

LỜI CAM ĐOAN

Tôi xin cam đoan rằng khóa luận tốt nghiệp với đề tài "Xây dựng chatbot hỗ trợ xử lý công văn đến của Trường Đại học Trà Vinh" là công trình nghiên cứu do chính tôi thực hiện dưới sự hướng dẫn của giảng viên hướng dẫn. Toàn bộ số liệu, kết quả và nội dung được trình bày trong khóa luận đều là trung thực, phản ánh đúng quá trình nghiên cứu và thử nghiệm của bản thân tôi.

Các tài liệu tham khảo được sử dụng trong khóa luận đều đã được trích dẫn, ghi nguồn đầy đủ và tuân thủ đúng quy định. Tôi xin chịu hoàn toàn trách nhiệm về tính chính xác, trung thực và tính nguyên bản của công trình nghiên cứu này.

Vĩnh Long, ngày ... tháng ... năm 2025

Sinh viên thực hiện

Dương Thành Tân

LỜI CẢM ƠN

Trước hết, em xin bày tỏ lòng biết ơn sâu sắc đến thầy Nguyễn Bảo Ân, giảng viên hướng dẫn của em. Thầy đã tận tình chỉ bảo, định hướng và hỗ trợ em trong suốt quá trình thực hiện đề tài. Những kiến thức chuyên môn, kinh nghiệm quý báu cùng sự khích lệ và động viên của thầy đã giúp em vượt qua nhiều khó khăn, từng bước hoàn thiện đề tài “Xây dựng chatbot hỗ trợ xử lý công văn đến của Trường Đại học Trà Vinh”.

Em cũng xin chân thành cảm ơn quý thầy cô trong Khoa Kỹ thuật và Công nghệ – Trường Đại học Trà Vinh, đã truyền đạt cho em nền tảng kiến thức vững chắc và tạo điều kiện thuận lợi trong suốt quá trình học tập cũng như nghiên cứu. Đây là những hành trang quý giá để em hoàn thành khóa luận và phát triển hơn nữa trong tương lai.

Mặc dù đã cố gắng hết sức để hoàn thiện, nhưng sẽ còn những thiếu sót không thể tránh khỏi. Em rất mong nhận được sự góp ý từ quý thầy cô để có thể rút kinh nghiệm và nâng cao năng lực trong các dự án tương lai.

Em xin chân thành cảm ơn!

(Của giảng viên hướng dẫn trong đề án, khoá luận của sinh viên)

This image shows a full page of white paper with horizontal dotted lines, typical of primary school writing paper. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

iii

(Của giảng viên hướng dẫn)

MSSV: 110121097

Khóa: 2021 - 2025

Họ và tên Giáo viên hướng dẫn: Nguyễn Bảo Ân

Chức danh: Học vi: Tiến Sĩ

NHẬN XÉT

1. Nội dung đề tài:

This image shows a blank sheet of white paper with horizontal ruling lines. The lines are evenly spaced and extend across the width of the page. There are no margins, text, or other markings on the paper.

2. Ưu điểm:

.....

.....

.....

.....

3. Khuyết điểm:

.....

.....

.....

.....

.....

4. Điểm mới đề tài:

.....

.....

.....

.....

.....

5. Giá trị thực trên đề tài:

.....

.....

.....

.....

.....

.....

.....

7. Đề nghị sửa chữa bổ sung:

.....

.....

.....

.....

.....

.....

.....

8. Đánh giá:

.....

.....

.....

.....

Vĩnh Long, ngày tháng năm 2025
Giảng viên hướng dẫn
(Ký & ghi rõ họ tên)

(Của giảng viên chấm trong đề án, khoá luận của sinh viên)

This image shows a full page of white paper with horizontal dotted lines, typical of primary school writing paper. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

vi

BẢN NHẬN XÉT ĐỒ ÁN, KHÓA LUẬN TỐT NGHIỆP

(Của cán bộ chấm đồ án, khóa luận)

Họ và tên người nhận xét:

Chức danh: Học vị:

Chuyên ngành:

Cơ quan công tác:

Tên sinh viên:

Tên đề tài đồ án, khóa luận tốt nghiệp:

.....
.....

I. Ý KIẾN NHẬN XÉT

1. Nội dung:

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

2. Điểm mới các kết quả của đồ án, khóa luận:

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

3. Ứng dụng thực tế:

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

(Các câu hỏi của giáo viên phản biện)

III. KẾT LUẬN

Người nhận xét
(Ký & ghi rõ họ tên)

MỤC LỤC

CHƯƠNG 1. ĐẶT VẤN ĐỀ	1
1.1. Lý do chọn đề tài.....	1
1.2. Mục tiêu nghiên cứu	1
1.3. Phạm vi nghiên cứu	1
1.4. Phương pháp nghiên cứu	1
CHƯƠNG 2. CƠ SỞ LÝ THUYẾT	2
2.1. Tổng quan về mô hình ngôn ngữ Generative Pre-trained Transformer – Open Source Series (GPT-OSS)	2
2.1.1. Giới thiệu về mô hình ngôn ngữ GPT-OSS.....	2
2.1.2. Kiến trúc và nguyên lý hoạt động	3
2.1.3. Cơ chế huấn luyện và tối ưu	4
2.1.4. Đặc điểm và ưu điểm nổi bật.....	6
2.1.5. Hạn chế và thách thức	8
2.1.6. So sánh với các mô hình thương mại khác.....	10
2.1.7. Ứng dụng của GPT-OSS trong xử lý ngôn ngữ tự nhiên	11
2.2. Tổng quan về Retrieve–Augment–Generate (RAG)	14
2.2.1. Nguyên lý hoạt động của RAG	14
2.2.2. Ưu điểm và hạn chế của RAG.....	15
2.2.3. Ứng dụng của RAG trong thực tiễn	18
2.3. Tổng quan về Model Context Protocol (MCP)	19
2.3.1. Khái niệm về MCP	19
2.3.2. Cơ chế hoạt động của MCP	20
2.3.3. Vai trò của MCP trong hệ sinh thái AI.....	21
2.4. Tổng quan về Tool Calling	22
2.4.1. Khái niệm về Tool Calling.....	22
2.4.2. Cơ chế hoạt động	23
2.4.3. So sánh Tool Calling và Function Calling.....	24
2.4.4. Ứng dụng thực tiễn.....	25
2.5. Tổng quan về Harmony Response Format	25
2.5.1. Khái niệm và mục đích sử dụng	25
2.5.2. Cấu trúc của Harmony Response Format.....	26

2.5.3. Ưu điểm và ứng dụng	27
2.6. Tổng quan về công nghệ phát triển ứng dụng web	28
2.6.1. Ngôn ngữ đánh dấu HyperText Markup Language (HTML).....	28
2.6.2. Ngôn ngữ tạo kiểu Cascading Style Sheets (CSS)	29
2.6.3. Ngôn ngữ lập trình JavaScript.....	30
2.6.4. Tailwind CSS	31
2.6.5. Ngôn ngữ lập trình Python	31
2.6.6. Node.js.....	32
2.6.7. Framework Litestar Python	33
2.6.8. Cơ sở dữ liệu quan hệ PostgreSQL	34
2.7. Tổng quan về nền tảng và công cụ hỗ trợ	35
2.7.1. Runpod	35
2.7.2. Hugging Face.....	37
2.7.3. Google Colab.....	38
2.7.4. Google Apps Script.....	38
CHƯƠNG 3. HIỆN THỰC HÓA NGHIÊN CỨU.....	40
3.1. Mô tả bài toán	40
3.2. Phân tích yêu cầu và lựa chọn công nghệ.....	40
3.2.1. Phân tích yêu cầu.....	40
3.2.2. Yêu cầu chức năng.....	41
3.2.3. Yêu cầu phi chức năng	42
3.2.4. Lựa chọn công nghệ	43
3.3. Thiết kế kiến trúc hệ thống.....	44
3.3.1. Thiết kế kiến trúc phần mềm	44
3.3.2. Thiết kế cơ sở dữ liệu	46
3.3.3. Các thực thể.....	47
3.4. Xây dựng dataset trên Google Colab.....	62
3.4.1. Thu nhập dữ liệu thô và tải lên Google Drive	62
3.4.2. Thiết lập môi trường và khởi tạo dự án trên Google Colab	62
3.4.3. Thiết lập hàm OCR và kết nối API với model Gemma.....	63
3.4.4. Trực quan hóa dữ liệu.....	64
3.5. Xây dựng API phía máy chủ (Back-end).....	66

3.5.1. Các API chức năng xác thực.....	66
3.5.2. Các API chức năng chat	67
3.5.4. Các API hệ thống chung	68
CHƯƠNG 4. KẾT QUẢ NGHIÊN CỨU	69
4.1. Giao diện đăng nhập	69
4.2. Giao diện đăng ký	69
4.3. Giao diện xác thực	70
4.4. Giao diện quên mật khẩu	71
4.5. Giao diện chat	71
4.5. Giao diện trang quản trị	73
CHƯƠNG 5. KẾT LUẬN	75
5.1. Kết luận	75
5.2. Hướng phát triển	75
DANH MỤC TÀI LIỆU THAM KHẢO	76

DANH MỤC CÁC BẢNG

Bảng 3. 1. Bảng phòng ban	47
Bảng 3. 2. Bảng người dùng.....	48
Bảng 3. 3. Bảng thiết lập người dùng.....	49
Bảng 3. 4. Bảng dự án	50
Bảng 3. 5. Bảng Model AI.....	50
Bảng 3. 6. Bảng lịch sử chat.....	51
Bảng 3. 7. Bảng tài liệu	52
Bảng 3. 8. Bảng đính kèm tài liệu	53
Bảng 3. 9. Bảng tin nhắn chat.....	53
Bảng 3. 10. Bảng phiên bản chat.....	54
Bảng 3. 11. Bảng phản hồi chat.....	54
Bảng 3. 12. Bảng email hẹn gửi	55
Bảng 3. 13. Bảng đính kèm Email.....	56
Bảng 3. 14. Bảng liên kế email và phòng ban.....	56
Bảng 3. 15. Bảng danh sách công cụ.....	57
Bảng 3. 16. Bảng công cụ đặc biệt trong chat.....	57
Bảng 3. 17. Bảng xác thực.....	58
Bảng 3. 18. Bảng token khôi phục mật khẩu.....	58
Bảng 3. 19. Bảng cấu hình hệ thống.....	59
Bảng 3. 20. Bảng liên kết công cụ chat.....	60
Bảng 3. 21. Bảng log lịch sử hệ thống	60
Bảng 3. 22. Bảng thông báo hệ thống	61
Bảng 3. 23. Bản người nhận hệ thống	61

MỤC LỤC CÁC ẢNH

Hình 2. 1. Minh họa GPT-OSS-20b (OpenAI, 2025). [1]	2
Hình 2. 2. Minh họa biểu đồ so sánh hiệu năng GPT-OSS với các mô hình khác [1]	2
Hình 2. 3. Minh họa mức ảnh hưởng của độ dài chuỗi suy luận GPT-OSS [2]	6
Hình 2. 4. Minh họa kiến trúc truyền thống và chuẩn hóa với MCP	20
Hình 2. 5. Minh họa cơ chế Tool Calling với tham số JSON.....	24
Hình 3. 1. Mô hình thực thể kết hợp.....	46
Hình 3. 2. Mô hình vật lý	47
Hình 3. 3. Minh họa biểu đồ phân bố loại văn bản hành chính.....	64
Hình 3. 4. Minh họa biểu đồ thống kê 20 đơn vị được phân công	65
Hình 3. 5. Phân bố độ dài văn bản trong tập dữ liệu	65
Hình 4. 1. Minh họa giao diện đăng nhập.....	69
Hình 4. 2. Minh họa giao diện đăng ký	70
Hình 4. 3. Minh họa giao diện nhận mã xác thực.....	70
Hình 4. 4. Minh họa giao diện quên mật khẩu	71
Hình 4. 5. Minh họa giao diện chat	72
Hình 4. 6. Minh họa cách model lựa chọn chế độ trả lời	73
Hình 4. 7. Minh họa cách model trả lời.....	73
Hình 4. 8. Minh họa giao diện trang quản trị	73

DANH MỤC TỪ VIẾT TẮT

Từ viết tắt	Ý nghĩa
AI	Artificial Intelligence
NLP	Natural Language Processing
GPT	Generative Pre-trained Transformer
API	Application Programming Interface
DBMS	Database Management System
UI/UX	User Interface / User Experience
HTML	HyperText Markup Language
CSS	Cascading Style Sheets

CHƯƠNG 1. ĐẶT VẤN ĐỀ

1.1. Lý do chọn đề tài

Trong bối cảnh hiện nay, khối lượng văn bản hành chính và thông tin cần xử lý tại các cơ quan, tổ chức ngày càng lớn, việc quản lý và phân loại thủ công không chỉ mất nhiều thời gian mà còn dễ phát sinh sai sót. Do đó, nhu cầu ứng dụng các công nghệ mới như Trí tuệ nhân tạo (AI), Nhận dạng ký tự quang học (OCR) và các mô hình ngôn ngữ lớn (LLM) trong xử lý văn bản là hết sức cần thiết. Đề tài được chọn nhằm giải quyết bài toán thực tiễn này, đồng thời đóng góp vào xu hướng hiện đại hóa công tác hành chính – văn thư.

1.2. Mục tiêu nghiên cứu

Mục tiêu của đề tài là xây dựng một hệ thống hỗ trợ quản lý, phân loại và xử lý văn bản hành chính dựa trên công nghệ AI. Cụ thể:

- Ứng dụng OCR để số hóa văn bản giấy;
- Xây dựng cơ chế phân tích và trích xuất thông tin quan trọng như tiêu đề, nơi nhận, hành động;
- Triển khai API và giao diện người dùng thân thiện;
- Đảm bảo khả năng tích hợp với hệ thống hiện hữu nhằm hỗ trợ công việc hành chính hiệu quả và chính xác.

1.3. Phạm vi nghiên cứu

Phạm vi nghiên cứu tập trung vào các loại văn bản hành chính phổ biến, có cấu trúc rõ ràng, bao gồm công văn, quyết định, thông báo, tờ trình... Hệ thống được thiết kế chủ yếu để hỗ trợ công tác quản lý văn bản trong phạm vi nội bộ một trường đại học hoặc cơ quan hành chính. Nghiên cứu không đi sâu vào tất cả loại tài liệu phi cấu trúc hoặc các ngôn ngữ khác ngoài tiếng Việt.

1.4. Phương pháp nghiên cứu

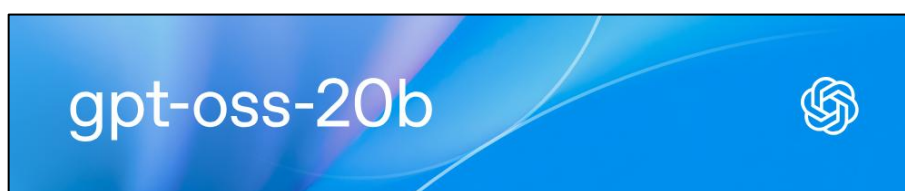
Đề tài áp dụng phương pháp nghiên cứu kết hợp giữa lý thuyết và thực nghiệm. Về lý thuyết, tiến hành khảo cứu các tài liệu liên quan đến xử lý ngôn ngữ tự nhiên, OCR và các mô hình LLM. Về thực nghiệm, xây dựng cơ sở dữ liệu mẫu, áp dụng quy trình OCR – RAG – LLM để phân tích văn bản, và triển khai thử nghiệm trên hệ thống thực tế. Song song đó, áp dụng phương pháp đánh giá định tính và định lượng để kiểm tra hiệu quả của hệ thống.

CHƯƠNG 2. CƠ SỞ LÝ THUYẾT

2.1. Tổng quan về mô hình ngôn ngữ Generative Pre-trained Transformer – Open Source Series (GPT-OSS)

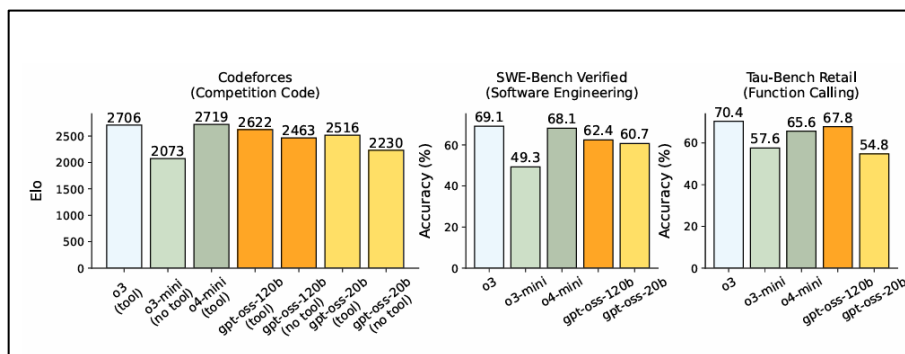
2.1.1. Giới thiệu về mô hình ngôn ngữ GPT-OSS

GPT-OSS (viết tắt của GPT Open-Source Series) là dòng mô hình ngôn ngữ lớn do OpenAI phát hành với trọng số mở, cho phép cộng đồng tải về và chạy mô hình một cách tự do trên hạ tầng của mình. Ra mắt ngày 5/8/2025, GPT-OSS gồm hai phiên bản: gpt-oss-120b (117 tỷ tham số) và gpt-oss-20b (21 tỷ tham số). Cả hai mô hình đều được cung cấp theo giấy phép Apache 2.0, đánh dấu lần đầu tiên OpenAI công bố mô hình ngôn ngữ lớn với trọng số mở kể từ GPT-2. [1]



Hình 2. 1. Minh họa GPT-OSS-20b (OpenAI, 2025). [1]

Mục tiêu của GPT-OSS là đẩy mạnh khả năng suy luận và tính minh bạch trong hệ sinh thái AI mã nguồn mở. Các mô hình này đạt hiệu năng thực tế ấn tượng với chi phí thấp, vượt trội các mô hình mã nguồn mở cùng kích cỡ trên các tác vụ suy luận. Đặc biệt, gpt-oss-120b đạt gần bằng hiệu năng với phiên bản thu nhỏ của o4-mini (OpenAI o4-mini) trên các bộ đánh giá về suy luận, trong khi có thể chạy hiệu quả chỉ trên một GPU 80GB. Phiên bản gpt-oss-20b thì đạt kết quả tương đương với mô hình o3-mini trên các benchmark phổ biến, và có thể vận hành trên các thiết bị có bộ nhớ 16GB VRAM GPU phổ thông. Nhờ đó, GPT-OSS mở ra khả năng triển khai mô hình ngôn ngữ mạnh mẽ ngay cả trên hạ tầng hạn chế, đáp ứng nhu cầu về riêng tư, chi phí thấp và tùy biến mà nhiều cá nhân, tổ chức mong đợi.



Hình 2. 2. Minh họa biểu đồ so sánh hiệu năng GPT-OSS với các mô hình khác [1]

Về mặt tính năng, GPT-OSS được thiết kế có khả năng sử dụng công cụ và thực hiện suy luận tốt. Mô hình đã được huấn luyện kết hợp kỹ thuật học tăng cường từ phản hồi (reinforcement learning) cùng các phương pháp tiên tiến tương tự những mô hình nội bộ mạnh nhất của OpenAI. Nhờ vậy, GPT-OSS không chỉ trả lời tốt các câu hỏi thông thường mà còn thực hiện được các tác vụ phức tạp có nhiều bước, biết gọi hàm (function calling) hoặc dùng công cụ tìm kiếm, và điều chỉnh mức độ suy luận tùy bài toán yêu cầu. Hai mô hình cũng cho phép trích xuất toàn bộ chuỗi suy luận (chain-of-thought) của chúng – một đặc điểm hiếm thấy ở các mô hình thương mại đóng – giúp người dùng có thể theo dõi hoặc phân tích được quá trình lập luận phía sau đáp án.

Tóm lại, GPT-OSS đại diện cho bước tiến quan trọng hướng tới AI mở: cung cấp sức mạnh mô hình ngôn ngữ hàng đầu trong một định dạng mở và minh bạch. Điều này cho phép các nhà phát triển, doanh nghiệp và cộng đồng dễ dàng tùy chỉnh và triển khai AI trên hạ tầng riêng, đồng thời thúc đẩy nghiên cứu và đổi mới một cách dân chủ hơn trong lĩnh vực xử lý ngôn ngữ tự nhiên (NLP).

2.1.2. Kiến trúc và nguyên lý hoạt động

Về kiến trúc, các mô hình GPT-OSS kế thừa nền tảng Transformer của GPT-2/GPT-3 nhưng được mở rộng bằng kỹ thuật Mixture-of-Experts (MoE) – hỗn hợp chuyên gia. Cụ thể, thay vì kích hoạt toàn bộ mạng neural cho mỗi token đầu vào, GPT-OSS chỉ kích hoạt một phần mạng (một tập chuyên gia) để xử lý, nhờ đó giảm số lượng tham số hoạt động tại mỗi bước suy luận.

Phiên bản gpt-oss-120b có tổng 117 tỷ tham số nhưng chỉ ~5,1 tỷ tham số kích hoạt trên mỗi token, với 36 tầng transformer và mỗi tầng bao gồm 128 chuyên gia nhưng chỉ 4 chuyên gia được chọn kích hoạt cho mỗi token. [2]

Tương tự, gpt-oss-20b có 21 tỷ tham số tổng cộng, ~3,6 tỷ tham số hoạt động mỗi token, với 24 tầng và 32 chuyên gia mỗi tầng (kích hoạt 4). Cơ chế MoE này cho phép mở rộng kích thước mô hình (về số tham số tổng) mà không làm tăng tương ứng chi phí tính toán cho mỗi bước suy luận, nhờ đó mô hình lớn có thể chạy hiệu quả hơn trên phần cứng giới hạn. [2]

Bên cạnh MoE, GPT-OSS còn tích hợp một số cải tiến về kiến trúc để tối ưu cho suy luận dài và hiệu năng cao. Mô hình sử dụng cấu trúc attention phân tán kiểu “dense-sparse” xen kẽ – tức là đan xen các lớp self-attention truyền thống với các lớp attention

cục bộ dạng băng (locally banded sparse attention), tương tự kỹ thuật trong GPT-3, giúp mô hình xử lý ngữ cảnh dài hiệu quả hơn.

Thêm vào đó, GPT-OSS áp dụng multi-query attention với nhóm khóa/giá trị chung (group size = 8) – một biến thể tối ưu hóa giúp giảm tải bộ nhớ và tăng tốc độ suy luận bằng cách cho nhiều head attention dùng chung tập khóa/trị. Về mã hóa vị trí, mô hình sử dụng RoPE (Rotary Positional Embedding), một kỹ thuật nhúng vị trí xoay chiều, cho phép hỗ trợ ngữ cảnh rất dài mà không bị suy giảm hiệu quả như vị trí tuyệt đối cố định. Nhờ những kỹ thuật trên, GPT-OSS có ngữ cảnh tối đa lên đến 128k token – vượt xa độ dài ngữ cảnh của các mô hình trước đó (GPT-4 thông thường là 8k-32k).

Quá trình hoạt động của GPT-OSS về cơ bản giống các GPT khác: mô hình dự đoán token tiếp theo dựa trên lịch sử chuỗi tokens trước đó (mô hình autoregressive). Tuy nhiên, do kiến trúc cải tiến, GPT-OSS có khả năng “nhìn lại” ngữ cảnh rất dài trong khi vẫn duy trì được tốc độ sinh đáp ứng nhanh. Chẳng hạn, với ngữ cảnh 128 nghìn token (tương đương hàng trăm trang văn bản), mô hình có thể ghi nhớ và tận dụng thông tin từ đầu văn bản khi tạo đầu ra ở cuối, điều mà các mô hình trước đây khó làm được nếu không có attention phân tán và RoPE.

Tóm lại, kiến trúc GPT-OSS được thiết kế để đạt hiệu năng suy luận cao và khả năng xử lý ngữ cảnh mở rộng, đồng thời tối ưu hóa tài nguyên tính toán. Thông qua MoE, mô hình tận dụng “chuyên gia” phù hợp cho từng đầu vào, tăng hiệu quả tính toán tương tự ý tưởng “chỉ dùng trí thông minh cần thiết cho mỗi nhiệm vụ”. Kết hợp với các kỹ thuật attention tiên tiến và cấu trúc transformer chuẩn, GPT-OSS có nền tảng kiến trúc vững chắc để thực hiện các tác vụ NLP phức tạp với độ chính xác và tốc độ cao.

2.1.3. Cơ chế huấn luyện và tối ưu

Quá trình huấn luyện GPT-OSS bao gồm hai giai đoạn chính: tiền huấn luyện (pre-training) trên corpora dữ liệu khổng lồ, sau đó là huấn luyện bổ sung (post-training) để dạy mô hình lập luận và sử dụng công cụ. [2]

Ở giai đoạn tiền huấn luyện, cả hai phiên bản GPT-OSS đều được huấn luyện theo phương pháp học tự giám sát truyền thống trên tập dữ liệu văn bản khổng lồ (trị số hàng nghìn tỷ token). Dữ liệu tập trung vào tiếng Anh, bao quát nhiều lĩnh vực với trọng tâm đặc biệt vào nội dung STEM, mã nguồn lập trình và kiến thức tổng hợp. Nhóm phát triển đã áp dụng các bộ lọc để loại trừ nội dung độc hại hoặc nhạy cảm (ví dụ: kiến thức về sinh học/công nghệ nguy hiểm) ngay từ khâu tiền huấn luyện, tái sử dụng bộ lọc

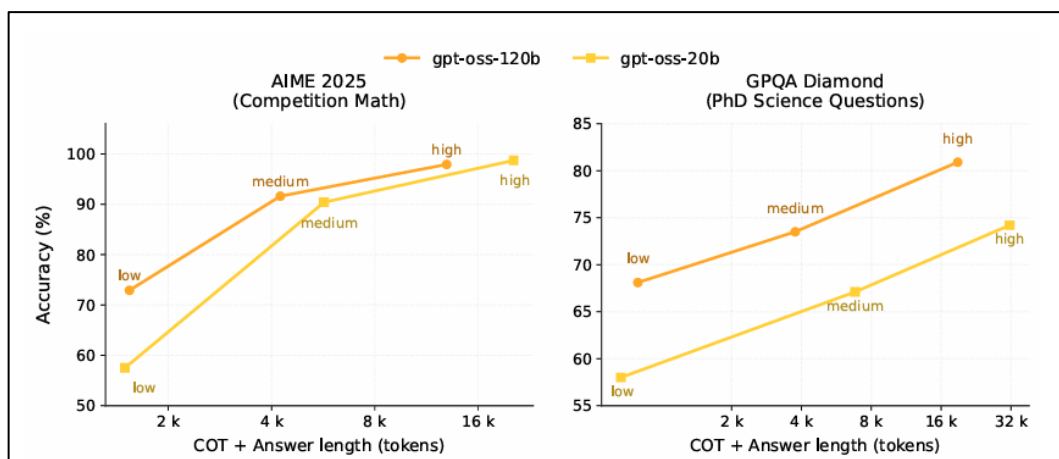
CBRN (hóa học, sinh học, phóng xạ, hạt nhân) từng dùng cho GPT-4. Nhờ đó, mô hình được hạn chế học những thông tin rủi ro ngay trong dữ liệu nền. Kết quả của giai đoạn này là GPT-OSS hình thành kiến thức nền tảng rất rộng (kiến thức cutoff của mô hình là tháng 6 năm 2024) và khả năng ngôn ngữ đa dụng tương tự các mô hình GPT khác. Việc huấn luyện được thực hiện trên hạ tầng GPU mạnh (ví dụ hàng nghìn GPU NVIDIA H100) với framework PyTorch và các tối ưu kernel tùy chỉnh (Triton) để tăng tốc. Mặc dù chi phí tính toán rất lớn (riêng bản 120B tham số tiêu thụ hàng triệu GPU-giờ huấn luyện), OpenAI đã tận dụng kỹ thuật phân tán và tối ưu hóa để thực hiện thành công việc huấn luyện mô hình khổng lồ này.

