

BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP.HCM
KHOA CÔNG NGHỆ THÔNG TIN



**TIÊU LUẬN CHUYÊN NGÀNH
ĐỀ TÀI: TÌM HIỂU VỀ CÁC THƯ VIỆN HỖ
TRỢ XÂY DỰNG CHATBOT**

SINH VIÊN THỰC HIỆN:

Võ Nhu Ý **MSSV:20133118**

Nguyễn Quang Phúc **MSSV:20133080**

GIÁO VIÊN HƯỚNG DẪN: Ts. Trần Nhật Quang

TP. Hồ Chí Minh - 2023

PHIẾU NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN

Họ và tên Sinh viên 1 : Võ Nhu Ý

MSSV 1: 20133118

Họ và tên Sinh viên 2 : Nguyễn Quang Phúc

MSSV 2: 20133080

Ngành: Kỹ thuật dữ liệu

Tên đề tài: Tìm hiểu về các thư viện hỗ trợ xây dựng chatbot

Họ và tên Giáo viên hướng dẫn: TS. Trần Nhật Quang

NHẬN XÉT

Về nội dung đề tài & khối lượng thực hiện:

.....
.....
.....
.....
.....
.....
.....
.....

1. Ưu điểm:

.....
.....
.....
.....
.....

2. Khuyết điểm

.....
.....
.....
.....
.....

3. Đề nghị cho bảo vệ hay không ?.....
4. Đánh giá loại :
5. Điểm :

Tp. Hồ Chí Minh, ngày 22 tháng 12 năm 2023

Giáo viên hướng dẫn

(Ký & ghi rõ họ tên)

PHIẾU NHẬN XÉT CỦA GIÁO VIÊN PHẢN BIỆN

Họ và tên Sinh viên 1 : Võ Nhu Ý

MSSV 1: 20133118

Họ và tên Sinh viên 2 : Nguyễn Quang Phúc

MSSV 2: 20133080

Ngành: Kỹ thuật dữ liệu

Tên đề tài: Tìm hiểu về các thư viện hỗ trợ xây dựng chatbot

Họ và tên Giáo viên phản biện: TS. Nguyễn Thành Sơn

NHẬN XÉT

Về nội dung đề tài & khối lượng thực hiện:

.....
.....
.....
.....
.....
.....
.....
.....

1. Ưu điểm:

.....
.....
.....
.....
.....
.....
.....
.....

2. Khuyết điểm

.....
.....

.....
.....
.....
.....

3. Đề nghị cho bảo vệ hay không ?
4. Đánh giá loại :
5. Điểm :

Tp. Hồ Chí Minh, ngày tháng 12 năm 2023

Giáo viên phản biện

(Ký & ghi rõ họ tên)

MỤC LỤC

LỜI CẢM ƠN	1
DANH MỤC BẢNG BIỂU	2
DANH MỤC HÌNH ẢNH	3
DANH MỤC CÁC TỪ VIẾT TẮT	6
CHƯƠNG 1. PHẦN MỞ ĐẦU	7
1.1. Lý do chọn đề tài	7
1.2. Mục tiêu nghiên cứu	8
1.3. Đối tượng và phạm vi nghiên cứu	8
1.4. Bố cục	8
CHƯƠNG 2. CƠ SỞ LÝ THUYẾT VỀ CHATBOT	10
2.1. Chatbot	10
2.1.1. Chatbot là gì?	10
2.1.2. Các thành phần của chatbot	12
2.1.3. Một số ứng dụng Chatbot	13
2.2. Tinh chỉnh mô hình (model fine-tuning)	14
2.2.1. Khái niệm	14
2.2.2. Các mô hình ngôn ngữ hỗ trợ tinh chỉnh	14
2.2.2.1. Mô hình BERT	14
2.2.2.2. Mô hình PhoBERT	20
2.2.2.3. Mô hình XLM-RoBERTa	21
CHƯƠNG 3. THU VIỆN VÀ API HỖ TRỢ XÂY DỰNG CHATBOT	23
3.1. Thư viện	23
3.1.1. Thư viện PyTorch	23
3.1.1.1. Giới thiệu chung	23
3.1.1.2. Kiến trúc xây dựng Chatbot trong PyTorch	23
3.1.1.3. Tính năng	24
3.1.2. Thư viện Hugging Face Transformers	24
3.1.2.1. Giới thiệu chung	24

3.1.2.2. Cấu trúc chung của thư viện Hugging Face Transformers:	25
3.2. API	27
3.2.1. Khái niệm API	27
3.2.2. Các API hỗ trợ xây dựng chatbot	28
3.2.2.1. API của OpenAI	28
3.2.2.2. Google Cloud API	29
3.2.2.3. Rapid API	31
CHƯƠNG 4. CÀI ĐẶT THU VIỆN VÀ THỰC HIỆN TINH CHỈNH MÔ HÌNH CHATBOT	33
4.1. Tổng quan	33
4.2. Cài đặt thư viện và API	34
4.2.1. Transformers	34
4.2.2. OpenAI API	37
4.2.3. Rapid API	39
4.2.4. Google Cloud API	42
4.2.5. So sánh các API và thư viện hỗ trợ xây dựng Chatbot	51
4.3. Thực hiện tinh chỉnh mô hình Chatbot	53
4.3.1. Môi trường cài đặt	53
4.3.2. Chuẩn bị dữ liệu	54
4.3.2.1. Mô hình PhoBERT	54
4.3.2.2. Mô hình XLM-RoBERTa	54
4.3.2.3. Mô hình OpenAI	55
4.3.3 Xử lý dữ liệu đầu vào	56
4.3.3.1. Mô hình PhoBERT	56
4.3.3.2. Mô hình XLM-RoBERTa	57
4.3.4. Xây dựng mô hình	60
4.3.4.1. Mô hình PhoBERT	60
4.3.4.2. Mô hình XLM-RoBERTa	63

4.3.4.3. Mô hình OpenAI	66
4.3.5. Kết quả	69
4.3.5.1. Mô hình PhoBERT	69
4.3.5.2. Mô hình XLM-R	70
4.3.5.3. Mô hình OpenAI	71
CHƯƠNG 5. XÂY DỰNG ỨNG DỤNG DEMO	73
5.1. Phân tích, đặc tả yêu cầu	73
5.1.1. Mô tả chung	73
5.1.2. Yêu cầu chức năng	73
5.1.3. Yêu cầu giao diện phần mềm	73
5.1.4. Ràng buộc thiết kế	74
5.2. Kết quả xây dựng ứng dụng	74
5.2.1. Nền tảng phát triển	74
5.2.2. Các chức năng chính của ứng dụng	74
CHƯƠNG 6. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN	83
6.1. Kết luận	83
6.1.1. Tóm tắt kết quả đạt được	83
6.1.2. Ý nghĩa	83
6.1.3. Những hạn chế, giới hạn	84
6.2. Hướng phát triển	84
TÀI LIỆU THAM KHẢO	86
PHỤ LỤC	88
Phụ lục A. Hướng dẫn cài đặt ứng dụng	88
A.1. Cài đặt thư viện	88
A.2. Chuẩn bị môi trường trình duyệt	89
A.3. Chạy thử ứng dụng	89
Phụ lục B. Hướng dẫn sử dụng các chức năng của ứng dụng	91
B.1. Hướng dẫn sử dụng chức năng ChatBot	91

B.2. Hướng dẫn sử dụng chức năng Hộp thoại gợi ý	94
B.3. Hướng dẫn sử dụng chức năng Rapid API	97
B.4. Hướng dẫn sử dụng chức năng Google Bard	98
B.5. Hướng dẫn sử dụng chức năng ChatGPT use APIKey	98
B.6. Hướng dẫn sử dụng chức năng Model FineTune	100
B.7. Hướng dẫn sử dụng chức năng Hugging Face	103

LỜI CẢM ƠN

Chúng em xin chân thành cảm ơn Ban Giám hiệu trường Đại học Sư phạm Kỹ thuật Thành phố Hồ Chí Minh đã tạo cho chúng em môi trường tốt để chúng em có thể học tập và tiếp thu được những kiến thức quý báu trong những năm qua.

Chúng em xin gửi lời cảm ơn sâu sắc đến Thầy giáo - TS. Trần Nhật Quang đã đồng hành hướng dẫn và hỗ trợ chúng em trong suốt quá trình thực hiện đề tài. Những điều này đã giúp chúng em khắc phục được những hạn chế của bản thân và những khó khăn để hoàn thành tiểu luận thành công và đúng thời hạn.

Chúng em cũng gửi lời cảm ơn chân thành tới các thầy cô trong trường, đặc biệt các thầy cô trong Khoa Công nghệ thông tin đã giảng dạy chúng em trong suốt thời gian học tập tại trường.

Chúng em cũng xin được gửi lời cảm ơn sâu sắc tới gia đình, những người thân và bạn bè đã thường xuyên quan tâm, giúp đỡ, động viên chúng em trong suốt quá trình thực hiện tiểu luận chuyên ngành.

Cuối cùng, mặc dù chúng em đã rất cố gắng để hoàn thành đề tài trong khả năng của mình, nhưng chắc chắn sẽ không tránh khỏi những thiếu sót, kính mong nhận được sự thông cảm và chỉ bảo của các Thầy, Cô và các bạn để đề tài cũng như bản thân chúng em hoàn thiện hơn.

Nhóm sinh viên thực hiện

Võ Nhu Ý
Nguyễn Quang Phúc

DANH MỤC BẢNG BIỂU

Bảng 4. 1 Sử dụng mô hình GPT2 Vietnamese có nguồn gốc Hugging Face	35
Bảng 4. 2 Đoạn code sử dụng mô hình ViT5 trên Hugging Face.....	36
Bảng 4. 3 Đoạn code xây dựng tích hợp 2 API trong Rapid API.....	41
Bảng 4. 4 Đoạn code sử dụng Speech-to-Text trong Google Cloud	47
Bảng 4. 5 So sánh API và thư viện hỗ trợ xây dựng Chatbot	53
Bảng 4. 6 Xây dựng Class Features	60
Bảng 4. 7 Xây dựng class PhoBERT_finetuned	61
Bảng 4. 8 Kết quả train trong mô hình PhoBERT	70
Bảng 4. 9 Kết quả quá trình huấn luyện mô hình XLM-R	71
Bảng 4. 10 Kết quả quá trình tinh chỉnh trên OpenAI	72

DANH MỤC HÌNH ẢNH

Hình 2. 1 Các thành phần cơ bản của chatbot	12
Hình 2. 2 Sơ đồ kiến trúc BERT cho tác vụ Masked ML [5]	16
Hình 2. 3 Sơ đồ kiến trúc mô hình BERT cho tác vụ NSP [5]	18
Hình 2. 4 Toàn bộ tiến trình đào tạo trước và tinh chỉnh của BERT. [5]	19
Hình 3. 1 Các thành phần của thư viện Transformers Hugging Face ^[9]	26
Hình 4. 1 Hiển thị thông tin cài đặt thư viện transformers	34
Hình 4. 2 Kết quả khi sử dụng mô hình GPT2 Vietnamese	35
Hình 4. 3 Kết quả khi chạy mô hình ViT5	36
Hình 4. 4 Hình ảnh kiểm tra cài đặt thư viện OpenAI trong Python	37
Hình 4. 5 Tạo new secret key của OpenAI	37
Hình 4. 6 Hình ảnh sau khi tạo API key thành công	38
Hình 4. 7 Một API key của OpenAI được tạo ra thành công	38
Hình 4. 8 Sử dụng API key từ thư viện OpenAI	39
Hình 4. 9 Kết quả trả về khi sử dụng mô hình từ thư viện OpenAI	39
Hình 4. 10 Giao diện trang web Rapid API	39
Hình 4. 11 Các API hỗ trợ xây dựng chatbot trong Rapid API	40
Hình 4. 12 Hình ảnh một API chatbot có tên là Harley the Chatbot	40
Hình 4. 13 Hình ảnh giá thành và số lượng sử dụng của API Harley the Chatbot	40
Hình 4. 14 Giá thành API Microsoft Translator Text trên Rapid API	41
Hình 4. 15 Kết quả trả về từ đoạn code tích hợp 2 API	42
Hình 4. 16 Hình ảnh trang Google Cloud Platform	42
Hình 4. 17 Kết quả trả về từ ô tìm kiếm của Google Cloud	42
Hình 4. 18 Hình ảnh Cloud Speech-to-Text API trong Google Cloud	43
Hình 4. 19 Cửa sổ xuất hiện khi kích vào create credentials	43
Hình 4. 20 Hình ảnh thông tin của Service account details	43
Hình 4. 21 Nhập thông tin vào Service account details	44
Hình 4. 22 Bước tiếp theo khi chọn CREATE AND CONTINUE	44
Hình 4. 23 Hình ảnh chọn Role trong Service account	45
Hình 4. 24 Hình ảnh chi tiết Service Account được tạo thành công	45
Hình 4. 25 Hình ảnh bên trong một service account	45
Hình 4. 26 Hình ảnh key của Service account	46
Hình 4. 27 Chọn kiểu trả về Credentials của Service Account	46
Hình 4. 28 File Json được tải về	46
Hình 4. 29 Đoạn code sử dụng Text-to-Speech trong Google Cloud	47
Hình 4. 30 Trang giao diện chính của Google Bard	48
Hình 4. 31 Hình ảnh giao diện cửa sổ DevTool của Google Bard	48

Hình 4. 32 Lấy thông tin các thuộc tính Cookies của Google Bard	49
Hình 4. 33 Đoạn code sử dụng Google Bard trên Streamlit	49
Hình 4. 34 Kết quả khi chạy câu lệnh run Streamlit	50
Hình 4. 35 Giao diện Streamlit khi sử dụng Google Bard	50
Hình 4. 36 Kết quả khi sử dụng Bard API	51
Hình 4. 37 Cấu hình T4 GPU trên Google Colab	54
Hình 4. 38 Định dạng Json của tập dữ liệu huấn luyện trong PhoBERT	54
Hình 4. 39 Minh họa tập dữ liệu UIT-ViSquad	55
Hình 4. 40 Tập dữ liệu có định dạng jsonl được dùng huấn luyện trong OpenAI	56
Hình 4. 41 Xây dựng hàm xử lý tập training của PhoBERT	56
Hình 4. 42 Xây dựng hàm xử lý dữ liệu của tập validation test	57
Hình 4. 43 Xây dựng phương thức init trong mô hình XLM-R	57
Hình 4. 44 Phương thức call trong class Create_datasetDict	58
Hình 4. 45 Tải dữ liệu tập đánh giá và tập đào tạo trong PhoBERT	61
Hình 4. 46 Thiết lập các tham số, tập train_data và val_data trong mô hình PhoBERT	62
Hình 4. 47 Xây dựng mô hình PhoBERT và tối ưu hóa bằng cách dùng AdamW	62
Hình 4. 48 Xây dựng hàm train và evaluate trong PhoBERT	63
Hình 4. 49 Xây dựng hàm StartTrain để bắt đầu training	63
Hình 4. 50 Khởi tạo mô hình, tokenizer và thiết lập các tham số	64
Hình 4. 51 Xây dựng tập train dataset và valid dataset trong XLM-R	64
Hình 4. 52 Thiết lập model và args trong mô hình XLM-R	66
Hình 4. 53 Khởi tạo đối tượng trainer và tiến hành train	66
Hình 4. 54 Chế độ Fine-tuning trên OpenAI Platform	67
Hình 4. 55 Hình ảnh chọn tập dữ liệu và mô hình để finetune trong OpenAI	67
Hình 4. 56 Hình ảnh quá trình finetune trên OpenAI Platform	68
Hình 4. 57 Hình ảnh hoàn thành các quá trình finetune trên OpenAI Platform	69
Hình 5. 1 Giao diện đăng nhập của ứng dụng chatbot	75
Hình 5. 2 Giao diện chính khi đăng nhập thành công	76
Hình 5. 3 Hình ảnh giao diện chính ở chế độ Model Finetune	76
Hình 5. 4 Hình ảnh nút gửi tin nhắn bằng văn bản	77
Hình 5. 5 Hình ảnh nút nhấn chat bằng giọng nói	77
Hình 5. 6 Thanh tạo đoạn chat mới trong ứng dụng	77
Hình 5. 7 Thành phần cấu hình Chat	78
Hình 5. 8 Nút tắt âm thanh phản hồi	78
Hình 5. 9 Sử dụng chatbot bằng văn bản	79
Hình 5. 10 Sử dụng chatbot bằng giọng nói	79
Hình 5. 11 Kết quả khi sử dụng Google Bard	80

Hình 5. 12 Hình ảnh kết quả trả về khi sử dụng chatbot ở chế độ Model Finetune mô hình PhoBERT	80
Hình 5. 13 Nút dừng trong quá trình chờ câu hỏi phản hồi	81
Hình 5. 14 Nút tạo lại khi hoàn thành câu trả lời	81
Hình 5. 15 Hình ảnh các thao tác với đoạn chat	82
Hình A. 1 Hình ảnh chọn Open Folder trong Visual Studio Code	88
Hình A. 2 Hình ảnh câu lệnh chạy ứng dụng trong Terminal	89
Hình A. 3 Hình ảnh giao diện ứng dụng khi mở trên trình duyệt	90
Hình B. 1 Giao diện chính của ứng dụng khi đăng nhập thành công	91
Hình B. 2 Chọn chức năng ChatBot-OpenAPI khi sử dụng chế độ cấu hình ChatBot	92
Hình B. 3 Chọn hình thức sử dụng Use_API_Key trong loại mô hình ChatBot_openAPI	92
Hình B. 4 Chọn mô hình sử dụng trong Use_API_Key	92
Hình B. 5 Sử dụng Use_Breaber_Token trong chế độ ChatBot	93
Hình B. 6 Sử dụng text-davinci-003 ở chế độ ChatBot	94
Hình B. 7 Hình ảnh chatbot ở chế độ Hộp thoại gợi ý	95
Hình B. 8 Hình ảnh các chế độ sử dụng trong hộp thoại gợi ý	95
Hình B. 9 Hình ảnh lời nhắc khi sử dụng 3 HCMUTE.json	96
Hình B. 10 Kết quả trả về khi sử dụng lời nhắc trong Hộp thoại gợi ý	96
Hình B. 11 Hình ảnh chatbot khi sử dụng Rapid API	97
Hình B. 12 Hình ảnh sử dụng Chatbot ở chế độ Google Bard	98
Hình B. 13 Hình ảnh chatbot khi sử dụng ChatGPT use APIkey	99
Hình B. 14 Sử dụng mô hình text-davinci-003 trong ChatGPT use APIKey	100
Hình B. 15 Giao diện chatbot khi dùng Model FineTune	100
Hình B. 16 Giao diện khi sử dụng FineTune_XLM_Roberta	101
Hình B. 17 Hình ảnh các phiên bản tinh chỉnh cho mô hình PhoBERT	101
Hình B. 18 Lựa chọn kích thước của phiên bản FineTune sử dụng	102
Hình B. 19 Lựa chọn mô hình FineTune_OpenAI trong Model FineTune	102
Hình B. 20 Hình thức sử dụng trong FineTune_OpenAI	102
Hình B. 21 Hình ảnh model khi sử dụng hình thức gpt-3.5-turbo ở FineTune_OpenAI	103
Hình B. 22 Hình ảnh model khi sử dụng hình thức text-davinci-002 ở FineTune_OpenAI	103
Hình B. 23 Hình ảnh ứng dụng chatbot khi sử dụng Hugging Face	104
Hình B. 24 Hình ảnh các mô hình chat ở chế độ Hugging Face	104

DANH MỤC CÁC TỪ VIẾT TẮT

Ký hiệu từ viết tắt	Tên đầy đủ	Điễn giải
API	Application Programming Interface	Giao diện lập trình ứng dụng
TTS	Text To Speech	Chuyển đổi văn bản thành giọng nói
NLU	Natural Language Understanding	Hiểu ngôn ngữ tự nhiên
NER	Name Entity Recognition	Nhận dạng thực thể
NLG	Natural language generation	Bộ sinh ngôn ngữ tự nhiên
AI	Artificial Intelligence	Trí tuệ nhân tạo
cmd	Command Prompt	Dấu nhắc lệnh
NLP	Natural Language Processing	Xử lý ngôn ngữ tự nhiên
XLM-R	Cross-lingual Language Model- RoBERTa	Mô hình ngôn ngữ đa ngôn ngữ dựa trên mô hình RoBERTa
NN	Neural Network	Mạng thần kinh nhân tạo
CNN	Convolutional Neural Network	Mô hình mạng neural tích chập

CHƯƠNG 1. PHẦN MỞ ĐẦU

1.1. Lý do chọn đề tài

Trong bối cảnh cuộc cách mạng công nghiệp 4.0 đang phát triển mạnh mẽ, nhu cầu của xã hội đối với phần mềm thông minh ngày càng tăng cao. Trong thời gian gần đây, sự quan tâm đặc biệt được dành cho việc phát triển và triển khai chatbot, một hệ thống đàm thoại dựa trên trí tuệ nhân tạo (AI) có khả năng xử lý ngôn ngữ tự nhiên (NLP) và mạng thần kinh (NN). Các thuật toán tiên tiến được áp dụng để nâng cao khả năng thông minh và chính xác của chatbot.

Hiện nay, chatbot đã được ứng dụng trên rất nhiều lĩnh vực như:

- Giải trí: Người dùng có thể nói chuyện và tương tác với chúng mọi lúc mọi nơi, nó trả lời câu hỏi của bạn theo cách nhân văn nhất và có thể hiểu được tâm trạng của bạn với ngôn ngữ mà bạn đang sử dụng. Các chatbot giải trí trực tuyến như là: KuKi AI, Rose, Insomnia Bot,...
- Thời tiết: Được thiết kế như một chuyên gia dự báo thời tiết và cảnh báo thời tiết xâu đối với người dùng ví dụ như Chatbot Poncho.
- Y tế: Chatbot này sẽ hỏi về các triệu chứng, các thông số cơ bản và lịch sử y tế, sau đó biên soạn ra một danh sách các nguyên nhân gây ra cũng như các loại bệnh có thể mắc phải theo thứ tự nghiêm trọng.
- Khách sạn và du lịch: Đây là một loại chatbot khá phổ biến và được sử dụng một cách rộng rãi giúp tiết kiệm thời gian và giảm chi phí nhân lực. Chúng được lập trình để có thể trò chuyện cùng khách hàng và nhờ đó có thể biết được các mong muốn và yêu cầu của khách hàng một cách đơn giản hơn.

Với những lý do nêu trên, nhóm chúng em đã quyết định chọn đề tài “**Tìm hiểu về các thư viện hỗ trợ xây dựng chatbot**” cho bài Tiểu luận Chuyên ngành thông qua các thư viện Transformers, PyTorch, Keras và các API như OpenAI, Rapid API, Google Cloud API và thực hiện tinh chỉnh mô hình(model fine-tuning) ngôn ngữ tiếng Việt như PhoBERT, XLM-R.

1.2. Mục tiêu nghiên cứu

- Nghiên cứu lý thuyết, cách sử dụng các thư viện, API và áp dụng trong việc xây dựng chatbot.
- Nắm được cách tinh chỉnh mô hình ngôn ngữ tiếng Việt trong bài toán xây dựng chatbot như PhoBERT, BERT, XLM-R.
- Tìm hiểu ưu nhược điểm, tính năng của từng thư viện API trong việc xây dựng chatbot .
- Xây dựng ứng dụng demo Chatbot.

1.3. Đối tượng và phạm vi nghiên cứu

Đối tượng nghiên cứu: Các thư viện hỗ trợ xây dựng Chatbot, các API hỗ trợ xây dựng chatbot và cách tinh chỉnh mô hình ngôn ngữ tiếng Việt trong việc xây dựng chatbot.

Phạm vi nghiên cứu:

- Phạm vi nghiên cứu của đề tài giới hạn trong việc tìm hiểu, cài đặt và so sánh các thư viện, API hỗ trợ xây dựng chatbot.
- Phạm vi nghiên cứu của đề tài được giới hạn trong ngôn ngữ lập trình Python.

1.4. Bộ cục

- Chương 1: Tổng quan: Sơ lược về đề tài, mục tiêu, đối tượng và phạm vi nghiên cứu, bộ cục cuốn tiểu luận.
- Chương 2: Cơ sở lý thuyết về Chatbot.
 - Giới thiệu chatbot, các thành phần cấu tạo và phân loại chatbot.
 - Tinh chỉnh mô hình trong xây dựng chatbot.
- Chương 3: Thư viện và API hỗ trợ xây dựng Chatbot.
 - Lý thuyết về thư viện xây dựng Chatbot trong ngôn ngữ lập trình Python.
 - Lý thuyết về API hỗ trợ xây dựng chatbot .
- Chương 4: Cài đặt thư viện và thực hiện tinh chỉnh mô hình Chatbot.
 - Mô tả cách cài đặt thư viện, API.
 - So sánh các thư viện và API hỗ trợ xây dựng chatbot.

- Mô tả cách thực hiện tinh chỉnh mô hình gồm có các bước cài đặt môi trường, chuẩn bị dữ liệu, xây dựng mô hình và đánh giá kết quả.
- Chương 5: Xây dựng ứng dụng demo.
 - Phân tích đặc tả yêu cầu xây dựng chatbot như yêu cầu chức năng, yêu cầu giao diện.
 - Xây dựng ứng dụng demo chatbot trên nền tảng Streamlit.
 - Trình bày kết quả xây dựng ứng dụng.
- Chương 6: Kết luận và hướng phát triển: Trình bày kết luận đề tài và hướng phát triển về sau.
 - Kết luận đề tài.
 - Hướng phát triển.
- Phụ lục: Trình bày cách cài đặt và hướng dẫn sử dụng ứng dụng.
 - Cài đặt ứng dụng.
 - Hướng dẫn sử dụng.

CHƯƠNG 2. CƠ SỞ LÝ THUYẾT VỀ CHATBOT

2.1. Chatbot

2.1.1. Chatbot là gì?

Chatbot (còn được gọi là chatterbot) là một ứng dụng phần mềm có thể được sử dụng để thực hiện các cuộc trò chuyện trực tuyến thông qua văn bản hoặc chuyên văn bản thành giọng nói thay vì tiếp xúc trực tiếp với con người. Trợ lý ảo Chatbot được sử dụng ngày càng nhiều để xử lý các tác vụ tìm kiếm đơn giản trong cả môi trường doanh nghiệp với người tiêu dùng và doanh nghiệp với doanh nghiệp.

Chatbot có thể có nhiều mức độ phức tạp khác nhau, không trạng thái hoặc có trạng thái. Một chatbot không trạng thái tiếp cận mỗi cuộc trò chuyện như thể nó đang tương tác với một người dùng mới. Ngược lại, một chatbot trạng thái có thể xem xét các tương tác trong quá khứ và định khung các phản hồi mới theo ngữ cảnh. ^[1]

Chatbot có thể được phân chia thành các loại sau đây:

- Chatbot có kịch bản hoặc trả lời nhanh - Đây là những chatbot cơ bản nhất; chúng hoạt động như một cây quyết định phân cấp. Các bot này tương tác với người dùng thông qua một tập hợp các câu hỏi được xác định trước sẽ tiến triển cho đến khi chatbot trả lời câu hỏi của người dùng. Tương tự, chatbot dựa trên danh mục (menu) yêu cầu người dùng thực hiện các lựa chọn từ danh sách được xác định trước để cung cấp cho bot hiểu sâu hơn về những gì khách hàng đang tìm kiếm.
- Chatbot dựa trên nhận dạng từ khóa - Những chatbot này phức tạp hơn, chúng có gắng lắng nghe những gì người dùng nhập và phản hồi tương ứng bằng cách sử dụng các từ khóa chọn được từ phản hồi của khách hàng. Các từ khóa có thể tùy chỉnh và trí tuệ nhân tạo được kết hợp trong bot này để cung cấp phản hồi thích hợp cho người dùng. Những chatbot này thường gặp khó khăn khi phải đổi mặt với việc sử dụng từ khóa lặp đi lặp lại hoặc các câu hỏi thừa.
- Chatbot kết hợp - Những chatbot này kết hợp các yếu tố của bot dựa trên danh mục

và nhận dạng từ khóa. Người dùng có thể chọn để câu hỏi của họ được trả lời trực tiếp, nhưng cũng có thể truy cập danh mục của chatbot để thực hiện lựa chọn nếu quá trình nhận dạng từ khóa tạo ra kết quả không hiệu quả.

- Chatbot theo ngữ cảnh - Những chatbot này phức tạp hơn những chatbot được liệt kê ở trên và yêu cầu tập trung vào dữ liệu. Sử dụng học máy (machine learning) và trí tuệ nhân tạo để ghi nhớ các cuộc trò chuyện và tương tác với người dùng, sau đó sử dụng những ký ức này để phát triển và cải thiện theo thời gian. Thay vì dựa vào từ khóa, những bot này sử dụng những gì khách hàng yêu cầu và cách chúng yêu cầu để đưa ra câu trả lời và tự cải thiện.
- Chatbot hỗ trợ giọng nói - Loại chatbot này là tương lai của công nghệ chatbot. Các chatbot hỗ trợ giọng nói sử dụng đối thoại bằng giọng nói từ người dùng làm dữ liệu đầu vào. Chúng có thể được tạo ra bằng cách sử dụng bộ công cụ chuyển đổi văn bản thành giọng nói (Chuyển đổi văn bản thành giọng nói – Text To Speech (TTS)) và giao diện ứng dụng nhận dạng giọng nói (API). ^[1]

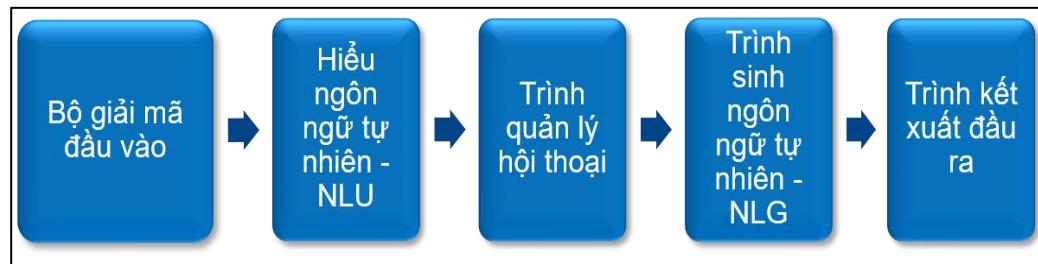
Ưu điểm của chatbot^[2]:

- Cung cấp dịch vụ khách hàng nhanh chóng:
 - Phần mềm này hỗ trợ doanh nghiệp cung cấp dịch vụ khách hàng 24 giờ/ngày, kể cả những ngày cuối tuần hay ngày lễ.
 - Khi khách hàng trực tuyến có thắc mắc, họ chỉ cần hỏi trong chatbot trên trang web mà không cần phải chờ đợi lâu để có câu trả lời. Bởi vì câu trả lời chỉ là một vài tổ hợp câu trả lời được lập trình sẵn.
- Làm tăng sự hài lòng của khách hàng:
 - Khi khách hàng nhận được câu trả lời thỏa đáng và nhanh chóng khi sử dụng chatbot, họ sẽ cảm thấy hài lòng hơn và tiếp tục mua sản phẩm.
- Giảm chi phí lao động:

- Chatbot giúp doanh nghiệp giữ chi phí kinh doanh thấp bởi vì số tiền bạn đầu tư vào chatbot ít hơn số tiền bạn phải trả cho nhân viên. Do đó, doanh nghiệp có thể tiết kiệm được rất nhiều tiền thay vì việc duy trì một trung tâm hỗ trợ khách hàng. Tính năng này sẽ giúp doanh nghiệp tiết kiệm tài chính, tránh những rắc rối trong quản lý nhân sự, và tiết kiệm được nhiều thời gian.
- Nhiều mục đích sử dụng:
 - Doanh nghiệp có thể sử dụng chatbot trong nhiều mảng, ví dụ như nhận đơn đặt hàng của khách, dịch vụ khách hàng và quảng cáo sản phẩm.

2.1.2. Các thành phần của chatbot

Một hệ thống chatbot bao gồm các thành phần sau: Bộ giải mã đầu vào (Input Decoder), hiểu ngôn ngữ tự nhiên (Natural Language Understanding - NLU), trình quản lý hội thoại (Dialogue Manager), sinh ngôn ngữ tự nhiên (Natural Language Generation - NLG) và trình kết xuất đầu ra (Output Renderer). Xem hình 2.1



Hình 2. 1 Các thành phần cơ bản của chatbot

- Bộ giải mã đầu vào: Thành phần này dùng để chuyển đổi input thành văn bản. Dữ liệu đầu vào (input) ở đây có thể là giọng nói, cử chỉ hoặc chữ viết tay.
- Hiểu ngôn ngữ tự nhiên: Thành phần này đóng vai trò rất quan trọng trong hệ thống chatbot. Nó trích ra các thông tin cần thiết từ dữ liệu đầu vào để thành phần trình quản lý hội thoại sử dụng. Hai thành phần không thể thiếu của hiểu ngôn ngữ

tự nhiên là bộ phân loại ý định và nhận dạng thực thể (Named Entity Recognition (NER)).

- Trình quản lý hội thoại: Thành phần này giúp quản lý luồng hội thoại. Dựa vào ngữ cảnh và dữ liệu từ hiểu ngôn ngữ tự nhiên thành phần trình quản lý hội thoại sẽ thực hiện truy vấn xuống cơ sở dữ liệu, hiển thị thông báo lỗi nếu có lỗi, hoặc trong trường hợp chưa hiểu ý rõ ý định của người dùng nó có thể tạo câu hỏi để biết thêm thông tin.
- Sinh ngôn ngữ tự nhiên: Thành phần này giúp tạo câu trả lời để trả về cho người dùng. Đối với hệ thống đơn giản chúng ta có thể định nghĩa trước những câu trả lời thay vì sử dụng sinh ngôn ngữ tự nhiên.
- Trình kết xuất đầu ra: thành phần này giúp biểu diễn câu trả lời cho người dùng. Câu trả lời có thể hiển thị dưới nhiều dạng như văn bản, âm thanh,...

