Python for Data Science

Final Assignment: AI Engineer Assignment

Group:

- Nguyễn Minh Đạt - 22280009
- Nguyễn Xuân Việt Đức - 22280012
- Lê Đức Hòa - 22280027

In [ ]:
```
!pip install selenium
!apt-get update
!apt install chromium-chromedriver
!cp /usr/lib/chromium-browser/chromedriver /usr/bin
!pip install -U langchain-community
!pip install langchain openai faiss-cpu
!pip install google-generativeai
```

In [19]:
```
import os
import json
import textwrap
from typing import List, Optional
from selenium import webdriver
from selenium.webdriver.chrome.options import Options
from selenium.webdriver.common.by import By
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain.vectorstores import FAISS
from langchain.embeddings.base import Embeddings
import google.generativeai as genai
from google.colab import userdata
```

Question 1: LLM integration (Score: 30%)

The task involves building an AI capable of language translation.

1.1 Single Text Translation: (Score: 15%)

You are asked to write a Python code using the OpenAI API to translate a given text into Vietnamese (You should check the text if it's already the destination language . For example, translating "Hello" into Vietnamese should return "Xin chào", but "Xin chào" sho ld return the same.

In [21]:
```python
def translate_to_vietnamese(text: str) -> Optional[str]:
    try:
        # Get API key from environment variable
        api_key= os.getenv('GEMINI_API_KEY')
        if not api_key:
            raise ValueError("Missing API key. Set GEMINI_API_KEY environment variable")

        # Configure the API
        genai.configure(api_key=api_key)
        model = genai.GenerativeModel("gemini-1.5-flash")

        # Create the prompt
        prompt = "Translate this text to Vietnamese. If it's already in Vietnamese, return it unchanged.."

        # Generate the translation
        response = model.generate_content(
        prompt + text,
        generation_config={
                "temperature": 0,  # creative
                "top_p": 0         # nucleus sampling.
            }
        )

        return response.text

    except Exception as e:
        print(f"Translation error: {str(e)}")
        return None

if __name__ == "__main__":
    while True:
        text = input("Enter text to translate (or type 'quit' to exit): ").strip()
```

```
        if text.lower() == "quit":
            print("Exiting program.")
            break

        result = translate_to_vietnamese(text)
        if result:
            print("Result:", result)
        else:
            print("Translation failed.")
```

```
Enter text to translate (or type 'quit' to exit): Xin chào
Result: Xin chào

Enter text to translate (or type 'quit' to exit): Hello
Result: Xin chào

Enter text to translate (or type 'quit' to exit): quit
Exiting program.
```

1.2 Multiple Texts Translation: (Score: 15%)

Similar to 2.1, but the input is a list of texts. The Python code should accept a list of strings and return their translations in the specified language. For instance, translating ["Hello", "I am John", "Tôi là sinh viên"] into Vietnamese should return ["Xin chào", "Tôi tên là John", "Tôi là sinh viên"].

In [18]:
```
def translate_texts_to_vietnamese(texts: List[str], api_key: str) -> List[str]:
  translations = []
  for text in texts:
    try:
        # Get API key from environment variable
        api_key= os.getenv('GEMINI_API_KEY')
        if not api_key:
            raise ValueError("Missing API key. Set GEMINI_API_KEY environment variable")

        # Configure the API
        genai.configure(api_key=api_key)
        model = genai.GenerativeModel("gemini-1.5-flash")

        # Create the prompt
```

```
        prompt = "Translate this text to Vietnamese. If it's already in Vietnamese, return it unchanged: "

        # Generate the translation
        response = model.generate_content(prompt + text)
        translated_text = response.text.strip()

        # Add to results
        translations.append(translated_text)

    except Exception as e:
        print(f"Translation error: {str(e)}")
  return translations

if __name__ == "__main__":
    # Your API key
    api_key = os.getenv('GEMINI_API_KEY')
    # Example texts
    sample_texts = ["Hello", "I am John", "Tôi là sinh viên"]

    # Get translations
    translated_texts = translate_texts_to_vietnamese(sample_texts, api_key)

    # Print results
    print (translated_texts)
    for original, translated in zip(sample_texts, translated_texts):
        print(f"Original: {original}")
        print(f"Translated: {translated}")
        print("-" * 30)
```

```
['Xin chào', 'Tôi là John', 'Tôi là sinh viên']
Original: Hello
Translated: Xin chào
------------------------------
Original: I am John
Translated: Tôi là John
------------------------------
Original: Tôi là sinh viên
Translated: Tôi là sinh viên
------------------------------
```

Question 2: Chatbot Development (Score: 70%)

Assignment Test: Chatbot Development from Website Data. The data is at [https://www.presight.io/privacy-policy.html]

2.1 Data Access and Indexing (Score: 40%)

Tasked with creating a chatbot, begin by web crawling the specified website to gather relevant data, then preprocess and structure this data into a searchable index, ready for query retrieval. (Short version: crawling then embedding data, you can use selenium or requests)

