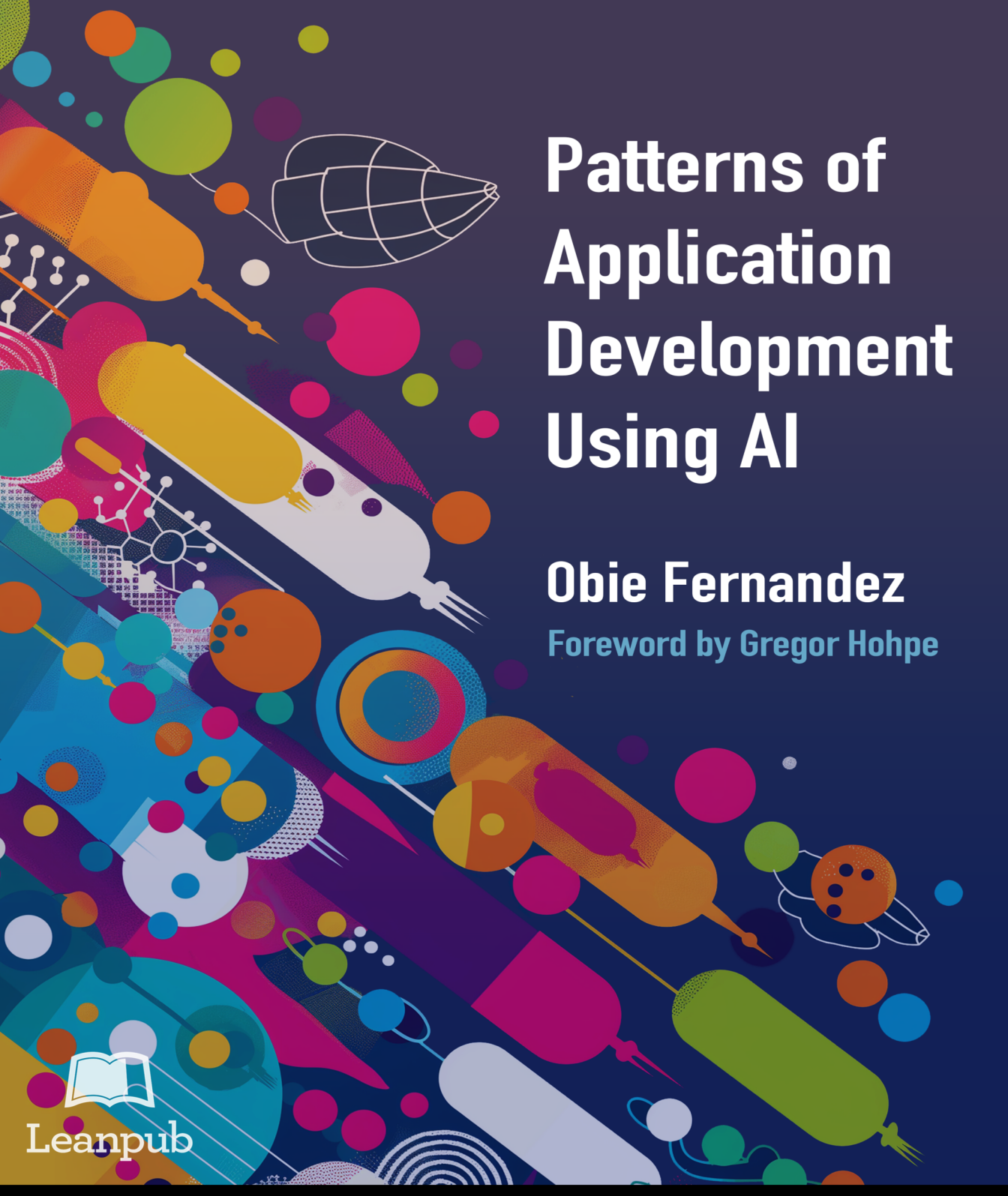


Phiên Bản Tiếng Việt

Mẫu Hình Phát Triển Ứng Dụng Sử Dụng AI (Phiên Bản Tiếng Việt)

