TransLingua: AI-Powered Multi-Language Translator

Team ID : LTVIP2026TMIDS41813

Team Size : 3

Team Leader : Aripalli Varsha

Team member : Sravanthi Bejjiparapu

Team member : Yekkaladevi Surendra

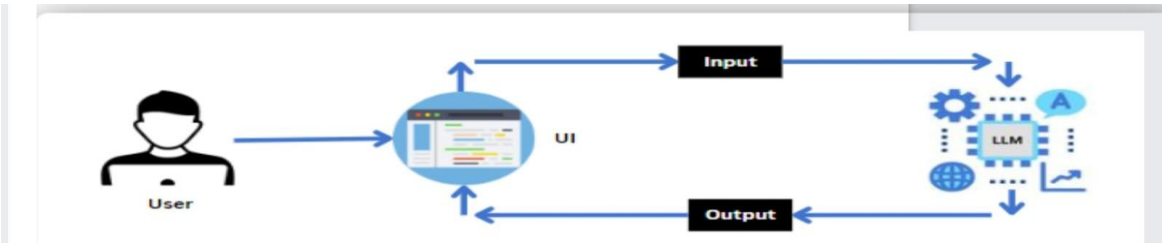
Introduction

TransLingua: AI-Powered Multi-Language Translator is an intelligent translation system designed to break language barriers by enabling fast, accurate, and seamless communication across multiple languages. Using advanced artificial intelligence and natural language processing techniques, TransLingua translates text efficiently while preserving the original meaning and context. It aims to support global communication in education, business, and everyday interactions by making language translation simple and accessible for everyone.

Description

TransLingua: AI-Powered Multi-Language Translator is a smart translation application that uses artificial intelligence to convert text from one language to another with high accuracy. It supports multiple languages and focuses on preserving meaning and context during translation. The system helps users communicate easily across language barriers, making it useful for education, travel, business, and global communication.

Architecture



Project Flow

- Users input the text they want to translate, select the source language, and choose the target language using the Streamlit UI.

• The input text and language selections are sent to the translation backend, which utilizes an AI-driven translation model to process the request.

• The AI model translates the text from the source language to the target language, providing an accurate and contextually relevant translation.

• The translated text is formatted and refined by the AI to ensure clarity and coherence in the target language.

• The translated text is sent back to the frontend of the Streamlit app for display to the user.

• Users can review the translated text, make additional modifications if needed, and use or save the translated content for their purposes.

To accomplish this, we have to complete all the activities listed below,

Initialize Gemini Pro LLM

- Generate Gemini Pro API
- Initialize the pre-trained model

Interfacing with Pre-trained Model

- Travel itinerary Generation

Model Deployment

- Deploy the application using Streamlit.

Prior Knowledge

You must have prior knowledge of the following topics to complete this project.

- **LLM & Gemini Pro**

A large language model is a type of artificial intelligence algorithm that applies neural network techniques with lots of parameters to process and understand human languages or text using self-supervised learning techniques. Tasks like text generation, machine translation, summary writing, image generation from texts, machine coding, chat-bots, or Conversational AI are applications of the Large Languag.e Model.

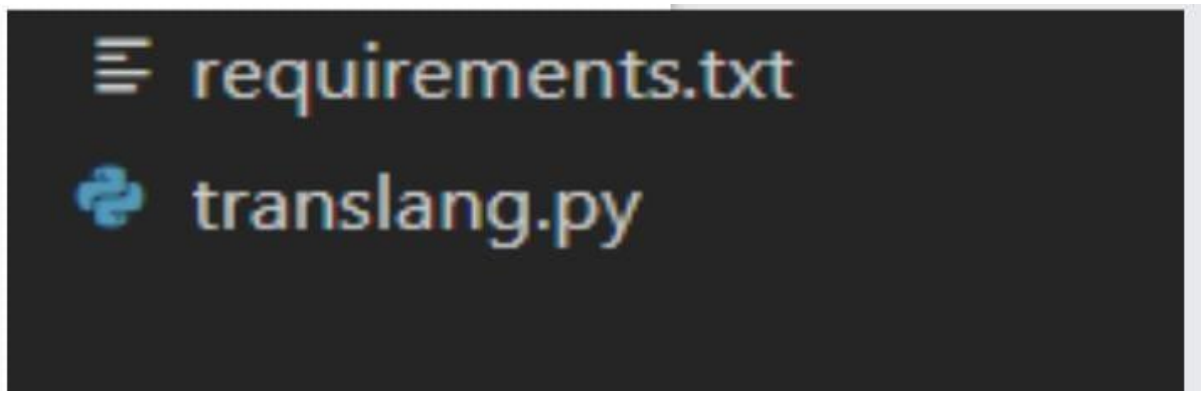
- **Streamlit**

Basic knowledge of building interactive web applications using Streamlit.