2.1.3. Một số ứng dụng Chatbot

- Thời tiết: Nổi bật nhất trong những chatbot về dự báo thời tiết là Poncho, bên cạnh việc dự báo thời tiết chatbot này còn gửi cảnh báo khi thời tiết xấu cho người dùng.
- Từ thiện: Chatbot Yesi là một chatbot đại diện cho các cô gái trẻ ở Ethiopia, những người phải đi bộ 2,5 giờ mỗi ngày để tìm nước sạch. Đây là một chatbot gửi hình ảnh, video, clip âm thanh và bản đồ để đem đến cho người dùng những cảm xúc chân thật về cuộc sống khắc nghiệt của người dân Ethiopia. Từ đó, nâng cao nhận thức của người dùng về sự thiếu nước nghiêm trọng ở Ethiopia.
- Nhà hàng và các ngành bán lẻ: Khách hàng được chatbot cung cấp các tùy chọn menu như: chọn vị trí chỗ ngồi, thanh toán và được thông báo thời gian chính xác để lấy thức ăn của họ.
- Khách sạn và Du lịch: Trong lĩnh vực khách sạn và du lịch thì chatbot có thể hỗ trợ một số tác vụ như là quản lý thời gian, chăm sóc dịch vụ khách hàng và giảm chi phí nhân lực.
- Y tế: Chatbot về lĩnh vực y tế sẽ hỏi về các triệu chứng bệnh, các thông số cơ thể

và lịch sử y tế sau đó đưa ra một danh sách các nguyên nhân gây ra hầu hết các triệu chứng và xếp hạng các bệnh theo thứ tự nghiêm trọng.^[3]

2.2. Tinh chỉnh mô hình (model fine-tuning)

2.2.1. Khái niệm

Tinh chỉnh mô hình là một phương pháp của học chuyển giao (transfer learning) trong học máy đây là một quá trình sử dụng một mô hình mạng đã được huấn luyện cho một nhiệm vụ nhất định từ trước để thực hiện một nhiệm vụ tương tự cụ thể. Mô hình này sử dụng một mô hình được đào tạo trước (pre-trained model) để huấn luyện với một bộ dữ liệu mới, phù hợp với mục đích của người dùng và số lượng tập dữ liệu thường nhỏ hơn khi mô hình được đào tạo trước. Việc này giúp mô hình chính xác hơn so với việc huấn luyện trực tiếp với tập dữ liệu nhỏ.

Khi thực hiện tinh chỉnh, phải huấn luyện toàn bộ hoặc một số lớp (layers) của mô hình, và cũng phải lưu lại tất cả các tham số của mô hình hoặc một số lớp của mô hình được tinh chỉnh. Tức là sẽ phải huấn luyện toàn bộ mô hình 10 lần với 10 nhiệm vụ phía sau (downstream tasks), và sau đó lưu lại trọng số (weight) của tất cả 10 mô hình (models).

Đối với những mô hình nhỏ thì không có vấn đề nhiều, tuy nhiên trong xu hướng sử dụng ngày nay là các mô hình cực nặng, từ vài trăm triệu đến vài tỉ tham số như là các mô hình: Llama, Stable Diffusion,... thì việc huấn luyện toàn bộ mô hình, và lưu toàn bộ mô hình là một vấn đề khá khó khăn với những người bị hạn chế về phần cứng.^[4]

2.2.2. Các mô hình ngôn ngữ hỗ trợ tinh chỉnh

2.2.2.1. Mô hình BERT

❖ Khái niệm

Mô hình BERT^[5] (Bidirectional Encoder Representation from Transformer) là mô hình biểu diễn từ theo 2 chiều ứng dụng kỹ thuật Transformer. Mô hình BERT được thiết kế là để huấn luyện trước các biểu diễn từ (pre-train word embedding). Điều đặc biệt của mô hình này là có thể điều hòa cân bằng bối cảnh theo cả 2 chiều phải và trái.

Cơ chế chú ý (attention) của Transformer sẽ truyền toàn bộ các từ trong câu văn đồng thời vào mô hình một lúc mà không cần quan tâm đến chiều của câu. Vì thế Transformer được xem như là huấn luyện hai chiều (bidirectional). Đặc điểm này cho phép mô hình học được bối cảnh của từ dựa trên toàn bộ các từ xung quanh nó bao gồm cả từ bên trái và từ bên phải.

Mô hình BERT mở rộng khả năng của các phương pháp trước đây bằng cách tạo các biểu diễn theo ngữ cảnh dựa trên các từ trước và sau đó để dẫn đến một mô hình ngôn ngữ với ngữ nghĩa phong phú hơn.

❖ **Kiến trúc**

Hiện tại có nhiều phiên bản khác nhau của mô hình BERT. Các phiên bản đều dựa trên việc thay đổi kiến trúc của Transformer và chủ yếu tập trung ở ba tham số:

- L : số lượng các khối lớp con trong transformer.
- H : kích thước ẩn của véc tơ (hidden size).
- A: Số lượng đầu (head) trong lớp multi-head.

Mỗi một đầu sẽ thực hiện một cơ chế chú ý (self- attention). Tên gọi của hai kiến trúc bao gồm:

- BERT_{BASE}(L = 12, H = 768, A = 12): Tổng tham số 110 triệu.
- BERT_{LARGE}(L = 24, H = 1024, A = 16): Tổng tham số 340 triệu.

Như vậy ở kiến trúc BERT Large có số tầng (L) tăng gấp đôi, kích thước ẩn của véc tơ (H) tăng gấp 1.33 lần và số lượng head trong tầng multi-head(A) tăng gấp 1.33 lần.

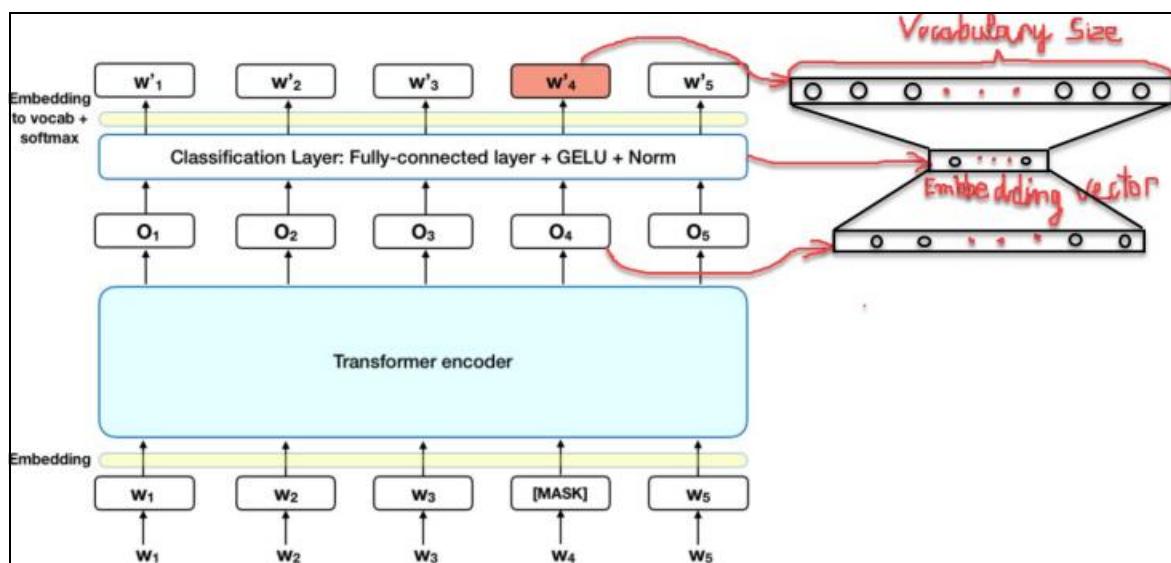
❖ **Nhiệm vụ được đào tạo trước(Pre-training Tasks)**

Đào tạo mô hình BERT bằng cách sử dụng 2 nhiệm vụ dự đoán không giám sát được gọi là mô hình ngôn ngữ được che giấu (Masked Language Model) và dự đoán câu tiếp theo (Next Sentence Prediction).

- **Mô hình ngôn ngữ được che giấu (Masked Language Model)**

Mô hình ngôn ngữ được che giấu là một tác vụ cho phép tinh chỉnh lại các biểu diễn từ trên các bộ dữ liệu văn bản không giám sát bất kỳ. Chúng ta có thể áp dụng mô hình ngôn ngữ được che giấu cho những ngôn ngữ khác nhau để tạo ra biểu diễn nhúng cho chúng. Các bộ dữ liệu tiếng anh có kích thước rất lớn tới vài trăm thậm chí vài nghìn GB được huấn luyện trên mô hình này đã tạo ra những kết quả rất ấn tượng.

Dưới là sơ đồ huấn luyện mô hình BERT theo tác vụ mô hình ngôn ngữ được che giấu (Xem Hình 2.2)



Hình 2. 2 Sơ đồ kiến trúc BERT cho tác vụ Masked ML [5]

Theo đó:

- ✓ Khoảng 15% các mã của câu đầu vào được thay thế bởi mã *[MASK]* trước khi truyền vào mô hình đại diện cho những từ bị che dấu (masked). Mô hình sẽ dựa trên các từ không được che dấu (non-masked) xung quanh *[MASK]* và đồng thời là bối cảnh của *[MASK]* để dự báo giá trị gốc của từ được che dấu. Số lượng từ được che dấu được lựa chọn là một số ít (15%) để tỷ lệ bối cảnh chiếm nhiều hơn (85%).
- ✓ Bản chất kiến trúc của mô hình BERT vẫn là một mô hình sequence to sequence (seq2seq) gồm hai pha mã hóa giúp các từ đầu vào và bộ giải mã giúp tìm ra phân phối xác suất của các từ ở đầu ra. Kiến trúc mã hóa Transformer được giữ lại trong tác vụ mô hình ngôn ngữ được che giấu. Sau khi thực hiện cơ chế liên quan đến các vị trí khác nhau của một chuỗi và sẽ thu được các vec tơ ở đầu ra là O_1, O_2, \dots, O_5 .
- ✓ Để tính toán phân phối xác suất cho từ đầu ra, ta thêm một lớp kết nối đầy đủ ngay sau bộ mã hóa Transformer. Hàm softmax có nhiệm vụ tính toán phân phối xác suất. Số lượng units của lớp kết nối đầy đủ phải bằng với kích thước của từ điển.
- ✓ Cuối cùng ta thu được vec tơ nhúng của mỗi một từ tại vị trí MASK sẽ là vec tơ O_i giảm chiều của vec tơ sau khi đi qua lớp kết nối đầy đủ như mô tả trên hình vẽ bên trên.

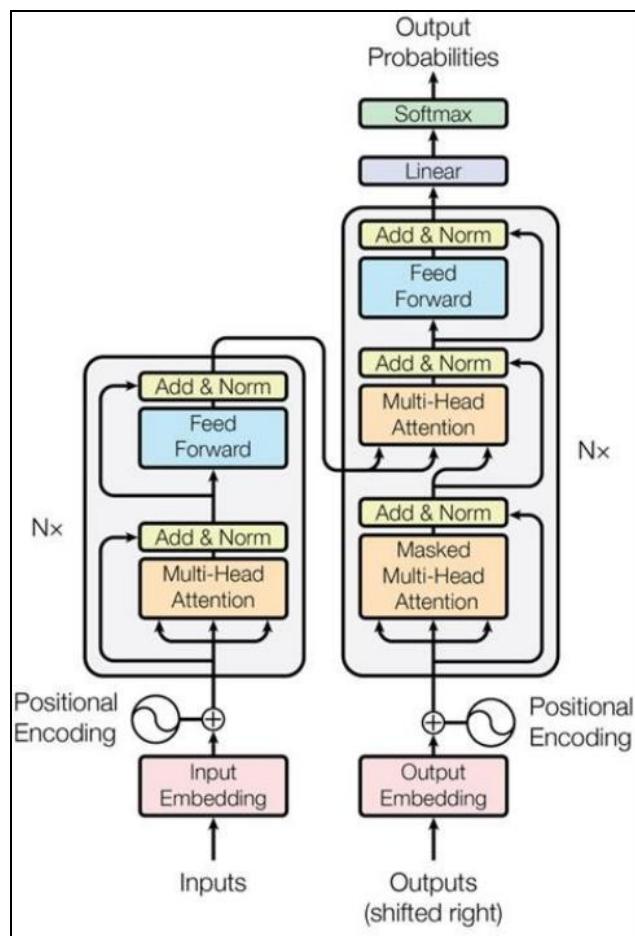
Hàm mát mát của mô hình BERT sẽ bỏ qua mát mát từ những từ không bị che dấu và chỉ đưa vào phần mát mát của những từ bị che dấu. Do đó mô hình sẽ hội tụ lâu hơn bởi vì đây là đặc tính bù trừ cho sự gia tăng ý thức về bối cảnh. Việc lựa chọn ngẫu nhiên 15% số lượng các từ bị che dấu cũng tạo ra vô số các kịch bản đầu vào cho mô hình huấn luyện nên mô hình sẽ cần phải huấn luyện rất lâu mới học được toàn diện các khả năng.

• Dự đoán câu tiếp theo (Next Sentence Prediction)

Đây là một bài toán phân loại học có giám sát với hai nhãn (hay còn gọi là phân loại nhị phân). Dữ liệu đầu vào của mô hình là một cặp câu sao cho 50% câu thứ hai được lựa chọn là câu tiếp theo của câu thứ nhất và 50% được lựa chọn một cách ngẫu

nhiên từ bộ văn bản mà không có mối liên hệ gì với câu thứ nhất. Nhãn của mô hình sẽ ứng với *IsNext* khi cặp câu là liên tiếp hoặc *NotNext* nếu cặp câu không liên tiếp.

Cũng tương tự như mô hình câu hỏi và câu trả lời, chúng ta cần đánh dấu các vị trí đầu câu thứ nhất bằng mã *[CLS]* và vị trí cuối các câu bằng mã *[SEP]*. Các mã này có tác dụng nhận biết các vị trí bắt đầu và kết thúc của từng câu thứ nhất và thứ hai.



Hình 2. 3 Sơ đồ kiến trúc mô hình BERT cho tác vụ NSP ^[5]

Thông tin đầu vào được xử lý trước khi đưa vào mô hình huấn luyện bao gồm:

- ✓ Ngữ nghĩa của từ: Thông qua các véc tơ cho từng từ. Các véc tơ được khởi tạo từ mô hình huấn luyện.

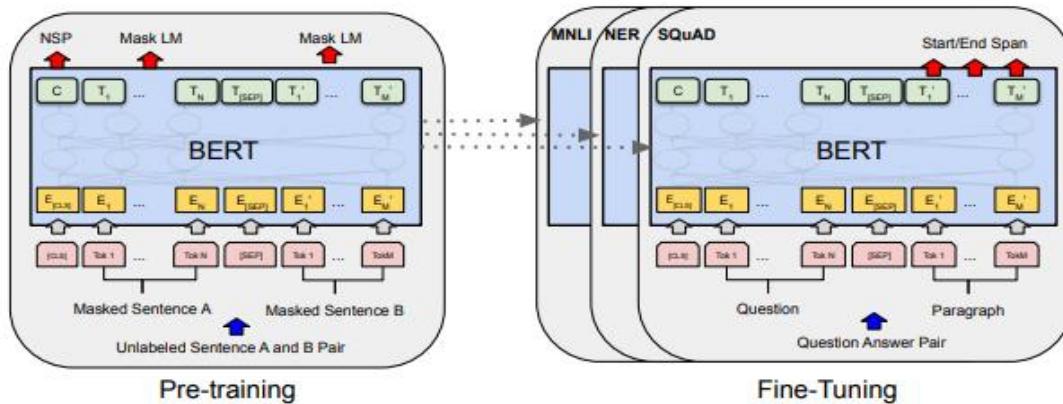
Ngoài véc tơ biểu diễn từ của các từ trong câu, mô hình còn gắn thêm vào một số thông tin:

- ✓ Loại câu: Gồm hai véc tơ là nếu từ thuộc câu thứ nhất và nếu từ thuộc câu thứ hai.
- ✓ Vị trí của từ trong câu: là các véc tơ. Tương tự như vị trí từ trong transformer.

Véc tơ đầu vào sẽ bằng tổng của cả ba thành phần gắn theo *từ*, *câu* và *vị trí*.

❖ Tinh chỉnh mô hình BERT

Một điều đặc biệt ở mô hình BERT mà các mô hình nhúng (model embedding) trước đây không có là kết quả huấn luyện có thể tinh chỉnh được. Thêm vào kiến trúc model một lớp đầu ra (output layer) để tùy biến theo tác vụ huấn luyện.



Hình 2. 4 Toàn bộ tiến trình đào tạo trước và tinh chỉnh của BERT.^[5]

Một kiến trúc tương tự được sử dụng cho cả huấn luyện mô hình và tinh chỉnh mô hình. Chúng ta sử dụng cùng một tham số huấn luyện để khởi tạo mô hình cho các tác vụ khác nhau. Trong suốt quá trình tinh chỉnh thì toàn bộ các tham số của các lớp học chuyển giao sẽ được chỉnh sửa. Đối với các tác vụ sử dụng đầu vào là một cặp nối tiếp (pair-sequence) ví dụ như câu hỏi và câu trả lời thì ta sẽ thêm mã khởi tạo là [CLS] ở đầu câu, mã [SEP] ở giữa để ngăn cách hai câu. Tiến trình áp dụng mô hình tinh chỉnh sẽ như sau:

- Bước 1: Nhúng (embedding) toàn bộ các mã của cặp câu bằng các véc tơ nhúng từ mô hình huấn luyện trước. Các mã nhúng bao gồm cả hai mã là [CLS] và [SEP]

để đánh dấu vị trí bắt đầu của câu hỏi và vị trí ngăn cách giữa 2 câu. Hai mã này sẽ được dự báo ở đầu ra để xác định các phần Bắt đầu/Kết thúc của câu đầu ra.

- Bước 2: Các véc tơ nhúng sau đó sẽ được truyền vào kiến trúc có cơ chế chú ý multi-head với nhiều khối mã (thường là 6, 12 hoặc 24 khối tùy theo kiến trúc của mô hình BERT). Từ đó thu được một véc tơ đầu ra ở bộ mã hóa.
- Bước 3: Để dự báo phân phối xác suất cho từng vị trí từ ở bộ giải mã, ở mỗi bước thời gian sẽ truyền vào véc tơ giải mã đầu ra của mã hóa và véc tơ nhúng đầu vào của bộ giải mã để tính. Sau đó tham chiếu qua lớp lót và hàm softmax để thu được phân phối xác suất cho đầu ra tương ứng ở các bước thời gian.
- Bước 4: Trong kết quả trả ra ở đầu ra của transformer ta sẽ cố định kết quả của câu hỏi sao cho trùng với câu hỏi ở đầu vào. Các vị trí còn lại sẽ là thành phần mở rộng khoảng Bắt đầu/Kết thúc tương ứng với câu trả lời tìm được từ câu đầu vào.

Quá trình huấn luyện sẽ phải tinh chỉnh lại toàn bộ các tham số của mô hình BERT đã cắt lớp tuyến tính trên cùng và huấn luyện lại từ đầu các tham số của lớp tuyến tính mà đã được thêm vào kiến trúc của mô hình BERT để thiết lập lại phù hợp với bài toán.

2.2.2.2. Mô hình PhoBERT

❖ Định nghĩa

Mô hình PhoBERT là một mô hình được đào tạo trước và được huấn luyện ngôn ngữ đơn ngữ (monolingual language), có nghĩa là chỉ huấn luyện dành riêng cho ngôn ngữ tiếng Việt. Việc huấn luyện dựa trên kiến trúc và cách tiếp cận giống mô hình RoBERTa của Facebook và đây là một nâng cấp so với mô hình BERT trước đây.^[6]

❖ Mô hình

Có hai hướng đi cho việc mô phỏng tiếng Việt: mô hình đa ngôn ngữ và mô hình đơn ngữ tiếng Việt.

PhoBERT là một mô hình tiếng Việt nhằm tối ưu hóa cung cấp một thước đo cơ sở cho các bài toán về tiếng Việt.

Có hai phiên bản của PhoBERT gồm có base và large. Cả hai đều có dùng chung kiến trúc của $BERT_{Base}$ và $BERT_{Large}$. Nhóm tác giả huấn luyện mô hình trên 20GB (Gigabyte) dữ liệu tiếng Việt trong đó có 1GB là các bài viết Wikipedia tiếng Việt và 19GB tin tức. Lượng dữ liệu đó tương đương 3 tỷ từ (token) tạo thành khoảng 145 triệu câu. Tuy nhiên, PhoBERT cần một bước tiền xử lý tách từ (ví dụ “tôi là học sinh” → [“tôi”, “là”, “học sinh”]) của một thư viện khác là VnCoreNLP, việc này làm chậm trong việc xử lý trong thời gian thực. Một nhược điểm nữa của mô hình đơn ngữ là công cụ có nhiệm vụ chia đoạn văn bản thành các đơn vị nhỏ hơn (tokenizer) không thể nhận diện từ vựng (out-of-vocabulary) bởi vì có nhiều từ mượn của các ngôn ngữ khác. Trong trường hợp đó, mô hình đa ngữ có lẽ là một sự lựa chọn hợp lý và an toàn hơn.^[7]

2.2.2.3. Mô hình XLM-RoBERTa

Mô hình XLM-RoBERTa (viết tắt của Cross-lingual Language Model - RoBERTa hay còn được gọi là XLM-R) là một mô hình đa ngôn ngữ được sử dụng rộng rãi. Mô hình này được huấn luyện trên 2.5TB(TB - Terabyte) dữ liệu bằng 100 ngôn ngữ được lọc từ bộ dữ liệu Common Crawl.

Mô hình XLM-RoBERTa có hai phiên bản Base và Large. Phiên bản Large có số tầng là 24 nhiều gấp đôi bản Base là 12, số lượng tham số cũng nhiều hơn là 550 triệu so với 270 triệu.^[7]

Mô hình RoBERTa được giới thiệu bởi Facebook là một phiên bản được huấn luyện lại của mô hình BERT với một phương pháp huấn luyện tốt hơn với dữ liệu nhiều hơn được tăng gấp 10 lần.

Để tăng cường quá trình huấn luyện, mô hình RoBERTa không sử dụng cơ chế dự đoán câu kế tiếp từ mô hình BERT mà sử dụng kỹ thuật mặt nạ động (dynamic masking), do đó các từ mặt nạ trong quá trình huấn luyện sẽ bị thay đổi. Sử dụng kích thước batch lớn hơn khi huấn luyện sẽ cho hiệu quả tốt hơn.

Mô hình RoBERTa sử dụng 160GB văn bản để huấn luyện. Trong đó, 16GB dữ liệu là sách và Wikipedia tiếng Anh được sử dụng trong huấn luyện mô hình BERT. Phần còn lại bao gồm tập dữ liệu Common Crawl News (gồm có 63 triệu bản tin và kích thước là 76 GB), dữ liệu văn bản Web (38 GB) và Common Crawl Stories (31 GB). Mô hình này được huấn luyện với GPU của Tesla 1024 V100 trong một ngày.^[8]

CHƯƠNG 3. THƯ VIỆN VÀ API HỖ TRỢ XÂY DỰNG CHATBOT

3.1. Thư viện

3.1.1. Thư viện PyTorch

3.1.1.1. Giới thiệu chung

PyTorch là một thư viện mã nguồn mở dựa trên Python, được phát triển bởi phòng thí nghiệm nghiên cứu AI của Facebook. Thư viện này cung cấp một cách tiếp cận linh hoạt và sáng tạo trong việc xây dựng các mô hình học máy và học sâu. PyTorch được thiết kế để tối ưu hóa việc xây dựng và huấn luyện các mạng nơ-ron sâu.

3.1.1.2. Kiến trúc xây dựng Chatbot trong PyTorch

❖ Lớp nhúng (Embedding Layer):

Trong PyTorch, lớp nhúng được sử dụng để ánh xạ từ vựng thành các vector, giúp cho việc hiểu và xử lý ngôn ngữ trong hệ thống chatbot.

❖ Mô Hình Chuỗi (RNN hoặc Transformer):

Thư viện PyTorch cung cấp sự linh hoạt trong việc chọn lựa giữa việc sử dụng mô hình RNN truyền thống hoặc mô hình học máy (Transformer) mạnh mẽ. Mô hình RNN thích hợp cho xử lý chuỗi, còn mô hình học máy mang lại khả năng học mối quan hệ xa và tốc độ cao.

❖ Cơ chế chú ý (Attention Mechanism):

Nếu chọn sử dụng mô hình học máy, việc tích hợp cơ chế chú ý có thể được thực hiện để tăng cường khả năng tập trung vào các phần quan trọng của đầu vào và đầu ra, hữu ích trong việc hiểu và tạo ra câu trả lời.

❖ Lớp tuyến tính và hàm kích hoạt(Activation Function):

PyTorch cung cấp các lớp tuyến tính và hàm kích hoạt để tạo ra các dự đoán và định dạng đầu ra của hệ thống chatbot, giúp tối ưu hóa hiệu suất của mô hình.

❖ Hàm mất mát và tối ưu hóa:

Việc lựa chọn hàm mất mát như CrossEntropyLoss và Optimizer như Adam trong PyTorch giúp điều chỉnh trọng số của mô hình trong quá trình huấn luyện.

3.1.1.3. Tính năng

Một số tính năng xây dựng chatbot của thư viện PyTorch:

- Xây dựng mô hình mạng nơ-ron: Có thể sử dụng các lớp và hàm của PyTorch để xây dựng mạng nơ-ron như RNN hoặc học máy để xử lý dữ liệu ngôn ngữ tự nhiên và đưa ra các dự đoán.
- Xử lý ngôn ngữ tự nhiên: PyTorch cung cấp các công cụ và thư viện hỗ trợ xử lý ngôn ngữ tự nhiên như torch text và transformers để xử lý ngôn ngữ tự nhiên. Sử dụng chúng để tiền xử lý dữ liệu văn bản, xử lý câu hỏi và phản hồi, và xây dựng các mô hình xử lý ngôn ngữ tự nhiên cho chatbot.
- Huấn luyện và tinh chỉnh mô hình: Sử dụng các thuật toán tối ưu và phương pháp tinh chỉnh, để cải thiện hiệu suất và khả năng dự đoán của chatbot.
- Tích hợp dễ dàng: PyTorch tích hợp tốt với các thư viện và framework khác như Hugging Face Transformers và Flask.
- Tính linh hoạt và mở rộng: PyTorch là một thư viện linh hoạt và mở rộng, cho phép xây dựng chatbot từ các mô hình đơn giản đến các mô hình phức tạp.

3.1.2. Thư viện Hugging Face Transformers

3.1.2.1. Giới thiệu chung

Kiến trúc học máy (Transformer) đã nhanh chóng trở thành một xu hướng trong xử lý ngôn ngữ tự nhiên kể từ khi được giới thiệu. Đặc biệt là, nó không chỉ là một lựa chọn mạnh mẽ, mà còn vượt trội so với các kiến trúc trước đó như CNN và RNN, đặc biệt là trong việc giải quyết các tác vụ phức tạp như hiểu ngôn ngữ và sinh ngôn ngữ tự nhiên.

Sự ra đời của mô hình đào tạo trước cho phép mô hình được huấn luyện trước với một lượng lớn văn bản và sau đó được điều chỉnh để tối ưu hóa cho các tác vụ cụ thể, mang lại hiệu suất tối ưu nhất.

Kiến trúc học máy đặc biệt linh hoạt khi áp dụng mô hình đào tạo trước trên các tập văn bản lớn, giúp tăng độ chính xác trong nhiều tác vụ như hiểu ngôn ngữ, phân loại văn bản.

Ưu điểm này đã đưa ra nhiều thách thức mới, đòi hỏi sự linh hoạt và khả năng sử dụng rộng rãi của mô hình không chỉ trong nghiên cứu mà còn trong các ứng dụng thực tế. Vì vậy, HuggingFace's Transformer ra đời như một nguồn tài nguyên hữu ích, không chỉ cung cấp công cụ huấn luyện, phân tích và đánh giá hiệu suất của mô hình mà còn hỗ trợ mở rộng và sự linh hoạt trên nhiều nền tảng.

3.1.2.2. Cấu trúc chung của thư viện Hugging Face Transformers:

Kiến trúc của thư viện Transformers lấy cảm hứng từ thư viện tensor2tensor và mã nguồn gốc của mô hình BERT, cả hai đều được phát triển bởi Google Research. Sự đổi mới này mang lại khái niệm về việc cung cấp bộ nhớ đệm cho các mô hình đào tạo lại.

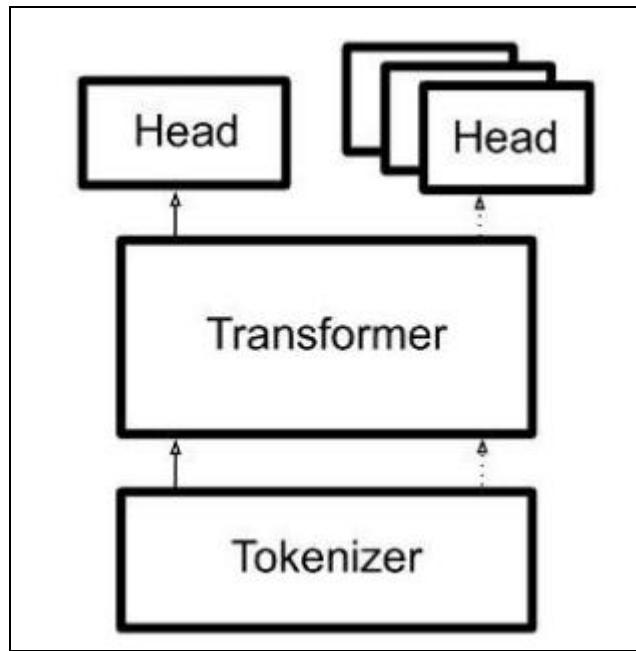
Xây dựng dựa trên những yếu tố này, Transformers mở rộng với các tính năng chính hướng đến người dùng, bao gồm quản lý bộ nhớ đệm, cho phép tải xuống và điều chỉnh mô hình cũng như chuyển đổi mô hình một cách liền mạch thành sản phẩm.

Thiết kế của Transformers tuân theo một đường ống (pipeline) tiêu chuẩn cho các bài toán xử lý ngôn ngữ tự nhiên, bao gồm xử lý dữ liệu, mô hình hóa dữ liệu và đưa ra dự đoán. Mỗi mô hình trong thư viện được định nghĩa bởi ba khối chính:

- Tokenizer: Chuyển đổi văn bản sang mã hóa chỉ mục.
- Transformer: Biến đổi mã hóa chỉ mục thành các vector nhúng theo ngữ cảnh.
- Head: Sử dụng các nhúng để đưa ra dự đoán cho từng tác vụ cụ thể.

Ba khối này có thể được xử lý đa số các tác vụ người dùng. Sơ đồ chi tiết bên dưới mô tả cấu trúc của một mô hình, được hình thành từ ba khối: tokenizer, transformer và head. Các mô hình có thể được điều chỉnh trước với một head cố định và có thể được điều chỉnh lại với các head khác nhau tùy thuộc vào nhiệm vụ cụ thể.

Các phần còn lại của sơ đồ mô tả chi tiết của các khối tokenizer, transformer và head tùy thuộc vào các nhiệm vụ khác nhau.



Hình 3. 1 Các thành phần của thư viện Transformers Hugging Face^[9]

❖ Tokenizer

Khối Tokenizers đóng vai trò quan trọng và không thể thiếu trong quá trình xử lý các bài toán xử lý ngôn ngữ tự nhiên. Các lớp Tokenizer được kế thừa từ một lớp cơ sở, có thể được khởi tạo từ một mô hình đào tạo trước tương ứng hoặc có thể được thiết lập thủ công. Các lớp này gồm các ánh xạ từ sang chỉ mục trong tập từ vựng của mô hình và các bộ mã hai chiều của chuỗi đầu vào cho quá trình mã hóa trong từng mô hình cụ thể.

Mỗi mô hình sử dụng một tokenizer cụ thể được viết bằng Python hoặc Rust. Cho dù có một số khác biệt nhỏ, nhưng chúng vẫn cần đồng bộ với quá trình đào tạo trước.

Người dùng được phép điều chỉnh khối Tokenizer thông qua giao diện để bổ sung ánh xạ token, token đặc biệt hoặc điều chỉnh kích thước của tập từ vựng.

Với các tập dữ liệu lớn, tokenizer viết bằng ngôn ngữ Python có thể chậm hơn so với ngôn ngữ Rust. Thư viện cấp thấp cho tokenizer viết bằng Rust đã được chọn làm mặc định do tốc độ tokenizer nhanh của nó, phù hợp cho cả quá trình huấn luyện và triển khai các mô hình.

❖ Transformers

Transformers đóng vai trò quan trọng nhất của thư viện và đã được thử nghiệm với nhiều biến thể khác nhau. Các biến thể này chia sẻ một lõi attention với nhiều đầu, nhưng chúng có sự khác biệt ở các khía cạnh như xử lý padding, biểu diễn vị trí, áp dụng mặt nạ và cách mà chúng được thiết kế cho các mô hình chuỗi đến chuỗi(sequence to sequence - seq2seq).

Trên thực tế, tất cả các mô hình đều sử dụng một lớp cơ sở trùu tượng, điều đó có nghĩa là một lớp được triển khai để tạo thành mô hình từ một bộ mã hóa đi qua các lớp chú ý và sau đó đi qua một bộ mã hóa cuối cùng.

Lớp cơ sở được tùy chỉnh cho từng mô hình cụ thể và phải tuân theo biến thể gốc, cho người dùng sự linh hoạt để phân tích các thành phần của nó. Trong hầu hết các trường hợp, mỗi mô hình được triển khai thành một tệp đơn lẻ để dễ dàng cho việc mở rộng.

❖ Heads

Các head có thể được kết hợp linh hoạt tùy thuộc vào nhiều tác vụ khác nhau. Mỗi head được triển khai dưới dạng các lớp nằm ở phía trên của lớp cơ sở, và sau đó được mở rộng với các lớp đầu ra độc lập và hàm mốt mát, được tích hợp vào phần nhúng ngữ cảnh của khối Transformer.

Các lớp được đặt tên theo một quy tắc chung: XXXForSequenceClassification, trong đó XXX là tên của mô hình được sử dụng để điều chỉnh hoặc huấn luyện trước. Một số head, như conditional generation, cung cấp các tính năng bổ sung như lấy mẫu và tìm kiếm theo phong cách "beam".

Đối với các mô hình được huấn luyện trước, các head có thể đã được sử dụng để huấn luyện trước chính mô hình đó, như trong trường hợp của mô hình BERT.

3.2. API

3.2.1. Khái niệm API

API là viết tắt của cụm từ "Application Programming Interface" (Giao diện Chương trình Ứng dụng).^[10] Cách API hoạt động thường được phân tích ở hai khía cạnh:

máy chủ và máy khách. Máy khách là nơi các ứng dụng gửi yêu cầu trong khi máy chủ là nơi các ứng dụng gửi phản hồi.

Có bốn cách chính để API hoạt động, bao gồm API SOAP, API RPC, API WebSocket và API REST. Trong đó, API REST được xem là linh hoạt nhất và được sử dụng phổ biến nhiều nhất.