Tiếp theo, ở giai đoạn hậu huấn luyện (post-training), GPT-OSS trải qua các bước tinh chỉnh nhằm nâng cao khả năng lập luận có chuỗi và tương tác công cụ – những kỹ năng không tự nhiên có được từ tiền huấn luyện. Quá trình này gồm hai thành phần chính: (1) huấn luyện giám sát với dữ liệu đặc thù và (2) huấn luyện tăng cường với tín hiệu từ mô hình/hàm thưởng. Cụ thể, OpenAI đã sử dụng kỹ thuật tương tự như với các mô hình suy luận nội bộ hàng đầu (như GPT-4) – tức là dạy mô hình Chain-of-Thought (CoT) và cách giải quyết vấn đề qua nhiều bước bằng các lời giải mẫu và feedback. Mô hình được cung cấp nhiều bài toán phức tạp về lập trình, toán học, khoa học, v.v., cùng với lời giải từng bước do mô hình tiên tiến hoặc con người viết, để học cách tư duy phân tích thay vì chỉ trả lời một bước. Đồng thời, nhóm phát triển tích hợp cơ chế học tăng cường có định hướng: mô hình sau khi tạo lời giải sẽ được đánh giá và điều chỉnh thông qua một hàm thưởng nhằm khuyến khích các đáp án vừa chính xác logic vừa tuân thủ hướng dẫn (giống RLHF – Reinforcement Learning from Human Feedback).

Đáng chú ý, GPT-OSS còn được huấn luyện đặc biệt về sử dụng công cụ (tool use) trong giai đoạn này. Mô hình học cách gọi các “hàm công cụ” nhất định để hỗ trợ trả lời: chẳng hạn khi gặp câu hỏi cần tra cứu, nó có thể gọi công cụ duyệt web; khi cần tính toán, nó có thể gọi máy tính; hoặc chạy code Python nếu cần. Để làm được điều này, nhóm đã định nghĩa một định dạng hội thoại/hàm chuẩn (Harmony format – xem chi tiết ở mục 2.6) và cung cấp các kịch bản mẫu: ví dụ, với câu hỏi kiến thức thời sự, mô hình được khuyến khích xuất ra hành động “Tìm kiếm web” thay vì đoán mò. Sau khi mô hình thực hiện hành động và nhận kết quả, nó lại được huấn luyện để tổng hợp kết quả đó vào câu trả lời cuối. Bằng cách huấn luyện qua nhiều vòng như vậy (có sử

dụng cả thuật toán phản hồi phê bình từ mô hình), GPT-OSS dần thành thạo việc kết hợp công cụ ngoại vi vào quá trình suy luận của mình.

Cuối cùng, để đảm bảo an toàn và độ tin cậy, OpenAI áp dụng thêm huấn luyện đối kháng và điều chỉnh lập trường cho GPT-OSS. Mô hình được kiểm tra trước các tình huống đầu vào nhạy cảm hoặc tấn công “jailbreak” và học cách từ chối hoặc trả lời an toàn. Dù vậy, do trọng số mở, nhóm cũng lưu ý rằng tác nhân xấu có thể fine-tune lại GPT-OSS để loại bỏ các rào chắn này, nên việc giám sát sử dụng là cần thiết.



Hình 2. 3. Minh họa mức ảnh hưởng của độ dài chuỗi suy luận GPT-OSS [2]

Tóm lại, quy trình huấn luyện GPT-OSS kết hợp khối lượng tiền huấn luyện khổng lồ để có kiến thức nền, với hậu huấn luyện tinh vi để mô hình có kỹ năng suy luận đa bước và sử dụng công cụ như một trợ lý AI thực thụ. Các tối ưu về dữ liệu và chiến lược huấn luyện (như dùng CoT, RLHF, định dạng Harmony) giúp GPT-OSS đạt chất lượng ngang ngửa những mô hình thương mại đóng trong nhiều tác vụ trọng yếu, đồng thời vẫn giữ được tính linh hoạt và mở của một mô hình nguồn mở.

2.1.4. Đặc điểm và ưu điểm nổi bật

GPT-OSS sở hữu nhiều đặc điểm ưu việt khiến nó nổi bật trong số các mô hình ngôn ngữ lớn hiện nay:

Hiệu năng suy luận hàng đầu: GPT-OSS-120b cho thấy khả năng suy luận gần đạt ngang tầm GPT-4 (phiên bản nhỏ) trên các benchmark cốt lõi. Theo báo cáo, phiên bản 120B đạt “gần ngang bằng o4-mini” (một biến thể thu nhỏ của GPT-4) trong các bài kiểm tra lý luận, đồng thời vượt trội các mô hình mã nguồn mở khác cùng kích thước. Phiên bản 20B tuy nhỏ hơn nhưng vẫn có hiệu năng tương đương hoặc hơn GPT-3.5 (o3-mini) trên nhiều thước đo chuẩn. Đặc biệt, GPT-OSS thể hiện xuất sắc ở các nhiệm vụ dùng công cụ, gọi hàm, suy luận đa bước (CoT): kết quả trên bộ đánh giá TauBench

(đánh giá khả năng tác vụ dạng agentic) và HealthBench (hỏi đáp y tế) cho thấy GPT-OSS-120b thậm chí vượt qua một số mô hình độc quyền như OpenAI o1 hay GPT-4o về độ chính xác.

Trọng số mở và khả năng tùy biến: Khác với các mô hình GPT-3/GPT-4 thương mại, GPT-OSS được OpenAI phát hành hoàn toàn mở. Điều này cho phép người dùng tự do chạy mô hình cục bộ, tùy chỉnh, hoặc fine-tune cho các nhiệm vụ chuyên biệt mà không bị ràng buộc bởi API trả phí. Các nhà phát triển có thể tích hợp GPT-OSS vào ứng dụng của riêng mình, tinh chỉnh mô hình với dữ liệu lĩnh vực hẹp, hoặc đóng gói triển khai nội bộ để đảm bảo an toàn dữ liệu. Theo OpenAI, việc cung cấp các mô hình mạnh dạng open-weight như GPT-OSS sẽ giúp hạ thấp rào cản tiếp cận AI cho nhiều tổ chức, thị trường mới nổi hay các nhóm nguồn lực hạn chế, vì họ không cần phụ thuộc vào dịch vụ độc quyền đắt đỏ.

Tối ưu cho triển khai thực tế: GPT-OSS được thiết kế có tính thực dụng cao – mô hình chạy hiệu quả trên phần cứng phổ thông. Bản 120B có thể chạy trên một máy có GPU ~80GB VRAM (ví dụ NVIDIA A100 80GB), thậm chí người dùng đam mê có thể chạy bằng cách ghép nhiều GPU gaming (4×24GB) hoặc chuyển bớt tham số vào RAM nhờ kỹ thuật offloading (dù tốc độ chậm hơn). Bản 20B thì “dễ thở” hơn, chạy tốt trên một GPU ~16GB (như RTX 3090). Ngoài ra, OpenAI cũng cung cấp sẵn các phiên bản nén/giảm độ chính xác (quantized) thông qua nền tảng như LM Studio để cộng đồng thuận tiện tải về và dùng. Khả năng triển khai linh hoạt này giúp GPT-OSS trở thành lựa chọn hấp dẫn cho cả môi trường máy chủ lẫn thiết bị biên.

Hỗ trợ ngữ cảnh rất dài (128k): Như đề cập ở phần kiến trúc, GPT-OSS có thể tiếp nhận ngữ cảnh lên tới 128 nghìn token (tương đương ~96 nghìn từ). Điều này mở ra những ứng dụng mới như xử lý toàn bộ một cuốn sách dày hoặc duy trì hội thoại liên tục cực dài mà không cần tóm tắt. Khả năng này vượt trội so với GPT-4 (tối đa 32k token ở phiên bản đặc biệt). Trong thực tế, mô hình có thể hấp thụ lượng thông tin lớn từ ngữ cảnh trước khi đưa ra câu trả lời, giảm nhu cầu phân chia đoạn hay rút gọn dữ liệu đầu vào.

Minh bạch trong suy luận: GPT-OSS cung cấp đầu ra dạng Harmony (xem mục 2.6) giúp tách bạch phần giải thích/luận nội bộ và phần trả lời cuối cùng. Mô hình sẵn sàng xuất chuỗi suy luận (analysis channel) của nó – tức những bước lập luận trung gian – trước khi đưa ra kết luận. Điều này hỗ trợ người dùng hiểu được “mô hình đang nghĩ

gì” và kiểm chứng logic của mô hình, tăng độ tin cậy trong các ứng dụng đòi hỏi tính giải thích. Đây là ưu điểm lớn so với các mô hình đóng thường coi chuỗi suy luận là ẩn.

Khả năng sử dụng công cụ và tương tác linh hoạt: GPT-OSS được huấn luyện để gọi các hàm công cụ (function calling) một cách chính xác khi cần. Nó có thể tự động quyết định khi nào nên tra cứu web, thực thi mã hay truy cập cơ sở dữ liệu trong quá trình trả lời. Kết hợp với độ dài ngữ cảnh lớn, mô hình có thể hoạt động như một tác nhân (agent) thực hiện nhiều bước liên tiếp: ví dụ, nhận câu hỏi, tra cứu thông tin qua trình duyệt tích hợp, sau đó tổng hợp trả lời. Điều này biến GPT-OSS thành nền tảng lý tưởng để phát triển các trợ lý đa năng, biết kết hợp nhiều công cụ khác nhau nhằm hoàn thành nhiệm vụ phức tạp.

Nhờ những đặc tính trên, GPT-OSS được đánh giá là đã thu hẹp khoảng cách giữa mô hình nguồn mở và các mô hình hàng đầu thương mại. Nó mang lại hiệu năng và tính năng tiên tiến nhưng trong một khuôn khổ mở, linh hoạt và thân thiện hơn với cộng đồng nghiên cứu cũng như nhà phát triển độc lập. GPT-OSS đại diện cho triết lý AI cởi mở: hiệu quả cao, minh bạch và thuộc về mọi người.

2.1.5. Hạn chế và thách thức

Mặc dù có nhiều ưu điểm, GPT-OSS cũng đối mặt với một số hạn chế và thách thức nhất định:

Đòi hỏi hạ tầng phần cứng mạnh: Dù đã tối ưu, phiên bản 120B của GPT-OSS vẫn cần GPU có VRAM rất lớn (ít nhất ~80GB) để chạy mượt mà. Việc chạy mô hình này ngoài thực tế không dễ đối với người dùng phổ thông thiếu phần cứng chuyên dụng. Các giải pháp như phân chia nhiều GPU hoặc chạy một phần trên RAM tuy khả thi nhưng phức tạp và tốc độ suy giảm đáng kể. Ngay cả phiên bản 20B cũng yêu cầu GPU 16GB để đạt hiệu năng tốt. Do đó, rào cản về hạ tầng vẫn là thách thức khi phổ biến rộng rãi GPT-OSS, đặc biệt ở các môi trường hạn chế tài nguyên.

Nguy cơ lạm dụng do trọng số mở: Việc công khai trọng số mô hình là con dao hai lưỡi. Mặc dù nó thúc đẩy nghiên cứu và ứng dụng tự do, song cũng đồng nghĩa bất kỳ ai (kể cả các nhóm có ý đồ xấu) đều có thể fine-tune lại GPT-OSS cho các mục đích không mong muốn. OpenAI đã huấn luyện các bộ lọc an toàn cho GPT-OSS, nhưng một tác nhân xấu có thể tinh chỉnh mô hình để gỡ bỏ các rào cản này và khai thác GPT-OSS cung cấp thông tin độc hại hoặc hướng dẫn vi phạm pháp luật. Khác với API đóng (OpenAI có thể kiểm soát nội dung trả về), phiên bản open-weight đặt trách nhiệm kiểm

soát sử dụng lên cộng đồng. Đây là thách thức về an toàn và đạo đức khi phát hành các mô hình mạnh dưới dạng mã nguồn mở.

Hạn chế về hiểu biết sau mốc huấn luyện: GPT-OSS có “kiến thức nền” dừng ở thời điểm dữ liệu huấn luyện (khoảng giữa năm 2024). Những sự kiện, sự thay đổi sau mốc đó mô hình không tự động cập nhật được. Mặc dù GPT-OSS có thể tra cứu thời gian thực qua công cụ (nếu được tích hợp), nhưng bản thân nó không có nhận thức về thông tin mới. Điều này giống hạn chế chung của các LLM, đòi hỏi phải cập nhật hoặc bổ sung qua cơ chế RAG (Retrieve-and-Generate) hoặc fine-tune bổ sung trên dữ liệu mới – những việc mà người dùng phải tự thực hiện.

Kiến trúc thuần văn bản – không hỗ trợ đa modal: GPT-OSS hiện tại chỉ xử lý đầu vào và đầu ra dạng văn bản (text-only). Nó không có khả năng trực tiếp nhận dạng hình ảnh, âm thanh hay video. Trong khi đó, mô hình GPT-4 thương mại có khả năng đa modal (hiểu hình ảnh). Điều này có nghĩa GPT-OSS không thể giải quyết những tác vụ yêu cầu trực quan như phân tích hình ảnh, video. Trong một số tình huống ứng dụng (như trợ lý phòng thí nghiệm khoa học), sự thiếu hiểu biết thị giác này là một hạn chế đáng kể.

Hiện tượng ảo giác và lỗi logic: Dù đã cải thiện qua huấn luyện chuỗi suy luận, GPT-OSS vẫn có thể gặp các lỗi đặc trưng của mô hình ngôn ngữ như hallucination (ảo giác thông tin sai) hoặc lập luận thiếu chặt chẽ. Mô hình đôi lúc có thể tạo ra câu trả lời nghe có vẻ hợp lý nhưng thực tế sai, hoặc diễn giải không đúng ngữ cảnh tài liệu truy xuất được. Đây là hệ quả chung khi mô hình dựa vào xác suất ngôn ngữ. GPT-OSS đã giảm thiểu phần nào nhờ có cơ chế tra cứu thông tin thật và trích nguồn, nhưng không loại trừ hoàn toàn khả năng tự “bịa” nếu dữ liệu truy xuất không đủ hoặc mâu thuẫn. Do đó, người dùng vẫn cần thận trọng kiểm chứng các đáp án quan trọng.

Khó khăn trong việc huấn luyện và nâng cấp liên tục: Mặc dù RAG giúp giảm tần suất phải huấn luyện lại, nhưng để tích hợp tri thức mới vào bản thân mô hình GPT-OSS vẫn đòi hỏi tài nguyên lớn. Môi trường triển khai GPT-OSS tự do có thể không có các bản cập nhật trọng số định kỳ như mô hình thương mại. Điều này có nghĩa qua thời gian mô hình có thể lạc hậu dần nếu người dùng không chủ động huấn luyện bổ sung hoặc tận dụng tốt cơ chế truy xuất thông tin ngoài.

Tổng quát, những hạn chế trên không làm lu mờ các ưu điểm của GPT-OSS, nhưng chúng nhấn mạnh rằng việc sử dụng mô hình mã nguồn mở mạnh mẽ cũng cần

đi kèm trách nhiệm và hiểu biết kỹ thuật. Cộng đồng AI cần hợp tác để chia sẻ phương pháp vận hành GPT-OSS an toàn, phát triển các bộ lọc và cập nhật kiến thức cho mô hình, cũng như đầu tư hạ tầng mở để nhiều người có thể truy cập sức mạnh của GPT-OSS một cách có kiểm soát.

2.1.6. So sánh với các mô hình thương mại khác

Sự ra đời của GPT-OSS làm dấy lên nhiều so sánh thú vị với các mô hình ngôn ngữ thương mại hàng đầu như GPT-4 của OpenAI, PaLM của Google hay Claude của Anthropic. Về tổng thể, GPT-OSS đại diện cho trường phái mã nguồn mở, còn các mô hình kia (GPT-4, PaLM 2, v.v.) là mã nguồn đóng thương mại. Dưới đây là một số điểm so sánh chính:

Hiệu năng và độ chính xác: Trên các benchmark chuẩn, GPT-OSS-120b đạt kết quả tiệm cận GPT-4 (bản nhỏ) ở nhiều hạng mục. Cụ thể, GPT-OSS-120b tương đương hoặc nhỉnh hơn mô hình OpenAI o4-mini (được coi là phiên bản rút gọn của GPT-4) trong các bài toán lập trình Codeforces, bài kiểm tra kiến thức tổng quát (MMLU) và công cụ (TauBench). Thậm chí, GPT-OSS còn vượt o4-mini trên bộ câu hỏi y tế HealthBench và cuộc thi toán AIME 2024-2025. Trong khi đó, GPT-4 “đầy đủ” vẫn nhỉnh hơn một chút ở khả năng hiểu ngôn ngữ phức tạp và sáng tạo – ví dụ GPT-4 nổi trội trong việc viết văn phong phong phú, sáng tạo hơn, còn GPT-OSS thiên về giải quyết vấn đề mang tính phân tích logic. GPT-4 cũng có khả năng đa phương tiện (hiểu ảnh) mà GPT-OSS không có. Tuy vậy, khoảng cách giữa GPT-OSS và GPT-4 không lớn ở các tác vụ thuần văn bản mang tính học thuật, điều vốn rất đáng nể khi GPT-OSS là mô hình mở.

Khả năng đa nhiệm và công cụ: GPT-4 nổi tiếng nhờ sự đa năng – từ viết code, soạn văn bản, trả lời kiến thức, cho đến hiểu hình ảnh. GPT-OSS cũng tỏ ra đa năng ở mảng văn bản và lập trình, nhưng chưa thể xử lý ảnh. Về sử dụng công cụ, GPT-4 (qua ChatGPT Plugins hay chức năng function calling) đã cho thấy khả năng tích hợp API hiệu quả, nhưng GPT-OSS cũng không kém cạnh: mô hình được huấn luyện đặc biệt để gọi hàm và duyệt web, cho thấy hiệu quả tương tự các hệ thống plugin đóng. Một khác biệt là GPT-OSS công khai định dạng Harmony, giúp lập trình viên tự do tích hợp thêm bất kỳ công cụ nào tương thích, trong khi GPT-4 qua API đòi hỏi dùng định dạng function calling do OpenAI quy định.

Minh bạch và kiểm chứng: GPT-OSS vượt trội về tính minh bạch so với mô hình thương mại. Trọng số mở cho phép cộng đồng kiểm tra trực tiếp xem mô hình học gì, phản hồi ra sao, thậm chí tự chạy thử các trường hợp nhạy cảm. Ngược lại, GPT-4 là hộp đen – người dùng chỉ có thể thông qua API, không biết được quá trình nội tại. Hơn nữa, GPT-OSS xuất chuỗi suy luận nội bộ, còn ChatGPT/GPT-4 thì không (trừ khi dùng bản debug đặc biệt). Điều này mang lại sự tin tưởng hơn phần nào khi dùng GPT-OSS trong các ứng dụng yêu cầu giải trình kết quả.

Hiệu năng trên các lĩnh vực chuyên biệt: Một số lĩnh vực GPT-4 có lợi thế như sáng tạo ngôn ngữ tự nhiên, làm thơ, viết văn phong nghệ thuật – nhờ huấn luyện tinh chỉnh nhiều qua phản hồi nhân loại. GPT-OSS có thể yếu hơn ở các mặt này, do OpenAI tập trung tối ưu GPT-OSS cho suy luận và công cụ nhiều hơn là sáng tạo văn chương. Ngược lại, GPT-OSS có vẻ mạnh ở các bài toán kỹ thuật: kết quả vượt trội ở toán và lập trình thi đấu cho thấy mô hình này rất giỏi suy luận logic và tính toán. Tùy mục đích sử dụng, có thể chọn GPT-OSS hay GPT-4 tương ứng: chẳng hạn, để viết một truyện ngắn sáng tạo có thể GPT-4 sẽ cho kết quả sinh động hơn; nhưng để giải chi tiết một bài toán toán học hóc búa, GPT-OSS có thể đưa ra lập luận mạch lạc không kém.

Chi phí và kiểm soát: GPT-4 và các dịch vụ tương tự thường yêu cầu trả phí theo số token hoặc thuê bao. Ngược lại, GPT-OSS sau khi tải về có thể chạy miễn phí (không tính chi phí phần cứng/điện), không giới hạn số lượt gọi. Điều này rất hấp dẫn cho doanh nghiệp muốn tránh chi phí API liên tục. Hơn nữa, GPT-OSS chạy cục bộ đảm bảo kiểm soát dữ liệu – đầu vào không rời khỏi hệ thống nội bộ, phù hợp các ứng dụng đòi hỏi riêng tư cao (ví dụ phân tích dữ liệu nội bộ doanh nghiệp). Đây là điểm hơn hẳn so với việc dùng GPT-4 qua API khi mà dữ liệu gửi lên OpenAI có thể tiềm ẩn rủi ro bảo mật.

2.1.7. Ứng dụng của GPT-OSS trong xử lý ngôn ngữ tự nhiên

Với sức mạnh và tính mở của mình, GPT-OSS mở ra nhiều ứng dụng trong lĩnh vực xử lý ngôn ngữ tự nhiên (NLP) cũng như các ngành liên quan. Một số hướng ứng dụng tiêu biểu của GPT-OSS bao gồm:

Trợ lý ảo và Chatbot thông minh: GPT-OSS có thể được dùng làm nền tảng cho các trợ lý hội thoại tương tự ChatGPT nhưng triển khai nội bộ. Nhờ khả năng hiểu ngữ cảnh dài và suy luận đa bước, trợ lý dùng GPT-OSS có thể duy trì các cuộc trò chuyện phức tạp với người dùng, trả lời câu hỏi, thực hiện mệnh lệnh (như đặt lịch, gửi

email) thông qua tích hợp công cụ. Các doanh nghiệp có thể tận dụng GPT-OSS để xây dựng chatbot chăm sóc khách hàng trên website mà không lo dữ liệu khách hàng bị gửi ra bên ngoài, do mọi xử lý nằm trên máy chủ của họ.

Hệ thống hỏi đáp kiến thức nội bộ (Enterprise Q&A): Với cơ chế RAG (truy xuất và phản hồi) tích hợp, GPT-OSS rất thích hợp để làm trợ lý tra cứu tài liệu nội bộ. Ví dụ, một công ty có thể kết hợp GPT-OSS với cơ sở tri thức nội bộ: khi nhân viên hỏi về chính sách hoặc tài liệu thiết kế, mô hình sẽ tìm đoạn liên quan trong tập tài liệu rồi tổng hợp câu trả lời. Điều này tương tự cách các sản phẩm như Bing Chat hoạt động, nhưng GPT-OSS cho phép triển khai hoàn toàn nội bộ, đảm bảo bí mật kinh doanh. Các tổ chức như chính phủ, y tế cũng có thể dùng GPT-OSS để tạo trợ lý vấn đáp trên kho dữ liệu lớn mà vẫn giữ dữ liệu tại chỗ.

Phân tích và tóm tắt văn bản dài: Nhờ ngữ cảnh 128k, GPT-OSS có thể xử lý văn bản rất dài liên mạch. Do đó, ứng dụng trong tóm tắt tài liệu là cực kỳ hứa hẹn – mô hình có thể đọc hiểu và tóm tắt một báo cáo dài hàng trăm trang hoặc một chuỗi email dài trong công ty. Tương tự, GPT-OSS có thể dùng để phân tích văn bản pháp lý (hợp đồng, luật) hoặc tài liệu nghiên cứu khoa học dài dòng: mô hình đọc toàn bộ văn bản gốc rồi đưa ra tóm lược ý chính, thậm chí trả lời các câu hỏi chi tiết từ văn bản. Điều này giảm rất nhiều công sức so với việc phải thủ công chia nhỏ tài liệu cho các mô hình ngữ cảnh ngắn.

Trợ giúp lập trình và phân tích code: GPT-OSS, đặc biệt bản 120B, có hiệu năng rất cao ở các bài lập trình thi đấu và gỡ lỗi mã. Vì vậy, nó có thể được dùng làm trợ lý lập trình (giống GitHub Copilot hoặc ChatGPT) nhưng chạy cục bộ. Lập trình viên có thể sử dụng GPT-OSS để hỗ trợ viết code, giải thích code phức tạp, hoặc tìm lỗi trong codebase lớn. Với ngữ cảnh dài, mô hình có thể nạp vào nguyên cả một dự án nhiều file và trả lời câu hỏi về sự tương tác giữa các thành phần. Đặc biệt, khả năng thực thi mã Python nội tuyến (nhờ tích hợp tool “python”) cho phép GPT-OSS tự chạy thử một đoạn code và dựa vào kết quả để hiệu chỉnh câu trả lời – rất hữu ích khi debug.

Xây dựng trợ lý tác vụ phức hợp (AI Agents): GPT-OSS là ứng viên lý tưởng để phát triển các agent AI tự động hoàn thành mục tiêu đa bước. Như đã đề cập, mô hình có thể lập kế hoạch và thực hiện chuỗi hành động: tìm kiếm thông tin -> phân tích -> gọi API khác -> tổng hợp kết quả. Do đó, các nhà phát triển có thể tạo ra những agent đảm nhiệm công việc cụ thể, ví dụ agent phân tích thị trường tài chính (tự thu thập dữ

liệu giá, chạy mô hình dự báo, rồi báo cáo), hoặc agent quản lý hệ thống IT (tự đọc log, chẩn đoán sự cố, đề xuất cách khắc phục). Tất cả đều có thể chạy trên GPT-OSS mà không phụ thuộc dịch vụ bên ngoài. Một số dự án mã nguồn mở đã bắt đầu tích hợp GPT-OSS vào khung agent như Auto-GPT, cho thấy kết quả hứa hẹn.

Ứng dụng trong nghiên cứu khoa học và giáo dục: Cộng đồng học thuật có thể tận dụng GPT-OSS để nghiên cứu thêm về cơ chế hoạt động của mô hình ngôn ngữ (vì trọng số mở nên có thể phân tích sâu), hoặc dùng nó làm công cụ trợ lý trong thí nghiệm. Ví dụ, trong hóa học/sinh học, dù GPT-OSS không biết trực tiếp làm thí nghiệm, nhưng có thể hỗ trợ thiết kế thí nghiệm, phân tích tài liệu khoa học, hay thậm chí đóng vai “người gia sư AI” giải thích kiến thức cho sinh viên. Các trường đại học có thể triển khai GPT-OSS trên máy chủ riêng để sinh viên hỏi đáp kiến thức hoặc hỗ trợ học tập mà không lo bị lệ thuộc vào một công ty bên ngoài.

Fine-tune tạo mô hình chuyên biệt: Do GPT-OSS cho phép fine-tune, cộng đồng có thể huấn luyện lại nó thành các mô hình chuyên biệt. Chẳng hạn, fine-tune GPT-OSS để làm thơ, để viết code một ngôn ngữ lập trình ít phổ biến, hoặc để giao tiếp theo phong cách một nhân vật nào đó. Điều này giống như cách người ta fine-tune LLaMA trước đây để tạo ra nhiều biến thể (Alpaca, Vicuna, etc.). Với nền tảng 120B mạnh mẽ, chỉ cần một lượng nhỏ dữ liệu tinh chỉnh, GPT-OSS có thể biến thành mô hình xuất sắc trong nhiệm vụ hẹp nào đó. Đây là cơ hội lớn cho sáng tạo cộng đồng – vốn bị hạn chế với các mô hình đóng.

