**Title: Transforming Tender Management in GE Healthcare with RAG Gen AI Solutions**

**Background**  
The tender management process at GE Healthcare involves reviewing, analyzing, and responding to a high volume of complex PDF documents detailing requirements, specifications, and compliance standards for healthcare equipment. Given the scale of our operations, these documents cover a broad range of products, compliance details, and customized terms based on region, regulatory standards, and client needs.

**Challenges in Tender Management**

1. **Manual Review and Analysis**
   * *Challenge*: Each tender document requires careful reading to identify eligibility criteria, product specifications, and other requirements. Reviewing these lengthy documents is time-consuming and prone to human oversight.
   * *Impact*: Teams often spend hours or even days analyzing individual documents, limiting the time available to respond strategically or bid on additional tenders.
2. **Complex Compliance and Requirement Matching**
   * *Challenge*: Verifying that our healthcare instruments meet specific requirements and compliance standards involves cross-referencing numerous product specs, compliance certifications, and technical documents.
   * *Impact*: This process is tedious and, if not executed accurately, could result in costly errors or disqualification from tenders due to non-compliance.
3. **Inconsistent Knowledge Retrieval**
   * *Challenge*: Finding relevant information across previous tenders, product data sheets, and historical bid responses can be inefficient. Manually locating this information often leads to inconsistencies in responses.
   * *Impact*: Delays in locating pertinent information can compromise the quality and competitiveness of our bids.
4. **Competitor Insights and Strategic Positioning**
   * *Challenge*: We lack quick and effective access to structured competitor insights from previous tenders, making it difficult to adapt our offerings and highlight our unique value.
   * *Impact*: Without clear data on competitor bids, it's harder to refine our strategy and deliver differentiated, competitive proposals.
5. **Workflow and Timeline Management**
   * *Challenge*: Coordinating multiple team members for tasks and ensuring adherence to critical milestones is complex without automated reminders and tracking.
   * *Impact*: Missed steps or delayed actions can jeopardize submission timelines and reduce operational efficiency.

**Solution: RAG-Powered Gen AI for Tender Management**

The introduction of a **RAG (Retrieval-Augmented Generation) Gen AI solution** would significantly alleviate these challenges by enhancing data access, ensuring consistency, and reducing manual effort.

**How RAG Gen AI Addresses These Challenges**

1. **Automated Document Summarization and Insights Extraction**
   * *Solution*: RAG Gen AI can parse, summarize, and highlight key points from tender documents instantly, including eligibility requirements, product specifications, and deadlines.
   * *Value*: This automation reduces document review time by up to 70%, enabling teams to make informed decisions faster and freeing up time to bid on more opportunities.
2. **Compliance and Requirement Validation**
   * *Solution*: Gen AI can cross-reference tender requirements with our internal compliance records, certifications, and product specs to ensure all bid submissions meet necessary standards.
   * *Value*: Ensures a higher accuracy rate for compliance, reducing the risk of disqualification and enhancing our reputation for reliability and attention to detail.
3. **Intelligent Search and Retrieval Across Documents**
   * *Solution*: By indexing and enabling smart search functions, Gen AI allows teams to retrieve information on previous bids, specifications, and pricing with pinpoint accuracy.
   * *Value*: Speeds up information retrieval, resulting in consistent and comprehensive bid responses that draw on past experience and insights, adding strategic value to each submission.
4. **Competitive Analysis and Market Insights**
   * *Solution*: Gen AI can mine competitor insights from public tender documents, analyzing pricing trends, product positioning, and market tactics.
   * *Value*: Empowers the team to create competitive proposals, identify gaps, and emphasize GE’s unique selling points, making bids more compelling and strategically aligned.
5. **Automated Workflow Management**
   * *Solution*: Gen AI can extract milestones and key deadlines, generating automated task reminders and workflows that align with each tender’s timeline.
   * *Value*: Improves coordination and adherence to deadlines, reducing risk of missed steps and enhancing the team’s efficiency in managing multiple tenders simultaneously.

**Value and Impact of RAG Gen AI Solution for GE Healthcare**

With the adoption of a RAG Gen AI solution, GE Healthcare can transform tender management in a way that:

* **Reduces Manual Workload**: Cuts down time spent on repetitive tasks like document review and requirement matching by up to 70%.
* **Enhances Accuracy**: Minimizes errors in compliance and requirement matching, protecting GE’s reputation and avoiding disqualification.
* **Increases Bid Competitiveness**: Equips teams with competitor insights, helping them craft data-backed, strategic proposals.
* **Improves Collaboration**: Supports better coordination and adherence to timelines, ensuring timely and cohesive bid submissions.
* **Enables Data-Driven Decision-Making**: RAG allows access to all relevant information, enabling decisions based on real-time insights and historical data.