API có nhiều loại, bao gồm API Cá nhân (API Private) chỉ dành cho kết nối trong cùng một công ty; API công cộng (API Public) là API mà bất kỳ ai cũng có thể sử dụng; và khi sử dụng các API này, người dùng sẽ nhận được yêu cầu ủy quyền, yêu cầu trả chi phí khi sử dụng hoặc thậm chí là không có yêu cầu gì cả.

3.2.2. Các API hỗ trợ xây dựng chatbot

3.2.2.1. API của OpenAI

❖ Khái niệm

API OpenAI là một bộ các giao diện lập trình ứng dụng mà OpenAI đã phát triển và duy trì, mang đến khả năng tiếp cận các công nghệ trí tuệ nhân tạo tiên tiến trong lĩnh vực xử lý ngôn ngữ tự nhiên. Các nhà phát triển có thể sử dụng API này để xây dựng và tích hợp các ứng dụng thông minh như chatbot, trợ lý viết AI, và nhiều ứng dụng khác.

Bằng cách sử dụng API OpenAI, những người phát triển có thể dễ dàng truy cập các mô hình ngôn ngữ quy mô lớn. Mô hình nổi bật nhất trong danh mục của OpenAI là GPT-3, đây là mô hình tạo ngôn ngữ tự nhiên lớn nhất hiện nay với khả năng đáp ứng và sáng tạo cao. Ngoài ra, API OpenAI còn cung cấp quyền truy cập vào các mô hình khác như mô hình chuyên tạo hình ảnh (DALL-E).

Các lệnh gọi API đơn giản giúp nhà phát triển tận dụng các khả năng mạnh mẽ của các mô hình này. Việc sử dụng API OpenAI giúp nhà phát triển dễ dàng tích hợp và triển khai các ứng dụng trí tuệ nhân tạo mà không cần phải xây dựng các mô hình từ đầu.

Đồng thời, API OpenAI cung cấp nhiều mô hình và bộ dữ liệu đã được đào tạo trước, giúp nhà phát triển tiếp cận một loạt các tác vụ xử lý ngôn ngữ tự nhiên khác nhau. Điều

này giúp tăng cường khả năng ứng dụng và linh hoạt trong việc giải quyết các thách thức liên quan đến xử lý ngôn ngữ.

Nó cho phép truy cập vào khả năng của ChatGPT để tạo ra các câu trả lời giống con người cho các câu hỏi và tham gia vào cuộc trò chuyện thông thường.

❖ Các tính năng

➤ Trò chuyện (Chat):

OpenAI API giúp nhà phát triển xây dựng chatbot tương tác tự nhiên, tạo trải nghiệm cuộc trò chuyện hấp dẫn cho người dùng. Điều này giúp cải thiện trải nghiệm và tạo ra ứng dụng trò chuyện thông minh.

➤ Âm thanh (Audio):

OpenAI API giúp các nhà phát triển có thể tiếp cận với các tính năng âm thanh dễ dàng hơn bao gồm:

- Tính năng tạo âm thanh: Tạo âm thanh từ đầu vào là văn bản.
- Tạo bản chép lời (transcription): Chuyển đổi đoạn âm thanh thành ngôn ngữ đầu vào.
- Tạo bản dịch (translation): Dùng để thực hiện chuyển đổi âm thanh sang ngôn ngữ tiếng Anh.

➤ Hỗ trợ tinh chỉnh mô hình:

OpenAI giúp các nhà phát triển có thể thực hiện xây dựng, tinh chỉnh mô hình riêng một cách dễ dàng với các mô hình như gpt-3.5-turbo, gpt-3.5-turbo-1106, text-davinci-002,...

3.2.2.2. Google Cloud API

❖ Khái niệm

Google Cloud API là một bộ các giao diện lập trình ứng dụng(API) mà Google Cloud cung cấp để hỗ trợ các nhà phát triển tích hợp và sử dụng các dịch vụ mạnh mẽ của Google Cloud trong ứng dụng của họ. Các API này cung cấp một loạt các chức năng và khả năng đa dạng, giúp giảm bớt độ phức tạp của quá trình phát triển ứng dụng và cho

phép nhà phát triển tập trung vào việc xây dựng chức năng cốt lõi của ứng dụng mà không cần phải lo lắng về quản lý cơ sở hạ tầng phức tạp.

Các API của Google Cloud API hỗ trợ xây dựng chatbot cung cấp các tính năng cần thiết để tạo ra các chatbot có thể hiểu và phản hồi ngôn ngữ tự nhiên, và cho phép người dùng trò chuyện với chatbot bằng giọng nói.

❖ **Tính năng**

- **Đa Dạng và Phong Phú:** Google Cloud API bao gồm nhiều lĩnh vực như xử lý hình ảnh, ngôn ngữ tự nhiên, lưu trữ, machine learning, dữ liệu lớn, bảo mật, và nhiều lĩnh vực khác, giúp đáp ứng nhu cầu đa dạng của các ứng dụng.
- **Linh Hoạt và Mở Rộng:** Các API của Google Cloud được thiết kế để linh hoạt và dễ tích hợp với nhiều loại ứng dụng và hệ thống. Người phát triển có thể sử dụng các API này từ nhiều ngôn ngữ lập trình và môi trường phát triển.
- **Bảo Mật Cao Cấp:** Các API của Google Cloud được tích hợp với các tính năng bảo mật mạnh mẽ như xác thực, ủy quyền, mã hóa dữ liệu, và quản lý chứng chỉ.
- **Quản Lý Dự Án và Tài Nguyên:** Google Cloud API hỗ trợ quản lý dự án và tài nguyên thông qua Google Cloud Console và Cloud Identity and Access Management (IAM).
- **Dịch Vụ Bản Đồ và Địa Lý:** Google Cloud API cung cấp các dịch vụ như Google Maps Platform để tích hợp thông tin địa lý và bản đồ vào ứng dụng.

❖ **API xây dựng chatbot**

- **Cloud Speech-to-Text API:** Cung cấp khả năng chuyển đổi âm thanh thành văn bản, hỗ trợ nhiều ngôn ngữ và định dạng âm thanh.
- **Cloud Text-to-Speech API:** Cung cấp khả năng chuyển đổi văn bản thành âm thanh, chuyển đổi văn bản thành giọng nói với chất lượng và ngữ điệu tự nhiên quen thuộc với tùy ngôn ngữ của từng quốc gia.
- **Dialogflow API:** Dialogflow cung cấp khả năng xử lý và hiểu ngôn ngữ tự nhiên để hiểu ý định và thực hiện các hành động tương ứng.

- Google Bard: Cung cấp khả năng tương tác, gửi và nhận câu trả lời từ chatbot của Google.

Kết hợp hai API như Cloud Speech-to-Text và Cloud Text-to-Speech chatbot có thể xử lý giọng nói người dùng nhập vào thành văn bản và phản hồi lại giọng nói từ văn bản chatbot trả về.

3.2.2.3. Rapid API

❖ Khái niệm

Rapid API là một nền tảng trung gian giữa các nhà phát triển và các dịch vụ API. Rapid API cung cấp một thị trường trực tuyến nơi mọi người có thể tìm kiếm, kết nối và sử dụng hàng trăm hay thậm chí hàng nghìn API khác nhau từ nhiều nhà cung cấp khác nhau. Rapid API giúp tạo ra một cộng đồng mạnh mẽ của nhà phát triển và cung cấp các công cụ quản lý API để theo dõi, đánh giá và quản lý việc sử dụng các dịch vụ API.

❖ Tính năng

- Thư viện API Đa Dạng: Rapid API cung cấp một thư viện lớn các API từ nhiều nguồn khác nhau. Người phát triển có thể tìm kiếm, so sánh và chọn lựa từ hàng ngàn API khác nhau dựa trên nhu cầu cụ thể của họ.
- Tìm Kiếm Nâng Cao: Rapid API cung cấp tính năng tìm kiếm nâng cao để giúp người phát triển dễ dàng tìm kiếm các API phù hợp với yêu cầu của họ, bao gồm các bộ lọc như loại API, giá cả, và nhiều yếu tố khác.
- Quản Lý API Key: Rapid API giúp người phát triển quản lý các API key từ nhiều dịch cung cấp dịch vụ API khác nhau. Điều này giúp quản lý và bảo mật truy cập vào các dịch vụ API mà người phát triển sử dụng.
- Thống Kê và Analytics: Rapid API cung cấp các công cụ thống kê và phân tích để người phát triển có thể theo dõi hiệu suất và sử dụng API của họ.
- Tích Hợp Linh Hoạt: Rapid API hỗ trợ tích hợp dễ dàng với nhiều ngôn ngữ lập trình khác nhau thông qua các thư viện và SDK.
- Bảo Mật: Nền tảng này cung cấp các tính năng bảo mật như quản lý quyền truy cập, mã hóa dữ liệu, và kiểm soát quyền truy cập API.

❖ API xây dựng Chatbot

Một số API phổ biến của Rapid API cho xây dựng chatbot bao gồm:

- ChatGPT API: Là một API dựa trên mô hình ngôn ngữ lớn của OpenAI. ChatGPT có thể được sử dụng để tạo các phản hồi tự nhiên và sinh động cho các truy vấn của người dùng.
- Dialog Flow API: Là một API dựa trên trí tuệ nhân tạo của Google. Dialogflow có thể được sử dụng để hiểu các truy vấn của người dùng, xác định ý định của họ và tạo các phản hồi phù hợp.
- Microsoft Bot Framework API: Là một API dựa trên trí tuệ nhân tạo của Microsoft, được sử dụng để tạo các chatbot có thể hiểu và phản hồi ngôn ngữ tự nhiên.

Ví dụ về cách các API của Rapid API sử dụng để xây dựng chatbot:

- Một chatbot có thể sử dụng ChatGPT API để tạo các phản hồi sáng tạo và hấp dẫn cho các truy vấn của người dùng.
- Một chatbot có thể sử dụng Dialogflow API để hiểu các truy vấn phức tạp của người dùng và tạo các phản hồi thích hợp.

CHƯƠNG 4. CÀI ĐẶT THƯ VIỆN VÀ THỰC HIỆN TINH CHỈNH MÔ HÌNH CHATBOT

4.1. Tổng quan

Chương 4 tập trung vào hai phần chính: phần một cài đặt thư viện và API, phần hai thực hiện tinh chỉnh mô hình ngôn ngữ tiếng Việt cho Chatbot dựa trên lý thuyết về thư viện, API được trình bày ở Chương 3 và phần Tinh chỉnh mô hình ở Chương 2. Quá trình này không chỉ hỗ trợ xây dựng cơ sở hạ tầng mà còn thiết lập nền tảng cho việc tinh chỉnh và đào tạo mô hình theo yêu cầu cụ thể của dự án.

Bố cục chương 4 chi tiết:

- Phần 1: Cài đặt thư viện và API (Xem chi tiết trong mục 4.2 Cài đặt thư viện và API):
 - ❖ Tập trung vào việc cài đặt các thư viện Transformers và sử dụng mô hình hỗ trợ tiếng Việt trong transformers như GPT2 Vietnamese của NlpHUST, ViT5 của VietAI.
 - ❖ Cài đặt và sử dụng OpenAI API và chỉ demo sử dụng tính năng Trò chuyện(Chat), tính năng tinh chỉnh mô hình sẽ được trình bày ở phần 2.
 - ❖ Trình bày cách đăng ký Google Cloud API, cài đặt và sử dụng các tính năng của Google Cloud API như tính năng chuyển đổi văn bản sang giọng nói và giọng nói sang văn bản. Ngoài ra còn trình bày chi tiết cách cài đặt và sử dụng Google Bard.
 - ❖ Đăng ký, cài đặt và sử dụng các dịch vụ của Rapid API.
 - ❖ Xây dựng bảng so sánh các thư viện và API đã xây dựng dựa trên các tiêu chí như tốc độ câu trả lời, chất lượng câu trả lời, giá thành cho việc tinh chỉnh mô hình, v.v
- Phần 2: Thực hiện tinh chỉnh mô hình Chatbot (Xem chi tiết trong mục 4.3 Thực hiện tinh chỉnh mô hình Chatbot):

Phần này sẽ trình bày cách nhóm cài đặt môi trường, chuẩn bị các tập dữ liệu cho việc tinh chỉnh mô hình rồi sau đó là xử lý dữ liệu đầu vào và xây dựng mô hình. Kết thúc phần này sẽ là bảng kết quả từ mô hình nhóm xây dựng.

4.2. Cài đặt thư viện và API

4.2.1. Transformers

Bước 1: Vào trong cửa sổ cmd(Command Prompt) hoặc terminal của ứng dụng Visual Studio Code và gõ lệnh **pip install transformers**.

- Vào trong cmd gõ lệnh **pip show transformers** để kiểm tra đã cài đặt thành công thư viện transformer hay chưa.
- Khi cài đặt thành công kết quả trả ra như hình bên dưới(Hình 4.1):

```
D:\Download\Code\Code_Demo>pip show transformers
WARNING: Skipping C:\Users\Admin\AppData\Roaming\Python\Python311\site-packages\wheel-0.40.0.dist-info due to invalid metadata entry 'name'
DEPRECATION: Loading egg at c:\program files\python311\lib\site-packages\yboxapi-1.0-py3.11.egg is deprecated. pip 24.3 will enforce this behaviour change. A possible replacement is to use pip for package installation.. Discussion can be found at https://github.com/pypa/pip/issues/12330
Version: 4.35.2
Summary: State-of-the-art Machine Learning for JAX, PyTorch and TensorFlow
Home-page: https://github.com/huggingface/transformers
Author: The Hugging Face team (past and future) with the help of all our contributors (https://github.com/huggingface/transformers/graphs/contributors)
Author-email: transformers@huggingface.co
License: Apache 2.0 License
Location: C:\Users\Admin\AppData\Roaming\Python\Python311\site-packages
Requires: filelock, huggingface-hub, numpy, packaging, pyyaml, regex, requests, safetensors, tokenizers, tqdm
Required-by: distilbert-punctuator, peft
```

Hình 4. 1 Hiển thị thông tin cài đặt thư viện transformers

Bước 2: Cài đặt thử nghiệm các mô hình ngôn ngữ tiếng Việt trong transformers như GPT2, ViT5

➤ Mô hình GPT2 Vietnamese

Đầu tiên khai báo thư viện Transformers và khai báo lớp GPT2LMHeadModel và tokenizer tương ứng với GPT2. Tiếp theo xây dựng hàm generate_text với đầu vào là một câu hoặc đoạn hội thoại. Khai báo các tokenizer, model từ mô hình đào tạo trước của GPT2. Thiết lập input_ids để xử lý giá trị đầu vào. Sử dụng model.generate để sinh văn bản dựa trên input_ids. Các tham số như do_sample, max_length, min_length, top_k, num_beams được sử dụng để điều chỉnh quá trình sinh văn bản, num_return_sequences chỉ định số lượng văn bản sinh ra. Bản ghi cuối cùng từ kết quả được giải mã bằng tokenizer và trả về.

```

import torch
from transformers import GPT2LMHeadModel, GPT2Tokenizer

def generate_text(prompt):
    tokenizer = GPT2Tokenizer.from_pretrained('NlpHUST/gpt2-vietnamese')
    model = GPT2LMHeadModel.from_pretrained('NlpHUST/gpt2-vietnamese')
    input_ids = tokenizer.encode(prompt, return_tensors='pt')
    max_length = 200
    sample_outputs = model.generate(input_ids, pad_token_id=tokenizer.eos_token_id,
                                    do_sample=True,
                                    max_length=max_length,
                                    min_length=max_length,
                                    top_k=40,
                                    num_beams=5,
                                    early_stopping=True,
                                    no_repeat_ngram_size=2,
                                    num_return_sequences=3)
    # Chỉ trả về bản ghi cuối cùng
    generated_text = tokenizer.decode(sample_outputs[-1].tolist())
    return generated_text

response = generate_text("Việt Nam là")
print(response)

```

Bảng 4. 1 Sử dụng mô hình GPT2 Vietnamese có nguồn gốc Hugging Face

Kết quả khi sử dụng mô hình GPT2-Vietnamese.

D:\Download\Code\Code_Demo>"C:/Program Files/Python311/python.exe" d:/Download/Code/Code_Demo/gpt2_NLPHUST.py
Việt Nam là một trong những quốc gia có tốc độ phát triển kinh tế nhanh nhất trên thế giới. Chính vì thế mà nhu cầu sử dụng các thiết bị điện tử ngày càng tăng cao. Đặc biệt là những sản phẩm công nghệ thông minh như điện thoại di động, máy tính bảng. Trong đó, Samsung là nhà sản xuất smartphone lớn nhất tại thị trường Việt Nam.

Samsung Galaxy S6 Edge Plus là chiếc smartphone cao cấp nhất của Samsung. Với thiết kế sang trọng, cấu hình mạnh mẽ cùng nhiều tính năng hấp dẫn, đây là sự lựa chọn hoàn hảo cho những ai đang muốn sở hữu một chiếc smartphone tầm trung. Tuy nhiên, không phải ai cũng đủ khả năng tài chính để mua cho mình chiếc Samsung Galaxy S6 edge plus. Bài viết dưới đây sẽ giúp bạn giải quyết vấn đề này một cách dễ dàng và nhanh chóng nhất. Mời các bạn cùng theo dõi nhé!

Trước khi đi vào bài viết, chúng ta sẽ cùng nhau tìm hiểu về cấu

Hình 4. 2 Kết quả khi sử dụng mô hình GPT2 Vietnamese

➤ Mô hình ViT5

Đầu tiên khai báo thư viện Transformers và khai báo lớp AutoModelForSeq2SeqLM và tokenizer tương ứng. Tiếp theo xây dựng hàm summarize_text với đầu vào là một câu hoặc đoạn hội thoại. Khai báo các tokenizer, model từ mô hình đào tạo trước của Seq2SeqLM. Thêm ký tự kết thúc câu ("</s>") vào đầu câu đầu vào và thiết lập input_ids để xử lý giá

trị đầu vào. Đưa input_ids và attention_masks vào mô hình seq2seq để sinh văn bản tổng hợp.

Sử dụng generate để tạo ra văn bản dựa trên đầu vào. Các tham số như max_length và early_stopping được sử dụng để kiểm soát quá trình sinh văn bản. Sau đó sử dụng tokenizer để giải mã và chuyển đổi văn bản được sinh ra từ định dạng số về dạng văn bản thường.

Kết quả được trả về là một đoạn văn bản tóm tắt.

```
from transformers import AutoModelForSeq2SeqLM, AutoTokenizer

def summarize_text(input_sentence):
    tokenizer = AutoTokenizer.from_pretrained("VietAI/vit5-base-vietnews-summarization")
    model = AutoModelForSeq2SeqLM.from_pretrained("VietAI/vit5-base-vietnews-summarization")
    sentence = input_sentence + "</s>"
    encoding = tokenizer(sentence, return_tensors="pt")
    input_ids, attention_masks = encoding["input_ids"], encoding["attention_mask"]
    outputs = model.generate(
        input_ids=input_ids, attention_mask=attention_masks,
        max_length=256,
        early_stopping=True
    )
    summary = tokenizer.decode(outputs[0], skip_special_tokens=True,
    clean_up_tokenization_spaces=True)
    return summary
response = summarize_text("VietAI là tổ chức phi lợi nhuận với sứ mệnh ươm mầm tài năng về trí tuệ nhân tạo và xây dựng cộng đồng các chuyên gia")
print(response)
```

Bảng 4. 2 Đoạn code sử dụng mô hình ViT5 trên Hugging Face

Kết quả khi cài đặt mô hình ViT5.

```
C:\Users\Admin\AppData\Roaming\Python\Python311\site-packages\transformers\generation\configuration_utils.py:418: UserWarning: `num_beams` is set to
However, `early_stopping` is set to `True` -- this flag is only used in beam-based generation modes. You should set `num_beams>1` or unset `early_stopping`.
  warnings.warn(
ViệtAI sẽ là một trong những nhà sáng chế trí tuệ nhân tạo hàng đầu thế giới về trí tuệ nhân tạo được thành lập năm 2000 tại Hoa Kỳ.
```

Hình 4. 3 Kết quả khi chạy mô hình ViT5

4.2.2. OpenAI API

Bước 1: Tải thư viện OpenAI trong Python bằng dòng lệnh ***pip install openai*** trong cmd hoặc Terminal của Visual Studio Code.

- Kiểm tra đã cài đặt thư viện thành công hay không bằng lệnh ***pip show openai***

```
C:\Users\Admin>pip show openai
WARNING: Skipping C:\Users\Admin\AppData\Roaming\Python\Python311\site-packages\wheel-0.40.0.dist-info due to invalid metadata entry 'name'
DEPRECATION: Loading egg at c:\program files\python311\lib\site-packages\vboxapi-1.0-py3.11.egg is deprecated. pip 24.3 will enforce this be
ip for package installation.. Discussion can be found at https://github.com/pypa/pip/issues/12330
Name: openai
Version: 0.27.9
Summary: Python client library for the OpenAI API
Home-page: https://github.com/openai/openai-python
Author: OpenAI
Author-email: support@openai.com
License:
Location: C:\Users\Admin\AppData\Roaming\Python\Python311\site-packages
Requires: aiohttp, requests, tqdm
Required-by: llama-index
```

Hình 4. 4 Hình ảnh kiểm tra cài đặt thư viện OpenAI trong Python

Bước 2: Vào trang OpenAI Platform (truy cập [tại đây](#)) vào mục API Key chọn **Create new secret key** → nhập tên và nhấn nút **Create secret key**.

Create new secret key

Name Optional

My Test Key

Cancel

Create secret key

Hình 4. 5 Tạo new secret key của OpenAI

- Kết quả sau khi tạo API key thành công.

Create new secret key

Please save this secret key somewhere safe and accessible. For security reasons, you won't be able to view it again through your OpenAI account. If you lose this secret key, you'll need to generate a new one.

sk-uWFTLCIz9wp8RvS73NoQT3BlbkFJhDXSeAI3qrMNUwqgKsI



Done

Hình 4. 6 Hình ảnh sau khi tạo API key thành công

- Kích vào nút sao chép để lưu lại giá trị key → nhấn Done.

Secret key sk-...KsNa 20 thg 12, 2023 Never

Hình 4. 7 Một API key của OpenAI được tạo ra thành công

Tiến hành sử dụng API key vừa tạo để tạo một đoạn chatbot.

Đầu tiên, khai báo thư viện và khóa API sử dụng (đã lấy được từ các bước trên). Tiếp theo thực hiện sử dụng phương thức `openai.ChatCompletion.create()` với model mà mô hình sử dụng, mô hình mặc định sẽ là `gpt-3.5-turbo` ngoài ra còn có các lựa chọn khác như `gpt-3.5-turbo-1106`, `text-davinci-003`, `message` là văn bản gửi tới chatbot trong đó có thể kèm nội dung đoạn hội thoại của chatbot hoặc chỉ giữ lại còn `messages` của user. Ngoài ra có còn thể đính kèm các tham số như `max_tokens` là số lượng từ tối đa mà chatbot phản hồi về. Các tham số khác có thể kể đến như `top_p`, `presence_penalty`, `frequency_penalty` hoặc `temperature`.

```

import openai

openai.api_key = 'sk-gnIBvx582GAFJQKArEWvT3BlbkJP00pYMTjIPmUJ3x09pZR'

# Tạo yêu cầu hoàn thành
response = openai.ChatCompletion.create(
    model="gpt-3.5-turbo",
    messages=[
        {"role": "user", "content": "Xin chào"}
    ]
)
print(response['choices'][0]['message']['content'])

```

Hình 4. 8 Sử dụng API key từ thư viện OpenAI

Kết quả trả về từ OpenAI được hiển thị như hình bên dưới (Hình 4.9).

```

D:\Download\Code\Code_Demo>C:/Program Files/Python311/python.exe" d:/Download/Code/Code_Demo/Model_2/demo.py
Xin chào! Tôi đây! Có điều gì tôi có thể giúp bạn không?

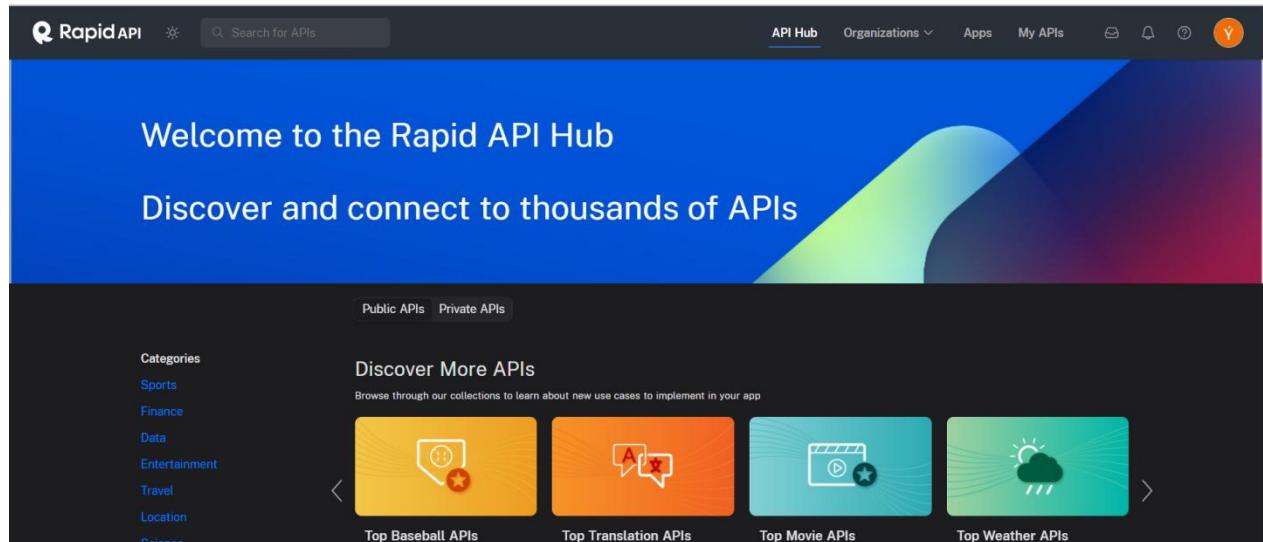
D:\Download\Code\Code_Demo>

```

Hình 4. 9 Kết quả trả về khi sử dụng mô hình từ thư viện OpenAI

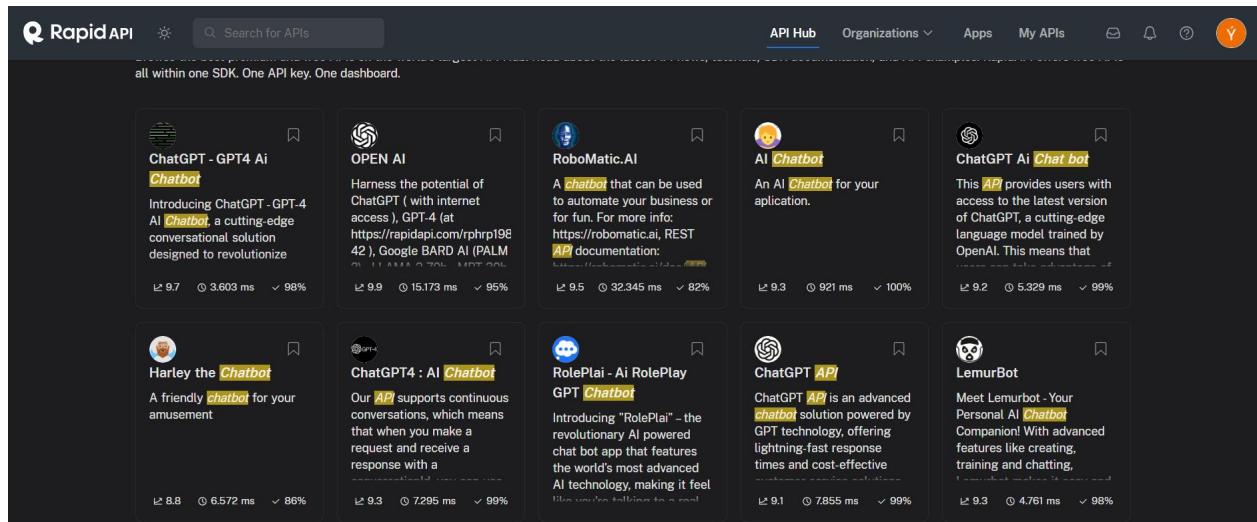
4.2.3. Rapid API

Bước 1: Đăng nhập thành công vào trang web của Rapid API (đường dẫn: [Link](#))



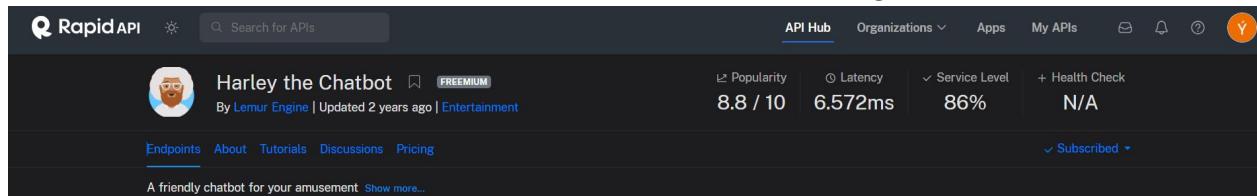
Hình 4. 10 Giao diện trang web Rapid API

Bước 2: Nhập vào thanh Search for APIs với nội dung là ‘**ChatBot API**’ và nhấn Enter, kết quả trả về như hình bên dưới (Hình 4.11)



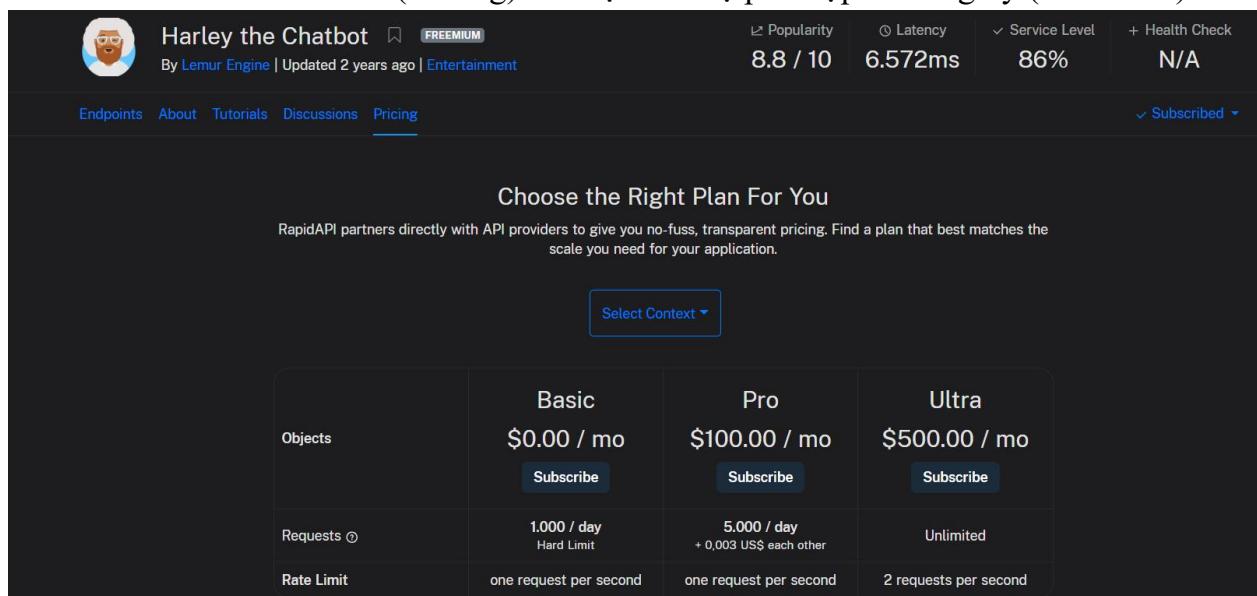
Hình 4. 11 Các API hỗ trợ xây dựng chatbot trong Rapid API

- Cửa sổ hiện ra như hình trên và chọn API muốn sử dụng.



Hình 4. 12 Hình ảnh một API chatbot có tên là Harley the Chatbot

- Nhấn nút thanh toán (Pricing) và chọn chế độ phù hợp để đăng ký (Subscribe).



Hình 4. 13 Hình ảnh giá thành và số lượng sử dụng của API Harley the Chatbot

- Thực hiện tương tự để đăng ký sử dụng API của Microsoft Translator Text dùng để dịch ngôn ngữ tiếng Anh về tiếng Việt.

Hình 4. 14 Giá thành API Microsoft Translator Text trên Rapid API

Bước 3: Sau khi chọn và đăng ký API thành công, tiến hành copy đoạn code python request tương ứng với mỗi API ở trang web Rapid API.

Bước 4: Thực hiện kết hợp hai API trình bày ở các bước trên lại với nhau để xây dựng chatbot.