Nhìn chung, GPT-OSS có thể đảm nhận hầu hết các tác vụ NLP mà các mô hình hàng đầu như GPT-4 có thể làm, từ dịch thuật, trả lời câu hỏi, sinh văn bản, đến phân loại, thực thể hóa... Điểm khác biệt chính là GPT-OSS cho phép các ứng dụng đó được thực hiện tại chỗ, tùy biến và mở rộng bởi chính người dùng. Từ các doanh nghiệp lớn cần giải pháp AI nội bộ, đến các lập trình viên cá nhân muốn một ChatGPT riêng tư trên máy tính, GPT-OSS đều có thể đáp ứng. Trong tương lai, với sự đóng góp của cộng đồng (chia sẻ dataset tinh chỉnh, cải thiện an toàn), hệ sinh thái ứng dụng quanh GPT-OSS sẽ còn phát triển mạnh mẽ, đóng vai trò như một nền tảng AI mở rộng lớn mà trên đó nhiều dịch vụ thông minh có thể được xây dựng, tương tự vai trò của Linux trong hệ sinh thái hệ điều hành.

2.2. Tổng quan về Retrieve–Augment–Generate (RAG)

2.2.1. Nguyên lý hoạt động của RAG

Retrieve–Augment–Generate (RAG) là một phương pháp kết hợp giữa truy xuất thông tin và mô hình sinh ngôn ngữ nhằm nâng cao chất lượng và độ tin cậy của các hệ thống AI sinh văn bản. Đúng như tên gọi, quy trình RAG gồm ba bước chính:

Truy xuất (Retrieve) -> Tăng cường (Augment) -> Sinh (Generate). [3]

Ở bước Truy xuất, hệ thống sẽ dùng truy vấn của người dùng để tìm kiếm các thông tin liên quan từ một nguồn kiến thức bên ngoài. Nguồn này có thể là cơ sở dữ liệu văn bản, tài liệu, website, hoặc bất kỳ kho dữ liệu nào được lập chỉ mục. Ví dụ, với câu hỏi của người dùng, hệ thống RAG có thể thực hiện tìm kiếm trong tập tài liệu nội bộ hoặc trên Wikipedia để tìm ra vài đoạn văn bản có nội dung liên quan nhất đến câu hỏi. Việc truy xuất thường dùng các kỹ thuật tìm kiếm thông tin truyền thống (tf-idf, BM25) hoặc tìm kiếm qua vector nhúng semantic (sử dụng embedding của câu hỏi và văn bản) để tìm đoạn phù hợp.

Tiếp theo, bước Tăng cường, các thông tin tìm được sẽ được kết hợp (augment) vào ngữ cảnh cung cấp cho mô hình ngôn ngữ. Hiểu đơn giản, hệ thống sẽ chèn các đoạn tư liệu truy xuất vào prompt đầu vào của mô hình dưới dạng “ngữ cảnh hỗ trợ” cho câu hỏi của người dùng. Quá trình này giúp “mở rộng bộ nhớ” của mô hình: thay vì chỉ dựa vào những gì đã học trong tham số (có thể đã cũ hoặc không đủ chi tiết), mô hình nay có thêm kiến thức cụ thể, cập nhật từ bên ngoài làm cơ sở trả lời. Bước augment cũng bao gồm xử lý định dạng: ví dụ có thể phân tách rõ phần thông tin dẫn chứng và phần câu hỏi, hoặc chuyển đổi thông tin truy xuất thành định dạng mà mô hình dễ tiêu hóa (tóm tắt, làm nổi bật ý chính).

Cuối cùng, bước Sinh, mô hình ngôn ngữ lớn (LLM) sẽ tạo ra câu phản hồi cuối cùng dựa trên cả câu hỏi gốc lẫn nguồn thông tin đã được bổ sung ở bước trước. Mô hình cố gắng đưa ra câu trả lời chính xác và có dẫn chứng từ những tài liệu vừa truy xuất. Lợi ích là đáp án sẽ được “nền tảng hóa” trên dữ liệu thực thay vì chỉ dựa hoàn toàn vào trí nhớ của mô hình. Trong nhiều hệ thống RAG, câu trả lời của LLM còn đi kèm trích dẫn nguồn (ví dụ liệt kê link hoặc đoạn văn nào được dùng) để người dùng có thể kiểm chứng lại thông tin, tăng độ tin cậy. [3]

Nguyên lý cốt lõi của RAG là việc “mở sổ” tri thức cho mô hình. Thay vì để LLM hoạt động kiểu “sách đóng” (closed-book) – tức chỉ dựa vào tri thức đã huấn luyện

trong tham số, RAG biến nó thành “sách mở” (open-book) bằng cách cung cấp cho nó quyền tra cứu một bộ nhớ ngoài chứa kiến thức thực tế. Mô hình sau đó sẽ kết hợp cả kiến thức bên ngoài và sức mạnh học được để tạo câu trả lời. Cách tiếp cận này giúp giải quyết tình huống khi mô hình không có sẵn thông tin cần thiết hoặc kiến thức đã lỗi thời, thay vì đoán mò (dẫn đến ảo giác), mô hình chủ động tìm và dựa vào dữ liệu thực tế hơn.

2.2.2. Ưu điểm và hạn chế của RAG

Phương pháp RAG mang lại nhiều lợi ích quan trọng cho các hệ thống AI sinh ngôn ngữ:

Tăng tính chính xác và cập nhật: Ưu điểm lớn nhất của RAG là đảm bảo mô hình có thể truy cập thông tin mới nhất và chính xác thay vì chỉ dựa vào những gì đã học. Các LLM truyền thống thường có “kiến thức” dừng lại ở thời điểm huấn luyện và có thể chứa thông tin lỗi thời hoặc không đầy đủ. Nhờ RAG, mô hình có thể tra cứu dữ liệu thời gian thực hoặc cơ sở tri thức hiện hành, do đó trả lời đúng hơn cho những câu hỏi yêu cầu kiến thức cập nhật hoặc chi tiết. Theo một nghiên cứu, các câu trả lời được hỗ trợ bởi RAG tăng độ chính xác gần 43% so với mô hình chỉ dùng param nội tại của nó. Ví dụ, hỏi về sự kiện rất mới (mà mô hình chưa được huấn luyện), một hệ thống RAG có thể tìm tin tức về sự kiện đó rồi trả lời, trong khi mô hình thường sẽ bó tay hoặc đoán bừa.

Giảm ảo giác và tăng độ tin cậy: Bởi vì mô hình RAG phải dựa vào các đoạn tài liệu thực có liên quan, nó ít có cơ hội “hallucinate” (bịa thông tin) hơn. Khi trả lời, mô hình có xu hướng trích dẫn hoặc tổng hợp từ nguồn tìm được, do đó nội dung có thể kiểm chứng được. Hơn nữa, các hệ thống RAG thường kèm khả năng cung cấp nguồn dẫn chứng (ví dụ chú thích đoạn văn hoặc đường link tài liệu) cùng câu trả lời. Điều này giúp người dùng tin tưởng hơn và có thể tự kiểm tra lại đáp án, khắc phục điểm yếu “hộp đen” của LLM. Như IBM Research nhận định, RAG giúp người dùng “thấy được mô hình dựa trên nội dung gốc nào để trả lời” và giúp mô hình “ít cơ hội kéo ra thông tin sai từ tham số đã huấn luyện”, từ đó giảm nguy cơ rò rỉ dữ liệu nhạy cảm hoặc thông tin bịa đặt.

Hạn chế nhu cầu huấn luyện lại thường xuyên: Do có cơ chế bổ sung kiến thức động, RAG làm giảm đáng kể việc phải huấn luyện lại mô hình khi có thông tin mới. Trong môi trường doanh nghiệp hay tin tức luôn thay đổi, nếu dùng LLM thuần, ta

phải định kỳ cập nhật trọng số mô hình với dữ liệu mới – việc vừa tốn kém tính toán, vừa không kịp thời. RAG cho phép chỉ cần cập nhật cơ sở tri thức (thêm tài liệu mới) là mô hình có thể truy xuất và trả lời dựa trên tài liệu mới đó ngay, không cần điều chỉnh tham số mô hình. Điều này tiết kiệm cả chi phí lẫn thời gian, nhất là khi triển khai chatbot doanh nghiệp: chỉ cần duy trì dữ liệu nội bộ, chatbot luôn trả lời theo tài liệu mới nhất mà không cần huấn luyện AI liên tục.

Giữ bí mật và an toàn dữ liệu: RAG cho phép xây dựng các hệ thống hỏi đáp nội bộ mà không phải đưa dữ liệu nhạy cảm vào quá trình huấn luyện mô hình (vốn có thể đẩy dữ liệu lên đám mây nhà cung cấp). Thay vào đó, dữ liệu nhạy cảm (ví dụ văn bản tài liệu nội bộ) chỉ cần lưu trong cơ sở tri thức tại chỗ, và mô hình có thể truy vấn khi cần. Mô hình LLM không nhất thiết học thuộc dữ liệu đó, do đó giảm nguy cơ vô tình làm rò rỉ thông tin trong phản hồi. Việc “neo” mô hình vào các nguồn kiểm chứng cũng giúp ngăn mô hình nói năng lung tung không kiểm soát, hữu ích trong bối cảnh yêu cầu AI giải thích được và tuân thủ chính sách (như lĩnh vực y tế, tài chính nơi cần căn cứ rõ ràng).

Hiệu quả với kiến thức chuyên sâu, hẹp: RAG đặc biệt có ích khi cần trả lời các câu hỏi kiến thức chuyên ngành hoặc chi tiết mà mô hình thường không nhớ rõ. Thay vì phải nhồi nhét mọi kiến thức hẹp vào tham số mô hình (vừa tốn tài nguyên vừa có giới hạn), ta chỉ cần cung cấp nguồn thông tin chuyên ngành khi có câu hỏi tương ứng. Điều này cho phép một mô hình tổng quát (như GPT-OSS) có thể xử lý nhiều lĩnh vực khác nhau mà không phải fine-tune riêng từng lĩnh vực. Ví dụ, một chatbot hỗ trợ kỹ thuật có thể sử dụng RAG để truy xuất hướng dẫn từ tài liệu kỹ thuật, dù tài liệu này quá chi tiết khó có trong kiến thức huấn luyện của mô hình.

Hạn chế của RAG cũng cần được lưu ý:

Phụ thuộc vào chất lượng bộ truy xuất: RAG tốt hay không trước hết nằm ở bước Retrieve. Nếu hệ thống tìm sai hoặc thiếu thông tin liên quan, phần trả lời sẽ bị ảnh hưởng nặng. Một tìm kiếm không hiệu quả có thể trả về các đoạn văn nhiễu hoặc không chính xác, dẫn đến mô hình tổng hợp sai. Do đó, RAG đòi hỏi phải xây dựng bộ phận tìm kiếm/thăm dò thật tốt (ví dụ hệ thống vector search chính xác, có thể cần điều chỉnh tham số hoặc mô hình retriever riêng). Đây là một lớp phức tạp bổ sung so với việc dùng LLM thuần. Ngoài ra, dữ liệu trong kho kiến thức nếu không được tổ chức

hoặc làm sạch tốt cũng sẽ khiến RAG lấy phải thông tin kém chất lượng (garbage in, garbage out).

Xử lý ngữ cảnh dài và độ trễ: Khi đưa thêm tài liệu vào prompt, độ dài ngữ cảnh của mô hình tăng lên đáng kể. Mặc dù các mô hình hiện nay hỗ trợ ngữ cảnh dài hơn trước, việc nhồi quá nhiều tài liệu có thể vẫn dẫn đến vượt giới hạn token hoặc làm chậm tốc độ xử lý. Hơn nữa, việc truy xuất và chuẩn bị thông tin bên ngoài cũng làm tăng độ trễ hệ thống (do phải tìm kiếm trong database, đôi khi vài giây, trước khi mô hình trả lời). Nếu không tối ưu, RAG có thể tạo độ trễ đáng kể trong trải nghiệm người dùng so với việc mô hình trả lời trực tiếp từ param.

Khó khăn khi thông tin mâu thuẫn hoặc không đủ: RAG cũng không đảm bảo 100% giải quyết được mọi câu hỏi. Nếu cơ sở tri thức thiếu thông tin cho một truy vấn (hoặc thông tin có nhưng truy xuất không tìm ra), mô hình sẽ vẫn bị “mù” phần đó. Trong trường hợp xấu, mô hình có thể cố gắng “sáng tạo bù” dựa trên hiểu biết cũ của nó dẫn đến câu trả lời sai mà trông có vẻ được hỗ trợ bởi tài liệu. Hoặc khi các nguồn tìm được có nội dung mâu thuẫn nhau, mô hình có thể lúng túng không biết tin nguồn nào, dẫn đến câu trả lời lộn xộn hoặc chấp vá chi tiết từ nhiều nguồn một cách sai lạc. Ví dụ, hỏi về một sự kiện gây tranh cãi với nhiều báo đưa tin khác nhau, RAG có thể trả về vài bản tin trái ngược; mô hình nếu không đủ thông minh phân giải sẽ dễ trộn lẫn chi tiết cũ mới, gây hiểu lầm.

Tính phức tạp hệ thống tăng: Triển khai RAG yêu cầu tích hợp nhiều thành phần: cơ sở dữ liệu tài liệu (hoặc công cụ tìm kiếm), module truy xuất, module xử lý prompt, rồi LLM. Việc này phức tạp hơn đáng kể so với gọi một LLM duy nhất. Nó đòi hỏi kỹ sư phải xây dựng pipeline cẩn thận, từ bước tiền xử lý tài liệu (chia nhỏ, tạo index vector...), đến thiết kế chiến lược truy vấn (có thể phải thử nhiều truy vấn khác nhau, hay truy xuất đa vòng). Ngoài ra, để RAG hiệu quả, có thể cần tinh chỉnh mô hình ở mức độ nhất định để biết sử dụng tài liệu được chèn vào prompt đúng cách (ví dụ huấn luyện nhẹ mô hình theo định dạng question + [docs] + answer). Tất cả những điều này khiến phát triển hệ thống RAG phức tạp hơn so với dùng LLM thông thường.

Tóm lại, RAG là một hướng kết hợp rất hữu ích, nhưng để vận hành tốt cần dữ liệu tốt và kỹ thuật phù hợp. Khi được thiết kế đúng, ưu điểm của RAG – thông tin cập nhật, chính xác, có nguồn kiểm chứng – sẽ vượt trội so với các hạn chế. Trên thực tế, nhiều nghiên cứu và triển khai đã chứng minh giá trị của RAG trong việc nâng độ tin

cây của chatbot cũng như giảm chi phí huấn luyện. Trong các trường hợp ứng dụng đòi hỏi sự đúng đắn cao (như trợ lý luật sư, bác sĩ AI, trợ lý doanh nghiệp), lợi ích của RAG thường lớn hơn hẳn so với việc chấp nhận một mô hình “biết tuốt nhưng ảo giác”. Tuy nhiên, người phát triển vẫn phải đầu tư vào khâu quản lý dữ liệu tri thức và thuật toán truy xuất để khắc phục các hạn chế nêu trên.

2.2.3. Ứng dụng của RAG trong thực tiễn

Kỹ thuật RAG hiện đã và đang được ứng dụng rộng rãi trong nhiều sản phẩm và lĩnh vực, nơi việc kết hợp tri thức ngoại vi với sức mạnh mô hình ngôn ngữ mang lại hiệu quả vượt trội. Một số ứng dụng thực tiễn tiêu biểu:

Chatbot hỗ trợ khách hàng và trợ lý doanh nghiệp: Nhiều công ty triển khai chatbot nội bộ dùng RAG để trả lời câu hỏi của khách hàng hoặc nhân viên dựa trên kho tài liệu của công ty. Ví dụ, chatbot ngành ngân hàng có thể truy xuất chính sách, điều khoản dịch vụ, dữ liệu tài khoản... và dùng GPT (kết hợp RAG) để trả lời chính xác theo thông tin hiện hành (lãi suất mới, quy định cập nhật). Tương tự, trợ lý nội bộ “hỏi gì cũng biết” cho nhân viên sử dụng RAG để đọc tài liệu hướng dẫn, quy trình công ty nhằm giải đáp thắc mắc. Điều này giúp giảm tải bộ phận hỗ trợ và đảm bảo câu trả lời luôn bám sát tài liệu chính thức.

Công cụ tìm kiếm thông minh và hệ thống hỏi đáp mở: RAG là nền tảng của các công cụ tìm kiếm thế hệ mới như Bing Chat hay Perplexity.ai. Khi người dùng đặt câu hỏi, hệ thống tìm kết quả web liên quan (Retrieve) rồi tổng hợp thành câu trả lời tự nhiên (Generate), thường kèm trích dẫn nguồn. Điều này biến trải nghiệm tìm kiếm truyền thống (chỉ trả link) thành trải nghiệm hỏi đáp thông minh. Trong lĩnh vực hỏi đáp mở (open-domain QA), RAG đã trở thành phương pháp SOTA, vượt qua các mô hình QA thuần túy về độ chính xác trên các bộ dữ liệu như Natural Questions hay TriviaQA.

Tóm tắt và báo cáo từ dữ liệu động: Một công cụ có thể tự động tổng hợp bản tin hàng ngày: truy xuất tin tức mới (Retrieve), rồi GPT viết bản tóm tắt (Generate). Trong doanh nghiệp, RAG có thể tạo báo cáo tự động từ database và trình bày ngắn gọn, dễ hiểu. Nhờ bước truy xuất, báo cáo đảm bảo chính xác về dữ liệu nhưng vẫn tự nhiên hơn so với template cứng. Ví dụ khác: phân tích y văn – hệ thống truy xuất bài nghiên cứu y học liên quan rồi GPT tóm lược, giúp bác sĩ nhanh chóng nắm kiến thức mới.

Hệ thống trợ lý code và truy vấn cơ sở dữ liệu: Trợ lý lập trình dùng RAG để tìm kiếm trong documentation hoặc codebase, trả lời chính xác dựa trên phiên bản mới

nhất thay vì dữ liệu huấn luyện cũ. Khi debug, trợ lý có thể tìm code liên quan trong project để hiệu chỉnh. Ngoài ra, RAG còn hỗ trợ natural language to SQL: người dùng hỏi bằng ngôn ngữ tự nhiên, hệ thống truy vấn database và GPT diễn giải kết quả, giúp người không biết SQL vẫn khai thác dữ liệu.

Lĩnh vực y tế và pháp lý: Trong y tế, trợ lý bác sĩ dùng RAG để truy xuất hướng dẫn điều trị, nghiên cứu lâm sàng mới nhất khi tư vấn. Trong pháp lý, trợ lý tìm luật, án lệ và văn bản liên quan, rồi tóm tắt cho luật sư kèm trích dẫn. Những ứng dụng này giúp chuyên gia (bác sĩ, luật sư) tăng hiệu quả và giảm sai sót nhờ câu trả lời luôn có nguồn kiểm chứng.

Trợ lý học tập cá nhân hóa: Trong giáo dục, RAG hỗ trợ sinh viên hỏi đáp dựa trên giáo trình hoặc bài giảng thực tế. Trợ lý có thể trích ví dụ, bài tập mẫu rồi hướng dẫn chi tiết, giúp việc học được cá nhân hóa thay vì chỉ nhận câu trả lời chung chung.

Tóm lại, RAG đóng vai trò cầu nối giữa hệ thống thông tin truyền thống (cơ sở dữ liệu, website, tài liệu) và trí tuệ ngôn ngữ nhân tạo. Sự kết hợp này tạo ra các ứng dụng “thông minh có cơ sở”: vừa linh hoạt trong sinh ngôn ngữ, vừa chính xác, đúng ngữ cảnh. Trong bối cảnh hiện tại, gần như mọi lĩnh vực làm việc với văn bản đều có thể hưởng lợi từ RAG – từ kinh doanh, chăm sóc khách hàng, giáo dục, đến y tế và pháp lý. RAG ngày càng chứng tỏ là “mảnh ghép không thể thiếu” để đưa các mô hình ngôn ngữ lớn vào ứng dụng thực tế một cách hiệu quả và đáng tin cậy.

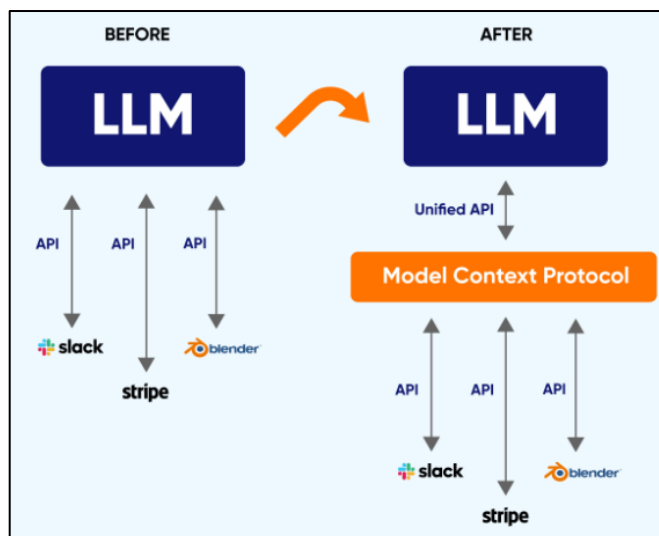
2.3. Tổng quan về Model Context Protocol (MCP)

2.3.1. Khái niệm về MCP

Model Context Protocol (MCP) là một chuẩn mở được giới thiệu nhằm chuẩn hóa cách các mô hình AI kết nối và tương tác với các nguồn dữ liệu và công cụ bên ngoài. MCP được phát triển bởi Anthropic (hãng tạo ra mô hình Claude) và công bố vào cuối năm 2024, với mục tiêu tạo ra một “giao thức chung” – tương tự như một “cổng USB-C cho AI” – giúp các ứng dụng AI có thể dễ dàng tiếp cận dữ liệu, dịch vụ ở nhiều hệ thống khác nhau theo một cách thức thống nhất. [4]

Khái niệm cốt lõi của MCP là xây dựng một “ngôn ngữ chung” để các mô hình ngôn ngữ (LLM) – trong vai trò client – có thể kết nối hai chiều an toàn với các MCP server, vốn đại diện cho những nguồn dữ liệu hoặc dịch vụ bên ngoài. MCP định nghĩa chuẩn giao tiếp (dựa trên JSON, HTTP, WebSocket, ...) cùng các thông điệp chuẩn, cho

phép mô hình AI gửi yêu cầu truy vấn dữ liệu đến dịch vụ, nhận lại kết quả, và thậm chí gửi cập nhật ngược trở lại nếu cần.



Hình 2. 4. Minh họa kiến trúc truyền thống và chuẩn hóa với MCP

Các nhà phát triển có thể triển khai một MCP server trước nguồn dữ liệu của họ (ví dụ: MCP server cho kho tài liệu, cho Slack, cho GitHub...). Khi đó, bất kỳ ứng dụng AI nào (Claude, GPT, hay model mã nguồn mở) hỗ trợ MCP client đều có thể kết nối đến server đó để lấy dữ liệu hoặc thực hiện hành động.

Nói cách khác, MCP tạo ra một tầng “kết dính chuẩn hóa” giữa AI assistants và hệ thống CNTT. Trước đây, để AI truy cập một cơ sở dữ liệu, lập trình viên phải tùy biến prompt hoặc viết code tích hợp riêng cho từng nguồn. Với mỗi dịch vụ mới (Google Drive, Jira, Outlook...), cần làm lại từ đầu. MCP thay thế mô hình tích hợp chắp vá đó bằng một chuẩn duy nhất: chỉ cần viết driver MCP cho dữ liệu X, mọi AI agent hỗ trợ MCP đều có thể sử dụng X ngay.

2.3.2. Cơ chế hoạt động của MCP

Cơ chế client–server của MCP: MCP hoạt động theo mô hình client–server quen thuộc. Ứng dụng AI (assistant) đóng vai trò client, còn các MCP server đại diện cho dịch vụ/công cụ bên ngoài (như Slack, Drive, Database...). Client biết server nào khả dụng thông qua file cấu hình mcp.json chứa URL và thông tin truy cập. Kết nối có thể qua HTTP(S) hoặc WebSocket, kèm xác thực (OAuth, token) nếu cần. Mỗi server cung cấp một endpoint giao tiếp chuẩn hóa để AI có thể gọi đến.

Quy trình Request–Response: Khi cần dữ liệu hay thực hiện hành động, AI gửi một request JSON tới server, gồm: server đích, loại action (search, get, execute...), tham số chi tiết và request_id. MCP server sau đó chuyển request thành thao tác thực sự trên

dịch vụ (gọi API Slack, query SQL, v.v.) rồi trả lại response JSON. Response chứa kết quả chính, metadata (trạng thái, nguồn, thời gian), và request_id để khớp lệnh. Nếu lỗi, server trả JSON báo lỗi rõ ràng. Điều này biến server MCP thành một adapter: dịch chuẩn MCP thành API cụ thể của từng dịch vụ.

Xử lý phía AI client: Client nhận response, parse JSON, và đưa thông tin đó cho mô hình sử dụng. Một số ứng dụng (như LM Studio) có thể chèn thẳng kết quả vào hội thoại với mô hình, hoặc để mô hình tự động phát động request tiếp theo dựa trên dữ liệu vừa nhận. Quá trình có thể lặp lại nhiều vòng, với nhiều server khác nhau. MCP cũng hỗ trợ quản lý context và phiên giao dịch, giúp AI duy trì mạch làm việc liên tục.

An toàn, mở rộng và đa nền tảng: Điểm mạnh của MCP là an toàn, server có thể giới hạn dữ liệu và hành động mà AI được phép thực hiện, thay vì để AI truy cập trực tiếp cơ sở dữ liệu thô. Nhờ cấu trúc chuẩn, việc mở rộng dễ dàng: muốn thêm nguồn mới chỉ cần thêm server, AI logic giữ nguyên. MCP có SDK nhiều ngôn ngữ (Python, Java, v.v.) nên developer dễ triển khai. Anthropic ví MCP như “hộp công cụ chuẩn hóa cho AI”: AI assistant không cần học cách dùng công cụ qua prompt tự do nữa, mà dùng API chung, có cấu trúc, thống nhất và an toàn.

2.3.3. Vai trò của MCP trong hệ sinh thái AI

Kết nối AI với hệ thống và dữ liệu liên mạch: Trước MCP, mỗi tích hợp AI với hệ thống ngoài đều cần giải pháp tùy chỉnh, gây phân mảnh. MCP xuất hiện như một chuẩn giao tiếp thống nhất, tương tự “USB-C cho AI”, giúp AI có thể truy cập mọi nguồn dữ liệu và công cụ hỗ trợ MCP một cách liên mạch. Điều này phá vỡ silo thông tin, mở ra khả năng AI vừa học từ dữ liệu huấn luyện, vừa kết nối được với thế giới thực.

Thúc đẩy tính mở và chuẩn chung: Không giống các hệ sinh thái đóng (Plugins của OpenAI, extensions của Google Bard hay Bing), MCP là chuẩn mở và trung lập. Ngay từ khi ra mắt, nhiều công ty và dự án mã nguồn mở đã tích hợp (Block, LM Studio, OpenRouter...). Nếu MCP thành công, nó sẽ đóng vai trò như HTML trong web – một chuẩn kỹ thuật chung giúp các AI agent tương tác với ứng dụng mà không bị khóa vào vendor nào.