Obie Fernandez

Bạn có thể mua cuốn sách này tại
http://leanpub.com/patterns-of-application-development-using-ai-vi

Phiên bản này được xuất bản vào 2025-01-23

Đây là một cuốn sách của Leanpub. Leanpub tăng cường năng lực cho các tác giả và nhà xuất bản thông qua quá trình Xuất Bản Lean. Xuất Bản Lean, hay quá trình xuất bản một ebook đang được tiến triển, là sử dụng các công cụ đơn giản và liên tục chỉnh sửa qua nhiều vòng lặp để thu thập phản hồi từ độc giả, linh hoạt điều chỉnh cho đến khi cuốn sách hoàn hảo và tạo dựng sự quan tâm ngay khi bạn thực hiện được điều này.

Tweet về Cuốn Sách này!

Hãy giúp Obie Fernandez lan truyền về cuốn sách này trên Twitter!

Hashtag được đề xuất cho cuốn sách này là #poaduai.

Tìm hiểu những gì mọi người đang nói về cuốn sách bằng cách nhấp vào liên kết này để tìm kiếm hashtag trên Twitter:

#poaduai

Gửi người hoàng hậu phi thường của tôi, nàng thơ của tôi, ánh sáng và tình yêu của tôi, Victoria

Also By Obie Fernandez

Patterns of Application Development Using AI

The Rails 8 Way

The Rails 7 Way

XML The Rails Way

Serverless

El Libro Principiante de Node

The Lean Enterprise

Mục lục

Lời tựa của Gregor Hohpe

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Lời nói đầu

Về Cuốn Sách

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Về Các Ví Dụ Mã Nguồn

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Những gì tôi không đề cập

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Cuốn sách này dành cho ai

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Xây dựng Từ vựng Chung

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Tham gia

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Lời cảm ơn

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Điều gì đặc biệt về các hình minh họa?

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

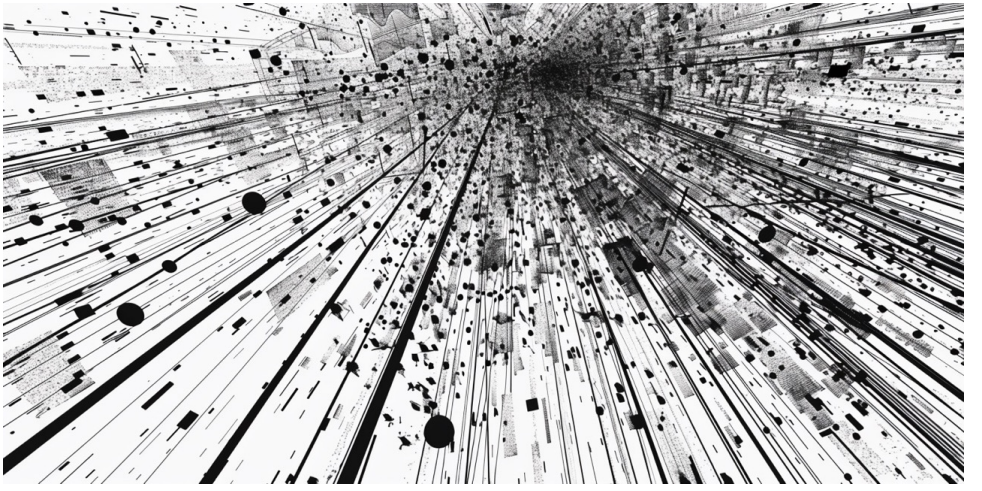
Về Xuất Bản Tinh Gọn

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Về Tác Giả

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Giới thiệu



Nếu bạn đang háo hức muốn bắt đầu tích hợp các Mô hình ngôn ngữ lớn AI (LLM) vào các dự án lập trình của mình, bạn có thể thoải mái đi thẳng vào các mẫu thiết kế và ví dụ mã nguồn được trình bày trong các chương sau. Tuy nhiên, để đánh giá đầy đủ sức mạnh và tiềm năng của những mẫu thiết kế này, việc dành chút thời gian để hiểu bối cảnh rộng hơn và cách tiếp cận mạch lạc mà chúng đại diện là rất đáng giá.

