ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH

Thực tập ngoài trường - CO3335

Báo cáo

AUTOFILL SERVICE

Giảng viên hướng dẫn: Trương Quỳnh Chi

Sinh viên thực hiện: 2111762 - Phạm Võ Quang Minh

TP. Hồ Chí Minh, 12/2024

Mục lục

I Giới thiệu doanh nghiệp 4
II Nội dung thực tập 5
III Báo cáo thực tập 7
III.1 Tuần 1 7
III.1.1 Mô tả công việc 7
III.1.2 Kết quả, kiến thức đạt được 7
III.2 Tuần 2 - 3 7
III.2.1 Mô tả công việc 7
III.2.2 Kết quả, kiến thức đạt được 7
III.2.2.1 Biến dữ liệu chưa có cấu trúc về dữ liệu có cấu trúc 7
III.2.2.2 Huấn luyện mô hình NER hoặc fine-tune mô hình có sẵn: 8
III.2.2.3 Học tăng cường (RAG) 9
III.2.2.4 Chức năng gọi hàm/công cụ (function/tool calling) của LLM 10
III.2.2.5 RAG 11
III.2.2.6 Langchain 12
III.2.2.7 Mô hình ngôn ngữ lớn (LLM) 12
III.2.2.8 Thư viện Fastapi 12
III.2.2.9 mupdf 12
III.2.2.10 tesseract 12
III.3 Tuần 4-5-6-7 12
III.3.1 Mô tả công việc 12
III.3.2 Kết quả, kiến thức đạt được 12
III.3.2.1 Prompt cơ bản 12
III.3.2.2 Schema cơ bản 13
III.4 Tuần 8 14
III.4.1 Mô tả công việc 14
III.4.2 Kết quả, kiến thức đạt được 14
III.5 Tuần 9 14
III.5.1 Mô tả công việc 14
III.5.2 Kết quả, kiến thức đạt được 14
III.6 Tuần 10 14
III.6.1 Mô tả công việc 14
III.6.2 Kết quả, kiến thức đạt được 15
III.6.2.1 Thư viện pydantic-dynamic-model 15
III.7 Tuần 11 17
III.7.1 Mô tả công việc 17
III.7.2 Kết quả, kiến thức đạt được 17
III.8 Tuần 12-13 18
III.8.1 Mô tả công việc 18
III.8.2 Kết quả, kiến thức đạt được 18
IV Kết luận 19
IV.1 Những Thành Tựu Kỹ Thuật Chính 19
IV.2 Kết Quả Học Tập Cá Nhân 19
IV.3 Tác Động Của Dự Án 19
IV.4 Đề Xuất Cho Tương Lai 19
Tài liệu tham khảo 20

Danh mục hình ảnh

Hình 1: Logo doanh nghiệp Apollogix 4
Hình 2: Minh họa công tác dãn nhãn dữ liệu [1] 9
Hình 3: Minh họa về cách gọi hàm của mô hình ngôn ngữ lớn 11
Hình 4: Sequence diagram của endpoint /api/ai/invoke 18

Danh mục bảng biểu

Bảng 1: Minh họa về cách chatbot hoạt động dựa trên mô hình ngôn ngữ 10

I Giới thiệu doanh nghiệp

Công ty TNHH Apollogix chuyên cung cấp các giải pháp phần mềm quản lý chuyên dụng trong lĩnh vực giao thông vận tải Thông tin công ty:

- Tên công ty: Công ty TNHH Apollogix
- Tên quốc tế: APOLLOGIX COMPANY LIMITED
- Tên viết tắt: APOLLOGIX CO LTD
- Trụ sở chính: 39 Đường B4, phường An Lợi Đông, Quận 2, thành phố Thủ Đức.
- Văn phòng làm việc: 39 Đường B4, phường An Lợi Đông, Quận 2, thành phố Thủ Đức.
- Điện thoại: 0796513044 (Ms. Huyền).
- Email: contact@apollogix.com.
- Weblink: apollogix.com.
- Logo nhận diện:

Hình 1: Logo doanh nghiệp Apollogix

II Nội dung thực tập

Trong hệ thống TMS có một chức năng gọi là Autofill, chức năng này có input đầu vào là file pdf booking, output mong đợi là đọc từ file pdf này ra cấu trúc json của dữ liệu Transport Job đang có để sau đó sẽ hiển thị data này trên giao diện form Transport Job cho end user xem trước khi lưu lại mà không cần phải nhập. Ngoài ra dịch vụ có thể được sử dụng để tạo đơn hàng có trạng thái chờ xác nhận từ email của khách hàng hoặc trên customer portal sau này.

File PDF Booking này khách hàng có được từ bên các hãng tàu hoặc các dịch vụ bên thứ 3 cung cấp, có chứa các thông tin về lịch cảng tàu, thông tin container cần vận chuyển, v.v

Sinh viên thực hiện: Phạm Võ Quang Minh.

Nhân sự hỗ trợ kỹ thuật: Anh Thi.

Nhân sự hỗ trợ nghiệp vụ: Anh Phúc, Chị Nguyệt, và các nhân sự bên vận hành.