Tăng khả năng AI assistants và giữ an toàn: Nhờ MCP, một AI assistant có thể làm nhiều việc hơn: lấy file từ Drive, tạo Docs, gửi Slack, query database... trở thành “siêu trợ lý” thực thụ. Điểm quan trọng là doanh nghiệp có thể triển khai MCP server nội bộ để kiểm soát và log mọi truy cập, bảo đảm an toàn và minh bạch. MCP tạo sự cân

bằng giữa hiệu quả sử dụng AI và tuân thủ bảo mật, giúp doanh nghiệp tin tưởng giao quyền truy cập hệ thống cho AI.

Giảm chi phí tích hợp và mở rộng: Thay vì xây tích hợp riêng lẻ cho từng hệ thống, với MCP chỉ cần viết một server cho mỗi nguồn dữ liệu, AI sẽ dùng ngay mà không cần thay đổi logic. Điều này tiết kiệm đáng kể chi phí và công sức mở rộng AI sang nhiều hệ thống. Tương lai có thể xuất hiện công cụ tự động sinh MCP server từ OpenAPI, giúp việc thêm “plugin” cho AI nhanh chóng và đơn giản.

Xây dựng hệ sinh thái dịch vụ quanh AI: MCP mở đường cho một “app store” mới – nơi nhà cung cấp tung ra MCP endpoints cho AI. Một dịch vụ thời tiết hay điều khiển nhà thông minh có thể viết MCP server và ngay lập tức trở thành công cụ mà nhiều AI khác nhau sử dụng được. Điều này tạo một thị trường plugin mở, khuyến khích đổi mới và gia tăng khả năng của AI assistants.

Chuẩn hóa bảo mật và quyền riêng tư: MCP được thiết kế có sẵn lớp bảo mật: mọi request–response đều có cấu trúc rõ ràng, dễ kiểm soát, log và audit. Doanh nghiệp có thể áp dụng policy, phân quyền, và đảm bảo tuân thủ quy định khi AI truy cập dữ liệu nhạy cảm. Đây là nền tảng để AI ứng dụng trong các ngành quan trọng như tài chính, y tế, luật pháp.

2.4. Tổng quan về Tool Calling

2.4.1. Khái niệm về Tool Calling

Tool Calling (hay Function Calling) là khả năng để LLM gọi và sử dụng các công cụ hoặc hàm bên ngoài nhằm mở rộng năng lực. Thay vì chỉ sinh văn bản từ kiến thức huấn luyện, mô hình có thể tra cứu thông tin, thực hiện phép tính, gọi API... rồi đưa kết quả đó vào câu trả lời. OpenAI từng triển khai function calling cho GPT-4 năm 2023 – đây chính là một cách triển khai cụ thể của tool calling. [5]

LLM vốn không có kiến thức thời gian thực, dễ sai trong tính toán, và không thể trực tiếp thao tác với hệ thống. Tool calling khắc phục nhược điểm này bằng cách để mô hình gọi web search, máy tính, database hoặc thậm chí các mô hình AI khác. Nhờ vậy, LLM không chỉ “trả lời bằng lời nói” mà có thể hành động để hoàn thành tác vụ, trở thành agentic AI – AI có khả năng tự lên kế hoạch, gọi công cụ phù hợp, rồi tiếp tục suy luận theo nhiệm vụ.

Quy trình thực hiện Tool Calling:

- Người dùng đưa yêu cầu.

- Mô hình nhận ra cần công cụ → sinh lệnh gọi tool (tên + tham số).
- Hệ thống thực thi lệnh (gọi API, chạy code, tìm kiếm).
- Kết quả trả về mô hình.

Các ứng dụng như AutoGPT, BabyAGI, Bing Chat hay ChatGPT Plugins đều dựa trên ý tưởng tool calling. Trước 2023, ý tưởng này đã có trong các nghiên cứu (WebGPT 2021, ReAct 2022), nhưng chỉ thực sự phổ biến khi OpenAI và các framework như LangChain quảng bá mạnh. Tool calling hiện được xem là nền tảng để AI trở thành trợ lý đa năng trong đời sống và công việc.

Tool calling là “chìa khóa” để biến LLM từ hệ thống chỉ xử lý ngôn ngữ thành một agent biết hành động. Nó vừa khắc phục hạn chế (ảo giác, thiếu cập nhật), vừa mở rộng phạm vi ứng dụng (quản lý dữ liệu, lập kế hoạch, thao tác hệ thống). Nói cách khác, tool calling chính là bước quan trọng để AI có thể tự động hóa quy trình phức tạp và gắn kết với thế giới thực.

2.4.2. Cơ chế hoạt động

Trước hết, hệ thống phải cho mô hình biết có những công cụ nào khả dụng và cách dùng chúng. Thông tin này được khai báo trong cấu hình hoặc prompt hệ thống, thường gồm tên tool, mô tả và tham số.

Phân tích yêu cầu & quyết định gọi tool: Khi người dùng đặt câu hỏi, mô hình suy luận xem có cần dùng công cụ hay không. Nếu câu hỏi vượt ngoài khả năng (như thông tin mới hoặc phép tính phức tạp), mô hình sẽ xuất ra lệnh gọi tool thay vì trả lời trực tiếp. Lệnh này có thể ở dạng JSON (function calling trong OpenAI API) hoặc dạng text hành động (trong ReAct, LangChain). [5]

Thực thi công cụ bên ngoài: Hệ thống runtime bên ngoài sẽ thực sự gọi tool tương ứng: chạy hàm Python, gửi request API, hoặc truy vấn database. Đây là phần nằm ngoài mô hình, do chương trình hoặc framework xử lý. Kết quả thu được sẽ được gom lại (ví dụ dữ liệu JSON, chuỗi text kết quả).

Truyền kết quả về cho mô hình: Kết quả từ tool được đưa ngược vào mô hình. Trong OpenAI API, kết quả được thêm như một message mới; trong LangChain, nó được chèn vào prompt dưới dạng “Observation”. Mô hình nhận biết đây là phản hồi từ tool mà nó đã gọi.

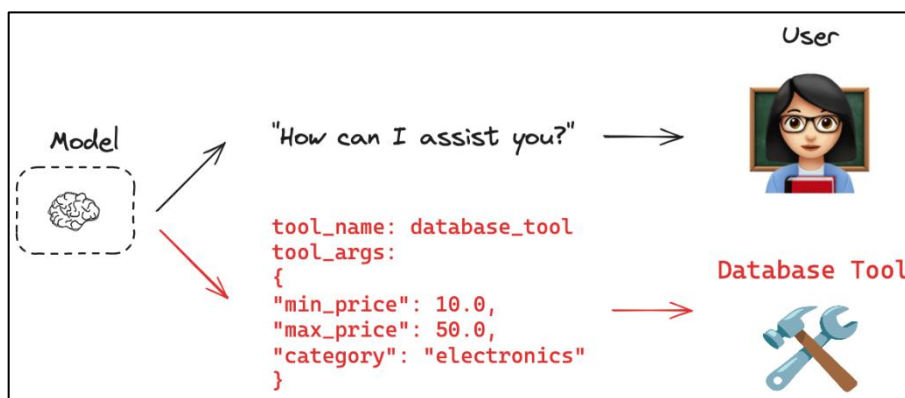
Sinh câu trả lời hoặc tiếp tục lặp: Dựa trên dữ liệu nhận được, mô hình có thể trả lời ngay cho người dùng. Hoặc nếu thấy chưa đủ, nó có thể quyết định gọi thêm một

tool khác, lặp lại vòng tròn request → execute → response. Chu trình này tiếp diễn cho đến khi mô hình đưa ra câu trả lời cuối cùng.

Cuối cùng, mô hình xuất câu trả lời chính thức (không còn action nữa). Người dùng chỉ thấy kết quả mượt mà, trong khi phía sau mô hình đã âm thầm thực hiện nhiều bước gọi công cụ.

2.4.3. So sánh Tool Calling và Function Calling

Tool Calling là khái niệm rộng, chỉ việc mô hình ngôn ngữ gọi công cụ hoặc hành động bên ngoài để mở rộng khả năng. Function Calling là một trường hợp cụ thể của Tool Calling, thường gắn với OpenAI API: mô hình trả về lời gọi hàm theo format JSON, hệ thống parse và thực thi. Vì vậy, mọi function calling đều là tool calling, nhưng tool calling có thể linh hoạt hơn và không nhất thiết phải theo schema chặt chẽ.



Hình 2. 5. Minh họa cơ chế Tool Calling với tham số JSON

Function calling trong OpenAI API cho phép developer định nghĩa hàm và schema JSON, đảm bảo an toàn và dễ kiểm soát. Ngược lại, tool calling kiểu agent (LangChain, ReAct) để mô hình sinh chuỗi hành động tự do (“Action: search(...”), linh hoạt hơn nhưng dễ lỗi định dạng. Về thuật ngữ, OpenAI phổ biến khái niệm “function calling”, trong khi các framework agent thường dùng “tool/agent”. Nhiều tài liệu kỹ thuật (IBM, OpenRouter) coi chúng gần như tương đồng.

Function calling mạnh ở độ tin cậy (nhờ JSON schema, ít rủi ro injection) và dễ dùng cho developer. Tool calling thì đa năng hơn, có thể multi-step và minh bạch với người dùng (AI “nói ra” quá trình), nhưng phức tạp hơn để triển khai. Tựu trung, function calling được xem như một dạng tool calling có cấu trúc chặt chẽ hơn; cả hai đều nhằm mở rộng năng lực LLM và kết nối với thế giới thực.

2.4.4. Ứng dụng thực tiễn

Việc cho phép các mô hình ngôn ngữ gọi công cụ đã mở ra rất nhiều ứng dụng thực tế mạnh mẽ, biến những ý tưởng trước đây khó khả thi thành hiện thực. Dưới đây là một số ứng dụng tiêu biểu của tool calling:

Tìm kiếm, hỗ trợ khách hàng & tác vụ nghiệp vụ: Tool calling cho phép AI kết nối với dữ liệu và dịch vụ thời gian thực. Ví dụ: ChatGPT Browsing hay Bing Chat có thể duyệt web, lấy tin tức mới, hay gọi API thời tiết để trả lời chính xác. Trong doanh nghiệp, chatbot nội bộ có thể gọi CRM hoặc hệ thống kho để trả lời về đơn hàng, tạo ticket IT, hoặc hỗ trợ nhân viên. Điều này biến AI từ chatbot tĩnh thành nhân viên ảo có khả năng truy cập và thao tác trên phần mềm công ty.

Tự động hóa & trợ lý thông minh: Các agent như Auto-GPT hay Sales Agent dùng tool calling để thực hiện chuỗi hành động nhiều bước: tìm kiếm, ghi file, phân tích dữ liệu, soạn email, thậm chí gửi mail đi. Trong lập trình và DevOps, AI có thể gọi công cụ chạy code, kiểm thử, triển khai dịch vụ... giúp lập trình viên và đội kỹ thuật tiết kiệm thời gian. Về phía người dùng cá nhân, Siri/Google Assistant thế hệ mới có thể đọc lịch, đặt bàn, gửi tin nhắn – tất cả chỉ bằng hội thoại tự nhiên nhờ tool calling.

Sáng tạo & tương tác thế giới thực: AI có thể kết hợp với công cụ đa phương tiện: gọi DALL·E để sinh ảnh, WolframAlpha để tính toán và vẽ biểu đồ, hay API SmartHome để bật/tắt thiết bị trong nhà. Nhờ đó, trợ lý AI không chỉ trả lời bằng văn bản mà còn tạo ra hình ảnh, báo cáo, hoặc thực hiện hành động thực tế. Đây là bước chuyển quan trọng: từ chatbot “nói cho vui” sang trợ lý “làm được việc thật”.

2.5. Tổng quan về Harmony Response Format

2.5.1. Khái niệm và mục đích sử dụng

Harmony Response Format (HRF) là định dạng cấu trúc do OpenAI giới thiệu cùng GPT-OSS (2025), nhằm chuẩn hóa cách mô hình tạo phản hồi trong hội thoại phức tạp. Nó cho phép phân tách rõ: câu trả lời cuối (final) dành cho người dùng, chuỗi suy nghĩ nội bộ (analysis/CoT), và các kết quả/trung gian như function call (commentary). Mục tiêu là làm cho hội thoại có cấu trúc, dễ kiểm soát, và minh bạch hơn. [6]

HRF bổ sung các vai trò (roles) như developer và tool được cấp bên cạnh user/assistant/system, cùng với các kênh (channels): final, analysis, commentary. Nhờ đó, mô hình có thể vừa hiển thị kết quả cuối cho người dùng, vừa ghi lại suy nghĩ nội bộ và lệnh gọi công cụ trong kênh riêng – giúp developer giám sát nhưng không lộ ra

ngoài. Ngoài ra, Harmony còn chuẩn hóa cách function calling: arguments và kết quả được đóng gói trong kênh commentary, có metadata rõ ràng để parser dễ xử lý.

Harmony Format được dùng trong API GPT-OSS, LM Studio và được open-source để cộng đồng áp dụng rộng rãi. Nó giúp quản lý hội thoại nhiều bước với tool calls và CoT minh bạch, hỗ trợ debugging, giám sát bias/misbehavior, đồng thời giảm rủi ro lẫn lộn suy nghĩ với câu trả lời. Nói cách khác, Harmony đóng vai trò như một “chuẩn HTML/CSS cho hội thoại AI” – thống nhất cách biểu diễn reasoning, tool use và response trong một khuôn khổ chặt chẽ.

2.5.2. Cấu trúc của *Harmony Response Format*

Harmony Response Format (HRF) tổ chức hội thoại thành chuỗi messages, mỗi message có các thuộc tính:

- **Role/Author:** gồm system, developer, user, assistant, và tool (author = tool name).
- **Content:** nội dung thông điệp, có thể văn bản, JSON, hoặc kết quả tool.
- **Channel:** riêng cho assistant/tool, gồm: analysis (CoT nội bộ), commentary (gọi tool, trung gian), final (trả lời người dùng).
- **Metadata:** content-type (<|constrain|> json, <|raw|> ...), recipient (vd: functions.get_weather khi gọi tool). [6]

Cấu trúc này giúp phân biệt rõ nguồn gốc và loại nội dung trong cuộc hội thoại. Ngoài ra, cơ chế hoạt động bao gồm:

- **Assistant (analysis):** giải thích suy luận, vì sao cần gọi tool.
- **Assistant (commentary):** in ra tham số JSON kèm recipient (các function cần gọi).
- **Tool (commentary):** trả về kết quả.
- **Assistant (analysis – tùy chọn):** phản ánh lại kết quả.
- **Assistant (final):** tạo câu trả lời cuối cho user.

Quy trình này cho phép tách bạch giữa suy nghĩ, hành động gọi công cụ, và trả lời cuối. Các token sentinel như <|constrain|> hay <|raw|> đảm bảo nội dung có định dạng rõ ràng, dễ parse và an toàn.

HRF thiết lập thứ tự ưu tiên giữa các role: system > developer > user > assistant > tool, nhằm xử lý xung đột chỉ dẫn. Nhờ cấu trúc này, GPT-OSS có thể quản lý hội thoại nhiều bước (multi-step) mà vẫn rõ ràng, đồng thời hỗ trợ developer quan sát chain-

of-thought mà không lộ cho người dùng. Dù phức tạp hơn format role+content cũ, HRF giúp tích hợp tool mượt mà, duy trì hội thoại trong sáng, và hỗ trợ stream output theo từng phần (analysis vs final).

2.5.3. Ưu điểm và ứng dụng

Ưu điểm của Harmony Response Format (HRF):

Harmony Response Format (HRF) cho phép tách biệt hoàn toàn phần suy luận nội bộ (analysis) và phần trả lời cho người dùng (final), giúp vừa đảm bảo an toàn (không rò rỉ chain-of-thought), vừa tăng minh bạch (developer vẫn xem được để debug). Bên cạnh đó, HRF chuẩn hóa function/tool calling thành dạng message có kênh riêng, thay thế cách nhúng JSON ad-hoc trước đây. Điều này giúp hỗ trợ tốt hơn cho các tình huống multi-step agent và làm cho việc tích hợp công cụ trở nên thống nhất, đáng tin cậy.

Nhờ giữ lại log reasoning và tool calls, HRF tạo điều kiện để developer phân tích chi tiết cách mô hình suy nghĩ và hành động, dễ dàng phát hiện sai sót logic hoặc lỗi tích hợp. Các UI như LM Studio có thể tận dụng format này để hiển thị tab riêng cho reasoning, tool log và final answer, cải thiện đáng kể trải nghiệm debug, huấn luyện và đánh giá. Đồng thời, HRF cũng cung cấp dữ liệu chain-of-thought chuẩn hóa để fine-tune hoặc nghiên cứu giám sát hành vi mô hình.

HRF hiện là định dạng mặc định cho GPT-OSS (20B, 120B), được tích hợp trong OpenAI Responses API và nhiều công cụ như LM Studio, OpenRouter. Với đặc tính mở, HRF hứa hẹn trở thành chuẩn chung cho hệ sinh thái agentic AI: một “ngôn ngữ” chung để mô hình, công cụ và UI phối hợp nhịp nhàng. Khi kết hợp cùng Model Context Protocol (MCP), HRF đảm nhiệm phần định dạng hội thoại và suy nghĩ, trong khi MCP lo phần giao tiếp với dữ liệu – tạo thành một “stack chuẩn” cho việc xây dựng AI agent phức tạp trong tương lai.

Ứng dụng thực tiễn của Harmony Response Format:

Harmony Response Format (HRF) đã trở thành định dạng mặc định khi làm việc với các mô hình GPT-OSS (20B, 120B), xuất hiện trong cả HuggingFace UI và LM Studio. Nó cũng là nền tảng cho OpenAI Responses API, cho phép developer tách riêng phần analysis và final khi lấy kết quả. Nhờ vậy, HRF không chỉ quen thuộc với cộng đồng open-source mà còn được kỳ vọng sẽ lan sang GPT-5 hoặc GPT-4 open-weight, trở thành chuẩn structured output cho API thế hệ mới.

Trong môi trường agent, HRF đóng vai trò như “mini protocol” giúp mô hình tự format luồng logic reasoning và tool calls, tránh phụ thuộc vào prompt fragile. Các dự án như Auto-GPT-OSS có thể tận dụng điều này để điều phối CoT và công cụ rõ ràng hơn. Đồng thời, việc phân tách final/analysis/commentary cho phép các UI như LM Studio hiển thị reasoning tab, tool log tab và final output riêng biệt – cải thiện đáng kể trải nghiệm debug, huấn luyện, fine-tune chain-of-thought, cũng như nghiên cứu giám sát AI theo thời gian thực.

Với khả năng chuẩn hóa giao tiếp, HRF không chỉ giúp quản lý mô hình GPT-OSS mà còn mở đường cho các chuẩn chung trong agentic AI. Khi kết hợp với Model Context Protocol (MCP), ta có một “stack” toàn diện: MCP xử lý kết nối AI-data, HRF xử lý cấu trúc reasoning và output. Nếu được cộng đồng chấp nhận rộng rãi, HRF có thể trở thành “HTML/CSS của AI agents”, giúp xây dựng hệ thống AI phức tạp an toàn, dễ kiểm soát và ít mang tính ad-hoc hơn.

2.6. Tổng quan về công nghệ phát triển ứng dụng web

2.6.1. Ngôn ngữ đánh dấu HyperText Markup Language (HTML)

HyperText Markup Language (HTML) là ngôn ngữ đánh dấu tiêu chuẩn dùng để xây dựng cấu trúc và nội dung cho các trang web. Ra đời năm 1991 bởi Tim Berners-Lee, HTML được xem như “bộ khung xương” của web: nó cho trình duyệt biết trang web gồm những gì (văn bản, hình ảnh, liên kết, biểu mẫu...) và mối quan hệ giữa chúng. Điểm đáng lưu ý là HTML không phải ngôn ngữ lập trình, mà là ngôn ngữ đánh dấu, sử dụng các thẻ (tags) để bao quanh và mô tả nội dung, chẳng hạn <p> cho đoạn văn, <h1> cho tiêu đề hay <a> cho liên kết. [7]

Mỗi tài liệu HTML mở đầu với khai báo <!DOCTYPE html> và được bao bởi cặp thẻ <html>...</html>. Bên trong, phần head chứa các siêu dữ liệu như tiêu đề trang, mã ký tự, liên kết CSS/JS, còn phần body chứa toàn bộ nội dung hiển thị: văn bản, hình ảnh, bảng biểu hay biểu mẫu. HTML cũng cho phép nhúng đa phương tiện thông qua các thẻ , <video>, <audio> và dùng các thẻ ngữ nghĩa như <header>, <main>, <footer> để tổ chức bố cục trang rõ ràng hơn.

Điểm cốt lõi của HTML là khả năng “siêu văn bản” – liên kết tài liệu thông qua thẻ <a>, tạo nên mạng lưới web toàn cầu. Qua các phiên bản, HTML đã không ngừng hoàn thiện. HTML5 (chuẩn hóa năm 2014) đánh dấu bước tiến lớn: bổ sung nhiều thẻ

ngữ nghĩa mới (<section>, <article>), hỗ trợ nhúng media trực tiếp mà không cần plugin, và mở rộng API cho ứng dụng web hiện đại như canvas đồ họa hay lưu trữ cục bộ.

HTML là nền tảng của mọi trang web: nó định nghĩa cấu trúc, còn CSS tạo phong cách và JavaScript mang lại tương tác. Ngay cả các framework hiện đại như React, Angular hay Vue cũng chỉ sinh ra HTML để trình duyệt hiển thị. Với cú pháp đơn giản, dễ học, tính chuẩn hóa và khả năng chạy trên mọi nền tảng, HTML vẫn là kiến thức nền tảng mà bất kỳ lập trình viên web nào cũng cần nắm vững.

2.6.2. Ngôn ngữ tạo kiểu Cascading Style Sheets (CSS)

Cascading Style Sheets (CSS) là ngôn ngữ định kiểu dùng để mô tả cách trình bày và bố cục của các phần tử trong HTML/XML. Nếu HTML định nghĩa khung và nội dung, thì CSS chính là lớp “áo” quyết định màu sắc, phông chữ, khoảng cách, kích thước và cách bố trí. CSS tách biệt phần nội dung và trình bày, nhờ đó việc bảo trì dễ dàng: thay đổi giao diện mà không cần chỉnh sửa cấu trúc HTML. [8]

Cú pháp CSS được tổ chức thành các quy tắc (rules) gồm selector để chỉ định phần tử cần áp dụng, và declaration chứa thuộc tính – giá trị (như color, margin, font-size). Điểm đặc trưng của CSS là tính “cascading” (phân tầng): khi nhiều quy tắc cùng áp dụng cho một phần tử, trình duyệt dựa vào mức độ ưu tiên (specificity) và thứ tự để chọn thuộc tính cuối cùng. Nhờ đó, CSS vừa linh hoạt vừa cho phép kế thừa tự nhiên – ví dụ đặt font chữ cho toàn trang tại <body>, các phần tử con tự động hưởng theo trừ khi bị ghi đè. [8]

CSS cung cấp bộ công cụ toàn diện cho thiết kế: kiểm soát kiểu chữ và màu sắc (color, font, background), định dạng bố cục (margin, padding, display, position), và xây dựng layout hiện đại với Flexbox và Grid. CSS3 mở rộng thêm hiệu ứng trực quan như bo góc, đổ bóng, chuyển sắc, hoạt hình và chuyển đổi 2D/3D mà trước đây phải dùng Flash hay JS. Đặc biệt, media queries hỗ trợ thiết kế đáp ứng (responsive design), cho phép giao diện tự điều chỉnh trên mọi thiết bị từ PC đến smartphone.

Tách riêng và tái sử dụng là thế mạnh của CSS: một file CSS có thể áp dụng cho nhiều trang, giúp duy trì phong cách nhất quán và dễ đổi theme. Các công cụ hiện đại như Sass, LESS hay framework như Bootstrap, Tailwind càng mở rộng sức mạnh của CSS, nhưng cuối cùng đều biên dịch về CSS thuần để trình duyệt xử lý. Vì vậy, cùng với HTML và JavaScript, CSS là nền tảng không thể thiếu của front-end web

development, biến khung HTML khô cứng thành giao diện trực quan, thẩm mỹ và thân thiện.

2.6.3. Ngôn ngữ lập trình JavaScript

JavaScript (JS) là ngôn ngữ lập trình kịch bản cấp cao, nổi bật nhất với vai trò chạy phía client trong trình duyệt, giúp trang web trở nên tương tác và động thay vì tĩnh. Cùng với HTML và CSS, nó tạo thành “bộ ba trụ cột” của World Wide Web. Được Brendan Eich phát triển năm 1995 trong Netscape Navigator, JS ban đầu chỉ xử lý tác vụ đơn giản như kiểm tra form. Tên “JavaScript” được đặt để tận dụng danh tiếng Java, nhưng về kỹ thuật hai ngôn ngữ hầu như không liên quan. [9]

JavaScript là ngôn ngữ thông dịch, động, có thể nhúng trực tiếp vào HTML qua thẻ `<script>` hoặc tách thành file .js. Trình duyệt với engine (như V8 của Chrome) sẽ thực thi JS ngay khi trang tải. JS vận hành trên mô hình sự kiện (event-driven) và thao tác trực tiếp với DOM, cho phép thay đổi nội dung, kiểu dáng và hành vi trang web theo hành động của người dùng. Đặc điểm này biến JS thành “xương sống” cho trải nghiệm web tương tác.

Sự ra đời của AJAX (Asynchronous JavaScript and XML) giữa những năm 2000 mở đường cho web động thế hệ mới: trang có thể gọi dữ liệu ngầm từ server mà không tải lại. Đây chính là nền tảng cho ứng dụng web đơn trang (SPA) như Gmail hay Google Maps. Sau đó, Node.js (2010) đưa JS lên server, biến nó thành ngôn ngữ full-stack, dùng chung cho cả client lẫn server, mở rộng khả năng từ web backend, script hệ thống đến ứng dụng desktop với Electron.

JavaScript sở hữu cộng đồng lớn với vô số thư viện và framework: jQuery (DOM), D3.js (đồ thị), React, Angular, Vue (UI component). Các chuẩn ECMAScript (ES) liên tục cải tiến ngôn ngữ: từ ES5 (2009) đến ES6/ES2015 (class, module, arrow function, async/await) và các bản hàng năm. Nhờ vậy, JS ngày càng hiện đại, dễ quản lý và có hiệu năng cao nhờ kỹ thuật JIT trong engine.

JavaScript có mặt ở mọi tầng: xử lý form, tạo nội dung động, kéo-thả, gọi API bất đồng bộ, xây dựng SPA, viết server bằng Node.js. Là ngôn ngữ duy nhất được mọi trình duyệt hiểu, JS trở thành tiêu chuẩn bắt buộc trong phát triển web. Dù có những nhược điểm lịch sử (thi hành khác nhau giữa trình duyệt, kiểu dữ liệu lỏng lẻo), các bản ES mới và công cụ như Babel đã khắc phục. Tóm lại, JS là công cụ không thể thiếu của lập trình web hiện đại, và mọi developer đều cần nắm vững.

2.6.4. Tailwind CSS

Tailwind CSS là một framework CSS hiện đại ra đời năm 2017, nổi bật với triết lý utility-first – cung cấp sẵn hàng loạt lớp tiện ích nhỏ (utility class) để lập trình viên ghép trực tiếp vào HTML. Khác với Bootstrap hay Foundation vốn có các thành phần UI cố định (button, card, navbar), Tailwind không áp đặt thiết kế sẵn mà cho phép tùy biến tối đa, giúp tạo giao diện nhanh và linh hoạt. [10]

Mỗi class trong Tailwind tương ứng một thuộc tính CSS cụ thể: ví dụ text-center (căn giữa), bg-blue-500 (nền xanh lam), p-4 (padding 1rem), flex (display: flex). Không cần viết CSS thủ công, dev chỉ cần kết hợp nhiều utility class để tạo thành UI mong muốn. Tailwind còn đi kèm hệ thống thiết kế chuẩn (design system) với bảng màu, khoảng cách, font-size theo scale hợp lý và có thể tùy chỉnh qua file tailwind.config.js.

