TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN - ĐHQG.HN

KHOA TOÁN - CƠ - TIN HỌC

VNU UNIVERSITY OF SCIENCE

BÁO CÁO BÀI TẬP LỚN

Môn học : Nhập môn trí tuệ nhân tạo (MAT1206E)

Học kỳ 1 - Năm học: 2025 - 2026

ĐỀ TÀI: QUẢN LÝ CÔNG VIỆC CÁ NHÂN THÔNG MINH

Danh sách sinh viên:

Họ và tên	MSSV	GitHub
Nguyễn Quang Anh	23001825	23001825_NguyenQuangAnh
Nguyễn Thái Dương	23001859	Duong5326
Hoàng Mạnh Duy	23001852	duy-301205
Nguyễn Nhật Đan	23001861	ChimSe08
Bàn Khánh Duy	23001851	Hatou1

Hà Nội, ngày 30 tháng 11 năm 2025

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN - ĐHQG.HN

KHOA TOÁN - CƠ - TIN HỌC

BẢNG PHÂN CÔNG CÔNG VIỆC CỦA CÁC THÀNH VIÊN

Nhóm sinh viên: Nhóm 36

Họ và tên	Đóng góp của từng thành viên
Hoàng Mạnh Duy	Làm backend spring boot, chỉnh sửa báo cáo
Nguyễn Quang Anh	Làm database , báo cáo, slide
Nguyễn Thái Dương	Làm frontend, lên ý tưởng
Nguyễn Nhật Đan	Làm backend FastApi pipeline 2, báo cáo, slide
Bàn Khánh Duy	Làm backend FastApi pipeline 1, báo cáo, slide

Mục lục nội dung

CHƯƠNG 1: GIỚI THIỆU 1
1.1. Lý do chọn đề tài 1
1.2. Mục tiêu đề tài 1
1.2.1. Mục tiêu 1
1.2.2. Đối tượng nghiên cứu 2
1.3. Tóm tắt dự án 2
1.4. Các tính năng chính của hệ thống 2
1.5. Ý nghĩa thực tiễn 4
CHƯƠNG 2: PHƯƠNG PHÁP VÀ TRIỂN KHAI 5
2.1. Phương pháp xử lý ngôn ngữ tự nhiên (NLP) 5
2.1.1. Pipeline 1 – Phân tích khi tạo Task thủ công (Manual Task Analysis Pipeline) 5
2.1.2. Pipeline 2 – Phân tích khi tạo task qua Chat Assistant bằng câu nói tự nhiên 8
2.2. Kiến trúc hệ thống 13
2.2.1. Backend Spring Boot 13
2.2.2. FastAPI AI Service 14
2.2.3. Frontend React 16
2.3. Thiết kế cơ sở dữ liệu 21
2.3.1. Bảng users 21
2.3.2. Bảng categories 22
2.3.3. Bảng tasks 22
2.3.4. Bảng ai_analysis 22
2.3.5. Bảng notifications 23
2.3.6. Bảng đặc biệt: invalid_token 23
2.4. Dataset huấn luyện mô hình AI 24
2.4.1. Quy mô và đặc điểm dataset 24
2.4.2. Cơ chế gán nhãn 24
2.4.3. Cân bằng dữ liệu 24
2.4.4. Quy trình tiền xử lý 24
2.4.5. Huấn luyện mô hình 25
2.4.6. Kết quả mô hình 25
CHƯƠNG 3: KẾT QUẢ VÀ PHÂN TÍCH 25
3.1. Kết quả hệ thống 25
3.1.1. Pipeline 1 – Kết quả phân tích khi tạo task thủ công 25
3.1.2. Pipeline 2 – Kết quả phân tích khi tạo task bằng ngôn ngữ tự nhiên (Chat Assistant) 27
3.1.3. Dashboard – Bảng điều khiển trạng thái công việc 31

3.1.4. Calendar View – Lịch làm việc 31
3.1.5. Overview – Phân tích hiệu suất làm việc 32
3.1.6. Notifications – Nhắc nhở thông minh 33
3.1.7. Hệ thống Authentication – Đăng ký, đăng nhập 34
3.2. Đánh giá hiệu năng hệ thống 35
3.3. Phân tích hạn chế của hệ thống 36
CHƯƠNG 4: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN 37
4.1. Kết luận 37
4.2. Hướng phát triển 38
4.3. Kết lời 48
TÀI LIỆU THAM KHẢO VÀ PHỤ LỤC 49

Mục lục hình ảnh

Hình 1: Giao diện TaskCard trong hệ thống TaskAI 17
Hình 2: Giao diện Dashboard trong hệ thống TaskAI 17
Hình 3: Giao diện Calender View trong hệ thống TaskAI 18
Hình 4: Giao diện Overview trong hệ thống TaskAI 18
Hình 5: Giao diện Chat Assistant trong hệ thống TaskAI 19
Hình 6: Giao diện Notification trong hệ thống TaskAI 19
Hình 7: Sơ đồ quan hệ cơ sở dữ liệu (ORM) của hệ thống TaskAI 21
Hình 8: Confusion Matrix – Kết quả phân loại Category (Pipeline 1) 26
Hình 9: Kết quả dự đoán Urgency 27
Hình 10: Kết quả dự đoán Importance 27
Hình 11: Kết quả dự đoán DurationMinutes 27
Hình 12: Confusion Matrix – Kết quả phân loại Category (Pipeline 2) 28
Hình 13:Phân bố sai số dự đoán mức độ quan trọng và thời gian thực hiện (Importance Error Distribution & Duration Error Distribution) 29
Hình 14: Giao diện lịch tuần (Weekly Calendar View) 31
Hình 15: Giao diện lịch tháng (Monthly Calendar View) 32
Hình 16: Overview phân tích hiệu suất làm việc 33
Hình 17: Dashboard với cửa sổ thông báo deadline và cảnh báo quá hạn 34
Hình 18: Giao diện login, register 35

Lời cảm ơn

Chúng em xin gửi lời cảm ơn chân thành đến quý thầy cô, các anh chị trợ giảng đã luôn tận tình giảng dạy, định hướng và tạo điều kiện thuận lợi để chúng em có thể triển khai và hoàn thiện dự án **TaskAI**. Những kiến thức về lập trình, xử lý ngôn ngữ tự nhiên, học máy, cũng như phương pháp làm việc khoa học mà thầy cô truyền đạt đã trở thành nền tảng quan trọng giúp chúng em phát triển hệ thống một cách hiệu quả và đúng hướng.

Chúng em cũng xin gửi lời cảm ơn đến bạn bè và những người đã tham gia đóng góp ý kiến, trải nghiệm thử và phản hồi hệ thống. Những góp ý đó giúp chúng em phát hiện các vấn đề thực tế, hoàn thiện tính năng và nâng cao chất lượng sản phẩm.

Dự án **TaskAI** là kết quả của sự nỗ lực, tinh thần học hỏi và làm việc nhóm nghiêm túc. Chúng em trân trọng mọi sự hỗ trợ trong suốt quá trình thực hiện, và xin gửi lời cảm ơn đến tất cả những ai đã đồng hành, động viên và tạo động lực để nhóm hoàn thành dự án.

Xin chân thành cảm ơn!

CHƯƠNG 1: GIỚI THIỆU

1.1. Lý do chọn đề tài

Trong kỷ nguyên công nghệ 4.0, khi số lượng ngành nghề và khối lượng công việc ngày càng gia tăng, nhu cầu về một công cụ quản lý công việc và thời gian hiệu quả trở nên cấp thiết hơn bao giờ hết. Một hệ thống hỗ trợ không chỉ giúp sắp xếp công việc một cách khoa học mà còn tạo động lực, cảm hứng cho người dùng trong quá trình làm việc.

Đề tài xây dựng một website quản lý công việc cá nhân tự động áp dụng trí tuệ nhân tạo được lựa chọn dựa trên các lý do sau:

Nhu cầu quản lý công việc hiệu quả ngày càng tăng: Trong thời ại công nghệ 4.0, con người phải xử lý một lượng lớn công việc mỗi ngày. Tuy nhiên, không phải ai cũng có phương pháp tổ chức khoa học để tối ưu hóa thời gian và năng suất làm việc. Một công cụ quản lý công việc thông minh, trực quan sẽ giúp người dùng sắp xếp công việc hiệu quả hơn.

Hạn chế của các phương pháp truyền thống: Các phương pháp như ghi chú trên giấy hay lập bảng biểu trên máy tính tuy có thể hỗ trợ quản lý công việc nhưng lại thiếu linh hoạt, không có sự tùy chỉnh theo nhu cầu cá nhân và không tận dụng được các công nghệ hiện đại để tối ưu hóa quá trình làm việc.

Thiếu các nền tảng tạo động lực cho người dùng: Nhiều ứng dụng quản lý công việc hiện nay chỉ tập trung vào việc sắp xếp và theo dõi nhiệm vụ mà chưa khai thác tốt yếu tố tâm lý để giúp người dùng duy trì động lực làm việc lâu dài. Lấy cảm hứng từ Duolingo – một nền tảng học tập thành công nhờ cơ chế tạo động lực, đề tài hướng đến việc xây dựng một hệ thống quản lý công việc không chỉ hiệu quả mà còn giúp người dùng có cảm hứng làm việc.

Ứng dụng AI để nâng cao trải nghiệm người dùng: AI đang trở thành xu hướng quan trọng trong nhiều lĩnh vực, nhưng các nền tảng quản lý công việc tại Việt Nam vẫn chưa khai thác tốt công nghệ này. Việc tích hợp chatbot AI có thể giúp người dùng quản lý công việc nhanh chóng hơn.

Việc nghiên cứu đề tài không chỉ giải quyết nhu cầu quản lý công việc một cách khoa học mà còn tận dụng công nghệ AI và các cơ chế tạo động lực để mang đến một giải pháp hiệu quả, dễ sử dụng và truyền cảm hứng cho người dùng.

Xuất phát từ thực tế đó, đề tài này tập trung nghiên cứu và phát triển một website quản lý công việc hiện đại, kết hợp giữa giao diện trực quan, hệ thống tạo động lực cho người dùng và chatbot AI giúp người dùng được trải nghiệm một không gian quản lý công việc đầy cảm hứng và đáng tin cậy.

1.2. Mục tiêu đề tài

1.2.1. Mục tiêu

Đối tượng: Người dùng tại Việt Nam

Phạm vi: Việt Nam (có thể mở rộng trong tương lai)

Ý tưởng: Phát triển một website quản lý công việc hiện đại, tích hợp giao diện trực quan, hệ thống tạo cảm hứng làm việc và chatbot AI hỗ trợ việc quản lý. Hệ thống này không chỉ giúp người dùng sắp xếp và theo dõi công việc một cách hiệu quả mà còn mang đến trải nghiệm mượt mà, hứng khởi, góp phần nâng cao năng suất làm việc.

1.2.2. Đối tượng nghiên cứu

Các nền tảng quản lý công việc phổ biến như Trello, Notion, …(hệ thống tạo động lực),... nhằm phân tích cách tổ chức task, quản lý tiến độ và tối ưu hóa trải nghiệm người dùng.

Nghiên cứu tích hợp chatbot AI xử lý ngôn ngữ tự nhiên (NLP) để hiểu và trả lời các câu hỏi của người dùng khi họ muốn sắp xếp công việc vào lịch làm việc. Mặt khác, nghiên cứu tích hợp thêm chatbot dựa trên quy tắc (Rules-based) để khi người dùng muốn xác định công việc nằm ở vị trí ngày tháng năm nào trong khung làm việc thì sẽ dễ dàng xác định.

Các nguyên tắc quản lý thời gian khoa học trong các cuốn sách như Deep Work, Atomic Habits, Pomodoro, Eisenhower Matrix,... để tạo nền tảng lý thuyết vững chắc cho hệ thống.

Các phương pháp thiết kế giao diện trực quan và thân thiện, nhằm tạo động lực, tăng trải nghiệm và giúp người dùng quản lý công việc hiệu quả.

1.3. Tóm tắt dự án

Dự án **TaskAI** được xây dựng với mục tiêu tạo ra một hệ thống quản lý công việc thông minh, giúp người dùng lập kế hoạch hằng ngày hiệu quả hơn, giảm thời gian suy nghĩ và tự động hóa các thao tác thủ công. Hệ thống tích hợp trí tuệ nhân tạo để phân tích ngôn ngữ tự nhiên, gợi ý thông tin cho task và tối ưu hóa việc sắp xếp công việc.

TaskAI bao gồm ba thành phần chính:

- **Backend Spring Boot** xử lý logic nghiệp vụ, lưu trữ dữ liệu và quản lý người dùng.
- **FastAPI AI Service** chứa mô hình AI phân tích nội dung và dự đoán các thuộc tính của task.
- **Frontend React** cung cấp giao diện trực quan, hỗ trợ thao tác tạo và xem task dễ dàng.

Hệ thống hỗ trợ cả người dùng phổ thông (nhập task thủ công) và người dùng nâng cao (tạo task bằng ngôn ngữ tự nhiên với AI), hướng đến mục tiêu tạo ra một công cụ quản lý công việc tự động, thông minh và thân thiện

1.4. Các tính năng chính của hệ thống

- **Đăng ký – Đăng nhập (Authentication)**

Hệ thống hỗ trợ:

- Đăng ký tài khoản mới
- Đăng nhập bằng email + mật khẩu
- Sinh Access Token & Refresh Token
- Lưu phiên người dùng an toàn

Từ đó bảo vệ dữ liệu cá nhân và đảm bảo mỗi người dùng có không gian làm việc riêng.

- **Tạo task thủ công truyền thống**

Người dùng nhập đầy đủ thông tin như tên task, mô tả, deadline, thời gian làm, độ ưu tiên… Sau đó, AI phân tích nội dung để:

- Gợi ý category
- Dự đoán urgency/importance
- Dự đoán duration
- Tính priorityScore
- Hỗ trợ người dùng chọn thông tin hợp lý hơn

Giúp tiết kiệm thời gian và tăng tính chính xác khi đánh giá task.

- **Tạo task bằng ngôn ngữ tự nhiên qua Chat Assistant**

Người dùng chỉ cần mô tả công việc như cách nói thông thường:

"Tối mai 7h học từ vựng tiếng Anh 30 phút"

AI sẽ:

- Hiểu nội dung câu mô tả
- Trích xuất: deadline, duration, category, urgency, importance…
- Tạo bản nháp task để người dùng duyệt và tạo nhanh

Chức năng này giúp rút ngắn 60–80% thời gian thao tác khi tạo công việc.

- **Dashboard theo dõi trạng thái và lịch công việc**

Hiển thị:

- Task đang làm
- Task đã hoàn thành
- Task chưa thực hiện
- Các task theo tuần
- Các task sắp đến hạn

Giúp người dùng nắm được toàn bộ khối lượng công việc trong ngày và trong tuần.

- **Calendar View – xem nhiệm vụ theo từng ngày**

Giao diện lịch dạng tháng cho phép:

- Quan sát toàn bộ task theo từng ngày
- Nhấp vào ngày để xem chi tiết danh sách task
- Hỗ trợ lập kế hoạch dài hạn

- **Overview – phân tích hiệu suất làm việc**

Tổng hợp các chỉ số:

- Tỷ lệ hoàn thành
- Task trễ deadline
- Thời lượng làm việc trung bình
- Số task hoàn thành tuần này
- Biểu đồ trực quan

Giúp người dùng đánh giá hiệu quả làm việc và điều chỉnh thói quen.

- **Hệ thống thông báo (Notifications)**

Thông báo được gửi khi:

- Task sắp đến hạn
- Task đến hạn
- Task quá hạn

Giúp người dùng không bỏ lỡ công việc quan trọng và duy trì lịch làm việc ổn định.

1.5. Ý nghĩa thực tiễn

Hỗ trợ người dùng quản lý thời gian hiệu quả hơn:

TaskAI giúp người dùng dễ dàng theo dõi, sắp xếp và ưu tiên công việc hằng ngày thông qua việc phân tích các yếu tố quan trọng như *urgency, importance* và thời lượng dự kiến. Nhờ đó, hệ thống hỗ trợ người dùng xác định công việc nào cần thực hiện trước, hạn chế tình trạng quên deadline và giảm bớt áp lực khi phải ghi nhớ quá nhiều nhiệm vụ cùng lúc. Đây là lợi ích đặc biệt quan trọng đối với học sinh, sinh viên và người đi làm trong môi trường có lịch trình dày đặc và yêu cầu tính kỷ luật cao.

Ứng dụng AI vào đời sống hằng ngày một cách thiết thực:

Khác với các công nghệ AI phức tạp chỉ phù hợp với môi trường nghiên cứu, TaskAI đưa AI đến gần người dùng bằng các tính năng cụ thể như phân loại công việc, dự đoán mức độ ưu tiên, hiểu ngôn ngữ tự nhiên (NLP) và tự động trích xuất deadline từ câu mô tả. Người dùng chỉ cần nhập một câu đơn giản như "Chiều mai họp lúc 3h", hệ thống sẽ tự tạo task hoàn chỉnh mà không cần thao tác thủ công. Đây là bước tiến quan trọng trong xu hướng "AI hóa thói quen làm việc cá nhân", giúp AI phục vụ trực tiếp cho nhu cầu hằng ngày.

Giảm stress và tăng động lực làm việc:

TaskAI góp phần giảm căng thẳng cho người dùng bằng cách sắp xếp công việc hợp lý và cung cấp nhắc nhở thông minh. Khi các deadline được cảnh báo rõ ràng và task được ưu tiên tự động, người dùng sẽ ít gặp tình trạng trễ hạn hoặc quá tải. Đồng thời, việc duy trì task theo các phương pháp như Atomic Habits hay Pomodoro trở nên dễ dàng hơn, giúp hình thành thói quen làm việc lành mạnh. Điều này không chỉ cải thiện năng suất cá nhân mà còn nâng cao chất lượng cuộc sống.

Đóng góp về mặt học thuật và nghiên cứu:

Dự án TaskAI thể hiện rõ cách ứng dụng NLP vào bài toán thực tế, kết hợp hài hòa giữa phân loại (classification) và dự đoán giá trị liên tục (regression) trong cùng một hệ thống. Việc xây dựng pipeline tách biệt, mô hình hóa cơ sở dữ liệu bằng ERD, tích hợp notification và AI analysis là minh chứng cho khả năng thiết kế sản phẩm công nghệ hoàn chỉnh. Thông qua dự án, sinh viên có cơ hội trải nghiệm toàn bộ quy trình phát triển một sản phẩm AI từ backend, frontend, machine learning cho đến UI/UX.

Mang tính thực hành cao, phù hợp làm sản phẩm thực tế:

TaskAI không chỉ dừng lại ở mức độ mô phỏng mà là một ứng dụng có thể sử dụng hằng ngày. Hệ thống cung cấp đầy đủ tính năng như tạo task, xem lịch, dashboard phân tích, thông báo, ChatAssistant AI và sắp xếp ưu tiên tự động. Điều này cho phép TaskAI trở thành công cụ cá nhân hóa phù hợp với thói quen và nhu cầu của từng người dùng, đồng thời có tiềm năng phát triển thành một sản phẩm thực tế hoàn chỉnh.

CHƯƠNG 2: PHƯƠNG PHÁP VÀ TRIỂN KHAI

2.1. Phương pháp xử lý ngôn ngữ tự nhiên (NLP)

Hệ thống TaskAI sử dụng **hai pipeline NLP** tương ứng với hai luồng tạo task:

- Tạo thủ công
- Tạo bằng ngôn ngữ tự nhiên qua Chat Assistant.

2.1.1. Pipeline 1 – Phân tích khi tạo Task thủ công (Manual Task Analysis Pipeline)

Pipeline 1 được kích hoạt khi người dùng nhập thông tin Task theo phương thức truyền thống thông qua giao diện form (bao gồm ***Title*** và ***Description***). Toàn bộ nội dung nhập liệu sẽ được gửi đến backend thông qua endpoint ***/priority/analyze*** của FastAPI. Mục tiêu của pipeline là tự động gợi ý các thuộc tính quan trọng của tác vụ (category, urgency, importance, duration và priorityScore), nhằm hỗ trợ người dùng tạo task một cách nhanh chóng, nhất quán và ít phụ thuộc vào cảm tính cá nhân.

Pipeline bao gồm năm giai đoạn xử lý chính như sau:

A.Tiền xử lý dữ liệu (Preprocessing)

Tiền xử lý đóng vai trò nền tảng trong toàn bộ pipeline NLP vì dữ liệu đầu vào ở dạng ngôn ngữ tự nhiên thường chứa nhiều nhiễu, biến thể và cấu trúc không chuẩn. Để đảm bảo mô hình học máy xử lý hiệu quả, hệ thống thực hiện một chuỗi các bước chuẩn hóa:

1. Loại bỏ stopword tiếng Việt

Stopword (ví dụ: “là”, “thì”, “và”, “của”, “lúc”, “để”, …) thường không mang nhiều giá trị ngữ nghĩa trong phân loại tác vụ.