Tuần	Nội dung	Kết quả mong đợi	Tài liệu	Báo cáo
1-2	Tìm hiểu nghiệp vụ và cơ sở dữ liệu liên quan Transport Job / và quản lý Container	Giải thích được các thuật ngữ, các trường dữ liệu trên 2 form liên quan	Thi, Phúc và bộ phận vận hành	Thi /Phúc / Nguyệt
2-3	Tìm hiểu các file. Booking mẫu và mapping trường dữ liệu trên file mẫu sang trường dữ liệu trên giao diện / database	Document mô tả các mapping của các tài liệu	Thi, Phúc và bộ phận vận hành	File doc/excel Thi /Phúc / Nguyệt
4-5-6-7	Tìm hiểu cơ chế input là file template, output là json có cấu trúc quy định được (lấy từ hệ thống TMS) sử dụng chatgpt. Xây dựng demo cho 2 case study	Document. Flowchart của giải pháp	Tài liệu tham khảo thêm ở: https://drive.google.com/drive/folders/1NodU9CvvOZ6Zya7_qS0Xsa5THUNEmJMi?usp=drive_link	Trình bày slide, báo cáo kĩ thuật và demo Thi /Phúc / Nguyệt/ A. Trường
8-9-10	Phát triển thành service api / hoặc tool trong đó input là template pdf/xml, file pdf/ xml, output là	Demo API. Lưu ý mã template có cấu trúc schema được định nghĩa trước.		Trình bày slide, báo cáo kĩ thuật và demo Thi /Phúc / Nguyệt

Tuần	Nội dung	Kết quả mong đợi	Tài liệu	Báo cáo
	json data của hệ thống TMS. Có thể nâng cấp được input là template, schema cấu trúc json cần output, file pdf/ xml cần xử lý, output là data đọc được theo cấu trúc input	Mục tiêu 1: có thể cover được các case pdf/xml template càng nhiều càng tốt. Mục tiêu 2: template pdf/xml động có thể được cung cấp mới mà ko phải chỉnh sửa code Mục tiêu 3: cấu trúc json output có thể động, có thể chỉnh sửa mà không phải chỉnh sửa code. (Lưu ý việc mapping từ ngữ giữa file mẫu và output json, quản lý file mẫu, và cấu trúc json output cần có database lưu trữ)		
11-12-13	Cải tiến, báo cáo và bàn giao	Báo cáo kĩ thuật / báo cáo thực tập với trường/ Slide thuyết trình		Trình bày slide, báo cáo kĩ thuật và demo Thi /Phúc / Nguyệt/ A. Trường

III Báo cáo thực tập

III.1 Tuần 1

III.1.1 Mô tả công việc

- Tìm hiểu nghiệp vụ công ty: Workflow của team vận hành
  1. Công ty nhận đặt hàng qua những form đặt hàng logistics.
  2. Nhân viên công ty tiến hành đọc file booking ở định dạng PDF và nhập liệu vào hệ thống.
  3. Việc nhập liệu này còn thủ công, người nhập phải map từng trường một của file pdf vào 1 form cho trước.
- Task được giao: Tối ưu quá trình nhập liệu bằng cách tạo ra một tool để prefill các giá trị vào form từ file pdf, giảm bớt công việc cho người nhập liệu.
- Setup môi trường: Java, đăng ký vào máy chủ SSH của công ty, Jira, Confluence.
- Làm quen với mentor và các đồng nghiệp.

III.1.2 Kết quả, kiến thức đạt được

- Hiểu được quy trình làm việc công ty, ngữ cảnh của project.
- Làm quen với văn hóa công ty.
- Hiểu được các tool project management nói chung và mục đích của chúng: Jira, Confluence.

III.2 Tuần 2 - 3

III.2.1 Mô tả công việc

- Phân tích các hướng giải quyết bài toán.
- Tìm hiểu các bài toán liên quan: Named Entity Recognition, Optical character recognition, xử lý PDF.
- Tìm hiểu các công nghệ được sử dụng: RAG, langchain, fastapi, mupdf.
- Báo cáo với mentor: tradeoff giữa các hướng tiếp cận.

III.2.2 Kết quả, kiến thức đạt được

III.2.2.1 Biến dữ liệu chưa có cấu trúc về dữ liệu có cấu trúc

Đề xuất hai phương pháp có thể sử dụng:

1. Huấn luyện mô hình:
   a. Xây dựng mô hình: Đưa bài toán về bài toán Nhận dạng Thực thể **NER** (Named Entity Recognition) và sử dụng mô hình học máy kinh điển để xử lý bài toán
   b. Fine-tuning (transfer learning): Huấn luyện mô hình ngôn ngữ để mô hình luôn sinh ra được câu trả lời có cấu trúc từ đầu vào
2. Sử dụng mô hình đã có sẵn (**RAG**): Sử dụng mô hình ngôn ngữ học sâu (deep learning model) như GPT-4o để trích xuất thông tin. Trong đó, người dùng yêu cầu mô hình ngôn ngữ trả về dạng chuỗi (string) theo format của của 1 dạng dữ liệu có cấu trúc như json.
   a. Sử dụng mô hình ngôn ngữ giao tiếp (chat model): chuyển dạng pdf/hình ảnh/... về dạng chuỗi bằng công nghệ OCR.
   b. Sử dụng mô hình đa thể thức (multi modal model): truyền thẳng dạng pdf/hình ảnh/... mà không cần có bước OCR.