Tailwind hỗ trợ responsive design qua prefix breakpoint (md:, lg:), state prefix như hover:, focus:, dark:. Điều này giúp styling cho nhiều màn hình và trạng thái trở nên ngắn gọn. Ngoài ra, hệ thống plugin phong phú (forms, typography, aspect-ratio, v.v.) giúp mở rộng tiện ích cho nhiều tình huống. Nhà phát triển cũng dễ dàng tùy biến để phù hợp với brand riêng. [10]

Làm việc trực tiếp trong HTML giúp tăng tốc độ phát triển UI, giảm nhu cầu đặt tên class phức tạp (BEM) hay viết CSS riêng. Dù nhiều class khiến HTML “dài hơn”, nhưng thực tế lại dễ đọc và dễ duy trì. Tailwind cũng có công cụ purge CSS, chỉ giữ lại class dùng thực tế nên file build cuối gọn nhẹ, thậm chí nhỏ hơn nhiều framework truyền thống.

Tailwind đang được ưa chuộng mạnh trong startup, prototyping nhanh, và đặc biệt hiệu quả khi kết hợp React, Vue hay các framework component-based. Nhiều theme, dashboard (Tailwind UI, Flowbite) ra đời dựa trên nó. Với cộng đồng phát triển lớn, Tailwind ngày càng trở thành một trong những framework CSS phổ biến nhất, mang lại tốc độ, sự linh hoạt và khả năng bảo trì cao cho dự án web hiện đại.

2.6.5. Ngôn ngữ lập trình Python

Python là một ngôn ngữ lập trình cấp cao, thông dịch, đa mục đích và nổi bật nhờ cú pháp rõ ràng, dễ đọc. Được Guido van Rossum giới thiệu năm 1991, Python hướng tới việc nâng cao hiệu suất lập trình viên hơn là tối ưu tốc độ máy. Triết lý Zen of Python nhấn mạnh sự đơn giản, trực quan, với câu nổi tiếng: “There should be one – and

preferably only one – obvious way to do it.” Chính nhờ triết lý này, Python trở thành ngôn ngữ được ưa chuộng cho cả người mới học lẫn chuyên gia. [11]

Python dùng thụt lề (indentation) thay vì dấu ngoặc nhọn, buộc code phải sạch và dễ đọc. Cú pháp gần ngôn ngữ tự nhiên, cho phép viết chương trình ngắn gọn và rõ ràng – thường ít dòng hơn Java hay C. Python còn cung cấp nhiều kiểu dữ liệu cấp cao như list, dict, set, tuple cùng cú pháp mạnh mẽ như list comprehension. Là ngôn ngữ động, thông dịch và có cơ chế quản lý bộ nhớ tự động, Python cho phép thử nghiệm nhanh trong REPL hoặc notebook như Jupyter.

Một trong những sức mạnh lớn nhất của Python là kho thư viện khổng lồ. Thư viện chuẩn hỗ trợ sẵn từ xử lý chuỗi, file, mạng, đến JSON/XML. Ngoài ra, cộng đồng phát triển cung cấp hàng trăm nghìn gói qua PyPI. Trong khoa học dữ liệu, Python gần như mặc định với NumPy, Pandas, SciPy, Matplotlib, scikit-learn. Trong AI/ML, nó thống trị nhờ TensorFlow, PyTorch. Trong web development, Django và Flask là lựa chọn phổ biến. Python cũng được dùng nhiều trong tự động hóa, scripting, DevOps, và thậm chí cả game, đồ họa, IoT. [11]

Python chạy trên hầu hết hệ điều hành, mã nguồn mở và có cộng đồng khổng lồ. Với khả năng tích hợp C/C++, Python vừa có thể làm “keo” kết nối hệ thống, vừa tối ưu được những phần tính toán nặng. Các dự án lớn như Google, Facebook, Dropbox, Blender đều dùng Python ở nhiều bộ phận. Thậm chí, MicroPython cho phép chạy Python trên vi điều khiển nhỏ, mở rộng sang lĩnh vực nhúng và IoT.

Nhờ dễ học, Python thường được chọn làm ngôn ngữ lập trình đầu tiên trong giáo dục. Trong doanh nghiệp, nó được dùng cho back-end, phân tích dữ liệu, nghiên cứu AI/ML. Trong cộng đồng, các hội nghị PyCon lan tỏa triết lý “Pythonic” – viết code đẹp, rõ ràng. Nhược điểm lớn nhất là hiệu năng chậm hơn C/Java, nhưng nhờ thư viện C tăng tốc, Python vẫn đủ mạnh cho phần lớn ứng dụng. Ngày nay, thành thạo Python được coi gần như kỹ năng bắt buộc trong khoa học dữ liệu và vô cùng hữu ích cho lập trình viên full-stack.

2.6.6. Node.js

Node.js là môi trường chạy JavaScript phía máy chủ, ra đời năm 2009 bởi Ryan Dahl và dựa trên engine V8 của Google Chrome. Trước Node.js, JavaScript gần như chỉ dùng ở trình duyệt; sự xuất hiện của Node đã mở rộng phạm vi sang back-end, biến JavaScript thành nền tảng viết server, API hoặc công cụ CLI. Nhờ đó, lập trình viên có

thể dùng chung một ngôn ngữ cho cả front-end và back-end, giảm rào cản học tập và tăng tốc phát triển fullstack.

Điểm nổi bật nhất của Node.js là mô hình event-driven, non-blocking IO và single-threaded event loop. Thay vì tạo nhiều luồng cho mỗi kết nối, Node xử lý tất cả trên một luồng và tận dụng callback/async-await khi IO hoàn tất. Cách tiếp cận này giúp Node có thể xử lý hàng chục nghìn kết nối đồng thời, cực kỳ phù hợp với ứng dụng realtime (chat, streaming, API). Kết hợp với engine V8 tốc độ cao và core viết bằng C++, Node có hiệu năng rất cạnh tranh trong các tác vụ IO-bound.

Node.js đi kèm hệ sinh thái không lồ thông qua npm – kho package lớn nhất hiện nay. Chỉ với npm install, dev có thể dùng hàng trăm nghìn thư viện: từ framework web (Express, Koa), kết nối database, xác thực, thanh toán... đến vô số tiện ích nhỏ. npm cũng là nền tảng của toàn bộ chuỗi công cụ front-end hiện đại như Webpack, Vite, ESLint, Jest. Vì vậy, ngay cả khi không dùng Node để viết server, hầu hết dev web đều cần Node để quản lý dependency và build toolchain.

Node đặc biệt mạnh trong ứng dụng realtime và microservices: chat server, game online, API JSON tốc độ cao, push notification. Nó cũng phổ biến trong công cụ phát triển front-end (bundler, linter, test runner). Các công ty lớn như Netflix, PayPal, Uber, Walmart, LinkedIn đều chọn Node cho hệ thống backend vì concurrency tốt, tốc độ phát triển nhanh, và dễ scale out bằng cách nhân bản process. Node cluster module còn cho phép tận dụng CPU đa lõi trong khi vẫn giữ mô hình đơn luồng thân thiện.

Nhược điểm của Node là không phù hợp cho CPU-bound tasks (xử lý video, machine learning), vì event loop dễ bị chặn. Callback hell cũng từng là vấn đề, nhưng đã được giải quyết nhờ promise và async/await; thêm vào đó, TypeScript giúp code Node dễ bảo trì hơn. Dù có giới hạn, Node vẫn là lựa chọn tuyệt vời cho ứng dụng IO-bound, realtime và microservices. Tóm lại, Node.js đã thay đổi cách xây dựng server hiện đại: nhẹ, nhanh, mở rộng tốt, và đặc biệt phù hợp với kỷ nguyên web realtime.

2.6.7. Framework Litestar Python

Litestar (trước đây gọi là Starlite) là một framework Python hiện đại, nhẹ và hiệu năng cao, được xây dựng trên nền tảng ASGI (Asynchronous Server Gateway Interface). Ra đời như một lựa chọn thay thế linh hoạt cho FastAPI hay Starlette, Litestar hướng tới triết lý batteries-included nhưng vẫn giữ sự tối giản và tập trung vào hiệu suất. Nhờ đó, nó nhanh chóng trở thành lựa chọn đáng chú ý để phát triển API và microservices.

Điểm mạnh cốt lõi của Litestar là khả năng xử lý bất đồng bộ: mọi request handler có thể viết bằng `async def`, giúp tận dụng tối đa I/O không chặn và cho throughput cao. Cú pháp định nghĩa route cũng gọn gàng và quen thuộc, với decorator như `@get("/path")` hay `@post("/path")`. Kết hợp cùng type hint, framework tự động parse dữ liệu và sinh ra OpenAPI schema, mang lại trải nghiệm tương tự FastAPI nhưng tối ưu hơn về hiệu suất.

Litestar cung cấp sẵn nhiều công cụ phục vụ phát triển API: hệ thống data validation dựa trên Pydantic hoặc Msgspec, tự động sinh tài liệu OpenAPI, cơ chế dependency injection, hỗ trợ middleware và hooks linh hoạt, cùng tích hợp với ORM (SQLAlchemy), cache (Redis), WebSocket và streaming response.

Mặc dù nhiều tính năng, Litestar vẫn giữ được sự modular và nhẹ. Framework cho phép dùng cả hàm `async` lẫn `sync` (`sync` sẽ được đưa vào threadpool để tránh chặn event loop). Triết lý thiết kế của Litestar là “extensibility with minimal opinions”, tức không ép buộc công nghệ cụ thể, đồng thời dễ mở rộng thông qua plugin.

Litestar phù hợp nhất cho API backend, microservices, và ứng dụng realtime cần hiệu năng và concurrency cao. Nó cũng hữu ích trong việc triển khai nhanh các service bọc mô hình machine learning, hoặc xây dựng public API với validation và docs rõ ràng. Dù cộng đồng còn nhỏ so với FastAPI, Litestar đang phát triển nhanh và được xem như một “điểm cân bằng” giữa FastAPI và Django: vừa mạnh mẽ, vừa gọn nhẹ. Tóm lại, Litestar là một lựa chọn đáng giá cho Python developer khi cần framework web bất đồng bộ, hiệu quả và dễ mở rộng.

2.6.8. Cơ sở dữ liệu quan hệ PostgreSQL

PostgreSQL (thường được gọi tắt là Postgres) là một hệ quản trị cơ sở dữ liệu quan hệ (RDBMS) mã nguồn mở, nổi bật về tính ổn định, khả năng mở rộng và độ tin cậy. Nó khởi nguồn từ dự án POSTGRES tại Đại học California, Berkeley vào thập niên 1980 dưới sự dẫn dắt của giáo sư Michael Stonebraker. Với triết lý tuân thủ chặt chẽ chuẩn SQL và bổ sung nhiều tính năng tiên tiến, PostgreSQL thường được mệnh danh là “hệ quản trị cơ sở dữ liệu mã nguồn mở tiên tiến nhất thế giới”.

PostgreSQL hỗ trợ đầy đủ các nguyên tắc ACID (Atomicity, Consistency, Isolation, Durability), nhờ đó đảm bảo tính toàn vẹn dữ liệu trong các giao dịch quan trọng như ngân hàng hay tài chính. Nó cung cấp hệ thống kiểu dữ liệu phong phú, hỗ trợ khóa ngoại, trigger, view, stored procedure đa ngôn ngữ, và mô hình đối tượng-quan hệ (object-relational). Bên cạnh đó, cơ chế MVCC (Multi-Version Concurrency Control)

giúp tăng hiệu năng truy cập đồng thời, cho phép nhiều transaction chạy song song mà không gây xung đột dữ liệu.

Điểm mạnh của PostgreSQL nằm ở khả năng mở rộng và tùy biến. Người dùng có thể định nghĩa kiểu dữ liệu, hàm, hoặc chỉ mục riêng; sử dụng nhiều loại index (B-tree, Hash, GIN, GiST, BRIN) để tối ưu các kiểu truy vấn khác nhau. PostgreSQL còn hỗ trợ dữ liệu phi quan hệ qua JSONB, cho phép lưu trữ và truy vấn tài liệu JSON hiệu quả. Ngoài ra, hệ thống còn tích hợp nhiều tính năng nâng cao như full-text search, PostGIS cho dữ liệu địa lý, Foreign Data Wrapper để kết nối cơ sở dữ liệu ngoài, và partitioning cho xử lý tập dữ liệu lớn.

PostgreSQL sử dụng cơ chế Write-Ahead Logging (WAL) để đảm bảo phục hồi khi hệ thống gặp sự cố, đồng thời hỗ trợ replication (primary–standby) nhằm tăng khả năng chịu tải đọc. Cơ chế role và phân quyền chi tiết giúp tăng cường bảo mật trong môi trường đa người dùng. Về hiệu năng, PostgreSQL đã cải thiện đáng kể, xử lý tốt dữ liệu hàng trăm triệu bản ghi, và có thể mở rộng thông qua các giải pháp như Citus. Bên cạnh đó, PostgreSQL có cộng đồng phát triển toàn cầu mạnh mẽ, với nhiều công cụ quản trị như pgAdmin, DBeaver, cùng driver hỗ trợ hầu hết ngôn ngữ lập trình.

PostgreSQL thường được lựa chọn trong các hệ thống doanh nghiệp cần độ tin cậy và tính toàn vẹn cao (tài chính, chính phủ), các ứng dụng web backend quy mô lớn, hay hệ thống GIS chuyên biệt nhờ PostGIS. Nhiều công ty lớn như Apple, Instagram, Reddit hay Skype đã sử dụng PostgreSQL trong hạ tầng của mình. Nhờ sự linh hoạt, tuân thủ chuẩn SQL, cùng sức mạnh mở rộng, PostgreSQL được ví như “Oracle mã nguồn mở”. Việc thành thạo PostgreSQL không chỉ hữu ích cho lập trình viên backend mà còn là kỹ năng thiết yếu với các DBA trong môi trường doanh nghiệp hiện đại.

2.7. Tổng quan về nền tảng và công cụ hỗ trợ

2.7.1. Runpod

Runpod là một nền tảng điện toán đám mây chuyên biệt cho các tác vụ AI, cho phép người dùng thuê và vận hành GPU mạnh mẽ với chi phí hợp lý. Ra đời nhằm giải quyết nhu cầu ngày càng cao về tài nguyên GPU cho huấn luyện mô hình học sâu, khai thác dữ liệu và suy luận mô hình lớn, Runpod tập trung vào việc đơn giản hóa trải nghiệm triển khai môi trường AI trên cloud. So với các nhà cung cấp lớn như AWS hay GCP, Runpod mang lại lựa chọn tối ưu hơn về chi phí và tính chuyên biệt cho cộng đồng AI.

Runpod cung cấp “pod” GPU với cấu hình linh hoạt (GPU, CPU, RAM, lưu trữ), hỗ trợ các GPU hiện đại như RTX 3090 hay A100. Người dùng có thể triển khai nhanh chóng qua giao diện web hoặc API, truy cập bằng SSH hay Jupyter Notebook. Điểm mạnh của Runpod nằm ở sự linh hoạt trong quản lý tài nguyên: chỉ tính phí khi máy chạy, hỗ trợ spot instance giá rẻ, cũng như dịch vụ serverless GPU endpoint để triển khai mô hình dưới dạng API tiết kiệm chi phí. Ngoài ra, các container sẵn của Runpod đã được cài đặt framework phổ biến (PyTorch, TensorFlow, Hugging Face), tích hợp persistent storage và hỗ trợ plugin cộng đồng, giúp người dùng tiết kiệm công sức thiết lập môi trường. [13]

Runpod đặc biệt hữu ích cho nhiều đối tượng. Với cá nhân hoặc nhà nghiên cứu, đây là giải pháp tiếp cận GPU mạnh với chi phí thấp, thay thế cho việc đầu tư phần cứng đắt đỏ. Với startup, Runpod giúp triển khai và mở rộng dịch vụ AI linh hoạt theo nhu cầu, giảm rủi ro về chi phí cố định. Cộng đồng AI mã nguồn mở cũng hưởng lợi khi có thể dễ dàng fine-tune và chia sẻ mô hình, tương tự Hugging Face Spaces nhưng tối ưu hơn cho GPU-heavy workloads. Các tình huống thực tế như huấn luyện mô hình ngôn ngữ lớn, triển khai inference qua API hay thử nghiệm thuật toán ML đều trở nên khả thi với Runpod.

So với Google Colab Pro, Runpod cho phép tùy chỉnh nhiều hơn và toàn quyền quản trị máy ảo; so với vast.ai, Runpod thân thiện hơn về giao diện và vận hành. Tuy vậy, Runpod vẫn có một số hạn chế: người dùng phải quản lý thủ công việc bật/tắt pod để tránh lãng phí chi phí; việc scale multi-node còn phức tạp; hiệu năng GPU đôi khi không đồng nhất do nguồn cung từ nhiều datacenter khác nhau; và việc truyền tải dữ liệu lớn còn phụ thuộc vào băng thông mạng.

Tóm lại, Runpod là một nền tảng điện toán đám mây hướng chuyên sâu cho AI, giúp dân chủ hóa khả năng tiếp cận GPU và giảm đáng kể rào cản phần cứng cho cả cá nhân lẫn doanh nghiệp. Với chi phí cạnh tranh, môi trường sẵn sàng cho ML/DL và khả năng triển khai nhanh chóng, Runpod ngày càng trở thành lựa chọn phổ biến để huấn luyện, thử nghiệm và triển khai các mô hình AI hiện đại. Nó không chỉ đóng vai trò công cụ hỗ trợ mà còn góp phần thúc đẩy hệ sinh thái AI mở, nơi bất kỳ ai cũng có thể khai thác sức mạnh tính toán GPU ở quy mô lớn.

2.7.2. *Hugging Face*

Hugging Face là một nền tảng AI mã nguồn mở nổi bật, tập trung vào xử lý ngôn ngữ tự nhiên (NLP) và học sâu. Bắt đầu từ một ứng dụng trò chuyện năm 2016, Hugging Face nhanh chóng trở thành trung tâm cộng đồng AI toàn cầu, nổi tiếng nhờ thư viện Transformers và Model Hub. Ngày nay, nó được xem như “GitHub của AI”, đóng vai trò cầu nối giữa nghiên cứu và ứng dụng, giúp chia sẻ, tái sử dụng mô hình và dữ liệu một cách dễ dàng. [14]

Transformers là sản phẩm chủ lực của Hugging Face, cung cấp API Python đơn giản để tải và sử dụng các mô hình pre-trained cho nhiều tác vụ NLP như phân loại, dịch, tóm tắt hay sinh văn bản. Thư viện này hỗ trợ hàng trăm mô hình nổi tiếng (BERT, GPT-2, T5, RoBERTa...) và tương thích với PyTorch, TensorFlow, JAX. Điểm mạnh là khả năng fine-tune nhanh chóng và dân chủ hóa việc tiếp cận mô hình mạnh mẽ: chỉ với vài dòng code, nhà phát triển có thể ứng dụng mô hình vào bài toán thực tế.

Model Hub là kho lưu trữ mở, hiện có hơn một triệu mô hình và hàng trăm nghìn dataset, đi kèm mô tả, config và widget thử trực tiếp trên web. Ngoài mô hình NLP, Hub còn chứa mô hình về thị giác máy tính, âm thanh và đa phương thức. Hugging Face cũng cung cấp Spaces, cho phép triển khai demo AI (qua Gradio hoặc Streamlit) chỉ trong vài phút. Nhờ đó, Hub trở thành nơi cộng đồng AI thảo luận, chia sẻ và khám phá nhanh chóng các tiến bộ mới.

Bên cạnh mã nguồn mở, Hugging Face cung cấp các dịch vụ trả phí như Inference API (gọi mô hình qua API mà không cần triển khai), AutoTrain (fine-tune tự động), hay Hub Enterprise cho quản lý mô hình riêng tư. Các nền tảng lớn như AWS, Azure, PyTorch cũng tích hợp trực tiếp Hugging Face, biến nó thành chuẩn de-facto trong AI. Sự phổ biến này rút ngắn khoảng cách từ nghiên cứu đến ứng dụng, khi hầu hết các mô hình mới đều được phát hành ngay trên Hugging Face.

Hugging Face theo đuổi triết lý “AI vì mọi người”, tích cực thúc đẩy mã nguồn mở, hỗ trợ các dự án cộng đồng như BLOOM (mô hình 176B open-weight). Nhờ cách tiếp cận mở và dân chủ, Hugging Face đã trở thành hạ tầng không thể thiếu cho lập trình viên, nhà nghiên cứu và doanh nghiệp. Tóm lại, Hugging Face không chỉ là nơi lưu trữ mô hình mà còn là hệ sinh thái toàn diện, giúp cộng đồng AI toàn cầu hợp tác, học hỏi và sáng tạo nhanh hơn.

2.7.3. Google Colab

Google Colab (Collaboratory) là dịch vụ notebook trực tuyến do Google cung cấp, cho phép viết và chạy mã Python ngay trên trình duyệt mà không cần cài đặt môi trường cục bộ. Ra mắt từ 2017, Colab nhanh chóng phổ biến trong cộng đồng học máy và khoa học dữ liệu nhờ sự tiện lợi, khả năng cộng tác giống Google Docs, và đặc biệt là tài nguyên tính toán miễn phí. [15]

Colab cung cấp GPU và cả TPU miễn phí theo phiên, giúp người dùng có thể huấn luyện và thử nghiệm mô hình deep learning mà không cần phần cứng riêng. Môi trường đi kèm sẵn các thư viện phổ biến như TensorFlow, PyTorch, scikit-learn, NumPy, Pandas..., đồng thời hỗ trợ cài thêm gói qua pip. Notebook được lưu trên Google Drive dưới định dạng .ipynb, dễ chia sẻ hoặc mở lại bất kỳ lúc nào. [15]

Người dùng có thể chia sẻ notebook Colab để cộng tác realtime, hoặc công khai cho cộng đồng sử dụng, tương tự như Google Docs. Colab cũng tích hợp tốt với GitHub, cho phép mở và chạy trực tiếp notebook từ repo. Nhờ đó, nhiều dự án mã nguồn mở và khoá học AI/ML đã chọn Colab làm công cụ chính để phát hành tutorial hoặc thực hành.

Ngoài bản miễn phí, Colab còn có gói Pro và Pro+ với chi phí hàng tháng, cung cấp GPU mạnh hơn (T4, P100), thời gian chạy lâu hơn và ít bị giới hạn tài nguyên. Tuy vậy, Colab vẫn có những hạn chế: dễ ngắt kết nối khi chạy lâu, không đảm bảo bảo mật cho dữ liệu nhạy cảm, và không phù hợp cho môi trường sản xuất hoặc tác vụ cần mở rộng quy mô lớn.

Colab đã góp phần quan trọng trong việc dân chủ hóa học máy: sinh viên, nhà nghiên cứu và lập trình viên ở mọi nơi đều có thể tiếp cận GPU để học tập và thử nghiệm. Các khóa học trực tuyến, bài giảng và dự án AI hiện nay hầu hết đều cung cấp phiên bản Colab để người học chạy trực tiếp. Nhờ đó, Colab không chỉ là công cụ tiện lợi mà còn là chất xúc tác thúc đẩy sự lan tỏa của machine learning trong giáo dục và nghiên cứu.

2.7.4. Google Apps Script

Google Apps Script là một nền tảng phát triển ứng dụng nhẹ chạy trên đám mây, cho phép mở rộng và tự động hóa các sản phẩm Google Workspace như Sheets, Docs, Forms, Gmail hay Drive. Nó sử dụng ngôn ngữ JavaScript (phiên bản ES5) cùng các API tích hợp sẵn từ Google, giúp người dùng dễ dàng viết kịch bản xử lý công việc mà không cần cài đặt thêm phần mềm hay máy chủ riêng. [16]

Điểm mạnh nhất của Apps Script là khả năng điều khiển trực tiếp các dịch vụ Google. Ví dụ: SpreadsheetApp thao tác trên Sheets, GmailApp gửi và lọc email, FormsApp quản lý form và câu trả lời, hay CalendarApp tạo và chỉnh sửa sự kiện. Nhờ đó, Apps Script trở thành “keo dính” kết nối các ứng dụng Google với nhau, giúp giảm đáng kể công việc thủ công.

Apps Script hỗ trợ trigger để script chạy tự động khi có sự kiện (người dùng sửa ô, nộp form, mở tài liệu) hoặc theo lịch định sẵn (chạy hằng ngày, hằng tuần). Điều này cho phép xây dựng quy trình hoàn toàn tự động – từ gửi báo cáo qua email, đồng bộ dữ liệu giữa nhiều bảng tính, cho đến cảnh báo khi có thay đổi quan trọng.

Script có thể gắn vào một file cụ thể hoặc tồn tại độc lập trên Google Drive. Người dùng chia sẻ và phân quyền tương tự tài liệu Google. Ngoài ra, Apps Script có thể được xuất bản thành web app hoặc API nhỏ, nghĩa là bạn có thể xây dựng một ứng dụng mini chạy hoàn toàn trên hạ tầng Google mà không tốn chi phí duy trì server.

Trong doanh nghiệp, Apps Script thường dùng để tạo báo cáo tự động, quản lý quy trình phê duyệt, đồng bộ dữ liệu, hay gửi email hàng loạt từ Google Sheets. Với cú pháp đơn giản và thư viện phong phú, ngay cả người dùng văn phòng không chuyên lập trình cũng có thể xây dựng công cụ tùy chỉnh. Nhờ vậy, Google Apps Script trở thành giải pháp “dân chủ hóa” tự động hóa trong môi trường làm việc hiện đại.

CHƯƠNG 3. HIỆN THỰC HÓA NGHIÊN CỨU

3.1. Mô tả bài toán

Hiện nay, tại Trường Đại học Trà Vinh, quy trình tiếp nhận và xử lý văn bản đến hiện nay vẫn chủ yếu thực hiện theo phương thức thủ công. Khi có một công văn hoặc tài liệu gửi đến, nhân viên văn thư phải tải tệp về, đọc nội dung, ghi chú lại các thông tin cần thiết rồi nhập tay vào sổ theo dõi hoặc phần mềm quản lý. Sau đó, họ phải xác định văn bản thuộc loại nào, cơ quan nào ban hành, và phòng ban nào có trách nhiệm xử lý. Nếu cần gửi thông báo cho các đơn vị liên quan, nhân viên cũng phải soạn email và đính kèm văn bản thủ công. Toàn bộ quá trình này vừa tốn nhiều thời gian, vừa dễ xảy ra nhầm lẫn khi số lượng văn bản đến ngày càng nhiều.

Đề tài này được thực hiện với mục tiêu xây dựng một hệ thống tự động hóa quy trình xử lý văn bản đến. Cụ thể, sau khi người dùng tải tệp văn bản lên, hệ thống sẽ tự động thực hiện nhận dạng ký tự (OCR), trích xuất các thông tin quan trọng như loại văn bản, cơ quan ban hành, tiêu đề và nội dung chính. Trên cơ sở đó, hệ thống phân tích và đưa ra gợi ý về hướng xử lý, ví dụ như chuyển cho phòng ban nào hoặc giao đơn vị nào phối hợp. Đồng thời, hệ thống có khả năng chuẩn bị sẵn bản nháp email thông báo kèm tệp văn bản, giúp người dùng dễ dàng duyệt, chỉnh sửa và xác nhận gửi đi.