2.2 Chatbot Development (Score: 30%)

Develop a chatbot that employs natural language processing to comprehend user questions, searches the indexed data from 2.1 for the best match, and delivers the most accurate response drawn from the website's information. (Use any distance/similarity metrics to get the best match paragraph then feed to LLM to get answer)

In [16]:
```python
def crawl_data_with_selenium(url):
    chrome_options = Options()
    chrome_options.add_argument("--headless")
    chrome_options.add_argument("--no-sandbox")
    chrome_options.add_argument("--disable-dev-shm-usage")
    chrome_options.add_argument("--disable-gpu")
    chrome_options.add_argument("--user-agent=Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/94
    chrome_options.add_argument("webdriver.chrome.driver=/usr/lib/chromium-browser/chromedriver")
    driver = webdriver.Chrome(options=chrome_options)

    try:
        driver.get(url)
        h2_elements = driver.find_elements(By.TAG_NAME, "h2")
        result = []
        concatenated_text = ""

        # Fetch the entire content of the page
        body_text = driver.find_element(By.TAG_NAME, "body").text
        concatenated_text = body_text.strip()
```

```
        for h2 in h2_elements:
            h2_text = h2.text
            try:
                next_sibling = h2.find_element(By.XPATH, "following-sibling::*")
                sibling_text = next_sibling.text
            except:
                sibling_text = ""
            result.append({"h2": h2_text, "content": sibling_text})

        output = {
            "data": result,
            "concatenatedText": concatenated_text
        }

        # Save the output object to a JSON file
        output_file = "result.json"
        with open(output_file, "w", encoding="utf-8") as f:
            json.dump(output, f, ensure_ascii=False, indent=4)

        return output

    finally:
        driver.quit()

# Gemini Embeddings Class
class GeminiEmbeddings(Embeddings):
    def embed_documents(self, texts):
        embeddings = []
        for text in texts:
            result = genai.embed_content(
                model="models/text-embedding-004",
                content=text
            )
            embeddings.append(result['embedding'])
        return embeddings

    def embed_query(self, text):
        result = genai.embed_content(
            model="models/text-embedding-004",
            content=text
        )
```

```
        return result['embedding']

# Function to process and vectorize extracted data
def process_extracted_data(data):
    text_contents = []
    for item in data['data']:
        combined_text = f"Section: {item['h2']}\nContent: {item['content']}"
        text_contents.append(combined_text)

    # add concatenatedText to enrich the context
    text_contents.append(data['concatenatedText'])

    text_splitter = RecursiveCharacterTextSplitter(
        chunk_size=1000,
        chunk_overlap=100,
        separators=["\n\n", "\n", ". ", " ", ""]
    )
    chunks = text_splitter.split_text('\n'.join(text_contents))
    embeddings = GeminiEmbeddings()
    vectorstore = FAISS.from_texts(chunks, embeddings)

    return vectorstore

    return vectorstore

# Function to generate chatbot responses
def get_chatbot_response(query, relevant_text, model_name):
    prompt = f"""Based on the following privacy policy sections, please provide a clear and direct answer to the question. If

    Privacy Policy Sections:
    {relevant_text}

    Question: {query}

    Instructions:
    - Use only information from the provided sections
    - Be specific and cite the relevant sections when possible
    - If information is not found, say so clearly
    """

    model = genai.GenerativeModel(model_name)
```

```
    response = model.generate_content(prompt)
    return response.text

# Function to process a query
def process_query(vectorstore, query, model_name):
    k = 15  # add more text chunks to retrieve

    #similarity searching
    results = vectorstore.similarity_search(query, k=k)
    relevant_texts = [doc.page_content for doc in results]

    # add weights
    query_terms = query.lower().split()
    weighted_texts = []

    for text in relevant_texts:
        score = 0
        text_lower = text.lower()
        for term in query_terms:
            if term in text_lower:
                score += 1
        weighted_texts.append((text, score))

    # Sắp xếp theo trọng số và lấy top k kết quả
    weighted_texts.sort(key=lambda x: x[1], reverse=True)
    final_texts = [text for text, _ in weighted_texts[:k]]

    relevant_text = "\n\n".join(final_texts)
    answer = get_chatbot_response(query, relevant_text, model_name)

    return {
        "query": query,
        "relevant_texts": final_texts,
        "answer": answer
    }


# Function to print wrapped text with separator
def print_wrapped_text_with_separator(text, width=150):
    wrapper = textwrap.TextWrapper(width=width)
    wrapped_text = wrapper.fill(text)
```

```
    print("\n" + wrapped_text + "\n" + "-" * width + "\n")

def main():
    os.environ['GEMINI_API_KEY'] = userdata.get('GEMINI_API_KEY')
    genai.configure(api_key=os.environ["GEMINI_API_KEY"])

    url = "https://www.presight.io/privacy-policy.html"
    vectorstore_path = "vectorstore_faiss_index"
    model_name = "gemini-1.5-flash"

    try:
        if os.path.exists(vectorstore_path):
            print("Loading existing vector store...")
            embeddings = GeminiEmbeddings()
            vectorstore = FAISS.load_local(vectorstore_path, embeddings, allow_dangerous_deserialization=True)