**Conclusion**  
By implementing RAG Gen AI, GE Healthcare’s tender management can evolve from a labor-intensive process to a streamlined, data-driven operation. This shift will enable the team to focus on strategic objectives, enhance competitive positioning, and secure more tender wins.

Steps –

1. Request model access from Amazon Bedrock

Amazon Bedrock is a fully managed service that offers a choice of high-performing foundation models (FMs) from leading AI companies like Anthropic, Meta, Mistral AI, and Amazon through a single API.

2. setting up your Databricks on AWS.

[

1. Login to your Databricks workspace
2. Set up a compute cluster for your Databricks notebook
3. Import notebooks into your Databricks workspace
4. Configure access to Databricks API for your notebooks
5. Configure access to AWS APIs for your notebooks
6. Configure other parameters for your notebooks

]

3. Ingest and prepare data

* [Load PDFs into our raw table in binary format using [Auto Loader](https://docs.databricks.com/en/ingestion/auto-loader/index.html)
* Parse text content from PDFs using [unstructured](https://github.com/Unstructured-IO/unstructured) open-source library within a Databricks [User-Defined Function (UDF)](https://docs.databricks.com/en/udf/index.html)
* Split text into chunks using [SentenceSplitter](https://docs.llamaindex.ai/en/stable/api/llama_index.core.node_parser.SentenceSplitter.html" \t "_blank) from [LlamaIndex](https://www.llamaindex.ai/" \t "_blank)
* Process data into vectors by using an embeddings model
* Store text chunks + embeddings in a [Delta Lake](https://docs.databricks.com/en/delta/index.html) table
* Prepare for retrieval by semantic similarity search using [Databricks Vector Search](https://docs.databricks.com/en/generative-ai/vector-search.html)

]

4. Prepare your chat model

[ Next, we will leverage [LangChain](https://python.langchain.com/docs/get_started/introduction" \t "_blank), a framework for developing applications powered by foundation models, to build out the chatbot.

Here's what we will do in the Notebook:

* Access Databricks native foundation models through [Foundation Model APIs](https://docs.databricks.com/en/machine-learning/foundation-models/index.html) and Amazon Bedrock models through [Databricks External Model Serving](https://docs.databricks.com/en/generative-ai/external-models/index.html)
* Add conversation history so that our Chatbot takes into account past messages when responding
* Add filters so that our Chatbot only answers questions about Databricks
* Integrate semantic similarity search so that our Chatbot can answer questions using the Databricks documentation we stored in Lab 1
* Store the model into the Databricks Unity Catalog
* Compare the final chatbot output when using Claude 3 Sonnet and DBRX Instruct

]

5.Deploy your chat model

Now that you have prepared your chat model, the next step is to deploy and test the model.

You will learn how to:

* Set up a [model serving endpoint](https://docs.databricks.com/en/machine-learning/model-serving/create-manage-serving-endpoints.html) to serve your own custom models
* Manage the lifecycle of your model using [MLFlow on Unity Catalog](https://docs.databricks.com/en/machine-learning/manage-model-lifecycle/index.html" \t "_blank)
* Configure built-in monitoring using [inference tables](https://docs.databricks.com/en/machine-learning/model-serving/inference-tables.html) for later evaluation
* Test your model using a Streamlit chat application on AWS

1. Ingest and extract information from unstructured documents
2. Setup vector search using a self-managed vector search index in a Delta Table
3. Build a chatbot application with advanced capabilities such as conversation history and filter for a specific topic
4. Deploy a model endpoint with inference tables for monitoring and debugging
5. Monitor and evaluate your generative AI application

### **Process Flow for Implementing a Retrieval-Augmented Generation (RAG) Chatbot Using Databricks and AWS**

#### **Lab 1: Data Ingestion and Preparation for RAG**

1. **Data Ingestion**
   * **Load PDFs**: Use Databricks’ Auto Loader to load PDFs into a raw table in binary format.
2. **Text Parsing and Preprocessing**
   * **Parse Text**: Apply the Unstructured open-source library within a Databricks User-Defined Function (UDF) to parse text content from the PDFs.
   * **Split Text**: Use SentenceSplitter from LlamaIndex to divide parsed text into manageable chunks that fit the input size of the embedding model.
3. **Embedding Generation**
   * **Generate Embeddings**: Process the text chunks to create embedding vectors using a chosen embeddings model.
4. **Data Storage in Delta Lake**
   * **Store Data**: Store text chunks, embeddings, and metadata in a Databricks Delta Table to ensure scalability and efficient retrieval.
5. **Index Creation for Semantic Search**
   * **Create Vector Search Index**: Set up a Vector Search Index on Databricks to enable similarity-based retrieval for incoming queries.

