Welcome to Part 6 of our series, "Building LLM-based Web Applications."

I'm Quang Duong, and today we will dive into the fascinating world of leveraging large language models (LLMs) to enhance web applications.

In this session, we will explore cutting-edge technologies and tools such as LangChain, Gradio, and the latest developments from industry leaders like OpenAI and Hugging Face.

By the end of this presentation, you will have a comprehensive understanding of how to integrate these powerful models into your web applications, creating more intelligent, responsive, and engaging user experiences. Let's get started!

---

In this part, we will focus on building a variety of LLM-based web applications.

Our journey will begin with creating task-specific AI assistants that can handle specialized tasks with precision.

Next, we'll develop a simple AI chatbot to understand the basics of conversational interfaces.

Moving forward, we'll explore the construction of RAG (Retrieval Augmented Generation)-based AI chatbots, which combine retrieval mechanisms with generation capabilities for enhanced responses.

Finally, we'll look into building agent-based AI chatbots, which offer more sophisticated and autonomous interaction capabilities.

By the end of this part, you'll have hands-on experience in developing these diverse and powerful AI tools.

---

For the app development, we will utilize a combination of backend and frontend frameworks to build our LLM-based web applications.

For the backend, we will primarily use Python. Specifically, for the demonstration purpose and the hardware and computation constraints, we will leverage the OpenAI API for GPT-3.5 as our large language model, orchestrate our LLM with LangChain, and use FAISS as our vector database. Additionally, we will integrate external tools such as Wikipedia and Tavily to enrich our applications.

On the frontend side, we will also use Python, particularly focusing on Gradio to create interactive user interfaces.

This combination of powerful backend and frontend technologies will enable us to build quick, efficient and user-friendly AI applications.

---

Let’s first explore building task-specific AI Assistant

Task-specific AI assistants operate by leveraging large language models (LLMs) trained on publicly available, non-private data. When a user makes a specific request related to a particular task, the AI assistant utilizes a customized prompt tailored to that request. The LLM processes this input and generates an appropriate response, which is then delivered back to the user. This interaction enables the AI assistant to provide precise and relevant answers, enhancing the user's experience and efficiency in handling specialized tasks. Through this mechanism, we can build intelligent systems that are highly effective in addressing targeted needs.

---

Task-specific AI assistants can be developed for a wide range of applications to cater to various needs and industries. In this section we will learn some examples of task-specific AI assistant applications:

* **Culinary AI Assistant:** Helps users with recipes, cooking tips.
* **Marketing AI Assistant:** Assists in creating marketing content,
* **Customer AI Assistant:** Provides customer support, answers queries,
* **SQL-querying AI Assistant:** Assists with writing and optimizing SQL queries for database management.
* **Travel AI Assistant:** Helps users plan book accommodations, provide travel tips.
* **Summarization AI Assistant:** Summarizes lengthy documents, articles, and reports.
* **Interview AI Assistant:** Prepares users for interviews by providing common questions and feedback.

These examples illustrate the versatility and potential of task-specific AI assistants in enhancing productivity and providing specialized support in various fields.

---

Before we jump into a specific AI assistant, here is a general code snippet for creating an AI assistant using a large language model (LLM) like GPT-3.5-turbo. This example demonstrates how to load environment variables, initialize the chat model, define the prompt template, create the prompt, and get the response from the model.

This script outlines the fundamental steps needed to set up an AI assistant. By customizing the prompt template, you can tailor the assistant to various specific tasks, making it a versatile tool for diverse applications.

---

Let’s talk about Culinary AI Assistant.

Let’s see how a Culinary AI Assistant works. This AI assistant, powered by OpenAI GPT-3.5-turbo API, LangChain, Wikipedia Search Tool, and Gradio, assists users in cooking by providing detailed information about various dishes.

The user can select a dish from a dropdown menu, such as Pizza Margherita, Spaghetti Carbonara, or Lasagna. Once a dish is selected, the assistant provides the ingredients and the recipe tailored to the number of persons specified by the user. Additionally, the assistant can offer interesting stories and facts related to the dish, enhancing the user's culinary experience with educational content.

This assistant is designed to make cooking more accessible and enjoyable, offering precise and relevant information to help users create delicious meals while also learning about the history and cultural significance of the dishes they prepare.

---

In this slide, we illustrate how to parameterize prompts for the Culinary AI Assistant. This involves dynamically generating a recipe based on user input, such as the selected dish and the number of persons.

**Parameterizing Prompt**

The user interface allows the selection of a dish (e.g., Pizza Margherita) and the number of persons the recipe should serve. Once these parameters are set, the AI generates a customized recipe.

This code demonstrates how to create a prompt template that incorporates user-specified parameters, such as the dish name and the number of servings. By using these parameters, the AI can provide a tailored response that includes a list of ingredients and a step-by-step recipe, ensuring the instructions are clear and relevant to the user's needs.

