LUMENDATA

Data Sheet

From Static Datasheets to Interactive Insights:

A Databricks-Powered Chatbot

5201 GREAT AMERICAN PARKWAY, SUITE 320
SANTA CLARA, CA 95054
Tel: (855) 695-8636
E-mail: info@lumendata.com
Website: www.lumendata.com

LumenData Introduces Databricks-Powered Chatbot Solution to Enable Interactive Insights

How your information is presented can significantly impact decision-making and client engagement. Traditional static datasheets, while informative, often fall short in delivering the dynamic experience that modern clients expect. Static datasheets make it hard for clients to find information quickly. They lack interactivity, which is crucial for engaging onboarding. How do we solve this?

Enter LumenData's Databricks-Powered Chatbot Solution!

The chatbot simplifies the onboarding process by guiding clients through each step and ensures they have a smooth and intuitive experience from the very beginning. It turns confusing data reports into clear, interactive insights. The solution allows clients to easily understand complex information and make better-informed decisions without the usual hassle of deciphering dense documents.

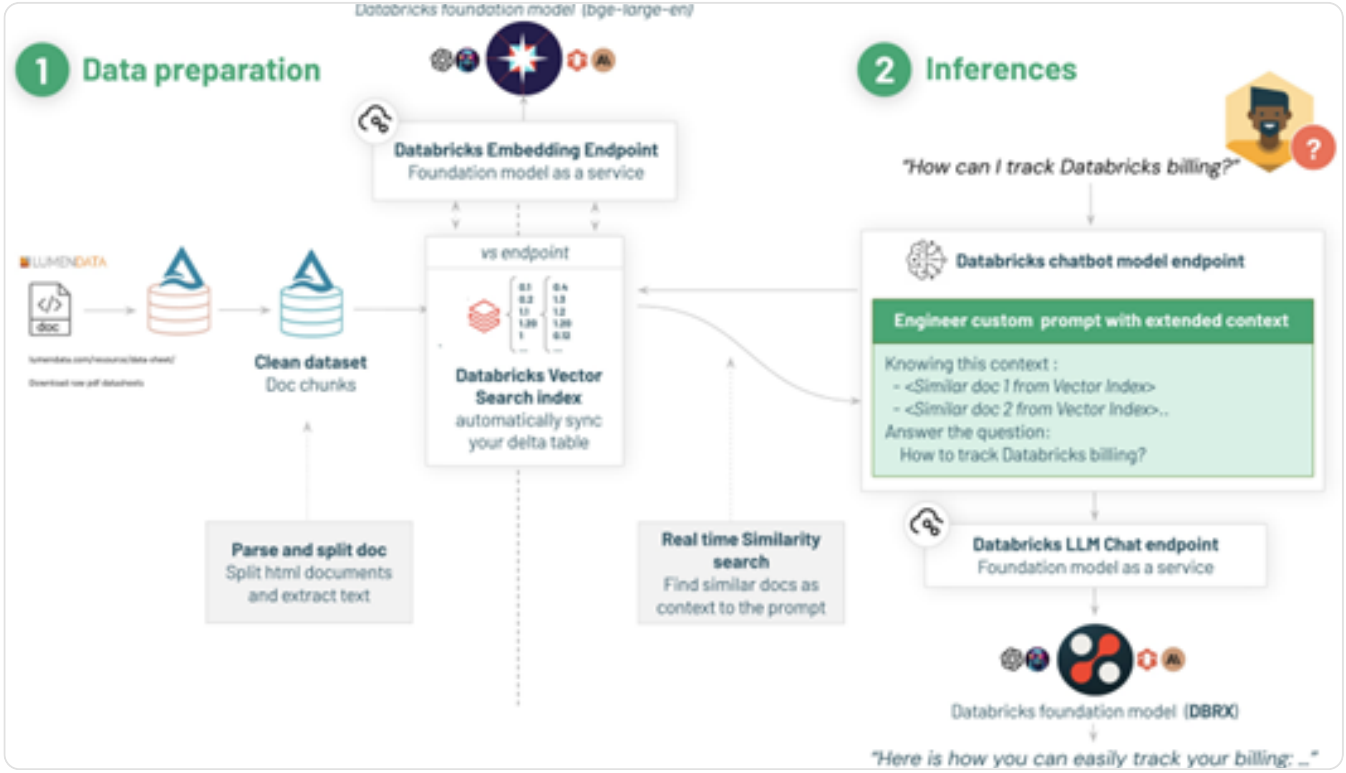
You make informed decisions and reach goals faster with real-time, actionable insights tailored to your specific needs and objectives. The chatbot's automated insights extraction will walk you through our data analytics skills/expertise and how we transform raw data into meaningful narratives that drive business growth and innovation.