Understanding of Streamlit's UI components and how to integrate them with backend logic.

Project Structure:

Create the Project folder which contains application file as shown below



Requirement Specilization

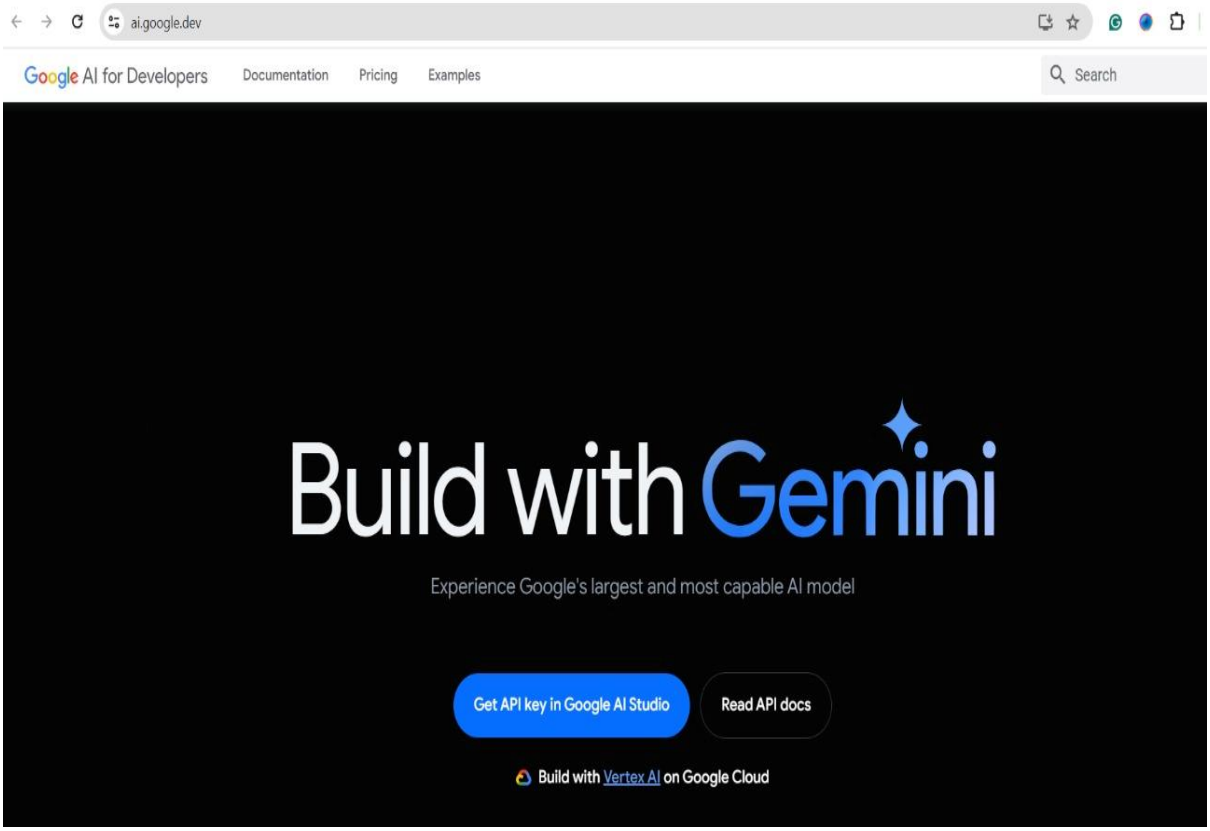
Specifying the required libraries in the requirements.txt file ensures seamless setup and reproducibility of the project environment, making it easier for others to replicate the development environment.