This approach allows the AI assistant to generate highly customized and accurate culinary information, making it a valuable tool for users looking to create specific dishes with precise quantities and instructions.

---

**Search Tool + Parameterized Prompt**

In this slide, we demonstrate how to combine a search tool with parameterized prompts to enhance the Culinary AI Assistant's capabilities. By integrating external data sources such as Wikipedia, we can enrich the assistant's responses with interesting stories or facts about the selected dish.

**Example Interface:**

The user selects a dish (e.g., Pizza Margherita) and the number of persons. The assistant then provides not only the recipe but also engaging background information about the dish.

This code snippet shows how to:

1. **Query Wikipedia**: Using a parameterized prompt to search for the origin story of the selected dish.
2. **Generate Interesting Stories**: Using the search results, the AI generates a prompt to present the three most interesting facts or stories about the dish.

**Example Output:**

The assistant might provide historical and cultural insights

By combining parameterized prompts with a search tool, the AI assistant can provide a richer and more informative user experience, offering both practical and engaging content.

---

**To Sum Up**

This app allows us to achieve two key functionalities:

* **Parameterizing efficiently prompt based on interactive user-input interface**: By providing an intuitive interface, users can input specific details that the AI assistant can use to generate tailored and accurate responses. This capability ensures that the assistant's output is highly relevant to the user's needs.
* **Combine parameterized prompt with search tool like Wikipedia**: Integrating search tools like Wikipedia with parameterized prompts enables the AI assistant to enrich its responses with additional information. This combination allows for generating content that is not only accurate but also informative and engaging, leveraging external data sources to provide deeper insights and context.

---

Let’s take a look at Marketing AI Assistant.

**Marketing AI Assistant**

The Marketing AI Assistant is designed to help create compelling marketing content for products, leveraging the power of OpenAI GPT-3.5-Turbo API, LangChain, and Gradio. This tool assists marketers in generating persuasive text tailored to their product features, target clients, and desired word count.

**Example Interface:**

The interface allows the user to input:

* **Product**: A new line of organic skincare products.
* **Features**: Natural Ingredients, Free from Harmful Chemicals, Environmentally Sustainable, Cruelty-Free.
* **Target clients**: Health-conscious consumers.
* **Word Count**: Adjustable between 100 and 300 words.

**Generated Text:**

The assistant produces a marketing message, ensuring it aligns with the specified features and appeals to the target audience. Here's an example output: …

Another style of this message could be: …

**Example Code Snippet:**

To achieve this, the following prompt and code can be used:

This prompt allows the AI to generate a tailored marketing message that highlights the product's unique features and appeals to the specified target audience. The flexibility in word count ensures that the content fits various marketing platforms, from social media posts to detailed product descriptions. By using this AI assistant, marketers can efficiently create high-quality, persuasive content that resonates with their audience.

---

**First Request**

In this example, the Marketing AI Assistant generates a marketing message based on user-provided inputs for a new product. The interactive interface allows the user to specify details about the product, its features, target clients, and desired word count.

**Example Interface:**

* **Product**: A new line of organic skincare products.
* **Features**: Natural Ingredients, Free from Harmful Chemicals, Environmentally Sustainable, Cruelty-Free.
* **Target clients**: Health-conscious consumers.
* **Word Count**: Set to 150 words.

**Generated Text:**

The assistant generates a marketing message that is tailored to these inputs:

**Example Code Snippet:**

The following code snippet shows how to construct and use the prompt for generating this marketing message:

This code dynamically incorporates the user inputs into the prompt, ensuring the generated message is specific and relevant to the product and audience. This example highlights the efficiency and customization capabilities of the Marketing AI Assistant, demonstrating its potential to create high-quality, targeted marketing content based on interactive user inputs.

---

**Second Request**

In this example, the Marketing AI Assistant is used to transform the previously generated marketing message into a new style, making it more engaging by using an upbeat tone and emojis. This demonstrates the assistant's ability to reformat and enhance content based on additional user specifications.

**Example Interface:**

The interface now shows the original generated text and allows the user to specify a new format for the message, including an upbeat tone and the use of emojis.

**Generated Text in New Style:**

**Example Code Snippet:**

The following code snippet shows how to request the new marketing message format:

This code dynamically takes the original marketing message and reformat it according to the new style specifications provided by the user.

This example illustrates how the Marketing AI Assistant can be used not only to generate initial content but also to refine and enhance existing content to meet different stylistic and engagement needs, making it a versatile tool for modern marketing efforts.

---

**Combining All Responses**

In this final step, we combine the original marketing message with the new style to create a comprehensive output that showcases different presentation formats for the same content. This approach provides versatility and allows users to choose the most suitable format for their marketing needs.

**Example Interface:**

The interface displays both the original and the reformatted marketing messages, separated by a clear delimiter for easy comparison.

