' Foundation Model API primarily supports personal access tokens (PATs) or service principal tokens over full OAuth flows for simplicity in production setups. OAuth M2M adds complexity (e.g., managing refresh tokens) without a clear advantage in this context.
Databricks Reference: "OAuth is supported in Databricks, but service principal tokens are simpler and sufficient for most API-based workloads" ("Databricks Authentication Guide," 2023).
Option D: Use an access token belonging to any workspace user
Using a user's access token ties the application to an individual's identity, violating security best practices. It risks exposure if the user leaves, changes roles, or has overly broad permissions, and it's not scalable or auditable for production.
Databricks Reference: "Avoid using personal user tokens for production applications due to security and governance concerns" ("Databricks Security Best Practices," 2023).
Conclusion: Option A is the best choice, as it uses a service principal's access token, aligning with Databricks' security best practices for production LLM applications. It ensures secure, application-specific authentication with minimal complexity, as explicitly recommended for Foundation Model API deployments.

NEW QUESTION: 34
A Generative AI Engineer needs to design an LLM pipeline to conduct multi-stage reasoning that leverages external tools. To be effective at this, the LLM will need to plan and adapt actions while performing complex reasoning tasks.
Which approach will do this?
A. Tram the LLM to generate a single, comprehensive response without interacting with any external tools, relying solely on its pre-trained knowledge.
B. Implement a framework like ReAct which allows the LLM to generate reasoning traces and perform task-specific actions that leverage external tools if necessary.
C. Encourage the LLM to make multiple API calls in sequence without planning or structuring the calls, allowing the LLM to decide when and how to use external tools spontaneously.
D. Use a Chain-of-Thought (CoT) prompting technique to guide the LLM through a series of reasoning steps, then manually input the results from external tools for the final answer.
Answer: (SHOW ANSWER)
The task requires an LLM pipeline for multi-stage reasoning with external tools, necessitating planning, adaptability, and complex reasoning. Let's evaluate the options based on Databricks' recommendations for advanced LLM workflows.

Option A: Train the LLM to generate a single, comprehensive response without interacting with any external tools, relying solely on its pre-trained knowledge This approach limits the LLM to its static knowledge base, excluding external tools and multi-stage reasoning. It can't adapt or plan actions dynamically, failing the requirements.
Databricks Reference: "External tools enhance LLM capabilities beyond pre-trained knowledge" ("Building LLM Applications with Databricks," 2023).
Option B: Implement a framework like ReAct which allows the LLM to generate reasoning traces and perform task-specific actions that leverage external tools if necessary ReAct (Reasoning + Acting) combines reasoning traces (step-by-step logic) with actions (e.g., tool calls), enabling the LLM to plan, adapt, and execute complex tasks iteratively. This meets all requirements: multi-stage reasoning, tool use, and adaptability.
Databricks Reference: "Frameworks like ReAct enable LLMs to interleave reasoning and external tool interactions for complex problem-solving" ("Generative AI Cookbook," 2023).
Option C: Encourage the LLM to make multiple API calls in sequence without planning or structuring the calls, allowing the LLM to decide when and how to use external tools spontaneously Unstructured, spontaneous API calls lack planning and may lead to inefficient or incorrect tool usage. This doesn't ensure effective multi-stage reasoning or adaptability.
Databricks Reference: Structured frameworks are preferred: "Ad-hoc tool calls can reduce reliability in complex tasks" ("Building LLM-Powered Applications").
Option D: Use a Chain-of-Thought (CoT) prompting technique to guide the LLM through a series of reasoning steps, then manually input the results from external tools for the final answer CoT improves reasoning but relies on manual tool interaction, breaking automation and adaptability. It's not a scalable pipeline solution.
Databricks Reference: "Manual intervention is impractical for production LLM pipelines" ("Databricks Generative AI Engineer Guide").
Conclusion: Option B (ReAct) is the best approach, as it integrates reasoning and tool use in a structured, adaptive framework, aligning with Databricks' guidance for complex LLM workflows.