Tóm lại, sau khi hoàn thiện, hệ thống sẽ hỗ trợ người dùng từ khâu tiếp nhận, phân tích, trích xuất thông tin, gợi ý xử lý đến tự động hóa việc thông báo qua email và lưu trữ dữ liệu. Nhờ vậy, quy trình làm việc trở nên nhanh chóng, chính xác và minh bạch hơn, giảm thiểu thao tác thủ công, nâng cao hiệu quả quản lý văn bản đến trong toàn trường.

3.2. Phân tích yêu cầu và lựa chọn công nghệ

3.2.1. Phân tích yêu cầu

Để xây dựng một hệ thống hỗ trợ xử lý văn bản đến tự động, trước hết cần phân tích kỹ các yêu cầu xuất phát từ thực tế nghiệp vụ. Quy trình hiện tại cho thấy có nhiều bước lặp đi lặp lại, đòi hỏi nhập liệu thủ công, trong khi khối lượng văn bản ngày càng lớn. Do đó, hệ thống cần đảm bảo ba yếu tố cốt lõi: (i) giảm tải thao tác thủ công, (ii) nâng cao độ chính xác trong phân loại và gợi ý xử lý, và (iii) giữ tính minh bạch, bảo mật trong toàn bộ quy trình.

Ngoài các yêu cầu nghiệp vụ, hệ thống cũng phải đáp ứng yêu cầu kỹ thuật. Dữ liệu đầu vào có thể đa dạng (PDF, ảnh scan, tài liệu văn bản số), chất lượng không đồng

đều. Kết quả đầu ra phải có dạng dữ liệu chuẩn hóa để lưu trữ trong cơ sở dữ liệu, đồng thời gắn với trạng thái xử lý và lịch sử thao tác của người dùng. Hệ thống phải dễ tích hợp với hạ tầng sẵn có của nhà trường, có khả năng mở rộng khi số lượng văn bản và người dùng tăng lên.

Từ những phân tích này, có thể rút ra yêu cầu chung: hệ thống cần đóng vai trò là một trợ lý tự động, giúp người dùng xử lý văn bản nhanh chóng, đồng thời vẫn cho phép người dùng kiểm soát, phê duyệt và chỉnh sửa trước khi quyết định cuối cùng được ban hành.

3.2.2. Yêu cầu chức năng

Các yêu cầu chức năng chính của hệ thống bao gồm:

- Đăng nhập và quản lý người dùng
 - Cho phép đăng nhập theo vai trò (nhân viên văn thư, quản trị viên).
 - Phân quyền để đảm bảo chỉ những người có thẩm quyền mới được thao tác nhất định.
- Tiếp nhận và lưu trữ văn bản
 - Người dùng chỉ cần tải hoặc gửi file văn bản (PDF, ảnh scan, tài liệu số) lên hệ thống.
 - Tập sẽ được lưu trữ an toàn, gắn với thông tin cơ bản để thuận tiện tra cứu sau này.
- Xử lý và trích xuất thông tin tự động
 - Sau khi nhận file, hệ thống tự động thực hiện OCR để nhận diện ký tự.
 - Trích xuất ra các thông tin quan trọng như loại văn bản, cơ quan ban hành, tiêu đề, nội dung chính.
 - Kết quả được chuẩn hóa và lưu trực tiếp vào cơ sở dữ liệu.
- Gợi ý hướng xử lý văn bản
 - Hệ thống phân tích nội dung và tự động đưa ra gợi ý “chuyên đến phòng ban nào” hoặc “giao đơn vị nào phối hợp”.
 - Người dùng chỉ cần xem lại, có thể chỉnh sửa trước khi xác nhận.
- Tạo và quản lý thông báo
 - Tự động tạo bản nháp email kèm tệp văn bản và nội dung tóm tắt.
 - Người dùng chỉ cần duyệt hoặc chỉnh sửa trước khi gửi đi.
 - Lưu lại toàn bộ lịch sử email (đã gửi, đang chờ, lỗi).

- Tra cứu và quản lý dữ liệu
 - Cho phép tìm kiếm, lọc văn bản theo nhiều tiêu chí như loại, cơ quan ban hành, phòng ban xử lý.
 - Hiện thị lịch sử xử lý để dễ dàng theo dõi và kiểm tra.
- Quản trị hệ thống
 - Quản lý người dùng, phân quyền, cấu hình các phòng ban và quy tắc xử lý.
 - Theo dõi log hoạt động để đảm bảo minh bạch và truy vết khi cần.

3.2.3. Yêu cầu phi chức năng

Ngoài những chức năng cốt lõi, hệ thống cần đáp ứng một số yêu cầu phi chức năng quan trọng để đảm bảo tính khả thi, ổn định và thân thiện khi đưa vào vận hành:

- Hiệu năng
 - Hệ thống phải xử lý một văn bản trong thời gian ngắn (ví dụ: dưới 30 giây cho một tệp PDF trung bình).
 - Cho phép nhiều người dùng truy cập và xử lý đồng thời mà không gây nghẽn hoặc chậm trễ.
- Độ chính xác
 - Tỷ lệ nhận dạng ký tự (OCR) cần đạt mức chấp nhận được đối với tài liệu tiếng Việt, kể cả khi văn bản có dấu hoặc chất lượng scan không hoàn hảo.
 - Gợi ý phân công xử lý phải có độ chính xác cao, giúp người dùng chỉ cần chỉnh sửa tối thiểu.
- Bảo mật
 - Dữ liệu văn bản hành chính phải được bảo vệ nghiêm ngặt: mã hóa khi lưu trữ và truyền tải, kiểm soát truy cập theo vai trò người dùng.
 - Hệ thống cần có cơ chế ghi nhật ký (log) đầy đủ để phục vụ giám sát và truy vết khi có sự cố.
- Khả năng mở rộng
 - Thiết kế hệ thống theo hướng linh hoạt, có thể mở rộng khi số lượng văn bản và người dùng tăng.
 - Dễ dàng tích hợp với các phần mềm quản lý văn bản hoặc hệ thống tác nghiệp hiện có của trường.
- Tính ổn định và dễ sử dụng

- Giao diện người dùng cần đơn giản, thân thiện và hỗ trợ tiếng Việt, giúp nhân viên văn thư thao tác dễ dàng.
- Hệ thống phải hoạt động ổn định, hạn chế lỗi và gián đoạn, đồng thời có khả năng khôi phục nhanh khi gặp sự cố
- **Minh bạch và truy vết**
 - Mọi thao tác xử lý văn bản cần được ghi nhận và lưu vết.
 - Người quản trị có thể theo dõi lại toàn bộ quá trình, từ lúc tiếp nhận cho đến khi văn bản được phân công xử lý và gửi thông báo.

3.2.4. Lựa chọn công nghệ

Dựa trên các yêu cầu đã phân tích, hệ thống được thiết kế với sự kết hợp của nhiều công nghệ hiện đại, nhằm đảm bảo hiệu quả, tính mở rộng và dễ triển khai. Các lựa chọn công nghệ chính bao gồm:

- **Ngôn ngữ và nền tảng lập trình**
 - Python được chọn làm ngôn ngữ chính cho backend vì có hệ sinh thái phong phú, hỗ trợ mạnh cho xử lý ngôn ngữ tự nhiên (NLP), trí tuệ nhân tạo (AI) và các thư viện phục vụ OCR, trích xuất văn bản. Python cũng dễ đọc, dễ bảo trì, thuận tiện cho phát triển nhóm.
 - Node.js được sử dụng hỗ trợ trong việc xây dựng giao diện và tooling frontend (ví dụ: tích hợp Tailwind CSS).
- **Framework backend**
 - Litestar (Python Framework) được lựa chọn để xây dựng backend nhờ khả năng định tuyến API RESTful rõ ràng, hiệu suất cao và dễ mở rộng.
- **Frontend và giao diện người dùng**
 - HTML, CSS, JavaScript là nền tảng cơ bản để xây dựng giao diện web.
 - Tailwind CSS giúp phát triển giao diện nhanh chóng, hiện đại và đồng bộ.
- **Xử lý và mô hình ngôn ngữ**
 - GPT-OSS (Generative Pre-trained Transformer – Open Source Series) là mô hình mã nguồn mở được chọn cho các tác vụ NLP như trích xuất thông tin, phân loại văn bản, sinh gợi ý. Ưu điểm là dễ tùy biến và triển khai độc lập.
 - LangChain và RAG (Retrieve–Augment–Generate) hỗ trợ xây dựng pipeline xử lý nâng cao, kết hợp truy xuất dữ liệu và sinh văn bản.

- MCP (Model Context Protocol) quản lý ngữ cảnh, giúp tích hợp nhiều công cụ AI một cách mạch lạc.
- **Cơ sở dữ liệu**
 - PostgreSQL được chọn nhờ tính ổn định, khả năng mở rộng và hỗ trợ tốt các kiểu dữ liệu phức tạp (JSONB, trigger, constraint). Đây là nền tảng phù hợp để lưu trữ metadata văn bản, người dùng và lịch sử thao tác.
- **Hạ tầng triển khai và công cụ hỗ trợ**
 - Runpod cung cấp hạ tầng GPU chuyên dụng để triển khai hoặc tinh chỉnh mô hình AI.
 - Hugging Face hỗ trợ quản lý mô hình, dataset và cung cấp endpoint thử nghiệm.
 - Google Colab được dùng như môi trường thí nghiệm và xử lý dữ liệu, đặc biệt hữu ích cho việc chạy OCR hàng loạt, tiền xử lý văn bản và tạo bộ dataset huấn luyện.
 - Google Apps Script được sử dụng để tự động hóa các tác vụ xử lý dữ liệu và tích hợp với Google Workspace, phục vụ việc làm sạch và chuẩn hóa dữ liệu trước khi đưa vào hệ thống.
- **Bảo mật dữ liệu**
 - AES-GCM dùng để mã hóa dữ liệu nhạy cảm, đảm bảo an toàn khi lưu trữ và truyền tải.
 - HMAC (Hash-based Message Authentication Code) để xác thực tính toàn vẹn của dữ liệu, chống giả mạo.

3.3. Thiết kế kiến trúc hệ thống

3.3.1. Thiết kế kiến trúc phần mềm

Kiến trúc phần mềm của hệ thống được thiết kế theo hướng modular, nghĩa là chia thành các thành phần riêng biệt, mỗi thành phần đảm nhận một nhiệm vụ cụ thể. Cách tiếp cận này giúp hệ thống dễ mở rộng, bảo trì và đảm bảo sự rõ ràng trong quá trình vận hành.

Thành phần thứ nhất là giao diện người dùng (Frontend/UI). Đây là nơi nhân viên văn thư trực tiếp thao tác với hệ thống. Giao diện được xây dựng bằng HTML, CSS, Tailwind CSS và JavaScript. Người dùng có thể tải file văn bản lên, xem kết quả xử lý, nhận gợi ý phân công, duyệt bản nháp email và thực hiện tra cứu lịch sử. Giao diện được

thiết kế theo hướng đơn giản, trực quan, ưu tiên sự tiện lợi để người dùng không cần nhiều kiến thức kỹ thuật vẫn có thể sử dụng thành thạo.

Thành phần thứ hai là dịch vụ ứng dụng (Backend Service). Đây là nơi xử lý nghiệp vụ chính của hệ thống, được triển khai bằng framework Litestar (Python). Backend cung cấp các API cho frontend gọi tới, bao gồm: tải và lưu trữ văn bản, thực hiện OCR, trích xuất thông tin, gợi ý phòng ban, quản lý người dùng, gửi email và lưu log. Mọi logic quan trọng như kết nối mô hình ngôn ngữ GPT-OSS, phối hợp với RAG và LangChain để sinh gợi ý đều diễn ra tại đây.

Thành phần thứ ba là xử lý trí tuệ nhân tạo (AI/NLP). Bộ phận này chịu trách nhiệm xử lý ngôn ngữ tự nhiên và nhận diện văn bản. Đầu tiên là OCR để nhận dạng chữ từ file scan hoặc PDF. Sau đó, hệ thống thực hiện tiền xử lý văn bản như làm sạch và chuẩn hóa tiếng Việt. Tiếp theo, mô hình GPT-OSS được sử dụng để trích xuất các trường thông tin cần thiết và đưa ra gợi ý xử lý. Quá trình này có sự hỗ trợ của LangChain và RAG nhằm kết hợp khả năng truy xuất dữ liệu và sinh văn bản. MCP đóng vai trò quản lý ngữ cảnh để chatbot duy trì mạch hội thoại xuyên suốt.

Thành phần thứ tư là cơ sở dữ liệu (Database). Toàn bộ thông tin văn bản, metadata, lịch sử xử lý, tài khoản người dùng và log hệ thống đều được lưu trữ trong PostgreSQL. Đây là hệ quản trị cơ sở dữ liệu mạnh mẽ, hỗ trợ tốt các mối quan hệ phức tạp và lưu trữ linh hoạt với JSONB. Việc tổ chức dữ liệu chặt chẽ giúp đảm bảo tính toàn vẹn, đồng thời tạo điều kiện thuận lợi cho việc tra cứu và truy vết sau này.

Thành phần thứ năm là hạ tầng và công cụ hỗ trợ. Các mô hình AI sẽ được triển khai trên Runpod, tận dụng GPU để tăng tốc độ xử lý. Hugging Face được dùng để quản lý mô hình và dataset trong quá trình thử nghiệm. Google Colab và Google Apps Script hỗ trợ xử lý dữ liệu và xây dựng bộ dataset huấn luyện, giúp hệ thống liên tục cải thiện chất lượng. Ngoài ra, dịch vụ email (tích hợp Gmail) được sử dụng để tự động hóa khâu gửi thông báo cho các đơn vị liên quan.

Thành phần cuối cùng là bảo mật và giám sát. Vì dữ liệu văn bản hành chính mang tính nhạy cảm, hệ thống áp dụng cơ chế mã hóa AES-GCM để bảo vệ dữ liệu khi lưu trữ và truyền tải, đồng thời sử dụng HMAC để xác thực tính toàn vẹn của thông tin. Bên cạnh đó, cơ chế phân quyền dựa trên vai trò người dùng được áp dụng chặt chẽ. Mọi thao tác đều được ghi log để phục vụ kiểm tra, giám sát và đảm bảo tính minh bạch trong quá trình vận hành.

system settings	user settings	verify codes	documents	document attachment	schedules email attachments
-----------------	---------------	--------------	-----------	---------------------	-----------------------------

$$G = \langle 11 : a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z \rangle$$
[illegible]

- Bảng người dùng:

Bảng 3. 2. Bảng người dùng

Trường	Kiểu dữ liệu	Mặc định	Bắt buộc	Giải thích
user_id	CHAR(36) PRIMARY KEY	-	Có	Mã người dùng (UUID).
user_name	VARCHAR(100) UNIQUE	-	Không	Tên đăng nhập duy nhất.
user_display_name	VARCHAR(255)	-	Không	Tên hiển thị của người dùng.
user_email	VARCHAR(255) UNIQUE	-	Có	Email người dùng.
user_password_hash	TEXT	-	Không	Mật khẩu đã được băm.
user_role	VARCHAR(20)	-	Có	Vai trò (admin, user, internal).
user_status	VARCHAR(20)	'active'	Không	Trạng thái (active, suspended, banned, deactivated).
user_oauth_provider	VARCHAR(50)	-	Không	Nhà cung cấp OAuth (nếu có).
user_oauth_sub	VARCHAR(255)	-	Không	Sub ID từ OAuth provider.
user_email_verified	BOOLEAN	FALSE	Không	Trạng thái xác minh email.
user_register_ip	INET	-	Không	IP khi đăng ký.
user_created_at	TIMESTAMP WITH TIME ZONE	CURRENT_TIMESTAMP	Không	Thời điểm tạo user.
user_updated_at	TIMESTAMP WITH TIME ZONE	CURRENT_TIMESTAMP	Không	Thời điểm cập nhật gần nhất.

(UNIQUE constraint)	(user_oauth_provider, user_oauth_sub)	-	-	Đảm bảo một cặp OAuth duy nhất.
---------------------	---------------------------------------	---	---	---------------------------------

- Bảng thiết lập người dùng:

Bảng 3. 3. Bảng thiết lập người dùng

Trường	Kiểu dữ liệu	Mặc định	Bắt buộc	Giải thích
setting_user_id	CHAR(36) PRIMARY KEY (FK)	-	Có	Khóa chính, đồng thời FK tham chiếu users.user_id.
setting_default_prompt	TEXT	-	Không	Prompt mặc định của người dùng.
setting_theme	VARCHAR(20)	'light'	Không	Giao diện: light, dark, hoặc system.
setting_user_avatar_url	TEXT	-	Không	Ảnh đại diện của người dùng.
setting_allow_memory_lookup	BOOLEAN	TRUE	Không	Cho phép tra cứu bộ nhớ hội thoại.
setting_allow_memory_storage	BOOLEAN	TRUE	Không	Cho phép lưu trữ bộ nhớ hội thoại.
setting_remembered_summary	TEXT	-	Không	Tóm tắt được lưu lại.
setting_timezone	VARCHAR(50)	-	Không	Múi giờ của người dùng.
setting_enable_mapping	BOOLEAN	FALSE	Không	Bật chức năng mapping đặc biệt.
setting_created_at	TIMESTAMP WITH TIME ZONE	CURRENT_TIMESTAMP	Không	Thời điểm tạo thiết lập.
setting_updated_at	TIMESTAMP WITH TIME ZONE	CURRENT_TIMESTAMP	Không	Thời điểm cập nhật gần nhất.

- Bảng dự án:

Bảng 3. 4. Bảng dự án

Trường	Kiểu dữ liệu	Mặc định	Bắt buộc	Giải thích
project_id	CHAR(36) PRIMARY KEY	-	Có	Mã dự án (UUID).
project_name	VARCHAR(255)	-	Có	Tên dự án.
project_description	TEXT	-	Không	Mô tả dự án.
project_owner_id	CHAR(36) (FK)	-	Không	Người sở hữu dự án, tham chiếu users.user_id.
project_prompt	TEXT	-	Không	Prompt mặc định của dự án.
project_color	VARCHAR(20)	-	Không	Màu đại diện cho dự án.
project_file_path	TEXT	-	Không	Đường dẫn file của dự án.
project_created_at	TIMESTAMP WITH TIME ZONE	CURRENT_TIMESTAMP	Không	Thời điểm tạo dự án.
project_updated_at	TIMESTAMP WITH TIME ZONE	CURRENT_TIMESTAMP	Không	Thời điểm cập nhật gần nhất.

- Bảng Model AI:

Bảng 3. 5. Bảng Model AI

Trường	Kiểu dữ liệu	Mặc định	Bắt buộc	Giải thích
model_id	CHAR(36) PRIMARY KEY	-	Có	Mã model (UUID).
model_name	VARCHAR(100) UNIQUE	-	Có	Tên model, duy nhất.
model_provider	VARCHAR(100)	-	Không	Nhà cung cấp model (ví dụ: OpenAI, Anthropic).

model_type	VARCHAR(50)	-	Không	Kiểu model (text, image, embedding...).
model_description	TEXT	-	Không	Mô tả chi tiết model.
model_enabled	BOOLEAN	TRUE	Không	Trạng thái bật/tắt model.
model_access_scope	VARCHAR(20)	'all'	Không	Phạm vi truy cập: all, user, internal, admin.
model_created_at	TIMESTAMP WITH TIME ZONE	CURRENT_TIMESTAMP	Không	Thời điểm tạo model.

- Bảng lịch sử chat:

Bảng 3. 6. Bảng lịch sử chat

Trường	Kiểu dữ liệu	Mặc định	Bắt buộc	Giải thích
chat_id	CHAR(36) PRIMARY KEY	-	Có	Mã cuộc chat (UUID).
chat_user_id	CHAR(36) (FK)	-	Không	Người tham gia, tham chiếu users.user_id.
chat_project_id	CHAR(36) (FK)	-	Không	Dự án liên quan, tham chiếu projects.project_id.
initial_model_id	CHAR(36) (FK)	-	Không	Model khởi tạo chat, tham chiếu model_variants.model_id.
chat_title	VARCHAR(255)	-	Không	Tiêu đề của chat.
chat_tokens_input	INT	0	Không	Số token input.
chat_tokens_output	INT	0	Không	Số token output.

chat_status	VARCHAR(20)	'active'	Có	Trạng thái: active, deleted, archived.
chat_visibility	VARCHAR(20)	'public'	Có	Quyền truy cập: private, public.
chat_created_at	TIMESTAMP WITH TIME ZONE	CURRENT_TIMESTAMP	Không	Thời điểm tạo chat.
chat_updated_at	TIMESTAMP WITH TIME ZONE	CURRENT_TIMESTAMP	Không	Thời điểm cập nhật gần nhất.

- Bảng tài liệu:

Bảng 3. 7. Bảng tài liệu

Trường	Kiểu dữ liệu	Mặc định	Bắt buộc	Giải thích
doc_id	CHAR(36) PRIMARY KEY	-	Có	Mã tài liệu (UUID).
doc_chat_id	CHAR(36) (FK)	-	Có	Thuộc cuộc chat, tham chiếu chat_histories.chat_id (ON DELETE CASCADE).
doc_file_path	TEXT	-	Có	Đường dẫn file gốc.
doc_ocr_text_path	TEXT	-	Có	Đường dẫn file OCR/text.
doc_title	VARCHAR(255)	-	Không	Tiêu đề tài liệu.
doc_status	VARCHAR(20)	'new'	Không	Trạng thái: new, routed, reviewed.
doc_created_at	TIMESTAMP WITH TIME ZONE	CURRENT_TIMESTAMP	Không	Thời điểm tạo.
doc_updated_at	TIMESTAMP WITH TIME ZONE	CURRENT_TIMESTAMP	Không	Thời điểm cập nhật gần nhất.

- Bảng file đính kèm tài liệu:

Bảng 3. 8. Bảng đính kèm tài liệu

Trường	Kiểu dữ liệu	Mặc định	Bắt buộc	Giải thích
attachment_id	CHAR(36) PRIMARY KEY	-	Có	Mã file đính kèm (UUID).
attachment_doc_id	CHAR(36) (FK)	-	Không	Tham chiếu documents.doc_id (ON DELETE CASCADE).
attachment_file_path	TEXT	-	Có	Đường dẫn file đính kèm.
attachment_description	TEXT	-	Không	Mô tả ngắn về file.
attachment_created_at	TIMESTAMP WITH TIME ZONE	CURRENT_TIMESTAMP	Không	Thời điểm tạo.

- Bảng tin nhắn chat:

Bảng 3. 9. Bảng tin nhắn chat

Trường	Kiểu dữ liệu	Mặc định	Bắt buộc	Giải thích
message_id	CHAR(36) PRIMARY KEY	-	Có	Mã tin nhắn (UUID).
message_chat_id	CHAR(36) (FK)	-	Có	Chat mà tin nhắn thuộc về, tham chiếu chat_histories.chat_id (ON DELETE CASCADE).
message_model_id	CHAR(36) (FK)	-	Có	Model trả lời, tham chiếu model_variants.model_id.
message_question	TEXT	-	Có	Nội dung câu hỏi từ người dùng.
message_ai_response	TEXT	-	Có	Nội dung phản hồi từ AI.

message_tokens_input	INT	0	Không	Token đầu vào.
message_tokens_output	INT	0	Không	Token đầu ra.
message_created_at	TIMESTAMP WITH TIME ZONE	CURRENT_TIMESTAMP	Không	Thời điểm tạo tin nhắn.
message_updated_at	TIMESTAMP WITH TIME ZONE	CURRENT_TIMESTAMP	Không	Thời điểm cập nhật gần nhất.

- Bảng phiên bản chat:

Bảng 3. 10. Bảng phiên bản chat

Trường	Kiểu dữ liệu	Mặc định	Bắt buộc	Giải thích
version_id	CHAR(36) PRIMARY KEY	-	Có	Mã phiên bản (UUID).
parent_chat_id	CHAR(36) (FK)	-	Có	Chat gốc, tham chiếu chat_histories.chat_id (ON DELETE CASCADE).
version_question	TEXT	-	Có	Nội dung câu hỏi.
version_ai_response	TEXT	-	Có	Nội dung trả lời AI.
version_created_at	TIMESTAMP WITH TIME ZONE	CURRENT_TIMESTAMP	Không	Thời điểm tạo phiên bản.

- Bảng phản hồi chat:

Bảng 3. 11. Bảng phản hồi chat

Trường	Kiểu dữ liệu	Mặc định	Bắt buộc	Giải thích
feedback_id	CHAR(36) PRIMARY KEY	-	Có	Mã phản hồi (UUID).

feedback_chat_id	CHAR(36) (FK)	-	Có	Chat liên quan, tham chiếu chat_histories.chat_id (ON DELETE CASCADE).
feedback_corrected_response	TEXT	-	Không	Câu trả lời được chỉnh sửa.
feedback_text	TEXT	-	Không	Nội dung góp ý/feedback.
feedback_created_at	TIMESTAMP WITH TIME ZONE	CURRENT_TIMESTAMP	Không	Thời điểm tạo phản hồi.

- Bảng email hẹn gửi:

Bảng 3. 12. Bảng email hẹn gửi

Trường	Kiểu dữ liệu	Mặc định	Bắt buộc	Giải thích
email_id	CHAR(36) PRIMARY KEY	-	Có	Mã email (UUID).
email_chat_id	CHAR(36) (FK)	-	Có	Chat liên quan, tham chiếu chat_histories.chat_id (ON DELETE CASCADE).
email_title	VARCHAR(255)	-	Có	Tiêu đề email.
email_recipient	VARCHAR(255)	-	Có	Người nhận email.
email_body	TEXT	-	Không	Nội dung email.
email_send_time	TIMESTAMP	-	Có	Thời điểm lên lịch gửi.
email_status	VARCHAR(20)	'scheduled'	Không	Trạng thái: scheduled, sent, failed.
email_created_at	TIMESTAMP WITH TIME ZONE	CURRENT_TIMESTAMP	Không	Thời điểm tạo.

email_updated_at	TIMESTAMP WITH TIME ZONE	CURRENT_TIMESTAMP	Không	Thời điểm cập nhật.
------------------	--------------------------	-------------------	-------	---------------------

- Bảng file đính kèm mail:

Bảng 3. 13. Bảng đính kèm Email

Trường	Kiểu dữ liệu	Mặc định	Bắt buộc	Giải thích
att_id	CHAR(36) PRIMARY KEY	-	Có	Mã file đính kèm (UUID).
att_email_id	CHAR(36) (FK)	-	Có	Tham chiếu scheduled_emails.email_id (ON DELETE CASCADE).
att_file_path	TEXT	-	Có	Đường dẫn file đính kèm.
att_description	TEXT	-	Không	Mô tả ngắn về file.
att_created_at	TIMESTAMP WITH TIME ZONE	CURRENT_TIMESTAMP	Không	Thời điểm tạo.

- Bảng liên kết email và phòng ban:

Bảng 3. 14. Bảng liên kế email và phòng ban

Trường	Kiểu dữ liệu	Mặc định	Bắt buộc	Giải thích
nm_email_id	CHAR(36) (FK)	-	Có	ID email, tham chiếu scheduled_emails.email_id (ON DELETE CASCADE).
nm_dept_id	CHAR(36) (FK)	-	Có	ID phòng ban, tham chiếu departments.dept_id (ON DELETE CASCADE).

nm_created_at	TIMESTAMP WITH TIME ZONE	CURRENT _TIMEST AMP	Không	Thời điểm tạo liên kết.
---------------	--------------------------------	---------------------------	-------	-------------------------

- Bảng danh sách công cụ:

Bảng 3. 15. Bảng danh sách công cụ

Trường	Kiểu dữ liệu	Mặc định	Bắt buộc	Giải thích
tool_id	CHAR(36) PRIMARY KEY	-	Có	Mã công cụ (UUID).
tool_name	VARCHAR(50) UNIQUE	-	Có	Tên công cụ.
tool_description	TEXT	-	Không	Mô tả công cụ.
tool_enabled	BOOLEAN	TRUE	Không	Trạng thái bật/tắt.
tool_access_scope	VARCHAR(20)	'all'	Không	Phạm vi truy cập: all, user, internal, admin.
tool_created_at	TIMESTAMP WITH TIME ZONE	CURRENT _TIMEST AMP	Không	Thời điểm tạo công cụ.