Đầu tiên sẽ đưa câu hỏi chatbot vào trong API chatbot (Harley the Chatbot) để xử lý. Tiếp theo sau khi nhận phản hồi sẽ chuyển sang tiếng Việt bằng API Microsoft Translator Text.

```

1 import json
2 import http.client
3 import requests
4
5 def main(question):
6     conn_bot = http.client.HTTPSConnection("harley-the-chatbot.p.rapidapi.com")
7     payload_bot = {
8         "client": "",
9         "bot": "harley",
10        "message": question
11    }
12    headers_bot = {
13        'content-type': "application/json",
14        'Accept': "application/json",
15        'X-RapidAPI-Key': "744560b352msh56a291550432aeeplc01d7jsn9226c469da4c",
16        'X-RapidAPI-Host': "harley-the-chatbot.p.rapidapi.com"
17    }
18    conn_bot.request("POST", "/talk/bot", json.dumps(payload_bot), headers_bot)
19
20    res_bot = conn_bot.getresponse()
21    data_bot = res_bot.read().decode("utf-8")
22    response_data_bot = json.loads(data_bot)
23    # Extract and print the 'output' field from chatbot response
24    if 'data' in response_data_bot and 'conversation' in response_data_bot['data']:
25        output_from_bot = response_data_bot['data'][0]['conversation'][0]['output']
26        url = "https://microsoft-translator-text.p.rapidapi.com/translate"
27
28        querystring = {
29            "to": "vi",
30            "api-version": "3.0",
31            "profanityAction": "NoAction",
32            "textType": "plain"
33        }
34
35        payload = [{"Text": output_from_bot}]
36
37        headers = {
38            'content-type': "application/json",
39            'X-RapidAPI-Key': "744560b352msh56a291550432aeeplc01d7jsn9226c469da4c",
40            'X-RapidAPI-Host': "microsoft-translator-text.p.rapidapi.com"
41        }
42
43        response = requests.post(url, json=payload, headers=headers, params=querystring)
44
45        if response.ok:
46            data = response.json()
47            if isinstance(data, list) and 'translations' in data[0] and isinstance(data[0]['translations'], list):
48                translated_text = data[0]['translations'][0]['text']
49                return translated_text
50            else:
51                return "Unable to extract 'text' from the translation response."
52        else:
53            return "Translation request failed with status code:", response.status_code
54    else:
55        return "Output field not found in the chatbot response."
56

```

Bảng 4. 3 Đoạn code xây dựng tích hợp 2 API trong Rapid API

Kết quả trả về khi thực hiện đoạn code trên được thể hiện như hình bên dưới(Hình 4.15)

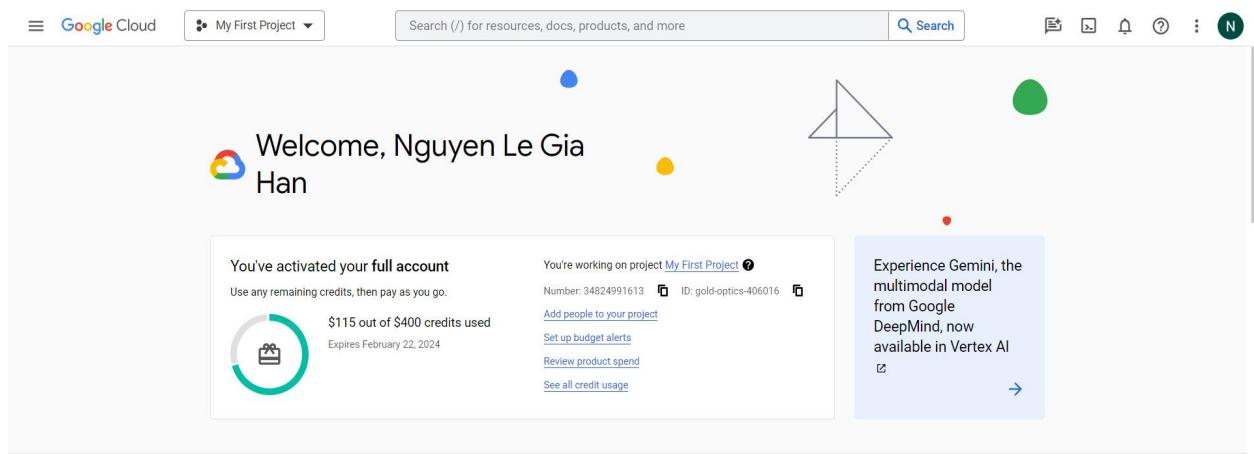
```
D:\11\20133>"C:/Program Files/Python311/python.exe" d:/11/20133/Demo/demo3.py
Translated Text: Tỉnh Phú Yên là một tỉnh xinh đẹp và danh lam thắng cảnh nằm ở Việt Nam. Nó được biết đến với bờ biển tuyệt đẹp, phong cảnh đẹp như tranh vẽ và di sản văn hóa phong phú. Nếu bạn có cơ hội đến thăm tỉnh Phú Yên, hãy chắc chắn khám phá những bãi biển hoang sơ của nó, chẳng hạn như Rạn san hô Đá Đèo và Bãi Xếp. Bạn cũng có thể ghé thăm các di tích lịch sử như Tháp Nhàn và Đầm Ô Loan. Tỉnh Phú Yên mang đến sự pha trộn hoàn hảo giữa vẻ đẹp tự nhiên và trải nghiệm văn hóa.

D:\11\20133>
```

Hình 4. 15 Kết quả trả về từ đoạn code tích hợp 2 API

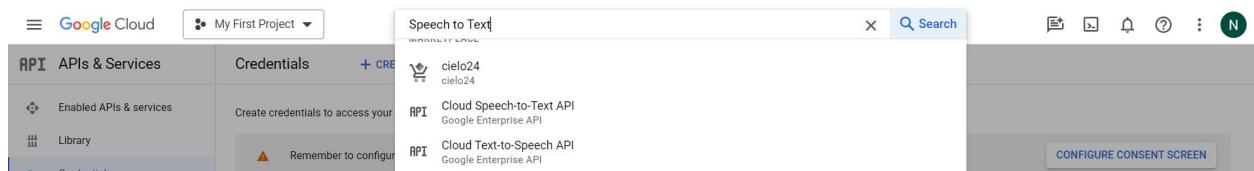
4.2.4. Google Cloud API

Bước 1: Vào trang Google Cloud Platform Console ([Link](#)), khi đăng nhập thành công có giao diện như hình dưới.(Lưu ý phải đăng ký tài khoản để sử dụng 300-400\$ miễn phí cho các dịch vụ sử dụng.)



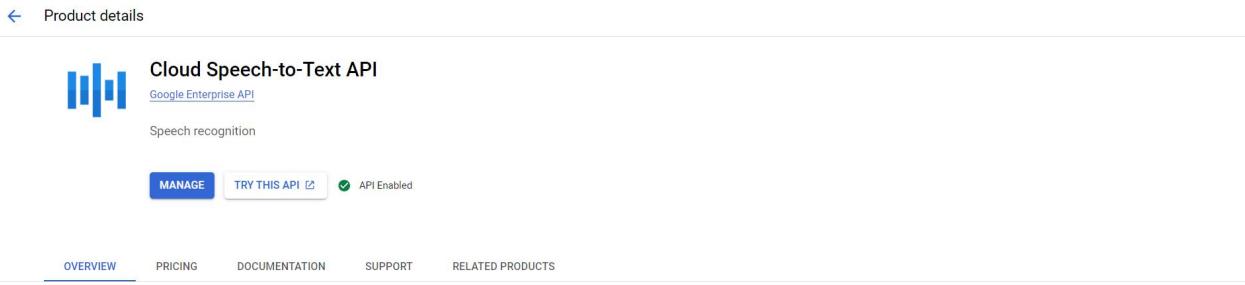
Hình 4. 16 Hình ảnh trang Google Cloud Platform

Bước 2: Nhập vào ô tìm kiếm nội dung là ‘Speech to text’ hoặc ‘Text to speech’.



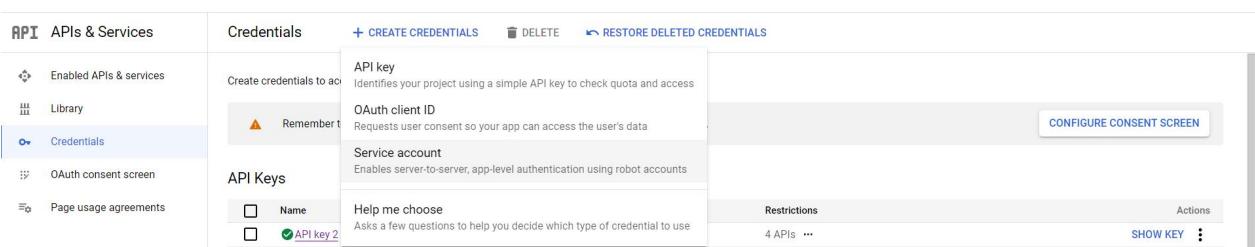
Hình 4. 17 Kết quả trả về từ ô tìm kiếm của Google Cloud

Sau khi có kết quả trả về từ tìm kiếm → chọn Cloud Speech-to-Text API và Cloud Text-to-Speech API và chọn Enable để kích hoạt API.



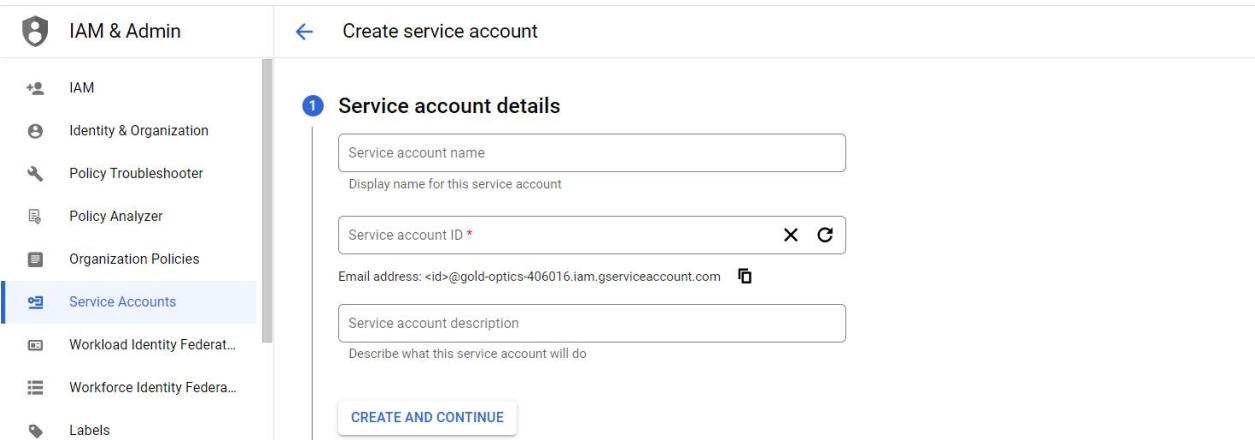
Hình 4. 18 Hình ảnh Cloud Speech-to-Text API trong Google Cloud

Bước 3: Tạo chứng chỉ Credentials cho tài khoản Google Cloud để sử dụng cho các tác vụ liên quan đến API Speech-to-Text và Text-to-Speech.
Chọn APIs & Services → Chọn CREATE CREDENTIALS.



Hình 4. 19 Cửa sổ xuất hiện khi kích vào create credentials

Chọn Service Account xuất hiện như hình dưới (Hình 4.20).



Hình 4. 20 Hình ảnh thông tin của Service account details

Nhập thông tin cần thiết vào như Service account details và Service account ID.

1 Service account details

Service account name: apigoole1

Display name for this service account

Service account ID *: apigoole1

Email address: apigoole1@gold-optics-406016.iam.gserviceaccount.com

Service account description

Describe what this service account will do

CREATE AND CONTINUE

Hình 4. 21 Nhập thông tin vào Service account details

Sau khi nhập xong chọn **CREATE AND CONTINUE** xuất hiện ra bước 2 như hình bên dưới(Hình 4.22).

2 Service account details

Grant this service account access to project (optional)

Grant this service account access to My First Project so that it has permission to complete specific actions on the resources in your project. [Learn more](#)

Select a role

IAM condition (optional) [?](#)

+ ADD IAM CONDITION

+ ADD ANOTHER ROLE

CONTINUE

3 Grant users access to this service account (optional)

DONE CANCEL

Hình 4. 22 Bước tiếp theo khi chọn CREATE AND CONTINUE

Tiếp theo tiến hành lựa chọn Role và nhấn Done. Ở đây tiến hành chọn Role là Owner để chính bản thân sử dụng các API đó.

Service account details

Grant this service account access to project (optional)

Role: Owner

IAM condition (optional) [?](#) [+ ADD IAM CONDITION](#)

Full access to most Google Cloud resources. See the list of included permissions.

[+ ADD ANOTHER ROLE](#)

Hình 4. 23 Hình ảnh chọn Role trong Service account

Nhấn Done ở nút bên dưới, kết quả trả về xem trong Hình 4.24.

	Email	Name	Actions
<input type="checkbox"/>	apigoogle@gold-optics-406016.iam.gserviceaccount.com	apigoogle	
<input type="checkbox"/>	apigoogle1@gold-optics-406016.iam.gserviceaccount.com	apigoogle1	
<input type="checkbox"/>	34824991613-compute@developer.gserviceaccount.com	Compute Engine default service account	
<input type="checkbox"/>	speech-demo123@gold-optics-406016.iam.gserviceaccount.com	speech-demo123	

Hình 4. 24 Hình ảnh chi tiết Service Account được tạo thành công

Kết quả sau khi tạo xong nhấn vào email mới tạo có tên Name là apigoogle1.

IAM & Admin

Service account details

Name: apigoogle1

Description: (empty)

Email: apigoogle1@gold-optics-406016.iam.gserviceaccount.com

Unique ID: 115347316602182048336

Service account status

Enabled

[DISABLE SERVICE ACCOUNT](#)

Advanced settings

Hình 4. 25 Hình ảnh bên trong một service account

Chọn Keys sẽ xuất hiện khung làm việc.

Hình 4. 26 Hình ảnh key của Service account

Hiện tại chưa có key nào nên nhấn vào ADD KEY → chọn Create new Key.

Hình 4. 27 Chọn kiểu trả về Credentials của Service Account

Chọn hình thức Key là JSON và nhấn Create, một file Json sẽ được tự động tải về khi thực hiện thao tác.

Hình 4. 28 File Json được tải về

Bước 4: Thiết lập đoạn code Python sử dụng Speech-to-Text và Text-to-Speech và xây dựng chatbot sử dụng Google Bard.

- Tải các thư viện cần thiết để xây dựng.

pip install pyaudio wave google google-cloud-texttospeech google-cloud-speech pygame

- Xây dựng đoạn code sử dụng Speech-to-Text với đặt cứng cho thời gian ghi âm là 15s.

```

1 import pyaudio
2 import wave
3 from google.cloud import speech_v1p1beta1 as speech
4 from google.oauth2 import service_account
5 import os
6 import io
7
8 def record_and_transcribe():
9     # Thông số ghi âm
10    FORMAT = pyaudio.paInt16
11    CHANNELS = 1
12    RATE = 48000
13    CHUNK = 1024
14    RECORD_SECONDS = 15
15    WAVE_OUTPUT_FILENAME = "output.wav"
16    GOOGLE_CLOUD_KEY_PATH = r'GoogleCloudKey_MyServiceAcct.json'
17    p = pyaudio.PyAudio()
18    stream = p.open(format=FORMAT,
19                      channels=CHANNELS,
20                      rate=RATE,
21                      input=True,
22                      frames_per_buffer=CHUNK)
23    print("Recording...")
24    frames = []
25    for i in range(0, int(RATE / CHUNK * RECORD_SECONDS)):
26        data = stream.read(CHUNK)
27        frames.append(data)
28    print("Finished recording.")
29    # Dừng ghi âm
30    stream.stop_stream()
31    stream.close()
32    p.terminate()
33    wf = wave.open(WAVE_OUTPUT_FILENAME, 'wb')
34
35    wf.setnchannels(CHANNELS)
36    wf.setsampwidth(p.get_sample_size(FORMAT))
37    wf.setframerate(RATE)
38    wf.writeframes(b''.join(frames))
39    wf.close()
40    print(f"Recording saved as {WAVE_OUTPUT_FILENAME}")
41    client = speech.SpeechClient.from_service_account_file(GOOGLE_CLOUD_KEY_PATH)
42    with io.open(WAVE_OUTPUT_FILENAME, 'rb') as f:
43        audio_data = f.read()
44    audio_file = speech.RecognitionAudio(content=audio_data)
45    config = speech.RecognitionConfig(
46        sample_rate_hertz=RATE,
47        enable_automatic_punctuation=True,
48        language_code='vi-VN')
49    response = client.recognize(
50        config=config,
51        audio=audio_file)
52
53    transcripts = []
54    for result in response.results:
55        transcripts.append(result.alternatives[0].transcript)
56
57    # Xóa file ghi âm
58    # os.remove(WAVE_OUTPUT_FILENAME)
59    print(f"Recording file {WAVE_OUTPUT_FILENAME} deleted.")
60    print(transcripts)
61    return transcripts

```

Bảng 4. 4 Đoạn code sử dụng Speech-to-Text trong Google Cloud

- Xây dựng đoạn code Text-to-Speech trong python gồm có phần đọc âm thanh từ văn bản và dừng âm thanh đó.

```

1 import io
2 import os
3 from threading import Thread
4 import pygame
5 from google.cloud import texttospeech_v1
6 os.environ['GOOGLE_APPLICATION_CREDENTIALS'] = r'GoogleCloudKey_MyServiceAcct.json'
7 stop_audio_flag = False
8 immediately_stop_flag = False
9 audio_thread = None
10 pygame.mixer.init()
11
12 def play_audio_thread(sound):
13     global stop_audio_flag, immediately_stop_flag
14     pygame.mixer.music.load(io.BytesIO(sound))
15     pygame.mixer.music.play()
16     while not stop_audio_flag and not immediately_stop_flag:
17         pygame.time.Clock().tick(10)
18         if pygame.mixer.music.get_busy() == 0:
19             break
20     pygame.mixer.music.stop()
21     print("Audio playback stopped.")
22
23 def play_sound(audio_data):
24     global stop_audio_flag, audio_thread, immediately_stop_flag
25     audio_thread = Thread(target=play_audio_thread, args=(audio_data,))
26     audio_thread.start()
27     return audio_thread
28
29 def stop_sound():
30     global stop_audio_flag, audio_thread, immediately_stop_flag
31     stop_audio_flag = True
32     immediately_stop_flag = True
33     audio_thread.join(timeout=0)
34
35 def text_to_speech(text, language_code="vi-VN"):
36     client = texttospeech_v1.TextToSpeechClient()
37
38     synthesis_input = texttospeech_v1.SynthesisInput(text=text)
39     voice = texttospeech_v1.VoiceSelectionParams(
40         language_code=language_code,
41         name=f'{language_code}-Wavenet-D',
42         ssml_gender=texttospeech_v1.SsmlVoiceGender.NEUTRAL
43     )
44     audio_config = texttospeech_v1.AudioConfig(
45         audio_encoding=texttospeech_v1.AudioEncoding.LINEAR16
46     )
47     response = client.synthesize_speech(
48         input=synthesis_input, voice=voice, audio_config=audio_config
49     )
50     audio_data = response.audio_content
51     return audio_data
52
53 def play_text_to_speech(text, language_code="vi-VN"):
54     global stop_audio_flag, audio_thread, immediately_stop_flag
55     stop_audio_flag = False
56     immediately_stop_flag = False
57     audio_data = text_to_speech(text, language_code)
58     audio_thread = play_sound(audio_data)
59     return audio_thread
60
61 def stop_text_to_speech():
62     global stop_audio_flag, audio_thread, immediately_stop_flag
63     stop_sound()
64     return audio_thread
65
66 def stop_stt():
67     global stop_audio_flag, immediately_stop_flag
68     stop_audio_flag = True # Dừng âm thanh nếu đang phát
69     immediately_stop_flag = True
70     stop_text_to_speech()
71     return True

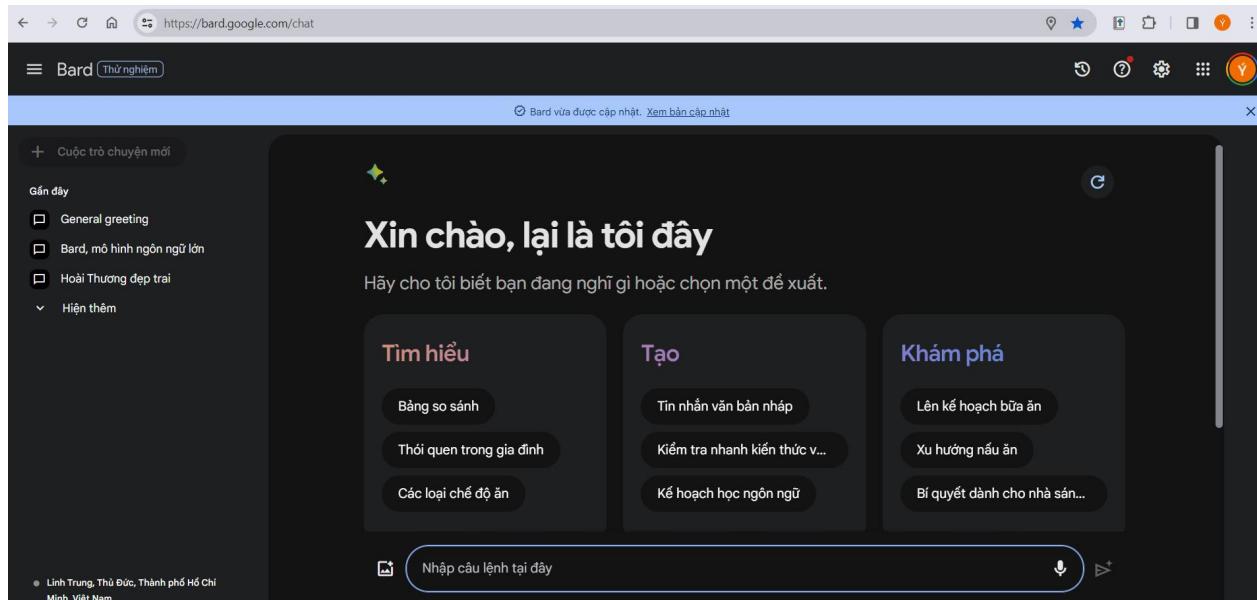
```

Hình 4. 29 Đoạn code sử dụng Text-to-Speech trong Google Cloud

- Cài đặt chatbot từ Google Bard API

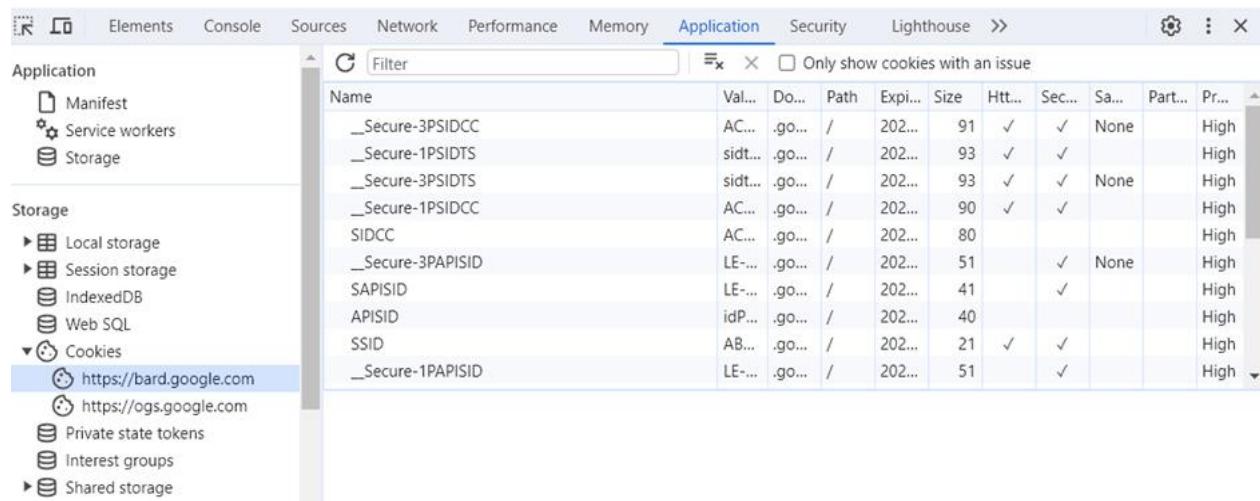
Bước 1: Vào trang Bard của Google theo đường dẫn [tại đây](https://bard.google.com/chat).

- Khi đăng nhập vào giao diện của Bard như hình dưới (Xem hình 4.30).



Hình 4. 30 Trang giao diện chính của Google Bard

Bước 2: Nhấn phím F12 trên trình duyệt → Vào cửa sổ DevTool → Vào Application chọn Cookies → chọn <https://bard.google.com>



Hình 4. 31 Hình ảnh giao diện cửa sổ DevTool của Google Bard

Bước 3: Lấy các thông tin về _Secure-1PSID, _Secure-1PSIDTS, __Secure-1PSIDCC

Name	Value	Domain	Path	Expiry	Size	HTTP	Security	SameSite	Partition	Priority
_Secure-3PSIDCC	AC...	.go...	/	202...	91	✓	✓	None	High	
_Secure-1PSIDTS	sidt...	.go...	/	202...	93	✓	✓	None	High	
_Secure-3PSIDTS	sidt...	.go...	/	202...	93	✓	✓	None	High	
<u>_Secure-1PSIDCC</u>	<u>AC...</u>	<u>.go...</u>	<u>/</u>	<u>202...</u>	<u>90</u>	<u>✓</u>	<u>✓</u>		<u>High</u>	
SIDCC	AC...	.go...	/	202...	80				High	
_Secure-3PAPISID	LE...	.go...	/	202...	51		✓	None	High	
SAPISID	LE...	.go...	/	202...	41		✓		High	
APISID	idP...	.go...	/	202...	40				High	
SSID	AB...	.go...	/	202...	21	✓	✓		High	
_Secure-1PAPISID	LE...	.go...	/	202...	51		✓		High	

Hình 4. 32 Lấy thông tin các thuộc tính Cookies của Google Bard

Bước 4: Xây dựng đoạn code demo Google Bard trên giao diện Streamlit.

```

1 import os
2 import streamlit as st
3 from bardapi import Bard
4 os.environ['_BARD_API_KEY'] = "cgj0KgkF5qUxhJNoy0HWGAPHON0SP06rC60jDKfrkaLrmf7N8HT1fKBWJqywHiwfRd_qQQ."
5
6 def get_bard_response(question):
7     bard = Bard()
8     return bard.get_answer(question)['content']
9
10 def update_chat_history(user_question, chat_history):
11     chat_history.append(f"You: {user_question}")
12     bard_response = get_bard_response(user_question)
13     chat_history.append(f"Bot: {bard_response}")
14     return chat_history
15
16 session_state = st.session_state
17 if 'chat_history' not in session_state:
18     session_state.chat_history = []
19 st.title("Bard API Chat")
20 user_question = st.text_input("Enter your question", "")
21
22 if st.button("Get Answer") and user_question:
23     session_state.chat_history = update_chat_history(user_question, session_state.chat_history)
24
25 chat_history = '\n'.join(session_state.chat_history)
26 st.markdown(f"<style>.reportview-container .main .block-container{{width: 1237px}}</style>", unsafe_allow_html=True)
27 st.text_area("Chat History", value=chat_history, height=485)

```

Hình 4. 33 Đoạn code sử dụng Google Bard trên Streamlit

Bước 5: Mở terminal và gõ lệnh: `python -m streamlit run <name_app>.py`

Bước 6: Sau khi chạy đoạn lệnh xong giao diện Streamlit sẽ mở trên web, nếu giao diện không mở được thì thực hiện bằng cách nháy chuột vào đường dẫn local.

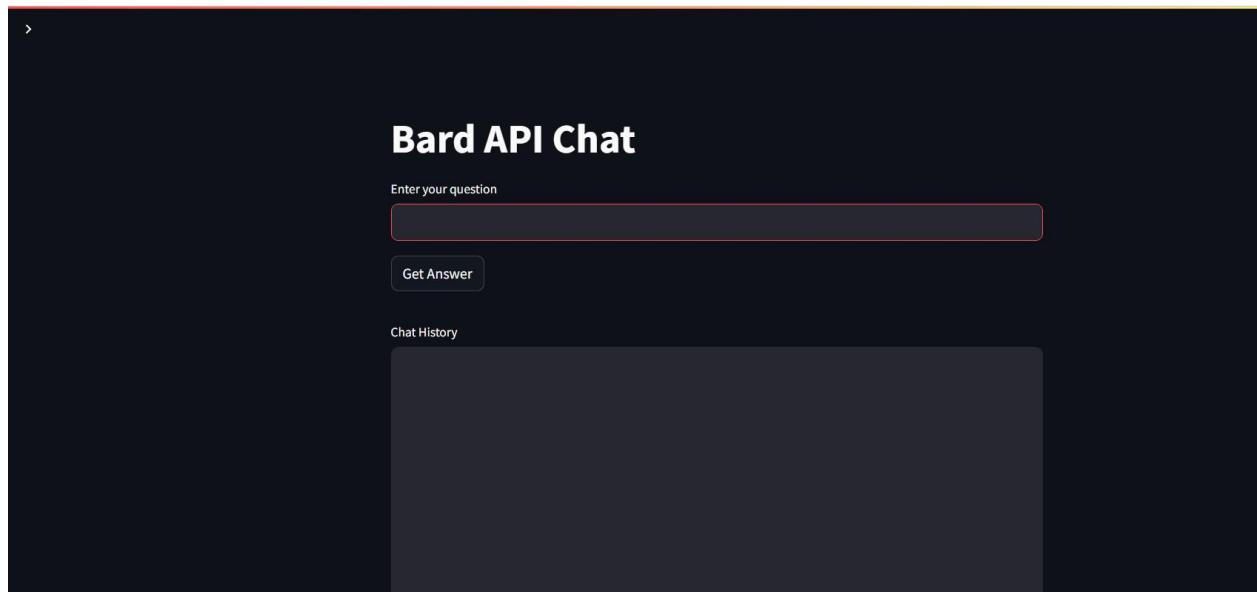
```
D:\11\20133>python -m streamlit run demo-chatbot.py

You can now view your Streamlit app in your browser.

Local URL: http://localhost:8501
Network URL: http://192.168.10.107:8501
```

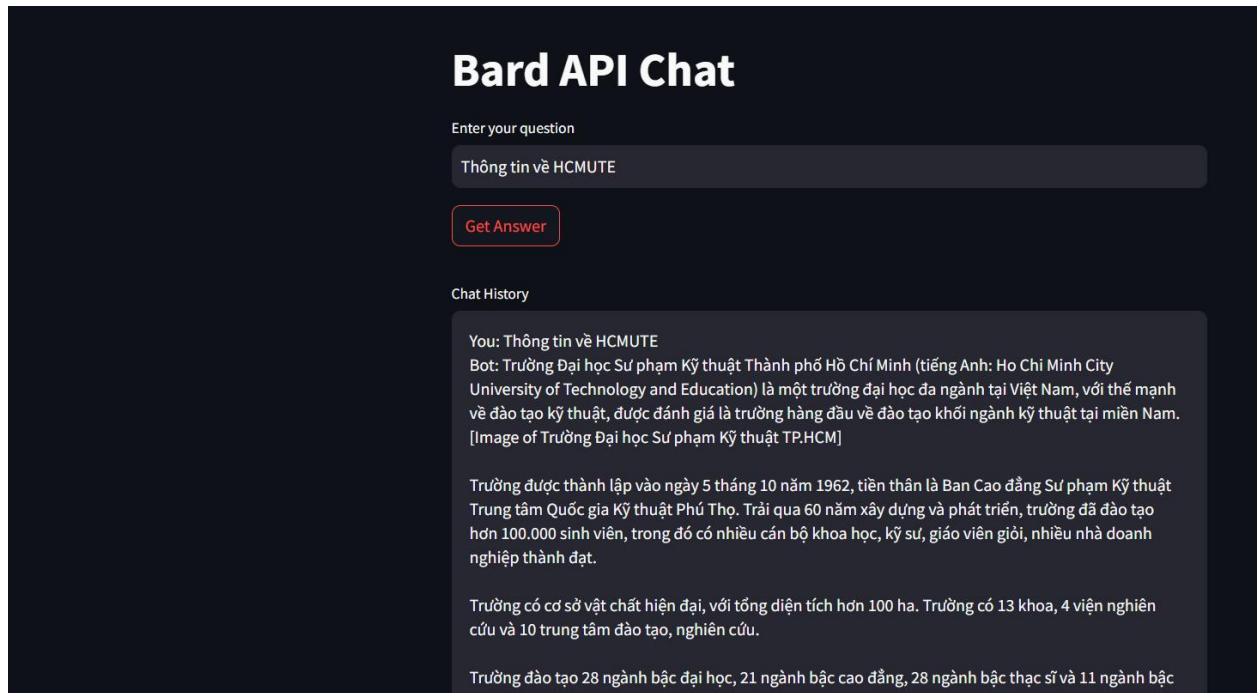
Hình 4. 34 Kết quả khi chạy câu lệnh run Streamlit

Hình ảnh giao diện sau khi chạy hoàn thành:



Hình 4. 35 Giao diện Streamlit khi sử dụng Google Bard

Để sử dụng tiến hành nhập câu hỏi vào thanh nhập câu hỏi, sau đó nhấn vào Get Answer và đợi kết quả. Kết quả chạy thử Google Bard trả về xuất hiện như trong hình bên dưới (Xem Hình 4.36).



Hình 4. 36 Kết quả khi sử dụng Bard API

4.2.5. So sánh các API và thư viện hỗ trợ xây dựng Chatbot

Dưới đây là bảng so sánh các API và thư viện hỗ trợ xây dựng chatbot (Xem bảng 4.5).

Tiêu chí/ Thư viện	Thư viện Transformers	Google Cloud API	OpenAI API	Rapid API
Nhà cung cấp	Hugging Face	Google	OpenAI	Rapid
Độ khó/dễ khi cài đặt	Trung bình	Trung bình	Dễ	Dễ
Tốc độ câu trả lời	Tùy thuộc vào cấu hình máy tính với máy tính có bộ xử lý, cấu hình mạnh thì mô hình có thể trả lời nhanh hơn.	Không có(sử dụng Cloud speech to text và Text to speech).	Tốc độ câu trả lời nhanh(Google Bard)	Tùy thuộc vào loại API sử dụng nhưng chậm hơn so với OpenAI và Google Bard. Do phải sử dụng qua trung gian.
Chất lượng câu trả lời	Chất lượng câu trả lời ở mức	Không có đánh giá(Text to speech)	Chất lượng câu trả lời ở mức tốt. Dữ	Chất lượng câu trả lời ở mức khá.

	trung bình	và speech to text) Chất lượng câu trả lời ở mức tốt(Google Bard). Dữ liệu trả lời được cập nhật tới thời điểm hiện tại.	liệu trả lời chỉ cập nhật đến tháng 1/2022.	
Chi phí sử dụng	Không tốn chi phí	Giá thành là 0,16\$ cho 1 triệu đơn vị của Text-to-Speech. Có giá thành 0.15\$ cho 1 triệu đơn vị của Speech-to-Text. Không tốn chi phí(Google Bard).	Mô hình Gpt-3.5-turbo-instruct có giá là 0,0015/1k token input và giá \$0.0020/1K tokens cho output. Mô hình Gpt-3.5-turbo-1106 có giá tương tự cho phần output nhưng giá input là 0,0010/1k tokens.	Chi phí tùy thuộc loại API sử dụng thông thường các API sẽ miễn phí với số lượt sử dụng là 20-50 lượt tháng. Để nâng lên các mức cao hơn phải tốn chi phí từ 5-10\$ trở lên.
Có hỗ trợ tinh chỉnh mô hình	Có hỗ trợ các tinh chỉnh mô hình như PhoBERT, XLM-R,...	Không có	Có hỗ trợ tinh chỉnh mô hình như gpt-3.5-turbo, davinci-002, babbage-002, gpt-3.5-turbo-1106	Không có

Độ khó/dễ khi tinh chỉnh mô hình	Khó	Không có	Trung bình	Không có
Chi phí tinh chỉnh mô hình	Không tồn chi phí	Không có	Đối với mô hình gpt-3.5-turbo chi phí đào tạo là 0.008\$/1k token. Dữ liệu sử dụng nhập vào là 0.003\$/1k token và 0.006\$/1k token cho việc sử dụng mô hình đã tinh chỉnh. Mô hình davinci- 002 có giá thành là 0.006\$ cho 1k token training, giá là 0,012\$/1k token cho cả phần input và output khi sử dụng mô hình đã tinh chỉnh.	Không có

Bảng 4. 5 So sánh API và thư viện hỗ trợ xây dựng Chatbot

4.3. Thực hiện tinh chỉnh mô hình Chatbot

4.3.1. Môi trường cài đặt

- Sử dụng Python phiên bản 3.11.2 trên phần mềm Visual Studio Code phiên bản mới nhất (2023) và Python 3.10 trên phiên bản Google Colab.
- Sử dụng T4 GPU của Google Colab thực hiện quá trình Fine Tune cho PhoBERT, XLM-R.
- Sử dụng máy tính cá nhân để train, sử dụng mô hình, với cấu hình: CPU i5-11400H, RAM 16GB.