III.2.2.2 Huấn luyện mô hình NER hoặc fine-tune mô hình có sẵn:

Để hiện thực hóa giải pháp NER, cần thực hiện các bước sau [2]:

1. Thu thập dữ liệu: Thu thập tập dữ liệu có chứa các văn bản đã được gắn nhãn thực thể. Việc gắn nhãn có thể thực hiện thủ công hoặc bằng các công cụ tự động hóa. Đảm bảo dữ liệu đa dạng và đại diện tốt cho bài toán.
2. Tiền xử lý dữ liệu: Làm sạch dữ liệu, loại bỏ các ký tự không cần thiết. Chuẩn hóa văn bản (ví dụ: chuyển chữ hoa thành chữ thường, xóa khoảng trắng thừa). Tách văn bản thành câu hoặc token (từ hoặc cụm từ).
3. Trích xuất đặc trưng: Sử dụng các kỹ thuật như gán nhãn từ loại (POS tagging), tạo vector từ biểu diễn (word embeddings), và khai thác ngữ cảnh xung quanh từ. Chọn các đặc trưng phù hợp với mô hình NER cụ thể.
4. Huấn luyện mô hình: Sử dụng mô hình học máy (như CRF, SVM) hoặc học sâu (như BiLSTM-CRF, Transformer-based models như BERT) để huấn luyện trên tập dữ liệu gắn nhãn. Điều chỉnh các siêu tham số (hyperparameters) để tối ưu hóa hiệu suất.
5. Đánh giá mô hình: Đánh giá chất lượng mô hình qua các chỉ số như Precision, Recall, và F1 Score. Phân tích các trường hợp mô hình nhận dạng sai để cải thiện.
6. Tinh chỉnh mô hình: Tăng cường dữ liệu huấn luyện hoặc sử dụng kỹ thuật như tăng cường dữ liệu (data augmentation). Áp dụng các phương pháp như học chuyển giao (transfer learning) để cải thiện độ chính xác.
7. Suy luận (Inference): Triển khai mô hình để xử lý các văn bản mới, gắn nhãn các thực thể trong văn bản.
8. Hậu xử lý: Liên kết thực thể với cơ sở tri thức hoặc các nguồn dữ liệu khác để làm phong phú thêm thông tin. Loại bỏ các thực thể không phù hợp hoặc hợp nhất các thực thể trùng lặp.

Ưu điểm:

- Khả năng tự động hóa cao, tiết kiệm thời gian so với các phương pháp thủ công.
- Có thể xử lý dữ liệu văn bản khổng lồ một cách hiệu quả.
- Kết hợp được nhiều loại đặc trưng để cải thiện độ chính xác.

Nhược điểm:

- Yêu cầu tập dữ liệu được gắn nhãn chất lượng cao, chi phí gắn nhãn lớn.
- Hiệu quả của mô hình phụ thuộc mạnh vào chất lượng dữ liệu và đặc trưng.
- Mô hình có thể gặp khó khăn khi xử lý ngôn ngữ không chính thức hoặc lỗi chính tả (tốn kém việc tiền xử lý sửa lỗi chính tả).

Hình 2: Minh họa công tác dãn nhãn dữ liệu [1]

Phân tích độ phù hợp của cách tiếp cận này với bài toán:
- Bộ dữ liệu cung cấp cho công việc huấn luyện quá ít ỏi (khoảng 40 mẫu).
- Việc dán nhãn phải làm thủ công, đòi hỏi người có chuyên môn hoặc quen với việc nhập mẫu (ví dụ: người bên nhánh vận hành của công ty). Nghĩa là chi phí dãn nhãn (thời gian, nhân lực) là quá lớn đối với 1 thực tập sinh trong 1 kỳ thực tập.

Vì vậy phương pháp này là không khả thi.

Tương tự, ta không cần phải đi sâu vào các bước thực hiện fine-tuning, vì nó cũng sẽ yêu cầu một lượng lớn dữ liệu huấn luyện và không khả thi trong bối cảnh hiện tại.

III.2.2.3 Học tăng cường (RAG)

Vì những nhược điểm nêu ra ở hướng tiếp cận trên. Tôi đã lựa chọn phương pháp học tăng cường.

Học tăng cường là phương pháp dựa trên mô hình đã được huấn luyện sẵn (các mô hình ngôn ngữ lớn như GPT-4o, Llama3, Mistral,...).

Phần lớn các mô hình ngôn ngữ lớn chỉ có chức năng nhận vào một đầu vào là một chuỗi nhập của người dùng và trả về một chuỗi kết quả. Năng lực của mô hình ngôn ngữ lớn như vậy là còn quá hạn hẹp.

Một ví dụ cho học tăng cường là ChatGPT hay nhiều chat bot khác, đầu vào và kết quả của lần gọi trước sẽ được nối với đầu vào mới và tạo thành 1 chuỗi, ta sẽ tạo được một mô hình có khả năng trò chuyện với người dùng.

	Người dùng nhập	Đầu vào	Kết quả
1	Xin chào	Human: Xin chào	Xin chào, tôi có thể giúp gì được cho bạn?
2	Tôi muốn đặt câu hỏi về học tăng cường.	Human: Xin chào AI: Xin chào, tôi có thể giúp gì được cho bạn? Human: Tôi muốn đặt câu hỏi về học tăng cường.	Học tăng cường là phương pháp dựa trên mô hình đã được huấn luyện sẵn...

Bảng 1: Minh họa về cách chatbot hoạt động dựa trên mô hình ngôn ngữ

Vì sao chọn phương pháp này phù hợp hơn để giải quyết bài toán thay cho phương pháp huấn luyện mô hình

Nhược điểm:
- Độ chính xác hội tụ sớm sau một thời gian ngắn sử dụng: Vì chúng ta sẽ không huấn luyện mô hình, nên cũng không thể tăng độ chính xác theo thời gian khi kích thước bộ dữ liệu tăng lên.
- Phụ thuộc lớn vào khả năng viết prompt *hay*.