**Generated Text:**

**Example Code Snippet:**

The following code combines the original and new styles of the marketing message into a final result:

This code ensures that both versions of the marketing message are presented in a cohesive format, making it easy for users to compare and choose the most effective style for their marketing strategy.

By combining all responses, the Marketing AI Assistant provides a complete solution that showcases the flexibility and creativity of AI-generated content, catering to different marketing needs and preferences.

---

**All-in-One Prompt**

This example demonstrates how to generate two distinct marketing messages for a product using a single all-in-one prompt. The generated messages cater to different styles: one in a professional tone as a paragraph and the other in a more engaging tone using main points and emojis.

**Example Interface:**

* **Product**: A new line of organic skincare products.
* **Features**: Natural Ingredients, Free from Harmful Chemicals, Environmentally Sustainable, Cruelty-Free.
* **Target clients**: Health-conscious consumers.
* **Word Count**: Set to 150 words.

**Generated Text:**

Message 1 (Professional Tone):

Message 2 (Upbeat Tone with Emojis):

**Example Code Snippet:**

The following code snippet shows how to construct the all-in-one prompt:

This code snippet dynamically incorporates the user inputs to generate two different styles of marketing messages in a single request.

**Combining Messages:**

By using this approach, the Marketing AI Assistant can provide versatile outputs in one go, making it efficient and effective for users to obtain multiple styles of content from a single input. This versatility helps cater to different marketing platforms and audience preferences, enhancing the overall impact of the marketing strategy.

---

**To Sum Up**

This app provides two main functionalities:

1. **Understanding how to create a chain of requests/prompts**:
   * The result of the first request becomes the input for the second one, allowing us to create more tailored and refined responses.
   * **Example**: Input 1 → Request 1 → Response 1 → Request 2 → Response 2
2. **Creating multiple responses in different styles based on a single prompt/request**:
   * Allows the generation of diverse content styles from one set of inputs, enhancing flexibility and adaptability for various marketing needs.
   * **Example**: Input → One Request → Response 1, Response 2, ...

By utilizing these capabilities, users can efficiently generate high-quality, versatile content that is well-suited to different contexts and audiences. This app showcases the power of leveraging AI to streamline and enhance the content creation process.

---

Now, let’s talk see another AI assistant example, Customer AI Assistant

**Customer AI Assistant**

The Customer AI Assistant is designed to analyze customer feedback and generate appropriate responses. Powered by OpenAI GPT-3.5-Turbo API, LangChain, and Gradio, this assistant helps businesses effectively manage customer relations by automating the sentiment analysis and response generation process.

**Example Interface:**

* **Customer Feedback**: Input provided by the customer, such as "I'm thrilled with my camera purchase! The image quality is exceptional, and it's remarkably user-friendly. This camera exceeded my expectations, and its portability is a bonus. It's a must-have for any photography enthusiast."
* **Review's Sentiment**: The AI analyzes the sentiment of the feedback and determines it to be "POSITIVE."
* **Generated Email**: The AI generates a response email based on the analyzed sentiment.

**How It Works:**

The AI assistant takes the customer feedback, analyzes the sentiment, and uses this information to generate a personalized email response. This process helps businesses maintain a positive relationship with customers by promptly addressing their feedback.

This code demonstrates how the AI assistant can efficiently process customer feedback, determine the sentiment, and generate a suitable response, enhancing customer service capabilities and ensuring timely communication.

By leveraging this Customer AI Assistant, businesses can improve their customer engagement and satisfaction through personalized and context-aware responses.

---

**Requesting Output in a Specific Format**

In this example, the Customer AI Assistant is configured to process customer feedback, determine the sentiment, and generate an email response. The assistant formats the output as a JSON object, making it easier to parse and use programmatically.

**Example Interface:**

* **Customer Feedback**: "I'm thrilled with my camera purchase! The image quality is exceptional, and it's remarkably user-friendly. This camera exceeded my expectations, and its portability is a bonus. It's a must-have for any photography enthusiast."
* **Review's Sentiment**: Determined to be "POSITIVE."
* **Generated Email**: The assistant generates a response email based on the sentiment.

**Explanation:**

1. **Define the Feedback Input**: Customer feedback is provided as an input.
2. **Define the Prompt**: The prompt requests the AI to determine the sentiment and generate an email response, formatting the output as a JSON object.
3. **Invoke the AI**: The chain.invoke() function processes the prompt.
4. **Parse the JSON Response**: The response is parsed as a JSON object to easily extract the sentiment and email response.

This approach ensures that the AI-generated content is structured and easy to integrate into applications, enhancing the efficiency of customer service operations by automating sentiment analysis and response generation.

---

**What We Have Learned**

* **Asking for Output in a Specific Format (e.g., JSON)**:
  + **Benefit**: This approach simplifies the integration of AI-generated responses into applications, making it easier to handle and manipulate the data programmatically.
  + **Implementation**: By specifying the desired format in the prompt, we can receive structured output that can be seamlessly integrated into various applications, enhancing the efficiency of data processing and response generation.