You can gain a deeper understanding of our offerings through interactive engagement, exploring various features and services at your own pace, and discovering how our solutions can be customized to meet your unique challenges. The chatbot delivers dynamic, personalized insights for each client.

Seamlessly integrate LumenData/client data into the Databricks platform, enabling real-time data processing and analysis.

Chatbot Development: Step-by-Step Explanation

Build a conversational interface using Databricks ML and Gen-AI services to provide personalized insights and recommendations.



1. Datasheet Processing Pipeline:

- **Defining file paths** to access PDF datasheets stored in a Databricks volume. It then iterates through each PDF.
- **Extracting Content:** The process_pdf function employs a library PyPDF2 to extract text from the PDF

```
def process_pdf(file_path):
    # create a loader
    loader = PyPDFLoader(file_path)
    # load your data
    data = loader.load()
    # Split your data up into smaller documents with Chunks
    text_splitter = RecursiveCharacterTextSplitter(chunk_size=500, chunk_overlap=0)
    documents = text_splitter.split_documents(data)
    #print(documents)
    # Convert Document objects into strings
    texts = [str(doc) for doc in documents]
    return texts
```

- **Data Preparation:** The extracted text undergoes further processing within extract_document_data_from_datasheet to clean, parse, and structure the information. Data such as page numbers and URLs are captured here.

```
def extract_document_data_from_datasheet(texts,datasheet_url):
    print("extract document data after process pdf and convert it into structured table...")
    try:
        content = ""
        #document_data = {}
        #document_columns = ["source","page_content","page_number"]
        page_content_list = []
        page_number_list = []
        page_source_list = []
        for lines in texts:
            print(lines)
            print("-----********-------")
            page_content = lines.split("metadata=")[0].split("page_content=")[1].replace("'","")+"\nLumenData datasheet Reference:{datasheet_url}".format
            (datasheet_url=datasheet_url)
            page_content_list.append(page_content)
            metadata = ast.literal_eval(lines.split("metadata=")[1])
            source = metadata['source']
            page_source_list.append(source)
            page_number = metadata['page']
            page_number_list.append(str(page_number))
            #document_data.append((source,page_content,page_number))
            #print(document['source'])
            #print(document['page_number'])
            print("------------")
        document_data = {"source":page_source_list,"page_content":page_content_list,"page_number":page_number_list,"source_url":datasheet_url}
        return document_data

    except Exception as e:
        print("Error extract_document_data_from_datasheet!!\n"+str(e))
        raise Exception("Error extract_document_data_from_datasheet!!\n"+str(e))
```

- **Delta Table Storage:** The prepared data is then saved to a Delta table (temp_delta_table_name) within Databricks. Delta tables offer ACID (Atomicity, Consistency, Isolation, Durability) transactions, making them suitable for reliable data storage and retrieval.

```
def store_data_to_delta_table(document_data):
    try:

        create_delta_table_query = """CREATE TABLE IF NOT EXISTS {temp_delta_table_name} TBLPROPERTIES (delta.enableChangeDataFeed =
        true)""".format(temp_delta_table_name=temp_delta_table_name)

        #create a dataframe
        print("creating spark dataframe...")
        pd_df = pd.DataFrame(document_data)
        document_data_df = spark.createDataFrame(pd_df)
        #display(document_data_df)

        print("creating table if not exists...")
        print("create_table_query::"+create_delta_table_query)
        spark.sql(create_delta_table_query)

        print("writing data to delta table...")
        #TODO->try using merge to only update new records
        document_data_df.write.option("mergeSchema", "true").mode("append").format("delta").insertInto(temp_delta_table_name)
        print("data write completed to delta table...!!")


    except Exception as e:
        print("Error store_data_to_delta_table!!\n"+str(e))
        raise Exception("Error store_data_to_delta_table!!\n"+str(e))
```