Ưu điểm:
- Không cần nghĩ đến việc huấn luyện: RAG sử dụng các mô hình ngôn ngữ lớn (LLMs) đã được huấn luyện trên kho dữ liệu khổng lồ, không yêu cầu một tập dữ liệu gắn nhãn cụ thể.
- Linh hoạt yêu cầu bài toán: Vì các LLM được huấn luyện dựa trên nhiều ngữ cảnh khác nhau nên có thể thích ứng được với nhiều loại yêu cầu.
- Dễ hiện thực: Không cần phải xây dựng chuỗi huấn luyện.
- Có thể dễ dàng khắc phục nhược điểm: Không thể tăng độ chính xác theo thời gian có thể được bỏ qua nhờ các kĩ thuật:
  - Việc viết lời gợi (prompting) hay hơn
  - Viết lời gợi kèm theo ví dụ (khoảng 2-5 ví dụ): Few-shot Prompting
- Một số mô hình được huấn luyện đặc biệt cho tác vụ gọi hàm (xem bên dưới)

III.2.2.4 Chức năng gọi hàm/công cụ (function/tool calling) của LLM

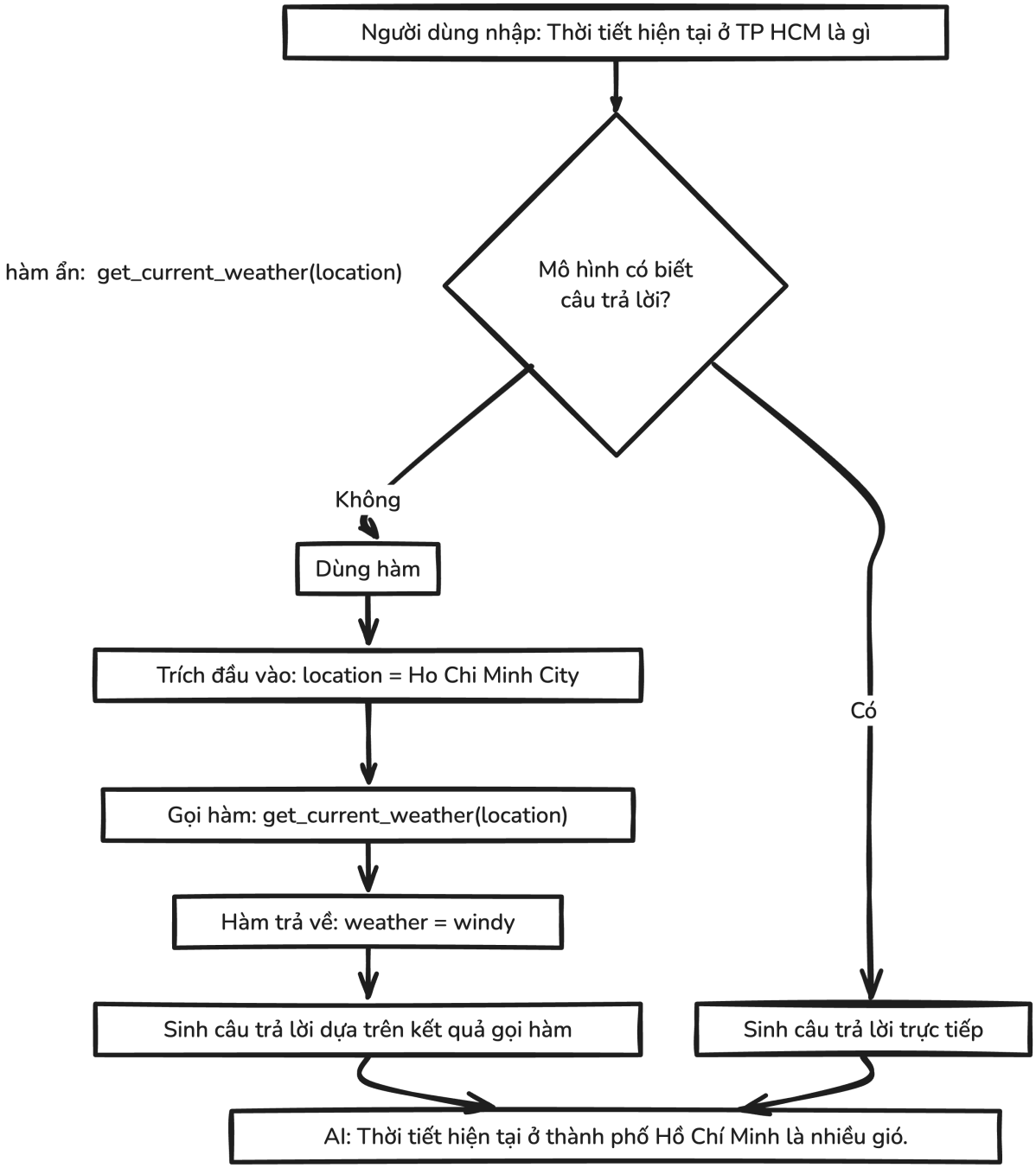
Mô hình ngôn ngữ ở dạng thuần túy nhất không có khả năng lấy được thông tin ở thời gian thực. Vì vậy ta không thể nào đặt câu hỏi mong chờ kết quả của thời gian thực. Các nhà cung cấp mô hình hiện nay đã cung cấp thêm cho mô hình khả năng gọi hàm. Gọi hàm gồm các bước trừu tượng như sau:
1. Trích đầu vào cho một hàm
2. Gọi hàm
3. Sinh ra câu trả lời cho người dùng dựa trên đầu ra của hàm

(Theo dõi hình minh họa ở dưới)

Ứng dụng của function calling để giải quyết bài toán

Dưới một góc độ khác, nếu ta bỏ qua bước trừu tượng 2 và 3, thì khả năng function calling của mô hình ngôn ngữ đơn giản là trích xuất tham số từ cuộc trò chuyện để sử dụng làm đầu vào cho các hàm.