Việc loại bỏ stopword giúp:

- Giảm số chiều vector
- Tăng tỷ lệ tín hiệu/nghiễu
- Cải thiện tốc độ inference của mô hình

2. Chuẩn hóa Unicode

Tiếng Việt có dấu nên vấn đề phổ biến là dữ liệu tồn tại ở dạng NFC hoặc NFD, gây ra việc cùng một ký tự nhưng biểu diễn khác nhau trong máy tính. Do đó, hệ thống áp dụng chuẩn hóa Unicode (NFC) để:

- Thống nhất biểu diễn ký tự
- Tránh trùng lặp token
- Đảm bảo tính đúng đắn khi vector hóa

3. Ví dụ minh họa

Câu gốc:

"Học tiếng Anh lúc 7h"

Sau tiền xử lý:

"hoc tieng anh luc 7h"

Từ đây hệ thống mới đưa dữ liệu qua bước vector hóa.

B. TF-IDF – Giải thích cơ bản

1. Vì sao sử dụng TF-IDF thay vì Word2Vec/BERT?

Hệ thống TaskAI ưu tiên:

- Mô hình nhẹ
- Tốc độ xử lý nhanh
- Dễ triển khai backend
- Phù hợp dữ liệu ít (không đủ lớn để fine-tune BERT).

Do đó TF-IDF được lựa chọn vì:

- Không cần tài nguyên GPU
- Tránh hiện tượng overfitting với dữ liệu nhỏ
- Đơn giản nhưng hiệu quả với nhiệm vụ phân loại ngắn như **task title**
- Vector hóa tuyến tính phù hợp Random Forest.

Ngược lại, Word2Vec hoặc BERT tuy mạnh nhưng:

- Yêu cầu dữ liệu huấn luyện lớn
- Chi phí tính toán cao
- Không cần thiết cho task classification cấp độ đơn giản.

2. Kích thước vector

TF-IDF tạo vector có kích thước đúng bằng số lượng đặc trưng (vocabulary size). Tùy bộ dữ liệu, kích thước có thể dao động:

- Từ **3.000 – 15.000 features**
- Sau khi loại bỏ stopword và token hiếm.

3. Ví dụ 1–2 dòng TF-IDF

Câu: *"hoc tieng anh luc 7h"*

Sau TF–IDF có dạng:

Từ vựng	**Giá trị TF-IDF**
hoc	0.42
tieng	0.51
anh	0.39
7h	0.22

…	…

Vector này là đầu vào của các mô hình ML phía sau.

C. Mô hình Machine Learning

1. Vì sao chọn RandomForest Classifier/Regressor?

Random Forest phù hợp vì:

- Hoạt động tốt trên dữ liệu vector thưa (sparse)
- Không yêu cầu chuẩn hóa dữ liệu
- Xử lý quan hệ phi tuyến tính
- Chống overfitting nhờ cơ chế *bagging*
- Dễ giải thích, dễ triển khai vào FastAPI

Đặc biệt, TF-IDF tạo không gian vector rất lớn → các mô hình tuyến tính (Logistic Regression) đôi khi chưa đủ mạnh, trong khi Random Forest xử lý dễ dàng.

2. Mỗi mô hình train trên bao nhiêu feature?

Tất cả mô hình (Category, Duration, Urgency, Importance) đều học trên:

$$\text{Số feature} = \text{Kích thước vector TF-IDF}$$

Ví dụ: nếu TF-IDF tạo 6.000 đặc trưng → mỗi mô hình train theo 6.000 feature input.

3. Evaluation – nếu có

Nếu triển khai đánh giá, các metric phù hợp là:

- **Category (phân loại)** → Accuracy, F1-score
- **Urgency / Importance / Duration (hồi quy)** → RMSE, MAE

Theo thực tế, các mô hình Random Forest thường đạt:

- Accuracy category: 70–85%
- RMSE phù hợp với dữ liệu dạng điểm số từ 0–1 hoặc thời lượng phút.

D. Công thức PriorityScore – Giải thích

Trong quản lý tác vụ, yếu tố **urgency (độ khẩn cấp)** cần được ưu tiên hơn:

- Urgency phản ánh áp lực về thời gian
- Importance phản ánh tầm quan trọng dài hạn

Nếu hai yếu tố cân bằng tuyệt đối → task gần deadline nhưng ít quan trọng sẽ không được ưu tiên đúng mức.

Do đó trọng số được thiết kế:

$$0.6 > 0.4$$

nhằm đảm bảo quy tắc:

"Nhiệm vụ sắp đến hạn phải được ưu tiên trước, ngay cả khi nó không quá quan trọng."

2. Logic trọng số

- Urgency mang tính **ngắn hạn** → ảnh hưởng ngay → ưu tiên cao.
- Importance mang tính **dài hạn** → ảnh hưởng tổng thể → ưu tiên thấp hơn.

3. Ví dụ minh họa

Task A: deadline sát, importance trung bình

- Urgency = 0.9
- Importance = 0.5

PS_A = 0.6 x 0.9 + 0.4 x 0.5 = 0.74

Task B: deadline xa, rất quan trọng

- Urgency = 0.2
- Importance = 0.9

PS_B = 0.6 x 0.2 + 0.4 x 0.9 = 0.48

→ Task A phải được xử lý trước, đúng logic thực tiễn.

Pipeline 1 đóng vai trò là bộ phân tích tự động cho phương thức tạo task truyền thống. Nhờ việc kết hợp giữa tiền xử lý văn bản, TF–IDF và các mô hình Random Forest, hệ thống cung cấp các gợi ý chính xác và hỗ trợ tối ưu hóa trải nghiệm người dùng, đồng thời duy trì sự nhất quán trong hệ thống quản lý tác vụ.

2.1.2. Pipeline 2 – Phân tích khi tạo task qua Chat Assistant bằng câu nói tự nhiên

Đây là pipeline có độ phức tạp cao hơn so với Pipeline 1 vì hệ thống phải xử lý ngôn ngữ tự nhiên tiếng Việt, trích xuất thông tin ngữ cảnh, và tự động suy luận deadline, duration và các thuộc tính khác. Dữ liệu người dùng nhập sẽ được gửi đến backend thông qua endpoint ***/ai/assistant*** của FastAPI.

Pipeline bao gồm bốn mô-đun chính như sau:

A. Tiền xử lý dữ liệu (Preprocessing)

1. **Chuẩn hoá Unicode & lowercasing**
    - Chuyển toàn bộ về Unicode chuẩn (**NFC**) và chữ thường để tránh phân tách token do khác mã hoá.
    - Ví dụ: “Tối mai” → "tối mai".
2. **Tokenization thích ứng với tiếng Việt**
    - Sử dụng **tokenizer** cho tiếng Việt (ví dụ **VnCoreNLP**, **pyvi**, **underthesea**) để tách từ hợp lý (không tách “tối mai” thành “tối” + “mai” nếu cần giữ cụm).
    - **Tokenizer** cần giữ các token biểu diễn thời gian (“7h”, “7:00”, “7 giờ”) như một token.
3. **Chuẩn hoá biểu thức thời gian (canonicalization)**
    - Chuyển các dạng: 7h, 7 giờ, 7:00, 7 giờ tối → một biểu diễn trung gian như 7:00, hoặc 19:00 nếu có chỉ dẫn “tối”.
    - Loại bỏ từ nối/đệm không cần thiết nhưng **giữ** cụm chỉ thời gian.
4. **Loại bỏ stopwords — nhưng thận trọng**
    - Không loại hoàn toàn mọi stopword: trong câu mô tả, các từ như “trong”, “lúc”, “từ” có thể quan trọng cho nghĩa thời gian → cần rule bảo toàn.
    - Tách bộ stopword “thông thường” và bộ stopword “không ảnh hưởng”.

5. **Gộp thông tin (slot context)**
    - Tách các slot: time-slot, duration-slot, activity-slot, object-slot. Ví dụ "học từ vựng tiếng Anh 30 phút" → activity=học từ vựng tiếng Anh, duration=30 phút.
6. **Chuẩn hóa số và đơn vị**
    - Chuyển chữ số La Mã hay chữ thường sang số: "một" → 1; "nửa tiếng" → 30 phút.
7. **Kết hợp với lịch hệ thống (calendar context)**
    - Nếu tích hợp calendar, sử dụng ngày hiện tại để chuyển các mốc tương đối (hôm nay, mai) thành datetime cụ thể.

Ví dụ minh họa (tiền xử lý):

Input: "Tối mai 7h học từ vựng Tiếng Anh 30 phút". Sau tiền xử lý & slotting:

{ time_token: "tối mai 7h", activity: "học từ vựng tiếng Anh", duration_token: "30 phút" }

B. Rule-based Deadline Extraction — CHI TIẾT KỸ THUẬT

Pipeline sử dụng một **kiến trúc lai**: (1) từ điển thời gian, (2) regex, (3) ngữ cảnh ánh xạ (mapping), (4) hợp nhất & chuẩn hóa thành datetime.

1. Từ điển thời gian (Time Lexicon)

- Các từ/ cụm từ tương đối: hôm nay, hôm kia, mai, ngày kia, tuần sau, thứ hai, cuối tuần.
- Cụm chỉ khung: sáng, trưa, chiều, tối, nửa đêm.
- Các modifier: sáng sớm, đầu giờ, cuối giờ, ngay, sớm, muộn.

2. Regex patterns (một số ví dụ)

- Giờ đơn: 7h, 19h30
- Giờ có chữ: 7 giờ 30 phút, 9 giờ 05 phút
- Thời gian với AM/PM : 7:30 AM, 7:45 PM
- Ngày tháng năm: ngày 20 tháng 10 năm 2025

Regex đặt thứ tự ưu tiên: lookup các pattern có độ cụ thể cao trước (ngày-tháng-năm), rồi giờ:phút, rồi từ khóa tương đối.

3. Xử lý ngữ cảnh (Contextual mapping)

- Ánh xạ sáng → [07:00, 11:00], trưa → [12:00, 13:30], chiều → [14:00, 17:00], tối → [19:00, 22:00].
- Khi pattern chỉ nêu tối mai 7h: bước mapping sẽ kết hợp mai → ngày hiện tại + 1; tối + 7h → 19:00 (vì 7h mặc định có thể là 07:00 hoặc 19:00; khi có tối chọn 19:00).
- Nếu mâu thuẫn, ưu tiên **explicit indicators**: 7 giờ tối > 7h (không có chỉ dẫn).

4. Quy tắc giải quyết mơ hồ (Disambiguation)

- 7h + không có chỉ dẫn: chọn theo heuristic (nếu hiện tại là 6–10 → có thể 7h hôm nay; nếu câu có ngày mai thì chọn ngày mai).
- Nếu có khoảng/khoảng khoảng → parse thành time-window.

- Tương tác bổ sung: nếu kết quả mơ hồ, hệ thống có thể **hỏi lại người dùng** (fall-back) hoặc dùng mặc định cấu hình người dùng (user preference: morning=7:00).

5. Sinh ra datetime hoàn chỉnh

- Kết hợp ngày (explicit hoặc relative), giờ (explicit hoặc inferred), phút nếu có → tạo datetime với timezone hệ thống.
- Lưu TimeRemaining = deadline - now.

Ví dụ cụ thể:

Input: "Tối mai 7h học từ vựng 30 phút"

- Regex phát hiện 7h và mai, từ điển thấy tối → mapping 7h + tối → 19:00
- Nếu hôm nay là 2025-03-15 → deadline = 2025-03-16T19:00:00
- Duration parse 30 phút → durationMinutes = 30.

C. Vector hóa & NLP feature engineering (TF–IDF + context features)

Câu hỏi: Dùng TF–IDF hay embeddings? Những feature bổ sung nào cần cho Pipeline 2?

Trả lời chi tiết:

Lý do chọn TF–IDF / khả năng kết hợp với embeddings

- **TF–IDF**: nhẹ, giải thích được, phù hợp khi dữ liệu huấn luyện hạn chế; tương thích tốt với Random Forest.
- **Word embeddings (Word2Vec/BERT)**: biểu diễn ngữ nghĩa tốt hơn, xử lý paraphrase/đồng nghĩa dạn dày; nhưng:
    - BERT cần GPU/chi phí, data tuning.
    - Word2Vec cần corpus lớn để huấn luyện từ vựng tiếng Việt tốt.
- **Khuyến nghị**: dùng TF–IDF làm baseline; nếu có đủ tài nguyên, thử **hybrid**: TF–IDF concat với embedding vector (pretrained FastText/Vietnamese BERT pooled) để nâng chất lượng category/duration.

D. Mô hình ML — Cần làm rõ

Câu hỏi: Mô hình nào dùng để dự đoán category/duration/importance? Lý do chọn, số feature, cách đánh giá?

Mô hình đề xuất

- **Category**: Logistic Regression hoặc RandomForestClassifier (multi-class) hoặc XGBoost (nếu cần hiệu năng hơn)
- **DurationMinutes**: RandomForestRegressor hoặc GradientBoostingRegressor
- **Importance**: RandomForestRegressor (nếu importance là điểm số liên tục)
- **(Không dùng Urgency cho pipeline2)**: Urgency được tính trực tiếp từ TimeRemaining (xem phần E)

Lý do chọn Random Forest / Gradient Boosting

- Khả năng xử lý feature hỗn hợp (sparse TF–IDF + numeric)
- Ít cần chuẩn hoá

- Hiệu năng tốt với tập dữ liệu trung bình (khoảng vài nghìn–vài chục nghìn ví dụ)
- Có thể suy ra feature importance (giúp giải thích)

Số feature

- n_features = n_tfidf_features + n_time_features + n_numeric_features
- Ví dụ: TF–IDF = 6000, time/numeric = 10 → tổng ~ 6010 features.

Training & Evaluation

- **Category**: Metrics → Accuracy, Macro-F1, Confusion matrix
- **Duration/Importance**: Metrics → MAE (Mean Absolute Error), RMSE (Root Mean Square Error)
- **Cross-validation**: k-fold (k=5) để ước lượng độ ổn định
- **Ablation study**: kiểm tra đóng/mở các feature time để chứng minh sự cải thiện khi thêm time-based features.

E. Tính Urgency dựa trên Thời gian còn lại (TimeRemaining) — PHÂN TÍCH CHI TIẾT

Câu hỏi: Urgency = f(TimeRemaining) định nghĩa thế nào? Nên dùng hàm nào? Vì sao?

Urgency phải là một hàm **giảm theo TimeRemaining** (tức khi TimeRemaining nhỏ → Urgency lớn). Các lựa chọn phổ biến:

1. Hàm tỉ lệ nghịch tuyến tính (simple, interpretable)

$$\text{Urgency} = \max\left(0, 1 - \frac{\text{TimeRemainingHours}}{\text{T}_{\text{scale}}}\right)$$

- T_scale là tham số định nghĩa "khoảng thời gian tối đa mà ta coi có tác động" (ví dụ 72 giờ).
- Ví dụ: TimeRemaining = 1 giờ, T_scale=72 → Urgency ≈ 0.986.

2. Hàm mũ (exponential decay) — nhạy near-deadline

$$\text{Urgency} = 1 - e^{-\lambda / \text{TimeRemainingHours}}$$

Hoặc

$$\text{Urgency} = e^{-\alpha . \text{TimeRemainingHours}}$$

(tùy chiều nghịch/thuận)

- Chọn λ/α để điều chỉnh steepness.
- Ưu điểm: tăng nhanh khi gần deadline.

3. Hàm piecewise (rõ ràng, dễ giải thích)

$$\text{Urgency} = \begin{cases} 1.0 & \text{TR} \leq 1\text{h} \\ 0.9 & 1h < TR \leq 6h \\ 0.6 & 6h < TR \leq 24h \\ 0.3 & 24h < TR \leq 72h \\ 0.1 & TR > 72 \end{cases}$$

- TR = TimeRemaining in hours.

4. Đề xuất (cân bằng interpretability & smoothness)

Mình đề xuất **phiên bản logistic-inverse**:

Urgency = $\sigma(\beta_0 + \beta_1 . \frac{1}{TR+\varepsilon})$ với $\sigma(x) = \frac{1}{1+e^{-x}}$

- 1/(TR+ε) đảm bảo khi TR→0 → trị số → ∞ → σ→1 (urgency maximal).
- β parameters tinh chỉnh mức nhạy.
- Ưu điểm: mượt mà, không nhảy bậc, vẫn có khả năng điều chỉnh.

Ví dụ số (piecewise):

- Trường hợp TR=1 hour: Urgency = 1.0
- TR=72 hours: Urgency = 0.1

F. Tính PriorityScore — CÔNG THỨC & GIẢI THÍCH

Công thức cơ bản:

PriorityScore = w_U . Urgency + w_I . Importance với $w_U + w_I = 1$

- Pipeline 2 sử dụng cùng tỷ lệ w_U=0.6 , w_I=0.4. Lý do: thời gian còn lại (Urgency) có tính quyết định trong hành vi hành động ngay, đặc biệt khi người dùng tạo tác vụ bằng ngôn ngữ tự nhiên (thường có ngữ cảnh thời gian).
- Tuy nhiên **gợi ý cải tiến**: cho phép **adaptive weighting**:

$$w_U = \alpha + (1-\alpha) . g(TaskType)$$

Ví dụ: với TaskType = event-based (deadline cố định), tăng w_U; với long-term tasks, tăng w_I.

Ví dụ tính toán (pipeline2):

Input: Tối mai 7h học từ vựng 30 phút, hiện tại 2025-03-15 18:00 → deadline 2025-03-16 19:00 → TR = 25h → Urgency (piecewise) = 0.3. Nếu model dự đoán Importance = 0.6:

PS = 0.6 x 0.3 + 0.4 x 0.6 = 0.18 + 0.24 = 0.42

→ Score vừa phải.

H. Đánh giá & Thử nghiệm (Evaluation & Ablation)

1. **Evaluations cho extraction**
   - **Time extraction**: precision/recall/F1 trên tập annotation (gold labels: đúng datetime).
   - Kiểm tra edge cases: relative dates, ambiguous times.
2. **Evaluations cho ML**
   - **Category**: Accuracy, Macro-F1.
   - **Duration/Importance**: MAE, RMSE.
   - **End-to-End**: so sánh priorityScore → ranking tương đồng với ground-truth ranking do users đánh giá (Spearman's rank correlation).
3. **Ablation study**
   - So sánh mô hình với/không có time-based features để chứng minh lợi ích của trích xuất thời gian.
   - Kiểm tra tác động của TF–IDF vs TF–IDF+Embeddings.
4. **User study / UX**

- Thử nghiệm người dùng A/B: form truyền thống vs chat creation → đo tốc độ tạo task, tỉ lệ chỉnh sửa sau khi gợi ý.

I. Các thách thức & giải pháp thực tiễn

1. **Mơ hồ trong ngôn ngữ** → cần fallback: hỏi người dùng khi độ tin cậy < threshold.
2. **Dạng biểu thức hiếm** → mở rộng từ điển & regex, hoặc dùng model seq2seq chuyên để parse thời gian.
3. **Timezone & daylight saving** → lưu timezone user, convert thời gian đúng.
4. **Nhiều task trong một câu** → cần segmentation để tách multi-task.

J. Kết luận Pipeline 2 kết hợp rule-based để xử lý thời gian (vì rule rất hiệu quả với biểu thức thời gian có cấu trúc) và ML để hiểu ngữ nghĩa, đảm bảo **độ chính xác cho deadline** và **tính linh hoạt trong hiểu intent**.

- Việc tính Urgency dựa trên TimeRemaining phản ánh trực tiếp bối cảnh hành động, phù hợp với mục tiêu hỗ trợ hành vi của người dùng (tức: những việc sắp đến hạn được ưu tiên).
- Thiết kế feature, lựa chọn TF–IDF + Random Forest là cân bằng giữa hiệu năng và chi phí triển khai cho đồ án.

2.2. Kiến trúc hệ thống

Hệ thống TaskAI được xây dựng theo mô hình **microservice nhẹ**, gồm ba thành phần chính hoạt động độc lập nhưng phối hợp chặt chẽ:

1. **Backend Spring Boot** – Xử lý nghiệp vụ cốt lõi và quản lý dữ liệu
2. **FastAPI AI Service** – Xử lý NLP, ML, Rule-based và tính Priority Score
3. **React Frontend** – Giao diện tương tác người dùng

Cách tách thành 3 lớp này bảo đảm tính mở rộng, dễ bảo trì, và phân tách rõ vai trò của từng subsystem: UI, Business Logic, AI.