Các mẫu thiết kế không đơn thuần là một tập hợp các kỹ thuật riêng lẻ mà là một khuôn khổ thống nhất để tích hợp AI vào ứng dụng của bạn. Tôi sử dụng Ruby on Rails, nhưng những mẫu thiết kế này nên hoạt động được trong hầu hết mọi môi trường lập trình khác. Chúng giải quyết nhiều vấn đề khác nhau, từ quản lý dữ liệu và tối ưu hóa hiệu suất đến trải nghiệm người dùng và bảo mật, cung cấp một bộ công cụ toàn diện để nâng cao các phương pháp lập trình truyền thống với khả năng của AI.

Mỗi danh mục mẫu thiết kế giải quyết một thách thức hoặc cơ hội cụ thể phát sinh khi tích hợp các thành phần AI vào ứng dụng của bạn. Bằng cách hiểu mối quan hệ và sự

phối hợp giữa các mẫu thiết kế này, bạn có thể đưa ra quyết định sáng suốt về việc nên áp dụng AI ở đâu và như thế nào một cách hiệu quả nhất.

Các mẫu thiết kế không bao giờ là giải pháp mang tính quy định và không nên được coi như vậy. Chúng được thiết kế để là những khối xây dựng có thể điều chỉnh được và nên được tùy biến theo các yêu cầu và ràng buộc riêng của ứng dụng của bạn. Việc áp dụng thành công các mẫu thiết kế này (như bất kỳ mẫu thiết kế nào khác trong lĩnh vực phần mềm) dựa trên sự hiểu biết sâu sắc về lĩnh vực vấn đề, nhu cầu người dùng và kiến trúc kỹ thuật tổng thể của dự án của bạn.

Suy nghĩ về Kiến trúc Phần mềm

Tôi bắt đầu lập trình vào những năm 1980 và đã tham gia vào cộng đồng hacker, và không bao giờ đánh mất tư duy hacker của mình, ngay cả sau khi trở thành một lập trình viên chuyên nghiệp. Ngay từ đầu, tôi luôn có một sự hoài nghi lành mạnh về việc các kiến trúc sư phần mềm trong tháp ngà của họ thực sự mang lại giá trị gì.

Một trong những lý do khiến cá nhân tôi rất phấn khích về những thay đổi do làn sóng công nghệ AI mạnh mẽ mới này mang lại là tác động của nó đến những gì chúng ta coi là quyết định về *kiến trúc phần mềm*. Nó thách thức những quan niệm truyền thống về cách "đúng đắn" để thiết kế và triển khai các dự án phần mềm của chúng ta. Nó cũng thách thức liệu kiến trúc có thể vẫn được coi là chủ yếu là *những phần của hệ thống khó thay đổi*, vì việc nâng cấp AI đang làm cho việc thay đổi bất kỳ phần nào của dự án của bạn, vào bất kỳ thời điểm nào, trở nên dễ dàng hơn bao giờ hết.

Có lẽ chúng ta đang bước vào những năm đỉnh cao của cách tiếp cận "hậu hiện đại" đối với kỹ thuật phần mềm. Trong bối cảnh này, hậu hiện đại ám chỉ một sự thay đổi cơ bản từ các mô hình truyền thống, nơi các nhà phát triển chịu trách nhiệm viết và duy trì từng dòng mã. Thay vào đó, nó chấp nhận ý tưởng ủy thác các tác vụ, như thao tác dữ liệu, thuật toán phức tạp, và thậm chí cả những phần logic ứng dụng hoàn chỉnh, cho các thư viện bên thứ ba và API bên ngoài. Sự thay đổi hậu hiện đại này thể hiện

một bước ngoặt đáng kể từ quan điểm truyền thống về việc xây dựng ứng dụng từ đầu, và nó thách thức các nhà phát triển phải suy nghĩ lại về vai trò của họ trong quá trình phát triển.