Ta có thể lợi dụng chuyện này, định nghĩa tham số đầu vào cho một hàm giả. Yêu cầu mô hình trích xuất tham số cho hàm giả đó, kết quả thu được sẽ là một mô hình có khả năng trích xuất văn bản.



Hình 3: Minh họa về cách gọi hàm của mô hình ngôn ngữ lớn

III.2.2.5 RAG

Về cơ bản, để hiện thực được một kiến trúc học tăng cường, thì ta chỉ cần các bước sau (có thể lồng ghép các bước sau nhiều lần):

- Chuẩn bị lời gọi: soạn prompt, chuẩn hóa dữ liệu, đóng gói thành lời gọi (request) qua API của nhà cung cấp mô hình ngôn ngữ lớn (ví dụ: OpenAI, các dịch vụ như Groq, Huggingface,...).
- Xử lý kết quả trả về của API.

Tuy nhiên, vì mỗi nhà cung cấp có một cấu trúc riêng về API, giả sử số nhà cung cấp là n. Ta phải thực hiện quy trình trên với độ phức tạp O(n). Ta có thể sử dụng thư viện (thư viện) Langchain để giúp đưa bài toán này về O(1).

III.2.2.6 Langchain

Langchain là một framework giúp nhà phát triển có thể nhanh chóng tạo ra mô hình học tăng cường một cách nhanh chóng bằng cách chuẩn hóa quy trình tạo kiến trúc học tăng cường.

Bên cạnh đó, Langchain đóng gói và trừu tượng hóa bước gọi API đến phần lớn các nhà cung cấp lớn trên thị trường. Vì thế, người dùng chỉ cần xây dựng kiến trúc và yêu cầu Langchain gọi dịch vụ. Nhờ đó mà nhà phát triển kiến trúc học tăng cường có thể thử nghiệm trên nhiều loại mô hình ngôn ngữ lớn khác nhau.

III.2.2.7 Mô hình ngôn ngữ lớn (LLM)

Vì Langchain cho phép dễ dàng thay đổi linh hoạt nhiều mô hình ngôn ngữ mà không cần phải xây dựng lại kiến trúc học tăng cường, tôi lựa chọn các mô hình ngôn ngữ được cung cấp bởi Meta (Llama 3.2 và các phiên bản tương tự), Google (Gemini, Vertex,...) vì họ cung cấp API miễn phí đối với cá nhân nhà phát triển. Đồng thời tôi cũng sẽ sử dụng các model được huấn luyện thêm (finetune) cho tác vụ gọi hàm (function calling) trên nền tảng Huggingface. Sau khi xây dựng kiến trúc học tăng cường hoạt động tốt thì sẽ cân nhắc chuyển sang dùng các model của OpenAI (GPT-4o,...) hay Anthropic (Claude Sonnet 3.5,...).

III.2.2.8 Thư viện Fastapi

Để hiện thực nhanh chóng microservice này, ta có thể sử dụng thư viện FastAPI của Python, thư viện này cho phép nhà phát triển nhanh chóng viết ra dịch vụ bởi hướng tiếp cận đơn giản và hướng dẫn sử dụng kĩ, cùng với đó là hệ sinh thái tốt và hỗ trợ lập trình async qua uvicorn server async (ASGI server) hoặc multi-processing (uvicorn workers).

III.2.2.9 mupdf

Có hai thư viện được sử dụng rộng rãi cho loại dữ liệu pdf là mupdf và poppler. Tôi quyết định lựa chọn mupdf vì đây là thư viện hiện đại và có tốc độ xử lý nhanh hơn poppler.

mupdf còn có thư viện bindings qua python là pymupdf, rất thuận tiện cho tác vụ xử lý file pdf.

III.2.2.10 tesseract

Là công cụ OCR nhỏ gọn có thể chạy trong local, vì dữ liệu đầu vào là một mẫu pdf khá nhỏ (1-2 trang), và mẫu pdf là dữ liệu theo thể thức chứ không phải chữ viết tay nên không đòi hỏi dịch vụ OCR mạnh hơn của bên thứ ba.

III.3 Tuần 4-5-6-7

III.3.1 Mô tả công việc

- Xây dựng extraction chain bằng Langchain phù hợp với nghiệp vụ công ty.
- Prototype extraction chain: Chuẩn bị prompt cơ bản, pipeline data để trích text từ pdf sang dạng string. Sử dụng mupdf và OCR.

III.3.2 Kết quả, kiến thức đạt được

III.3.2.1 Prompt cơ bản

Prompt sẽ bảo gồm:
- Một lời nhắn của hệ thống (system message), để tạo cho mô hình

ngôn ngữ hiểu được ngữ cảnh của công việc.