- Bảng công cụ đặc biệt trong chat:

Bảng 3. 16. Bảng công cụ đặc biệt trong chat

Trường	Kiểu dữ liệu	Mặc định	Bắt buộc	Giải thích
cf_id	CHAR(36) PRIMARY KEY	-	Có	Mã record (UUID).
cf_chat_id	CHAR(36) (FK)	-	Có	Chat liên quan, tham chiếu chat_histories.chat_id (ON DELETE CASCADE).
cf_tool_id	CHAR(36) (FK)	-	Không	Công cụ liên quan, tham chiếu tool_definitions.tool_id.
cf_type_name	VARCHAR(50)	-	Không	Loại: search, deep research,

				deep_research_pdf.
cf_metadata	JSONB	-	Không	Metadata của công cụ.
cf_created_at	TIMESTAMP WITH TIME ZONE	CURRENT_TIMESTAMP	Không	Thời điểm tạo.

- Bảng xác thực:

Bảng 3. 17. Bảng xác thực

Trường	Kiểu dữ liệu	Mặc định	Bắt buộc	Giải thích
vc_id	CHAR(36) PRIMARY KEY	-	Có	Mã verify (UUID).
vc_user_id	CHAR(36) (FK)	-	Có	Tham chiếu users.user_id.
vc_email	VARCHAR(255)	-	Có	Email xác thực.
vc_code	CHAR(6)	-	Có	Mã code ngắn (6 ký tự).
vc_attempts	INT	0	Không	Số lần nhập sai.
vc_max_attempt	INT	5	Không	Giới hạn số lần.
vc_send_count	INT	1	Không	Số lần đã gửi.
vc_expires_at	TIMESTAMP WITH TIME ZONE	-	Có	Hạn sử dụng mã.
vc_created_at	TIMESTAMP WITH TIME ZONE	CURRENT_TIMESTAMP	Không	Thời điểm tạo.

- Bảng token khôi phục mật khẩu:

Bảng 3. 18. Bảng token khôi phục mật khẩu

Trường	Kiểu dữ liệu	Mặc định	Bắt buộc	Giải thích
prt_id	CHAR(36) PRIMARY KEY	-	Có	Mã token (UUID).

prt_user_id	CHAR(36) (FK)	-	Có	Người dùng liên quan, tham chiếu users.user_id.
prt_token	VARCHAR(64) UNIQUE	-	Có	Token khôi phục duy nhất.
prt_expires_at	TIMESTAMP WITH TIME ZONE	-	Có	Thời điểm hết hạn.
prt_created_at	TIMESTAMP WITH TIME ZONE	CURRENT_TIMESTAMP	Không	Thời điểm tạo token.

- Bảng cấu hình hệ thống:

Bảng 3. 19. Bảng cấu hình hệ thống

Trường	Kiểu dữ liệu	Mặc định	Bắt buộc	Giải thích
system_id	CHAR(36) PRIMARY KEY	-	Có	Mã cấu hình (UUID).
system_register	BOOLEAN	FALSE	Không	Cho phép đăng ký mới.
system_login	BOOLEAN	TRUE	Không	Cho phép đăng nhập.
system_maintenance	BOOLEAN	FALSE	Không	Chế độ bảo trì.
setting_updated_at	TIMESTAMP WITH TIME ZONE	CURRENT_TIMESTAMP	Không	Thời điểm cập nhật.
updated_by_user_id	CHAR(36) (FK)	-	Không	Ai cập nhật, tham chiếu users.user_id.
system_domain_mode	VARCHAR(20)	'none'	Không	Chế độ tên miền: none, tvu, tvu_and_sttvu.
system_allowed_models	TEXT[]	-	Không	Danh sách model cho phép.
system_enabled_tools	TEXT[]	-	Không	Danh sách công cụ bật.

- Bảng liên kết model với công cụ chat:

Bảng 3. 20. Bảng liên kết công cụ chat

Trường	Kiểu dữ liệu	Mặc định	Bắt buộc	Giải thích
cf_id	CHAR(36) (FK)	-	Có	Công cụ, tham chiếu chat_features.cf_id (ON DELETE CASCADE).
model_id	CHAR(36) (FK)	-	Có	Model, tham chiếu model_variants.model_id (ON DELETE CASCADE).
link_created_at	TIMESTAMP WITH TIME ZONE	CURRENT_TIMESTAMP	Không	Thời điểm tạo liên kết.

- Bảng log lịch sử hoạt động hệ thống:

Bảng 3. 21. Bảng log lịch sử hệ thống

Trường	Kiểu dữ liệu	Mặc định	Bắt buộc	Giải thích
log_id	CHAR(36) PRIMARY KEY	-	Có	Mã log (UUID).
log_admin_id	CHAR(36) (FK)	-	Có	Quản trị viên, tham chiếu users.user_id.
log_action	VARCHAR(100)	-	Có	Hành động thực hiện.
log_target_table	VARCHAR(100)	-	Không	Bảng bị tác động.
log_target_id	CHAR(36)	-	Không	ID bản ghi bị tác động.
log_description	TEXT	-	Không	Mô tả chi tiết.
log_before	JSONB	-	Không	Dữ liệu trước thay đổi.
log_after	JSONB	-	Không	Dữ liệu sau thay đổi.

log_created_at	TIMESTAMP WITH TIME ZONE	CURRENT _TIMEST AMP	Không	Thời điểm tạo log.
----------------	--------------------------------	---------------------------	-------	--------------------

- Bảng thông báo hệ thống:

Bảng 3. 22. Bảng thông báo hệ thống

Trường	Kiểu dữ liệu	Mặc định	Bắt buộc	Giải thích
notify_id	CHAR(36) PRIMARY KEY	-	Có	Mã thông báo (UUID).
notify_content	TEXT	-	Có	Nội dung thông báo.
notify_visible	BOOLEAN	TRUE	Không	Trạng thái hiển thị.
notify_target_roles	TEXT[]	ARRAY['u ser','interna l','admin']	Không	Nhóm người nhận mặc định.
notify_created_by	CHAR(36) (FK)	-	Không	Người tạo, tham chiếu users.user_id.
notify_created_at	TIMESTAMP WITH TIME ZONE	CURRENT _TIMEST AMP	Không	Thời điểm tạo.

- Bảng người nhận hệ thống:

Bảng 3. 23. Bảng người nhận hệ thống

Trường	Kiểu dữ liệu	Mặc định	Bắt buộc	Giải thích
notify_id	CHAR(36) (FK)	-	Có	Tham chiếu system_notifications.notify_id (ON DELETE CASCADE).
user_id	CHAR(36) (FK)	-	Có	Người nhận, tham chiếu users.user_id (ON DELETE CASCADE).

read_at	TIMESTAMP WITH TIME ZONE	-	Không	Thời điểm đọc thông báo.
---------	--------------------------------	---	-------	-----------------------------

3.4. Xây dựng dataset trên Google Colab

3.4.1. Thu nhập dữ liệu thô và tải lên Google Drive

Bước đầu tiên là trích xuất thủ công các văn bản hành chính tiếng Việt từ hệ thống nguồn dưới dạng PDF, DOCX hoặc ảnh. Trong quá trình này ưu tiên bản quét rõ nét, đủ biên, hạn chế bóng mờ và méo ảnh để bảo đảm chất lượng nhận dạng ở các bước sau. Các phụ lục chỉ có bảng biểu hoặc ảnh khó đọc được tách riêng để không gây nhiễu cho OCR.

Tất cả tệp sau khi tải về được đổi tên theo một quy ước nhất quán nhằm tránh trùng lặp và lỗi ký tự. Ngay trong khâu nhập liệu tiến hành rà nhãn ban đầu. Những văn bản có trường gửi cho phòng ban rỗng, đều được tạm xếp vào nhóm chưa có nhãn. Các trường hợp mở đầu bằng từ “Chuyên” nhưng không ghi rõ đơn vị nhận cũng được xem là chưa có nhãn để xử lý bổ sung ở các bước kế tiếp.

Dữ liệu được tổ chức lại trên Google Drive theo một cấu trúc ổn định để phục vụ tự động hoá, bao gồm:

- Thư mục DocAIX/data/vanban_goc dùng để lưu trữ toàn bộ tệp gốc ở định dạng ban đầu.
- Thư mục DocAIX/data/vanban_dataset dùng để lưu các bảng dữ liệu trung gian và đầu ra.
- Thư mục DocAIX/data/vanban_logs dùng để lưu nhật ký nhập liệu cũng như các ghi chú liên quan đến nguồn và chất lượng tệp.

3.4.2. Thiết lập môi trường và khởi tạo dự án trên Google Colab

Sau khi đã thiết lập thư mục lưu trữ các tệp văn bản hành chính trên Google Drive, bước tiếp theo là khởi tạo môi trường làm việc trên Google Colab để thuận tiện cho quá trình xử lý dữ liệu. Google Colab cho phép chạy mã trực tuyến với tài nguyên tính toán miễn phí, đồng thời hỗ trợ kết nối trực tiếp với Google Drive nên toàn bộ dữ liệu ở bước trước có thể được gọi ra và thao tác ngay mà không cần tải xuống thủ công.

Trong môi trường Colab, tiến hành cài đặt các thư viện cần thiết và gắn kết tài khoản Google Drive với không gian làm việc. Sau khi kết nối thành công, toàn bộ cấu

trúc thư mục đã chuẩn bị trước đó (bao gồm `vanban_goc`, `vanban_dataset` và `vanban_logs`) được ánh xạ vào Colab để sẵn sàng cho các bước xử lý dữ liệu tự động.

Mục tiêu chính của giai đoạn này là lọc và chuẩn hóa nhãn `doc_action`. Thay vì giữ nguyên dữ liệu thô có nhiều cách ghi khác nhau, quy trình lọc được thiết kế thành ba lượt liên tiếp nhằm loại bỏ sai lệch, đảm bảo tính thống nhất và hình thành một bộ nhãn sạch cuối cùng.

Ở lượt lọc thứ nhất, hệ thống tập trung vào việc làm sạch cách ghi các cụm “Chuyển” đến phòng ban. Những biến thể về cách viết, ngắt dòng hay thừa từ đều được đưa về một biểu mẫu đồng nhất. Đến lượt lọc thứ hai, quá trình chuẩn hóa chính tả được áp dụng cho tên phòng ban, nhằm khắc phục các lỗi OCR thường gặp như thiếu dấu hoặc ghi sai chữ. Sau cùng, ở lượt thứ ba, kết quả được tinh chỉnh để chuẩn hóa cách ghi tên phòng ban theo đúng danh mục thống nhất, bảo đảm mỗi đơn vị chỉ còn một cách viết duy nhất.

Kết quả của ba lượt lọc này được lưu trực tiếp vào Google Sheet, vừa tiện cho việc theo dõi, vừa có thể rà soát và hiệu chỉnh thủ công khi cần thiết. Nhờ vậy, nhãn `doc_action` sau cùng đạt được sự đồng bộ và nhất quán, tạo nền tảng cho những bước phân tích và trực quan hóa ở các mục sau.

Song song với đó, dự kiến bộ dữ liệu hoàn chỉnh sẽ bao gồm năm cột chính: `doc_type`, `doc_issuer`, `doc_title`, `doc_content` và `doc_action`. Trong đó, bốn cột đầu được tự động trích xuất từ nội dung văn bản, còn `doc_action` là nhãn hành động được chuẩn hóa qua ba lượt lọc.

3.4.3. Thiết lập hàm OCR và kết nối API với model Gemma

Tiếp nối sau khi đã chuẩn hóa nhãn `doc_action` ở bước trước, giai đoạn này tập trung vào việc tạo ra nội dung văn bản số từ các file gốc và tách thành những cột dữ liệu cần thiết. Để làm được điều đó, cần kết hợp giữa hàm OCR và API của OpenRouter, trong đó model Gemma 12B đóng vai trò hỗ trợ phân tích ba cột đầu tiên.

Trước hết, hàm OCR được viết và chạy trong Google Colab để xử lý các file PDF hoặc ảnh đã lưu trên Google Drive. Hàm này sẽ nhận diện phần chữ trong văn bản và xuất ra kết quả ở dạng text thô, giúp loại bỏ những phần không cần thiết như nền ảnh hoặc đường viền. Đây là bước chuẩn bị quan trọng để dữ liệu có thể được tự động phân loại ở những thao tác tiếp theo.

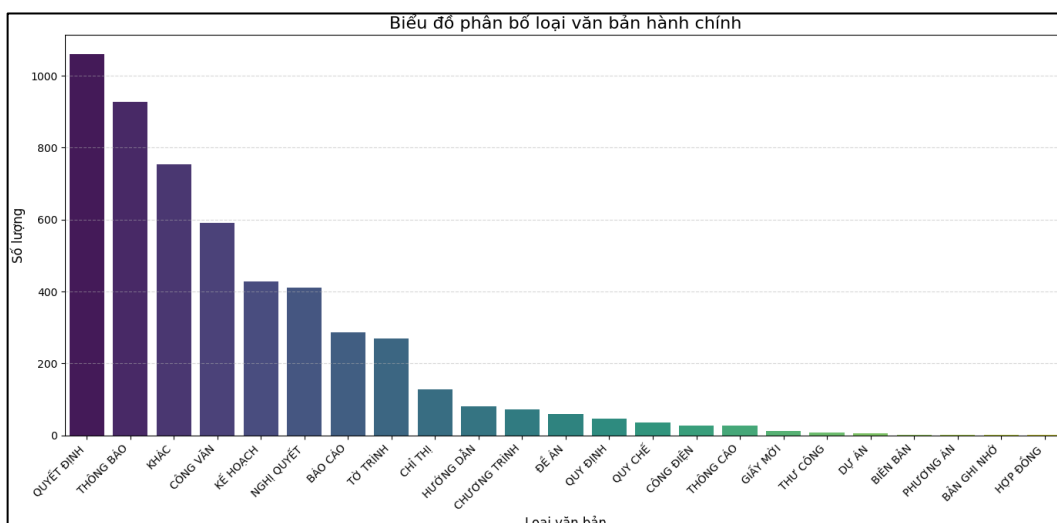
Sau khi có văn bản số, hệ thống được kết nối với API OpenRouter để điền tự động ba trường `doc_type`, `doc_issuer` và `doc_title`. Model Gemma 12B được gọi để phân tích ngữ cảnh và rút ra loại văn bản, cơ quan ban hành và tiêu đề.

Đối với cột `doc_content`, thay vì dùng API, một hàm Python riêng được thiết lập để cắt lấy hai trang đầu và một trang cuối của văn bản. Phần này sau đó được ghép lại thành một chuỗi nội dung hoàn chỉnh, đồng thời nếu xuất hiện cụm “Nơi nhận” thì được đưa xuống cuối để giữ đúng thể thức văn bản hành chính.

Nhờ kết hợp giữa OCR và API, kết quả cuối cùng của bước này là bộ dữ liệu có đủ bốn cột chính: `doc_type`, `doc_issuer`, `doc_title` và `doc_content`. Khi ghép với cột `doc_action` đã làm sạch trước đó, dataset đã đạt mức hoàn chỉnh và có thể dùng trực tiếp cho các bước phân tích ở các bước sau.

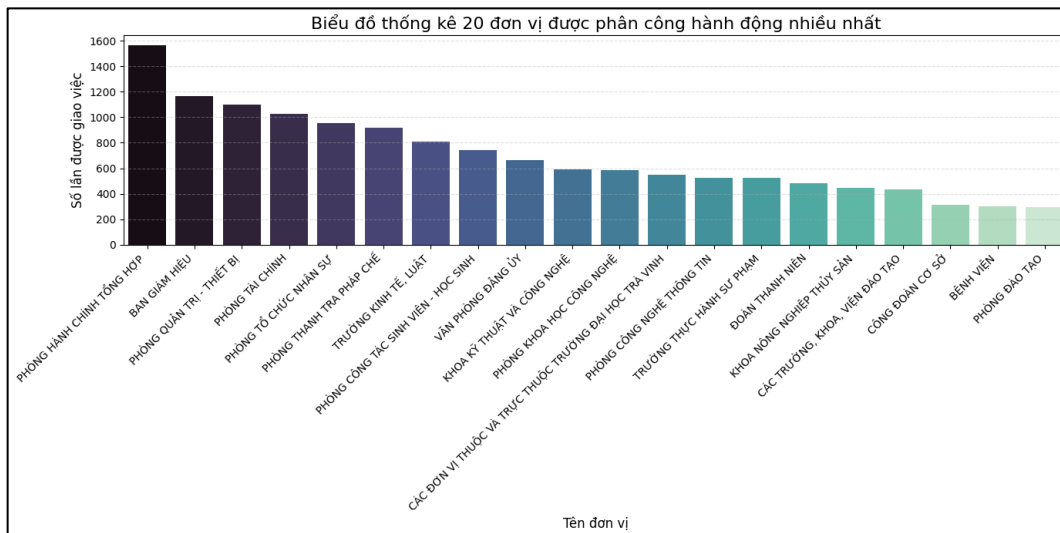
3.4.4. Thực quan hóa dữ liệu

Sau khi hoàn thiện bộ dataset với đầy đủ năm cột chính, bước tiếp theo là tiến hành thực quan hóa để quan sát bức tranh tổng thể về cấu trúc dữ liệu. Việc này không chỉ giúp nhận diện nhanh đặc điểm của tập văn bản hành chính mà còn hỗ trợ đánh giá rủi ro và đưa ra các hướng xử lý phù hợp trước khi huấn luyện mô hình.



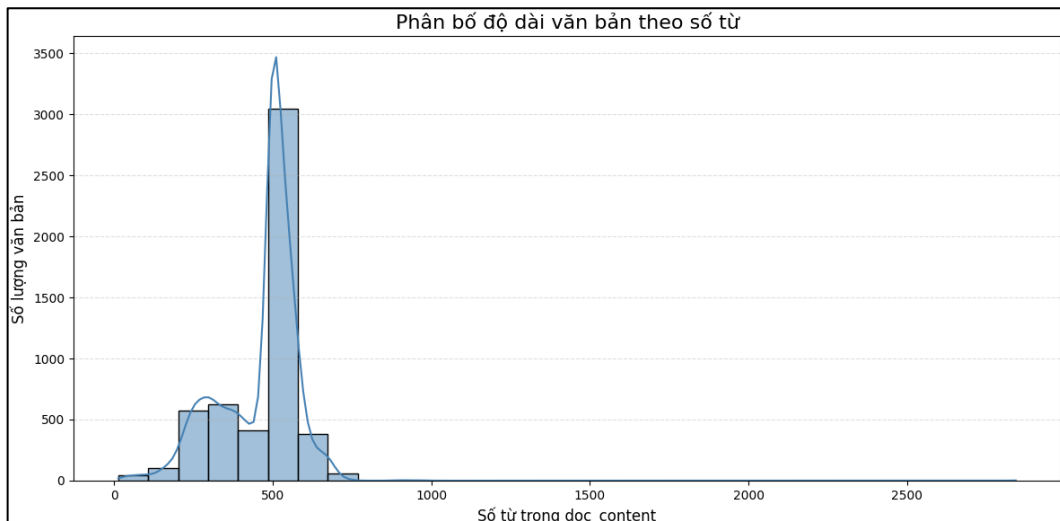
Hình 3. 3. Minh họa biểu đồ phân bố loại văn bản hành chính

Kết quả thống kê cho thấy ba nhóm văn bản hành chính phổ biến nhất là Quyết định, Thông báo, chiếm tỷ lệ áp đảo so với các loại khác. Các nhóm Kế hoạch, Nghị quyết, Báo cáo, Tờ trình xuất hiện ở mức trung bình, trong khi một số loại rời rạc được gom lại dưới nhãn “Khác” để tránh phân mảnh dữ liệu. Phân bố này gợi ý nguy cơ mất cân bằng lớp, do đó cần cân nhắc các kỹ thuật như lấy mẫu có trọng số hoặc tăng cường dữ liệu nhân tạo cho những lớp hiếm.



Hình 3. 4. Minh họa biểu đồ thống kê 20 đơn vị được phân công

Xét theo trường `doc_action`, dữ liệu phản ánh sự chênh lệch rõ rệt giữa các đơn vị được giao nhiệm vụ. Một số đầu mỗi nhận tần suất phân công cao vượt trội, đóng vai trò trung tâm trong quy trình xử lý văn bản, trong khi phần còn lại xuất hiện ít hơn đáng kể. Điều này có thể dẫn đến lệch nhãn khi huấn luyện, vì vậy việc chuẩn hóa tên đơn vị theo một từ điển thống nhất và đặt ngưỡng khống chế tần suất tối đa là cần thiết để giảm thiểu nhiễu.



Hình 3. 5. Phân bố độ dài văn bản trong tập dữ liệu

Về độ dài văn bản (tính theo số từ), phân bố tập trung nhiều quanh mức trung bình nhưng có xu hướng kéo dài về phía các văn bản dài hơn. Đặc điểm này phù hợp với tính chất văn bản hành chính: đủ dài để bao quát nội dung chính yếu nhưng không lan man. Kết quả này giúp xác định ngưỡng hợp lý khi chia khối văn bản (chunking), đồng thời điều chỉnh tham số tiền xử lý để mô hình có thể tiếp nhận dữ liệu ở dạng tối ưu hơn.

3.5. Xây dựng API phía máy chủ (Back-end)

3.5.1. Các API chức năng xác thực

Các API xác thực trong hệ thống được triển khai nhằm hỗ trợ nhiều hình thức đăng nhập khác nhau, bao gồm đăng nhập qua Google, Microsoft, tài khoản mật khẩu truyền thống, cũng như các quy trình xác minh email và quản lý phiên đăng nhập. Các API này đều tuân thủ cài đặt hệ thống (SystemSettings), trong đó quy định chế độ đăng nhập, đăng ký và giới hạn miền email.

Đăng nhập bằng Google OAuth: Quy trình bắt đầu bằng việc chuyển hướng người dùng sang trang xác thực Google với các thông số cần thiết. Sau khi người dùng chấp thuận, hệ thống nhận lại mã code, đổi lấy access token, rồi lấy thông tin tài khoản (email, tên hiển thị, ảnh đại diện). Email được kiểm tra miền theo cấu hình; nếu hợp lệ thì tiến hành liên kết với tài khoản sẵn có hoặc tạo mới khi được phép. Trong trường hợp tài khoản đã tồn tại, các thông tin hiển thị và timezone được cập nhật. Nếu người dùng bị khóa hoặc vi phạm miền, hệ thống trả về trang lỗi. Sau khi hợp lệ, hệ thống tạo cookie bảo mật và csrf-token để duy trì phiên đăng nhập an toàn.

Đăng nhập bằng Microsoft OAuth: Cách thức tương tự Google nhưng tích hợp với API của Microsoft Graph. Sau khi xác thực thành công, hệ thống lấy thông tin cơ bản và ảnh đại diện, rồi kiểm tra điều kiện miền và quyền đăng ký. Người dùng mới được tạo khi được phép, còn người dùng cũ sẽ được đồng bộ hóa thông tin (avatar, timezone, IP). Nếu tài khoản bị khóa, hệ thống từ chối truy cập. Sau bước xác thực, cookie và csrf-token cũng được thiết lập để duy trì phiên.

Xác thực bằng mật khẩu (Password-flow)P: Trong chế độ cho phép, hệ thống hỗ trợ đăng nhập và đăng ký bằng email/mật khẩu. Quy trình đăng nhập kiểm tra định dạng email, xác thực mật khẩu bằng thuật toán hash an toàn (bcrypt hoặc pbkdf2:sha256), đồng thời tự động nâng cấp mật khẩu cũ sang chuẩn mới nếu phát hiện. Với đăng ký, hệ thống kiểm tra điều kiện hợp lệ của tên người dùng và email, đồng thời tạo bản ghi tài khoản kèm mật khẩu hash bcrypt ngay từ đầu. Sau khi đăng ký, một mã xác minh email được gửi để hoàn tất kích hoạt tài khoản.

Quy trình quên mật khẩu và đặt lại mật khẩu: Người dùng có thể yêu cầu đặt lại mật khẩu nếu quên. Khi đó, hệ thống tạo token có thời hạn ngắn (5 phút), gửi kèm liên kết qua email. Người dùng dùng liên kết để nhập mật khẩu mới, hệ thống kiểm tra token còn hạn, chặn các tài khoản không hợp lệ, đồng thời đảm bảo mật khẩu mới đáp

ứng yêu cầu về độ dài, số và ký tự đặc biệt. Sau khi đặt lại, mật khẩu luôn được lưu trữ dưới dạng hash script.

Xác minh email: Hệ thống sử dụng mã xác minh gửi qua email để đảm bảo tính xác thực địa chỉ liên hệ. Người dùng cần nhập mã trong khoảng thời gian hiệu lực, số lần sai sẽ giảm dần cho đến khi bị khóa. API quản lý việc gửi lại mã xác minh với thời gian chờ tăng dần sau mỗi lần gửi. Khi xác minh thành công, trạng thái người dùng được cập nhật và mã xác minh được xóa khỏi cơ sở dữ liệu.

Đăng xuất: API đăng xuất thực hiện xóa toàn bộ cookie bảo mật và csrf-token, đảm bảo kết thúc phiên một cách an toàn.

3.5.2. Các API chức năng chat

Các API chat được thiết kế để hỗ trợ toàn bộ quy trình đối thoại, từ việc người dùng gửi văn bản và tệp đính kèm cho đến xử lý hậu kỳ bằng AI và phát sinh thông báo. Chúng tạo nên một kiến trúc kết hợp nhiều tầng: OCR để trích xuất văn bản, tìm kiếm dữ liệu nền (RAG), phân tích bằng mô hình ngôn ngữ (LLM JSON), chuẩn hóa kết quả, ghi nhận vào hệ thống, và cuối cùng là hỗ trợ gửi email thật sự khi có yêu cầu.

Xử lý văn bản đầu vào và tệp đính kèm: Người dùng có thể gửi tin nhắn kèm theo văn bản, tài liệu chính hoặc tệp đính kèm. Hệ thống nhận và lưu trữ các tệp này, đồng thời thực hiện OCR để trích xuất nội dung chữ từ file PDF hoặc hình ảnh. Khi không có file chính, hệ thống vẫn cho phép xử lý với các tệp đính kèm độc lập. Toàn bộ dữ liệu được ghi nhận vào cơ sở dữ liệu với thông tin chi tiết để quản lý lâu dài.

Luồng xử lý dữ liệu: Sau khi nhận văn bản, hệ thống kích hoạt tiến trình xử lý bất đồng bộ. Văn bản được cắt từ phần “Nơi nhận” để tập trung vào nội dung có ý nghĩa, sau đó so khớp với tập dữ liệu mẫu bằng kỹ thuật RAG (retrieval-augmented generation) để tìm các trường hợp tương tự. Tiếp đó, mô hình LLM được yêu cầu trả về đề xuất ở dạng JSON chuẩn, bao gồm các trường chính: chủ đề, nơi nhận, hành động cần thực hiện và lý do. Kết quả này được chuẩn hóa, ghi lại vào nhật ký (log) và đồng thời tạo bản ghi outbox kèm trích nội dung. Hệ thống còn sinh giao diện hiển thị dễ hiểu với chủ đề, danh sách nơi nhận và hành động dưới dạng gợi ý để người dùng theo dõi.