#### **Lab 2: Chat Model Preparation**

1. **LLM Integration for Reasoning**
   * **Access Models**: Explore Databricks foundation models via Foundation Model APIs and integrate with Amazon Bedrock models through Databricks External Model Serving.
2. **Chatbot Enhancements**
   * **Add Conversation History**: Implement context preservation to maintain past chat interactions.
   * **Filtering Capabilities**: Add filters to restrict the chatbot’s scope to questions about Databricks, ensuring relevance to domain-specific data.
3. **Semantic Search Integration**
   * **Retrieve Relevant Documents**: Link semantic similarity search to retrieve specific document information stored in Lab 1’s Delta Lake table.
4. **Model Storage in Unity Catalog**
   * **Model Management**: Store the chatbot model in Databricks Unity Catalog for version control and lifecycle management.
5. **Model Comparison**
   * **Evaluate Outputs**: Compare chatbot outputs between **Claude 3 Sonnet and DBRX Instruct to select the most effective model.**

#### **Lab 3: Model Deployment and Testing**

1. **Model Deployment on AWS**
   * **Set Up Model Serving Endpoint**: Deploy the model using a custom model serving endpoint for real-time inference.
2. **Model Lifecycle Management**
   * **MLFlow on Unity Catalog**: Use MLFlow on Unity Catalog to manage the model's lifecycle, tracking versions and deployment history.
3. **Monitoring and Evaluation**
   * **Built-in Monitoring**: Set up inference tables to track and evaluate model performance over time.
4. **Testing with Streamlit**
   * **Streamlit User Interface**: Deploy a Streamlit application on AWS for users to interact with the chatbot.
   * **Integrate Model**: Connect the Streamlit app to the Databricks model serving endpoint for real-time responses.

#### **AWS Chatbot Application Testing**

1. **Application Access and Configuration**
   * **Access Chatbot Application**: Use InstanceAccessURL from the AWS Event Dashboard to access the Streamlit chatbot.
   * **Configuration**: Enter the URL of the Databricks model serving endpoint and provide the Databricks Personal Access Token (PAT) to authenticate.
2. **Model Challenge**
   * **User Testing**: Engage the chatbot to retrieve and answer questions based on the uploaded documents, validating the RAG functionality and model effectiveness.

### **Lab 1: Data Ingestion and Preparation for RAG**

1. **Data Ingestion**
   * **Service**: **Databricks Auto Loader** is used to load PDF documents into a raw table in binary format.
   * **Purpose**: Efficiently loads raw data for further processing and indexing.
2. **Text Parsing and Preprocessing**
   * **Service**: **Databricks User-Defined Function (UDF)** in conjunction with **Unstructured open-source library**.
   * **Purpose**: Parses text content from the PDFs, preparing it for chunking and embedding.
3. **Chunking Text**
   * **Service**: **LlamaIndex SentenceSplitter**
   * **Purpose**: Splits parsed text into smaller chunks that fit the maximum input size of the embedding model, ensuring data is easily searchable.
4. **Embedding Generation**
   * **Service**: Embeddings Model on **Databricks**
   * **Purpose**: Converts text chunks into vector representations that can be used for semantic similarity searches.
5. **Data Storage in Delta Lake**
   * **Service**: **Delta Lake on Databricks**
   * **Purpose**: Stores text chunks, embeddings, and metadata in a structured format for fast retrieval.
6. **Index Creation for Semantic Search**
   * **Service**: **Databricks Vector Search**
   * **Purpose**: Allows fast similarity-based searches of embedded data, essential for RAG to retrieve relevant information for queries.