NEW QUESTION: 35
A Generative AI Engineer interfaces with an LLM with prompt/response behavior that has been trained on customer calls inquiring about product availability. The LLM is designed to output "In Stock" if the product is available or only the term "Out of Stock" if not.
Which prompt will work to allow the engineer to respond to call classification labels correctly?
A. Respond with "In Stock" if the customer asks for a product.
B. You will be given a customer call transcript where the customer asks about product availability. The outputs are either "In Stock" or "Out of Stock". Format the output in JSON, for example: {"call_id": "123", "label": "In Stock"}.
C. Respond with "Out of Stock" if the customer asks for a product.
D. You will be given a customer call transcript where the customer inquires about product availability.Respond with "In Stock" if the product is available or "Out of Stock" if not.
Answer: (SHOW ANSWER)

* Problem Context: The Generative AI Engineer needs a prompt that will enable an LLM trained on customer call transcripts to classify and respond correctly regarding product availability. The desired response should clearly indicate whether a product is "In Stock" or "Out of Stock," and it should be formatted in a way that is structured and easy to parse programmatically, such as JSON.
* Explanation of Options:
* Option A: Respond with "In Stock" if the customer asks for a product. This prompt is too generic and does not specify how to handle the case when a product is not available, nor does it provide a structured output format.
* Option B: This option is correctly formatted and explicit. It instructs the LLM to respond based on the availability mentioned in the customer call transcript and to format the response in JSON.
This structure allows for easy integration into systems that may need to process this information automatically, such as customer service dashboards or databases.
* Option C: Respond with "Out of Stock" if the customer asks for a product. Like option A, this prompt is also insufficient as it only covers the scenario where a product is unavailable and does not provide a structured output.
* Option D: While this prompt correctly specifies how to respond based on product availability, it lacks the structured output format, making it less suitable for systems that require formatted data for further processing.
Given the requirements for clear, programmatically usable outputs,Option Bis the optimal choice because it provides precise instructions on how to respond and includes a JSON format example for structuring the output, which is ideal for automated systems or further data handling.

NEW QUESTION: 36
A Generative Al Engineer is building a RAG application that answers questions about internal documents for the company SnoPen AI.
The source documents may contain a significant amount of irrelevant content, such as advertisements, sports news, or entertainment news, or content about other companies.
Which approach is advisable when building a RAG application to achieve this goal of filtering irrelevant information?
A. Keep all articles because the RAG application needs to understand non-company content to avoid answering questions about them.
B. Include in the system prompt that any information it sees will be about SnoPenAI, even if no data filtering is performed.
C. Include in the system prompt that the application is not supposed to answer any questions unrelated to SnoPen Al.
D. Consolidate all SnoPen AI related documents into a single chunk in the vector database.
Answer: C (LEAVE A REPLY)
In a Retrieval-Augmented Generation (RAG) application built to answer questions about internal documents, especially when the dataset contains irrelevant content, it's crucial to guide the system to focus on the right information. The best way to achieve this is byincluding a clear instruction in the system prompt(option C).

* System Prompt as Guidance:The system prompt is an effective way to instruct the LLM to limit its focus to SnoPen AI-related content. By clearly specifying that the model should avoid answering questions unrelated to SnoPen AI, you add an additional layer of control that helps the model stay on- topic, even if irrelevant content is present in the dataset.
* Why This Approach Works:The prompt acts as a guiding principle for the model, narrowing its focus to specific domains. This prevents the model from generating answers based on irrelevant content, such as advertisements or news unrelated to SnoPen AI.
* Why Other Options Are Less Suitable:
* A (Keep All Articles): Retaining all content, including irrelevant materials, without any filtering makes the system prone to generating answers based on unwanted data.
* B (Include in the System Prompt about SnoPen AI): This option doesn't address irrelevant content directly, and without filtering, the model might still retrieve and use irrelevant data.
* D (Consolidating Documents into a Single Chunk): Grouping documents into a single chunk makes the retrieval process less efficient and won't help filter out irrelevant content effectively.
Therefore, instructing the system in the prompt not to answer questions unrelated to SnoPen AI (option C) is the best approach to ensure the system filters out irrelevant information.