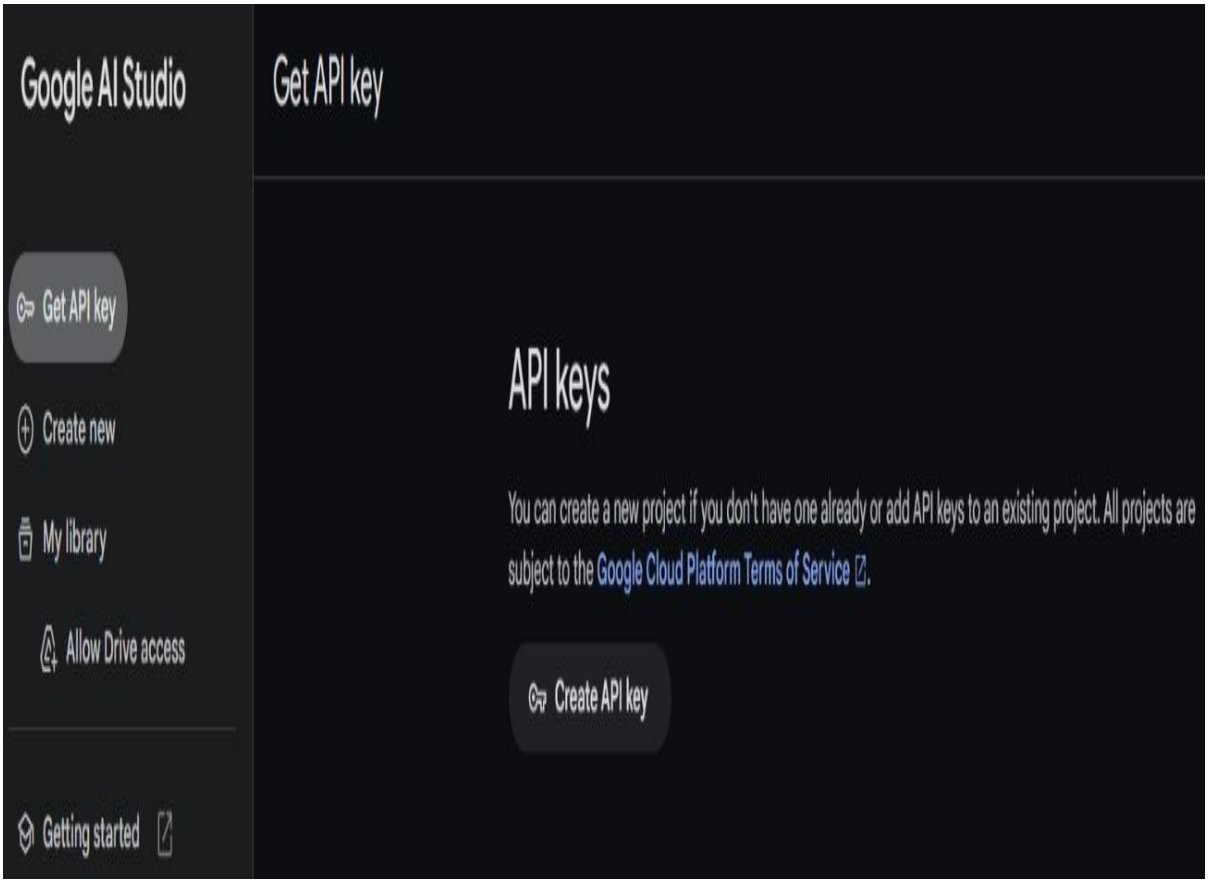
Initializing the Models

For initializing the model we need to generate PALM API.

Generate PALMAPI

Click on the link (https://developers.generativeai.google/). Then click on "Get API key in Google AI Studio". Click on "Get API key" from the right navigation menu. Now click on "Create API key".(Refer the below images) Copy the API key.


ai.google.dev
Google AI for Developers
Documentation
Pricing
Examples
Search
Build with Gemini
Experience Google's largest and most capable AI model
Get API key in Google AI Studio
Read API docs
Build with Vertex AI on Google Cloud


Google AI Studio
Get API key
Create new
My library
Allow Drive access
Getting started
Get API key
API keys
You can create a new project if you don't have one already or add API keys to an existing project. All projects are subject to the Google Cloud Platform Terms of Service.
Create API key

Initialize the pre-trained model

Import necessary files

```
from dotenv import load_dotenv # typ
import streamlit as st
import os
import google.generativeai as genai
```

- Streamlit, a popular Python library, is imported as st, enabling the creation of user interfaces directly within the Python script.
- Google Generative AI (genai): Imported to interact with the Gemini Pro model.
- The `load_dotenv` function loads key-value pairs from a `.env` file into environment variables.
- The `os` module is then used to access and manage these variables.

Configuration of the Gemini Pro API.

```
# Load environment variables
load_dotenv()

api_key = "AIzaSyB5U5-f1edVl99djSKEcqDoFLcI2l6uYyI"
genai.configure(api_key=api_key)
```

- Configuring the API key: The configure function is used to set up or configure the Google API with an API key. The provided API key, in this case, is

" AIzaSyB5U5-f1edVl99djSKEcqDoFLcXXXXXX".

Define the model to be used

```
model = genai.GenerativeModel('gemini-1.5-flash')
```

- Created an instance of GenerativeModel with the model_name set to "gemini-1.5-flash".

Interfacing with Pre-Trained Model

In this milestone, we will build a prompt template to generate feedback based on the project details entered by the user.

Create a function to generate travel guide.