NVIDIA-SMI 535.104.05			Driver Version: 535.104.05		CUDA Version: 12.2				
GPU	Name	Persistence-M	Bus-Id	Disp.A	Volatile	Uncorr. ECC			
Fan	Temp	Perf	Pwr:Usage/Cap	Memory-Usage	GPU-Util	Compute M.			
0	Tesla T4	Off	00000000:00:04.0	Off	0%	Default	0		
N/A	33C	P8	9W / 70W	0MiB / 15360MiB			N/A		
====									
Processes:									
GPU	GI	CI	PID	Type	Process name	GPU Memory Usage			
ID		ID							
====									
No running processes found									

Hình 4. 37 Cấu hình T4 GPU trên Google Colab

4.3.2. Chuẩn bị dữ liệu

4.3.2.1. Mô hình PhoBERT

Sử dụng tập dữ liệu nhóm tự xây dựng về trường gồm có 3307 câu hỏi (chứa câu hỏi thắc mắc về tư vấn, câu hỏi về môn học của trường, thông tin giảng viên,...), tập dữ liệu được định dạng Json theo hình dưới đây.

```

  "tag": "tag_without_accents",
  "patterns": [item["prompt"]],
  "responses": [item["response"]]

```

Hình 4. 38 Định dạng Json của tập dữ liệu huấn luyện trong PhoBERT

Trong đó tag là thẻ của tiêu đề ví dụ như “toan1”, pattern chứa danh sách câu hỏi và responses chứa danh sách câu trả lời (xem chi tiết [tại đây](#)).

Tập dữ liệu gồm có các tập tin là content.json dùng để thực hiện quá trình tinh chỉnh, val_content.json tập đánh giá và test_content.json tập kiểm tra.

4.3.2.2. Mô hình XLM-RoBERTa

Sử dụng tập dữ liệu UIT-ViSQua (xem chi tiết [tại đây](#)) có cấu trúc gồm 3 thành phần được định dạng ở Json.



Hình 4. 39 Minh họa tập dữ liệu UIT-ViSquad

Vì giới hạn tài nguyên của Google Colab nên chỉ có thể train khoảng tối đa 5 epoch cho mô hình XLM-R.

4.3.2.3. Mô hình OpenAI

Sử dụng tập dữ liệu trích xuất ban đầu về bộ dữ liệu của trường (chứa câu hỏi thắc mắc về tư vấn, câu hỏi về môn học của trường, câu hỏi về giảng viên,...) gồm có 1397 câu hỏi và tiến hành chọn lọc ngẫu nhiên ra 1000 câu hỏi (source: xem chi tiết [tại đây](#)) được định dạng dưới hình thức jsonl.

Hình 4. 40 Tập dữ liệu có định dạng jsonl được dùng huấn luyện trong OpenAI

4.3.3 Xử lý dữ liệu đầu vào

4.3.3.1. Mô hình PhoBERT

- Xây dựng hàm xử lý tập dữ liệu training.

```
# Process the train dataset:
tags = []
x = []
y = []

for content in contents['intents']:
    tag = content['tag']
    for pattern in content['patterns']:
        x.append(pattern)
        tags.append(tag)

tags_set = sorted(set(tags))

for tag in tags:
    label = tags_set.index(tag)
    y.append(label)
token_train = {}
token_train = tokenizer.batch_encode_plus(
    x,
    max_length=13,
    padding='max_length',
    truncation=True
)
x_train_mask = torch.tensor(token_train['attention_mask'])
X_train = torch.tensor(token_train['input_ids'])
y_train = torch.tensor(y)
```

Hình 4.41 Xây dựng hàm xử lý tập training của PhoBERT

- Xây dựng hàm xử lý tập dữ liệu đánh giá.

```

# Process the validation dataset:
tags_val = []
X_val = []
y_val = []

for val_content in val_contents['intents']:
    tag_val = val_content['tag']
    for val_pattern in val_content['patterns']:
        X_val.append(val_pattern)
        tags_val.append(tag_val)

for tag_val in tags_val:
    label = tags_set.index(tag_val)
    y_val.append(label)
    token_val = {}
    token_val = tokenizer.batch_encode_plus(
        X_val,
        max_length=256,
        padding='max_length',
        truncation=True
    )
    X_val_mask = torch.tensor(token_val['attention_mask'])
    X_val = torch.tensor(token_val['input_ids'])
    y_val = torch.tensor(y_val)

```

Hình 4. 42 Xây dựng hàm xử lý dữ liệu của tập validation test

4.3.3.2. Mô hình XLM-RoBERTa

- Khởi tạo phương thức init.

```

class Create_datasetDict:
    def __init__(self, dataset):
        self.dataset = dataset

    def preprocessing(self):
        contexts = []
        questions = []
        answers = []
        ids = []
        for data in self.dataset['data']:
            for para in data['paragraphs']:
                context = para['context']
                for qa in para['qas']:
                    question = qa['question']
                    for answer in qa['answers']:
                        contexts.append(context)
                        questions.append(question)
                        answers.append({'answer_start': [answer['answer_start']], 'text': [answer['text']]})
                        ids.append(qa['id'])
        icqa = [ids, contexts, questions, answers]
        return icqa

```

Hình 4. 43 Xây dựng phương thức init trong mô hình XLM-R

- Xây dựng phương thức call .

```

def __call__(self, test_size):
    if self.dataset is None:
        # Load your dataset here if needed
        pass

    icqa = self.preprocessing()
    data = {'id': icqa[0], 'context': icqa[1], 'question': icqa[2], 'answer': icqa[3]}

    # Building dataset with format same as SQuAD 2.0
    df = pd.DataFrame(data=data)
    train_df, valid_df = train_test_s (variable) train_df: Any ze, random_state=1, shuffle=True)

    train_dict = Dataset.from_pandas(train_df)
    valid_dict = Dataset.from_pandas(valid_df)
    dataset_dict = DatasetDict({'train': train_dict, 'validation': valid_dict})
    return dataset_dict

```

Hình 4. 44 Phương thức call trong class Create_datasetDict

- Xây dựng Class Features để xử lý dữ liệu gồm có các hàm train_processing, valid_processing.

```

class Features:
    def __init__(self, tokenizer, max_length, stride):
        self.max_length = max_length
        self.stride = stride
        self.tokenizer = tokenizer

    def train_processing(self, dataset):
        questions = [q.strip() for q in dataset['question']]
        inputs = self.tokenizer(questions, dataset['context'],
                               max_length = self.max_length,
                               truncation = 'only_second',
                               stride = self.stride,
                               return_overflowing_tokens = True,
                               return_offsets_mapping = True,
                               padding = 'max_length')
        # Start char and end char of each token
        offset_mapping = inputs.pop('offset_mapping')
        sample_map = inputs.pop('overflow_to_sample_mapping')
        answers = dataset['answer']
        start_positions = []
        end_positions = []

        for i, offset in enumerate(offset_mapping):
            sample_idx = sample_map[i]
            answer = answers[sample_idx]

```

```

start_char = answer['answer_start'][0]
end_char = start_char + len(answer['text'][0])
sequence_ids = inputs.sequence_ids(1)

idx = 0
while sequence_ids[idx] != 1:
    idx += 1
# Index of the start token of the context
context_start = idx

while sequence_ids[idx] == 1:
    idx += 1
# Index of the end token of the context
context_end = idx - 1

# Create label
if offset[context_start][0] > start_char or offset[context_end][1] < end_char:
    start_positions.append(0)
    end_positions.append(0)
else:
    idx = context_start
    while idx <= context_end and offset[idx][0] <= start_char:
        idx += 1
    start_positions.append(idx - 1)

    idx = context_end
    while idx >= context_start and offset[idx][1] >= end_char:
        idx -= 1
    end_positions.append(idx + 1)

inputs['start_positions'] = start_positions
inputs['end_positions'] = end_positions

return inputs

def valid_processing(self, dataset):
    questions = [q.strip() for q in dataset['question']]
    inputs = self.tokenizer(questions, dataset['context'],
                           max_length = self.max_length,
                           truncation = 'only_second',
                           stride = self.stride,

```

```

        return_overflowing_tokens = True,
        return_offsets_mapping = True,
        padding = 'max_length')

sample_map = inputs.pop('overflow_to_sample_mapping')
example_ids = []
for i in range(len(inputs['input_ids'])):
    sample_idx = sample_map[i]
    example_ids.append(dataset['id'][sample_idx])

    sequence_ids = inputs.sequence_ids(i)
    offset = inputs['offset_mapping'][i]
    inputs['offset_mapping'][i] = [j if sequence_ids[k] == 1 else None for k, j in
enumerate(offset)]

inputs['example_id'] = example_ids
return inputs

```

Bảng 4. 6 Xây dựng Class Features

4.3.4. Xây dựng mô hình

4.3.4.1. Mô hình PhoBERT

- Xây dựng file phobert_finetune.py trong đó xây dựng class PhoBert_finetuned gồm có phương thức khởi tạo(init) và forward.
- Trong đó gồm có các thuộc tính:
 - + phobert: Mô hình PhoBERT (hoặc một mô hình BERT tương tự) được chuyển vào để sử dụng trong lớp PhoBERT_finetuned.
 - + hidden_size: Kích thước của không gian biểu diễn ẩn (hidden representation).
 - + num_class: Số lượng lớp đầu ra (số lượng lớp trong bài toán phân loại).
 - + sent_id: Là đầu vào của mô hình, biểu diễn các token của câu dưới dạng một tensor.
 - + mask: Một tensor đánh dấu các vị trí có giá trị 1 cho các token thực sự và 0 cho các vị trí padding. Được sử dụng để ẩn các vị trí padding trong quá trình tính toán.
 - + self.phobert: tham số đầu vào của lớp, biểu diễn mô hình PhoBERT (hoặc một mô hình BERT tương tự).
 - + self.relu: Một lớp ReLU (Rectified Linear Unit), được sử dụng để áp dụng hàm kích hoạt ReLU cho đầu ra của lớp tuyến tính self.layer1

- + self.dropout: Một lớp dropout với tỷ lệ dropout là 0.1. Lớp này được sử dụng để ngẫu nhiên "tắt" một số đơn vị trong quá trình huấn luyện, giúp chống lại overfitting.
- + self.layer2: Một lớp tuyến tính khác chuyển đổi không gian biểu diễn có kích thước hidden_size thành không gian biểu diễn có kích thước num_class.
- + self.softmax: Một lớp log-softmax, áp dụng để đầu ra cuối cùng để chuyển đổi thành xác suất và tích log của nó.

```

import torch.nn as nn

class PhoBERT_finetuned(nn.Module):
    def __init__(self, phobert, hidden_size, num_class):
        super(PhoBERT_finetuned, self).__init__()
        self.phobert = phobert
        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(0.1)
        self.layer1 = nn.Linear(768, hidden_size)
        self.layer2 = nn.Linear(hidden_size, num_class)
        self.softmax = nn.LogSoftmax(dim=1)
    def forward(self, sent_id, mask):
        _, cls_hs = self.phobert(sent_id, attention_mask=mask,
                               return_dict=False)
        x = self.layer1(cls_hs)
        x = self.relu(x)
        x = self.dropout(x)
        x = self.layer2(x)
        x = self.softmax(x)
        return x

```

Bảng 4. 7 Xây dựng class PhoBERT_finetuned

- Thực hiện tải tập dữ liệu đào tạo, tập dữ liệu đánh giá, tải mô hình và tokenizer của chính mô hình phobert-base.

```

# Load train and validation dataset
with open('content.json', 'r', encoding="utf-8") as c:
    contents = json.load(c)
with open('val_content.json', 'r', encoding="utf-8") as v:
    val_contents = json.load(v)
# Load model PhoBERT and its tokenizer
phobert = AutoModel.from_pretrained('vinai/phobert-base')
tokenizer = AutoTokenizer.from_pretrained('vinai/phobert-base')

```

Hình 4. 45 Tải dữ liệu tập đánh giá và tập đào tạo trong PhoBERT

- Tạo tập train_data và val_data để đánh giá dữ liệu trong quá trình huấn luyện.
- Thiết lập các siêu tham số(hyperparameter).

```
# Hyperparameter:
batch_size = 8 # Cần tùy chỉnh dựa trên tài nguyên GPU và kích thước dữ liệu
hidden_size = 512 # Tăng lên nếu mô hình quá đơn giản
num_class = len(tags_set) # Số lớp trong bài toán phân loại
lr = 1e-5 # Cần tinh chỉnh, thử các giá trị như 1e-3, 5e-5, 1e-5
num_epoch = 300 # Số lượng epochs, điều chỉnh dựa trên việc mô hình còn cần học hay không

dataset = TensorDataset(x_train, x_train_mask, y_train)
train_data = DataLoader(dataset=dataset, batch_size=batch_size,
                           shuffle=True)
val_dataset = TensorDataset(x_val, x_val_mask, y_val)
val_data = DataLoader(dataset=val_dataset, batch_size=batch_size,
                           shuffle=True)
```

Hình 4. 46 Thiết lập các tham số, tập train_data và val_data trong mô hình PhoBERT

- Xây dựng mô hình truyền vào các tham số như số lượng class, kích thước ẩn.
- Sử dụng AdamW để tối ưu hóa tham số và learning rate.

```
# Model:
for param in phobert.parameters():
    param.requires_grad = False
model = PhoBERT_finetuned(phobert, hidden_size=hidden_size,
                           num_class=num_class)
model = model.to(device)
optimizer = AdamW(model.parameters(), lr=lr)
loss_f = nn.NLLLoss()
```

Hình 4. 47 Xây dựng mô hình PhoBERT và tối ưu hóa bằng cách dùng AdamW

- Xây dựng hàm train và hàm evaluate.

```

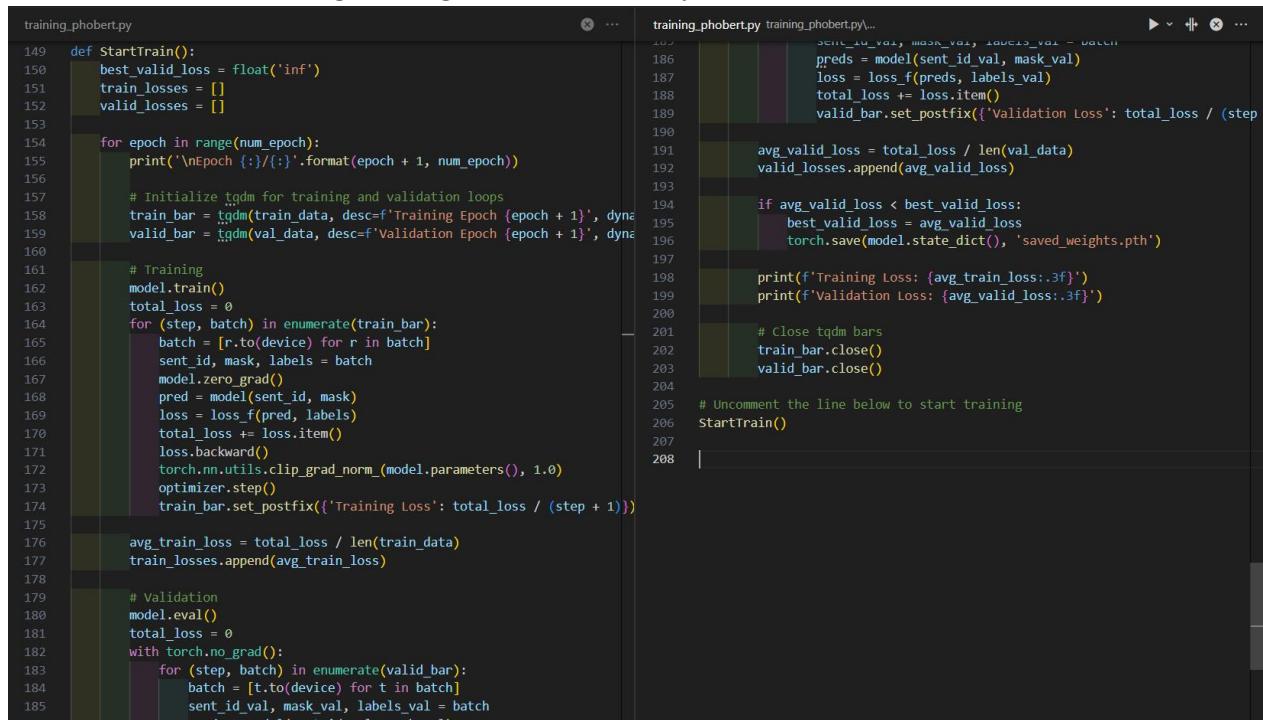
122 def evaluate():
123     print("Evaluating...")
124     # deactivate dropout layers
125     model.eval()
126     total_loss = 0
127     # iterate over batches
128     for (step, batch) in enumerate(val_data):
129         # push the batch to gpu
130         batch = [t.to(device) for t in batch]
131         sent_id_val, mask_val, labels_val = batch
132
133         # deactivate autograd
134         with torch.no_grad():
135             # model predictions
136             preds = model(sent_id_val, mask_val)
137             # compute the validation loss between a
138             loss = loss_fn(preds, labels_val)
139             total_loss += loss.item()
140             preds = preds.detach().cpu().numpy()
141
142             # compute the validation loss of the epoch
143             avg_loss = total_loss / len(val_data)
144
145     return avg_loss

146 def train():
147     print("Training...")
148     model.train()
149     total_loss = 0
150     for (step, batch) in enumerate(train_data):
151         # push the batch to gpu
152         batch = [r.to(device) for r in batch]
153         sent_id, mask, labels = batch
154         # clear previously calculated gradients
155         model.zero_grad()
156         pred = model(sent_id, mask)
157         loss = loss_fn(pred, labels)
158         total_loss += loss.item()
159         loss.backward()
160         # clip the gradients to 1.0.
161         # It helps in preventing the exploding gradient problem
162         torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)
163         optimizer.step()
164         avg_loss = total_loss / len(train_data)
165     return avg_loss

```

Hình 4. 48 Xây dựng hàm train và evaluate trong PhoBERT

- Xây dựng hàm StartTrain để bắt đầu training và hiển thị các giá trị mất mát trong đào tạo và đánh giá trong mỗi lần huấn luyện.



```

149 def StartTrain():
150     best_valid_loss = float('inf')
151     train_losses = []
152     valid_losses = []
153
154     for epoch in range(num_epoch):
155         print('Epoch {}/{}'.format(epoch + 1, num_epoch))
156
157         # Initialize tqdm for training and validation loops
158         train_bar = tqdm(train_data, desc=f'Training Epoch {epoch + 1}', dynamic_ncols=True)
159         valid_bar = tqdm(val_data, desc=f'Validation Epoch {epoch + 1}', dynamic_ncols=True)
160
161         # Training
162         model.train()
163         total_loss = 0
164         for (step, batch) in enumerate(train_bar):
165             batch = [r.to(device) for r in batch]
166             sent_id, mask, labels = batch
167             model.zero_grad()
168             pred = model(sent_id, mask)
169             loss = loss_fn(pred, labels)
170             total_loss += loss.item()
171             loss.backward()
172             torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)
173             optimizer.step()
174             train_bar.set_postfix({'Training Loss': total_loss / (step + 1)})
175
176             avg_train_loss = total_loss / len(train_data)
177             train_losses.append(avg_train_loss)
178
179         # Validation
180         model.eval()
181         total_loss = 0
182         with torch.no_grad():
183             for (step, batch) in enumerate(valid_bar):
184                 batch = [t.to(device) for t in batch]
185                 sent_id_val, mask_val, labels_val = batch
186
187                 preds = model(sent_id_val, mask_val)
188                 loss = loss_fn(preds, labels_val)
189                 total_loss += loss.item()
190
191                 avg_valid_loss = total_loss / len(val_data)
192                 valid_losses.append(avg_valid_loss)
193
194             if avg_valid_loss < best_valid_loss:
195                 best_valid_loss = avg_valid_loss
196                 torch.save(model.state_dict(), 'saved_weights.pth')
197
198             print(f'Training Loss: {avg_train_loss:.3f}')
199             print(f'Validation Loss: {avg_valid_loss:.3f}')
200
201             # Close tqdm bars
202             train_bar.close()
203             valid_bar.close()
204
205             # Uncomment the line below to start training
206             StartTrain()
207
208

```

Hình 4. 49 Xây dựng hàm StartTrain để bắt đầu training

4.3.4.2. Mô hình XLM-RoBERTa

- Khởi tạo model, tokenizer và thiết lập các tham số như hình bên dưới(Hình 4.50).
Trong đó :

- + test_size: Tỷ lệ dữ liệu được chia thành tập kiểm tra.
- + max_length: Độ dài từ tối đa của chuỗi đầu vào.

- + stride: bước trượt khi tạo các cửa sổ trượt cho dữ liệu đầu vào. Được sử dụng trong quá trình xử lý dữ liệu đầu vào, đặc biệt là khi sử dụng cửa sổ trượt để tạo các mẫu từ dữ liệu dài.
- + batch_size: kích thước của mỗi batch dữ liệu được sử dụng trong quá trình huấn luyện mô hình.
- + learning_rate: tỷ lệ học của thuật toán tối ưu hóa.
- + epochs: số lượng lần duyệt qua toàn bộ tập dữ liệu huấn luyện trong quá trình huấn luyện.
- + max_norm: giới hạn cho chuẩn của gradient. Nếu gradient vượt quá giới hạn này, chúng sẽ được điều chỉnh để không vượt quá giới hạn.
- + model_name_fine_tuned: Tên mô hình đã được finetune.
- + VERBOSE: một cờ để hiển thị thông tin chi tiết trong quá trình huấn luyện nếu được đặt thành True.

```

12 # xlm-roberta-base
13 model_checkpoint = 'bhavikardeshna/xlm-roberta-base-vietnamese'
14 prepare_model = XLMRobertaModel.from_pretrained(model_checkpoint)
15 tokenizer = XLMRobertaTokenizerFast.from_pretrained(model_checkpoint)
16 config = XLMRobertaConfig()
17 config.vocab_size = tokenizer.vocab_size
18
19 test_size = 0.06
20 max_length = 256
21 stride = 64
22 batch_size = 32
23 learning_rate = 2e-5
24 epochs = 3
25 max_norm = 1.0
26 model_name_fine_tuned = 'fine-tune-xlm-roberta-base-vietnamese-for-viqaquad'
27 VERBOSE = True

```

Hình 4. 50 Khởi tạo mô hình, tokenizer và thiết lập các tham số

- Khởi tạo các đối tượng Features để xây dựng tập dữ liệu đào tạo và đánh giá.

```

train_fn = Features.train_processing
valid_fn = Features.valid_processing
train_dataset = vn_qadts['train'].map(train_fn,
                                       batched = True,
                                       remove_columns = vn_qadts['train'].column_names)
valid_dataset = vn_qadts['validation'].map(valid_fn,
                                             batched = True,
                                             remove_columns = vn_qadts['validation'].column_names)

```

Hình 4. 51 Xây dựng tập train dataset và valid dataset trong XLM-R

- Thực hiện Config model và cấu hình các thuộc tính của args.
- Tham số trong cấu hình args:
 - + model_name_finetuned: Tên của mô hình hoặc đường dẫn đến mô hình đã được fine-tune.
 - + evaluation_strategy: Chiến lược đánh giá mô hình sau mỗi epoch.
 - + do_train: Chỉ định liệu có thực hiện quá trình huấn luyện hay không
 - + per_device_train_batch_size: Kích thước của mỗi batch dữ liệu trong quá trình huấn luyện
 - + per_device_eval_batch_size: Kích thước của mỗi batch dữ liệu trong quá trình đánh giá:
 - + learning_rate: Tỷ lệ học, tỷ lệ độ dời các trọng số của mô hình trong quá trình huấn luyện.
 - + weight_decay: Hệ số giảm trọng lượng (weight decay) được áp dụng cho các trọng số của mô hình để giảm nguy cơ overfitting.
 - + adam_beta1: Hệ số beta1 trong thuật toán tối ưu hóa Adam.
 - + adam_beta2: Hệ số beta2 trong thuật toán tối ưu hóa Adam.
 - + adam_epsilon: Giá trị epsilon được sử dụng để tránh chia cho 0 trong thuật toán tối ưu hóa Adam.
 - + max_grad_norm: Giới hạn giá trị của gradient để tránh exploding gradients.
 - + fp16: Sử dụng phép tính toán half-precision (float16) hay không.
 - + num_train_epochs: Số lược epochs
 - + logging_strategy: Chiến lược ghi log. Trong trường hợp này, log được ghi sau mỗi epoch.

```

QA_model = XLMRobertaForQuestionAnswering(prepare_model.config).from_pretrained(model_checkpoint)

# Fine-Tune with TrainingArguments and Trainer
args = transformers.TrainingArguments(model_name_fine_tuned,
                                       evaluation_strategy = 'no',
                                       do_train = True,
                                       per_device_train_batch_size = batch_size,
                                       per_device_eval_batch_size = batch_size,
                                       learning_rate = learning_rate,
                                       weight_decay = 0.01,
                                       adam_beta1 = 0.9,
                                       adam_beta2 = 0.999,
                                       adam_epsilon = 1e-8,
                                       max_grad_norm = max_norm,
                                       fp16 = True,
                                       num_train_epochs = epochs,
                                       logging_strategy = 'epoch')

```

Hình 4. 52 Thiết lập model và args trong mô hình XLM-R

- Tạo đối tượng trainer gồm có các tham số:
 - + model: Mô hình cần finetune.
 - + args: Cấu hình training được định nghĩa bởi đối tượng args ở trên.
 - + train_dataset: Tập dữ liệu huấn luyện dùng để tinh chỉnh mô hình.
 - + eval_dataset: Tập dữ liệu đánh giá dùng để đánh giá hiệu suất của mô hình.
 - + tokenizer: Sử dụng để chuyển đổi văn bản thành dạng số hóa mà mô hình có thể hiểu được và ngược lại tokenizer phải tương ứng với mô hình được sử dụng.

```

trainer = transformers.Trainer(model = QA_model,
                               args = args,
                               train_dataset = train_dataset,
                               eval_dataset = valid_dataset,
                               tokenizer = tokenizer)

# train
trainer.train()

```

Hình 4. 53 Khởi tạo đối tượng trainer và tiến hành train

4.3.4.3. Mô hình OpenAI

- Nhóm sử dụng mô hình gpt-3.5-turbo-1106 của OpenAI để thực hiện. Vì một số lý do nên nhóm sẽ không sử dụng code trong phần này và toàn bộ quá trình thực hiện trên OpenAI Platform.
 - Thực hiện vào trong OpenAI Platform (<https://platform.openai.com/>) tiến hành Login đăng nhập → Chọn Fine-tuning → Chọn Create.

The screenshot shows the 'Fine-tuning' section of the OpenAI Platform. On the left, a sidebar lists 'Playground', 'Assistants', 'Fine-tuning' (which is selected and highlighted in green), 'API keys', 'Files', 'Usage', and 'Settings'. The main area is titled 'Fine-tuning' and has tabs for 'All', 'Successful', and 'Failed'. There are four successful runs listed:

Run ID	Model	Created
ft:gpt-3.5-turbo-1106:personal::8UW4Guat	ft:gpt-3.5-turbo-1106:personal	16:01 11/12/2023
ft:gpt-3.5-turbo-1106:personal::8US0zR3P	ft:gpt-3.5-turbo-1106:personal	10:30 11/12/2023
ft:davinci-002:personal::8TuEWmt4	ft:davinci-002:personal	23:10 9/12/2023
ft:gpt-3.5-turbo-1106:personal::8ToizNbY	ft:gpt-3.5-turbo-1106:personal	17:15 9/12/2023

At the top right, there are 'Learn more' and 'Create' buttons.

Hình 4. 54 Ché độ Fine-tuning trên OpenAI Platform

- Đây tập dữ liệu đã chuẩn bị trong mục 4.2.2.3 lên trên OpenAI Platform để thực hiện quá trình tinh chỉnh.

The dialog box is titled 'Create a fine-tuned model'. It has several sections:

- Base model:** A dropdown menu showing 'gpt-3.5-turbo-1106'.
- Training data:** A section for adding a jsonl file for training. It shows 'file-fyRYLh1Oa0SOdNiyVx6bPriP' and a 'Browse files' button.
- Validation data:** A section for adding a jsonl file for validation metrics. It shows 'None' selected.
- Buttons:** 'Learn about fine-tuning', 'Cancel', and a green 'Create' button.

Hình 4. 55 Hình ảnh chọn tập dữ liệu và mô hình để finetune trong OpenAI

Chọn create và màn hình sẽ chuyển sang hiện thị như hình bên dưới (Xem hình 4.56)

MODEL

gpt-3.5-turbo-1106 Validating files...

Cancel fine-tuning job

Job ID	ftjob-S6uCaewWjyY93di86JEWbOCb
Base model	gpt-3.5-turbo-1106
Created at	10:30 11 thg 12, 2023
Trained tokens	-
Epochs	auto

Files

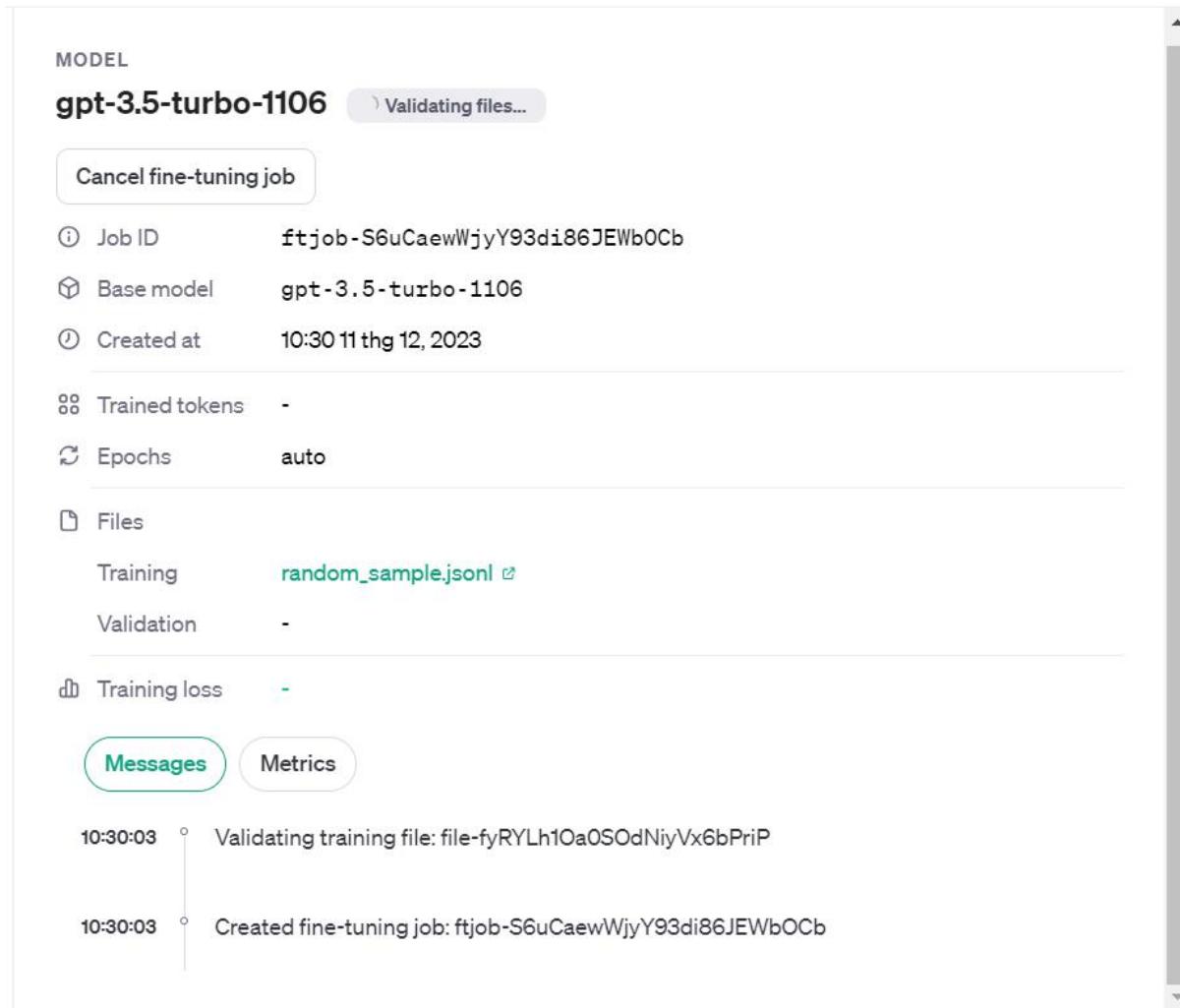
Training	random_sample.jsonl
Validation	-

Training loss -

Messages Metrics

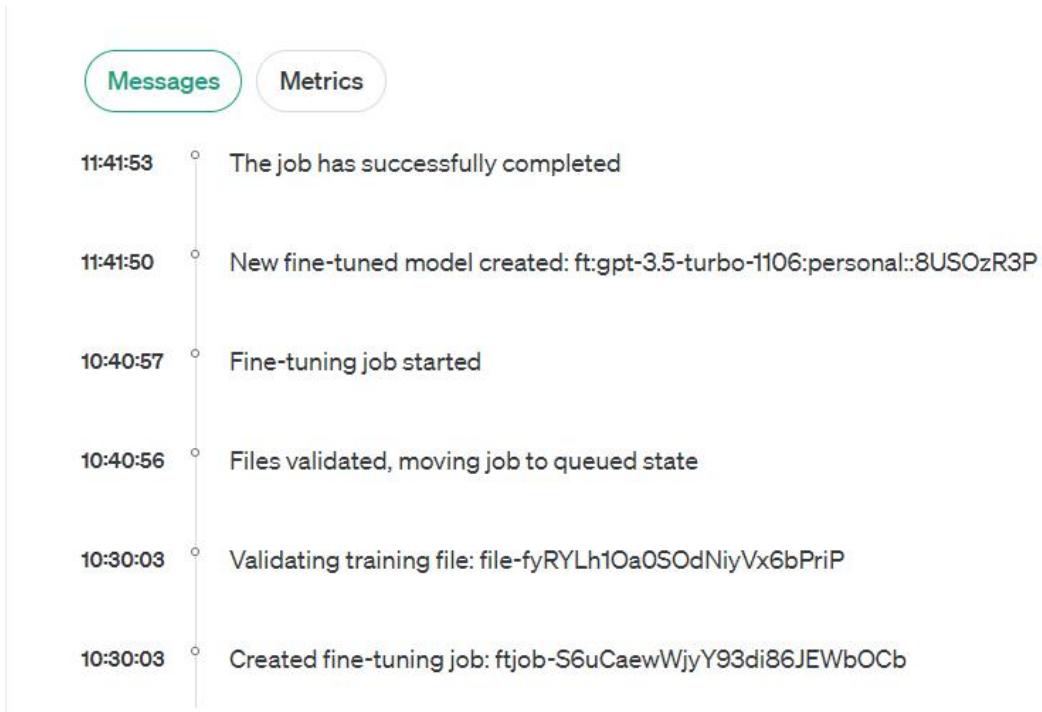
10:30:03 Validating training file: file-fyRYLh1Oa0SOdNiyVx6bPriP

10:30:03 Created fine-tuning job: ftjob-S6uCaewWjyY93di86JEWbOCb



Hình 4. 56 Hình ảnh quá trình finetune trên OpenAI Platform

Hình ảnh quá trình tinh chỉnh mô hình hoàn thành. Xem hình bên dưới (Hình 4.57).



