

# DS314 Introduction to Generative AI

PORTFOLIO 1 - Description

# Implementing a Real-Time Application with LangChain and Hugging Face

**Introduction** In this session, we will create and deploy a real-time working application using LangChain and Hugging Face with a proper user interface (UI). This step-by-step guide is tailored for beginners, demonstrating how to set up the environment, edit code, and deploy the application to production.

## Setting Up the Environment

## Using Hugging Face for Deployment

- 1. Introduction to Hugging Face Spaces:
  - Purpose: Hugging Face Spaces allow you to create, share, and deploy machine learning applications.
  - Components: Use pre-built APIs and modules available on the platform to build your applications.
- 2. Creating an Account:
  - API Keys: Generate access tokens for API authentication.
  - Access: Ensure you have your API key ready for use.

## **Building Your Application**

## Step-by-Step Guide

#### 1. Accessing Hugging Face Spaces:

- Navigate to the Hugging Face website and log in.
- Go to the "Spaces" tab to create your application.

#### 2. Creating a New Space:

- Create New Space: Click on the "Create new space" option.
- Naming: Name your space (e.g., LLMsIntro).
- Framework Selection: Choose Streamlit as the framework. Streamlit is an open-source framework that allows rapid development of applications.
- Resource Allocation: Select the free tier option (CPU only, vCPU, and 16 GB RAM).

## 3. Setting Up the Environment:

- Files and Dependencies:
  - requirements.txt: Create this file to list all Python dependencies. For example:

langchain streamlit openai

- .env-sample: Create this file to store environment variables, excluding sensitive information.
 Use a key-value pair format.

#### 4. Adding Required Files:

- Create Files:
  - Go to the "Files" section in your space.
  - requirements.txt: List all necessary libraries and their versions.
  - .env-sample: Store API keys and other sensitive information in a secure manner.

## 5. Main Application File (app.py):

- Purpose: This file will contain the main code for your application.
- Initial Setup: Add a boilerplate code to start your Streamlit application.

**Example Boilerplate Code for app.py:** This app.py file is a boilerplate code for a Streamlit application that uses LangChain and OpenAI to create an interactive AI-powered interface. The application takes user input, processes it using a language model, and displays the output.

```
import streamlit as st
from langchain_openai import OpenAI
```

#### Importing Libraries

- **Streamlit:** An open-source framework for building web applications, particularly useful for creating interactive data visualizations and deploying machine learning models.
- OpenAI from LangChain: This library allows interaction with OpenAI's language models (LLMs). It provides a straightforward interface to invoke the models and get responses.

```
#import os
#os.environ["OPENAI_API_KEY"] = ""
```

#### Setting Environment Variables

• Environment Variables: Uncommenting these lines would set the OpenAI API key as an environment variable. When deploying on Hugging Face Spaces, you should use the Variables & Secrets setting to securely pass these values instead.

```
def load_answer(question):
    llm = OpenAI(model_name="gpt-3.5-turbo-instruct", temperature=0)
    answer = llm.invoke(question)
    return answer
```

#### Function to Return the Response

- Function Definition (load\_answer): This function takes a question as input.
- Creating an LLM Instance (llm): An instance of the OpenAI class is created with a specific model (gpt-3.5-turbo-instruct) and temperature set to 0 (temperature controls the randomness of the model's output).
- Invoking the Model: The invoke method of the 11m object sends the question to the model and stores the response.
- Returning the Answer: The function returns the generated response.

```
st.set_page_config(page_title="LangChain Demo", page_icon=":robot:")
st.header("LangChain Demo")
```

#### App UI Starts Here

- Setting Page Configuration:
  - page\_title: Sets the title of the web page.
  - page\_icon: Sets the icon displayed in the browser tab.
- **Header:** Sets a header for the application.

```
def get_text():
    input_text = st.text_input("You: ", key="input")
    return input_text

user_input = get_text()
```

#### Getting User Input

- Function Definition (get\_text): This function creates a text input box where the user can type their question.
- Text Input (st.text\_input): This Streamlit function creates an input field labeled "You:".
- User Input: The user input is captured and returned by the function.
- Calling get\_text: The user input is stored in the user\_input variable.

```
response = load_answer(user_input)
submit = st.button('Generate')
```

## Generating and Displaying the Response

- Calling load\_answer: The load\_answer function is called with user\_input and the response is stored in the response variable.
- Generate Button (st.button): A button labeled "Generate" is created. When clicked, it triggers the action.

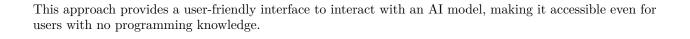
```
if submit:
    st.subheader("Answer:")
    st.write(response)
```

#### Handling Button Click and Displaying the Response

- Conditional Statement (if submit): Checks if the "Generate" button was clicked.
- Subheader (st.subheader): Adds a subheader titled "Answer:" to the application.
- Displaying Response (st.write): Displays the model's response below the subheader.

#### This Streamlit application:

- 1. Imports necessary libraries for building the UI and interacting with OpenAI's language model.
- 2. Defines a function to get responses from the language model.
- 3. Sets up the Streamlit page configuration and header.
- 4. Creates a text input field for user queries.
- 5. Generates a response from the language model when the "Generate" button is clicked.
- 6. Displays the generated response in the application interface.



## Uploading and Editing Code

## **Uploading Code**

## 1. Uploading Files:

- If you have already implemented the code locally, you can upload the file directly to your repository on Hugging Face.
- Upload Process: Click on "Add File," navigate to your file, select it, and commit the changes.

## Deploying and Running the Application

## 1. Committing Changes:

• Commit all changes to the repository to deploy the application.

## 2. Testing the Application:

- Navigate to the application URL provided by Hugging Face.
- Test the application by entering prompts and checking the responses generated by the LLM.

#### **Advanced Customizations**

#### 1. Enhancing the UI:

• Use Streamlit's components to add more features such as buttons, sliders, and charts.

#### 2. Integrating Additional Models:

• Expand the functionality by integrating other models or APIs as needed.

## 3. Handling Errors:

• Implement error handling to manage invalid inputs or API errors gracefully.

```
if user_input:
    try:
        response = load_answer(user_input)
        if st.button("Generate"):
            st.subheader("Answer")
            st.write(response)
    except Exception as e:
        st.error(f"An error occurred: {e}")
```

## Example Error Handling: