A Generative AI Chatbot for Apple Website Using LLMs from Hugging Face

Team Name: AppleIQ
Prem Rakesh Alasakani - S565102
Sweshik Rao Nemarukommula - S565485
Rakesh Somanaboina - S566617
Varsha Pravallika Valavala - S556143
Krishna Vasanthi Yakkala - S566493

1 Tools, Technologies, and Implementation

1.1 Tools and Technologies

For the development of the generative AI chatbot for the Apple website, we plan to use the following tools and technologies:

- **Hugging Face Transformers Library:** This will provide access to a wide range of pre-trained language models (LLMs) such as GPT-based models or BERT, enabling natural language understanding and generation capabilities.
- **Python:** The main programming language for the implementation of the chatbot. Python is ideal for integrating machine learning models and building web-based applications.
- **Flask/Django:** Used as the web framework to deploy the chatbot on the Apple website and to handle user requests.
- **JavaScript (React.js):** For developing the front-end interface of the chatbot, ensuring a smooth user experience and real-time interaction.
- **REST APIs:** For integration of real-time product data from the Apple website such as product details, prices, and promotions.
- **Cloud Hosting (e.g., AWS, GCP, or Azure):** To deploy the application and ensure scalability and availability.
- **Database (e.g., PostgreSQL, MongoDB):** To store dynamic product data and customer interactions, ensuring the chatbot has real-time access to the necessary information.

- **NLP Tools:** Additional NLP libraries such as spaCy or NLTK for text preprocessing, entity recognition, and more.

1.2 High-Level Architecture and Methodology

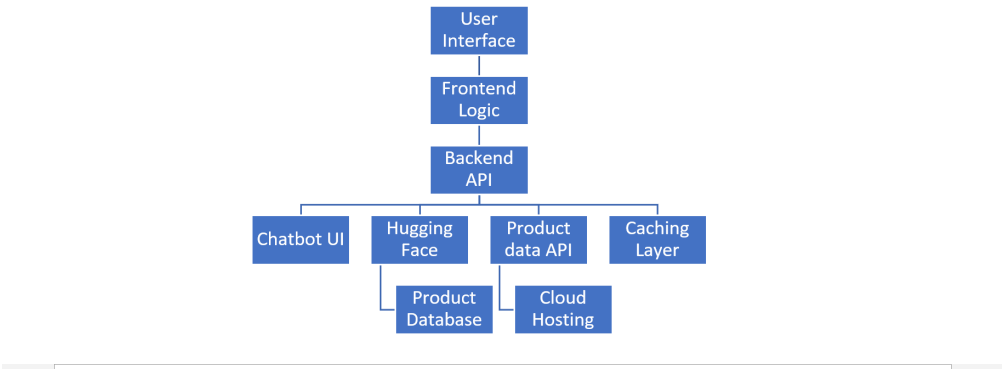

Figure 1: High-Level Architecture of the Chatbot System

1.3 Explanation of the Diagram

The high-level architecture of the chatbot system includes:

- **User Interface:** The chatbot interface is integrated into the Apple website using React.js, allowing customers to initiate conversations and ask questions about Apple products.
- **Frontend:** Handles all user inputs and sends queries to the backend via HTTP requests (REST API). Designed to be intuitive and responsive.
- **Backend (Flask/Django API):** Receives user queries and passes them to the Hugging Face model. Also handles requests for real-time product information from the database and third-party APIs.
- **Hugging Face LLM (e.g., GPT-4):** Processes the user's query, interprets the intent, and generates relevant responses while accessing real-time product data when required.
- **Product Data API:** Fetches up-to-date details on Apple products, including pricing, availability, and current promotions.
- **Database:** Stores and manages dynamic product information, customer interactions, and chatbot logs, ensuring accurate and updated data.
- **Cloud Hosting:** Hosts the entire system on the cloud, providing scalability, availability, and secure access.

1.4 Implementation Steps

The steps followed in implementing the chatbot:

1. **Setting Up Development Environment:** Install Python, Flask/Django, JavaScript (React.js), and necessary libraries for Hugging Face, databases, and other tools.
2. **Model Selection and Training:** Choose an appropriate language model from Hugging Face's pre-trained models (such as GPT-4 or BERT). Fine-tune the model on relevant datasets to ensure it can understand product-related queries.
3. **Backend Development:** Develop API endpoints using Flask/Django, integrate the Hugging Face model, and set up connections to the product data API.
4. **Frontend Development:** Build the user interface using React.js, with features such as input fields, buttons, and conversation history display.
5. **Testing and Optimization:** Conduct thorough testing of chatbot responses, accuracy of product information, and user experience.
6. **Deployment:** Host the application on the cloud, ensuring accessibility and scalability.

2 Enhanced Chatbot Development Details

2.1 Chatbot Description

The chatbot was developed as a generative AI application tailored for the Apple website for tasks such as customer assistance and troubleshooting. The chatbot uses LangChain, OpenAI models, and tools like Neo4j, Streamlit , and others, tailored for Apple's domain. The chatbot integrates libraries like LangChain, tiktoken, and yfiles jupyter graphs. It also involves tokenization, language modeling, and graph-based visualizations.

- **LangChain Framework:** Handles advanced workflows and integration with knowledge sources.
- **OpenAI Models:** Provides language understanding and generation.
- **Neo4j Integration:** Enables graph-based knowledge storage and retrieval.
- **Streamlit:** Serves as the user interface for interaction.
- **Additional Libraries:** Includes `tiktoken` and `yfiles_jupyter_graphs` for advanced processing and visualization.

2.2 Domain

Apple Inc.