This capability underscores the flexibility and utility of AI in creating structured, actionable outputs that can be directly used in software systems, improving overall workflow and user experience.

---

Now let’s explore another interesting AI assistant. SLQ-querying AI Assistant

**SQL-Querying AI Assistant**

The motivation to build the SQL-Querying AI Assistant stems from the need to make database querying accessible to non-technical users, enhance efficiency by automating query generation, reduce errors, and support quick, accurate decision-making. This app bridges the gap between natural language and SQL, enabling users to retrieve data effortlessly while also serving as an educational tool and providing scalable, versatile solutions for various industries. By leveraging AI to interpret and execute queries, the assistant ensures reliable and precise data retrieval, significantly improving user experience and productivity.

The SQL-Querying AI Assistant helps users interact with databases by generating SQL queries based on natural language questions. Powered by OpenAI GPT-3.5-Turbo API, LangChain, SQLite, and Gradio, this assistant simplifies the process of querying databases and interpreting results.

**Example Interface:**

* **Choose a database**: Chinook.db
* **Ask a question about db**: "How many employees are there?"

**Generated SQL Query:**

Response in Natural Language:

**How It Works:**

1. **User Input**: The user selects a database and asks a question in natural language.
2. **Query Generation**: The assistant converts the natural language question into an SQL query.
3. **Execution and Interpretation**: The generated SQL query is executed against the selected database, and the result is interpreted and presented in natural language.

This AI assistant streamlines the process of interacting with databases, making it accessible even to those who may not be proficient in SQL. It enhances productivity by automating query generation and result interpretation.

---

**SQL-Querying AI Assistant - Complete Workflow**

This implementation includes the entire process from receiving a user's natural language question to generating a corresponding SQL query and providing a natural language response based on the query results.

**Key Functions and Workflow**

1. **Predict Query Function**:
   * Initializes the language model if needed.
   * Defines a template for the SQL query prompt.
   * Connects to the specified database.

**Get Schema Function**:

* Retrieves the schema information of the tables from the database.

**Chain to Generate SQL Query**:

* Combines schema retrieval and prompt to generate the SQL query using a language model.

**Generate SQL Query**:

* Uses the chain to generate the SQL query.

**Final Prompt and Chain for Natural Language Response**:

* Defines the final template and chain to generate a natural language response based on the SQL query and its results.

**Summary:**

* **Predict Query Function**: Initializes the language model and connects to the database.
* **Get Schema Function**: Retrieves the schema information.
* **Chain to Generate SQL Query**: Creates an SQL query based on the user's question.
* **Final Chain**: Generates a natural language response using the query and its results.

This complete implementation showcases the SQL-Querying AI Assistant's ability to convert natural language questions into SQL queries and provide comprehensive responses, making database interactions more accessible and efficient.

---

**What We Have Learned from the SQL-Querying AI Assistant**

1. **Natural Language to SQL Conversion**:
   * The assistant can effectively convert natural language questions into SQL queries.
   * This capability makes it easier for users who may not be proficient in SQL to interact with databases and retrieve the information they need.
2. **Schema Utilization for Accurate Query Generation**:
   * By retrieving and utilizing the database schema, the assistant ensures that the generated SQL queries are accurate and relevant to the database structure.
   * This process enhances the reliability and precision of the queries, reducing errors and improving the efficiency of database interactions.
3. **Integration and Response Generation**:
   * The assistant can execute the generated SQL queries and produce natural language responses based on the query results.
   * This feature makes the data retrieval process more user-friendly and accessible, providing clear and understandable results that can be easily integrated into applications or used for decision-making.

---

Now let’s have a look at more ideas for AI assistant applications. I will just give you an introduction to the user interface, but not detailed instructions for each component. As always, you have the source code to play with, so let’s explore it on your own.

---

Let’s take a look at the Travel AI Assistant

The "Travel AI Assistant" app is designed to help users plan their trips by providing personalized travel advice and recommendations. Here’s the general idea of how it works and what the expected outputs are:

**How It Works:**

1. **User Input**: The user specifies their travel destination and can adjust a slider to indicate their top interests (e.g., cultural sites, food, adventure activities, etc.).
2. **AI Processing**: Powered by OpenAI’s GPT-3.5-Turbo API, along with Langchain and DuckDuckGoSearch tools, the app processes the user's input to generate relevant travel advice.
3. **Recommendations**: The AI assistant provides a list of top tourist places and famous foods based on the user's chosen destination and interests.

**Expected Outputs:**

* **Tourist Places**: The app suggests a list of must-visit tourist attractions at the specified destination. For example, if the destination is France, it might list the Eiffel Tower, the Louvre Museum, the Palace of Versailles, Mont Saint-Michel, and the French Riviera.
* **Famous Foods**: The app also provides recommendations for famous local dishes that the user should try. In the case of France, it might include dishes like croissants, Coq au Vin, Ratatouille, Bouillabaisse, and Crème Brûlée.