2. Video Text Extraction (Optional):

- The code checks for video files (.mp4) in a designated location.
- If videos are present, use the MoviePy library to convert them to a more manageable audio format (e.g., MP3) for easier transcription.
- Whisper, a pre-trained speech recognition model, is employed to transcribe the audio content.
- Extracted text undergoes processing like the PDF pipeline and is stored in a Delta table

```
def covert_mp4_to_mp3(ld_videos):
    print("converting mp4 videos to mp3 format...")
    extracted_audio = []
    try:
        video_process_count = 0
        for video_file in ld_videos:
            print("------------------------------------------------")
            #convert mp4 to mp3
            print("converting video to audio::"+video_file)
            video = VideoFileClip(video_file)

            #write audio file to volume
            print("writing audio file to volume..")
            audio_name = video_file.split("/")[-1].replace(".mp4",".mp3")
            volume_location = os.path.dirname(video_file)
            print("audio_name::"+audio_name+"||volume_location::"+volume_location)

            video.audio.write_audiofile("/"+audio_name, codec="mp3")
            print("audio extraction completed...")

            #mv file to volume location
            print("moving audio files from base location to volume location...")
            #shutil.move("/"+audio_name,volume_location)

            shutil.move(os.path.join("/", audio_name), os.path.join(volume_location, audio_name))
            print("file_movement completed!!")

            extracted_audio.append(audio_name)
            video_process_count =video_process_count + 1
            print("completed video conversion no.--->"+str(video_process_count))
            print("**********************************")

        return extracted_audio
    except Exception as e:
```

3. Vector Search and LLM Integration:

- A unique row number is added to each record in the final Delta table (vector_search_delta_table_name) for indexing purposes.

```
def create_vector_search_delta_table(vector_search_delta_table_name):
    try:
        print("creating and adding unique row numbers to delta table")
        row_number_query = """CREATE OR REPLACE TABLE {vector_search_delta_table_name} AS SELECT ROW_NUMBER() OVER(ORDER BY PAGE_NUMBER) id,source,
        source_url,page_content,page_number FROM {temp_delta_table_name}""".format(vector_search_delta_table_name=vector_search_delta_table_name,
        temp_delta_table_name=temp_delta_table_name)
        print("Executing row number query::"+row_number_query)
        spark.sql(row_number_query)

        #alter table query to enable change data feed
        alter_table_query = """ALTER TABLE {vector_search_delta_table_name} SET TBLPROPERTIES (delta.enableChangeDataFeed = true)""".format
        (vector_search_delta_table_name=vector_search_delta_table_name)
        print("alter table query to enable change data feed::"+alter_table_query)
        spark.sql(alter_table_query)

        print("creating vector search delta table completed....")

        drop_temp_delta_table = """DROP TABLE IF EXISTS {temp_delta_table_name}""".format(temp_delta_table_name=temp_delta_table_name)
        print("dropping temp delta table created::"+drop_temp_delta_table)
        spark.sql(drop_temp_delta_table)
        print("temp table deleted...")


    except Exception as e:
        print("Error create_vector_search_delta_table!!\n"+str(e))
        raise Exception("Error create_vector_search_delta_table!!\n"+str(e))
```

