**Báo cáo**

**Assignment 1- Chat PDF**

# TODO 1: (20 points)

* Hiện tại, hàm ask\_question nối tiếp câu hỏi question và nội dung trả lời content vào biến toàn cục chat\_history. Điều này sẽ giúp cho đoạn hội thoại có tính liên tục. Như ví dụ dịch nội dung trên, OpenAI chỉ dịch câu trả lời trước đó (về mạng NN) chứ không phải dịch tài liệu gốc. Tuy nhiên, hiện tại nó thiếu khả năng chỉnh sửa các cuộc trò chuyện trong quá khứ hoặc đặt lại cuộc trò chuyện. Cần phát triển tiếp các chức năng reset cuộc trò chuyện, delete hoặc edit các câu hỏi trước đó, tương tự như giao diện ChatGPT.
  + ask\_question("Translate the above to Vietnamese", reset\_chat=True) sẽ reset lịch sử chat
  + ask\_question("Translate the above to German", edit\_last\_question=True) sẽ chỉnh sửa tại chỗ câu hỏi cuối cùng trong chat\_history và lấy câu trả lời mới

## Thực hiện hàm reset lịch sử chat như sau :

-Ý tưởng: giử lại role system, và đưa question vào role user, cho chat\_history

-Hàm thực hiện reset như sau :

def ask\_question\_reset(question,reset\_chat=True):

    global chat\_history

    if reset\_chat:

      chat\_history=[]

      chat\_history.append({"role": "system", "content": f"""Bạn là một trợ lý truy xuất thông tin.

Thông tin văn bản như sau: {text3}

### nhiệm vụ của bạn là: Trả lời các câu hỏi chỉ sử dụng thông tin từ văn bản trên"""})

    chat\_history.append({"role": "user", "content": question})

    # Define parameters for the OpenAI ChatCompletion API

    params = dict(model="gpt-3.5-turbo", messages=chat\_history)

    # Generate a response from the AI model

    response = openai.ChatCompletion.create(\*\*params)

    content = response.choices[0].message.content

    chat\_history.append({"role": "assistant", "content": content})

    # Print and return the assistant's response

    print(content)

    print(response)

    return response

## Thực hiện edit lại câu hỏi cuối cùng :

* Ý tưởng : Xóa 2 phần tử cuối của chat\_history
* Hàm thực hiện :
* def ask\_question\_edit(question,edit\_last\_question=True):
* global chat\_history
* if edit\_last\_question and len(chat\_history) > 2:
* chat\_history=chat\_history[:-2]
* chat\_history.append({"role": "user", "content": question})
* # Define parameters for the OpenAI ChatCompletion API
* params = dict(model="gpt-3.5-turbo", messages=chat\_history)
* # Generate a response from the AI model
* response = openai.ChatCompletion.create(\*\*params)
* content = response.choices[0].message.content
* chat\_history.append({"role": "assistant", "content": content})
* return response

## Link Colabs:

<https://colab.research.google.com/drive/1RVBjMIFPVDY-yyZJ_WBZq_B_WGnUs9ko?usp=sharing>

# Chat với đoạn văn dài (Section 2)

## TODO 2: (20 points)

Hiện tại, mỗi khi người dùng đặt câu hỏi, ta sẽ thực hiện quy trình knowBase.similarity\_search để xác định trang có liên quan chặt chẽ nhất với câu hỏi xét về mức độ tương đồng nội dung. Điều này là cần thiết vì ta không thể chứa tất cả 100 trang trong giới hạn của OpenAI. Để nâng cao trải nghiệm người dùng, ứng dụng cho phép người dùng xác thực và tiếp tục cuộc trò chuyện của họ trong bối cảnh của cùng một trang, loại bỏ nhu cầu truy xuất tài liệu mới.

* Input: question
* Output: def answer(question, retrieve\_new\_page=False) - Cho phép người dùng tìm hiểu về trang hiện tại mà không cần chuyển đổi.

### Thực hiện hàm, tra cứu lại mà không cần thực hiện similarity\_search

* Ý tưởng : Lưu similarity\_search trước đó và sử dụng lại
* Hàm thực hiện :
* def getAnswerWithOldSimilar(self,docsSimilarity,question):
* fileProcess = FileProcess()
* kq={"docsSimilarity": [], "answer":""}
* kq["docsSimilarity"]=docsSimilarity
* kq["answer"]=fileProcess.getAnswer(kq["docsSimilarity"][:1],question)
* return kq

docsSimilarity : là kết quả của lần thực hiện similarity\_search trước đó

## TODO 3.2:

* def answer(question, retrieve\_diff\_page=True) để khám phá các trang khác nhau. =True) - Cho phép người dùng đặt câu hỏi liên quan đến trang tiếp theo.

### Thực hiện

Ý tưởng : kết quả similarity\_search trước đó cho ta 4 cái docs, cái đầu tiên đã sử dụng,bây giờ sử dụng 3 cái còn lại

## TODO 4 (10 points):

Output: def answer(question, retrieve\_external\_knowledge=True) - Cho phép người dùng tìm kiếm thông tin ngoài phạm vi của các file PDF hiện tại.

* Ý tưởng thực hiện : tạo ra cái docs rổng và đưa vào.