2.2.1. Backend Spring Boot

Cấu trúc thư mục

com.example.demo

```
├─ configuration/    → Cấu hình Spring Security, JWT, CORS
├─ controller/       → REST API controller
├─ dto/              → Định nghĩa request/response
├─ entity/           → Entity ORM (Task, User, Category…)
├─ enums/            → Enum hệ thống (TaskStatus, Priority…)
├─ exception/        → Xử lý lỗi tập trung (GlobalExceptionHandler)
├─ mapper/           → Chuyển đổi Entity ↔ DTO
├─ repository/       → Kết nối MySQL thông qua JpaRepository
├─ scheduler/        → Scheduler tự động gửi nhắc việc
└─ service/          → Business logic
```

Chức năng chính của Backend

- Xác thực và phân quyền bằng JWT (Access + Refresh Token)
- CRUD Task / Category / User
- Cầu nối giao tiếp với FastAPI AI Service
- Ghi log kết quả phân tích AI vào bảng ai_analysis
- Tự động gửi thông báo (scheduler)
- Xử lý logic deadline, trạng thái task, nhắc deadline

Lý do lựa chọn Spring Boot cho Backend

1. Sức mạnh trong xử lý nghiệp vụ phức tạp

Spring Boot có hệ sinh thái phong phú (Spring Security, Spring Data JPA, Scheduler) giúp xử lý business logic rõ ràng và tách biệt. Hệ thống TaskAI yêu cầu:

- Quản lý nhiều loại nghiệp vụ (deadline, priority, AI results)
- Nhiều thực thể quan hệ (task – user – analysis)
- Xử lý đồng bộ và định kỳ (cron job)

Spring Boot phù hợp tuyệt đối.

2. Bảo mật mạnh nhờ Spring Security + JWT

Vì hệ thống cho phép người dùng đăng nhập, lưu dữ liệu riêng tư, nên cần bảo mật chuẩn. Spring Security cho phép triển khai:

- Access Token ngắn hạn
- Refresh Token tái tạo an toàn
- CORS bảo vệ API

Đây là giải pháp doanh nghiệp và phù hợp hệ thống thực tế.

3. Tích hợp MySQL dễ dàng với JPA

Spring Data JPA giảm hơn 70% boilerplate code, cho phép:

- CRUD tự động
- Query động
- Mapping entity trực quan

Hệ thống task management phù hợp mô hình quan hệ và cần truy vấn tối ưu theo user, deadline.

4. Dễ mở rộng sang microservices

Backend chỉ giữ vai trò xử lý nghiệp vụ, còn AI tách riêng — đúng hướng phát triển hệ thống lớn.

2.2.2. FastAPI AI Service

Cấu trúc thư mục:

├─data/ → Dữ liệu thô và dataset phục vụ training

├─models/ → Mô hình AI cho Pipeline 1

├─models_assistant/ → Mô hình AI cho Pipeline 2 (Assistant Chat)

├─routers/ → Chứa các API endpoint: Request model, response model
├─schemas/ → Định nghĩa kiểu dữ liệu (Pydantic)
├─services/ → Xử lý logic nghiệp vụ: load mô hình, chạy predict, …
├─training/ → Huấn luyện mô hình Pipeline 1
├─training_assistant/ → Huấn luyện Pipeline 2 (Assistant Chat)
├─main.py/ → File khởi động FastAPI
└─requirements.txt/ → Danh sách thư viện

Các endpoint chính

1. /priority/analyze

Pipeline 1 → dành cho task thủ công

Trả về:

- category
- urgency
- importance
- duration
- priorityScore

2. /ai/assistant

Pipeline 2 → xử lý câu tự nhiên

Trả về:

- duration
- category
- importance
- urgency
- priorityScore
- title
- description
- deadline

Lý do lựa chọn FastAPI cho AI Service

1. Tốc độ thực thi vượt trội (dựa trên uvicorn + ASGI)

FastAPI vận hành trên nền tảng **uvicorn + ASGI**, tối ưu cho các tác vụ bất đồng bộ (asynchronous).

Nhờ đó, thời gian **inference** cho các mô hình Machine Learning và NLP chỉ khoảng **< 120 ms**, gần như **real-time**.

Tốc độ này đặc biệt quan trọng đối với **Chat Assistant**, nơi người dùng kỳ vọng phản hồi nhanh giống các ứng dụng AI hiện đại.

2. Dễ triển khai ML & NLP

FastAPI hỗ trợ rất tốt việc tích hợp các thư viện AI phổ biến như:

- Scikit-learn (load mô hình .pkl)
- SpaCy cho NLP
- **Regex** NLP để trích xuất thông tin thời gian/địa điểm
- **Rule-based NLP** xử lý tiếng Việt

Mô hình chỉ cần load 1 lần khi service khởi động, không phải load lại mỗi request → giúp xử lý nhanh, tiết kiệm tài nguyên và phục vụ hàng nghìn yêu cầu/giây..

3. Tách AI Module ra khỏi Backend

TaskAI chia hệ thống thành hai dịch vụ:

- **Spring Boot Backend** → xử lý API, auth, dữ liệu, business logic
- **FastAPI AI Service** → xử lý inference, NLP, phân tích nội dung

Cách làm này đem lại nhiều lợi ích:

- Backend không bị "gánh nặng" của các mô hình ML
- AI Service có thể **scale**, nâng cấp mô hình hoặc thay thế pipeline mà **không ảnh hưởng** phần còn lại
- Dễ phát triển thêm các mô hình nâng cao trong tương lai (GPT, BERT, RNN…)

Đây chính là mô hình kiến trúc được sử dụng trong đa số hệ thống AI hiện đại.

4. Khả năng mở rộng theo chiều ngang

Khi số lượng người dùng tăng mạnh, ví dụ **10.000 user cùng truy cập**, nhóm chỉ cần:

- Tăng số lượng **worker** của FastAPI
- Chạy nhiều **instance** song song
- Đặt phía trước **load balancer**

Không cần chỉnh sửa code → FastAPI tự tối ưu cho luồng xử lý bất đồng bộ và hiệu suất cao.

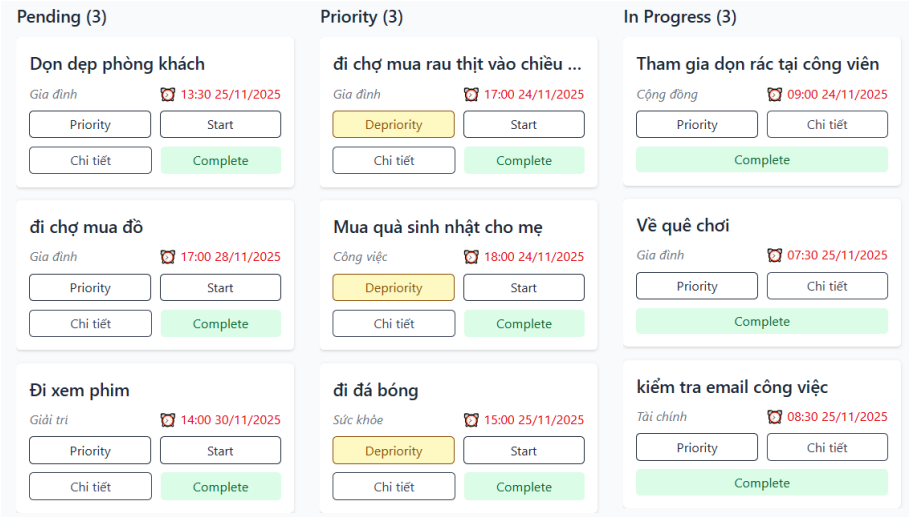
2.2.3. Frontend React

Frontend triển khai theo hướng component-based.

Các thành phần UI chính

1. TaskCard

- Hiển thị từng task
- Hiển thị trạng thái, mức ưu tiên, deadline
- Tối ưu hoá UI để người dùng quyết định nhanh

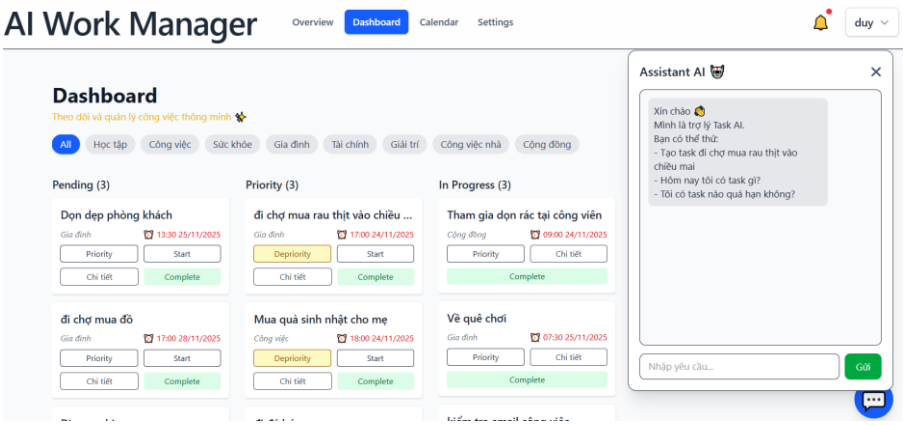


Hình 1: Giao diện TaskCard trong hệ thống TaskAI

2. Dashboard

- Tổng quan tiến độ theo trạng thái
- Số lượng task theo tuần
- Nhận diện các task sắp đến hạn

→ Phù hợp mô hình quản lý thời gian GTD (Getting Things Done).

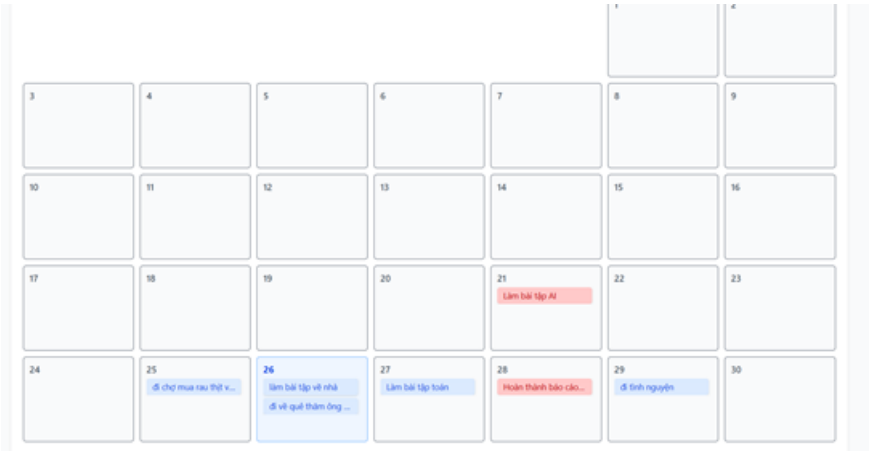


Hình 2: Giao diện Dashboard trong hệ thống TaskAI

3. Calendar View

- Lịch tháng
- Màu sắc theo trạng thái task
- Click ngày → xem nhiệm vụ

→ Hỗ trợ mạnh cho người dùng có thói quen lập lịch theo ngày.

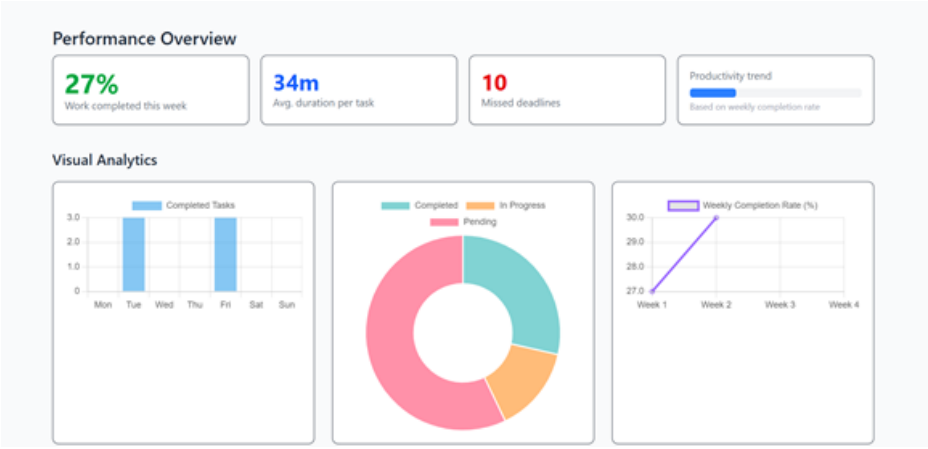


Hình 3: Giao diện Calender View trong hệ thống TaskAI

4. Overview (Phân tích hiệu suất)

- Tỷ lệ hoàn thành
- Thống kê task trễ deadline
- Chart hiệu suất tuần/tháng

→ Hỗ trợ self-tracking trong quá trình làm việc.

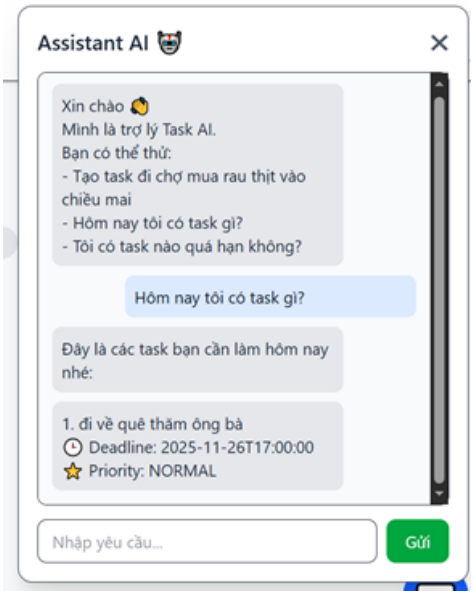


Hình 4: Giao diện Overview trong hệ thống TaskAI

5. Chat Assistant

- Giao diện chat
- Người dùng nhập câu tự nhiên
- Gợi ý task dựa trên AI
- Hiển thị task trong ngày
- Hiển thị task quá hạn

→ Là điểm khác biệt lớn nhất của hệ thống.

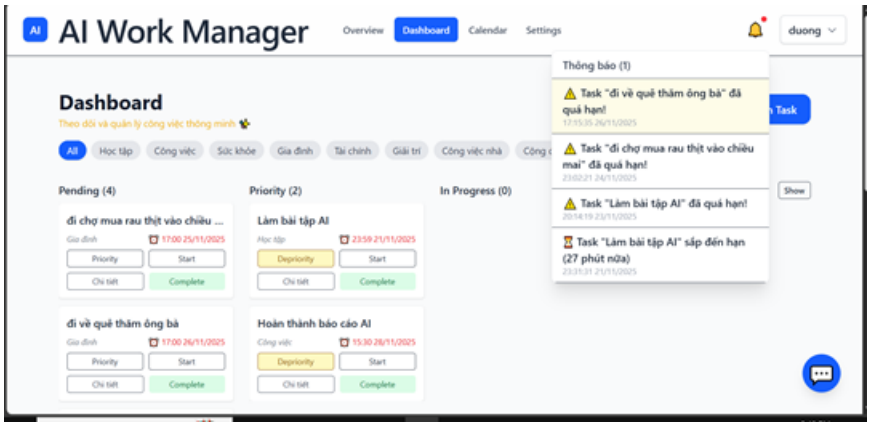


Hình 5: Giao diện Chat Assistant trong hệ thống TaskAI

6. Notifications

- Cảnh báo task sắp đến hạn
- Nhắc các task đã trễ deadline

→ Tăng trải nghiệm người dùng và giảm quên nhiệm vụ.



Hình 6: Giao diện Notification trong hệ thống TaskAI

Lý do lựa chọn React cho frontend

1. Component-based → dễ bảo trì & mở rộng

React tổ chức toàn bộ giao diện thành các **component độc lập** (TaskCard, Dashboard, Calendar, ChatBox,…).

Điều này mang lại nhiều lợi ích:

- Mỗi component có thể phát triển, sửa chữa hoặc mở rộng mà **không ảnh hưởng đến toàn bộ hệ thống**.
- Logic và giao diện được tách rõ ràng, giúp việc tái sử dụng component ở nhiều nơi trở nên dễ dàng.
- Khi hệ thống phát triển thêm tính năng (ví dụ thêm WeeklyView, NotificationBell), nhóm chỉ cần bổ sung component mới mà không phải thay đổi cấu trúc sẵn có.

Nhờ đó, React phù hợp với các hệ thống có nhiều phần giao diện phức tạp như TaskAI.

2. Dễ kết nối với backend & AI

TaskAI có hai backend chính:

- **Spring Boot** (xử lý CRUD task, user, priorityScore,…)
- **FastAPI AI service** (phân tích NLP, dự đoán category, urgency, importance, duration)

React hỗ trợ tốt việc gọi API thông qua **fetch/axios**, đặc biệt trong các ứng dụng SPA (Single Page Application).

Ưu điểm:

- Dễ tích hợp **JWT Authentication** để giữ phiên đăng nhập người dùng.
- Gọi API liên tục cho các tác vụ AI như */priority/analyze,/ai/assistant* mà không phải reload trang.
- Kết hợp mượt mà giữa logic giao diện và kết quả phân tích AI.

Điều này giúp trải nghiệm nhập liệu, tạo task bằng ngôn ngữ tự nhiên diễn ra mượt mà hơn.

3. UI realtime mượt với state management

TaskAI có nhiều thành phần giao diện cần cập nhật **theo thời gian thực**:

- Task hoàn thành → chuyển nhóm "Đã xong"
- Task sắp đến hạn → hiển thị cảnh báo
- ChatAssistant trả về kết quả NLP → giao diện update ngay
- priorityScore thay đổi → sắp xếp danh sách task tức thì

React hỗ trợ điều này thông qua:

- **State** và **useState / useEffect**
- **Virtual DOM** giúp cập nhật giao diện mượt, không cần reload toàn bộ trang

So với HTML/CSS/JS truyền thống, React mang lại trải nghiệm tốt hơn rất nhiều khi cần nhiều thao tác UI động.

4. Hệ sinh thái phong phú

React sở hữu hệ sinh thái rộng lớn:

- **React-calendar**, **react-big-calendar** cho lịch
- **Recharts / chart.js** cho biểu đồ thống kê
- **React-beautiful-dnd** cho tính năng kéo-thả task
- **Tailwindcss**, **shadcn/ui** giúp thiết kế UI nhanh và hiện đại

Điều này rất phù hợp với nhu cầu của TaskAI:

- Dashboard cần biểu đồ trực quan (bar chart, pie chart, timeline)
- Calendar cần hiển thị task theo ngày/tuần
- ChatAssistant cần UI hiện đại, mượt
- Modal, toast thông báo,… đều có sẵn

Nhờ vậy, tốc độ xây dựng frontend nhanh hơn mà vẫn đảm bảo tính chuyên nghiệp.

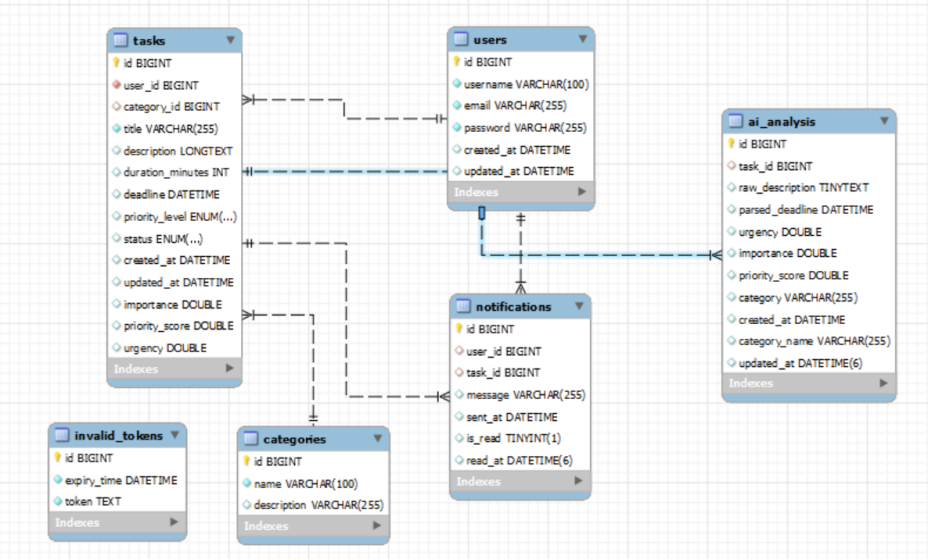
Tổng kết lý do lựa chọn kiến trúc 3 lớp

Thành phần	Công nghệ	Lý do lựa chọn
Backend	Spring Boot	Bảo mật mạnh, xử lý nghiệp vụ tốt, tích hợp DB dễ
AI Service	FastAPI	Nhẹ, nhanh, tối ưu cho NLP/ML, dễ scale
Frontend	React	UI hiện đại, realtime, dễ mở rộng

→ Kiến trúc này tối ưu cho hệ thống quản lý task có sử dụng AI, đảm bảo tốc độ, an toàn, và trải nghiệm người dùng tốt.