- The function defines pre-configured variables for the vector search endpoint (vector_search_endpoint) and index name (vector_search_index). These point to existing resources within Databricks for efficient data retrieval.
- The setup_vector_search_endpoint_and_index function establishes a connection to the vector search infrastructure and retrieves the specified index. This index helps the chatbot quickly locate relevant information within the Delta tables.

```
def setup_vector_search_endpoint_and_index(vector_search_endpoint,vector_search_index):
    try:
        vsc = VectorSearchClient()
        deploy_client = mlflow.deployments.get_deploy_client("databricks")


        print("check for endpoint creation....")
        try:
            endpoint = vsc.get_endpoint(name=vector_search_endpoint)
            print("endpoint is already created and provisioned...skipping the endpoint creation!!")
        except Exception as e:
            print("Error\n"+str(e))
            print("creating vector search endpoint:: "+vector_search_endpoint)
            vsc.create_endpoint(
                name=vector_search_endpoint,
                endpoint_type="STANDARD"
            )
            print("endpoint created...")
```

```
    #get vector search index
    print("get vector index...")
    try:
        index = vsc.get_index(endpoint_name=vector_search_endpoint, index_name=vector_search_index)
        index.describe()
        print("vector index already created...skipping the creation process!!Syncing the index...")
        #vsc.get_index(endpoint_name=vector_search_endpoint, index_name=vector_search_index).sync()
    except Exception as e:
        print("Error\n"+str(e))
        #create vector index
        print("creating vector index..."+vector_search_index)
        #setting up model endpoint to be used for embedding creation
        embedding_model_endpoint = "databricks-bge-large-en"
```

```
    index = vsc.create_delta_sync_index(endpoint_name=vector_search_endpoint,
    source_table_name=vector_search_delta_table_name,
    index_name=vector_search_index,
    pipeline_type='TRIGGERED',
    primary_key="id", #get this column for delta table
    embedding_source_column="page_content", #get this column for delta table
    embedding_model_endpoint_name=embedding_model_endpoint
    )


    #check whether index is online -> might take few minutes
    import time
    while not index.describe().get('status').get('detailed_state').startswith('ONLINE'):
        print("Waiting for index to be ONLINE...")
        time.sleep(5)
    print("Index is ONLINE!!!")
    index.describe()

#test the similarity search using created vs endpoint and index
print("test the similarity search using created vs endpoint and index....")
test_question = "steps involved in developing databricks custom notification?"
results = index.similarity_search(
query_text=test_question,columns=["source","source_url","page_content"],num_results=2)

docs = results.get('result', {}).get('data_array', [])
print("result::")
print(docs)
return index

print("vector search setup completed...!!")
```

- The get_retriever function creates a "retriever" object that acts as an intermediary between the chatbot and the vector search system.
- To demonstrate functionality, the code retrieves the documents related to a sample query.

```
def get_retriever(vs_index):
    print("get and setup retriever....")

    try:
        vsc = VectorSearchClient()
        #vs_index = vsc.get_index(endpoint_name=vector_search_endpoint, index_name=vector_search_index).sync()
        #vs_index.describe()
        # Create the retriever
        vectorstore = DatabricksVectorSearch(
        vs_index, text_column="page_content", embedding="databricks-bge-large-en",columns=["source_url"]
        )

        return vectorstore.as_retriever(search_kwargs={'k': 4})

    except Exception as e:
            print("Error get_retriever\n"+str(e))
            raise Exception("Error get_retriever\n"+str(e))
```