**Summary:**

The Travel AI Assistant helps users by offering curated travel recommendations tailored to their interests and destination choices. This helps users quickly identify key attractions and local culinary experiences, making their travel planning more efficient and enjoyable.

---

Now let’s look at Summarization AI Assistant

The "Summarization AI Assistant" is an application designed to generate concise summaries of long texts using advanced language models. Here’s the general idea of how it works and what the expected outputs are:

**How It Works:**

1. **User Input**: The user inputs a lengthy text into the provided text box.
2. **AI Processing**: Using the OpenAI GPT-3.5-Turbo API, along with Langchain and Gradio, the app processes the input text to extract the most critical points and condense the information.
3. **Summary Generation**: The AI generates a summary that captures the key elements and main ideas of the input text.

**Expected Outputs:**

* **Concise Summary**: The output is a summarized version of the input text. It includes the essential points, making the original information easier and quicker to understand. For example, the text about climate change is summarized into bullet points highlighting the main threats, causes, and necessary actions to address climate change.

**Summary:**

The Summarization AI Assistant helps users by transforming long and detailed texts into brief, digestible summaries. This assists in quickly understanding the core message of the content, which is particularly useful for reviewing large volumes of information efficiently.

---

Now let’s look at Interview AI Assistant

The "Interview AI Assistant" is an application designed to help users prepare for job interviews by generating relevant interview questions and answers tailored to specific job positions and programming languages. Here’s the general idea of how it works and what the expected outputs are:

**How It Works:**

1. **User Input**: The user selects a job position (e.g., AI Engineer) and a programming language (e.g., Python). They can also specify the number of question-answer pairs they want to generate.
2. **AI Processing**: Utilizing the OpenAI GPT-3.5-Turbo API, along with Langchain and Gradio, the app processes this information to generate a list of relevant interview questions and corresponding answers.
3. **Question and Answer Generation**: The AI creates a set of questions that are typically asked for the specified job role, along with detailed answers that demonstrate a thorough understanding of the topics.

**Expected Outputs:**

* **Interview Questions**: A list of interview questions related to the job position and programming language chosen by the user. For instance, questions might cover fundamental concepts, practical applications, and specific technical skills.
* **Detailed Answers**: The app provides comprehensive answers to each of the generated questions, helping the user understand the key points that should be covered in their responses.

**Example:**

For the position of an AI Engineer with Python, the app might generate questions like:

1. **What is the difference between deep learning and machine learning in the context of AI?**
   * Answer: Explanation about deep learning as a subset of machine learning that uses neural networks, compared to traditional machine learning which requires manual feature engineering.
2. **How can you prevent overfitting in a deep learning model?**
   * Answer: Techniques such as dropout, early stopping, regularization, data augmentation, and using more data.
3. **Explain the concept of transfer learning in the context of AI and how it can be applied in deep learning models.**
   * Answer: Description of transfer learning, leveraging pre-trained models, and its application in improving model performance on new tasks with limited data.

**Summary:**

The Interview AI Assistant aids users in preparing for job interviews by generating relevant and customized interview questions and answers. This tool helps candidates anticipate potential questions and practice their responses, thereby improving their confidence and readiness for actual interviews.

---

Now let’s talk about AI chatbot based on Large Language Model. Let’s start with a simple AI Chatbot.

---

This diagram illustrates the basic interaction between a user and an AI chatbot powered by large language models, or LLMs. The AI chatbot, trained on publicly available, non-private data up to a specific point in time, processes the user's questions and generates relevant answers. The flow of communication is straightforward: the user inputs a question, and the AI chatbot responds with an answer. This simple yet powerful exchange demonstrates how AI can efficiently handle a wide range of inquiries, providing users with quick and accurate information.

---

Now let's take a closer look at a simple AI chatbot demo. This demo showcases an AI chatbot powered by OpenAI's GPT-3.5-Turbo API, integrated with Langchain and Gradio for enhanced functionality. In this example, the user asks, "What is climate change?" The chatbot responds with a detailed explanation, describing climate change as long-term alterations in temperature, precipitation, and other atmospheric conditions, primarily driven by human activities such as burning fossil fuels and deforestation. This example highlights the chatbot's ability to provide comprehensive, informative answers to user queries, demonstrating its potential in facilitating user interactions and delivering accurate information efficiently.

---

This slide illustrates the chat interface block used in our AI chatbot. On the left, we see a snippet of code defining the chat interface using the gr.ChatInterface function from the Gradio library. This function integrates the prediction model and provides example queries such as "What is climate change?" and "What are the benefits of renewable energy?" to guide users in their interactions. On the right, we have the visual representation of the chat interface where users can type their questions and receive responses. This setup demonstrates how the backend code translates into a user-friendly interface, enabling seamless interaction between the user and the AI chatbot.