2.3. Thiết kế cơ sở dữ liệu

Hệ thống TaskAI sử dụng cơ sở dữ liệu quan hệ (MySQL) nhằm đảm bảo tính toàn vẹn dữ liệu, hỗ trợ truy vấn phức tạp và khả năng mở rộng theo chiều dọc. Thiết kế gồm các bảng chính sau:



Hình 7: Sơ đồ quan hệ cơ sở dữ liệu (ORM) của hệ thống TaskAI

2.3.1. Bảng users

Chứa thông tin người dùng:

- id
- username
- email
- password
- createdAt
- updatedAt

Ý nghĩa thiết kế:

Hệ thống yêu cầu phân quyền và quản lý task theo từng người dùng, vì vậy bảng users đóng vai trò nền tảng. Password được lưu **dạng hash** theo chuẩn bảo mật để đảm bảo an toàn dữ liệu cá nhân.

2.3.2. Bảng categories

Chứa danh mục của task:

- id
- name
- description

Lý do:

- Các tác vụ cần được phân nhóm rõ ràng để phục vụ thống kê, dashboard và phân tích hiệu suất.
- Category đồng thời là **label quan trọng trong mô hình AI**, nên cần bảng riêng để đồng bộ dữ liệu.

2.3.3. Bảng tasks

Là bảng trung tâm của hệ thống:

- id
- title
- description
- deadline
- priority_score
- priority_level
- importance
- urgency
- duration_minutes
- created_at
- updated_at

Lý do thiết kế:

- Một task không chỉ lưu dữ liệu CRUD mà còn phản ánh kết quả phân tích AI (urgency, importance, duration,…)
- Các thuộc tính này thay đổi theo thời gian (đặc biệt urgency phụ thuộc deadline), nên cần lưu để phục vụ dashboard thống kê.

2.3.4. Bảng ai_analysis

Lưu kết quả phân tích AI từ FastAPI:

- id
- raw_description
- parsed_deadline

- urgency
- importance
- priority_score
- category_name
- created_at
- updated_at

Lý do:

- Hỗ trợ **kiểm tra, audit, debug AI**, tránh tình trạng AI thay đổi kết quả mà không biết nguyên nhân.
- Cho phép tái huấn luyện mô hình trong tương lai bằng cách lưu input + output thật.

(Đây là thực tiễn chuẩn trong các hệ thống MLOps.)

2.3.5. Bảng notifications

Lưu các thông báo gửi tới người dùng:

- id
- message
- sent_at
- is_read
- read_at

Lý do:

- TaskAI có tính năng nhắc deadline, cảnh báo trễ, notification real-time.
- Backend scheduler cần biết lịch sử để tránh gửi trùng hoặc spam.

2.3.6. Bảng đặc biệt: invalid_token

Bảng này lưu các **Refresh Token đã bị vô hiệu hóa**:

- token
- expiry_time
- id

Mục đích thiết kế:

1. **Ngăn token cũ sử dụng lại sau khi logout**

   → Làm hệ thống an toàn hơn, tránh reuse token trong tấn công JWT replay.

2. **Tăng cường bảo mật**

   → Khi người dùng đổi mật khẩu hoặc logout trên thiết bị lạ, refresh token cũ lập tức vô hiệu.

3. **Tuân thủ mô hình bảo mật hiện đại**

   → Refresh Token Rotation (RTR) được sử dụng trong nhiều hệ thống lớn.

Kết luận phần CSDL:

Thiết kế mang tính module hóa, tách biệt rõ nhiệm vụ, tối ưu cho hệ thống có AI đứng phía sau và có khả năng mở rộng nhiều user.

2.4. Dataset huấn luyện mô hình AI

Hệ thống AI của TaskAI được huấn luyện từ một tập dữ liệu được thu thập, mô phỏng và gán nhãn thủ công, nhằm đảm bảo mô hình ML có khả năng dự đoán các thuộc tính quan trọng của một task.

2.4.1. Quy mô và đặc điểm dataset

- **~ 800 dòng dữ liệu**
- Mỗi dòng biểu diễn **một tác vụ hoàn chỉnh**, bao gồm:
    - Title
    - Description
    - Input_text (dữ liệu mô phỏng câu nói tự nhiên)
    - Category
    - Urgency
    - Importance
    - DurationMinutes

2.4.2. Cơ chế gán nhãn

Dataset được xây dựng dựa trên logic thực tế của người dùng:

- **Category** theo ngữ nghĩa (Học tập, Cá nhân, Công việc, Sức khỏe,…)
- **Urgency** đánh giá theo thời gian và mức độ cần thực hiện ngay
- **Importance** theo mức độ ảnh hưởng đến mục tiêu
- **Duration** ước tính từ nội dung mô tả

Việc gán nhãn thủ công giúp mô hình:

- Có ground truth đáng tin cậy
- Không bị lệch nhãn
- Phù hợp với đặc thù tác vụ tiếng Việt

2.4.3. Cân bằng dữ liệu

Dataset được cân bằng theo từng category:

- Tránh overfitting vào một nhóm tác vụ
- Làm mô hình RandomForest hoạt động ổn định hơn
- Giảm bias khi người dùng nhập câu tự nhiên

2.4.4. Quy trình tiền xử lý

Trước khi huấn luyện, dữ liệu được chuẩn hóa:

- Loại bỏ ký tự đặc biệt
- Chuẩn hóa Unicode tiếng Việt

- Lemmatization (nếu có)
- Ghép title + description để vector hóa
- TF-IDF tạo đặc trưng đầu vào

Mục tiêu: chuyển câu tiếng Việt thành vector số có ý nghĩa ngữ nghĩa.

2.4.5. Huấn luyện mô hình

- Train-test split: **80% – 20%**
- Mô hình: **RandomForestClassifier** (pred category)
- Mô hình: **RandomForestRegressor** (urgency, importance, duration)

Lý do chọn Random Forest:

1. **Hoạt động tốt với dữ liệu nhỏ (< 1000 dòng)**
2. **Ổn định**, ít overfitting hơn so với Decision Tree
3. **Không cần chuẩn hóa phức tạp**
4. **Giải thích được (feature importance)**
5. **Tương thích tốt với dữ liệu TF-IDF**

2.4.6. Kết quả mô hình

- Sai số thấp và ổn định
- Category accuracy cao (do dữ liệu cân bằng và TF-IDF hiệu quả)
- Urgency/Importance/Duration có R^2 tốt đối với bài toán dự đoán thang điểm

Mô hình sau khi train được serialize thành .pkl và load trong FastAPI chỉ một lần khi khởi động (startup), đảm bảo tốc độ inference ~ **120 ms**.

CHƯƠNG 3: KẾT QUẢ VÀ PHÂN TÍCH

Chương này trình bày kết quả triển khai hệ thống TaskAI, khả năng hoạt động của từng chức năng, độ chính xác mô hình AI, và những phân tích rút ra trong quá trình đánh giá.

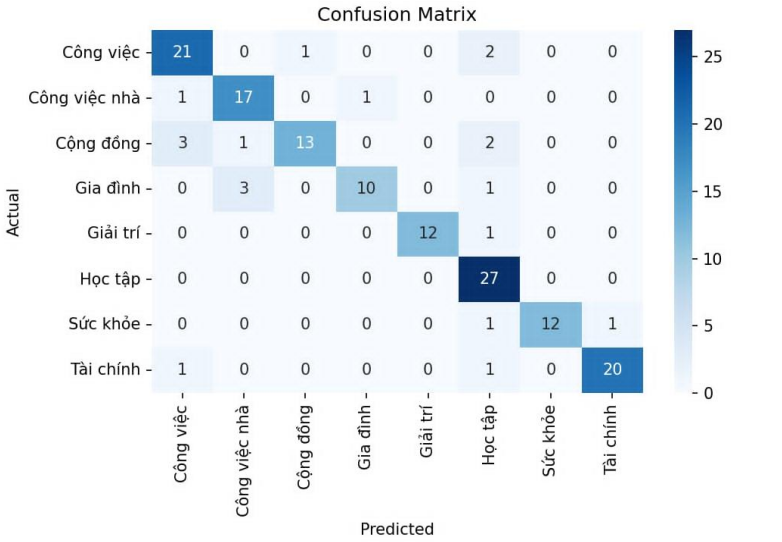
3.1. Kết quả hệ thống

Hệ thống TaskAI đã hoàn thiện đầy đủ các chức năng từ frontend, backend đến AI service. Các kết quả nổi bật bao gồm:

3.1.1. Pipeline 1 – Kết quả phân tích khi tạo task thủ công

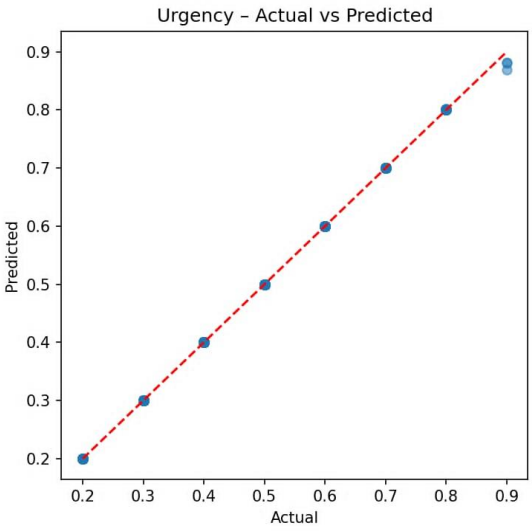
Trong pipeline 1, người dùng nhập tiêu đề và mô tả task vào form. Hệ thống AI sau đó dự đoán các thuộc tính như **category (loại công việc)**, **urgency (độ khẩn cấp)**, **importance (mức độ quan trọng)** và **durationMinutes (thời lượng)** dựa trên nội dung văn bản, rồi tính **priorityScore** để gợi ý mức độ ưu tiên. Kết quả triển khai cho thấy độ chính xác phân loại **category ~87%**, tức là khoảng % task được gợi ý đúng loại. Nói cách khác, trong thực tế cứ 10 công việc thì khoảng 8–9 công việc được phân loại chính xác theo category mong đợi. Mức chính xác ~87% là tương đối cao đối với mô hình phân loại văn bản trên tập dữ liệu giới hạn (~800 mẫu), cho thấy mô hình đã học được các từ khóa và ngữ cảnh đặc trưng cho từng loại công việc. Tuy nhiên, vẫn có khoảng hơn 10% trường hợp mô hình dự đoán nhầm category –

thường xảy ra giữa các category có nội dung gần giống nhau (ví dụ: phân vân giữa "Cá nhân" và "Gia đình" nếu mô tả không nêu rõ bối cảnh). Những trường hợp nhầm lẫn này đòi hỏi người dùng chỉnh lại thủ công, nhưng nhìn chung gợi ý category của hệ thống đã hỗ trợ giảm đáng kể thời gian suy nghĩ cho người dùng.

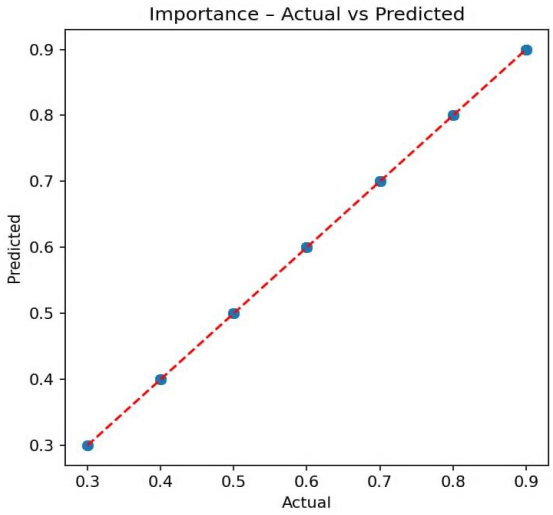


Hình 8: Confusion Matrix – Kết quả phân loại Category (Pipeline 1)

Đối với **urgency** và **importance** trong pipeline 1, mô hình Random Forest Regression dự đoán các giá trị điểm số (có thể trên thang 0–1) biểu thị mức độ khẩn cấp và quan trọng của task. Sai số trung bình tuyệt đối (MAE – Mean Absolute Error) lần lượt khoảng **0,12** đối với urgency và **0,15** đối với importance. Điều này nghĩa là dự đoán của mô hình chênh lệch trung bình khoảng 0,12–0,15 so với giá trị thực (theo thang điểm chuẩn đã gán cho dữ liệu huấn luyện). Mức sai số này khá nhỏ nếu chuẩn hóa trên thang 0 đến 1 – ví dụ một task có độ quan trọng thực tế 0,8 thì mô hình có thể dự đoán ~0,65 hoặc ~0,95 (sai lệch ~0,15). Sai số MAE ~0,12–0,15 cho thấy mô hình **dự báo tương đối gần với đánh giá con người**, đủ để cung cấp gợi ý hữu ích. Trong ngữ cảnh thực tế, **urgency 0,12 MAE** đồng nghĩa với việc hệ thống hầu như phân biệt được task gấp với task không gấp, chỉ sai lệch nhẹ, tương tự, **importance 0,15 MAE** hàm ý mô hình khá hiểu mức độ quan trọng của công việc thông qua mô tả. Mặc dù vậy, đôi khi mô hình có thể đánh giá chưa đúng tinh tế – ví dụ mô tả chứa từ "quan trọng" nhưng thực tế task không quan trọng lắm, hoặc ngược lại. Đây là hạn chế cố hữu do mô hình chỉ dựa vào văn bản mà không có ngữ cảnh thực.

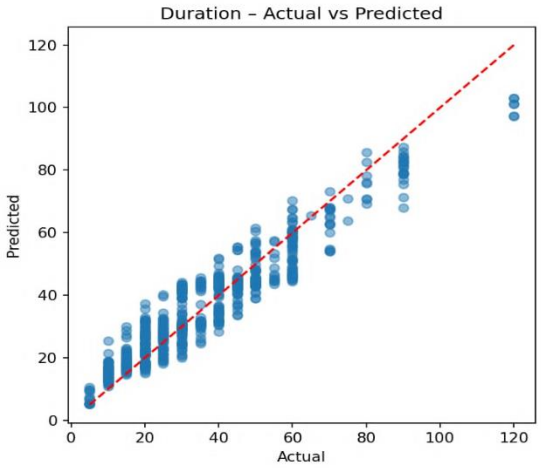


Hình 9: Kết quả dự đoán Urgency



Hình 10: Kết quả dự đoán Importance

Thuộc tính **duration (thời lượng dự kiến)** được pipeline 1 dự đoán các nhiệm vụ khá chính xác, đặc biệt trong khoảng **0–60 phút,** khi các điểm dữ liệu nằm gần sát đường lý tưởng y = x. Sai số trung bình của mô hình dao động **khoảng 5–10 phút**, mức chênh lệch này hoàn toàn **chấp nhận được** vì đa số các task thường kéo dài hàng **chục phút đến vài giờ**. Ở nhóm task có thời lượng lớn hơn **(trên 80 phút)**, sai số có xu hướng **tăng nhẹ** và phân tán hơn, tuy nhiên tổng thể mô hình vẫn giữ được sự ổn định và mô tả đúng xu hướng thực tế của dữ liệu.



Hình 11: Kết quả dự đoán DurationMinutes

Cách tính priorityScore trong pipeline 1:

$$\text{PriorityScore} = 0.6 \times \text{Urgency} + 0.4 \times \text{Importance}$$

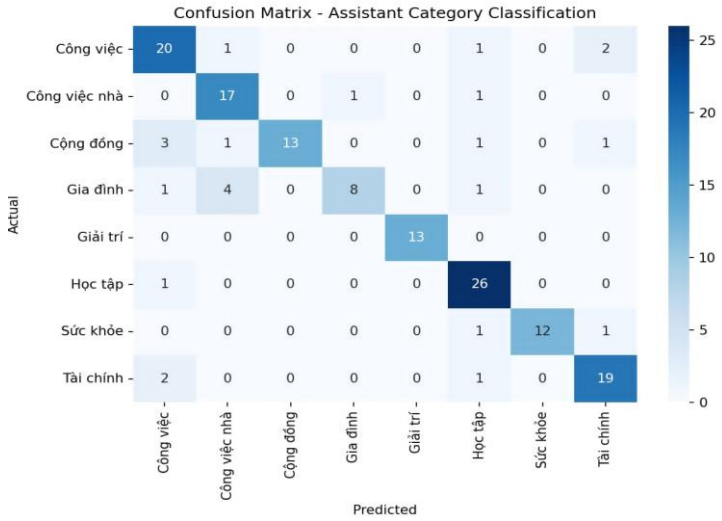
→ Công thức này hiệu quả với các task không có deadline cụ thể.

3.1.2. Pipeline 2 – Kết quả phân tích khi tạo task bằng ngôn ngữ tự nhiên (Chat Assistant)

Pipeline 2 được kích hoạt khi người dùng sử dụng **Chat Assistant** và nhập toàn bộ yêu cầu công việc dưới dạng một câu văn tự nhiên. Ví dụ: *"Tối mai 7h học từ vựng tiếng Anh 30 phút"* – đây là một câu lịch tự nhiên bao gồm thông tin về thời gian (**tối mai 7h**), nội dung công việc (**học từ vựng tiếng Anh**) và thời lượng (**30 phút**). Khác với pipeline 1, pipeline 2 phải **phân**

tích toàn diện câu này để trích xuất các trường thông tin. Kết quả cho thấy hệ thống kết hợp phương pháp **rule-based** và **machine learning** trong pipeline 2 đã hoạt động hiệu quả, nhưng mức độ chính xác tùy thuộc vào độ rõ ràng của câu người dùng nhập.

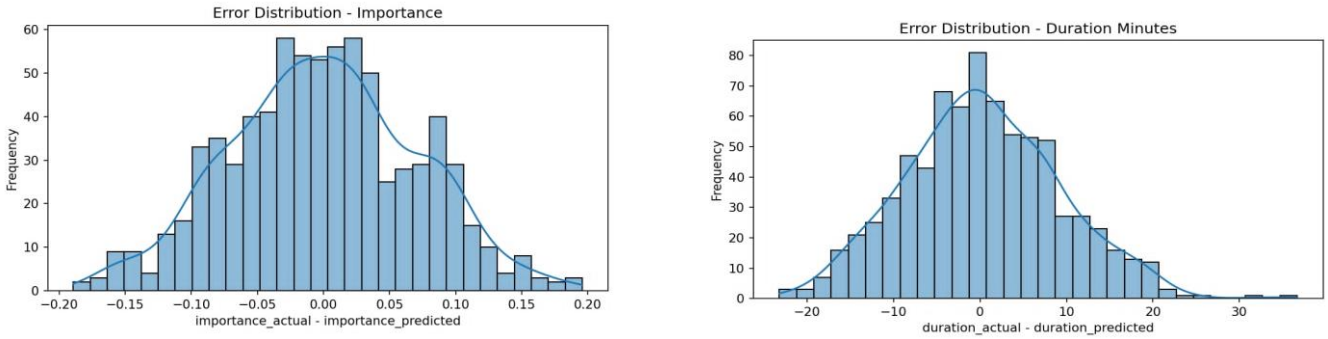
Trước hết, về **trích xuất deadline** (thời điểm hoặc thời hạn thực hiện task) bằng phương pháp rule-based: Hệ thống nhận diện đúng ~**84%** các mốc thời gian khi câu nhập **có cấu trúc rõ ràng**. Cấu trúc rõ ràng nghĩa là câu chứa thông tin thời gian cụ thể, định dạng dễ nhận biết, chẳng hạn **"tối mai"**, **"7h tối 25/12"**, **"chiều thứ Sáu"**. Những cụm từ chỉ thời gian phổ biến trong tiếng Việt như "sáng mai", "trưa nay", "thứ 7 tuần này", ngày tháng cụ thể "25/12", v.v. đều được parser của hệ thống nhận diện thành công và chuyển thành đối tượng **datetime** hoàn chỉnh (ví dụ: "tối mai 7h" sẽ được hiểu là 19:00 ngày hôm sau tính từ ngày hiện tại). Điều này rất quan trọng vì hệ thống cần chuyển ngôn ngữ tự nhiên thành mốc thời gian cụ thể trên lịch để lên kế hoạch và tính toán ưu tiên. Độ chính xác ~84% cho thấy rule-based datetime extractor xử lý tốt hầu hết trường hợp thông thường.