- Những gợi ý (tips) thông qua lời nhắn của con người (human message), để chỉ dẫn, đưa ra gợi ý cho model trả lời phụ hợp với ý của người sử dụng hơn.
- Một lời nhắn đầu vào của con người, chúng ta sẽ để trống tin nhắn này và truyền vào khi gọi chain (xem phần Extraction chain).

```
from langchain_core.messages import SystemMessage
from langchain_core.prompts import ChatPromptTemplate, MessagesPlaceholder
from pydantic import BaseModel

messages = [
    SystemMessage(
        content="Extract shipping job details from PDF text. "
        "If any value is missing, return default value. "
        "Use field descriptions for guidance, and please use enum values "
        "if provided",
    ),
    ("human", "This is what you have to extract from: {context}")
]
simple_extraction_prompt = ChatPromptTemplate.from_messages(messages)
```

III.3.2.2 Schema cơ bản

Ta có thể định nghĩa một schema cố định như sau:

```
from pydantic import BaseModel, Field
class Container(BaseModel):
    containerNumber: Optional[str] = Field(default=None, description="")
    sealNumber: Optional[str] = Field(default=None, description="")
    dropMode: Optional[str] = Field(
        default=None,
        description="",
        enum=[
            "Sideloader Wait Unpacking/Packing",
            "Standard Trailer-Drop Trailer",
            "Standard Trailer Wait Unpacking/Packing",
            "Sideloader",
        ],
    )
    grossWeight: int = Field(default=0, description="")
    doorType: str = Field(default="any", enum=["any", "rear", "fwd"])
class Job(BaseModel):
    jobType: str = Field(
        ...,
        description="Classify job type based on context, possible values: "
        "IMPORT, EXPORT, MISC, RAIL_INBOUND, RAIL_OUTBOUND",
        enum=["IMPORT", "EXPORT", "MISC", "RAIL_INBOUND", "RAIL_OUTBOUND"],
    )
    cusRefId: Optional[str] = Field(
        default=None, description="customer referal id")
        vessel: Optional[str] = Field(
        default=None, description='Vessel name (e.g., "MV Oceanic")'
    )
    vessel: Optional[str] = Field(
        default=None, description='Vessel name (e.g., "MV Oceanic")'
    )
    voyage: Optional[str] = Field(
        default=None, description='Voyage name (e.g., "VOY123")'
    )
```

```
portOfLoading: Optional[str] = Field(
    default=None, description="loading port location"
)
portOfDischarge: Optional[str] = Field(
    default=None, description="discharge port location"
)
eta: Optional[datetime] = Field(default=None, description="")
etd: Optional[datetime] = Field(default=None, description="")
agentName: Optional[str] = Field(default=None, description="")
consigneeName: Optional[str] = Field(default=None, description="")
consignorName: Optional[str] = Field(default=None, description="")
warehouseName: Optional[str] = Field(default=None, description="")
accountReceivableName: Optional[str] = Field(default=None, description="")
# Lồng kiểu Container trong kiểu Job
jobContainers: List[Container] = Field(
    default=None,
    description="A list container information according to the schema",
)
```

III.4 Tuần 8

III.4.1 Mô tả công việc

- Test extraction chain trên nhiều LLM khác nhau của nhiều nhà cung cấp: Groq, Google Gemini, OpenAI, Anthropic.
- Refine prompt để nâng cao độ chính xác kết quả - trong lúc thử nghiệm không có model nào cho kết quả mong muốn, ví dụ sai format output.

III.4.2 Kết quả, kiến thức đạt được

- Chỉ có 2 LLM là Llama-3.2-70b và Llama-3.2-8b được Groq finetune cho tác vụ function calling cho ra một kết quả chấp nhận được.

III.5 Tuần 9

III.5.1 Mô tả công việc

- Chuẩn bị data: Xây dựng module cho phép thêm ví dụ vào prompt nhằm tăng độ chính xác của kết quả nhờ kĩ thuật few-shot prompting.
- Dùng extraction chain protoype để generate 1 bộ example.
- Viết 1 web UI để có thể hợp tác với team vận hành sửa lỗi mà model sinh ra.

III.5.2 Kết quả, kiến thức đạt được

- Giao diện web UI.
- Test data example.

III.6 Tuần 10

III.6.1 Mô tả công việc

- Viết thư viện pydantic-dynamic-model cho phép định nghĩa schema một cách linh động.

III.6.2 Kết quả, kiến thức đạt được

III.6.2.1 Thư viện pydantic-dynamic-model

- Thư viện Python cho phép đọc định nghĩa của schema theo cấu trúc của ModelDefinition cơ bản như sau (xem source code của thư viện tại:

github.com/qmi03/pydantic_dynamic_model):

```
class ModelDefinition(BaseModel):
    model_name: str = Field(..., pattern="^[a-zA-Z_][a-zA-Z0-9_]*$")
    fields: List[FieldDefinition]
```

Lớp ModelDefinition
- model_name (kiểu: string): Định nghĩa tên của schema, và
- fields (kiểu: [FieldDefinition): Định nghĩa danh sách các miền dữ liệu ở thuộc tính fields.