            print("Vector store loaded successfully!\n")
        else:
            print("Crawling and processing content...")
            data = crawl_data_with_selenium(url)
            vectorstore = process_extracted_data(data)
            vectorstore.save_local(vectorstore_path)
            print("Content processed successfully!\n")

        print("Chatbot is ready! Type 'quit' to stop.\n")
        while True:
            try:
                query = input("You: ").strip()
                if query.lower() == 'quit':
                    print("Thank you for using the chatbot. Goodbye!")
                    break

                if not query:
                    print("Please enter a valid question.")
                    continue

                results = process_query(vectorstore, query, model_name)
                if not results['relevant_texts']:
                    print("\nNo relevant information found. Please try rephrasing your question.")
                    continue

                # Print the answer with formatting and separator
```

```
                print_wrapped_text_with_separator(f"Answer: {results['answer']}\n", width=150)

            except Exception as e:
                print(f"Error processing query: {str(e)}")
                print("Please try another question.")

    except Exception as e:
        print(f"Critical error: {str(e)}")
        print("Please restart the application.")

if __name__ == "__main__":
    main()
```

```
Loading existing vector store...
Vector store loaded successfully!

Chatbot is ready! Type 'quit' to stop.

You: what is policy

Answer: The provided text describes Presight's privacy policy.  Key aspects include:  * **Information Collection and Use:** Pre
sight collects personal
data (email, name, phone, address, etc.) and usage data (IP address, browser type, etc.) to provide and improve its service (Se
ctions: Information
Collection and Use, Types of Data Collected, Use of Data).  They use automated edit checks to ensure data accuracy (Section: Au
tomated Edit Checks).
* **Data Security:**  Data is encrypted in transit and at rest; security audits are performed; employee access is restricted (S
ection: Data Security).
* **Data Retention and Disposal:** Data is retained while an account is active, then for 60 days after closure before removal
(Section: Data Retention
& Disposal).  * **Data Subject Responsibilities:** Users are responsible for providing accurate information and notifying Presi
ght of any inaccuracies
(Section: Quality, Including Data Subjects' Responsibilities for Quality).  * **Third-Party Sharing:**  Presight may disclose d
ata to service
providers (Datadog, AWS, Google Cloud, Google Workspace) and in response to legal requests (Section: Disclosure of Informatio
n).  However, they do
*not* share personal data with third parties for AI model development (Section: Sharing of Personal Data).  They also specify t
hat Google User Data is
not transferred to third-party AI tools for AI model development (Section: Google User Data and Google Workspace APIs).  * **Us
er Access to Data:**
Users can access, correct, amend, or append their personal information through the application (Section: Access to Personal Inf
ormation).  *
**Compliance:** Presight is committed to complying with data protection laws and cooperating with data protection authorities
(Section: Monitoring and
Enforcement).  * **Cookies:** Presight uses cookies to enhance website experience, controllable through browser settings (Secti
on: Cookies).  *
**Third-Party Websites:** Presight is not responsible for the privacy practices of linked third-party websites (Section: Third-
Party Websites).  *
**Policy Updates:** The policy may be updated from time to time (Section: Changes to Privacy Policy).  The last update was Sept
ember 15, 2023
(Section: PRIVACY POLICY).
------------------------------------------------------------------------------------------------------------------------
----------------------
```

You: how do i get in touch with you

Answer: You can contact Presight through the customer portal or by email at presight@presight.io. (Contact Us section)

You: when was the last update date

Answer: The last update date of the privacy policy is September 15, 2023. This is stated in the "PRIVACY POLICY" section.

You: what is a cookie policy

Answer: The provided text states, in the "Cookies" section, that "We use cookies to enhance your experience on our website. You can control the use of
cookies through your web browser settings." No further details on a specific cookie policy are available in the provided sections.

You: is there any risk of third-party software caused by cookies

Answer: The provided text mentions that cookies are used to enhance the website experience ("Cookies" section), and that users can control cookie use
through their browser settings. However, there is no information about the risk of third-party software caused by cookies.

You: can personal information be leaked

Answer: The provided text states that in the event of a data breach or unauthorized access to personal information, Presight will notify users and the
appropriate authorities as required by law (Monitoring and Enforcement section). However, it does not explicitly state that personal information
can be leaked, only that Presight has procedures in place to address breaches if they occur. The policy also describes security measures such as
data encryption (Data Security section).

```
You: quit
Thank you for using the chatbot. Goodbye!
```