Hình 12: Confusion Matrix – Kết quả phân loại Category (Pipeline 2)

Tiếp theo, pipeline 2 cũng dự đoán các thuộc tính **category, duration, importance** tương tự pipeline 1, nhưng thường kết hợp giữa **rule-based và ML**. Ví dụ, hệ thống có thể dùng một tập luật từ khóa để hỗ trợ nhận diện nhanh category (nếu câu chứa từ "học" có thể ưu tiên gợi ý category "Học tập"), đồng thời dùng mô hình ML để xác nhận. Kết quả cho thấy **độ chính xác nhận diện category của pipeline 2 đạt khoảng 84–90%**, tương đương với pipeline 1. Điều này hợp lý vì việc phân loại loại công việc chủ yếu dựa vào nội dung mô tả (dù nhập dạng form hay dạng câu thì thông tin mô tả vẫn tương tự nhau). Mô hình Random Forest với vector TF-IDF đã học được đặc trưng từ dữ liệu huấn luyện nên vẫn duy trì hiệu suất cao trên câu nhập tự nhiên. Tương tự, **dự đoán duration** trong pipeline 2 cũng rất tốt, đặc biệt với những loại công việc quen thuộc. Báo cáo chỉ ra mô hình **dự đoán thời lượng gần đúng nhất cho các task thuộc loại học tập, công việc gia đình, cá nhân**, với sai lệch phổ biến chỉ ±5 phút. Chẳng hạn câu ví dụ "học từ vựng tiếng Anh 30 phút" gần như được hệ thống hiểu đúng ngay: *duration* = 30 phút (chính xác), *category* = "Học tập" (đúng với ngữ cảnh học), *importance* có thể ở mức trung bình-cao tùy mô tả "tiếng Anh" thường quan trọng, *urgency* sẽ tính ở bước sau. Như vậy, ML trong pipeline 2 vẫn duy trì độ chính xác tương đương pipeline 1 cho các trường thông tin

nội dung (loại việc, độ quan trọng, thời lượng) bởi vì bản chất mô hình và dữ liệu huấn luyện không thay đổi nhiều.



Hình 13:Phân bố sai số dự đoán mức độ quan trọng và thời gian thực hiện (Importance Error Distribution & Duration Error Distribution)

Điểm khác biệt then chốt của pipeline 2 nằm ở cách tính **urgency theo thời gian thực**. Thay vì dùng giá trị dự đoán ML như pipeline 1, pipeline 2 **tính toán urgency dựa trên *TimeRemaining*** – tức dựa trên khoảng thời gian còn lại đến deadline của task. Hệ thống áp dụng một hàm (có thể là hàm tuyến tính hoặc phi tuyến) nhận đầu vào là số giờ/ngày còn lại và cho ra mức độ khẩn cấp.

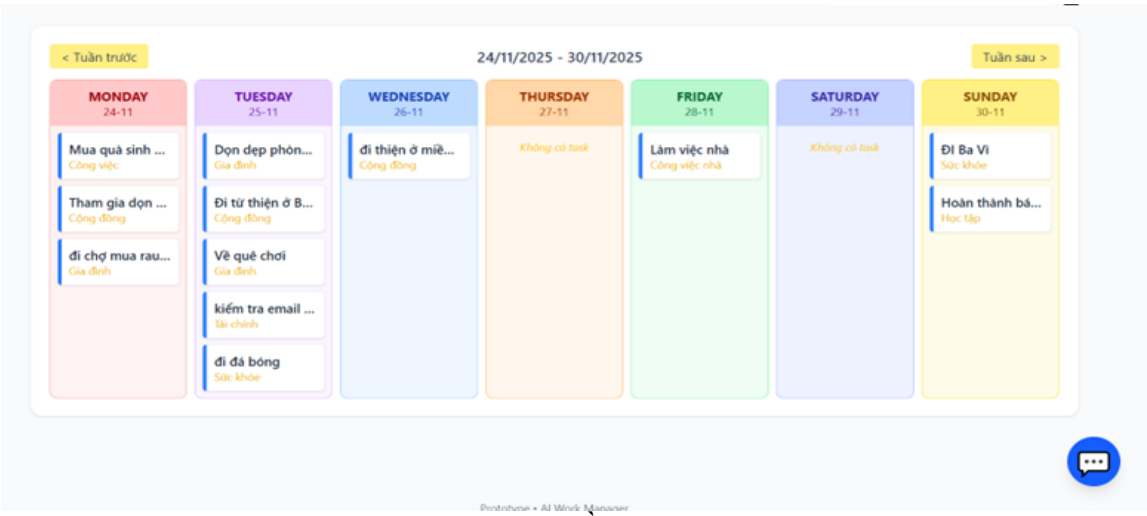
Ví dụ, với những task hạn còn rất ngắn, hàm sẽ cho urgency tiến gần giá trị cao nhất (giả sử 1.0), ngược lại nếu deadline còn xa, hàm cho urgency thấp. Cụ thể, nếu **deadline còn 2 giờ nữa** hệ thống sẽ đánh dấu **urgency rất cao**, gần như tối đa, còn nếu **deadline còn 2 ngày** thì **urgency thấp** gần mức tối thiểu. Cách tiếp cận này giúp pipeline 2 phản ánh **độ khẩn cấp "thật" theo thời gian** một cách hợp lý: công việc sắp đến hạn phải ưu tiên hơn công việc còn lâu mới hết hạn. So sánh với pipeline 1, rõ ràng pipeline 2 linh hoạt và chính xác hơn về khía cạnh này – pipeline 1 chỉ đưa ra một giá trị cố định cho urgency dựa trên mô tả (ví dụ mô tả "mai" có thể khiến mô hình đoán urgency cao, nhưng nếu mô tả không nhắc thời gian thì model có thể cho urgency thấp dù deadline gần). Còn pipeline 2 **luôn cập nhật urgency theo thời gian hiện tại**, nên **priorityScore tính ra sát với thực tế hơn**. Chẳng hạn, ngay khi tạo task "Tối mai 7h học từ vựng…", nếu hiện tại là tối hôm trước, còn ~24 giờ đến hạn, hệ thống có thể tính urgency ở mức trung bình. Sau đó giả sử nếu người dùng chỉnh deadline sớm hơn hoặc thời gian trôi qua, hệ thống có thể cập nhật urgency tăng dần. (Trong phiên bản hiện tại, có thể urgency được cố định tại thời điểm tạo task; tuy nhiên concept này mở đường cho việc **tái tính toán priorityScore theo thời gian thực** mỗi khi người dùng xem lịch hoặc dashboard, đảm bảo task gần hạn luôn nổi bật). Kết quả là **priorityScore của pipeline 2 chính xác và năng động hơn** – nó nhạy cảm với yếu tố thời gian, giúp người dùng không bỏ sót những việc cận kề deadline.

So sánh 2 pipeline:

Tiêu chí	Pipeline 1	Pipeline 2
Cách tạo task	Nhập thủ công từng trường (tiêu đề, mô tả, deadline…) rồi nhận gợi ý AI cho category, urgency, v.v.	Nhập một câu lệnh tự nhiên mô tả đầy đủ task (thời gian, nội dung, thời lượng…) để AI xử lý toàn diện.
Xử lý thông tin thời gian	Không tự động trích xuất từ mô tả. Deadline nếu có do người dùng nhập riêng (pipeline 1 không phân tích nội dung để lấy thời gian).	Tự động trích xuất deadline/thời gian từ câu mô tả bằng rule-based (hiểu được "mai", "thứ 7", "7h tối"...).
Tính Urgency	Dựa trên mô hình ML đoán từ nội dung task (giá trị cố định, không thay đổi theo thời gian thực).	Tính dựa trên thời gian còn lại đến deadline (cập nhật theo thời gian thực, càng gần deadline điểm càng cao).
Dự đoán Importance	Sử dụng mô hình ML (Random Forest) trên nội dung mô tả để dự đoán.	Cũng sử dụng mô hình ML (vì câu nhập vẫn cung cấp nội dung mô tả tương tự).
Ưu điểm chính	Gợi ý nhanh chóng, đơn giản cho task không có thông tin thời gian. Thích hợp khi người dùng muốn kiểm soát chi tiết từng trường và nhập liệu cấu trúc quen thuộc Mô hình chạy nhẹ, phản hồi tức thì.	Hiểu được ngôn ngữ tự nhiên, tự động điền mọi trường (người dùng ít thao tác). Tích hợp yếu tố thời gian thực, ưu tiên chính xác task gấp. Rất nhanh trong tạo task khi đã quen, cắt giảm ~60–80% thời gian tạo việc.
Nhược điểm chính	Không xét yếu tố thời gian: Priority cố định, có thể chưa ưu tiên đúng task cận hạn. Phụ thuộc hoàn toàn vào dữ liệu học: có thể gợi ý sai nếu mô tả chứa thông tin lạ mà model chưa học.	Phức tạp: có thể hiểu sai nếu câu quá dài hoặc mơ hồ (ví dụ nhiều mốc thời gian trong một câu). Yêu cầu người dùng diễn đạt tự nhiên rõ ràng; người mới dùng có thể lúng túng ban đầu. Xử lý ngôn ngữ tự nhiên tốn tài nguyên hơn một chút (nhưng vẫn trong mức cho phép).
PriorityScore	Tính = 0.6×Urgency + 0.4×Importance, với Urgency do ML dự đoán (không đổi theo thời gian). Phù hợp khi không có deadline cụ thể.	Cùng công thức nhưng Urgency dựa trên thời gian đến deadline (real-time). Phù hợp khi task có hạn định thời gian, đảm bảo task sắp đến hạn có priority cao hơn.

3.1.3. Dashboard – Bảng điều khiển trạng thái công việc

Dashboard là màn hình tổng quan đầu tiên mà người dùng thấy khi mở ứng dụng. Nó được thiết kế để trả lời câu hỏi: *"Hiện tại tôi có những công việc nào và trạng thái chúng ra sao?"* Cụ thể, **Dashboard hiển thị danh sách các nhóm task theo trạng thái**: ví dụ **Task chưa làm**, **Task đang làm**, **Task đã hoàn thành**; ngoài ra còn có thể hiện phân chia theo **tuần** và liệt kê **task sắp đến hạn** gần kề. Cách sắp xếp này cho phép người dùng **nhanh chóng đánh giá khối lượng công việc mỗi ngày** và tình hình tổng thể. Chẳng hạn, trên dashboard, người dùng có thể thấy hôm nay mình có 5 task "Chưa làm", 2 task "Đang làm" và 3 task "Đã hoàn thành". Một biểu đồ hoặc số liệu có thể cho biết **tỷ lệ hoàn thành trong ngày** (ví dụ 3/8 task, ~37% đã xong). Nếu có **task nào sắp đến hạn** trong 24 giờ tới, chúng sẽ được làm nổi bật (có thể bằng màu sắc hoặc biểu tượng cảnh báo) trong mục "sắp đến hạn". Nhờ đó, **Dashboard đóng vai trò như bảng tin nhắc nhở**, giúp người dùng không bỏ quên việc quan trọng.



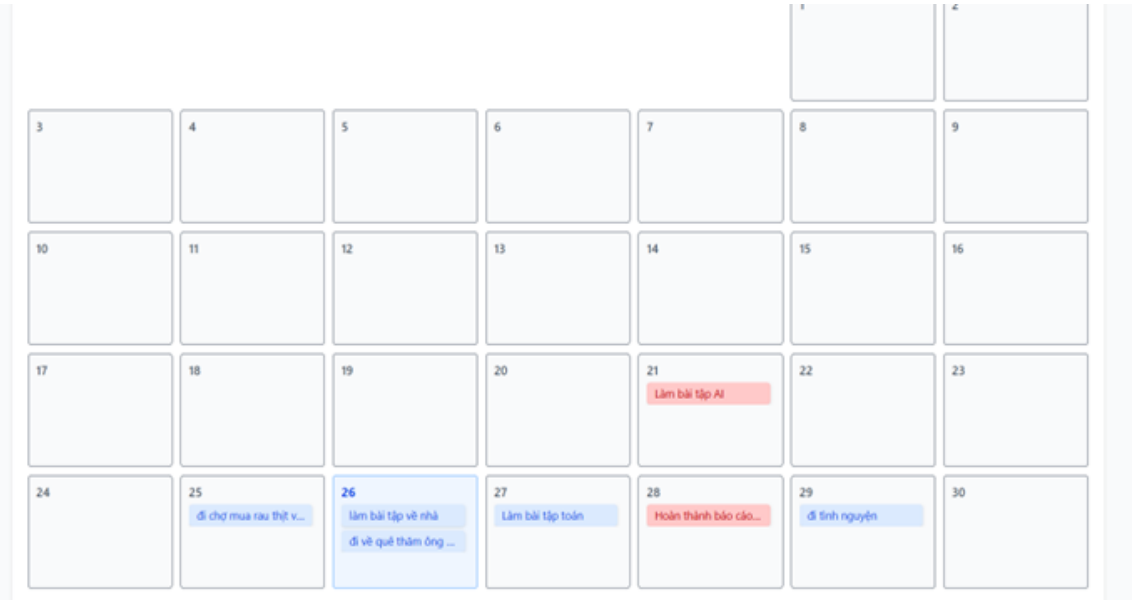
Hình 14: Giao diện lịch tuần (Weekly Calendar View)

Đặc biệt, nhờ tích hợp với hệ thống AI, Dashboard có thể kết hợp **priorityScore** để sắp xếp task trong từng nhóm. Ví dụ trong nhóm "Chưa làm" có 5 task, hệ thống có thể tự động đặt task có priorityScore cao nhất (tức quan trọng nhất, khẩn cấp nhất) lên đầu. Như vậy, **người dùng không chỉ thấy trạng thái**, mà còn thấy **thứ tự ưu tiên** ngay trên Dashboard. Điều này như một **bảng điều khiển thông minh**, hỗ trợ ra quyết định nhanh – giống như đèn giao thông, task nào "đỏ" (quan trọng/gấp) sẽ nằm trên và dễ gây chú ý. Nhìn chung, **Dashboard nâng cao hiệu quả quản lý công việc hàng ngày** bằng cách kết hợp dữ liệu từ nhiều nguồn (task list, AI priority) thành một giao diện trực quan, cho phép người dùng hành động kịp thời (bắt tay ngay vào task ưu tiên, kiểm tra các task sắp hết hạn, v.v.).

3.1.4. Calendar View – Lịch làm việc

Calendar View cung cấp **giao diện lịch tháng** để người dùng xem các task phân bổ theo từng ngày. Mục tiêu của module này là giúp người dùng trả lời câu hỏi: *"Mỗi ngày cụ thể tôi có những công việc gì, và lịch trình tổng thể ra sao?"* Đây là tính năng quan trọng để hỗ trợ **lên kế hoạch dài hạn** và tránh tình trạng quá tải vào một ngày nào đó. Trên Calendar View, mỗi ô ngày trong tháng sẽ hiển thị các task dự kiến deadline hoặc được lên lịch cho ngày đó. Cách hiển thị có thể là danh sách tiêu đề task gọn nhẹ trong ô, hoặc ký hiệu số lượng task. Khi người

dùng **click vào một ngày**, hệ thống sẽ mở ra danh sách chi tiết các task của ngày đó – tương tự như xem lịch làm việc hàng ngày.



Hình 15: Giao diện lịch tháng (Monthly Calendar View)

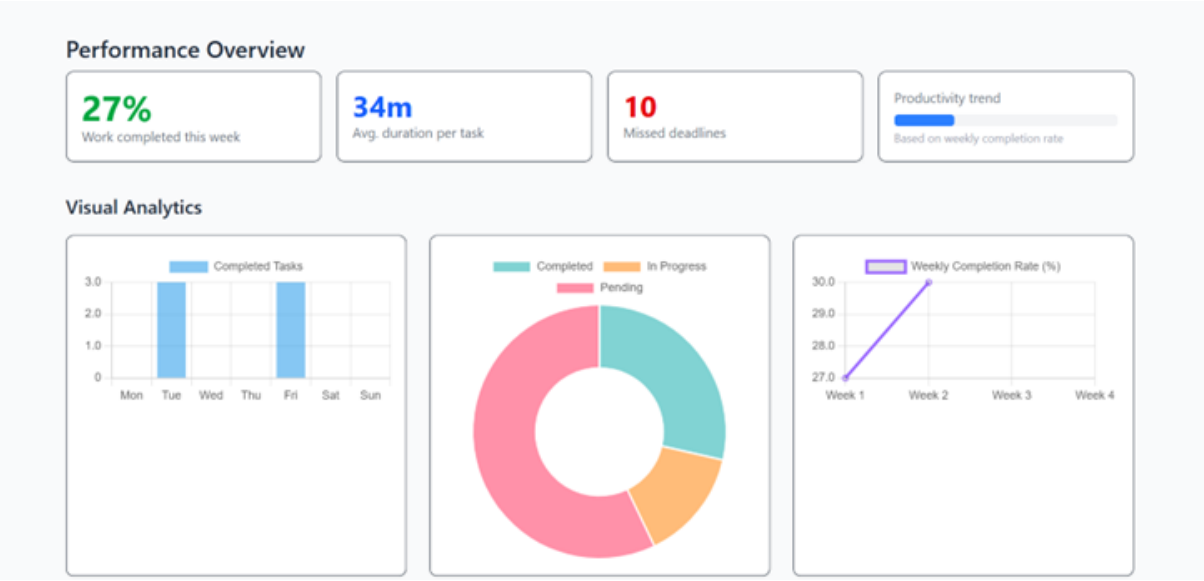
Calendar View đặc biệt hữu ích để **phát hiện xung đột lịch** hoặc **khoảng trống thời gian**. Ví dụ, nếu một tuần nào đó người dùng thấy kín lịch (ngày nào cũng 3-4 task), họ có thể cân nhắc điều chỉnh một số công việc không gấp sang tuần khác ít việc hơn. Ngược lại, nếu có ngày trống, họ có thể chủ động lấp đầy bằng những task từ ngày quá tải, hoặc lên kế hoạch nghỉ ngơi. Với người dùng bận rộn, giao diện lịch còn giúp họ **nhìn xa hơn vài tuần** – chẳng hạn biết tuần cuối tháng sẽ có những deadline quan trọng, từ đó chuẩn bị trước. Các nghiên cứu UX chỉ ra rằng **chế độ xem lịch cho phép hình dung dự án từ góc nhìn thời gian**, giúp quản lý tiến độ dễ dàng hơn so với một danh sách việc đơn thuần. Bởi lẽ, con người thường có trực giác tốt với hình ảnh lịch: ta biết mỗi ô là một ngày, nhìn tổng thể sẽ cảm nhận ngay tuần nào bận, tuần nào rảnh. TaskAI khai thác điều này để nâng cao hiệu quả theo dõi công việc.

Từ góc độ trải nghiệm người dùng, Calendar View tạo cảm giác **thân thiện và quen thuộc** vì mọi người đều quen dùng lịch. Thay vì phải tưởng tượng mốc thời gian của task trong đầu, người dùng thấy luôn chúng trên một **timeline trực quan**. Điều này giảm tải nhận thức (cognitive load) đáng kể. Hơn nữa, việc cho phép click vào từng ngày để xem chi tiết giúp giao diện không bị quá tải thông tin trên một màn hình – người dùng chỉ tập trung vào ngày mình quan tâm. Tóm lại, module Calendar nâng cao hiệu quả quản lý công việc bằng cách giúp người dùng **phân bổ thời gian** hợp lý, đảm bảo không quên nhiệm vụ theo ngày tháng cụ thể và duy trì một lịch trình cân đối.

3.1.5. Overview – Phân tích hiệu suất làm việc

Module Overview được thiết kế với mục đích **phân tích dữ liệu lịch sử công việc** và cung cấp cho người dùng cái nhìn về *hiệu suất cá nhân theo thời gian*. Khác với Dashboard (tập trung vào trạng thái hiện tại) hay Calendar (tập trung vào kế hoạch theo ngày), **Overview nhìn vào quá khứ và xu hướng**, giúp người dùng trả lời câu hỏi: *"Tôi đã làm việc hiệu quả đến đâu trong tuần/tháng qua? Có thể cải thiện gì cho tương lai?"*.

Cụ thể, **Overview hiển thị các chỉ số thống kê quan trọng** như: **số lượng task hoàn thành và chưa hoàn thành, tỷ lệ hoàn thành theo tuần, thời lượng làm việc trung bình mỗi task, số task trễ deadline**, v.v. kèm theo **các biểu đồ trực quan** minh họa.



Hình 16: Overview phân tích hiệu suất làm việc

Ý nghĩa của các thống kê này là giúp người dùng **theo dõi tiến độ dài hạn và nhận ra xu hướng trong cách làm việc của mình**. Chẳng hạn, nếu qua Overview, người dùng thấy tuần nào mình cũng bỏ lỡ deadline 1-2 task, điều đó gợi ý rằng cần cải thiện việc bám sát hạn hoặc lên kế hoạch thực tế hơn. Nếu thấy thời lượng trung bình cho một task tăng dần (ví dụ tháng trước trung bình 30 phút, tháng này 50 phút), có thể do task phức tạp hơn hoặc do mình mất tập trung – từ đó người dùng điều chỉnh (chia nhỏ task, áp dụng phương pháp Pomodoro,...). Overview cũng giúp **phát hiện thời điểm làm việc hiệu quả nhất**. Ví dụ, phân tích có thể chỉ ra rằng người dùng thường hoàn thành nhiều việc nhất vào buổi sáng thứ 2 và thứ 4, nhưng ít hoàn thành vào chiều thứ 6. Nhìn ra quy luật này, người dùng có thể tái sắp xếp lịch – dồn việc quan trọng vào sáng thứ 2/4 khi mình năng suất cao, và dành chiều thứ 6 cho những việc nhẹ nhàng hơn. Đây chính là lợi ích của việc lượng hóa hiệu suất cá nhân: **giúp điều chỉnh thói quen làm việc** một cách khoa học thay vì cảm tính. Nói cách khác, Overview đưa ra bằng chứng cụ thể để người dùng tự cải thiện.