Hình 4. 57 Hình ảnh hoàn thành các quá trình finetune trên OpenAI Platform

4.3.5. Kết quả

4.3.5.1. Mô hình PhoBERT

Sau quá trình tinh chỉnh mô hình PhoBERT cho ngôn ngữ tiếng Việt trên Google Colab nhóm thu được bảng kết quả quá trình tinh chỉnh như bảng dưới đây (Xem bảng 4.8)

Kích thước batch	Num Epoch	Tổng thời gian train	Time train epoch	Time evaluate epoch	Loss Train Lần epoch cuối cùng	Loss Validation lần epoch cuối cùng
32	25	22 phút	6s	52s	0.720	6.613
32	50	50 phút	6s	53s	0.587	6.65
32	100	1 giờ 40 phút	8s	55s	0.467	6.764
32	150	2 giờ 20 phút	8s	54s	0.422	6.675
64	25	22 phút	8s	52s	0.762	6.225

64	50	45 phút	7s	52s	0.576	6.446
64	100	1 giờ	8s	53s	0.461	6.420

Bảng 4. 8 Kết quả train trong mô hình PhoBERT

Nhận xét:

Hiệu suất tăng lên với kích thước batch lớn hơn (64):

- Kích thước batch 64 có vẻ mang lại hiệu suất tốt hơn so với kích thước batch 32, nhất là trên tập đào tạo.
- Ví dụ, mất mát trên tập đào tạo giảm từ 0.720 (kích thước batch 32) xuống còn 0.576 (kích thước batch 64) sau 50 epoch.
- Điều này có thể là do mô hình tận dụng tốt hơn GPU với kích thước batch lớn hơn, tăng cường tính toán đồng thời.

Hiệu suất đạt đến ổn định sau khoảng 100 epoch:

- Trên cả kích thước batch 32 và 64, có vẻ như hiệu suất của mô hình đạt đến ổn định sau khoảng 100 epoch.
- Mất mát trên cả tập đào tạo và validation giảm chậm lại sau khoảng này, và thời gian đào tạo cũng có dấu hiệu ổn định.

Thời gian đào tạo và đánh giá tăng dần theo số epoch:

- Thời gian đào tạo và đánh giá có vẻ tăng dần theo số epoch, đặc biệt là với kích thước batch lớn hơn (64).
- Có thể cần xem xét liệu có cách nào để tối ưu hóa hiệu suất tính toán hoặc cân nhắc sử dụng kỹ thuật đào tạo phân tán nếu cần thiết.

4.3.5.2. Mô hình XLM-R

Sau quá trình thực hiện tinh chỉnh mô hình XLM-R trên Google Colab nhóm thu được bảng kết quả quá trình tinh chỉnh như bảng dưới đây(Xem bảng 4.9) .

Epoch	Step	Training loss
1	766	1.152400

2	1532	0.859100
3	2298	0.670300
4	3064	0.543800
5	3830	0.458900

Bảng 4. 9 Kết quả quá trình huấn luyện mô hình XLM-R

Nhận xét:

- Mất mát trên tập đào tạo giảm đáng kể qua các epoch, từ 1.1524 ở epoch 1 xuống còn 0.4589 ở epoch 5. Sự giảm này có thể là dấu hiệu của việc mô hình đang học và điều chỉnh tham số để tối ưu hóa hiệu suất.
- Số bước đào tạo (steps) tăng theo cấp số nhân qua mỗi epoch (tăng thêm 766 steps mỗi epoch). Điều này có thể ảnh hưởng đến thời gian đào tạo và yêu cầu tài nguyên tính toán tăng theo.
- Tỉ lệ giảm mất mát đào tạo (training loss) giảm dần qua các epoch ($1.1524 \rightarrow 0.8591 \rightarrow 0.6703 \rightarrow 0.5438 \rightarrow 0.4589$). Sự giảm này có vẻ ổn định, nhưng cần theo dõi để đảm bảo mô hình không bị quá mức đào tạo và vẫn có khả năng tổng quát hóa tốt trên dữ liệu mới.

4.3.5.3. Mô hình OpenAI

Sau quá trình thực hiện tinh chỉnh mô hình gpt-3.5-turbo-1106 trên OpenAI nhóm thu được bảng kết quả quá trình tinh chỉnh như bảng dưới đây (Xem bảng 4.10)

Step	Training Loss
1	3.7745
101	0.9955
201	0.3051
301	1.0774
401	0.9675
501	0.7114

601	0.7811
701	0.9367
801	0.7749
901	0.3532
1001	0.1627
1101	1.1331
1201	0.8478
1301	0.6552
1401	0.1936

Bảng 4. 10 Kết quả quá trình tinh chỉnh trên OpenAI

Nhận xét:

- Mất mát đào tạo (training loss) có vẻ biến động đáng kể qua các bước (steps), từ 3.7745 xuống còn 0.1936. Các biến động này có thể là do sự đa dạng của dữ liệu.
- Mất mát đào tạo giảm một cách đáng kể từ 3.7745 xuống 0.1936 qua 1401 bước. Sự giảm mất mát này có vẻ tích cực và thường là dấu hiệu của việc mô hình đang học và tinh chỉnh tham số hiệu quả, hoặc sự nhạy cảm của mô hình với các mẫu cụ thể.
- Bước thứ 1101 có mất mát đào tạo lớn là 1.1331. Điều này có thể là do mô hình gặp phải một trường hợp khó hoặc không phô biến trong dữ liệu, hoặc có thể là do sự nhạy cảm của mô hình đối với một số mẫu cụ thể.

CHƯƠNG 5. XÂY DỰNG ỨNG DỤNG DEMO

5.1. Phân tích, đặc tả yêu cầu

5.1.1. Mô tả chung

Sản phẩm cung cấp một giao diện trực quan cho người dùng, cho phép họ thuận tiện ghi âm và nhập văn bản trực tiếp trên trình duyệt web (ở đây trên nền tảng Streamlit). Ứng dụng chatbot được thiết kế với mục đích cung cấp một giao diện thân thiện, cho phép người dùng đặt câu hỏi một cách dễ dàng bằng cả văn bản và giọng nói, và nhận câu trả lời ngay lập tức.

5.1.2. Yêu cầu chức năng

Yêu cầu chức năng chatbot bằng văn bản:

- Dữ liệu vào: Yêu cầu của người dùng nhập dưới dạng văn bản.
- Xử lý: Thiết bị nhận văn bản để biết yêu cầu của người dùng. Nếu yêu cầu được hỗ trợ, hệ thống xử lý yêu cầu đó và phản hồi bằng giọng nói và văn bản cho người dùng.
- Kết quả: Phản hồi bằng văn bản hiển thị lên màn hình ứng dụng và giọng nói của thiết bị sử dụng.

Yêu cầu chức năng chatbot bằng giọng nói:

- Dữ liệu vào: Yêu cầu của người dùng dưới dạng âm thanh.
- Xử lý: Thiết bị biến giọng nói vào thành văn bản để biết yêu cầu của người dùng. Nếu yêu cầu được hỗ trợ, hệ thống xử lý yêu cầu đó và phản hồi bằng giọng nói và văn bản cho người dùng.
- Kết quả: Phản hồi bằng văn bản hiển thị lên màn hình ứng dụng và giọng nói của thiết bị.

5.1.3. Yêu cầu giao diện phần mềm

Yêu cầu giao diện cho chức năng cho ứng dụng chatbot: Giao diện đơn giản, dễ sử dụng.

Yêu cầu giao diện cho chức năng Chat bằng văn bản:

- Giao diện đơn giản, dễ sử dụng
- Câu lệnh ngắn gọn, dễ ghi
- Phản hồi ngắn gọn dễ đọc
- Phản hồi mọi câu lệnh dù câu lệnh đó không được hỗ trợ

Yêu cầu giao diện cho chức năng Chat bằng giọng nói:

- Giao diện đơn giản, dễ sử dụng
- Câu lệnh ngắn gọn, dễ đọc
- Phản hồi ngắn gọn, dễ nghe, dễ đọc
- Phản hồi mọi câu lệnh dù câu lệnh đó không được hỗ trợ
- Phải có ghi âm để phản hồi trực quan

5.1.4. Ràng buộc thiết kế

- Sản phẩm được thiết kế bằng tiếng Việt, bao gồm giao diện người dùng, các phản hồi bằng văn bản và giọng nói.
- Sản phẩm ứng dụng cần đảm bảo tính thẩm mỹ, dễ sử dụng.

5.2. Kết quả xây dựng ứng dụng

5.2.1. Nền tảng phát triển

Hiện tại, ứng dụng đã được phát triển trên nền tảng Streamlit và chỉ có thể truy cập local tại máy tính cá nhân. Ngoài ra nhóm còn có phiên bản demo giới hạn hơn (số lượng thuộc tính demo nhỏ hơn, trong đó không có chức năng giọng nói) có thể truy cập tại [Link](#).

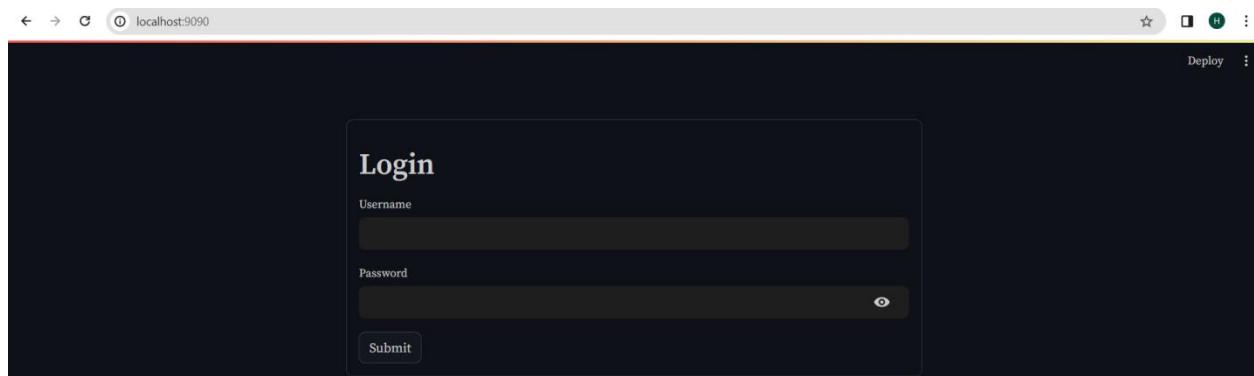
5.2.2. Các chức năng chính của ứng dụng

Ứng dụng chatbot được sử dụng với ngôn ngữ chính là tiếng Việt, dữ liệu đầu vào và đầu ra là văn bản hoặc giọng nói. Đầu ra là câu trả lời dạng văn bản và giọng nói. Gồm có các chức năng chính như sau:

- Sử dụng Chatbot gồm có ChatGPT, Google Bard.
- Sử dụng Hộp thoại Prompt để sử dụng các gợi ý các câu lời nhắc gồm có tiếng Anh, tiếng Trung và tiếng Việt.
- Chức năng sử dụng Google Bard với khóa token cá nhân gồm có 1_PSID, 1_PSIDCC, 1_PSIDTS.
- Chức năng sử dụng khóa API cá nhân để sử dụng mô hình của OpenAI.
- Sử dụng mô hình do nhóm thực hiện tinh chỉnh cho ngôn ngữ tiếng Việt gồm có XLM-R, OpenAI và PhoBERT.
- Sử dụng một số mô hình trong Transformers của Hugging Face như ViT5, GPT2.
- Sử dụng các API hỗ trợ xây dựng chatbot của Rapid API ví dụ như BingGPT, ChatGPT, v.v.
- Tắt âm thanh văn bản tạo ra khi có phản hồi, sử dụng giọng nói để đặt câu hỏi cho chatbot.

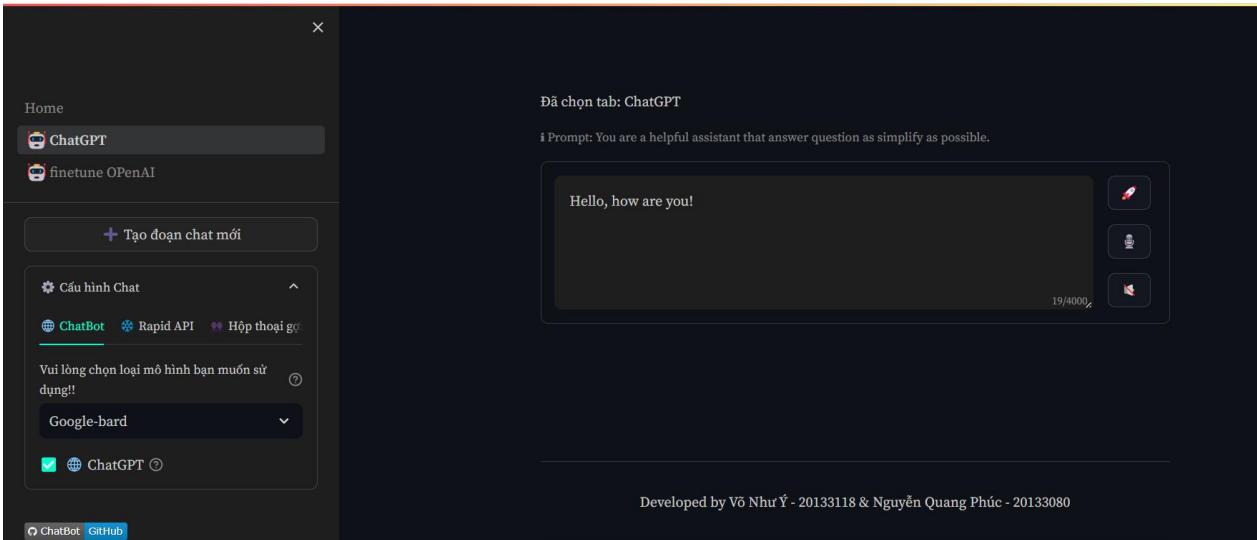
Giao diện chatbot:

- Giao diện đăng nhập ứng dụng khi mở ứng dụng lên sẽ hiển thị như hình bên dưới (Hình 5.1).

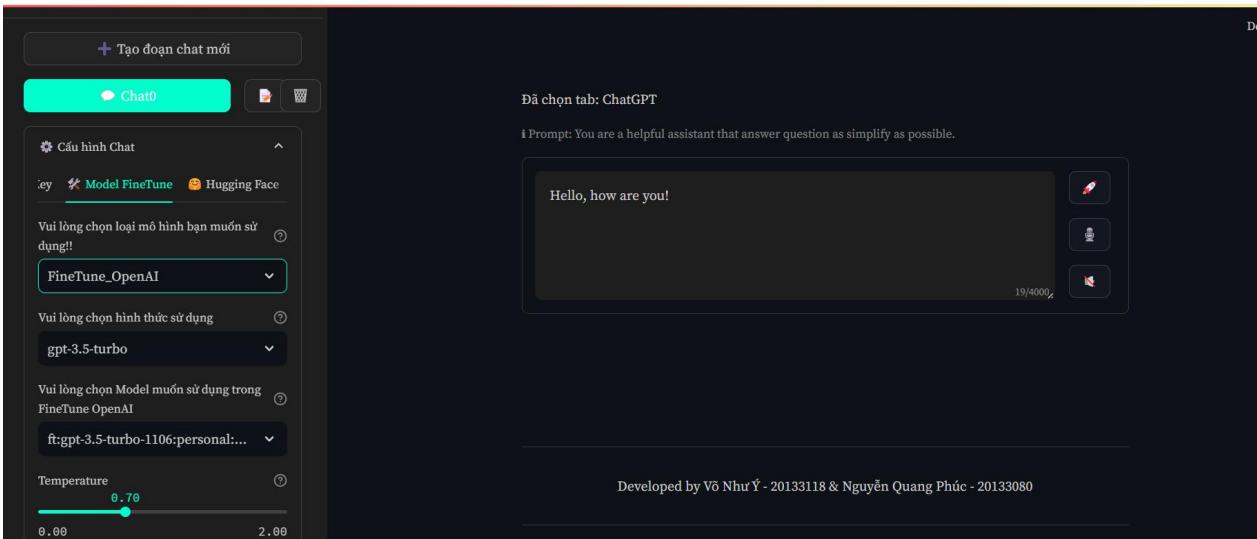


Hình 5. 1 Giao diện đăng nhập của ứng dụng chatbot

- Hình ảnh giao diện đăng nhập khi đăng nhập thành công hiện ra như hình bên dưới (Hình 5.2 và Hình 5.3)



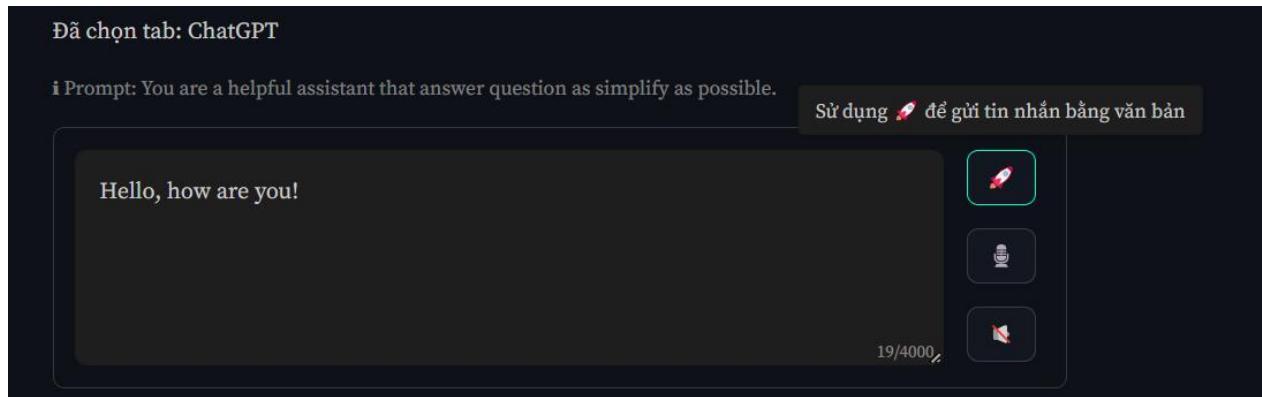
Hình 5. 2 Giao diện chính khi đăng nhập thành công



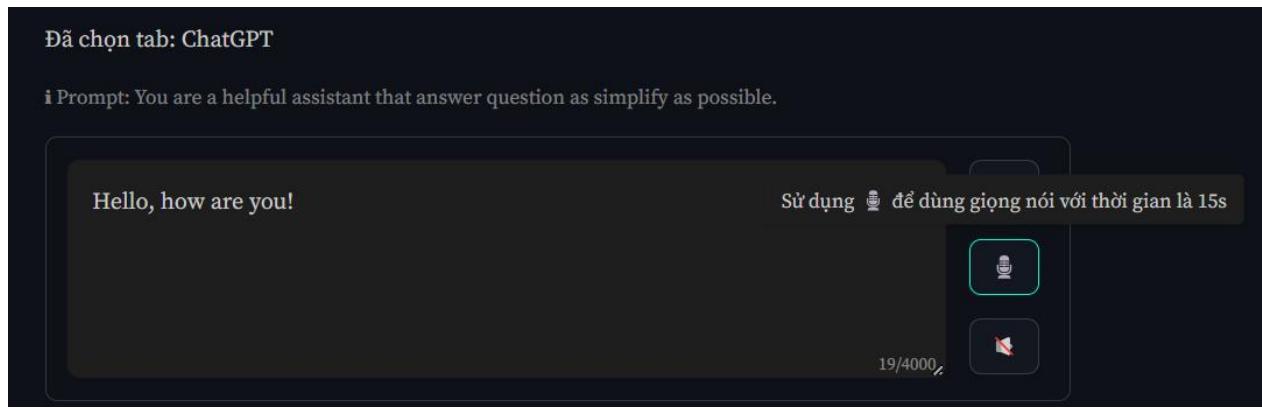
Hình 5. 3 Hình ảnh giao diện chính ở chế độ Model Finetune

Giao diện gồm các phần sau đây:

Thanh nhập dữ liệu để người dùng nhập văn bản, nút bấm để thực hiện gửi yêu cầu từ người dùng đến hệ thống. Khi người dùng sử dụng giọng nói thì chỉ cần nhấn nút ghi âm và đợi phản hồi từ chatbot.

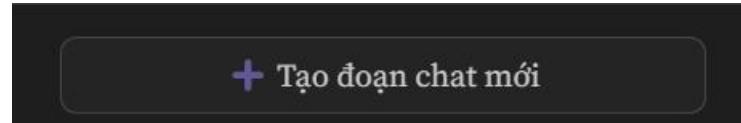


Hình 5. 4 Hình ảnh nút gửi tin nhắn bằng văn bản



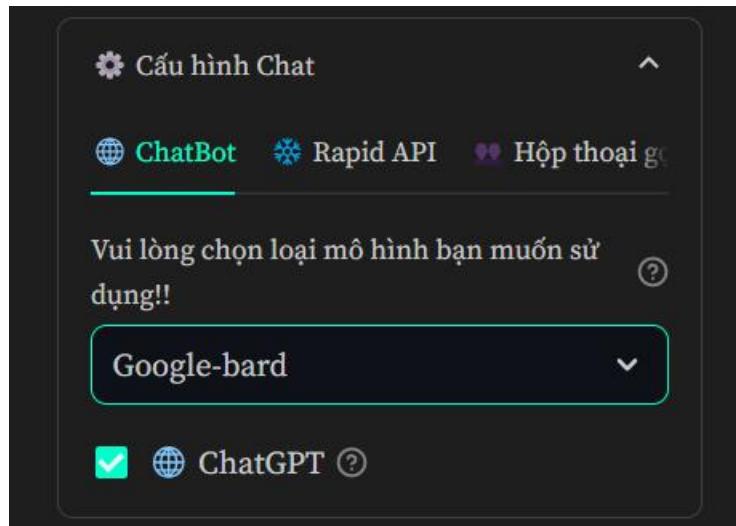
Hình 5. 5 Hình ảnh nút nhấn chat bằng giọng nói

Thanh tạo đoạn chat mới dùng để tạo các đoạn hội thoại mới.



Hình 5. 6 Thanh tạo đoạn chat mới trong ứng dụng

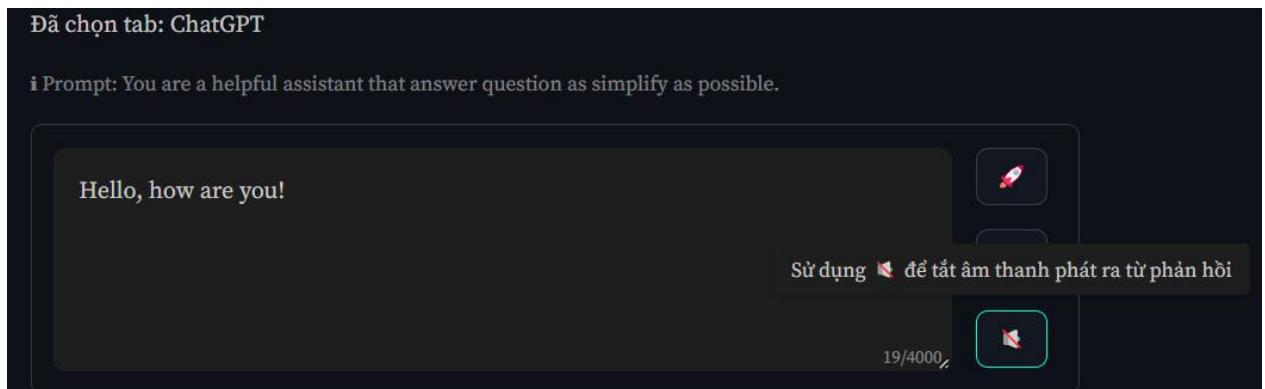
Thanh cấu hình chat dùng để lựa chọn mô hình và các tính năng trong mô hình đó. Trong đó có các lựa chọn sử dụng như Chatbot, Rapid API, Model Finetune, Hugging Face, Prompt, Chat GPT use API key và Google Bard.



Hình 5. 7 Thành phần cấu hình Chat

Nút chọn checkbox để chọn loại mô hình muốn sử dụng (lưu ý là tích 2 lần để không bị trùng lặp lại mô hình sử dụng trước đó).

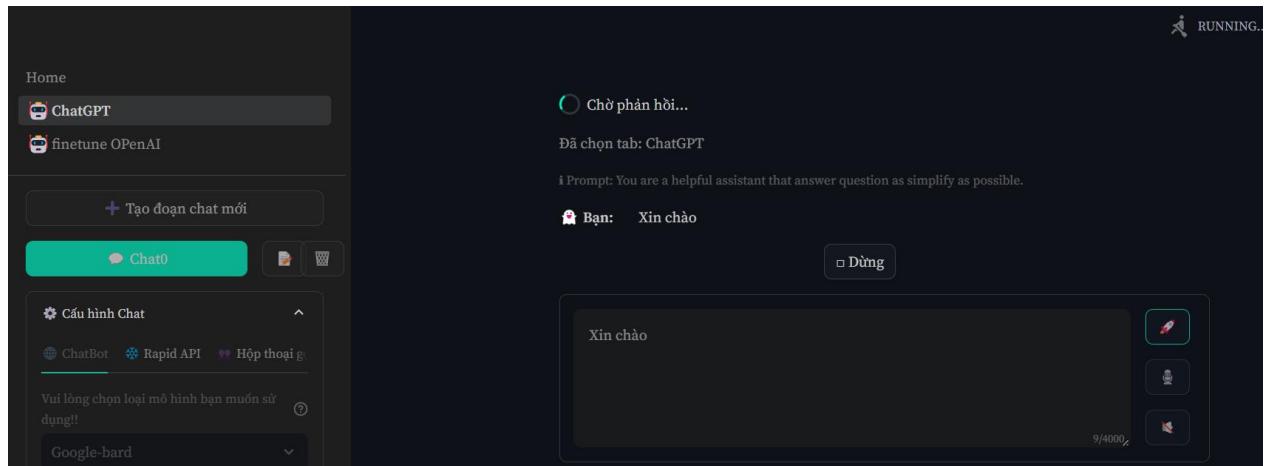
Nút bấm tắt âm thanh phát ra từ văn bản phản hồi.



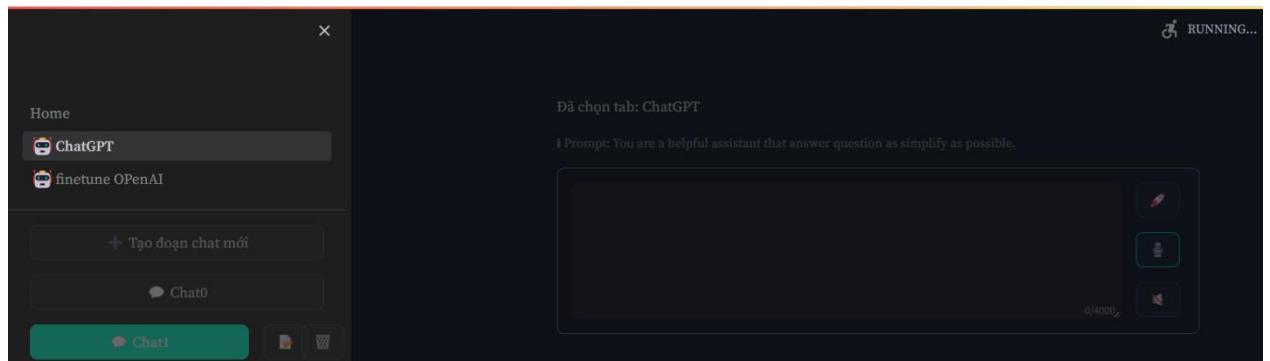
Hình 5. 8 Nút tắt âm thanh phản hồi

Bên trái khung hình hiển thị câu trả lời từ ứng dụng.

- Để sử dụng người dùng nhập đoạn tin nhắn, chọn loại mô hình và nhấn biểu tượng hoặc dùng tổ hợp phím Ctrl + Enter với tin nhắn văn bản hoặc sử dụng biểu tượng ghi âm với giọng nói.