Trong đó, lớp FieldDefinition được định dạng như sau:

```
class FieldDefinition(Frozen):
    name: str = Field(..., pattern="^[a-zA-Z_][a-zA-Z0-9_]*$")
    base_type: Union[SimpleType, "ModelDefinition"]
    wrappers: List[WrapperType] = Field(default_factory=list)
    required: bool = True
    default: Optional[Any] = None
    description: Optional[str] = None
    validator_defs: List[FieldValidatorDef] = Field(default_factory=list)
    enum_values: Optional[List[str]] = None
    metadata: Dict[str, Any] = Field(default_factory=dict)
```

bao gồm:

- name (kiểu: string) Tên của miền dữ liệu.
- base_type (kiểu: SimpleType hoặc ModelDefinition): kiểu dữ liệu gốc, gồm các giá trị cơ bản như “string”, “integer”, “float”, “boolean”, “datetime”, “date”, và thuộc kiểu ModelDefinition, cho phép dữ liệu được lồng. Ví dụ, trong trường hợp của chúng ta: kiểu Container được lồng trong kiểu Job.
- wrapper_type (kiểu: [WrapperType]): Kiểu dữ liệu lồng lấy kiểu dữ liệu gốc (base_type). Các kiểu lồng hỗ trợ: là list, dict, enum. Nhập danh sách rỗng nếu không có kiểu lồng.
- required (kiểu: bool): Đánh dấu liệu trường này có bắt buộc không. Mặc định là True.
- default (kiểu: Optional[Any]): Giá trị mặc định của trường nếu có.
- description (kiểu: Optional[str]): Mô tả ngắn gọn về trường dữ liệu.
- enum_values (kiểu: Optional[List[str]]): Giá trị liệt kê cho các trường kiểu enum.
- metadata (kiểu: Dict[str, Any]): Metadata liên quan đến trường.

Hàm tạo ra schema linh động: def create_dynamic_model(model_definition: ModelDefinition) -> type[BaseModel]

Ví dụ cách sử dụng hàm trong hệ thống hiện tại:
1. Đọc định nghĩa của schema trong schema.json và chuyển về dạng ModelDefinition
2. Gọi hàm create_dynamic_model()

```
import json
schema_definition = json.loads("./schema.json")
parsed_schema_definition = ModelDefinition.model_validate(schema_definition)

schema = create_dynamic_model(parsed_schema_definition)
```

```
// Ví dụ ngắn của schema.json
{
  "model_name": "BaseTransportJobInformation",
  "fields": [
    {
      "name": "jobType",
      "base_type": "string",
      "wrappers": [],
      "required": true,
      "default": null,
      "description": "Classify job type based on context, possible values: IMPORT, EXPORT, MISC, RAIL_INBOUND,
RAIL_OUTBOUND",
      "validator_defs": [
        {
          "validator_type": "custom",
          "params": {
            "customFunctionDef": "def validate(cls, v):\n    valid_types = ['IMPORT', 'EXPORT', 'MISC', 'RAIL_INBOUND',
'RAIL_OUTBOUND']\n    if v.upper() not in valid_types:\n        raise ValueError(f'Invalid jobType: {v}')\n    return
v.upper()\n    "
          },
          "error_message": "Invalid job type"
        }
      ],
      "enum_values": [
        "IMPORT",
        "EXPORT",
        "MISC",
        "RAIL_INBOUND",
        "RAIL_OUTBOUND"
      ],
      "metadata": {}
    },
    {
      "name": "voyage",
      "base_type": "string",
      "wrappers": ["optional"],
      "required": false,
      "default": null,
      "description": "Voyage name (e.g., \"VOY123\")",
      "validator_defs": [],
      "enum_values": null,
      "metadata": {}
    },
    {
      "name": "accountReceivableName",
      "base_type": "string",
      "wrappers": ["optional"],
      "required": false,
      "default": null,
      "description": "",
      "validator_defs": [],
      "enum_values": null,
```

```
      "metadata": {}
    },
    {
      "name": "jobContainers",
      "base_type": {
        "model_name": "Container",
        "fields": [
          {
            "name": "grossWeight",
            "base_type": "float",
            "wrappers": [],
            "required": true,
            "default": null,
            "description": "",
            "validator_defs": [],
            "enum_values": null,
            "metadata": {}
          },
        ]
      },
      "wrappers": ["optional", "list"],
      "required": false,
      "default": null,
      "description": "A list container information according to the schema",
      "validator_defs": [],
      "enum_values": null,
      "metadata": {}
    }
  ]
}
```