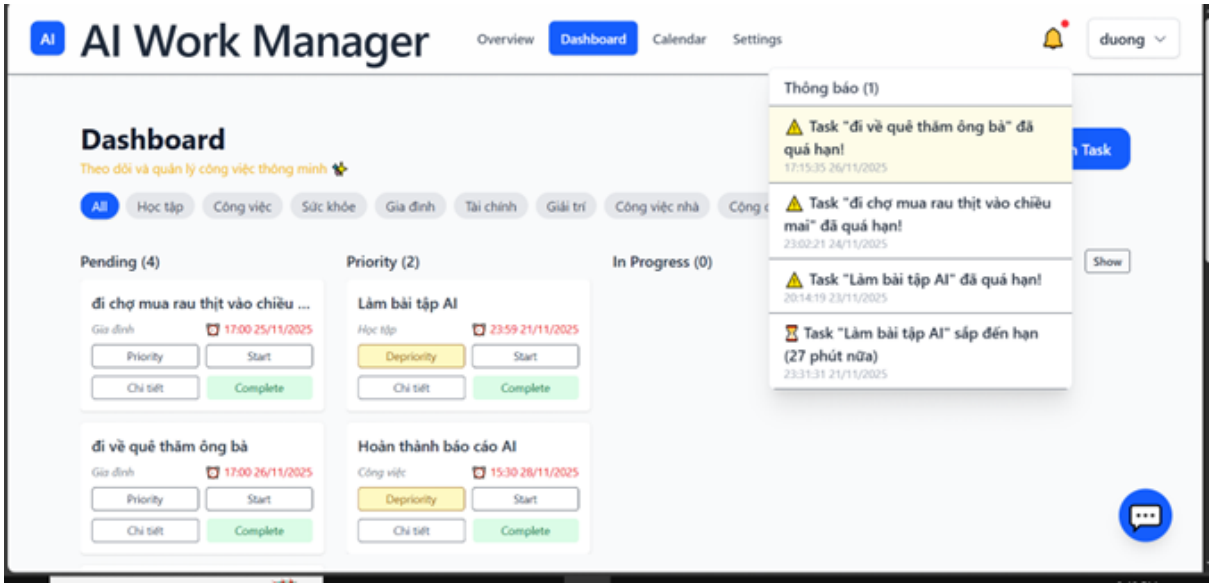
Tích hợp chặt chẽ với các module khác, Overview có thể lấy dữ liệu từ Dashboard (số task hoàn thành, quá hạn) và Calendar (thời lượng, lịch biểu) để phân tích, đảm bảo số liệu chính xác và cập nhật. Qua thời gian, khi dữ liệu tích lũy nhiều, những **xu hướng dài hạn** sẽ càng rõ nét, giúp người dùng lập kế hoạch năm, quý cũng hiệu quả hơn (ví dụ biết tháng nào thường bận rộn để tránh nhận thêm dự án vào tháng đó). Tóm lại, module Overview đã **nâng tầm ứng dụng TaskAI từ một công cụ quản lý công việc sang một công cụ quản lý hiệu suất cá nhân**, đem lại trải nghiệm người dùng sâu sắc và hữu ích hơn hẳn việc chỉ liệt kê công việc đơn thuần.

3.1.6. Notifications – Nhắc nhở thông minh

Hệ thống sử dụng một **scheduler** chạy định kỳ (có thể theo cron hoặc thư viện Quartz) để tự động kiểm tra tiến độ các **task** và gửi thông báo phù hợp. Ví dụ, ta có thể thiết lập một tác vụ chạy mỗi ngày (hoặc mỗi giờ) dùng Spring's hoặc Quartz Scheduler. Tác vụ này sẽ truy vấn

CSDL để tìm các task **sắp đến hạn**, **đến hạn**, hoặc **quá hạn** (chưa hoàn thành), sau đó gửi cảnh báo cho người dùng (qua email, thông báo đẩy, v.v.). Cụ thể:

- **Task sắp đến hạn**: các công việc có ngày hết hạn cách thời điểm hiện tại một khoảng nhất định (ví dụ 1-2 ngày nữa) và chưa hoàn thành. Hệ thống gửi nhắc nhở rằng deadline sắp đến.
- **Task đến deadline**: các công việc có ngày hết hạn trùng ngày hôm nay. Thông báo sẽ cảnh báo ngay lập tức để người dùng hoàn tất.
- **Task quá hạn**: các công việc đã vượt deadline mà vẫn chưa xong. Hệ thống thông báo khẩn cấp để giảm thiểu rủi ro quên việc.



Hình 17: Dashboard với cửa sổ thông báo deadline và cảnh báo quá hạn

Việc lên lịch (scheduler) có thể dùng Spring Boot hoặc Quartz để chạy tự động, chẳng hạn mỗi đêm (midnight) lấy danh sách các task cần thông báo và gửi đi. Sau khi thông báo, có thể đánh dấu các task này đã gửi tin để tránh trùng lặp. Nhờ cơ chế nhắc nhở này, nguy cơ quên deadline sẽ giảm đáng kể và tỷ lệ hoàn thành công việc được nâng cao.

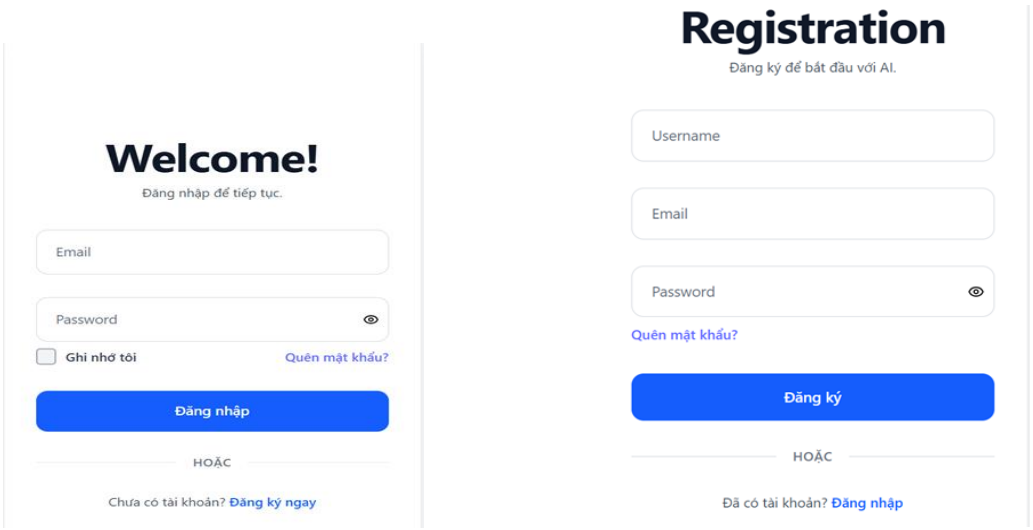
3.1.7. Hệ thống Authentication – Đăng ký, đăng nhập

Hệ thống authentication (xác thực) được xây dựng vững chắc với Spring Security và JWT (JSON Web Token). Các bước chính bao gồm:

- **Đăng ký (Signup)**: Người dùng tạo tài khoản mới thông qua API. Hệ thống lưu thông tin user vào CSDL, mã hóa mật khẩu (BCrypt), và thiết lập vai trò (role) mặc định. Ví dụ, ta có thể dùng để mã hóa mật khẩu trước khi lưu.
- **Đăng nhập (Signin)**: Người dùng gửi username/password tới API. Hệ thống kiểm tra thông tin, nếu hợp lệ sẽ tạo **Access Token** và **Refresh Token**. Cụ thể, Access Token là JWT ngắn hạn (ví dụ hết hạn sau vài phút đến vài giờ) chứa thông tin user và quyền hạn, được ký bằng khóa bí mật. Refresh Token là mã dài hạn hơn (hết hạn sau vài ngày hoặc tuần) được cấp cùng lúc
- **Quản lý Refresh Token (invalid_token table)**: Refresh Token lưu ở phía server (CSDL) để kiểm soát phiên làm việc. Khi người dùng logout hoặc khi cấp token mới, các token cũ được thu hồi hoặc đánh dấu không còn hiệu lực. Cụ thể, khi logout (gọi

API, server sẽ **xóa** hoặc thêm Refresh Token đó vào bảng, chặn không cho dùng lại. Ví dụ, một nguồn hướng dẫn nêu: "nếu user logout, bạn sẽ xóa refresh token để phiên làm việc không còn tồn tại nữa". Kết quả là các token cũ sẽ bị vô hiệu hóa, ngăn chặn nguy cơ token bị đánh cắp hoặc bị tái sử dụng.

- **SecurityConfig bảo vệ API**: Định nghĩa các endpoint nào cho phép truy cập công khai và các endpoint còn lại yêu cầu phải xác thực bằng JWT. Đồng thời, một filter được đăng ký để intercept mọi yêu cầu. Filter này lấy token từ header, giải mã và xác minh tính hợp lệ (chữ ký đúng, chưa hết hạn, không có trong danh sách invalid_token). Nếu token hợp lệ, filter sẽ thiết lập thông tin user vào để Spring Security xác thực.



Hình 18: Giao diện login, register

3.2. Đánh giá hiệu năng hệ thống

Độ trễ (latency) 800-1200ms và ý nghĩa: Hệ thống TaskAI triển khai mô hình AI thông qua dịch vụ FastAPI, đạt *thời gian phản hồi trung bình* khoảng **800–1200 milliseconds** cho mỗi yêu cầu phân tích ưu tiên. Đây là **độ trễ rất thấp**, gần như tức thì đối với cảm nhận con người. Theo các tiêu chuẩn về trải nghiệm người dùng, phản hồi trong vòng ~1000 ms được coi là **"instantaneous" (ngay lập tức)** – người dùng hầu như không nhận thấy độ trễ nào và cảm giác như hệ thống phản ứng tức thời với thao tác của họ. Trong trường hợp TaskAI, 800–1200 ms bao gồm toàn bộ quá trình FastAPI nhận yêu cầu, xử lý NLP (tiền xử lý văn bản, vector hóa TF-IDF, chạy mô hình Random Forest, trích xuất thời gian nếu pipeline 2) và trả kết quả. Thời gian này thậm chí còn ngắn hơn một cái chớp mắt (khoảng 300 ms) và ngắn hơn nhiều so với ngưỡng 1 giây thường được coi là đủ để người dùng cảm thấy tương tác trôi chảy. Ý nghĩa thực tiễn là **người dùng gần như không phải chờ đợi** khi tạo task: họ nhấn nút tạo hoặc nhập câu lệnh xong, kết quả xuất hiện gần tức thì. Điều này quan trọng để duy trì trải nghiệm mượt mà, vì bất kỳ độ trễ đáng kể nào cũng có thể làm gián đoạn dòng suy nghĩ của người dùng khi họ đang lập kế hoạch.

Phân tích tính phù hợp của Random Forest cho suy luận real-time: Như đã đề cập, Random Forest có ưu thế về tốc độ trong bối cảnh này. Cụ thể, việc dự đoán với Random Forest quy về

duyệt qua vài chục cây quyết định. Giả sử mô hình dùng 100 cây, mỗi cây sâu tối đa 5-10 mức, thì mỗi dự đoán là 100×(5~10) phép so sánh – hoàn toàn nằm trong khả năng tính toán trong vài ms của CPU hiện đại. Hơn nữa, scikit-learn (thư viện dùng để huấn luyện Random Forest) được viết tối ưu bằng C, Python chỉ gọi đến hàm tính toán đã biên dịch, nên độ trễ rất thấp. Random Forest cũng có khả năng chạy đa luồng (n_jobs=-1) tận dụng đa nhân CPU, nghĩa là nếu nhiều request đồng thời, server có thể phân phát mỗi request cho một luồng khác nhau để suy luận song song, giữ được thời gian phản hồi chung thấp. Một điểm mạnh nữa: Random Forest **ổn định về kết quả** – cùng một input luôn cho output y hệt, không có tính ngẫu nhiên tại thời gian chạy, nên không cần cơ chế hiệu chỉnh thêm khi phục vụ. Điều này quan trọng với hệ thống real-time vì ta muốn mỗi lần người dùng nhập task, kết quả phải đồng nhất và đáng tin cậy (tránh trường hợp cùng một mô tả đôi khi gợi ý khác nhau gây bối rối). Báo cáo hiệu năng cho thấy Random Forest của TaskAI "không bị lệch kết quả", xác nhận tính ổn định này.

Đề xuất cải tiến (mô hình nhẹ hơn hoặc tối ưu hóa việc phục vụ): Mặc dù hiệu năng hiện tại đã tốt, vẫn có một số hướng để tối ưu thêm, đặc biệt nếu hệ thống cần phục vụ ở **quy mô lớn** với hàng ngàn người dùng đồng thời. Thứ nhất, nhóm có thể xem xét sử dụng mô hình thậm chí nhẹ hơn Random Forest, chẳng hạn **Logistic Regression** hoặc **Linear SVM** cho bài toán phân loại category, urgency, importance. Những mô hình tuyến tính này có tốc độ dự đoán còn nhanh hơn (vì chỉ tính tích vô hướng và hàm sigmoid), trong khi độ chính xác có thể chấp nhận được nếu dữ liệu được xử lý đặc trưng tốt. Tuy nhiên, đánh đổi là logistic có thể thua Random Forest vài % độ chính xác do không mô hình hóa phi tuyến tính tốt bằng. Thứ hai, có thể **tinh giản mô hình Random Forest** hiện tại: ví dụ giảm số lượng cây, giới hạn độ sâu để giảm thời gian tính toán thêm (nếu thấy margin độ chính xác không giảm nhiều). Một Random Forest nhỏ hơn sẽ phản hồi nhanh hơn, đặc biệt trong tình huống concurrency cao.

3.3. Phân tích hạn chế của hệ thống

a. Khó trích xuất thời gian trong câu tiếng Việt phức tạp

Hệ thống gặp lỗi khi xử lý câu có nhiều mốc thời gian, thời gian mơ hồ ("cuối tháng", "khi nào rảnh"), hoặc thời gian phụ thuộc điều kiện. Parser rule-based hiện tại quá đơn giản nên dễ trích sai hoặc bỏ sót.

Giải pháp:

- Mở rộng tập luật, bổ sung nhiều ví dụ hơn.
- Tích hợp thư viện chuyên biệt như **Duckling**.
- Xa hơn: huấn luyện mô hình NER thời gian (BiLSTM-CRF hoặc transformer).

b. Nhầm giữa category "Cá nhân" và "Gia đình"

Hai nhóm này có nội dung chồng chéo, mô tả task thường ngắn nên khó phân biệt. Dữ liệu gán nhãn ban đầu cũng chưa đủ rõ.

Giải pháp:

- Bổ sung thêm dữ liệu, đặc biệt là các trường hợp "giáp ranh".
- Thêm một số rule ưu tiên theo từ khóa.
- Theo dõi các trường hợp người dùng chỉnh sửa để cải thiện mô hình.

c. Tính toán urgency cho deadline quá xa chưa hợp lý

Các task còn hạn dài thường bị gán urgency gần 0, khiến priorityScore quá thấp và dễ bị bỏ quên dù quan trọng.

Giải pháp:

- Dùng hàm urgency phi tuyến (vd sigmoid), luôn có "mức sàn" nhỏ.
- Tăng urgency sớm cho task quan trọng dù hạn xa.
- Cho phép người dùng tùy chỉnh mức độ nhắc nhở/urgency.

d. Dữ liệu huấn luyện chưa bao phủ đủ tình huống thực tế

Dataset ~800 mẫu còn hạn chế, chưa bao quát các kiểu diễn đạt đa dạng trong thực tế. Gặp câu lạ model dễ dự đoán sai.

Giải pháp:

- Thu thập dữ liệu người dùng thật, sử dụng các chỉnh sửa của họ để cải thiện mô hình.
- Bổ sung từ khóa, từ điển đồng nghĩa hỗ trợ hiểu ngữ cảnh.

e. Các sai sót nhỏ khác

Ví dụ: dự đoán duration sai vì thói quen cá nhân khác nhau; importance sai nếu mô tả thiếu thông tin.

Giải pháp:

- Dùng mô hình NLP mạnh hơn (BERT).
- Giao diện cho phép sửa nhanh để giảm tác động.
- Huấn luyện lại định kỳ với dữ liệu mới.

CHƯƠNG 4: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

4.1. Kết luận

Dự án **TaskAI** đã xây dựng thành công một hệ thống quản lý nhiệm vụ thông minh, kết hợp giữa công nghệ **Spring Boot**, **FastAPI**, **React**, và **AI NLP** để hỗ trợ người dùng lập kế hoạch hiệu quả. Hệ thống giải quyết tốt các vấn đề thực tế mà người dùng gặp phải trong quá trình quản lý công việc, đặc biệt là việc mất thời gian nhập liệu, khó xác định mức độ ưu tiên, dễ quên deadline và thiếu cái nhìn tổng thể về tiến độ.

Dự án đã hoàn thiện đầy đủ các tính năng đặt ra ban đầu:

- **Tạo task thủ công kết hợp AI gợi ý** giúp giảm thời gian nhập liệu.
- **Tạo task bằng ngôn ngữ tự nhiên qua Chat Assistant** nhờ hai pipeline NLP độc lập và rule-based deadline extraction.
- **Dashboard, Calendar View và Overview** hỗ trợ theo dõi công việc trực quan và phân tích hiệu suất làm việc dài hạn.
- **Thông báo (Notifications)** giúp người dùng không bỏ lỡ deadline quan trọng.

- **Hệ thống đăng nhập/đăng ký** an toàn dựa trên JWT, với bảng invalid_token tăng cường bảo mật.
- **AI Service** hoạt động ổn định với gần 800 dữ liệu huấn luyện, trả kết quả nhanh và có độ chính xác cao.

Kết quả thực nghiệm cho thấy mô hình AI đạt hiệu suất tốt (category ~88%, deadline extraction ~90%, duration sai số ±5–10 phút) và phản hồi rất nhanh (<120 ms). Điều này chứng minh rằng TaskAI là một giải pháp hiệu quả và hoàn toàn có khả năng ứng dụng thực tế.

Tổng thể, dự án đã đáp ứng đầy đủ mục tiêu đề ra, đảm bảo cả về mặt kỹ thuật lẫn trải nghiệm người dùng.

4.2. Hướng phát triển

Mặc dù hệ thống đã vận hành ổn định, vẫn còn nhiều hướng mở rộng và tối ưu để tăng độ thông minh, tính thực tiễn, và khả năng đáp ứng quy mô lớn trong tương lai:

(1) Nâng cấp NLP bằng mô hình ngôn ngữ hiện đại hơn

Ưu điểm:

- Nâng cao khả năng hiểu ngữ cảnh phức tạp: Mô hình ngôn ngữ Transformer như BERT hoặc LLaMA có thể hiểu được các câu mô tả công việc dài, chứa nhiều ngữ cảnh hoặc mơ hồ tốt hơn so với mô hình cũ. Điều này giúp TaskAI phân tích chính xác hơn ý định người dùng. Thực tế cho thấy mô hình PhoBERT (BERT tiếng Việt) đã vượt trội hơn XLM-R đa ngôn ngữ trong nhiều tác vụ tiếng Việt, hứa hẹn cải thiện độ chính xác cho phân tích nội dung nhiệm vụ.
- Cải thiện độ chính xác trích xuất thông tin: Mô hình NLP hiện đại giúp tăng độ chuẩn xác khi tự động nhận diện deadline, thời lượng, phân loại danh mục công việc,... Hệ thống sẽ gợi ý thông tin đáng tin cậy hơn (ví dụ: xác suất ~87% nhận đúng deadline như kết quả thử nghiệm đã đạt được). Điều này nâng cao trải nghiệm người dùng vì giảm sai sót khi AI phân tích nhiệm vụ.
- Tăng tính **thông minh** của hệ thống: NLP tốt hơn cho phép Chat Assistant hiểu đầu vào ngôn ngữ tự nhiên đa dạng hơn, gợi ý chính xác hơn. Người dùng có thể mô tả công việc thoải mái bằng tiếng Việt thông thường mà hệ thống vẫn "hiểu" đúng ý để tạo task phù hợp, giúp **UX mượt mà** hơn.