Chức năng gửi mail: Bên cạnh gợi ý xử lý văn bản, API còn hỗ trợ lệnh trực tiếp từ người dùng như “Gửi mail”. Khi nhận lệnh này, hệ thống lấy thông tin đã phân tích (chủ đề, nơi nhận, trích nội dung) và kết hợp với các tệp đính kèm để soạn email hoàn chỉnh. Thư được gửi đi qua SMTP thật đến địa chỉ đã cấu hình mặc định cho phòng

ban, kèm xác nhận trong giao diện chat rằng email đã được gửi thành công. Điều này cho phép chuyển đổi mượt mà giữa việc tham khảo đề xuất và hành động thực tế.

Các chức năng bổ trợ: Bên cạnh API chính để gửi và xử lý chat, hệ thống còn có các API hiển thị danh sách mô hình AI theo quyền người dùng, cập nhật thanh header và sidebar, quản lý lịch sử trò chuyện, cũng như đồng bộ thông báo hệ thống. Các API này đảm bảo trải nghiệm nhất quán, cho phép người dùng xem lại lịch sử, nhận cảnh báo mới và lựa chọn mô hình phù hợp với nhu cầu.

3.5.4. Các API hệ thống chung

Các API phía máy chủ đóng vai trò trung gian giữa giao diện người dùng và cơ sở dữ liệu, đồng thời chịu trách nhiệm điều phối các tác vụ AI như nhận dạng văn bản, tìm kiếm dữ liệu nền và sinh đề xuất xử lý. Nhóm API này được thiết kế theo nguyên tắc RESTful, vừa đảm bảo bảo mật bằng cơ chế cookie an toàn, vừa cho phép mở rộng linh hoạt cho nhiều loại nghiệp vụ. Có hai nhóm chính: API xác thực và API chat.

Các API chức năng xác thực: Nhóm API này chịu trách nhiệm kiểm tra và quản lý quyền truy cập của người dùng. Khi có yêu cầu từ phía client, hệ thống đọc cookie bảo mật để nhận diện danh tính, xác minh trạng thái hoạt động của tài khoản và quyết định quyền hạn truy cập. Nếu người dùng hợp lệ, API phản hồi thông tin xác thực cùng dữ liệu liên quan như danh sách mô hình AI được phép sử dụng. Ngược lại, các yêu cầu từ người dùng chưa đăng nhập hoặc không đủ quyền sẽ bị từ chối. Việc triển khai theo cách này đảm bảo tính an toàn và phân quyền chính xác cho toàn bộ hệ thống.

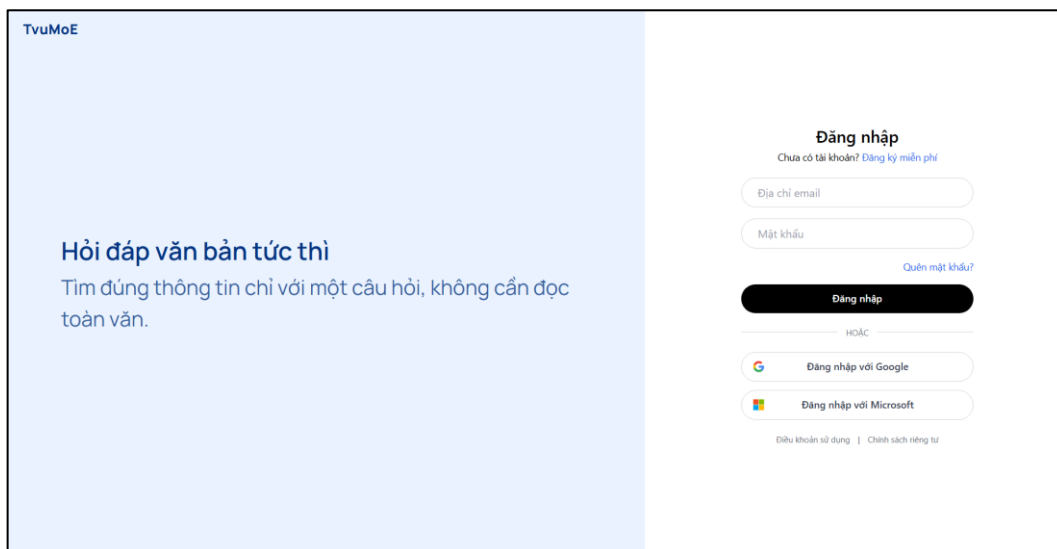
Các API chức năng chat: Các API chat hỗ trợ toàn bộ vòng đời xử lý văn bản. Người dùng có thể gửi tin nhắn kèm văn bản hoặc tài liệu đính kèm; hệ thống tự động lưu trữ, trích xuất chữ bằng OCR và khởi tạo bản ghi trong cơ sở dữ liệu. Sau đó, tiến trình bắt đồng bộ kết hợp nhiều bước: cắt văn bản từ phần “Nơi nhận”, tìm kiếm trong tập dữ liệu mẫu (RAG), yêu cầu mô hình AI sinh JSON đề xuất với các trường chính như chủ đề, nơi nhận và hành động.

Ngoài ra, API còn hỗ trợ lệnh gửi mail thực tế. Khi người dùng chọn “Gửi mail”, hệ thống lấy kết quả phân tích, kết hợp với các tệp gốc, và gửi email qua SMTP thật đến địa chỉ phòng ban mặc định. Người dùng được xác nhận ngay trong khung chat về việc email đã được gửi thành công. Bên cạnh đó, các API bổ trợ khác đảm bảo quản lý lịch sử hội thoại, đồng bộ thông báo hệ thống và hiển thị thanh công cụ (header, sidebar), giúp trải nghiệm chat luôn đầy đủ và liền mạch.

CHƯƠNG 4. KẾT QUẢ NGHIÊN CỨU

4.1. Giao diện đăng nhập

Giao diện đăng nhập được chia thành hai phần cân đối. Bên trái là khu vực giới thiệu ngắn gọn với khẩu hiệu định hướng, tạo ấn tượng ban đầu cho người dùng và giúp khẳng định mục tiêu ứng dụng. Phần nền màu xanh nhạt giúp giao diện trở nên nhẹ nhàng, dễ nhìn và tập trung sự chú ý vào nội dung chính.



Hình 4. 1. Minh họa giao diện đăng nhập

Bên phải là biểu mẫu đăng nhập gồm các trường nhập email và mật khẩu, nút đăng nhập và tùy chọn khôi phục mật khẩu khi cần. Ngoài phương thức đăng nhập thông thường, hệ thống còn hỗ trợ đăng nhập nhanh bằng tài khoản Google hoặc Microsoft, giúp người dùng tiết kiệm thời gian và linh hoạt hơn trong việc truy cập dịch vụ.

4.2. Giao diện đăng ký

Giao diện đăng ký tài khoản cũng được bố trí thành hai phần rõ ràng. Bên trái là phần giới thiệu với thông điệp ngắn gọn, nhấn mạnh mục tiêu ứng dụng trong việc hỗ trợ quản lý công vụ thông minh. Thiết kế tối giản với tông nền xanh nhạt giúp giữ sự tập trung và tạo cảm giác chuyên nghiệp.

Bên phải là biểu mẫu đăng ký, cho phép người dùng điền tên đăng nhập, địa chỉ email và mật khẩu để tạo tài khoản mới. Ngoài cách đăng ký thủ công, hệ thống hỗ trợ liên kết tài khoản Google hoặc Microsoft, giúp quá trình đăng ký nhanh chóng và thuận tiện hơn. Dưới cùng là liên kết điều khoản sử dụng và chính sách riêng tư, đảm bảo tính minh bạch và tuân thủ các quy định bảo mật thông tin.

Hình 4. 2. Minh họa giao diện đăng ký

4.3. Giao diện xác thực

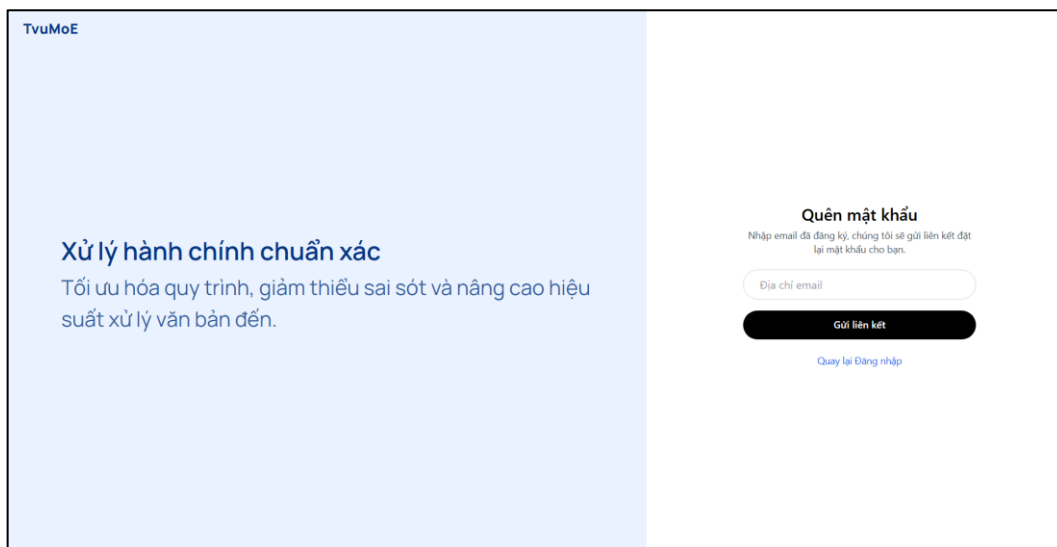
Ở bước xác minh, giao diện tiếp tục giữ bố cục quen thuộc hai phần cân đối. Bên trái là phần giới thiệu ngắn gọn về tính năng “Tiếp nhận văn bản tự động”, nhấn mạnh khả năng phân loại, trích xuất và luân chuyển văn bản nhanh chóng, chính xác. Cách trình bày này vừa quảng bá chức năng hệ thống, vừa giúp giao diện bớt đơn điệu.

Hình 4. 3. Minh họa giao diện nhận mã xác thực

Bên phải là khu vực chính dành cho người dùng, nơi họ được yêu cầu nhập mã xác minh gồm 6 chữ số đã được gửi về email. Thiết kế hiển thị rõ ràng địa chỉ email nhận mã, ô nhập mã và nút xác nhận, giúp thao tác trực quan. Ngoài ra, hệ thống cung cấp tính năng gửi lại mã với bộ đếm thời gian, đảm bảo người dùng có thể hoàn tất quá trình xác minh ngay cả khi không nhận được mã ngay từ lần đầu.

4.4. Giao diện quên mật khẩu

Giao diện quên mật khẩu được thiết kế đơn giản, trực quan với hai phần rõ rệt. Bên trái hiển thị thông điệp “Xử lý hành chính chuẩn xác”, nhấn mạnh vai trò của hệ thống trong việc tối ưu hóa quy trình, giảm thiểu sai sót và nâng cao hiệu quả xử lý văn bản. Đây là cách lồng ghép quảng bá tính năng hệ thống ngay cả trong các bước thao tác kỹ thuật của người dùng.



Hình 4. 4. Minh họa giao diện quên mật khẩu

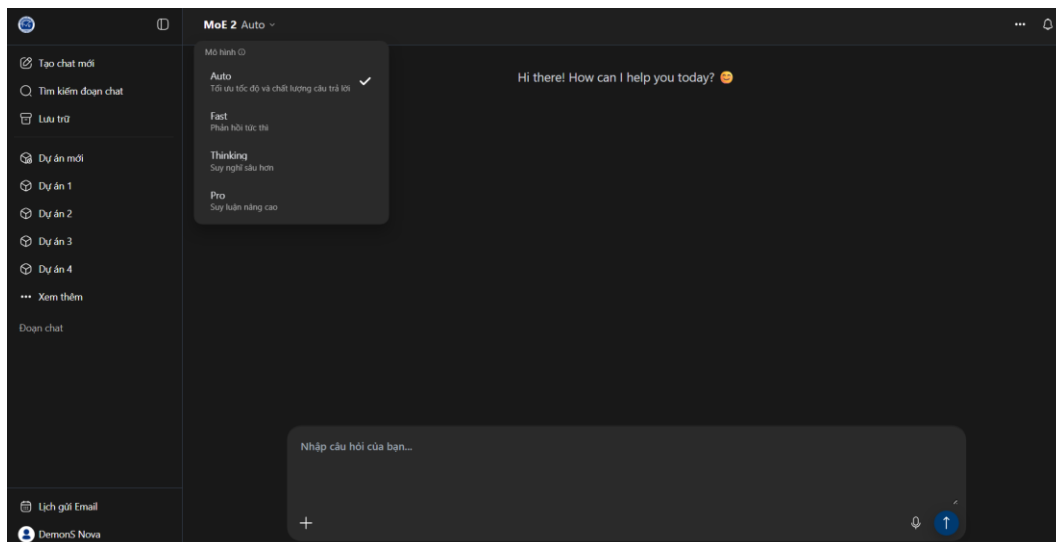
Phần bên phải là khu vực chức năng, nơi người dùng có thể nhập địa chỉ email đã đăng ký để nhận liên kết đặt lại mật khẩu. Bố cục gồm ô nhập email, nút “Gửi liên kết” nổi bật và đường dẫn quay lại trang đăng nhập, tất cả đều được sắp xếp gọn gàng, đảm bảo thao tác nhanh chóng và dễ dàng. Thiết kế này vừa đáp ứng yêu cầu bảo mật, vừa tạo trải nghiệm thuận tiện cho người dùng khi cần khôi phục tài khoản.

4.5. Giao diện chat

Màn hình chat được xem là trung tâm của hệ thống, nơi người dùng trực tiếp trao đổi và tương tác với mô hình AI. Bố cục tổng thể được chia thành ba phần rõ ràng: thanh bên trái hiển thị danh sách các dự án và các đoạn hội thoại đã lưu, khu vực trung tâm là khung hội thoại chính, còn phía dưới là ô nhập liệu để người dùng gửi câu hỏi hoặc văn bản. Giao diện được thiết kế theo tông nền tối hiện đại, vừa tạo cảm giác tập trung vừa giúp giảm mỏi mắt khi sử dụng trong thời gian dài.

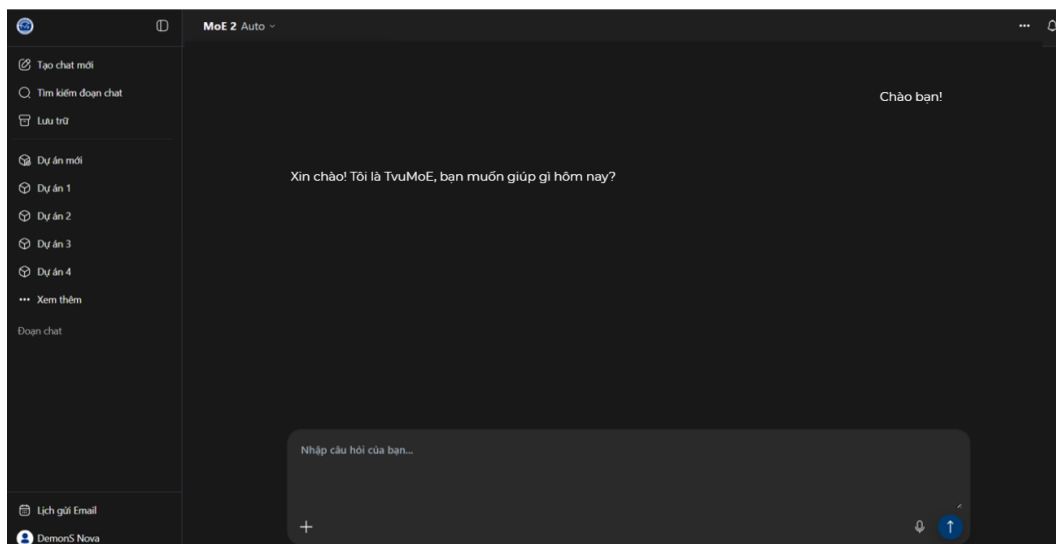
Một trong những điểm nổi bật của giao diện này là khả năng lựa chọn mô hình AI phù hợp với từng nhu cầu. Người dùng có thể chuyển đổi nhanh giữa các chế độ như Auto (tự động tối ưu tốc độ và chất lượng), Fast (phản hồi tức thì), Thinking (suy nghĩ chuyên sâu) và Pro (lý luận nâng cao). Sự đa dạng này giúp hệ thống đáp ứng linh hoạt,

từ những truy vấn đơn giản cho đến các yêu cầu phân tích phức tạp, nâng cao trải nghiệm tổng thể cho người dùng..



Hình 4. 5. Minh họa giao diện chat

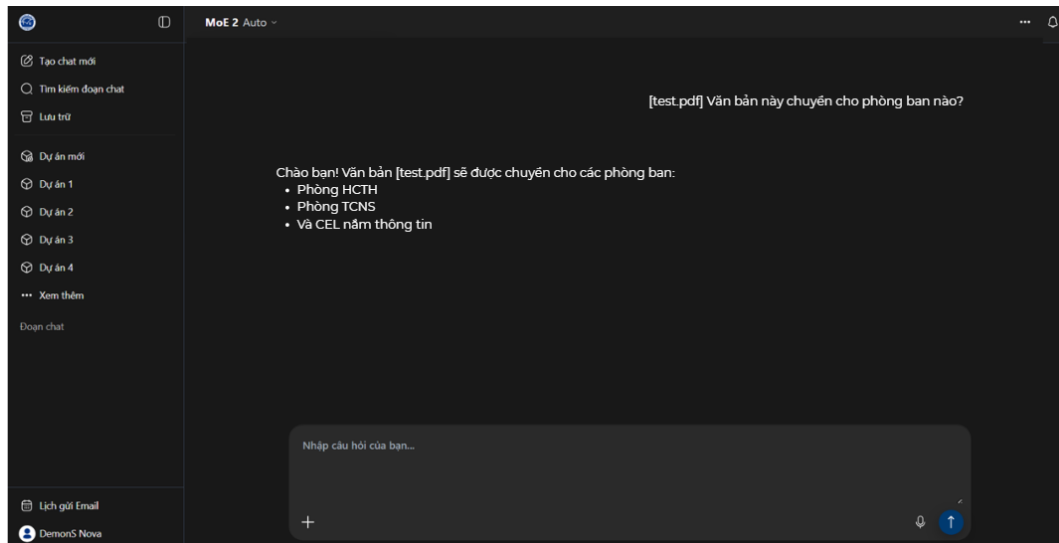
Đặc biệt, ở chế độ Auto, hệ thống không chỉ phản hồi tức thì mà còn có khả năng tự động đánh giá mức độ khó của câu hỏi để lựa chọn chế độ Thinking phù hợp trong ba mức: Low, Medium hoặc High. Với các câu hỏi đơn giản, hệ thống chọn mức Low để đưa ra câu trả lời nhanh gọn; trong khi với các câu hỏi đòi hỏi suy luận chi tiết hơn, hệ thống sẽ tự động chuyển sang mức Medium hoặc High để dành nhiều thời gian xử lý, từ đó cho ra câu trả lời có độ chính xác và chất lượng cao hơn.



Hình 4.6. Minh họa cách model lựa chọn chế độ trả lời

Khi người dùng tải lên một tệp PDF, hệ thống sẽ tự động nhận toàn bộ nội dung text thô của văn bản, sau đó áp dụng cơ chế RAG để cắt tách và chọn ra những phần quan trọng. Toàn bộ dữ liệu này được điều phối bởi MCP nhằm tạo thành một prompt

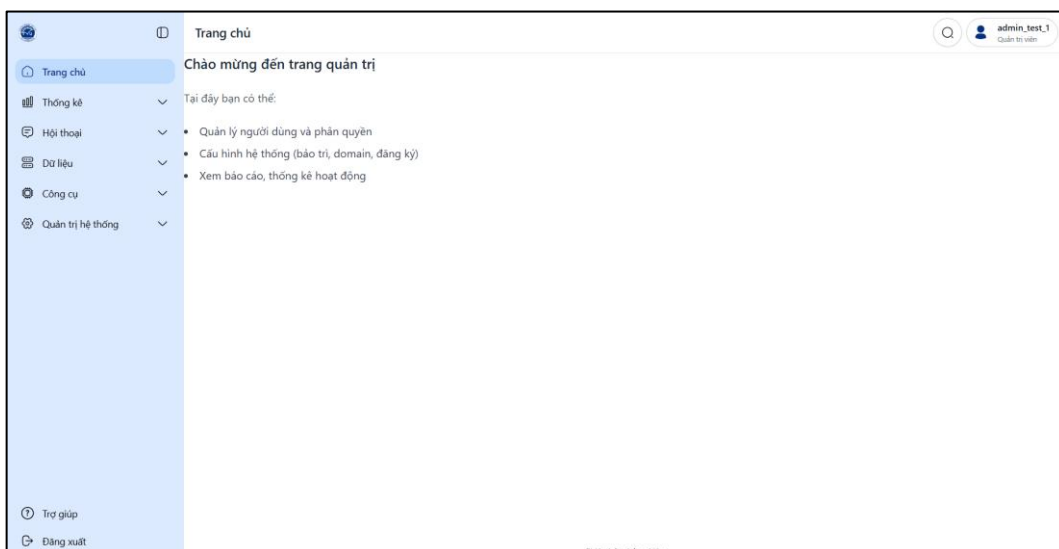
hoàn chỉnh, giúp mô hình hiểu rõ ngữ cảnh và lựa chọn cách trả lời hợp lý nhất. Cách làm này vừa bảo đảm tính đầy đủ của thông tin, vừa giữ được độ chính xác khi phản hồi.



Hình 4.7. Minh họa cách model trả lời

4.5. Giao diện trang quản trị

Giao diện quản trị đóng vai trò trung tâm trong việc kiểm soát và điều phối toàn bộ hoạt động của hệ thống. Tại đây, quản trị viên có thể dễ dàng theo dõi tình trạng hoạt động, quản lý người dùng, phân quyền truy cập cũng như thực hiện các cấu hình cần thiết cho hệ thống. Bố cục được thiết kế trực quan với thanh điều hướng bên trái, cho phép truy cập nhanh vào các nhóm chức năng như thống kê, hội thoại, dữ liệu, công cụ và quản trị hệ thống. Cách sắp xếp này giúp tối ưu trải nghiệm sử dụng, giảm thiểu thao tác thừa và hỗ trợ quản trị viên thao tác nhanh chóng.



Hình 4. 8. Minh họa giao diện trang quản trị

Ngoài ra, giao diện quản trị còn hỗ trợ hiển thị báo cáo và thống kê chi tiết, giúp nắm bắt kịp thời tình trạng vận hành cũng như hiệu quả sử dụng hệ thống. Các thông tin như lịch sử hoạt động, cấu hình hệ thống, báo cáo người dùng hay thống kê tương tác đều có thể được truy cập ngay từ giao diện này. Nhờ vậy, quản trị viên không chỉ kiểm soát được toàn bộ quy trình mà còn có thể chủ động điều chỉnh để bảo đảm hệ thống vận hành ổn định, an toàn và phù hợp với nhu cầu thực tế.

CHƯƠNG 5. KẾT LUẬN

5.1. Kết luận

Hệ thống được xây dựng đã đáp ứng mục tiêu ban đầu là hỗ trợ quản lý, phân loại và xử lý văn bản hành chính một cách tự động, thống nhất và hiệu quả. Từ khâu thu thập, nhận dạng OCR, tiền xử lý dữ liệu cho đến việc triển khai API và giao diện người dùng, toàn bộ quy trình đã tạo thành một nền tảng hoàn chỉnh, có khả năng phục vụ cho nhiều tình huống thực tiễn. Kết quả nghiên cứu cho thấy việc ứng dụng AI và các công nghệ hỗ trợ đã góp phần giảm tải công việc thủ công, nâng cao độ chính xác và tiết kiệm thời gian xử lý.

Bên cạnh giá trị thực tiễn, hệ thống cũng chứng minh được tính khả thi trong việc tích hợp công nghệ trí tuệ nhân tạo vào quy trình quản lý công vụ. Các chức năng từ xác thực, chat, quản trị đến hỗ trợ tìm kiếm thông tin đều hoạt động ổn định, góp phần tạo nên một công cụ hữu ích cho nghiên cứu cũng như triển khai thực tế trong tổ chức.

5.2. Hướng phát triển

Trong tương lai, hệ thống có thể được mở rộng để hỗ trợ thêm nhiều loại văn bản và tình huống xử lý phức tạp hơn, đồng thời tăng cường khả năng học hỏi liên tục từ dữ liệu mới. Việc áp dụng các kỹ thuật xử lý ngôn ngữ tự nhiên tiên tiến, kết hợp với cơ chế tối ưu hóa mô hình, sẽ giúp nâng cao chất lượng phân tích và khả năng phản hồi chính xác.

Ngoài ra, hướng phát triển quan trọng là cải thiện trải nghiệm người dùng và tăng cường khả năng tích hợp với các nền tảng khác. Hệ thống có thể được bổ sung các chức năng gợi ý thông minh, thống kê nâng cao, cũng như hỗ trợ đa ngôn ngữ để mở rộng phạm vi ứng dụng. Việc triển khai trên môi trường điện toán đám mây cũng sẽ giúp nâng cao tính linh hoạt, khả năng mở rộng và độ tin cậy trong khai thác lâu dài.

DANH MỤC TÀI LIỆU THAM KHẢO

- [1] OpenAI, “Introducing gpt-oss: open-weight models for reasoning,” Aug. 5, 2025.
- [2] OpenAI, “gpt-oss-120b & gpt-oss-20b Model Card,” OpenAI, Aug. 5, 2025.
- [3] V. Samuel, “How Retrieve-Augment-Generate (RAG) improves generative AI models,” TechFinitive, Jul. 09, 2024.
- [4] Anthropic, “Introducing the Model Context Protocol,” Anthropic News, Nov. 25, 2024.
- [5] IBM, “What is tool calling? Key enabler of agentic AI,” IBM Think Blog, Sep. 2025.
- [6] Dominik Kundel, “OpenAI Harmony Response Format – structured messages for conversations,” OpenAI Cookbook, Aug. 5, 2025.
- [7] ChemiCloud, “What Is HTML? – HyperText Markup Language definition,” ChemiCloud Glossary, 2023.
- [8] SEO.ai, “Cascading Style Sheets (CSS) – presentation of HTML documents,” SEO.AI Knowledge Base, 2023.
- [9] MDN Web Docs, “What is JavaScript? – Explanation and uses,” developer.mozilla.org, 2023.
- [10] OpenAI, “Tailwind CSS: a utility-first CSS framework for rapid UI development,” OpenAI Tutorials, 2024.
- [11] U-M ICT, “Python – an interpreted, high-level programming language,” University of Michigan Software Directory, 2025.
- [12] Martin Fowler, “Function calling using LLMs – integrating tools via JSON,” martinowler.com, Mar. 2025.
- [13] Intel Capital, “Runpod – the cloud built for AI (platform overview),” intelcapital.com, 2023.
- [14] Hugging Face, “The AI community building the future – platform with 1M+ models,” huggingface.co, 2025.
- [15] Google, “Google Colab – Hosted Jupyter notebooks with free GPUs,” research.google.com/colaboratory, 2024.
- [16] Google Developers, “Google Apps Script overview – rapid development platform for Workspace,” developers.google.com, 2025.