### **Lab 2: Chat Model Preparation**

1. **Model Integration for Reasoning**
   * **Service**: **Databricks Foundation Model APIs** and **Amazon Bedrock**
   * **Purpose**: Provides access to a range of large language models (LLMs) for generating responses based on user queries.
2. **Adding Conversation History**
   * **Service**: **LangChain Framework**
   * **Purpose**: Retains chat history, enabling the chatbot to consider past interactions in generating responses, which improves coherence and context.
3. **Filtering Capabilities**
   * **Service**: **Databricks UDFs and Filters**
   * **Purpose**: Restricts the chatbot to respond only to queries related to Databricks data, enhancing response relevance.
4. **Semantic Search Integration**
   * **Service**: **Databricks Vector Search**
   * **Purpose**: Searches the Delta Lake index for relevant documents to enhance the response accuracy and relevance.
5. **Model Storage in Unity Catalog**
   * **Service**: **Databricks Unity Catalog**
   * **Purpose**: Manages and versions the chatbot model, providing lifecycle control and ensuring version consistency.
6. **Model Comparison**
   * **Service**: Comparison between **Claude 3 Sonnet** and **DBRX Instruct**
   * **Purpose**: Evaluates the effectiveness of different models to select the best one for the chatbot.

### **Lab 3: Model Deployment and Testing**

1. **Model Serving Endpoint Setup**
   * **Service**: **AWS Custom Model Serving Endpoint**
   * **Purpose**: Deploys the model for real-time inference, making it accessible for user interactions via the Streamlit application.
2. **Model Lifecycle Management**
   * **Service**: **MLFlow on Unity Catalog**
   * **Purpose**: Tracks versions, deployment history, and other lifecycle aspects of the model, supporting updates and changes.
3. **Built-in Monitoring**
   * **Service**: **Databricks Inference Tables**
   * **Purpose**: Collects model inference data for ongoing monitoring and evaluation, allowing for improvements and debugging.
4. **User Interface with Streamlit**
   * **Service**: **Streamlit on AWS**
   * **Purpose**: Provides a simple web-based UI for users to interact with the chatbot. Streamlit integrates with the model serving endpoint to retrieve responses based on user input.
5. **Testing and Access**
   * **Service**: **AWS InstanceAccessURL** and **Personal Access Token (PAT)**
   * **Purpose**: Authenticates access to the model and chatbot application, enabling secure testing of the chatbot’s performance and relevance.

Speaker notes 🡪

**Introduction: Industry and Topic**

* **Industry**: Healthcare
* **Topic**: Tender Management for Healthcare Instruments  
  "I chose the healthcare industry because tender management is critical for a company like GE Healthcare, which deals with a high volume of complex bids for healthcare equipment. Navigating through extensive tender documents in PDF format, ensuring compliance, and drafting timely responses are essential for successfully securing these tenders."

**Why I Chose This Topic**

* "This challenge directly impacts productivity, bid accuracy, and competitiveness. By addressing inefficiencies in tender management, we can save significant time, reduce human error, and ultimately improve our chances of winning contracts. Gen AI, specifically Retrieval-Augmented Generation (RAG), presents a unique solution to make this process far more efficient and strategic."

**Dataset and Model Details**

* **Dataset**: PDF-based tender documents with requirements, compliance criteria, and specifications for various healthcare equipment.
* **Model**: A RAG Gen AI approach is used here, where the model combines retrieval capabilities with generative language understanding. "To tackle this, I’m using a setup built with AWS services and Databricks, which allows us to index and retrieve document information efficiently and combine it with Gen AI's language generation capabilities to parse, extract, and respond to tender document queries."

**Technical Solution Details and Techniques**

1. **RAG Approach**
   * "Using Retrieval-Augmented Generation, the solution first retrieves relevant sections from tender documents based on specific queries. Then, the generative AI interprets and summarizes this content, allowing the system to generate structured responses or summaries of key information, such as requirements and deadlines, from a large volume of data."
2. **Technical Workflow**
   * "The solution integrates AWS services with Databricks, leveraging indexing and search functionality on Databricks, and uses AWS's compute power to facilitate fast retrieval and generation processes."
   * "I've also implemented a Streamlit application as the user interface. This application allows the end-user to interact easily with the AI, query specific details, or get summaries, making the complex data accessible and user-friendly."

**Demo of Functioning Project**

* **Footage**: Demonstrate the application on Streamlit, showcasing:
  + Querying a sample tender document and retrieving the required information instantly.
  + Highlighting how users can obtain summaries of compliance requirements or competitor analysis based on past data, and showcasing the system’s capability to manage the workflow from document ingestion to response generation.

**Closing**

* "This RAG-based Gen AI solution is a game-changer for GE Healthcare's tender management process. By improving speed, accuracy, and strategic insight, we can transform the way our team approaches tenders, ultimately making us more competitive and effective in the market."