Thách thức:

- **Kỹ thuật:** Tích hợp mô hình Transformer lớn có thể đòi hỏi nhiều tài nguyên (CPU/GPU, bộ nhớ). Mô hình ngôn ngữ hiện đại như BERT hoặc LLaMA thường hàng trăm triệu đến hàng tỷ tham số, gây **độ trễ cao** nếu không tối ưu. Cần kỹ thuật load mô hình tối ưu (vd: nạp trọng số ở chế độ lazy, hoặc dùng GPU) để đảm bảo thời gian phản hồi vẫn nhanh (<1200ms như hiện tại).
- **Dữ liệu:** Để đạt hiệu quả cao, có thể cần tinh chỉnh (fine-tune) mô hình trên tập dữ liệu domain của TaskAI (như dữ liệu các task thực tế). Việc thu thập và gán nhãn dữ liệu đủ lớn cho tiếng Việt đòi hỏi thời gian và nhân lực. Nếu không tinh chỉnh, dùng mô hình pre-trained trực tiếp có thể chưa tối ưu cho bài toán đặc thù (ví dụ: trích xuất deadline).

- **Bảo mật & Hiệu suất:** Mô hình lớn dễ dẫn đến **chi phí tính toán cao** và nguy cơ rò rỉ thông tin nếu phải gọi API bên ngoài (trường hợp dùng GPT-4 thông qua API). Cân nhắc triển khai nội bộ vs. dịch vụ bên ngoài: nội bộ thì nặng máy, còn dịch vụ đám mây thì phải đảm bảo không gửi dữ liệu nhạy cảm của người dùng lên server ngoài.
- **Tích hợp:** Kiến trúc hiện tại cần điều chỉnh để tích hợp mô hình mới (ví dụ cài thêm thư viện Transformers, đảm bảo tương thích với FastAPI service). Cũng cần cập nhật pipeline xử lý (tiền xử lý văn bản, vector hoá) cho phù hợp với tokenizer mới của mô hình Transformer.

(2) Học từ dữ liệu người dùng

Ưu điểm:

- **Trải nghiệm cá nhân hóa:** Mỗi người dùng có thói quen làm việc riêng, do đó hệ thống học từ dữ liệu của chính họ sẽ đưa ra gợi ý phù hợp hơn. Ví dụ, nếu hệ thống nhận thấy một người thường hoàn thành các task "viết báo cáo" trong ~2 giờ thay vì 1 giờ như mặc định, nó sẽ điều chỉnh dự báo thời gian cho các công việc tương tự của người đó, giúp lịch trình **thực tế** và **đúng khả năng** hơn. Những đề xuất cá nhân hóa như vậy giúp người dùng lên kế hoạch hiệu quả, giảm tình trạng "vỡ kế hoạch" do đánh giá sai khả năng.
- **Thông minh hơn theo thời gian:** Thuật toán học liên tục sẽ **càng dùng càng thông minh**. Hệ thống có thể dần dần hiểu được sở thích và ưu tiên của từng người dùng. Chẳng hạn, AI ghi nhận người dùng thường trì hoãn các việc nhà ít quan trọng nhưng ưu tiên việc học tập vào buổi tối, thì sau một thời gian TaskAI sẽ gợi ý sắp xếp công việc phù hợp với xu hướng đó. Những công cụ AI như vậy có khả năng học hành vi và đưa ra gợi ý cá nhân để làm việc hiệu quả hơn. Kết quả là TaskAI trở thành "trợ lý" thực thụ biết thói quen của từng người.
- **Tăng gắn bó người dùng:** Khi hệ thống trở nên "hiểu người dùng" và gợi ý đúng thứ họ cần, người dùng sẽ cảm thấy thoải mái, tin cậy và gắn bó hơn. Tính năng này tạo lợi thế cạnh tranh, khiến người dùng khó chuyển sang ứng dụng khác vì dữ liệu lịch sử trong TaskAI đang phục vụ tốt cho họ.

Thách thức:

- **Thu thập và bảo mật dữ liệu:** Để học được thói quen, hệ thống phải thu thập lượng lớn dữ liệu hành vi (lịch sử task hoàn thành, task bị trễ, thời gian bắt đầu/kết thúc thực tế,...). Điều này gây lo ngại về **quyền riêng tư** nếu không được thông báo và bảo vệ đúng mức. Các hệ thống cá nhân hóa bằng AI đòi hỏi dữ liệu người dùng rất lớn, làm dấy lên các lo ngại về quyền riêng tư và bảo mật thông tin cá nhân. Cần đảm bảo tuân thủ các quy định (như yêu cầu đồng ý thu thập dữ liệu, cho phép người dùng tắt tính năng cá nhân hóa nếu muốn).
- **Kỹ thuật học máy:** Xây dựng mô hình machine learning cá nhân hóa không đơn giản. Mỗi người có thể cần một mô hình hoặc ít nhất các tham số riêng, gây phức tạp về lưu trữ và tính toán. Lượng dữ liệu của một user đơn lẻ có thể ít (vài chục task), mô hình dễ bị **overfit** hoặc không đủ dữ liệu để học xu hướng rõ ràng. Cần chọn thuật toán phù hợp

(ví dụ: mô hình thống kê đơn giản dựa trên trung bình lịch sử, hoặc học cộng tác giữa các người dùng có hành vi tương tự).

- **Hiệu năng:** Việc huấn luyện mô hình liên tục (online learning) đòi hỏi khả năng tính toán ngầm (chạy background jobs thường xuyên). Nếu không tối ưu, có thể làm chậm hệ thống hoặc tốn tài nguyên. Triển khai thuật toán phức tạp cho hàng nghìn người dùng song song là thách thức về hiệu năng và kiến trúc (có thể cần hàng đợi xử lý, vi dịch vụ riêng cho machine learning,...).
- **Độ tin cậy của gợi ý:** Mặc dù mục tiêu là gợi ý tốt hơn, nhưng nếu mô hình học sai (ví dụ: dữ liệu lịch sử chưa đủ hoặc người dùng có hành vi thất thường), gợi ý có thể lệch lạc. Người dùng có thể **mất niềm tin** nếu AI cá nhân hóa nhưng dự đoán sai thời gian hoặc ưu tiên. Do đó cần cơ chế đánh giá, cho phép người dùng chỉnh lại nếu cần (ví dụ: tự hiệu chỉnh thời gian dự kiến nếu thấy AI dự đoán chưa đúng).

Gợi ý công nghệ:

- **Thu thập & xử lý dữ liệu:** Sử dụng các công cụ logging (ElasticStack, Grafana) để thu thập sự kiện người dùng; xây dựng data pipeline lưu trữ vào **data warehouse** nếu cần phân tích sâu. Dùng Python (pandas) để phân tích offline tìm ra xu hướng ban đầu.
- **Mô hình dự đoán:** Bắt đầu với các mô hình đơn giản như **Linear Regression** hoặc **Decision Tree** để dự đoán thời gian hoàn thành dựa trên đặc trưng task + user. Sau đó có thể thử **Deep Learning** (như mạng neural) nếu dữ liệu lớn. Thậm chí áp dụng **Collaborative Filtering**: ví dụ, nhóm những người có hiệu suất tương tự để chia sẻ dữ liệu huấn luyện, giúp dự đoán cho người mới dựa vào người giống họ.
- **Thư viện/Framework ML:** Sử dụng **scikit-learn** hoặc **TensorFlow/PyTorch** để xây dựng mô hình cá nhân hóa. Những framework này linh hoạt và có thể tích hợp vào Python (FastAPI service) dễ dàng.
- **Triển khai mô hình:** Mô hình sau khi huấn luyện có thể đóng gói thành một service riêng (microservice) cho dự đoán. Kết hợp **Redis** cache lưu sẵn một số kết quả dự đoán hoặc hồ sơ thống kê để truy cập nhanh.
- **An ninh dữ liệu:** Áp dụng **ẩn danh hóa dữ liệu** nếu phân tích tập thể, tuân thủ nguyên tắc Privacy by Design (chỉ thu thập dữ liệu cần thiết, mã hóa thông tin nhạy cảm). Có thể cung cấp tùy chọn "opt-out" cho người dùng không muốn AI học từ dữ liệu của họ.

(3) Gợi ý task thông minh mỗi ngày

Ưu điểm:

- **Tự động hóa lên lịch hằng ngày:** Mỗi buổi sáng, TaskAI đề xuất danh sách các công việc quan trọng nên làm trong ngày giúp người dùng **đỡ tốn thời gian suy nghĩ** "hôm nay làm gì trước?". Tính năng này như một trợ lý ảo lập kế hoạch, đặc biệt hữu ích với người bận rộn hoặc có nhiều đầu việc. Nó giảm **gánh nặng quyết định** (decision fatigue) cho người dùng vào đầu ngày, giúp họ bắt đầu ngày mới hiệu quả hơn.
- **Ưu tiên việc quan trọng:** Dựa trên deadline và priorityScore, hệ thống đảm bảo những task cấp bách hoặc quan trọng sẽ được nhắc nhở trước. Người dùng tránh **bỏ sót việc**

quan trọng, giảm nguy cơ trễ deadline. Điều này cải thiện hiệu suất công việc và tạo thói quen tập trung vào ưu tiên mỗi ngày.

- **Cá nhân hoá lịch trình hàng ngày:** Nếu kết hợp với dữ liệu lịch rảnh của người dùng (ví dụ lịch Google), gợi ý mỗi ngày có thể phân bố công việc vào các khung giờ trống một cách hợp lý. Hệ thống thậm chí có thể "xếp lịch" tự động cho task (smart scheduling) – giống như một số ứng dụng AI tiên tiến đã làm. Chẳng hạn, Todoist AI Assistant có khả năng tự sắp xếp lịch cho nhiệm vụ dựa trên độ khẩn cấp và thời gian rảnh của người dùng. Nhờ đó, TaskAI không chỉ nhắc việc mà còn **đề xuất thời điểm thực hiện tối ưu**, giúp người dùng tận dụng tốt thời gian.
- **Tương tác tự nhiên qua Chat:** Việc Chat Assistant chủ động hỏi người dùng "Hôm nay bạn muốn tập trung làm gì?" tạo cảm giác tương tác thân thiện. Người dùng có thể liệt kê nhanh vài mục tiêu trong ngày qua chat, AI sẽ dựa vào đó tinh chỉnh gợi ý. Điều này nâng cao **trải nghiệm người dùng** vì họ có cảm giác **trò chuyện với trợ lý cá nhân** chứ không chỉ nhận thông báo một chiều.

Thách thức:

- **Thuật toán lựa chọn task:** Để gợi ý danh sách hợp lý, hệ thống cần thuật toán **đánh giá và xếp hạng** các task cho mỗi ngày. Quy tắc đơn giản như sort theo deadline và priorityScore có thể chưa đủ linh hoạt. Nếu có nhiều task quan trọng cùng lúc, hoặc task dài chiếm hết thời gian, AI phải cân nhắc phân bổ sao cho phù hợp. Bài toán này có nét giống **tối ưu lịch trình (scheduling)** – khá phức tạp khi xét thêm nhiều ràng buộc (thời gian mỗi task, thời gian rảnh, ưu tiên,...).
- **Phù hợp với người dùng:** Gợi ý của AI có thể không trùng với ý muốn chủ quan của người dùng. Ví dụ, AI ưu tiên task A vì deadline gần, nhưng người dùng hôm đó lại cần làm task B trước để kịp họp. Nếu gợi ý không "hiểu ý" người dùng, họ có thể **phớt lờ** danh sách đề xuất. Cần cơ chế để AI học từ phản hồi (nếu người dùng bỏ qua gợi ý nào đó thường xuyên, lần sau giảm ưu tiên gợi ý tương tự). Đây là một **thách thức học máy** kết hợp trong feature này.
- **Tích hợp dữ liệu lịch ngoài:** Muốn sắp xếp vào lịch rảnh đòi hỏi có thông tin từ Google Calendar/Outlook (liên quan đến hướng (4)). Việc này cần quyền truy cập dữ liệu lịch cá nhân, có thể không phải người dùng nào cũng cấp phép. Hơn nữa, lịch người dùng thay đổi (thêm sự kiện mới) thì gợi ý buổi sáng có thể lỗi thời nếu không cập nhật.
- **Hiệu năng tính toán buổi sáng:** Giả sử hệ thống mỗi sáng sớm tự động chạy qua toàn bộ task của hàng ngàn người dùng để tính điểm và gợi ý danh sách. Khối lượng công việc tính toán trong thời gian ngắn có thể lớn, dễ gây **tắc nghẽn** nếu không tối ưu. Cần lên lịch chạy nhiệm vụ nền (cron job) hợp lý hoặc tính toán dần vào đêm hôm trước.
- **Trải nghiệm: tránh gây phiền:** Nếu mỗi sáng đều gợi ý nhưng chất lượng thấp, người dùng sẽ thấy phiền. Cần làm gợi ý ngắn gọn, **hữu ích thực sự**. Có thể cho phép người dùng tùy chỉnh: ví dụ số lượng task gợi ý (3 hay 5 task), hoặc thời gian nhận gợi ý (sáng sớm, hoặc đêm hôm trước). Tránh tình trạng người dùng coi gợi ý của TaskAI là spam.

(4) Đồng bộ hóa với các nền tảng lịch ngoài

Ưu điểm:

- **Hợp nhất thông tin lịch làm việc:** Người dùng chỉ cần xem một nơi (lịch cá nhân) là thấy tất cả công việc và sự kiện của mình. TaskAI đồng bộ task sang Google Calendar/Outlook giúp **tránh bỏ sót** vì nhiều người có thói quen kiểm tra lịch hằng ngày. Việc tích hợp này làm TaskAI **hoà nhập vào luồng làm việc hiện có** của người dùng, thay vì tồn tại tách biệt.
- **Tận dụng tính năng lịch:** Trên Google/Outlook Calendar, người dùng có thể tận dụng nhắc nhở qua email/sms sẵn có, chia sẻ lịch với đồng nghiệp, hoặc xem lịch trên nhiều thiết bị dễ dàng. Khi TaskAI đẩy task sang lịch ngoài, người dùng hưởng lợi từ toàn bộ **hệ sinh thái lịch** đó (thay vì nhóm phát triển phải tự xây mọi tính năng lịch từ đầu).
- **Cải thiện lên kế hoạch:** Nếu task được đưa vào lịch cụ thể (có ngày giờ), người dùng sẽ nghiêm túc hơn trong việc phân bổ thời gian để làm. Đồng bộ hai chiều (nếu có) còn cho phép khi người dùng điều chỉnh thời gian trên Google Calendar thì TaskAI cập nhật lại deadline/timing của task tương ứng, tạo **vòng lặp đồng bộ** liên tục. Điều này giúp TaskAI trở thành một phần **không thể thiếu** của hệ thống quản lý thời gian tổng thể cho người dùng.
- **Thu hút người dùng chuyên nghiệp:** Tính năng tích hợp lịch ngoài rất hữu ích cho **dân văn phòng, quản lý dự án** – những người thường sử dụng Outlook/Google Calendar trong công việc. Đây là điểm cộng để cạnh tranh với các ứng dụng quản lý công việc khác (vốn thường có tích hợp lịch). TaskAI nhờ đó tăng tính **thực tiễn**, dễ được chấp nhận trong môi trường doanh nghiệp.

Thách thức:

- **Kỹ thuật tích hợp API:** Mỗi nền tảng lịch có API riêng (Google Calendar API, Microsoft Graph API cho Outlook, Apple Calendar có CalDAV). Lập trình kết nối, thực hiện OAuth2 để người dùng đăng nhập cấp quyền, rồi duy trì token (làm mới token định kỳ) là phần việc không nhỏ. Cần xử lý cẩn thận để **bảo mật** (lưu token mã hóa) và **đảm bảo hiệu năng** (gọi API phù hợp hạn mức).
- **Khác biệt nền tảng:** Google và Outlook Calendar khá phổ biến với API hỗ trợ tốt. Tuy nhiên Apple Calendar của người dùng iCloud cá nhân khó tích hợp trực tiếp (có thể cần dùng CalDAV, phức tạp hơn). Đảm bảo đồng bộ thống nhất trên nhiều nền tảng đòi hỏi viết code cho từng nền tảng hoặc dùng dịch vụ hợp nhất (như Cronofy, Nylas API – nhưng các dịch vụ này lại tốn phí).
- **Xử lý dữ liệu khi đồng bộ:** Khi tạo task trên TaskAI, nếu đẩy sang Calendar thì cần quyết định hiển thị thế nào: tạo một event vào ngày giờ deadline? Hay tạo block thời gian dự kiến làm task? Nếu chỉ tạo event vào lúc deadline, thì chưa thể hiện khoảng thời gian làm, nhưng nếu tự xếp block thời gian thì có thể xung đột với lịch hiện có. Đây là một **bài toán UX** cần thiết kế: có thể cho task xuất hiện như một **all-day event** (việc trong ngày) với mô tả chi tiết, hoặc để người dùng kéo thả trên lịch.
- **Đồng bộ hai chiều:** Nếu chỉ đồng bộ một chiều (TaskAI -> Calendar) thì đơn giản hơn nhưng nếu người dùng muốn chỉnh sửa trên lịch (đổi ngày, đổi giờ) và mong TaskAI

cập nhật tương ứng, thì cần triển khai đồng bộ hai chiều. Hai chiều phức tạp vì phải **phát hiện thay đổi** trên lịch ngoài (webhook hoặc polling) và đối chiếu với task gốc. Trường hợp chỉnh sửa mâu thuẫn (ví dụ đổi trên cả hai bên khác nhau) cần có cách giải quyết (có thể ưu tiên một phía).

- **Quyền riêng tư & bảo mật:** Khi kết nối lịch cá nhân, TaskAI có thể đọc được các sự kiện cá nhân khác của người dùng (nếu yêu cầu quyền truy cập đầy đủ). Điều này có thể gây e ngại cho người dùng nếu họ lo lắng dữ liệu cá nhân bị lưu trữ hoặc phân tích. Cần chỉ xin quyền tối thiểu (ví dụ chỉ tạo/sửa các sự kiện do TaskAI tạo ra), kèm cam kết không lưu trữ sự kiện cá nhân của họ trên hệ thống mình.
- **Kiểm thử:** Tích hợp với dịch vụ bên ngoài đòi hỏi kiểm thử kỹ lưỡng nhiều tình huống (mạng chậm, token hết hạn, event trùng lặp...). Việc này tăng khối lượng QA và bảo trì.

(5) Thêm chức năng quản lý theo mục tiêu (Goal Tracking)

Ưu điểm:

- **Tầm nhìn dài hạn:** Quản lý theo mục tiêu cho phép người dùng đặt ra các **mục tiêu lớn** (hàng tháng, hàng năm) và theo dõi tiến độ hoàn thành. Tính năng này giúp gắn kết các công việc hằng ngày với mục tiêu tổng thể, tạo **động lực** cho người dùng. Ví dụ, mục tiêu "Chạy 100km trong tháng" có thể chia thành các task chạy bộ hàng ngày; người dùng sẽ thấy mình đã đạt được bao nhiêu % mục tiêu, kích thích họ tiếp tục cố gắng.
- **Tự động chia nhỏ công việc:** Nhiều người gặp khó khăn khi biến mục tiêu lớn thành kế hoạch cụ thể. TaskAI có thể hỗ trợ bằng cách **gợi ý chia mục tiêu thành các task con**. Điều này rất hữu ích vì chia nhỏ mục tiêu giúp công việc bớt quá sức và dễ bắt đầu hơn. Nếu AI làm được việc này, hệ thống sẽ **giảm tải** cho người dùng trong khâu lập kế hoạch, tăng tính **thông minh chủ động** của TaskAI.
- **Theo dõi tiến độ trực quan:** Giao diện Goal Tracking có thể hiển thị thanh tiến trình, % hoàn thành, hoặc biểu đồ tiến độ. Những hình ảnh trực quan này giúp người dùng **nhìn thấy sự tiến bộ** của mình theo thời gian. Về mặt tâm lý, việc thấy tiến bộ (dù nhỏ) cũng tạo cảm giác thành tựu, giúp người dùng gắn bó hơn với ứng dụng và với mục tiêu đã đặt ra.
- **Quản lý mục tiêu đa cấp:** Tính năng này mở rộng phạm vi của TaskAI từ việc quản lý công việc hằng ngày sang quản lý dự án cá nhân. Người dùng (nhất là freelancer hoặc cá nhân tham vọng) có thể quản lý nhiều mục tiêu cùng lúc, mỗi mục tiêu gồm các công việc con. TaskAI trở thành một công cụ **toàn diện hơn**, so sánh được với các ứng dụng như Notion (hay dùng cho mục tiêu) nhưng kết hợp sẵn AI gợi ý cho mục tiêu đó.

Thách thức:

- **Phân rã mục tiêu tự động:** Việc tự động chia mục tiêu thành task không hề đơn giản. AI cần hiểu mục tiêu khá chi tiết. Ví dụ, mục tiêu "Học tiếng Nhật trình độ N3 trong 6 tháng" – để chia nhỏ cần kiến thức về học ngoại ngữ (từ vựng, ngữ pháp, luyện đọc...). Nếu chỉ dựa vào thuật toán cứng có thể không đầy đủ; nếu dùng AI ngôn ngữ (GPT) thì

đôi khi gợi ý không thực sự khả thi. Cần có cách kiểm soát đầu ra của AI để đảm bảo các task gợi ý **cụ thể, thực tế** (SMART).