4. Foundation LLM Setup:

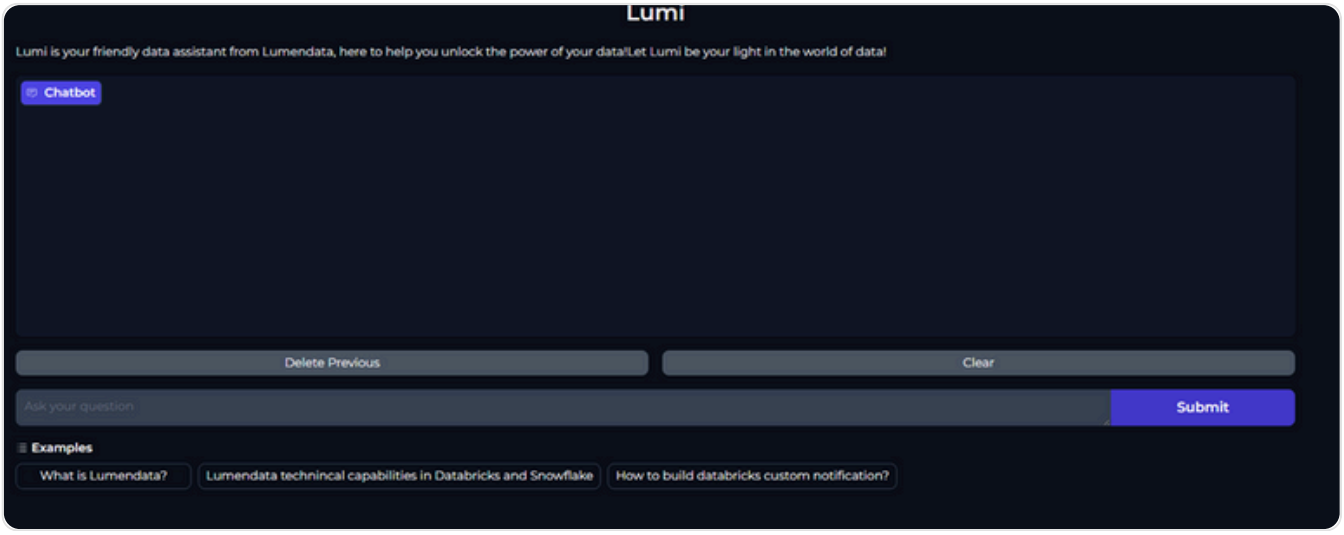
- The setup_foundation_llm_databricks function configures the chatbot's LLM capabilities. Databricks offers Foundation LLMs, pre-trained language models that can be fine-tuned for specific tasks. In this case, the code interacts with a Databricks Foundation LLM named "dbrx."

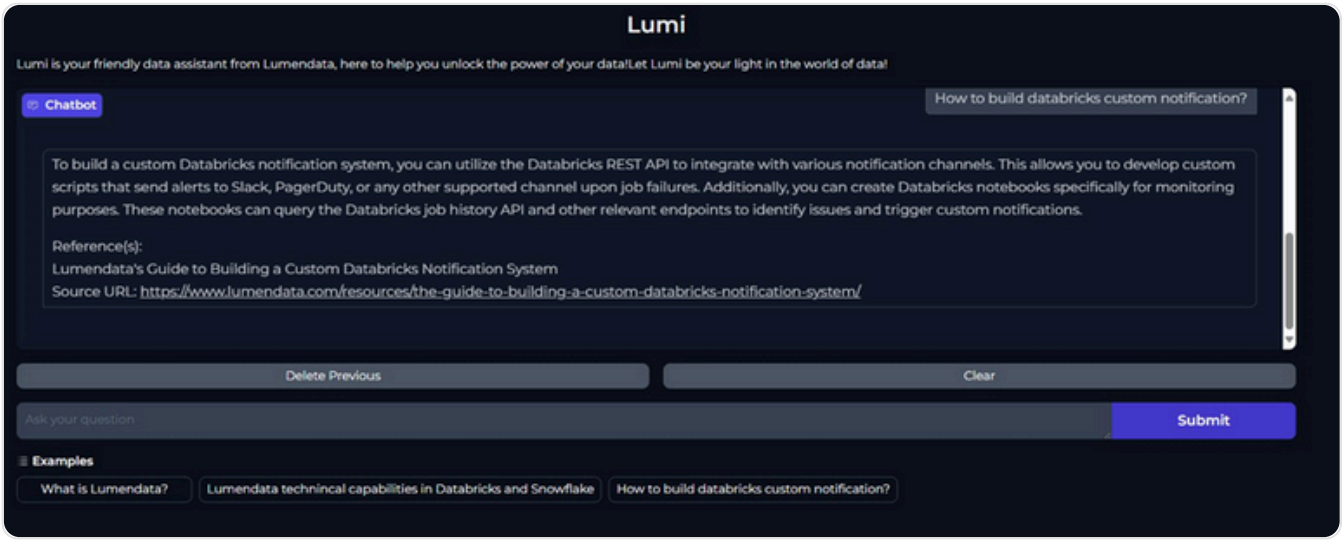
```
def setup_foundation_llm_databricks(vector_store):
    print("setting up databricks foundation llm model for querying....")

    try:
        prompt_template = """You are an assistant for Lumendata company users and customer. You are answering question related to Lumendata technical
        capabilities along with data analytics datasheets only. If the question is not related to one of these topics, kindly decline to answer. If you
        don't know the answer, just say that you don't know, don't try to make up an answer. Keep the answer as concise as possible.Also provide
        lumendata's datasheet source url links at the bottom section of answers from where the answer is derived from.
        Answer every question only referencing Lumendata datasheets and lumendata technical capabilities. Decline any non relevant questions.
        Use the following pieces of context to answer the question at the end:
        {context}
        Question: {question}
        Answer:

        References:
        """
        prompt = PromptTemplate(template=prompt_template, input_variables=["context", "question"])

        chain = RetrievalQA.from_chain_type(
            llm=ChatDatabricks(endpoint="databricks-dbrx-instruct"),
            chain_type="stuff",
            retriever=vector_store,
            chain_type_kwargs={"prompt": prompt}
        )

        return chain

    except Exception as e:
        print("Error setup_foundation_llm_databricks\n"+str(e))
        raise Exception("Error setup_foundation_llm_databricks\n"+str(e))
```