---

This slide demonstrates the implementation of a simple AI chatbot without memory. The code snippet on the slide shows the step-by-step process to initialize and run a basic chatbot using the GPT-3.5-turbo model. We start by initializing the chat model with a specified temperature setting. Next, we define a prompt template that instructs the chatbot to act as a helpful assistant and respond to user queries. The prompt is then formatted into a template, and a chatbot chain is created by linking the prompt with the language model and an output parser. Finally, we define the inputs for the chatbot and invoke the chain to get the chatbot's response. This example highlights how a straightforward setup can create an effective AI chatbot capable of providing consistent and accurate answers without retaining conversation history.

---

In this slide, we see an example interaction with a simple AI chatbot that operates without memory. The conversation begins with the user, Quang, introducing himself. The chatbot responds warmly, asking how it can assist. When Quang asks the chatbot to recall his name, the chatbot politely explains that it doesn't have access to personal information, highlighting its lack of memory retention. This example underscores the chatbot's design focus on privacy and security, ensuring user data isn't stored or used beyond the immediate session. Despite not retaining context, the chatbot remains functional and helpful within individual interactions.

---

This slide showcases the implementation of a simple AI chatbot with memory. The code on the left initializes the chat model using the GPT-3.5-turbo model and defines a prompt template that includes the conversation history. This allows the chatbot to provide consistent and accurate responses while maintaining context. On the right, we see the steps to initialize conversation memory using the ConversationBufferMemory function. The chatbot chain is defined to pass the conversation history, ensuring the chatbot can recall previous interactions. Inputs are defined and processed, and the chatbot's response is generated and stored for future reference. This setup highlights how incorporating memory into the chatbot enables it to deliver more personalized and context-aware interactions, enhancing user experience by remembering prior conversations.

---

In this slide, we observe an interaction with a simple AI chatbot equipped with memory. The user, Quang, introduces himself, and the chatbot greets him warmly. When Quang later asks, "What is my name?" the chatbot correctly recalls and responds, "Your name is Quang." This example demonstrates the chatbot's ability to retain and utilize information from previous interactions, enhancing the conversational experience. By maintaining context and remembering key details, the chatbot can provide more personalized and coherent responses, which is particularly valuable for extended or ongoing user interactions.

---

Now let’s talk about RAG-based AI Chatbot

This slide highlights some limitations of simple AI chatbots. As depicted, these chatbots are trained on publicly available, non-private data up to a specific point in time. Consequently, they may struggle to provide accurate responses regarding recent developments or access private knowledge. For example, they cannot offer up-to-date information on recent events or personal details specific to individual users. These limitations emphasize the need for continuous updates and the integration of secure, private data handling mechanisms to enhance the chatbot's accuracy and personalization capabilities.

---

This slide introduces the concept of a Retrieval-Augmented Generation (RAG)-based AI chatbot. Unlike traditional AI chatbots, a RAG-based chatbot can access up-to-date or private data sources, such as text files, PDFs, tables, and Word documents. The diagram illustrates how the information retrieval process works: the AI chatbot, trained on publicly available, non-private data up to a certain point, can fetch relevant information from these external data sources to provide more accurate and current responses. This approach addresses the limitations of standard chatbots by integrating recent and private knowledge, thus enhancing the chatbot's ability to deliver precise and contextually relevant answers to user queries.

---

Building a RAG-based AI chatbot involves three key steps. First, we generate a vector database, which involves converting textual data into vectors that the model can understand and process. Second, we implement information retrieval, enabling the chatbot to fetch relevant data from various sources, such as documents, databases, and other repositories. Lastly, we integrate augmented generation with a chat user interface (UI). This step combines the retrieved information with the AI model's capabilities to generate coherent and contextually accurate responses, which are then presented through a user-friendly chat interface. These steps collectively ensure that the RAG-based AI chatbot can provide up-to-date and precise answers, leveraging both the pre-trained model and real-time information retrieval.

---

The first step in building a RAG-based AI chatbot is generating a vector database. This process begins with loading data from various up-to-date or private data sources, such as text files, PDFs, tables, and Word documents. Once the data is loaded, it undergoes vector embedding, where the textual information is converted into numerical vectors that the AI model can process. These vectors are then stored in a vector database. This step is crucial as it prepares the data in a format that facilitates efficient retrieval and processing during the chatbot's operations, enabling it to provide accurate and relevant responses based on the most current and specific information available.

---

The second step in building a RAG-based AI chatbot is information retrieval. When a user poses a question, it is first converted into an embedding, a numerical representation that captures the semantic meaning of the query. This embedding is then used to perform a similarity search against the vector database we generated in the first step. The similarity search identifies the top-k pieces of information, or chunks of text, that are most relevant to the user's query. These retrieved pieces of information are then used by the AI chatbot to generate a precise and contextually appropriate response. This process ensures that the chatbot can provide answers based on the most relevant and current data available.