NEW QUESTION: 37
A Generative Al Engineer is responsible for developing a chatbot to enable their company's internal HelpDesk Call Center team to more quickly find related tickets and provide resolution. While creating the GenAI application work breakdown tasks for this project, they realize they need to start planning which data sources (either Unity Catalog volume or Delta table) they could choose for this application. They have collected several candidate data sources for consideration:
call_rep_history: a Delta table with primary keys representative_id, call_id. This table is maintained to calculate representatives' call resolution from fields call_duration and call start_time.
transcript Volume: a Unity Catalog Volume of all recordings as a *.wav files, but also a text transcript as *.txt files.
call_cust_history: a Delta table with primary keys customer_id, cal1_id. This table is maintained to calculate how much internal customers use the HelpDesk to make sure that the charge back model is consistent with actual service use.
call_detail: a Delta table that includes a snapshot of all call details updated hourly. It includes root_cause and resolution fields, but those fields may be empty for calls that are still active.
maintenance_schedule - a Delta table that includes a listing of both HelpDesk application outages as well as planned upcoming maintenance downtimes.
They need sources that could add context to best identify ticket root cause and resolution.
Which TWO sources do that? (Choose two.)
A. call_cust_history
B. maintenance_schedule
C. call_rep_history
D. call_detail
E. transcript Volume

Answer: D,E (LEAVE A REPLY)

In the context of developing a chatbot for a company's internal HelpDesk Call Center, the key is to select data sources that provide the most contextual and detailed information about the issues being addressed. This includes identifying the root cause and suggesting resolutions. The two most appropriate sources from the list are:

* Call Detail (Option D):
* Contents: This Delta table includes a snapshot of all call details updated hourly, featuring essential fields like root_cause and resolution.
* Relevance: The inclusion of root_cause and resolution fields makes this source particularly valuable, as it directly contains the information necessary to understand and resolve the issues discussed in the calls. Even if some records are incomplete, the data provided is crucial for a chatbot aimed at speeding up resolution identification.
* Transcript Volume (Option E):
* Contents: This Unity Catalog Volume contains recordings in .wav format and text transcripts in .txt files.
* Relevance: The text transcripts of call recordings can provide in-depth context that the chatbot can analyze to understand the nuances of each issue. The chatbot can use natural language processing techniques to extract themes, identify problems, and suggest resolutions based on previous similar interactions documented in the transcripts.

Why Other Options Are Less Suitable:

* A (Call Cust History): While it provides insights into customer interactions with the HelpDesk, it focuses more on the usage metrics rather than the content of the calls or the issues discussed.
* B (Maintenance Schedule): This data is useful for understanding when services may not be available but does not contribute directly to resolving user issues or identifying root causes.
* C (Call Rep History): Though it offers data on call durations and start times, which could help in assessing performance, it lacks direct information on the issues being resolved.

Therefore, Call Detail and Transcript Volume are the most relevant data sources for a chatbot designed to assist with identifying and resolving issues in a HelpDesk Call Center setting, as they provide direct and contextual information related to customer issues.

NEW QUESTION: 38

A Generative AI Engineer has created a RAG application to look up answers to questions about a series of fantasy novels that are being asked on the author's web forum. The fantasy novel texts are chunked and embedded into a vector store with metadata (page number, chapter number, book title), retrieved with the user's query, and provided to an LLM for response generation. The Generative AI Engineer used their intuition to pick the chunking strategy and associated configurations but now wants to more methodically choose the best values.

Which TWO strategies should the Generative AI Engineer take to optimize their chunking strategy and parameters? (Choose two.)

A. Change embedding models and compare performance.

B. Add a classifier for user queries that predicts which book will best contain the answer. Use this to filter retrieval.

C. Pass known questions and best answers to an LLM and instruct the LLM to provide the best token count. Use a summary statistic (mean, median, etc.) of the best token counts to choose chunk size.
D. Create an LLM-as-a-judge metric to evaluate how well previous questions are answered by the most appropriate chunk. Optimize the chunking parameters based upon the values of the metric.
E. Choose an appropriate evaluation metric (such as recall or NDCG) and experiment with changes in the chunking strategy, such as splitting chunks by paragraphs or chapters.
Choose the strategy that gives the best performance metric.
Answer: (SHOW ANSWER)