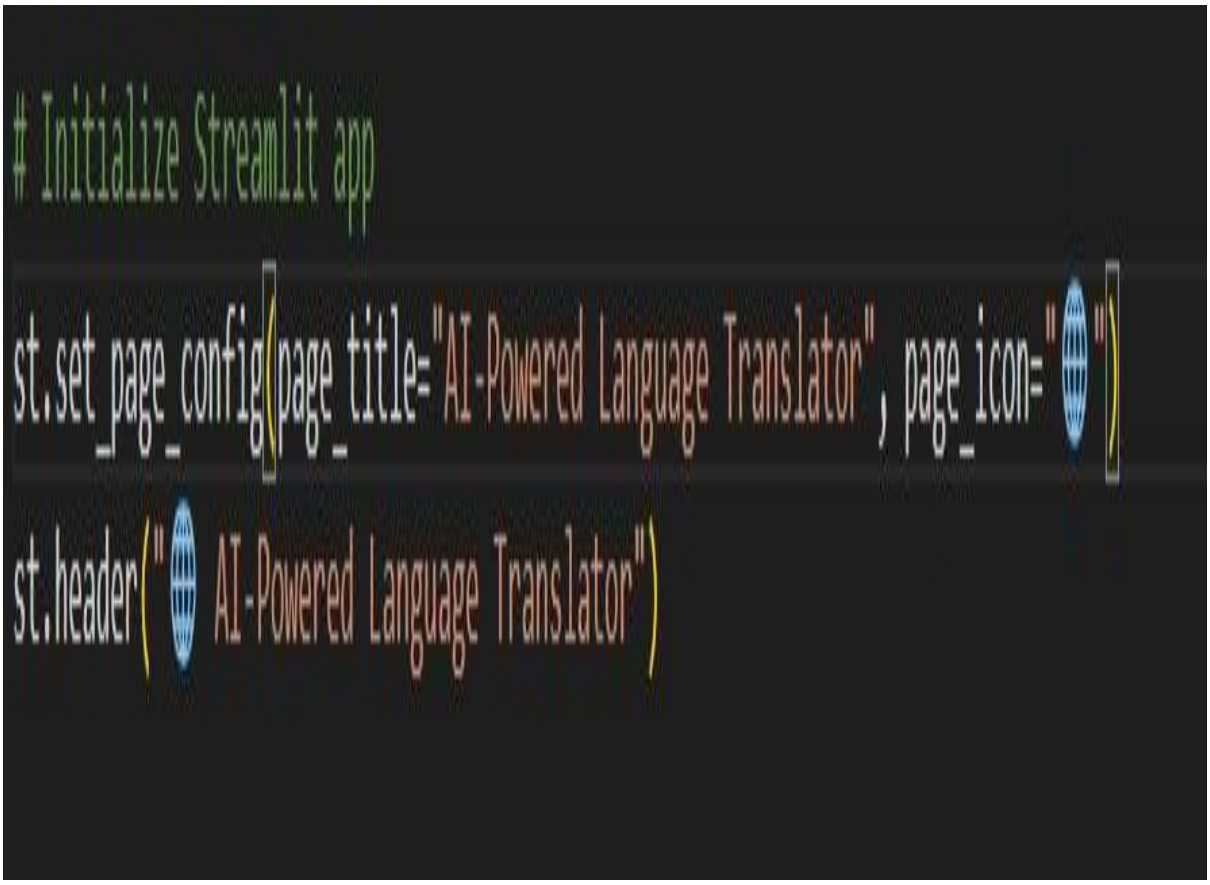
```
# Function to translate text
def translate_text(text, source_language, target_language):
    model = genai.GenerativeModel('gemini-1.5-flash')  # Replace with the correct model name
    prompt = (
        f"Translate the following text from {source_language} to {target_language}: {text}"
    )
    response = model.generate_content([prompt])
    return response.text
```

- def translate_text(text, source_language, target_language):: Defines a function translate_text that takes three parameters: text (the text to be translated), source_language (the language of the input text), and target_language (the language to which the text should be translated).
- model = genai.GenerativeModel('gemini-1.5-flash'): Initializes the generative model with the specified model name ('gemini-1.5-flash'). Replace with the appropriate model name if different.
- prompt = (f"Translate the following text from {source_language} to {target_language}: {text}"): Constructs a prompt string that instructs the model to translate the provided text from the source language to the target language.
- response = model.generate_content([prompt]): Sends the constructed prompt to the model to generate the translated content. The generate_content method processes the prompt and returns the model's response.

Model Deployment

In this milestone, we are deploying the created model using streamlit. Model deployment using Streamlit involves creating a user-friendly web interface, enabling users to interact with the model through a browser. Streamlit provides easy-to-use tools for developing and deploying data-driven applications, allowing for seamless integration of models into web-based applications.

Give the project title



- The main function to start the Streamlit application, allowing users to interact with the web app.
- st.set_page_config(page_title="AI-Powered Language Translator", page_icon="????"): Sets the app's page title and icon for the browser tab.
- st.header("???? AI-Powered Language Translator"): Displays a prominent header with the app's title, clarifying its purpose.

