Product Questions: 61
Version: 5.0

Question: 1

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

Explanation:
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.

Question: 2

A Generative AI Engineer is designing a RAG application for answering user questions on technical regulations as they learn a new sport.
What are the steps needed to build this RAG application and deploy it?

A. Ingest documents from a source –> Index the documents and saves to Vector Search –> User submits queries against an LLM –> LLM retrieves relevant documents –> Evaluate model –> LLM generates a response –> Deploy it using Model Serving
B. Ingest documents from a source –> Index the documents and save to Vector Search –> User submits queries against an LLM –> LLM retrieves relevant documents –> LLM generates a response -> Evaluate model –> Deploy it using Model Serving
C. Ingest documents from a source –> Index the documents and save to Vector Search –> Evaluate model –> Deploy it using Model Serving
D. User submits queries against an LLM –> Ingest documents from a source –> Index the documents and save to Vector Search –> LLM retrieves relevant documents –> LLM generates a response –> Evaluate model –> Deploy it using Model Serving

Answer: B

Explanation:
The Generative AI Engineer needs to follow a methodical pipeline to build and deploy a Retrieval-Augmented Generation (RAG) application. The steps outlined in option B accurately reflect this process:
Ingest documents from a source: This is the first step, where the engineer collects documents (e.g., technical regulations) that will be used for retrieval when the application answers user questions.
Index the documents and save to Vector Search: Once the documents are ingested, they need to be

embedded using a technique like embeddings (e.g., with a pre-trained model like BERT) and stored in a vector database (such as Pinecone or FAISS). This enables fast retrieval based on user queries.
User submits queries against an LLM: Users interact with the application by submitting their queries. These queries will be passed to the LLM.
LLM retrieves relevant documents: The LLM works with the vector store to retrieve the most relevant documents based on their vector representations.
LLM generates a response: Using the retrieved documents, the LLM generates a response that is tailored to the user's question.
Evaluate model: After generating responses, the system must be evaluated to ensure the retrieved documents are relevant and the generated response is accurate. Metrics such as accuracy, relevance, and user satisfaction can be used for evaluation.
Deploy it using Model Serving: Once the RAG pipeline is ready and evaluated, it is deployed using a model-serving platform such as Databricks Model Serving. This enables real-time inference and response generation for users.
By following these steps, the Generative AI Engineer ensures that the RAG application is both efficient and effective for the task of answering technical regulation questions.

Question: 3

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

Explanation:
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 why A is 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.

Question: 4

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. The match 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 Al 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

Explanation:
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 the employee profiles and project 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 A suggests 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 B involves 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 C suggests 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 D is 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 leveraging vector embeddings and similarity search techniques, both of which are fundamental tools in Generative AI engineering for 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 like FAISS or Milvus allow 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 in Generative AI, such as vector embeddings and semantic search.

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: C

Explanation:
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.

Question: 6

A Generative AI Engineer has a provisioned throughput model serving endpoint as part of a RAG application and would like to monitor the serving endpoint's incoming requests and outgoing responses. The current approach is to include a micro-service in between the endpoint and the user interface to write logs to a remote server.
Which Databricks feature should they use instead which will perform the same task?

A. Vector Search
B. Lakeview
C. DBSQL
D. Inference Tables

Answer: D

Explanation:
Problem Context: The goal is to monitor the serving endpoint for incoming requests and outgoing responses in a provisioned throughput model serving endpoint within a Retrieval-Augmented Generation (RAG) application. The current approach involves using a microservice to log requests and responses to a remote server, but the Generative AI Engineer is looking for a more streamlined solution within Databricks.
Explanation of Options:
Option A: Vector Search: This feature is used to perform similarity searches within vector databases. It doesn't provide functionality for logging or monitoring requests and responses in a serving endpoint, so it's not applicable here.
Option B: Lakeview: Lakeview is not a feature relevant to monitoring or logging request-response cycles for serving endpoints. It might be more related to viewing data in Databricks Lakehouse but doesn't fulfill the specific monitoring requirement.
Option C: DBSQL: Databricks SQL (DBSQL) is used for running SQL queries on data stored in

Databricks, primarily for analytics purposes. It doesn't provide the direct functionality needed to monitor requests and responses in real-time for an inference endpoint.
Option D: Inference Tables: This is the correct answer. Inference Tables in Databricks are designed to store the results and metadata of inference runs. This allows the system to log incoming requests and outgoing responses directly within Databricks, making it an ideal choice for monitoring the behavior of a provisioned serving endpoint. Inference Tables can be queried and analyzed, enabling easier monitoring and debugging compared to a custom microservice.
Thus, Inference Tables are the optimal feature for monitoring request and response logs within the Databricks infrastructure for a model serving endpoint.

Question: 7

A Generative Al Engineer is tasked with improving the RAG quality by addressing its inflammatory outputs.
Which action would be most effective in mitigating the problem of offensive text outputs?

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

Answer: D

Explanation:
Addressing offensive or inflammatory outputs in a Retrieval-Augmented Generation (RAG) system is critical for improving user experience and ensuring ethical AI deployment. Here's why D is the most effective approach:
Manual data curation: The root cause of offensive outputs often comes from the underlying data used to train the model or populate the retrieval system. By manually curating the upstream data and conducting thorough reviews before the data is fed into the RAG system, the engineer can filter out harmful, offensive, or inappropriate content.
Improving data quality: Curating data ensures the system retrieves and generates responses from a high-quality, well-vetted dataset. This directly impacts the relevance and appropriateness of the outputs from the RAG system, preventing inflammatory content from being included in responses.
Effectiveness: This strategy directly tackles the problem at its source (the data) rather than just mitigating the consequences (such as informing users or restricting access). It ensures that the system consistently provides non-offensive, relevant information.
Other options, such as increasing the frequency of data updates or informing users about behavior expectations, may not directly mitigate the generation of inflammatory outputs.

Question: 8

A Generative Al Engineer is creating an LLM-based application. The documents for its retriever have been chunked to a maximum of 512 tokens each. The Generative Al 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

Explanation:
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.

Question: 9

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

Explanation:
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 why B is 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.

Question: 10

A Generative AI 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: DE

Explanation:
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):