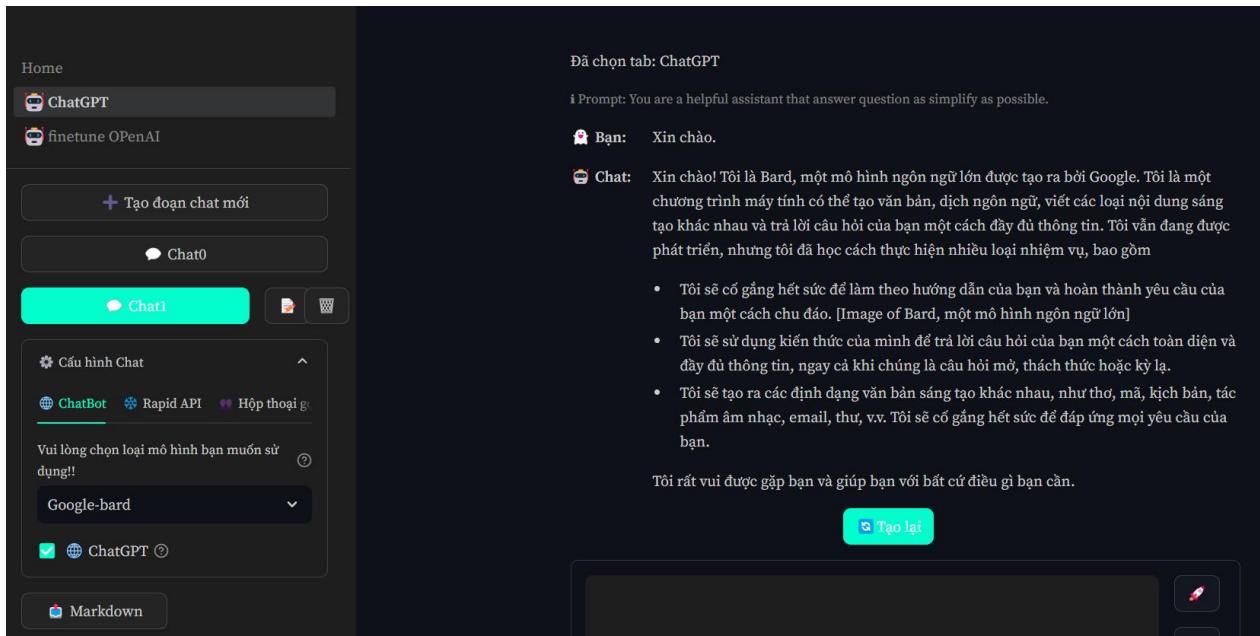


Hình 5. 9 Sử dụng chatbot bằng văn bản

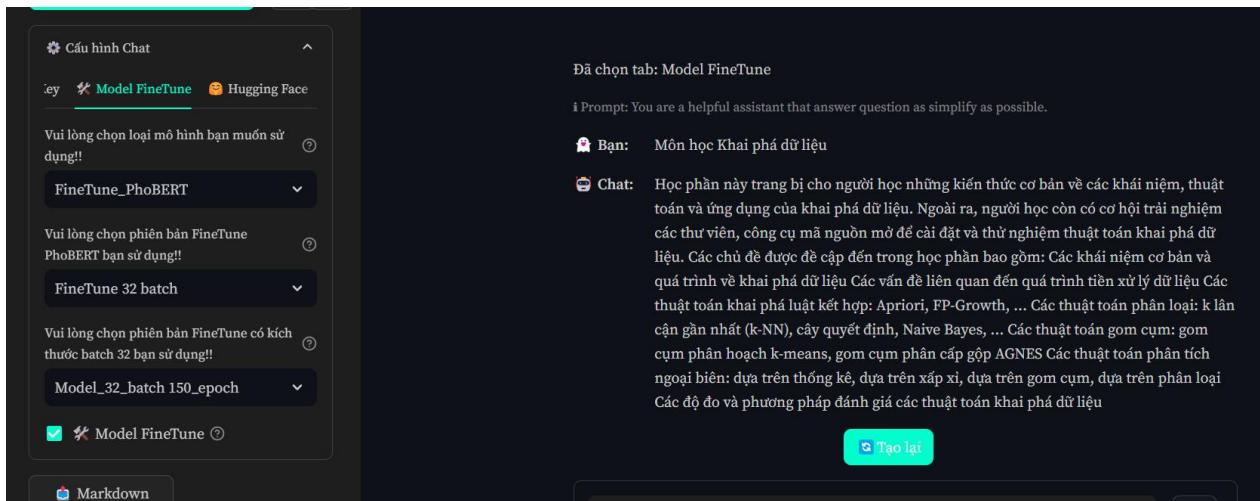


Hình 5. 10 Sử dụng chatbot bằng giọng nói

Kết quả khi sử dụng chatbot

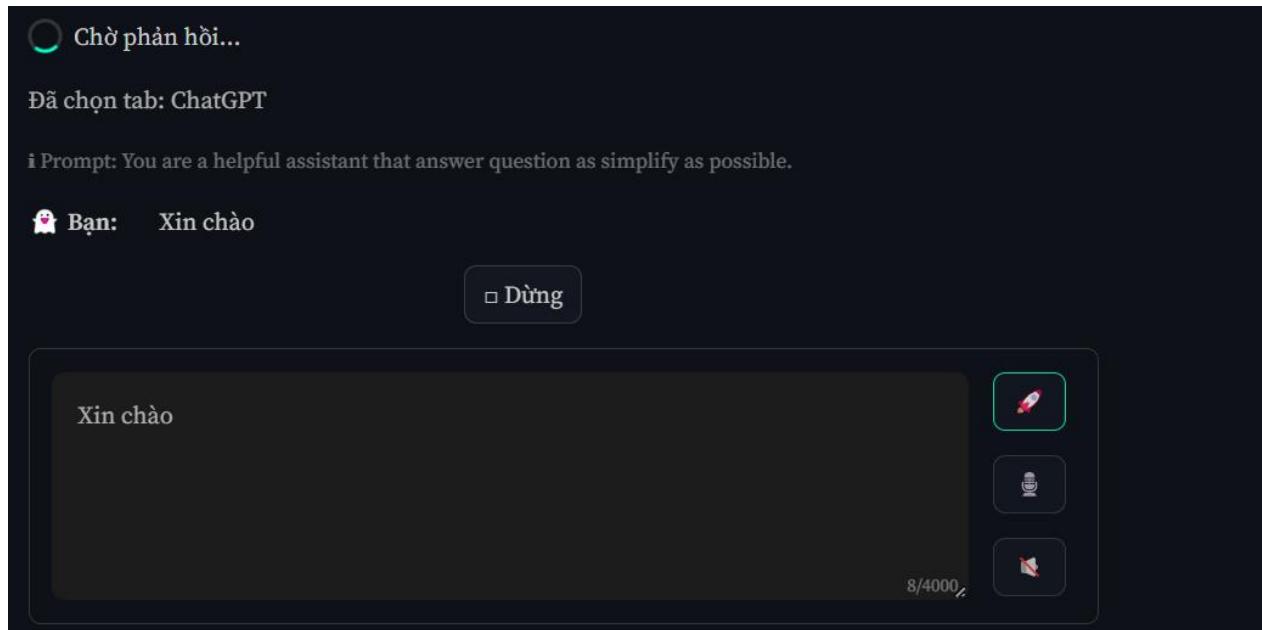


Hình 5. 11 Kết quả khi sử dụng Google Bard



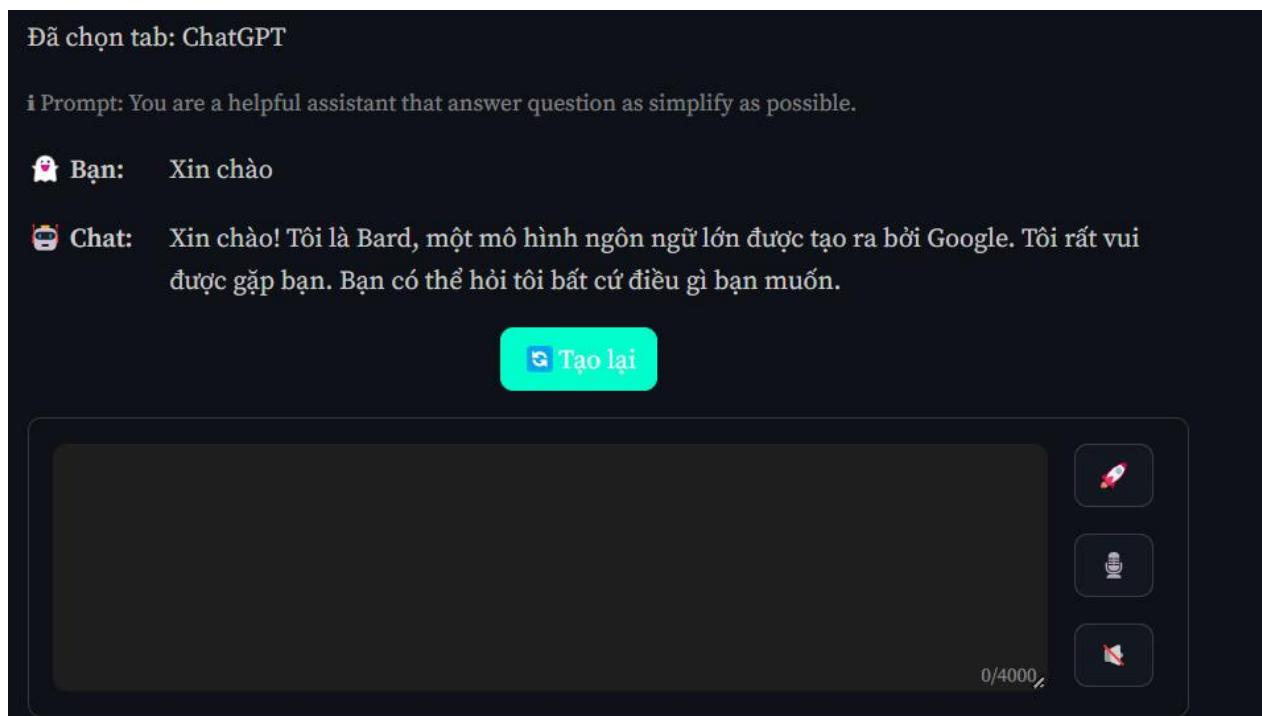
Hình 5. 12 Hình ảnh kết quả trả về khi sử dụng chatbot ở chế độ Model Finetune mô hình PhoBERT

Nút Dừng dùng để dừng, chấm dứt không tạo phản hồi cho câu hỏi đó nữa. Khi thực hiện kết quả hiện lại là câu hỏi còn câu trả lời sẽ bị mất đi.



Hình 5. 13 Nút dừng trong quá trình chờ câu hỏi phản hồi

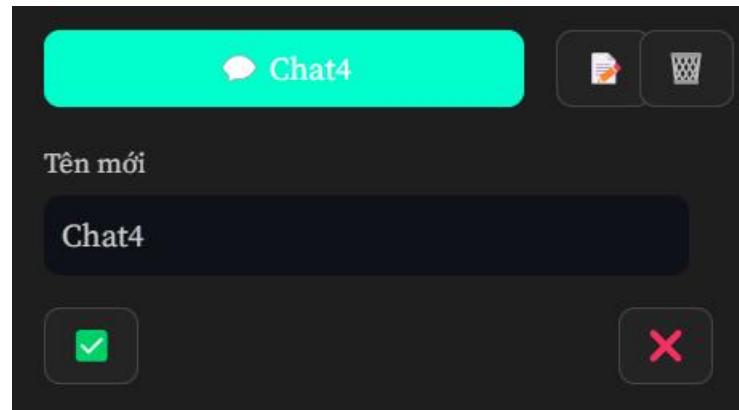
Nút Tạo lại dùng để khởi tạo lại câu hỏi để nhận phản hồi mới nếu thấy chưa phù hợp.



Hình 5. 14 Nút tạo lại khi hoàn thành câu trả lời

Nút Tạo lại dùng để thực hiện thay đổi tên đoạn chat trong đó có 2 nút nhấn là ✓ dùng để thực hiện cập nhật tên mới và ✗ dùng để hủy bỏ thao tác đổi tên.

Nút dùng để xóa đoạn tin nhắn của đoạn Chat.



Hình 5. 15 Hình ảnh các thao tác với đoạn chat

Phần cài đặt ứng dụng và sử dụng chi tiết các chức năng sẽ được trình bày trong phần Phụ lục.

CHƯƠNG 6. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

6.1. Kết luận

6.1.1. Tóm tắt kết quả đạt được

Trong quá trình thực hiện tiểu luận chuyên ngành, nhóm đã học hỏi và đạt được một số kết quả. Cụ thể:

Về mặt nghiên cứu và ứng dụng:

- Tìm hiểu các thư viện hỗ trợ xây dựng chatbot trong ngôn ngữ Python và các API hỗ trợ xây dựng chatbot.
- Mô tả được cách cài đặt, huấn luyện và so sánh thư viện, API với nhau.
- Nghiên cứu sử dụng phương pháp chuyển đổi giọng nói thành văn bản và chuyển đổi văn bản thành giọng nói với bộ thư viện Speech-to-Text và Text-to-Speech của Google Cloud API trong việc xây dựng chatbot.
- Tìm hiểu được cách tinh chỉnh các mô hình ngôn ngữ tiếng Việt để xây dựng chatbot như PhoBERT, XLM-R.
- Tìm hiểu cách tinh chỉnh mô hình thông qua OpenAI.
- Tạo bộ dữ liệu huấn luyện bao gồm khoảng 1397 câu hỏi cho OpenAI và bộ 3397 câu hỏi cho mô hình PhoBERT.

Về mặt kỹ năng mềm:

- Kỹ năng đọc tài liệu và phân tích vấn đề.
- Kỹ năng nghiên cứu dự án.
- Kỹ năng làm việc nhóm và phân chia công việc.
- Kỹ năng trình bày và thuyết trình, viết báo cáo.

6.1.2. Ý nghĩa

Ngoài những kết quả đã đạt được, việc hoàn thành đề tài "**Tìm hiểu về các thư viện hỗ trợ xây dựng chatbot**" còn mang lại những ý nghĩa quan trọng:

Minh chứng cho tính khả thi của đề tài: Kết quả của đề tài không chỉ là một bảng điểm cho sự cố gắng và nỗ lực cá nhân mà còn là minh chứng rõ ràng về tính khả thi của việc nghiên cứu và phát triển chatbot. Qua quá trình tìm hiểu thư viện xây dựng chatbot, đề tài đã làm rõ sự hiệu quả và khả năng ứng dụng thực tế của công nghệ này.

Chứng minh ý nghĩa của một chatbot trong đời sống hàng ngày: Đề tài không chỉ là một dự án nghiên cứu trừu tượng mà còn là bằng chứng về ý nghĩa thực tế của chatbot trong cuộc sống hàng ngày. Với khả năng nghe hiểu và phản hồi bằng giọng nói và văn bản tiếng Việt, chatbot không chỉ là một công cụ thông tin mà còn là một người trợ lý ảo, giúp giải quyết nhanh chóng và hiệu quả các vấn đề của người dùng.

6.1.3. Những hạn chế, giới hạn

- Ứng dụng chatbot chưa áp dụng được thư viện hỗ trợ khác như Rasa, Snips NLU, v.v
- Số lượng thư viện, API hỗ trợ xây dựng chatbot chưa được đa dạng.
- Chất lượng câu trả lời từ việc tinh chỉnh mô hình như PhoBERT, XLM-R chưa đạt được kết quả tốt.
- Vấn đề xác định tag trong mô hình PhoBERT vẫn còn nhầm lẫn khá cao và chưa được chính xác.
- Chất lượng dịch từ giọng nói sang văn bản đôi khi còn bị nhầm lẫn từ như từ ‘Võ’ thành từ ‘bỏ’.
- Phần chat bằng giọng nói vẫn đang bị giới hạn phần cứng khi setup thắng số giây ghi âm là 15s.

6.2. Hướng phát triển

- Từ những hạn chế đã được nêu trong phần Kết luận 6.1, trong tương lai nhóm có những dự định bao gồm cải tiến thư viện, API. Xây dựng thuật toán phân tích ý định (intent), trích xuất các thực thể, dịch thuật dựa trên thư viện Snips NLU.

- Xây dựng bộ dữ liệu câu hỏi đa dạng hơn về trường để có thể tăng số lượng đào tạo trong quá trình huấn luyện, tinh chỉnh.
- Phát triển xây dựng website với nhiều chức năng tùy chọn trong chatbot.
- Tối ưu phần chat bằng giọng nói để có thể sử dụng cho bất kỳ đoạn câu nói nào mà không cần phải bị giới hạn thời gian là 15s.

TÀI LIỆU THAM KHẢO

[1] Kate Brush, Jesse Scardina. (November 2021). Definition chatbot. Retrieved December 4, 2022, from

<https://www.techtarget.com/searchcustomerexperience/definition/chatbot>

[2] Đỗ Minh Đức. (2023, May 26). Tổng hợp những ưu và nhược điểm của chatbot có thể bạn chưa biết. Retrieved December 10, 2023, from <https://bizfly.vn/techblog/uu-va-nhuoc-diem-cua-chatbot.html>

[3] Tailieuthamkao.com. (2022, December 12). Nghiên cứu và xây dựng chatbot hỗ trợ người dùng trong ngân hàng. Retrieved December 7, 2023, from

<https://tailieuthamkao.com/nghien-cuu-va-xay-dung-chatbot-ho-tro-nguoi-dung-trong-ngan-hang-2-52710>

[4] Nguyen Mai. (2023, June 23). Fine-tuning một cách hiệu quả và thân thiện với phần cứng: Adapters và LoRA. Retrieved December 11, 2023, from <https://viblo.asia/p/fine-tuning-mot-cach-hieu-qua-va-than-thien-voi-phan-cung-adapters-va-lora-5pPLkj3eJRZ>

[5] Phạm Đình Khánh. (23 May 2020). BERT model. Retrieved December 10, 2023, from <https://phamdinhkhanh.github.io/2020/05/23/BERTModel.html>

[6] Phạm Hữu Quang. (19 July 2020). BERT, RoBERTa, PhoBERT, BERTweet: Ứng dụng state-of-the-art pre-trained model cho bài toán phân loại văn bản. Retrieved December 10, 2023, from <https://viblo.asia/p/bert-roberta-phobert-bertweet-ung-dung-state-of-the-art-pre-trained-model-cho-bai-toan-phan-loai-van-ban-4P856PEWZY3>

[7] SDSRV. (2023, October 26). Xây dựng mô hình hỏi đáp đơn giản cho tiếng Việt. Retrieved December 10, 2023, from <https://sdsrv.ai/blog/xay-dung-mo-hinh-hoi-dap-don-gian-cho-tieng-viet/>

[8] Trituenhantao.io. (2020, April 29). BERT, RoBERTa, DistilBERT, XLNet – Chọn cái nào?. Retrieved December 10, 2023, from <https://trituenhantao.io/kien-thuc/bert-roberta-distilbert-xlnet-chon-cai-nao/>

[9] Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., & Brew, J. (2020). HuggingFace's Transformers: State-of-

the-art Natural Language Processing. Retrieved December 10, 2023, from

<https://arxiv.org/abs/1910.03771>

[10] FPT Aptech. ((2022, November 14). *API là gì? Các kiến thức về API dành cho người mới bắt đầu*. Retrieved December 10, 2023, from <https://aptech.fpt.edu.vn/api-la-gi.html>

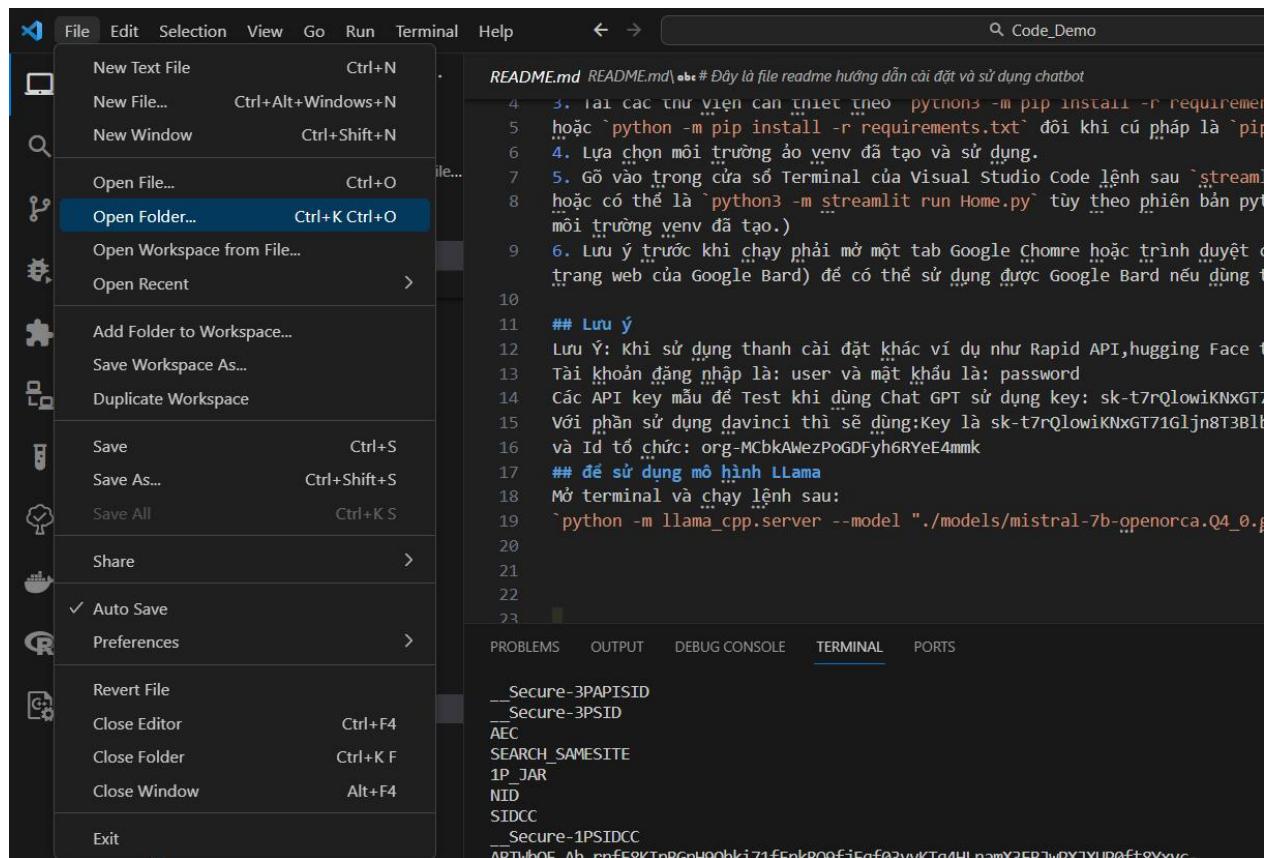
PHỤ LỤC

Phụ lục A. Hướng dẫn cài đặt ứng dụng

A.1. Cài đặt thư viện

Bước 1: Đầu tiên cần lén đường dẫn thư mục chứa code (truy cập [tại đây](#)) tải file zip về và giải nén.

Bước 2: Vào trong ứng dụng Visual Studio Code → Chọn File → Chọn Open Folder → Chọn Folder đã tải về (đã giải nén).



Hình A. 1 Hình ảnh chọn Open Folder trong Visual Studio Code

Bước 3: Mở cửa sổ Terminal và gõ lệnh `python -m venv venv` để tạo môi trường ảo.

Bước 4: Tiến hành kích hoạt môi trường ảo bằng lệnh `.\venv\Scripts\activate`

Bước 5: Trong cửa sổ Terminal của ứng dụng ứng dụng gõ lệnh ***pip install -r requirements.txt*** để tải các thư viện cần thiết.

A.2. Chuẩn bị môi trường trình duyệt

Ứng dụng được chạy trên môi trường trình duyệt Google Chrome với một Gmail đã được tạo sẵn và lưu ý là Gmail đăng nhập vào được trang web

<https://bard.google.com/chat> và số lượng Cookies sử dụng của Gmail đó thấp để ít xảy ra tình trạng lỗi.

A.3. Chạy thử ứng dụng

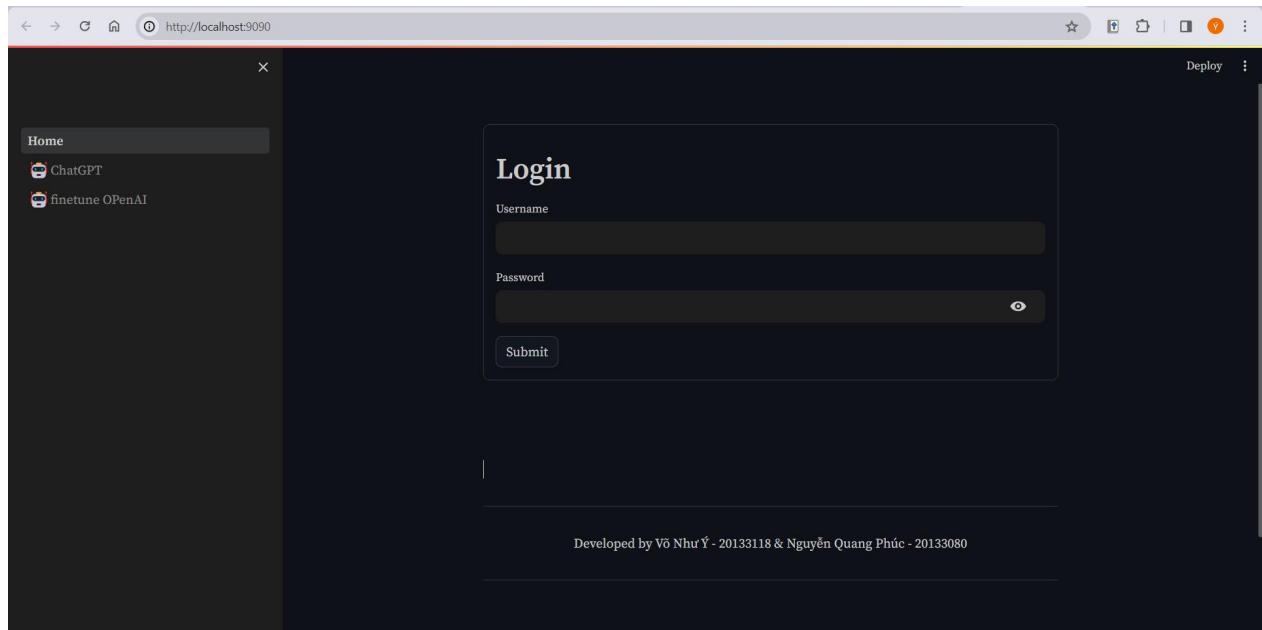
Quay lại trong Visual Studio Code trong cửa sổ Terminal gõ lệnh ***python -m streamlit run Home.py*** hoặc ***streamlit run Home.py*** tùy theo phiên bản Python sử dụng.

```
D:\Download\Code\Code Demo>python -m streamlit run Home.py
2023-12-22 19:57:04.603 "theme.accentcolor" is not a valid config option. If you previously had this config option set, it may have been removed.
You can now view your Streamlit app in your browser.

Local URL: http://localhost:9090
Network URL: http://192.168.10.111:9090
```

Hình A. 2 Hình ảnh câu lệnh chạy ứng dụng trong Terminal

Kết quả sau khi chạy câu lệnh trên là trình duyệt sẽ tự động mở vào đường dẫn <http://localhost:9090/>. Giao diện của ứng dụng xem hình bên dưới

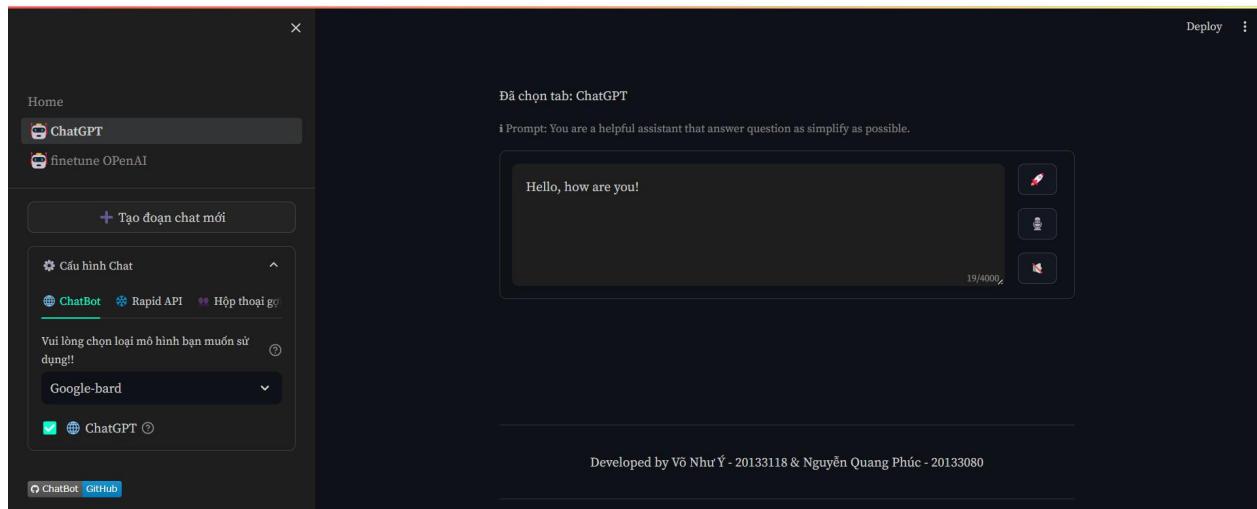


Hình A. 3 Hình ảnh giao diện ứng dụng khi mở trên trình duyệt

Phụ lục B. Hướng dẫn sử dụng các chức năng của ứng dụng

B.1. Hướng dẫn sử dụng chức năng ChatBot

Tiến hành đăng nhập vào Chatbot với tài khoản là user và mật khẩu là password để đăng nhập vào giao diện chính và sau đó chọn LetChats.



Hình B. 1 Giao diện chính của ứng dụng khi đăng nhập thành công

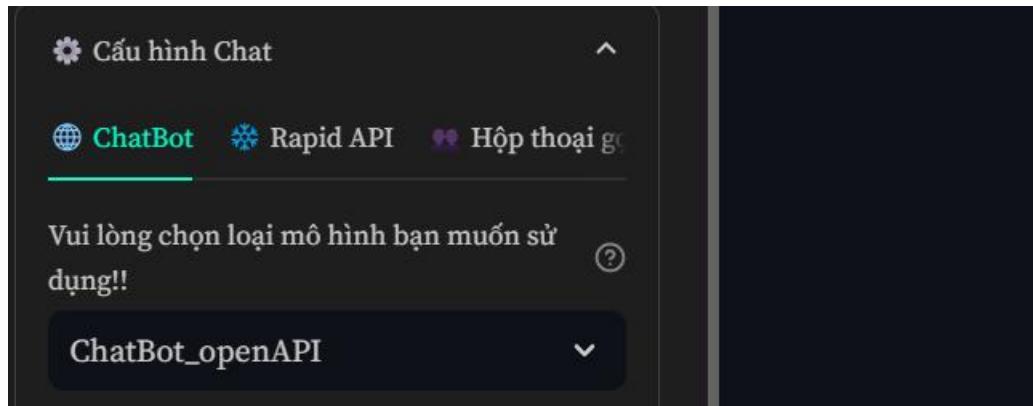
❖ Sử dụng chức năng Google Bard.

Bước 1: Đầu tiên vào thanh cấu hình Chat chọn vào ChatBot và chọn Google-bard (Xem hình B.1)

Bước 2: Sử dụng giọng nói hoặc nút nhấn gửi văn bản để thực hiện. Sau đó đợi câu trả lời từ chatbot phản hồi về.

❖ Sử dụng OpenAI

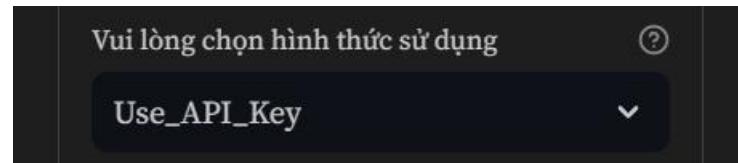
Bước 1: Để sử dụng OpenAI trong thanh cấu hình Chat → chọn ChatBot_openAI



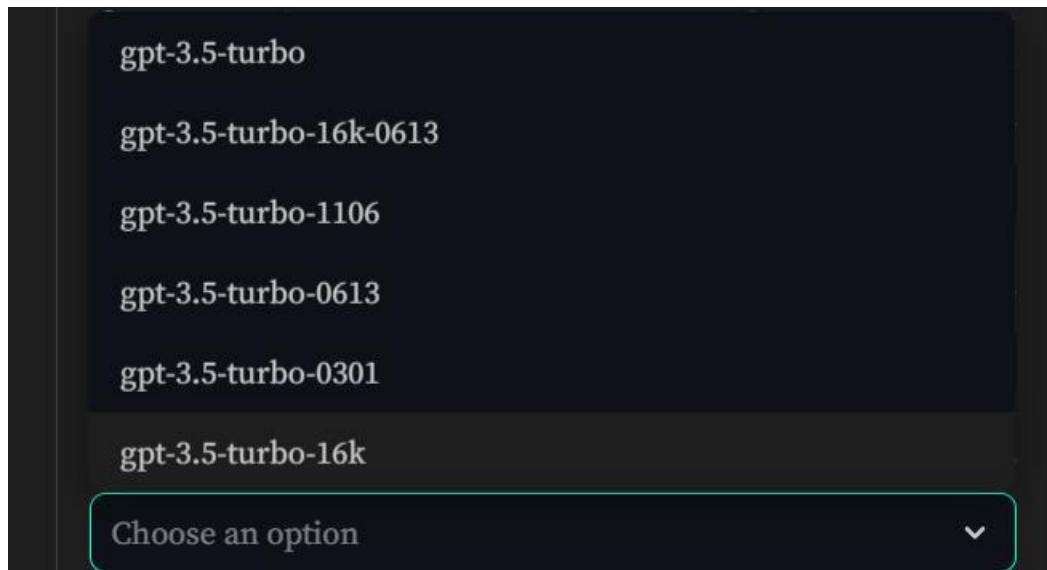
Hình B. 2 Chọn chức năng ChatBot-OpenAPI khi sử dụng chế độ cấu hình ChatBot

❖ Sử dụng API Key

Sau khi chọn ChatBot_openAI → chọn hình thức sử dụng trong ChatBot_openAPI → Chọn Use_API_Key → Chọn mô hình sử dụng trong chức năng này.



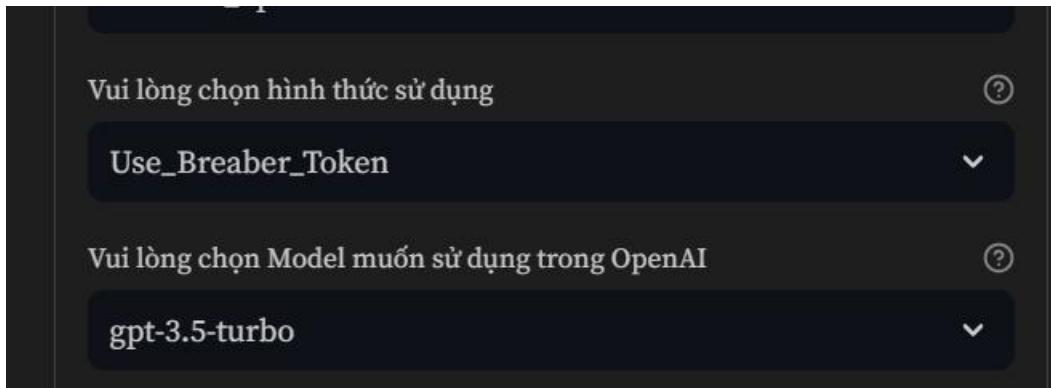
Hình B. 3 Chọn hình thức sử dụng Use_API_Key trong loại mô hình ChatBot_openAPI



Hình B. 4 Chọn mô hình sử dụng trong Use_API_Key

❖ Sử dụng User Breaber Token

Tương tự như sử dụng API key nhưng thay vào đó chọn hình thức sử dụng trong ChatBot_openAPI là Use_Breaber_Token → Chọn mô hình bạn muốn sử dụng.



Hình B. 5 Sử dụng Use_Breaber_Token trong chế độ ChatBot

Giao diện khi chọn chức năng và sử dụng tương tự như sử dụng API Key

❖ **Sử dụng mô hình Text-davinci-003**

Trong mô hình ChatBot_openAI → chọn hình thức sử dụng là text-davinci-003_API

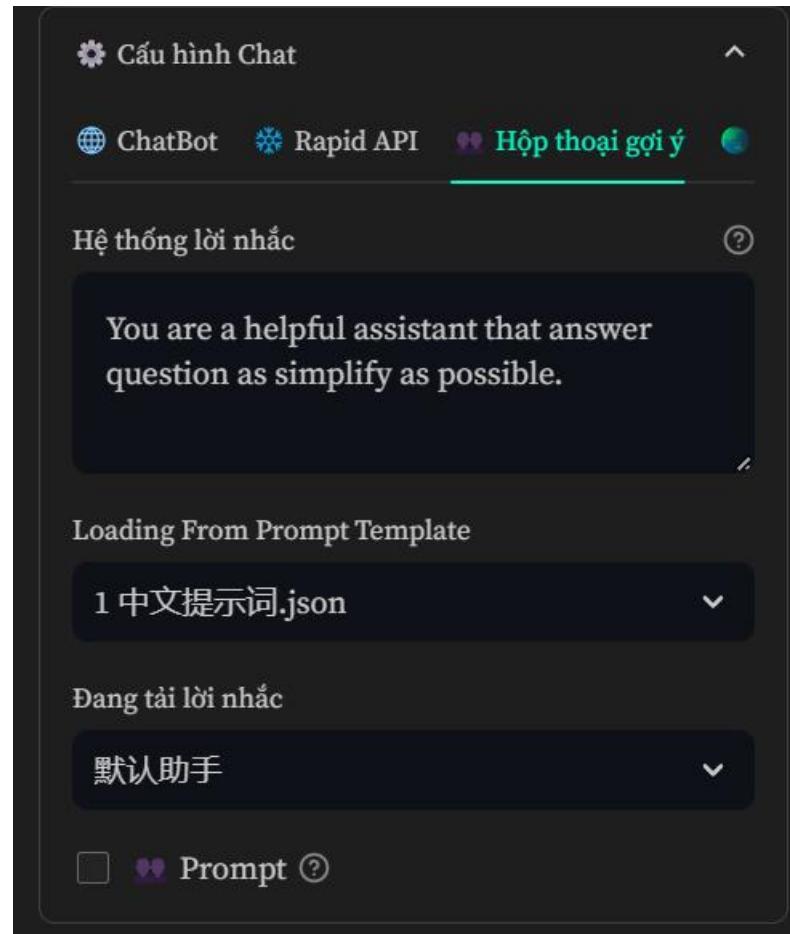
Giao diện được hiển thị như hình bên dưới.