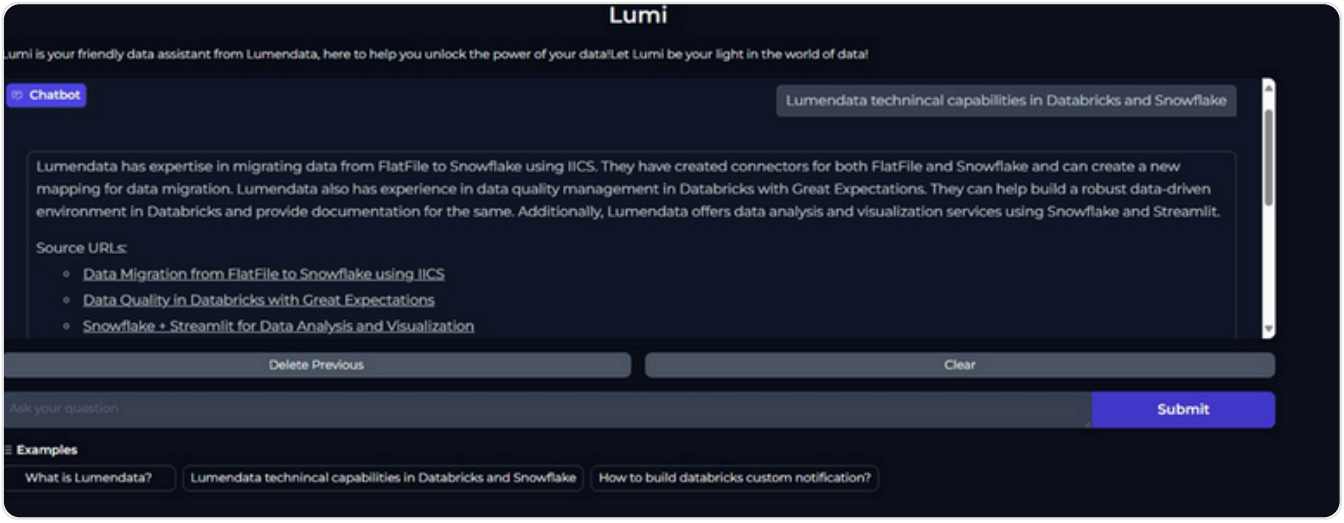
- The chain variable represents a sequence or pipeline that integrates the retriever (vector search) with the dbrx LLM, enabling the chatbot to access and process information retrieved from the Delta tables.