2.3 Block Diagram

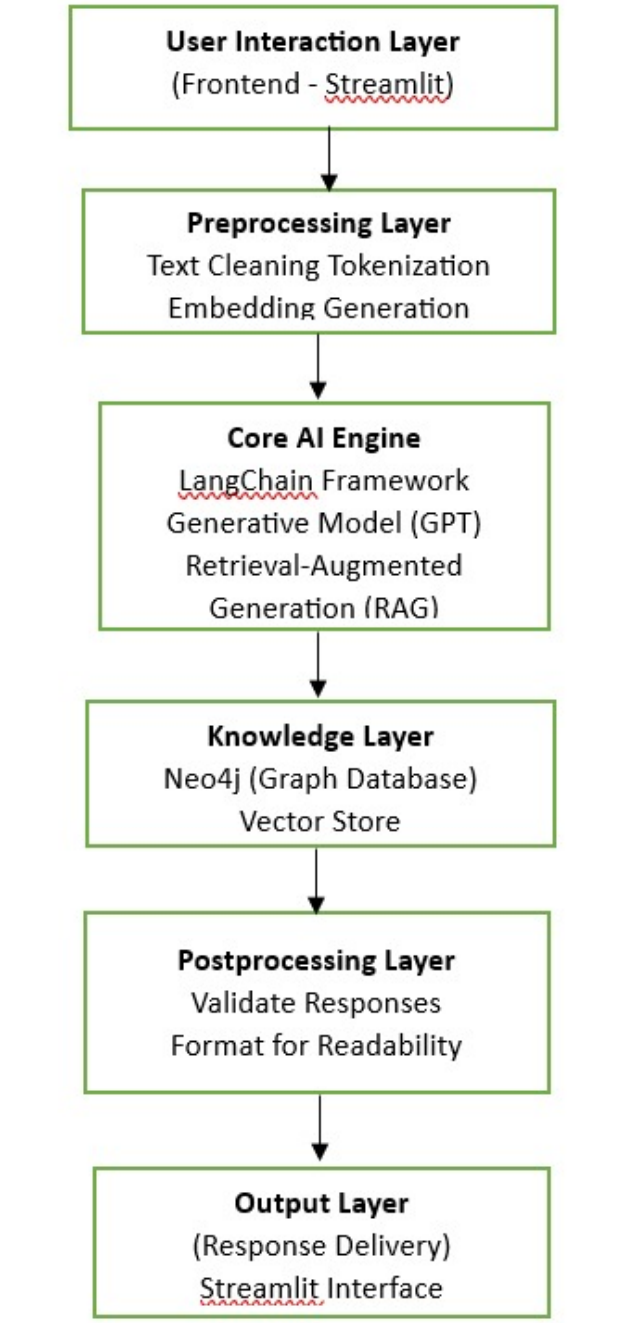

Figure 2: Chatbot Architecture Overview

2.4 Architecture

The architecture comprises:

- **Input Layer:** Captures user queries via the Streamlit interface.
- **Processing Layer:** Uses LangChain and OpenAI's models to generate responses.
- **Knowledge Layer:** Employs Neo4j for graph-based storage and retrieval of domain-specific data.
- **Output Layer:** Delivers responses back to users through the Streamlit interface.

2.5 Code Snippet

A sample code snippet demonstrating data loading and Neo4j integration:

```
from langchain_community.document_loaders import WebBaseLoader

loader = WebBaseLoader([
    "https://www.apple.com",
    "https://www.apple.com/store",
    "https://www.apple.com/mac/",
    "https://www.apple.com/ipad/",
    "https://www.apple.com/iphone/",
    "https://www.apple.com/watch/",
    "https://www.apple.com/apple-vision-pro/",
    "https://www.apple.com/airpods/",
    "https://www.apple.com/tv-home/",
    "https://www.apple.com/shop/accessories/all"
])

loader.requests_kwargs = {'verify': False}
data = loader.load()
# print(data)


llm = ChatOpenAI(temperature=0, model_name="gpt-3.5-turbo-0125")# gpt-4-0125-preview occ
llm_transformer = LLMGraphTransformer(llm=llm)


# Following needs to be used only first time to load the data. It took 3m 44s
graph_documents = llm_transformer.convert_to_graph_documents(documents)
graph.add_graph_documents(graph_documents, baseEntityLabel=True, include_source=True)
```

```
# directly show the graph resulting from the given Cypher query
default_cypher = "MATCH (s)-[r:!MENTIONS]->(t) RETURN s,r,t LIMIT 50"

def showGraph(cypher: str = default_cypher):
    # create a neo4j session to run queries
    driver = GraphDatabase.driver(
        uri = os.environ["NEO4J_URI"],
        auth = (os.environ["NEO4J_USERNAME"],
                os.environ["NEO4J_PASSWORD"]))
    session = driver.session()
    widget = GraphWidget(graph = session.run(cypher).graph())
    widget.node_label_mapping = 'id'
    #display(widget)
    return widget

showGraph()
```

2.6 Tests and Results

The chatbot achieved an accuracy of approximately 80%. Most queries were handled successfully, but a few errors were observed due to domain-specific complexities.

2.7 Improvements

Future enhancements will include:

- Personalization by learning user preferences.
- Enhanced knowledge retrieval using graph-based searches.
- Multimodal input support for richer interaction(e.g., text and images)..

2.8 References

- Apple Official Website
- Apple Store
- Apple Mac
- Apple iPad
- Apple iPhone
- Apple Watch
- Apple Vision Pro

- Apple AirPods
- Apple TV and Home
- Apple Accessories

2.9 GitHub Repository

- GitHub Repository: AppleIQ AI Chatbot