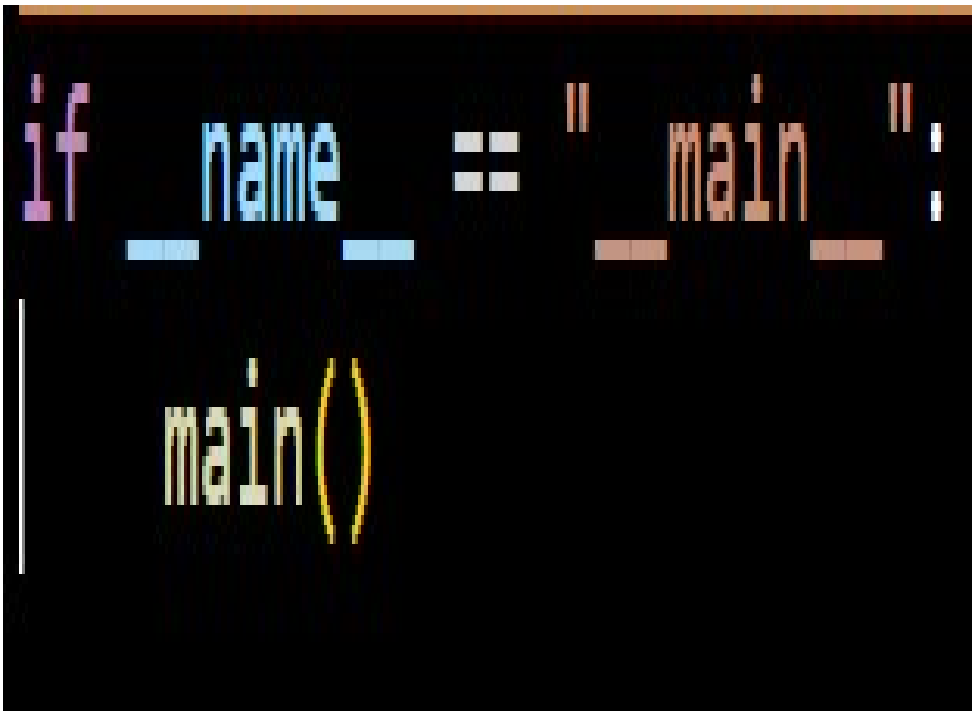
Create fields for user to input data for generating blog

```python
# User input for text, source language, and target language
text = st.text_area("📝 Enter text to translate:")
source_language = st.selectbox("🌍 Select source language:", ["English", "Spanish", "French", "German", "Chinese"])
target_language = st.selectbox("🌎 Select target language:", ["English", "Spanish", "French", "German", "Chinese"])

# Translate text button
if st.button("🔄 Translate"):
    if text and source_language and target_language:
        try:
            translated_text = translate_text(text, source_language, target_language)
            st.subheader("🔤 Translated Text:")
            st.write(translated_text)
        except Exception as e:
            st.error(f"⚠️ Error: {str(e)}")
    else:
        st.warning("⚠️ Please fill in all fields.")
```

- text = st.text_area("???? Enter text to translate:"): Creates a text input field labeled "???? Enter text to translate:" where users can type the text they want to translate. The default value is an empty string.
- source_language = st.selectbox("???? Select source language:", ["English", "Spanish", "French", "German", "Chinese"]): Provides a dropdown menu labeled "???? Select source language:" for users to choose the language of the input text.
- target_language = st.selectbox("???? Select target language:", ["English", "Spanish", "French", "German", "Chinese"]): Provides a dropdown menu labeled "???? Select target language:" for users to select the language into which the text should be translated.
- if st.button("???? Translate"):: Adds a button labeled "???? Translate." When clicked, the app attempts to translate the input text based on the selected source and target languages.
- if text and source_language and target_language:: Checks if all required fields (text, source language, and target language) are filled out before proceeding with translation.
- translated_text = translate_text(text, source_language, target_language): Calls the translate_text function to get the translated text using the provided inputs.
- st.subheader("????? Translated Text:"): Displays a subheaderlabeled "????? Translated Text:" before showing the translated text.
- st.write(translated_text): Displays the translated text on the web page.

- else:: Executes if any input field is empty or missing.
- st.warning("?? Please fill in all fields."): Shows a warning message if any required field is empty, prompting users to complete all fields.