Chatbot UI

- **Gradio simplifies UI creation:** ChatInterface makes building chat interfaces easy, letting developers focus on functionality and aesthetics without complex code.
- **Customizable components:** Developers can adjust the chatbot's height, text input, title, theme, and button labels to match brand identity and user needs.
- **User-friendly features:** Features like examples, undo, and clear buttons enhance user experience by making interactions easier and correcting mistakes simpler.
- **Shareability:** The share=True feature allows the chatbot to be shared publicly, facilitating demos, user testing, and broader reach.

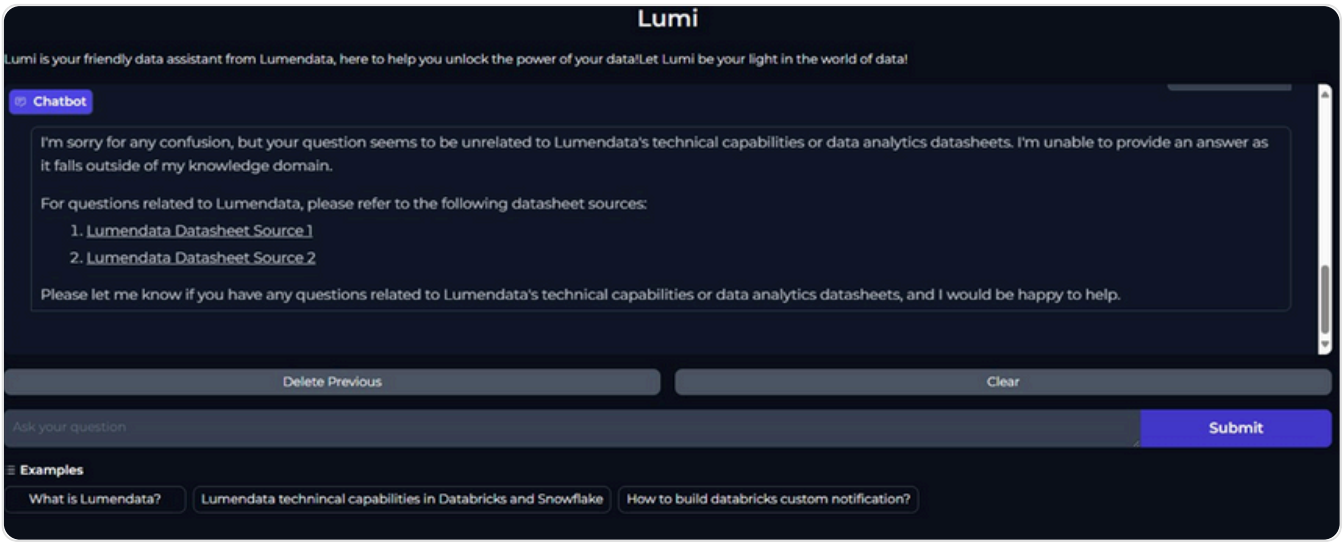






The chatbot is designed with a predefined feature that ensures it remains focused on LumenData-related queries. This functionality prevents the chatbot from answering any questions that fall outside the scope of LumenData, allowing it to provide more accurate and relevant information to users.

By doing so, the chatbot maintains a high level of expertise and reliability in its responses, ensuring that users receive the most pertinent insights and support regarding LumenData's offerings and services.



We have introduced several advanced features to enhance the capabilities of our chatbot. One such feature is the ability to convert video content into audio, and subsequently into text transcripts. This ensures that our chatbot processes and understands information from various media formats, making it more versatile and comprehensive in its responses.

Authors

Ritesh Chidrewar

Senior Consultant

About LumenData

LumenData is a leading provider of **Enterprise Data Management, Cloud & Analytics** solutions. We help businesses navigate their data visualization and analytics anxieties and enable them to accelerate their innovation journeys.

Founded in 2008, with locations in multiple countries, LumenData is privileged to serve over 100 leading companies. LumenData is **SOC2 certified** and has instituted extensive controls to protect client data, including adherence to GDPR and CCPA regulations.