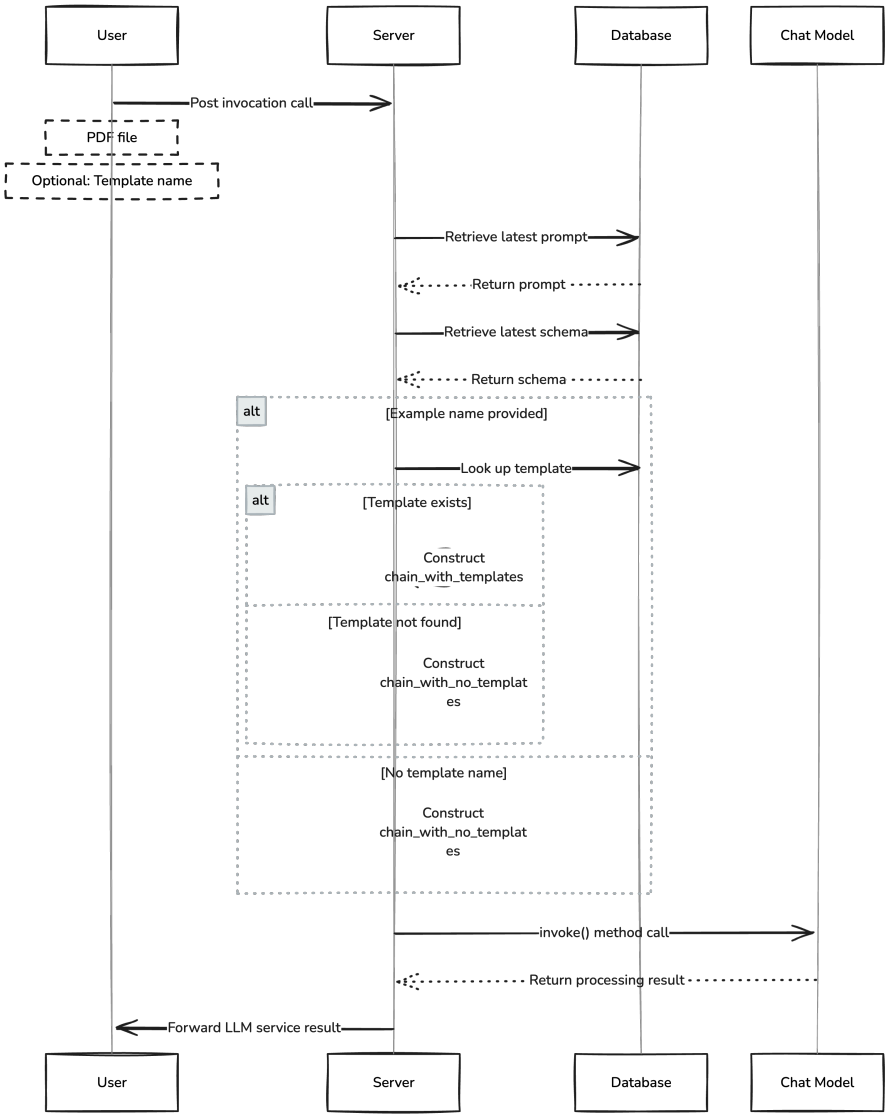

III.7 Tuần 11

III.7.1 Mô tả công việc

- Xây dựng Web API service bằng thư viện FastAPI để cung cấp dịch vụ autofill cho hệ thống TMS.

III.7.2 Kết quả, kiến thức đạt được

- 1 flow để tích hợp với hệ thống như sau:



Hình 4: Sequence diagram của endpoint /api/ai/invoke

III.8 Tuần 12-13

III.8.1 Mô tả công việc

- Deploy trên test server.
- Viết build configuration + CI/CD: Dockerfile, flake.nix, v.v
- Viết user guide cho hệ thống.

III.8.2 Kết quả, kiến thức đạt được

- Biết cách deploy trên nhiều kiến trúc khác nhau (x86-64, aarch64,...)
- Hiểu cách viết Dockerfile
- Sử dụng nix để setup project
- Biết cách viết documentation

IV Kết luận

Trong suốt 13 tuần thực tập, tôi đã thành công phát triển một giải pháp đổi mới để tối ưu hóa quy trình nhập liệu logistics bằng cách tạo ra một hệ thống điền form tự động sử dụng AI. Dự án đã giải quyết một thách thức quan trọng trong quy trình vận hành của công ty, nơi việc nhập liệu PDF thủ công rất tốn thời gian và dễ xảy ra sai sót.

IV.1 Những Thành Tựu Kỹ Thuật Chính

1. **Phương Pháp Công Nghệ**
   - Triển khai chiến lược Học Tăng Cường (RAG) sử dụng các mô hình ngôn ngữ lớn (LLMs)
   - Phát triển chuỗi trích xuất với Langchain để phân tích dữ liệu thông minh
   - Tạo thư viện sinh schema động (pydantic-dynamic-model)
2. **Công Nghệ và Công Cụ**
   - Ngôn ngữ lập trình: Python
   - Framework: FastAPI, Langchain
   - Thư viện: MuPDF, Tesseract OCR
   - Mô hình Ngôn ngữ Lớn: Thử nghiệm mô hình từ Groq, Google, Meta (Llama)
3. **Giải Pháp Sáng Tạo**
   - Phát triển giao diện web để hợp tác sửa lỗi
   - Tạo hệ thống định nghĩa schema linh hoạt
   - Triển khai luồng tích hợp API dựa trên sequence

IV.2 Kết Quả Học Tập Cá Nhân

- Hiểu sâu về các kỹ thuật xử lý ngôn ngữ tự nhiên
- Kinh nghiệm thực tế về tích hợp mô hình học máy
- Kiến thức thực tiễn về phát triển API và kiến trúc microservice
- Tiếp xúc với các nhà cung cấp LLM và khả năng của họ
- Kỹ năng về kỹ thuật prompt và tối ưu hóa mô hình

IV.3 Tác Động Của Dự Án

Hệ thống được phát triển đáng kể giảm thiểu thời gian nhập liệu thủ công, cải thiện độ chính xác của dữ liệu và cung cấp một giải pháp có thể mở rộng cho việc xử lý thông tin logistics của công ty. Thiết kế mô-đun cho phép dễ dàng điều chỉnh và nâng cấp trong tương lai.

IV.4 Đề Xuất Cho Tương Lai

- Mở rộng hỗ trợ cho nhiều loại tài liệu khác
- Triển khai các cơ chế phát hiện và sửa lỗi nâng cao hơn

Tài liệu tham khảo

[1] [Online]. Available at: https://medium.com/data-science-in-your-pocket/named-entity-recognition-ner-using-conditional-random-fields-in-nlp-3660df22e95c

[2] [Online]. Available at: https://www.ibm.com/topics/named-entity-recognition