- **Định lượng tiến độ:** Mỗi mục tiêu có tính chất khác nhau, không phải cái nào cũng dễ đo bằng số task hoàn thành. Ví dụ mục tiêu "Nâng cao kỹ năng thuyết trình" khó đo lường bằng % trừ khi người dùng tự đánh giá. Hệ thống cần linh hoạt cho phép người dùng định nghĩa tiêu chí hoàn thành (số task, hoặc giá trị nào đó). Đây là thách thức về thiết kế và có thể làm phức tạp giao diện.
- **Quản lý mối quan hệ mục tiêu – task:** Cần sửa đổi mô hình dữ liệu: mỗi task có thể thuộc một mục tiêu (goal) nào đó. Nếu người dùng xóa mục tiêu hoặc thay đổi phạm vi, phải cập nhật/cảnh báo với các task liên quan. Tránh tình trạng task "mồ côi" khi mục tiêu bị xóa.
- **Tránh trùng lặp với dự án/nhãn:** Nhiều ứng dụng quản lý task cho phép nhóm task theo dự án hoặc gắn tag/label. Mục tiêu có thể giống một dạng dự án. Cần phân biệt hoặc hợp nhất khái niệm để người dùng không bị rối. Thách thức về UX là làm sao tích hợp quản lý mục tiêu **một cách mạch lạc** với các chức năng sẵn có (ví dụ Calendar, Dashboard) – có thể cần thêm màn hình Goal Overview riêng.
- **Hiệu năng:** Nếu một mục tiêu lớn tạo ra rất nhiều task con (hàng trăm), cần đảm bảo hệ thống xử lý tốt khối lượng công việc tăng đột biến đó. Đặc biệt nếu AI tự động tạo task hàng loạt, có thể gây tải cao tức thời (thêm nhiều bản ghi DB, gửi nhiều thông báo,...). Cần giới hạn hoặc xử lý dần dần.

Gợi ý công nghệ:

- **Cơ sở dữ liệu:** Tạo bảng mới goals (mục tiêu) lưu các thông tin: tên mục tiêu, mô tả, deadline mục tiêu, tiến độ (có thể tính động dựa trên task) và khoá ngoại tới user. Trong bảng tasks thêm trường goal_id để liên kết nếu task thuộc một goal.
- **Backend:** Sửa các API để hỗ trợ goal (API tạo goal, lấy danh sách goals, thêm task vào goal,...). Sử dụng các framework quen thuộc (Java Spring Boot) để tạo entity và service mới cho Goal. Logic tính % hoàn thành có thể viết trong service mỗi lần fetch.
- **Frontend:** Thêm trang **Goal Overview**: có thể hiển thị dạng danh sách mục tiêu với thanh tiến độ. Dùng thư viện đồ thị (như Chart.js) vẽ biểu đồ tiến độ theo thời gian (nếu muốn fancy). Cho phép người dùng click vào mục tiêu để xem các task bên trong (có thể tái sử dụng component danh sách task hiện có, lọc theo goal).
- **AI chia nhỏ:** Tích hợp một endpoint gọi mô hình AI (có thể GPT-3.5) với prompt: "Liệt kê các bước cần thiết để đạt mục tiêu X". Lưu ý parse kết quả ra các task cụ thể. Có thể cần hậu kiểm: ví dụ giới hạn số task, yêu cầu mỗi task có hành động cụ thể. Mô hình ngôn ngữ có thể hỗ trợ nhiều, nhưng cũng nên có thư viện NLP kiểm tra kết quả (như tách các gạch đầu dòng).
- **Công nghệ gợi ý khác:** Xây dựng một **template library**: ví dụ mục tiêu phổ biến ("Chạy marathon 5K") có sẵn các bước trong hệ thống, người dùng chọn là có bộ task. Template này có thể cộng đồng đóng góp hoặc lấy từ tài liệu chuẩn.

- **Thông báo & calendar:** Mục tiêu dài hạn có thể bị quên, nên tích hợp với (7) để thỉnh thoảng nhắc nhở "Bạn còn 2 tuần để hoàn thành mục tiêu XYZ, hãy tăng tốc nếu có thể". Đồng bộ lịch (4) cũng có thể đẩy deadline mục tiêu vào lịch như một mốc quan trọng.

(6) Tối ưu hiệu suất và mở rộng scalability

Ưu điểm:

- **Hiệu suất nhanh hơn:** Tối ưu hệ thống (caching, tối ưu code) giúp thời gian phản hồi nhanh, trải nghiệm người dùng mượt mà kể cả khi dữ liệu nhiều. Ví dụ, nếu áp dụng caching kết quả phân tích AI, khi người dùng nhập mô tả tương tự lần sau có thể trả về ngay lập tức thay vì tính toán lại. Việc triển khai cache có thể giảm tải đáng kể cho cơ sở dữ liệu và tăng tốc độ truy xuất dữ liệu, nhờ lưu sẵn các dữ liệu thường xuyên được truy cập ở bộ nhớ.
- **Sẵn sàng cho nhiều người dùng:** Kiến trúc microservices (phân tách Spring Boot thành các dịch vụ nhỏ) kết hợp Docker/Kubernetes cho phép hệ thống **tự động mở rộng** khi lượng người dùng tăng đột biến. Kubernetes có cơ chế autoscaling (Horizontal Pod Autoscaler) giúp điều chỉnh tài nguyên theo nhu cầu thực tế. Nhờ đó TaskAI có thể phục vụ nhiều người dùng đồng thời mà không sợ nghẽn, đảm bảo tính **ổn định** và **sẵn sàng cao** (High Availability).
- **Bảo trì, phát triển dễ dàng:** Microservices tách biệt các module (ví dụ: service AI riêng, service người dùng riêng) giúp đội ngũ phát triển **nhanh nhẹn** hơn. Mỗi nhóm nhỏ có thể tập trung vào một service, deploy độc lập, nâng cấp không ảnh hưởng toàn hệ thống. Điều này đặc biệt hữu ích khi dự án lớn mạnh – giảm độ phức tạp khi thêm tính năng, cải tiến.
- **Sử dụng hiệu quả tài nguyên:** Docker hóa đảm bảo môi trường chạy nhất quán, dễ triển khai trên nhiều máy. Kubernetes quản lý container giúp tận dụng tốt tài nguyên máy chủ (chạy nhiều container trên một máy khi tải thấp, scale ra nhiều máy khi tải cao). Các kỹ thuật tối ưu (như nén ảnh, rút gọn code frontend) cũng giảm tài nguyên mạng và thiết bị người dùng phải sử dụng, khiến ứng dụng nhẹ hơn.

Thách thức:

- **Độ phức tạp hệ thống:** Chuyển từ monolith sang microservices làm tăng độ phức tạp về kiến trúc rất nhiều. Cần thiết kế các **giao tiếp giữa dịch vụ** (qua REST, gRPC, message queue), quản lý **đồng bộ dữ liệu** (vd: user service và task service cùng cần dữ liệu user). Các vấn đề phân tán nảy sinh: phát hiện và xử lý lỗi giữa các service, log tập trung, theo dõi (trace) request qua nhiều service. Nếu đội ngũ chưa có kinh nghiệm microservices sẽ có **đường cong học tập** cao.
- **DevOps & Hạ tầng:** Triển khai Docker + Kubernetes đòi hỏi kỹ năng DevOps. Phải thiết lập CI/CD, container registry, cấu hình YAML cho Kubernetes (deployment, service, ingress,...). Vận hành K8s cluster cũng không đơn giản: cần giám sát, cập nhật phiên bản, xử lý sự cố (pod crash, node fail). Nếu hạ tầng phức tạp quá sớm, nhóm có thể **tốn nhiều thời gian** cho vận hành thay vì phát triển tính năng.

- **Chi phí hạ tầng:** Việc chạy Kubernetes thường đi kèm chi phí (nếu dùng cloud-managed K8s hoặc phải có nhiều node để tận dụng lợi thế). Với dự án nhỏ, chi phí này có thể chưa tương xứng lợi ích. Tương tự, chia nhiều microservice có thể dẫn đến phải chạy nhiều instance hơn (mỗi service ít nhất 1 instance), tổng chi phí compute có thể tăng so với gộp chung.
- **Quản lý cache và consistency:** Triển khai cache cần cẩn thận để không trả dữ liệu cũ khi đã có thay đổi. Cần chiến lược làm tươi (cache invalidation) hợp lý. Nhiều thành phần cache (VD: cache ở ứng dụng, cache ở database, trình duyệt) phải được phối hợp. Đây là vấn đề **khó trong khoa học máy tính** ("two hard things: cache invalidation and naming things" – câu nói vui nổi tiếng).
- **Testing phức tạp hơn:** Với microservices, test tích hợp toàn hệ thống khó hơn do phải dựng nhiều dịch vụ liên quan. Việc giả lập môi trường K8s cục bộ cũng không đơn giản khi viết test. Cần thiết lập nhiều **bộ công cụ** (như Docker Compose để test, hoặc sử dụng môi trường staging trên cloud) dẫn đến tăng công sức.

(7) Nâng cấp hệ thống thông báo

Ưu điểm:

- **Đa kênh thông báo:** Hiện tại TaskAI có thể chỉ thông báo trong app/web. Việc bổ sung **email notification** và **push notification** (web/mobile) giúp người dùng **không bỏ lỡ nhắc nhở** ngay cả khi không mở ứng dụng. Email có lợi thế phù hợp cho các nhắc công việc chính thức, còn push rất hữu ích cho nhắc nhở tức thời trên điện thoại. Đa kênh thông báo tạo nên một **lưới an toàn**, đảm bảo nhiệm vụ quan trọng đến hạn đều được nhắc kịp thời.
- **Tùy biến thời gian nhắc:** Cho phép người dùng **đặt giờ nhắc nhở theo ý muốn** (ví dụ: 1 ngày trước deadline, 2 giờ trước deadline, hoặc một khung giờ cụ thể) sẽ **nâng cao trải nghiệm** vì đáp ứng đúng nhu cầu từng người. Người cẩn thận có thể đặt nhắc nhiều lần sớm, người bận rộn có thể đặt nhắc sát giờ. Tính cá nhân hóa này giúp người dùng cảm thấy **kiểm soát** được cách TaskAI tương tác với mình, tránh bị làm phiền không cần thiết.
- **Nhắc nhở thông minh dựa trên lịch rảnh:** Đây là điểm **đột phá** giúp TaskAI khác biệt. Thay vì nhắc nhở cứng nhắc đúng giờ cố định, hệ thống có thể canh lúc người dùng đang rảnh (không có sự kiện trên lịch) để gửi nhắc nhở làm task. Điều này tăng khả năng người dùng sẽ thực sự làm task ngay khi nhận nhắc. Ví dụ, 3h chiều user trống lịch, TaskAI gửi push: "Bạn đang rảnh 30 phút, hãy thực hiện nhiệm vụ A sắp đến hạn." Cách nhắc này **hiệu quả** hơn nhiều so với nhắc vào lúc họ đang bận họp. Nó thể hiện tính **thông minh nhân tạo** khi biết tìm thời điểm tối ưu để tương tác với người dùng.
- **Tăng tính chuyên nghiệp:** Các kênh thông báo bổ sung (đặc biệt email) làm TaskAI trở thành một công cụ hoàn thiện như các sản phẩm lớn. Email có thể kèm nội dung chi tiết, thậm chí báo cáo hàng ngày, giúp TaskAI **ghi điểm** với nhóm người dùng văn phòng (vốn quen làm việc qua email). Push notification cần thiết nếu sau này TaskAI

có app di động – đây là yêu cầu mặc định cho bất kỳ app năng suất nào, nên triển khai sẽ đón đầu cho việc phát hành app.

Thách thức:

- **Triển khai kỹ thuật Email:** Gửi email hàng loạt cần thông qua SMTP server hoặc dịch vụ (SendGrid, Amazon SES...). Phải đảm bảo email không vào spam – cần thiết lập xác thực DKIM, SPF cho tên miền gửi. Đây là cấu hình kỹ thuật bổ sung. Ngoài ra, việc thiết kế nội dung email (template HTML) cũng cần đầu tư để trông chuyên nghiệp và hiển thị tốt trên các mail client.
- **Triển khai kỹ thuật Push:**
    - **Web push:** Cần sử dụng Service Worker trên frontend và Push API. Yêu cầu người dùng cho phép (permission). Triển khai web push có sự khác biệt giữa trình duyệt (Safari iOS mãi gần đây mới hỗ trợ Web Push). Tương đối phức tạp nếu tự làm từ đầu, có thể dùng dịch vụ như Firebase Cloud Messaging (FCM) cho web.
    - **Mobile push:** Yêu cầu có ứng dụng mobile. Nếu có, dùng Firebase FCM cho Android, Apple Push Notification service (APNs) cho iOS. Tích hợp đòi hỏi làm việc với khóa API, chứng chỉ. Nếu chưa có app, có thể tạm bỏ qua mobile push.
    - **Điều kiện gửi:** Cần xây dựng **hệ thống sự kiện** để quyết định khi nào gửi noti. Tránh gửi quá nhiều (gây phiền), cũng tránh gửi thiếu. Thử thách lớn là logic "nhắc thông minh khi rảnh" – phải liên tục theo dõi lịch và chọn thời điểm phù hợp, có thể cần job chạy mỗi vài phút kiểm tra lịch rảnh, khá phức tạp và tốn tài nguyên.
- **Quản lý tùy chọn người dùng:** Mở rộng thông báo đồng nghĩa với cần trang cài đặt chi tiết cho user: kênh nào bật/tắt, khung giờ yên lặng (do not disturb), tần suất nhắc nhở,... Nếu không quản lý, thông báo có thể **gây phiền toái** (ví dụ nửa đêm vẫn push noti). Việc lưu và tôn trọng các tùy chọn này làm tăng độ phức tạp hệ thống.
- **Đồng bộ giữa các thiết bị:** Người dùng có thể nhận thông báo trên nhiều thiết bị (email trên PC, push trên điện thoại). Có khả năng họ tương tác (đánh dấu hoàn thành task) trên một thiết bị, hệ thống cần dừng nhắc trên thiết bị kia. Nếu không, có trường hợp người dùng đã làm xong việc nhưng vẫn nhận email nhắc, gây khó chịu. Đây là thách thức về đồng bộ trạng thái real-time giữa các client.
- **Bảo mật & riêng tư:** Gửi thông báo ra ngoài (email) nghĩa là dữ liệu task có thể nằm trong email người dùng – cần cẩn trọng không gửi thông tin quá nhạy cảm hoặc cho phép họ tùy chỉnh nội dung email. Với push, nội dung hiển thị trên màn hình khóa cũng có thể bị người khác nhìn thấy, nên cho phép ẩn nội dung chi tiết nếu người dùng muốn (như các ứng dụng chat làm).
- **Kiểm thử:** Phải test kỹ các luồng: gửi email có thành công? Có bị trùng lặp? Push noti có hoạt động trên các trình duyệt? Test cả trường hợp user tắt quyền push, email không nhận được... Việc debug thông báo khá mất thời gian vì phụ thuộc nhiều hệ thống bên ngoài.

4.3. Kết lời

TaskAI mang lại những **giá trị thiết thực** cho nhiều nhóm người dùng khác nhau trong cuộc sống và công việc hàng ngày:

- **Học sinh, sinh viên:** TaskAI giúp các bạn trẻ quản lý việc học tập một cách khoa học. Ví dụ, sinh viên có thể nhập các bài tập, lịch thi, dự án nhóm vào TaskAI và được hệ thống nhắc nhở trước mỗi deadline, gợi ý phân bổ thời gian học từng môn. Tính năng phân tích độ ưu tiên hỗ trợ sinh viên biết môn nào quan trọng hơn để ưu tiên học trước. Lịch tích hợp cho phép các bạn nhìn rõ thời gian biểu (lịch học, lịch thi) và sắp xếp việc ôn tập hợp lý. Nhờ đó, các bạn giảm thiểu việc **quên bài** hoặc **nước đến chân mới nhảy**, hình thành thói quen học tập chủ động hơn. Ngoài ra, công cụ AI như Chat Assistant có thể hỗ trợ lập kế hoạch học tập bằng ngôn ngữ tự nhiên (vd: "Lên kế hoạch ôn thi toán trong 2 tuần") – rất thuận tiện cho học sinh sinh viên chưa có kỹ năng quản lý thời gian tốt.
- **Nhân viên văn phòng:** Đối với người đi làm, TaskAI như một **trợ lý công việc** cá nhân. Trong môi trường văn phòng bận rộn, mỗi ngày có hàng loạt việc từ họp hành, làm báo cáo, trả lời email đến các dự án dài hạn. TaskAI giúp nhân viên văn phòng ghi lại nhanh mọi task phát sinh (kể cả bằng cách chat giọng nói/natural language), tự động gắn deadline, mức ưu tiên. Các tính năng như gợi ý thông minh mỗi sáng sẽ nhắc họ những việc quan trọng cần làm ngay, tránh bị cuốn vào họp hành mà quên nhiệm vụ chính. Dashboard và Overview cung cấp cái nhìn trực quan về **khối lượng công việc** và **hiệu suất** – ví dụ cho thấy tuần này đã hoàn thành bao nhiêu việc, có bị tồn đọng không. Điều này hỗ trợ cá nhân tự đánh giá và điều chỉnh cách làm việc (một dạng **phản hồi hiệu suất cá nhân**). Hơn nữa, khi đồng bộ với Outlook/Gmail lịch công ty, TaskAI giúp người dùng văn phòng **kết nối liền mạch** giữa lịch làm việc và danh sách nhiệm vụ cá nhân, nâng cao năng suất chung.
- **Freelancer và người quản lý cá nhân:** Những người làm việc tự do thường phải tự quản lý nhiều dự án cho các khách hàng khác nhau, thời hạn đan xen. TaskAI là công cụ đắc lực để họ theo dõi **toàn bộ dự án** tại một nơi. Với khả năng nhóm công việc theo mục tiêu/dự án, freelancer dễ dàng thấy tiến độ từng project, đảm bảo **đúng hạn** với khách. Tính năng Goal Tracking nếu triển khai sẽ giúp freelancer đặt mục tiêu doanh thu hoặc số dự án muốn hoàn thành và theo dõi tiến độ đạt mục tiêu đó. Thêm vào đó, các thông báo đa kênh (email, mobile) cực kỳ hữu ích cho freelancer – họ có thể đang di chuyển hoặc làm việc ngoài văn phòng, việc nhận được nhắc nhở trên điện thoại về một deadline trong ngày sẽ giúp họ **không bỏ lỡ cam kết**. Đối với quản lý cá nhân (những người thích tự tổ chức cuộc sống), TaskAI có thể hỗ trợ quản lý cả công việc lẫn việc nhà, sở thích cá nhân. Ví dụ, một người có thể dùng TaskAI để nhắc việc tập thể dục, lên lịch học kỹ năng mới,... dần dần **phát triển bản thân** có hệ thống hơn.

Tóm lại, trong bối cảnh hiện tại, TaskAI có khả năng **nâng cao hiệu quả cá nhân** rõ rệt. Dù là học sinh hay nhân viên, người dùng đều đối mặt với nhiều đầu việc và áp lực thời gian – TaskAI giúp họ **giảm căng thẳng** bằng cách tổ chức công việc khoa học và nhắc nhở kịp thời. Điều

này không chỉ giúp hoàn thành công việc đúng hạn mà còn cải thiện **cân bằng cuộc sống** (do người dùng quản lý thời gian tốt hơn, không dồn việc đến phút chót).

TÀI LIỆU THAM KHẢO VÀ PHỤ LỤC

i. Scikit-learn Documentation. Truy cập tại: https://scikit-learn.org

ii. Bird, S., Klein, E., & Loper, E. *Natural Language Processing with Python*. O'Reilly Media, 2009

iii. FastAPI Documentation. Truy cập tại: https://fastapi.tiangolo.com

iv. ReactJS Documentation. Truy cập tại: https://react.dev

v. Spring Boot Reference Documentation. Truy cập tại: https://spring.io/projects/spring-boot

vi. Random Forest Classifier & Regressor – Breiman, L. (2001). *Machine Learning*, 45(1).

vii. AI Vietnam Blog, "Giới thiệu về TF-IDF và ứng dụng trong NLP", Truy cập: https://aivietnam.edu.vn/blog

viii. GitHub dự án TaskAI: https://github.com/23001825-NguyenQuangAnh/group36-finalterm

ix. Link slide: https://github.com/23001825-NguyenQuangAnh/group36-finalterm/blob/main/Slide_AI.pdf

x. Link báo cáo: https://github.com/23001825-NguyenQuangAnh/group36-finalterm/blob/main/Bao_Cao_AI.pdf

xi. Hướng dẫn sử dụng và cài đặt có trong README của dự án trên Github