Hình B. 6 Sử dụng text-davinci-003 ở chế độ ChatBot

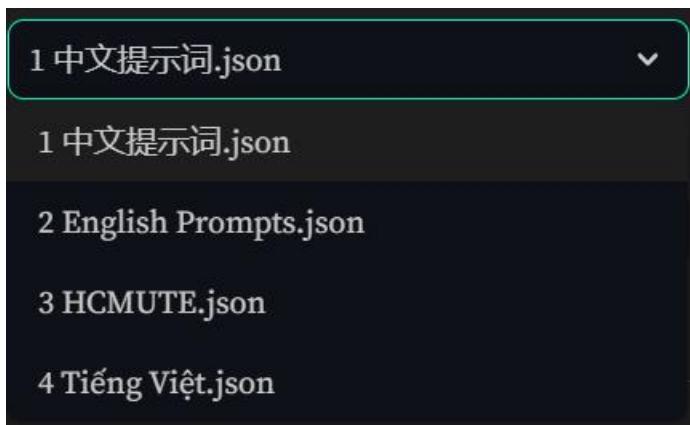
B.2. Hướng dẫn sử dụng chức năng Hộp thoại gợi ý

Bước 1: Vào thành Cấu hình chat chọn Hộp thoại gợi ý và tích vào ô checkbox Prompt 2 lần



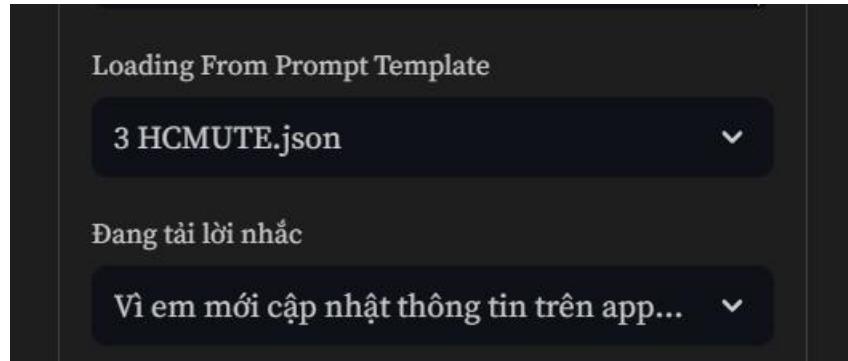
Hình B. 7 Hình ảnh chatbot ở chế độ Hộp thoại gợi ý

Bước 2: Trong Hộp thoại gợi ý để sử dụng lời nhắc tiếng Việt tích vào Loading From Prompt Template và chọn **4 Tiếng Việt.json**, sử dụng tiếng Anh kích vào **2 English Prompts.json** hoặc chọn **3 HCMUTE.json** để sử dụng câu hỏi về trường.



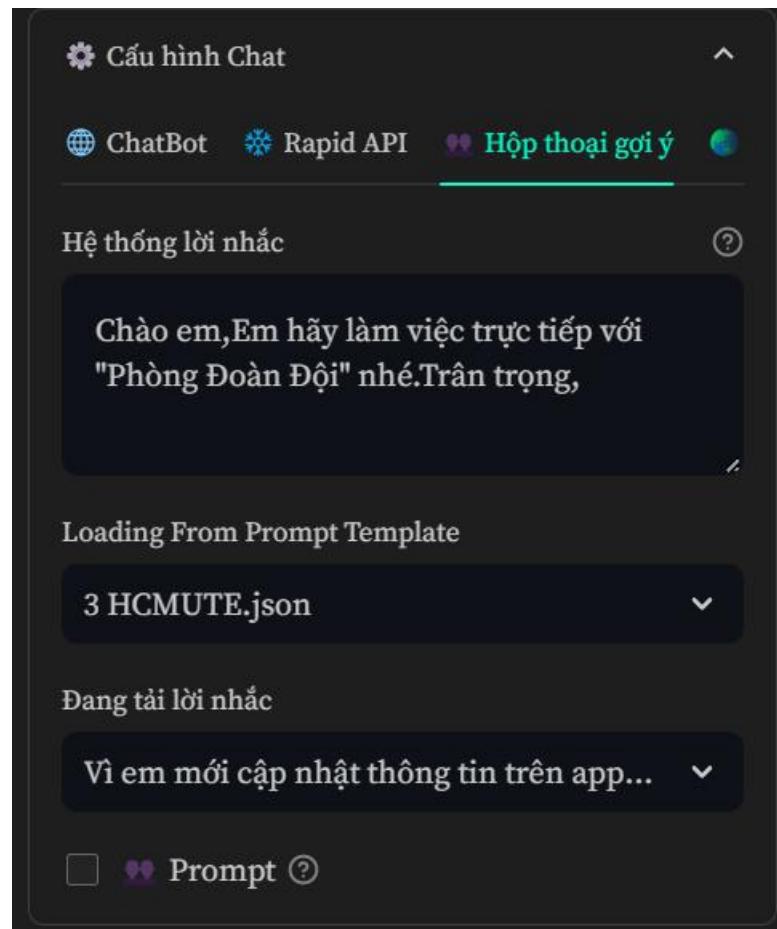
Hình B. 8 Hình ảnh các chế độ sử dụng trong hộp thoại gợi ý

Bước 3: Sau khi lựa chọn lời nhắc cho ngôn ngữ sử dụng trong thanh Đang tải lời nhắc -- → chọn lời nhắc cần xem.



Hình B. 9 Hình ảnh lời nhắc khi sử dụng 3 HCMUTE.json

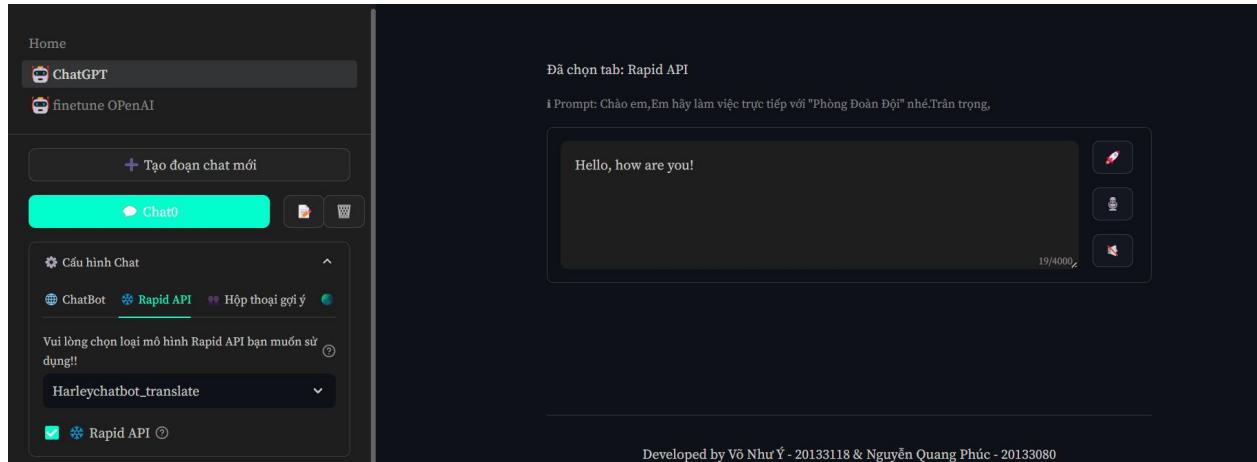
Kết quả hiển thị lên trên ô Hệ thống lời nhắc hiển thị như hình bên dưới (Xem hình B.10)



Hình B. 10 Kết quả trả về khi sử dụng lời nhắc trong Hộp thoại gợi ý

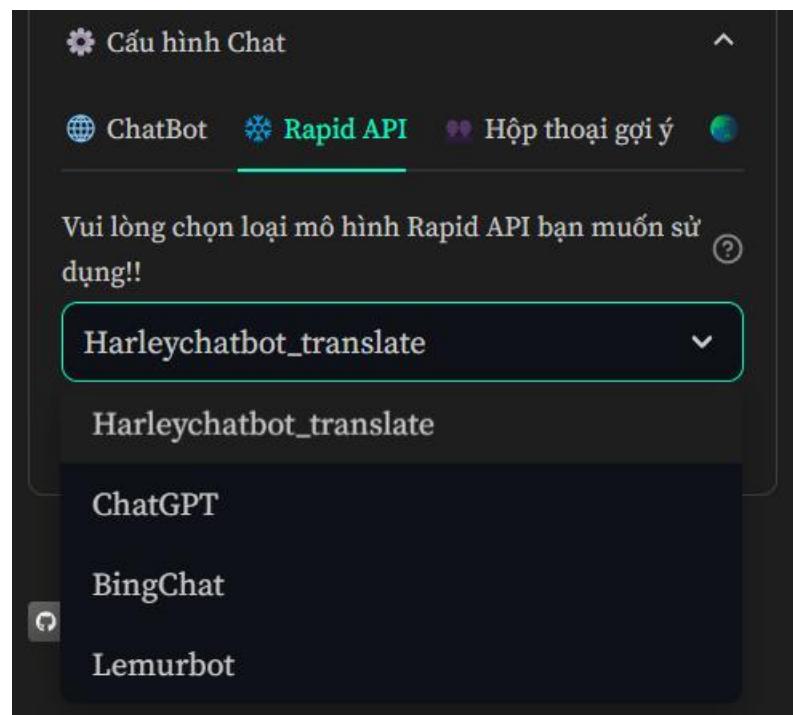
B.3. Hướng dẫn sử dụng chức năng Rapid API

Bước 1: Tiến hành vào thành Cấu hình chat lựa chọn Rapid API và tích 2 lần vào checkbox Rapid API để sử dụng.



Hình B. 11 Hình ảnh chatbot khi sử dụng Rapid API

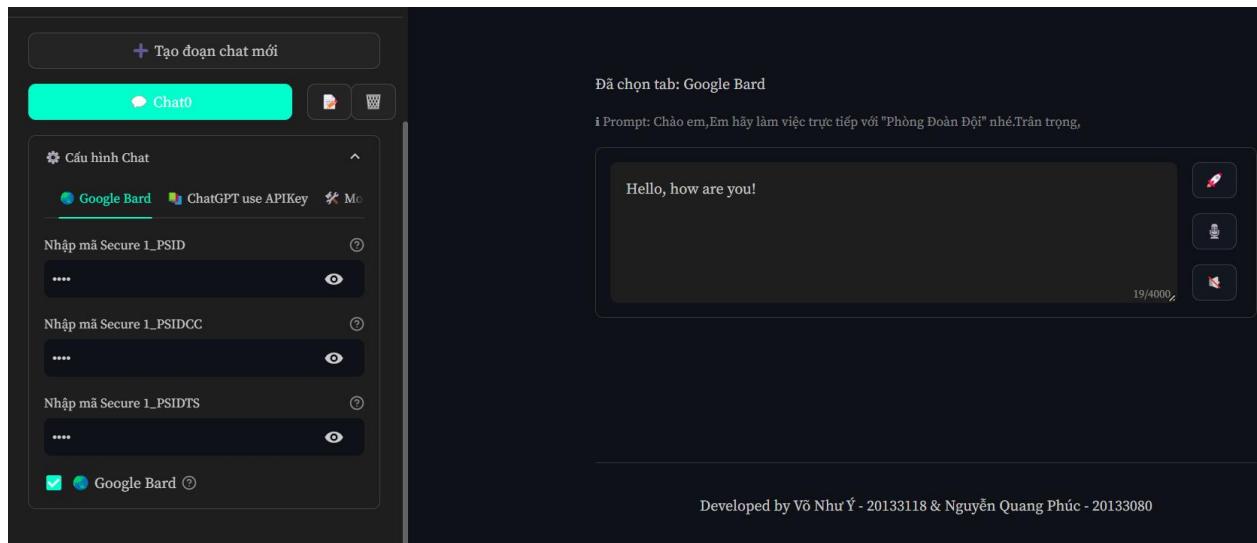
Bước 2: Lựa chọn mô hình muốn sử dụng trong Rapid API như Harleychatbot_translate, ChatGPT, Bing GPT hoặc sử dụng Lemurbot và thực hiện tương tự các chức năng chat bằng giọng nói hoặc văn bản.



B.4. Hướng dẫn sử dụng chức năng Google Bard

Đây là một phiên bản khác so với sử dụng Google Bard ở phần ChatBot trong này người dùng sẽ phải tiến hành nhập các mã khóa Cookies cá nhân tương tự như trong Bước 2 và 3 phần Cài đặt chatbot từ Google Bard API (Trang 48, 49).

Sau khi nhập xong người dùng sẽ tiến hành tích 2 lần vào ô checkbox Google Bard và sử dụng các chức năng như gửi bằng văn bản hoặc giọng nói để thực hiện trò chuyện với chatbot.

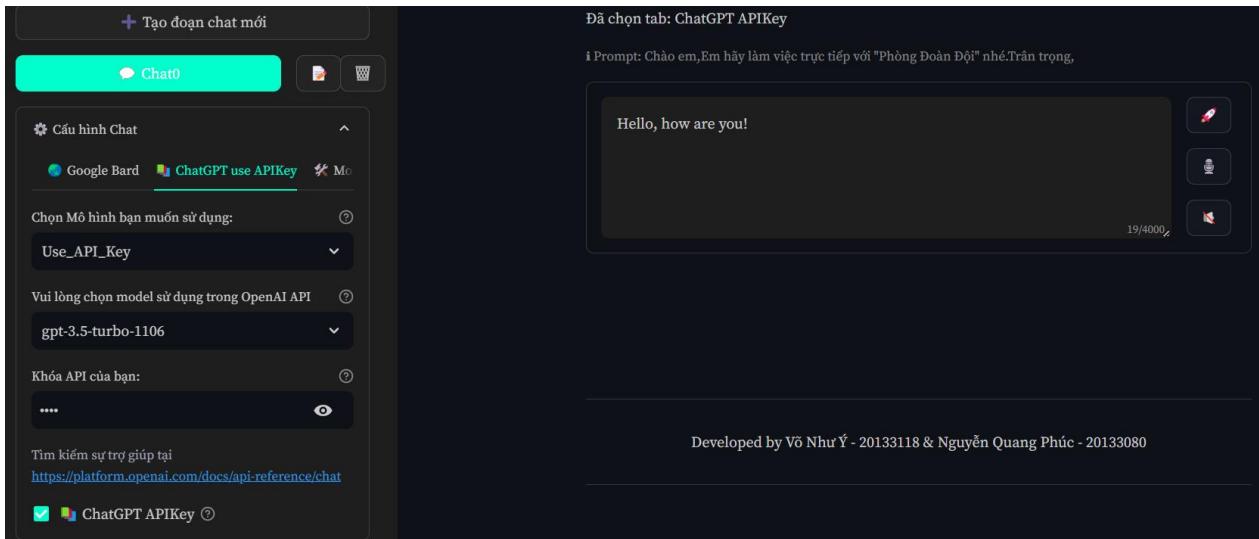


Hình B. 12 Hình ảnh sử dụng Chatbot ở chế độ Google Bard

B.5. Hướng dẫn sử dụng chức năng ChatGPT use APIKey

Đây là một phần nhóm tạo dựng để người dùng có thể sử dụng khóa OpenAI cá nhân của họ để thực hiện các tác vụ về chatbot như sử dụng key để thực hiện sử dụng tính năng trò chuyện hoặc dùng text-davinci-002 để sinh văn bản.

Đầu tiên trong thanh Cấu hình Chat hãy chọn ChatGPT use API key và tích 2 lần vào ô checkbox có tên là ChatGPT APIKey.



Hình B. 13 Hình ảnh chatbot khi sử dụng ChatGPT use APIkey

❖ Sử dụng Use_API_Key

Bước 1: Tích vào Mô hình bạn muốn sử dụng chọn Use_API_Key → Chọn mô hình sử dụng như gpt-3.5-turbo,...

Bước 2: điền Khóa API của bạn vào để sử dụng.

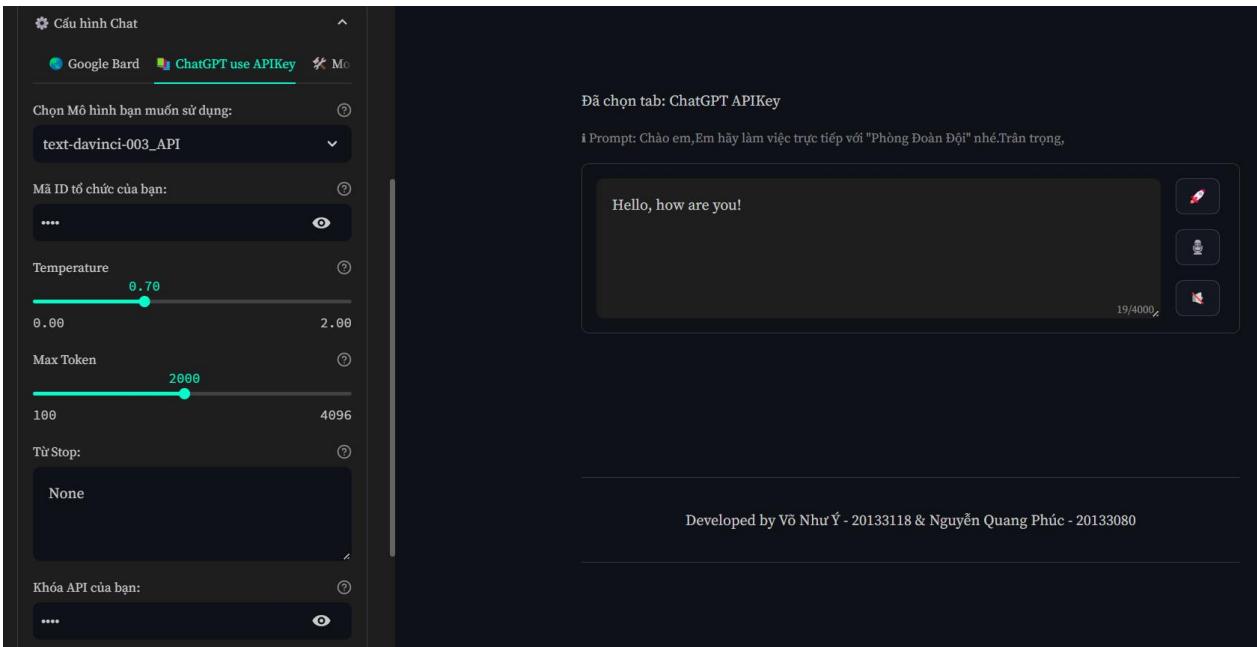
Bước 3: nhập văn bản hoặc sử dụng giọng nói để thực hiện cuộc trò chuyện với chatbot.

❖ Sử dụng text-davinci-003_API

Bước 1: Trong mô hình bạn muốn sử dụng hãy chọn text-davinci-003_API.

Bước 2: Điền khóa API key như phần Sử dụng Use_API_Key ở trên và phải cần điền thêm Mã ID tổ chức. Ngoài ra có thể chọn các tính năng như Từ Stop(từ để dừng trong đoạn văn bản), hoặc Max Token(số lượng từ tối đa) và Temperature.

Sau khi hoàn thành những yêu cầu trên thì chỉ cần sử dụng giọng nói hoặc nút nhấn gửi tin nhắn để thực hiện cuộc trò chuyện.

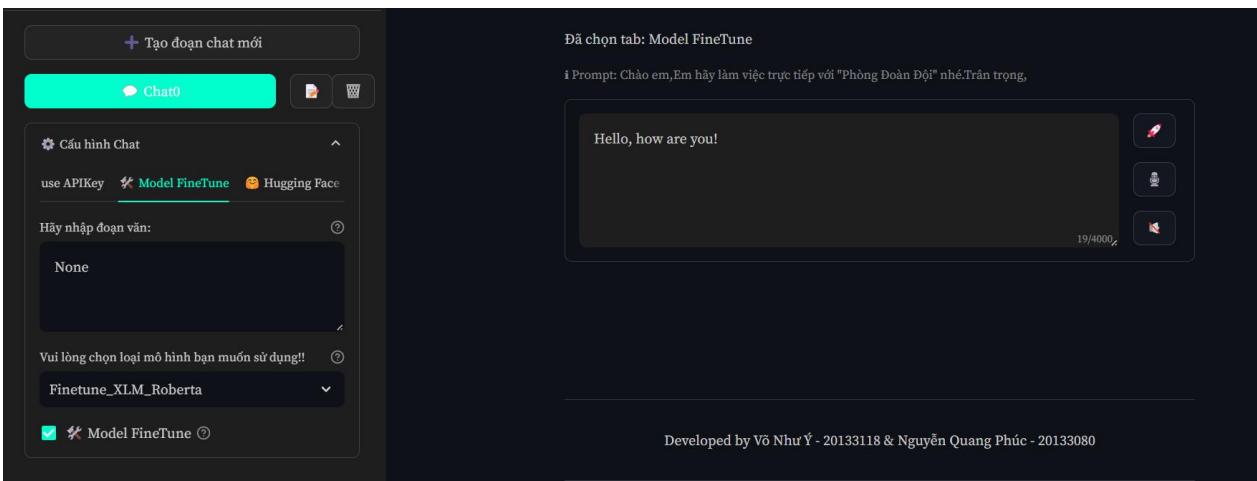


Hình B. 14 Sử dụng mô hình text-davinci-003 trong ChatGPT use APIKey

B.6. Hướng dẫn sử dụng chức năng Model FineTune

Như đã đề cập ở Chương 4 thì nhóm đã xây được các tinh chỉnh cho mô hình ngôn ngữ tiếng Việt như XLM-RoBERTa, PhoBERT,..

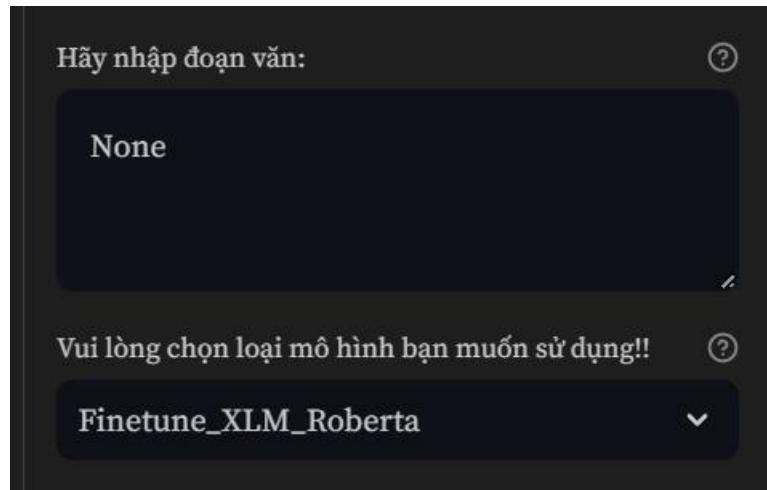
Trong thanh Cấu hình chat hãy tích vào Model FineTune và tích 2 lần vào ô checkbox Model FineTune để chuyển sang chế độ Model FineTune.



Hình B. 15 Giao diện chatbot khi dùng Model FineTune

❖ Sử dụng mô hình XLM-RoBERTa

Bước 1: Trong thanh chọn mô hình bạn muốn sử dụng chọn FineTune_XLM_Roberta. Giao diện được hiển thị như hình bên dưới (Xem hình B. 16)



Hình B. 16 Giao diện khi sử dụng FineTune_XLM_Roberta.

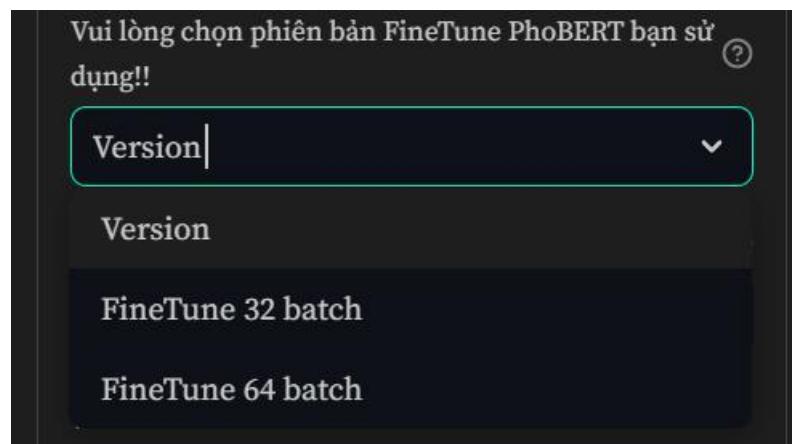
Bước 2: Nhập đoạn văn mà bạn muốn sử dụng trong mô hình vào ô có chữ None.

Bước 3: Đặt câu hỏi cho đoạn văn mà bạn vừa dán vào trong phần nhập tin nhắn và nhấn nút gửi để sử dụng mô hình hoặc sử dụng giọng nói.

❖ Sử dụng mô hình PhoBERT

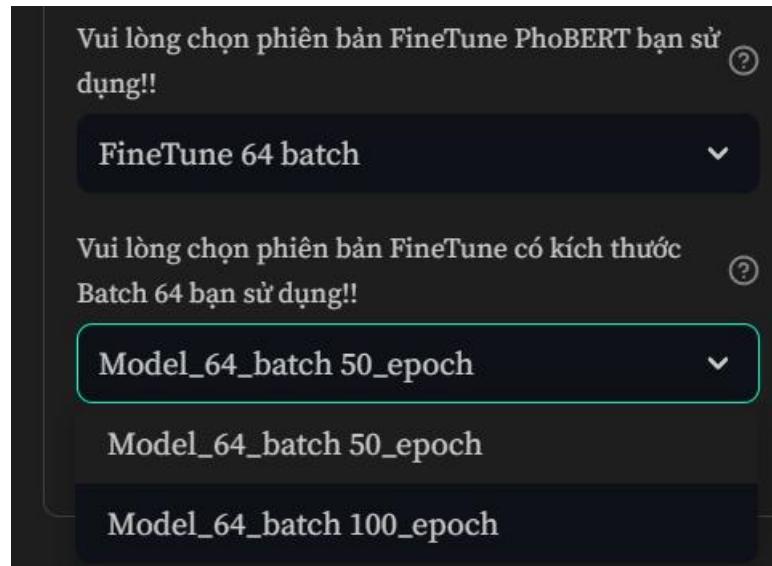
Bước 1: Trong thanh chọn mô hình bạn muốn sử dụng chọn FineTune_PhobERT.

Bước 2: Chọn phiên bản FineTune sử dụng.



Hình B. 17 Hình ảnh các phiên bản tinh chỉnh cho mô hình PhoBERT

Bước 3: Chọn phiên bản tương ứng với mỗi phiên bản FineTune mà bạn sử dụng.

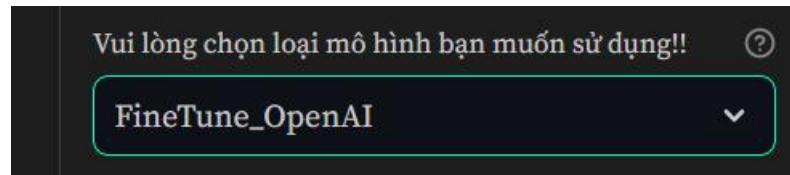


Hình B. 18 Lựa chọn kích thước của phiên bản FineTune sử dụng

Bước 4: Nhập câu hỏi và nhấn nút gửi tin nhắn hoặc sử dụng chức năng chat bằng âm thanh để thực hiện đoạn chat.

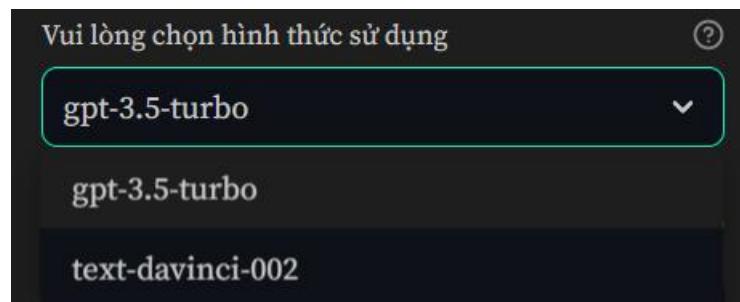
❖ **Sử dụng Mô hình thực hiện tinh chỉnh thông qua OpenAI**

Bước 1: Trong thanh chọn mô hình bạn muốn sử dụng chọn FineTune_OpenAI.



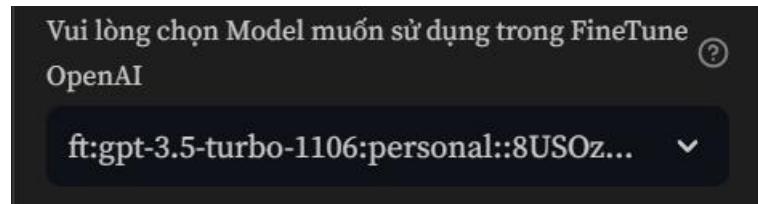
Hình B. 19 Lựa chọn mô hình FineTune_OpenAI trong Model FineTune

Bước 2: Chọn hình thức sử dụng gồm 2 phần là sử dụng gpt-3.5-turbo hoặc text-davinci-002. Nếu sử dụng gpt-3.5-turbo hãy xem bước 2.1, sử dụng text-davinci-002 hãy xem tiếp bước 2.2 (bỏ qua bước 2.1)



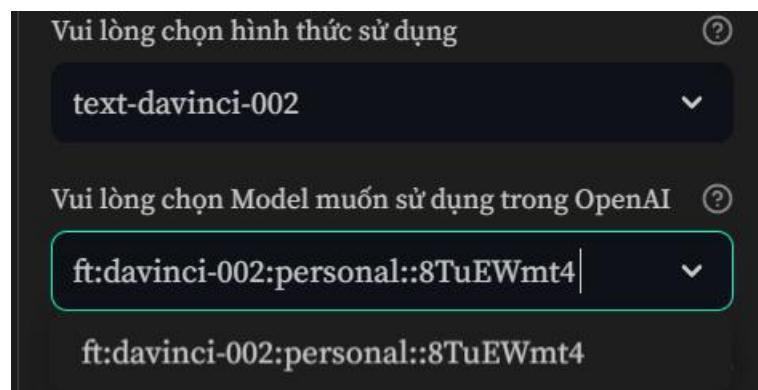
Hình B. 20 Hình thức sử dụng trong FineTune_OpenAI

Bước 2.1: Với hình thức chọn gpt-3.5-turbo sau khi chọn hình thức, tiếp theo chọn Model sử dụng trong OpenAI ví dụ như hình bên dưới



Hình B. 21 Hình ảnh model khi sử dụng hình thức gpt-3.5-turbo ở FineTune_OpenAI

Bước 2.2: Với hình thức chọn text-davinci-002 sau khi chọn hình thức, tiếp theo chọn Model sử dụng trong OpenAI ví dụ như hình bên dưới

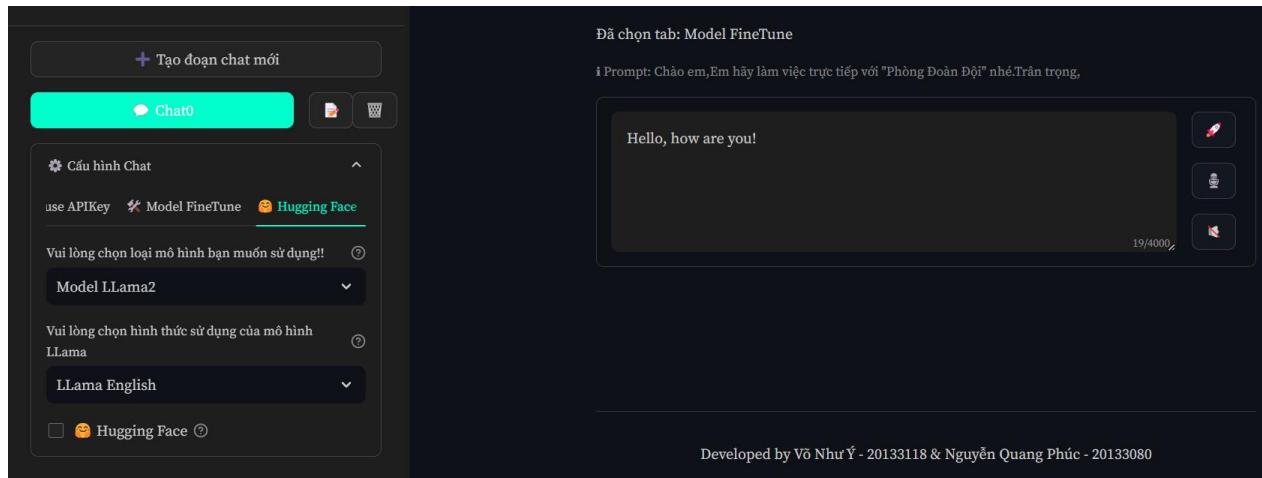


Hình B. 22 Hình ảnh model khi sử dụng hình thức text-davinci-002 ở FineTune_OpenAI

Bước 3: Sau khi hoàn thành xong hãy tiến hành nhập đoạn chat để sử dụng chức năng hoặc dùng giọng nói.

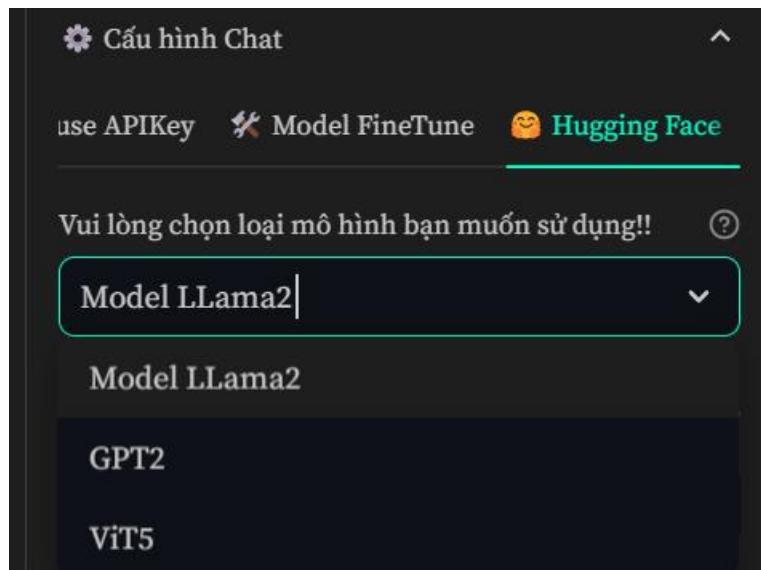
B.7. Hướng dẫn sử dụng chức năng Hugging Face

Bước 1: Trong thanh cấu hình Chat chọn Hugging Face và tích 2 lần vào ô checkbox Hugging Face.



Hình B. 23 Hình ảnh ứng dụng chatbot khi sử dụng Hugging Face

Bước 2: Chọn hình mô hình bạn muốn sử dụng



Hình B. 24 Hình ảnh các mô hình chat ở chế độ Hugging Face

Bước 3: Sau khi chọn mô hình xong hãy nhập tin nhắn hoặc sử dụng giọng nói để thực hiện cuộc trò chuyện với chatbot.