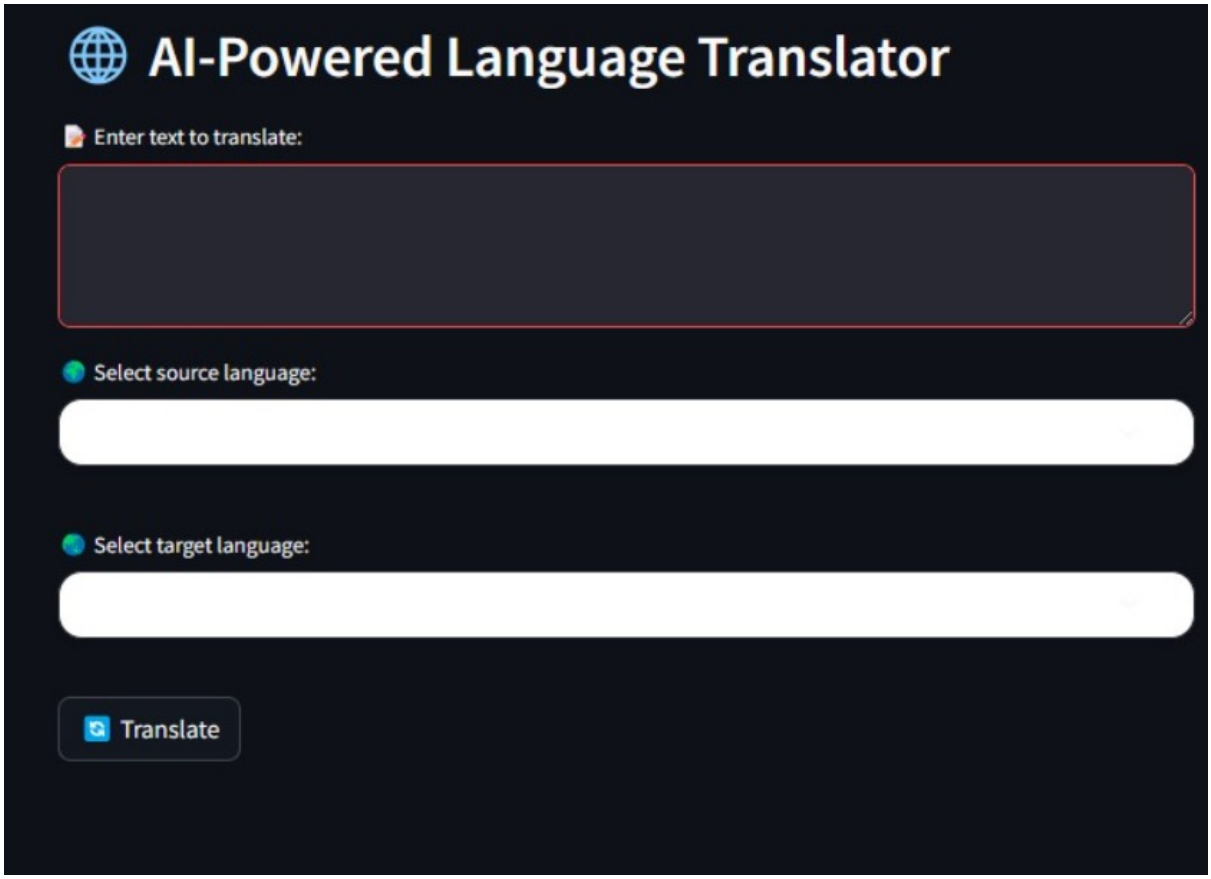


- Finally, the main() function is called to execute the Streamlit app.