---

The final step in building a RAG-based AI chatbot is augmented generation combined with a chat user interface (UI). Once the top-k retrieved information chunks have been identified through the similarity search, these chunks are used by the AI chatbot, which is powered by large language models (LLMs) trained on publicly available data. The chatbot processes this information to generate accurate and contextually relevant answers to user questions. These answers are then delivered to the user through a seamless chat interface, facilitating an interactive and user-friendly experience. This integration ensures that the chatbot not only leverages its pre-trained knowledge but also incorporates up-to-date and specific information, providing more comprehensive and reliable responses.

---

These slides provide a detailed look at the implementation of a RAG-based AI chatbot. The process starts by importing necessary modules from the LangChain library, such as ChatOpenAI, FAISS, PromptTemplate, and PyPDFLoader. The first step involves creating chunks of text from documents loaded using PyPDFLoader and then splitting these documents into manageable chunks using CharacterTextSplitter. These text chunks are then embedded and stored in a vector store using FAISS. Next, we set up a retriever from the vector store to handle search queries.

The second slide demonstrates how to initialize the OpenAI model, defining a prompt template for the chatbot to use retrieved context for answering questions. The LangChain is created, linking the context retriever, prompt, and the language model (LLM). A function, make\_llm\_response, generates a response by invoking the RAG chain with the user's question. An example question illustrates how the chatbot processes and responds to queries using the retrieved context.

This detailed implementation showcases the integration of various components to build an effective RAG-based AI chatbot capable of providing contextually enriched responses by leveraging both pre-trained knowledge and real-time information retrieval.

---

This slide demonstrates a practical application of a RAG-based AI chatbot in action. The demo showcases how the chatbot can interact with a PDF file, leveraging the power of OpenAI's GPT-3.5-Turbo API, Langchain, and Gradio. In this example, a user uploads a PDF document containing a recipe for Margherita pizza. The chatbot processes the document and successfully retrieves the context to answer the user's question, "What is the topic of this document?" The chatbot responds accurately by identifying the document's topic as a recipe for Margherita pizza. This demo illustrates the chatbot's capability to handle and interpret external documents, providing users with precise and contextually relevant information based on the content of the uploaded files.

---

Now let’s look at Agent-based AI Chatbot

While AI chatbots have made significant strides in recent years, they still have some notable limitations. One challenge is their ability to access and process daily news. Since chatbots are typically trained on data available up to a certain point in time, they may not have the latest information on current events unless specifically designed to retrieve and process real-time data. Another limitation is their capacity to perform complex mathematical calculations. While they can handle basic arithmetic, more advanced mathematical operations often require additional programming and integration with specialized tools. Additionally, AI chatbots might struggle with context-specific tasks (ECT) that require a deep understanding of nuanced or specialized information. These limitations highlight the areas where ongoing improvements and integrations are necessary to enhance the functionality and reliability of AI chatbots.

---

This slide provides concrete examples of the limitations of AI chatbots. The first example shows a chatbot accurately calculating the result of 68 raised to the 0.36 power, demonstrating its capability to handle mathematical queries to a certain extent. However, the second example highlights a significant limitation: the chatbot's inability to provide real-time information. When asked about the winner of the men's singles at Roland Garros 2024, the chatbot responds that it does not have real-time information and suggests checking a reliable sports news source. These examples illustrate that while AI chatbots can perform certain tasks well, they may struggle with up-to-date information retrieval and complex mathematical computations, emphasizing the need for continuous improvements and integrations.

---

This slide introduces the concept of an agent-based AI chatbot, which leverages external tools to enhance its functionality. Unlike traditional AI chatbots, which rely solely on pre-trained data up to a certain point in time, agent-based AI chatbots can query external tools such as search engines and mathematical computation tools to retrieve real-time information and perform complex calculations. When a user asks a question, the AI chatbot can determine if external tools are needed to generate an accurate response. It then sends a query to these tools, retrieves the necessary data, and integrates this information into its response back to the user. This approach significantly expands the chatbot's capabilities, allowing it to provide more current and precise answers by accessing a broader range of resources.

---

Now let’s take a closer look at AI chatbot for solving math problems

This slide illustrates the limitations of an AI chatbot when it operates without the integration of specialized math tools. The chatbot, powered by OpenAI GPT-3.5-Turbo API, Langchain, and Gradio, handles mathematical queries using its built-in capabilities. For example, it can calculate the result of 68 raised to the 0.36 power and determine the cosine of an angle of 15.78 degrees. While the chatbot provides approximate answers, these calculations highlight the inherent limitations in accuracy and complexity that arise without dedicated mathematical tools. This demonstrates the need for integrating external computational tools to enhance the chatbot's ability to handle more complex and precise mathematical operations.

---

These slides illustrate how to implement an AI chatbot with basic math tools. The first part of the code demonstrates the initialization process. It starts with defining the main function where the language model (GPT-3.5-Turbo) is initialized with conversational memory capabilities using ConversationBufferMemory. The math tool is set up with the Tool.from\_function method, allowing the chatbot to perform mathematical calculations.