NEW QUESTION: 39
A Generative AI Engineer has been reviewing issues with their company's LLM-based question-answering assistant and has determined that a technique called prompt chaining could help alleviate some performance concerns. However, to suggest this to their team, they have to clearly explain how it works and how it can benefit their question-answering assistant. Which explanation do they communicate to the team?
A. It allows you to break down complex tasks into multiple independent subtasks. This enables the assistant to generate more comprehensive and accurate responses.
B. It allows you to reduce the latency of your applications. By having multiple chains participating in the response as a chain, you increase the rate at which the response is generated.
C. It allows you to decrease the effort involved in crafting a prompt. Chains make it possible to reuse prompt text across multiple different use cases.
D. It reduces the average cost of a typical request. Chains make more efficient use of the tokens produced to generate higher quality responses with fewer tokens.
Answer: A (LEAVE A REPLY)
Prompt chaining is a fundamental design pattern in LLM application development used to handle complexity. Instead of sending a single, massive, and highly complex prompt to an LLM-which often results in reasoning errors or hallucinations-chaining breaks the logic into a sequence of smaller, targeted steps. For example, a legal assistant might first chain a step to "identify the legal jurisdiction," followed by a step to "extract relevant statutes," and finally a step to "summarize the findings." This modularity improves reliability because each prompt has a narrower focus, making it easier for the model to follow instructions accurately. While it may actually increase latency (contradicting B) and cost (contradicting D) due to multiple API calls, the primary engineering benefit is the significant boost in the quality and robustness of the output. It also allows for intermediate validation and error handling between steps, which is impossible in a single-call architecture.

NEW QUESTION: 40
A Generative AI Engineer is testing a simple prompt template in LangChain using the code below, but is getting an error.

```
from langchain.chains import LLMChain
from langchain_community.llms import OpenAI
from langchain_core.prompts import PromptTemplate

prompt_template = "Tell me a {adjective} joke"

prompt = PromptTemplate(
    input_variables=["adjective"],
    template=prompt_template
)

llm = LLMChain(prompt=prompt)
llm.generate([{"adjective": "funny"}])
```

Assuming the API key was properly defined, what change does the Generative AI Engineer need to make to fix their chain?

A.
```
prompt_template = "Tell me a {adjective} joke"

prompt = PromptTemplate(
    input_variables=["adjective"],
    template=prompt_template
)

llm = LLMChain(prompt=prompt)
llm.generate("funny")
```

B.
```
prompt_template = "Tell me a {adjective} joke"

prompt = PromptTemplate(
    input_variables=["adjective"],
    template=prompt_template
)

llm = LLMChain(prompt=prompt.format("funny"))
llm.generate()
```

C.
```
prompt_template = "Tell me a {adjective} joke"

prompt = PromptTemplate(
    input_variables=["adjective"],
    template=prompt_template
    llm=OpenAI()
)

llm = LLMChain(prompt=prompt)
llm.generate([{"adjective": "funny"}])
```

D.

Answer: C (LEAVE A REPLY)

To fix the error in the LangChain code provided for using a simple prompt template, the correct approach is Option C. Here's a detailed breakdown of why Option C is the right choice and how it addresses the issue:

* Proper Initialization: In Option C, the LLMChain is correctly initialized with the LLM instance specified as OpenAI(), which likely represents a language model (like GPT) from OpenAI. This is crucial as it specifies which model to use for generating responses.
* Correct Use of Classes and Methods:
* The PromptTemplate is defined with the correct format, specifying that adjective is a variable within the template. This allows dynamic insertion of values into the template when generating text.
* The prompt variable is properly linked with the PromptTemplate, and the final template string is passed correctly.
* The LLMChain correctly references the prompt and the initialized OpenAI() instance, ensuring that the template and the model are properly linked for generating output.

Why Other Options Are Incorrect:

* Option A: Misuses the parameter passing in generate method by incorrectly structuring the dictionary.
* Option B: Incorrectly uses prompt.format method which does not exist in the context of LLMChain and PromptTemplate configuration, resulting in potential errors.
* Option D: Incorrect order and setup in the initialization parameters for LLMChain, which would likely lead to a failure in recognizing the correct configuration for prompt and LLM usage.

Thus, Option C is correct because it ensures that the LangChain components are correctly set up and integrated, adhering to proper syntax and logical flow required by LangChain's architecture. This setup avoids common pitfalls such as type errors or method misuses, which are evident in other options.