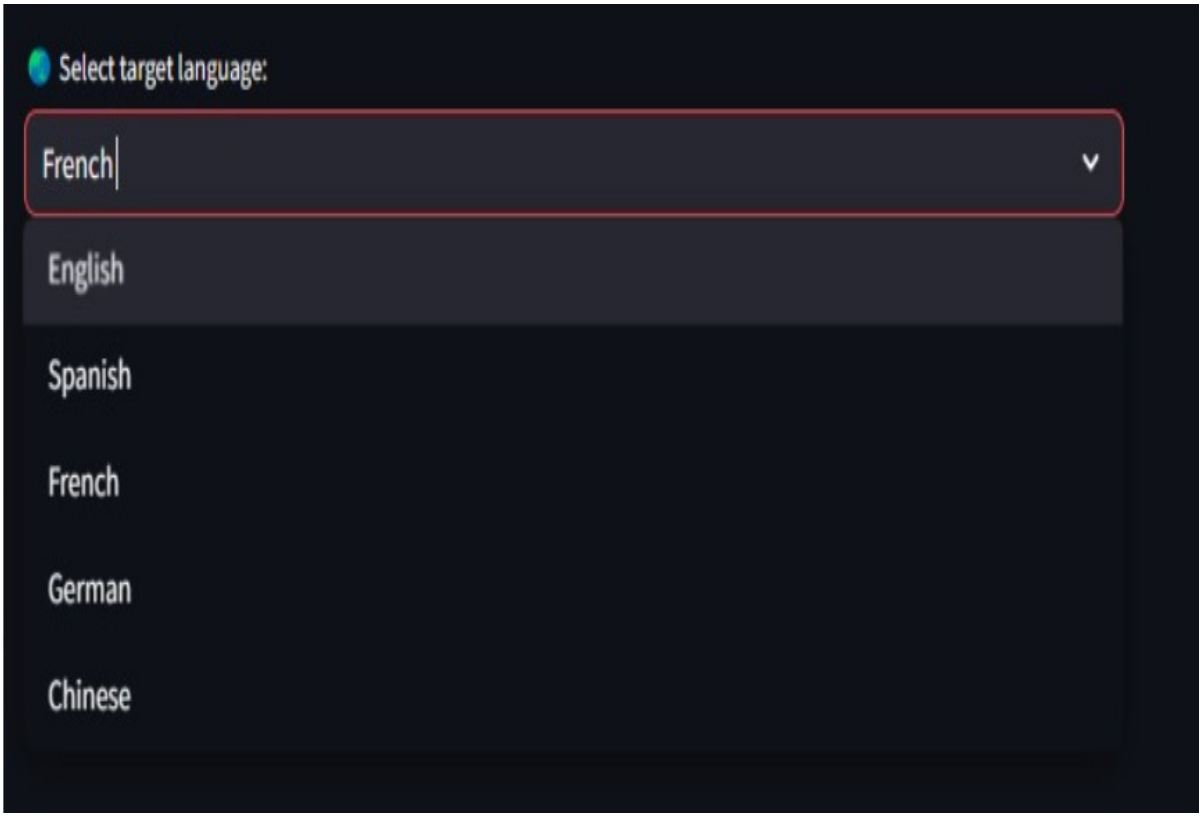
Running the web application

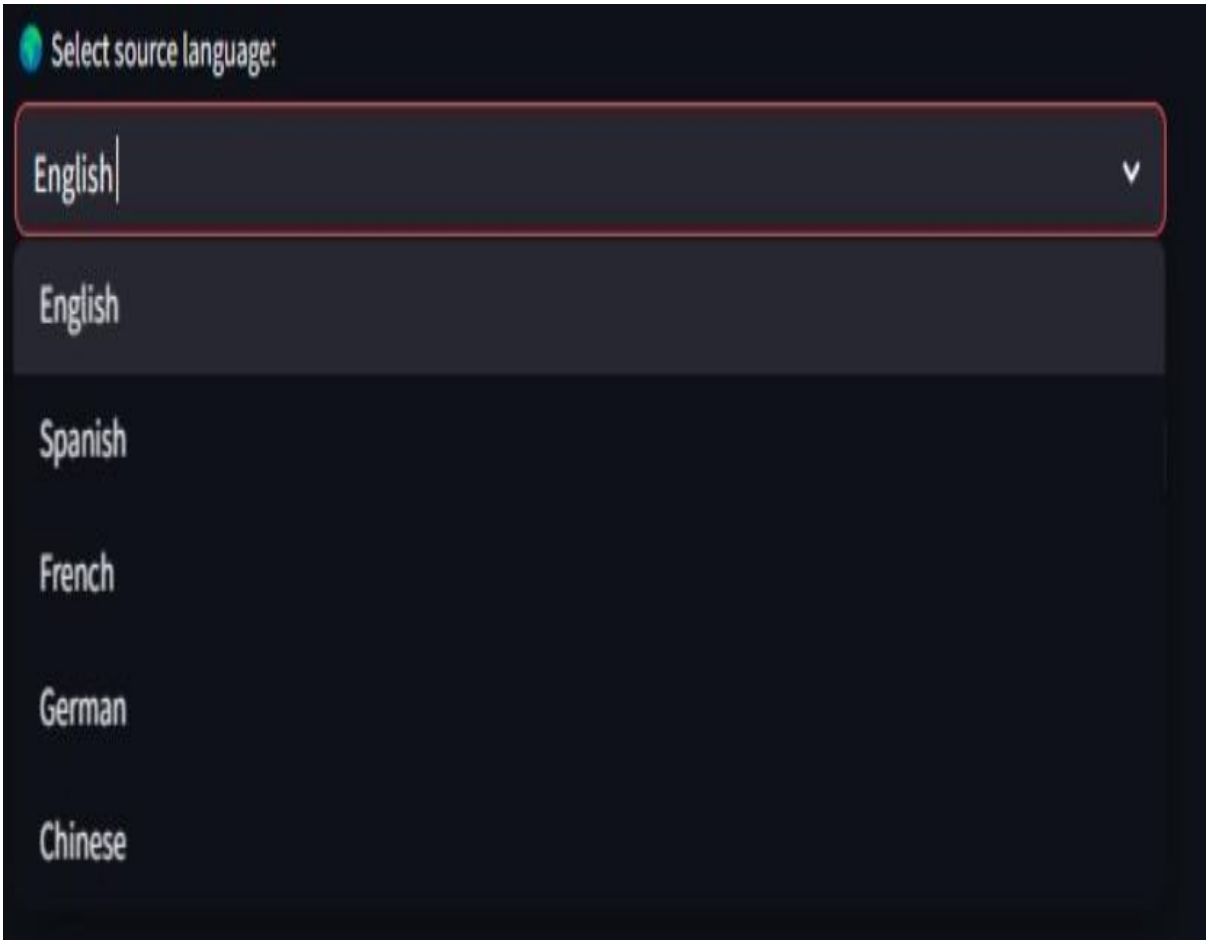
- Open the anaconda prompt from the start menu
- Navigate to the folder where your Python script is.
- Now type “streamlit run app.py” command
- Navigate to the localhost where you can view your web page.