In the second part, the tools, including the math tool, are defined and initialized with the agent. The agent is configured to handle parsing errors, apply early stopping methods, and use conversational memory to maintain context throughout interactions.

The third part of the code shows the predict function. This function ensures the agent is initialized if it is not already. It then runs the agent with the user's message to generate a response, utilizing the integrated math tool for any necessary calculations.

This implementation demonstrates how adding a basic math tool can enhance the capabilities of an AI chatbot, allowing it to handle more complex queries involving mathematical computations effectively.

---

This slide showcases a demonstration of an agent-based AI chatbot equipped with a basic math tool. Powered by OpenAI GPT-3.5-Turbo API, Langchain, and Gradio, this chatbot can handle more complex mathematical queries accurately. For instance, when asked to calculate the result of 68 raised to the 0.36 power, the chatbot provides a precise answer of 4.567759279639116. Similarly, when requested to calculate the cosine of an angle of 15.78 degrees, it returns the correct value of -0.9974064764544136. This enhanced capability is achieved by integrating a specialized math tool within the chatbot, demonstrating how such integrations can significantly improve the performance and accuracy of AI-driven interactions, particularly for mathematical computations.

---

This slide demonstrates the integration of customized math tools into an AI chatbot. The code example shows the creation of a CosineDegreeTool class, which is designed to calculate the cosine of an angle given in degrees. The tool converts the angle from degrees to radians before computing the cosine using Python's math library. This customized tool is then integrated into the AI chatbot alongside the basic math tool. In the main function, the set of tools for the agent includes both the math\_tool and the newly created CosineDegreeTool. This customization enhances the chatbot's capability to handle specific mathematical queries more accurately and efficiently, showcasing the flexibility of integrating specialized tools to meet particular requirements.

---

This slide demonstrates the enhanced capabilities of an agent-based AI chatbot equipped with customized math tools. Powered by OpenAI GPT-3.5-Turbo API, Langchain, and Gradio, this chatbot can accurately handle more complex mathematical queries. For example, when asked to calculate the result of 68 raised to the 0.36 power, the chatbot provides a precise answer of 4.567759279639116. Additionally, with the integration of a customized cosine calculator, the chatbot can accurately compute the cosine of an angle of 15.78 degrees, returning a value of 0.9623129786416633. This demonstration highlights how integrating specialized tools can significantly enhance the functionality and accuracy of AI chatbots, making them more versatile and capable of addressing a wider range of user queries.

---

Now let’s verify AI Chatbot capabilities with Search problems

This slide demonstrates the limitations of an AI chatbot when it lacks access to a search tool. The AI chatbot, powered by OpenAI GPT-3.5-Turbo API, Langchain, and Gradio, is unable to provide real-time information, as evidenced by its response to the query about the winner of the men's singles at the 2024 French Open. The chatbot acknowledges its limitation, directing the user to check the latest updates on the official French Open website or sports news websites for accurate information. This highlights the importance of integrating search tools into AI chatbots to enable them to retrieve up-to-date information and respond to real-time queries effectively.

---

This slide showcases the implementation of an AI Chatbot Agent equipped with search tools to address the limitation of not being able to access real-time information. The Python code on the left demonstrates how to initialize the chatbot using OpenAI's GPT-3.5-Turbo model with a search tool integrated. The search tool, TavilySearchResults, is configured to return a maximum of one result. The prompt template is designed to make the chatbot a helpful assistant while optionally using external tools. The right side of the slide shows the predict function, which ensures that the agent is initialized and able to invoke the search tool to fetch real-time data when needed. This integration allows the chatbot to provide up-to-date and relevant responses by leveraging external search capabilities, thereby enhancing its functionality and user experience

---

This slide presents an example of an AI Chatbot Agent equipped with search tools to overcome the limitation of not having access to real-time information. The interface demonstrates the chatbot's ability to retrieve up-to-date information about the winner of the 2024 French Open Men's singles. By leveraging the capabilities of OpenAI GPT-3.5-Turbo API, Langchain, Tavily, and Gradio, the chatbot successfully retrieves and displays the latest information. This integration allows the chatbot to enhance its responses with current and accurate data, thereby significantly improving the user experience by providing timely and relevant answers.

---

In summary, this presentation showcased how to leverage different frameworks to build LLM-based web application demos. On the backend, we utilized Python, LangChain, OpenAI API, FAISS, and various external tools like Wikipedia and Tavily. For the frontend, we employed Python and Gradio. Throughout the course of this presentation, we learned how to build four types of LLM-based applications: task-specific AI assistants, simple AI chatbots, RAG-based AI chatbots, and agent-based AI chatbots. These demonstrations highlighted the versatility and capabilities of LLMs in creating sophisticated AI applications.