Now, the application will open in the web browser,

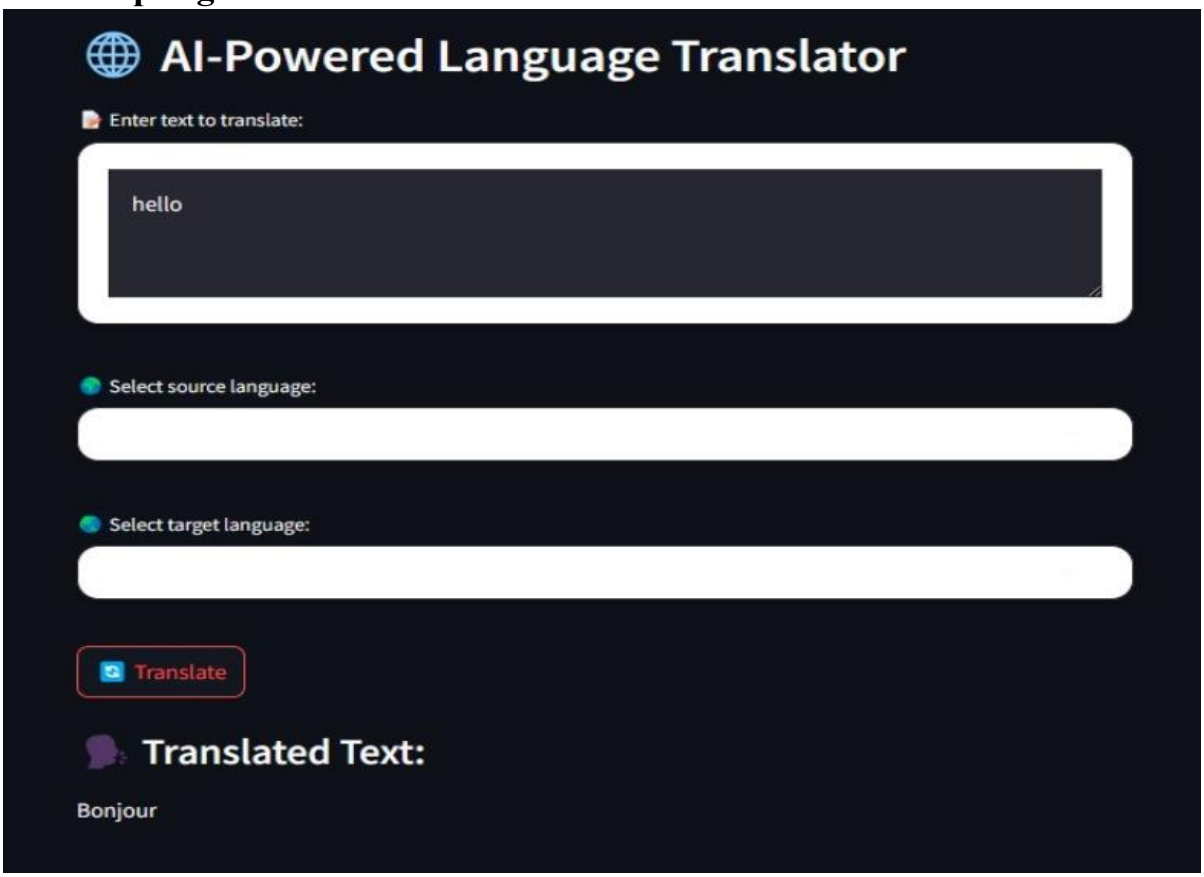


After giving the input


Select target language:
French
English
Spanish
French
German
Chinese


Select source language:
English
English
Spanish
French
German
Chinese

The Output generate



Conculsion

The Language Translator project is a user-friendly web application that leverages advanced AI technology to facilitate seamless language translation. By integrating Streamlit with a robust translation API, users can effortlessly translate text between different languages by specifying the source and target languages. The application provides real-time input validation and instant translation results, ensuring an efficient and interactive experience. This project demonstrates the practical application of AI in bridging language barriers, offering an accessible and effective tool for accurate and personalized translation needs.