Tôi luôn tin rằng các lập trình viên giỏi chỉ viết những đoạn mã thực sự cần thiết phải viết, dựa trên những bài học từ Larry Wall và những nhân vật hacker nổi tiếng khác như ông. Bằng cách giảm thiểu lượng mã được viết, chúng ta có thể di chuyển nhanh hơn, giảm bớt diện tích cho lỗi, đơn giản hóa việc bảo trì và cải thiện độ tin cậy tổng thể của ứng dụng. Ít mã hơn cho phép chúng ta tập trung vào logic nghiệp vụ cốt lõi và trải nghiệm người dùng, trong khi ủy thác công việc khác cho các dịch vụ khác.

Giờ đây khi các hệ thống được hỗ trợ bởi AI có thể xử lý các tác vụ trước đây chỉ thuộc phạm vi của mã do con người viết, chúng ta sẽ có thể năng suất và linh hoạt hơn nữa, với sự tập trung hơn bao giờ hết vào việc tạo ra giá trị kinh doanh và trải nghiệm người dùng.

Tất nhiên có những đánh đổi khi ủy thác những phần lớn dự án của bạn cho các hệ thống AI, chẳng hạn như khả năng mất kiểm soát và nhu cầu về các cơ chế giám sát và phản hồi mạnh mẽ. Đó là lý do tại sao nó đòi hỏi một bộ kỹ năng và kiến thức mới, bao gồm ít nhất một số hiểu biết cơ bản về cách AI hoạt động.

Mô hình Ngôn ngữ Lớn là gì?

Các Mô hình Ngôn ngữ Lớn (LLM) là một loại mô hình trí tuệ nhân tạo đã thu hút được sự chú ý đáng kể trong những năm gần đây, kể từ khi OpenAI ra mắt GPT-3 vào năm 2020. LLM được thiết kế để xử lý, hiểu và tạo ra ngôn ngữ của con người với độ chính xác và trôi chảy đáng kinh ngạc. Trong phần này, chúng ta sẽ xem xét sơ lược cách LLM hoạt động và tại sao chúng phù hợp để xây dựng các thành phần hệ thống thông minh.

Về cốt lõi, LLM dựa trên các thuật toán học sâu, cụ thể là mạng nơ-ron. Các mạng này bao gồm các nút kết nối, hay nơ-ron, xử lý và truyền thông tin. Kiến trúc được lựa chọn

cho LLM thường là mô hình Transformer, đã được chứng minh là rất hiệu quả trong việc xử lý dữ liệu tuần tự như văn bản.

Các mô hình Transformer được xây dựng dựa trên cơ chế tập trung và chủ yếu được sử dụng cho các tác vụ liên quan đến dữ liệu tuần tự, như xử lý ngôn ngữ tự nhiên. Transformer xử lý dữ liệu đầu vào cùng một lúc thay vì tuần tự, cho phép chúng nắm bắt các phụ thuộc tầm xa hiệu quả hơn. Chúng có các lớp của cơ chế tập trung giúp mô hình tập trung vào các phần khác nhau của dữ liệu đầu vào để hiểu ngữ cảnh và mối quan hệ.

Quá trình huấn luyện cho các mô hình ngôn ngữ lớn (LLM) bao gồm việc cho mô hình tiếp xúc với một lượng lớn dữ liệu văn bản, như sách, bài báo, trang web và kho mã nguồn. Trong quá trình huấn luyện, mô hình học cách nhận biết các mẫu, mối quan hệ và cấu trúc trong văn bản. Nó nắm bắt các đặc tính thống kê của ngôn ngữ, như quy tắc ngữ pháp, sự kết hợp từ và ý nghĩa ngữ cảnh.

Một trong những kỹ thuật chính được sử dụng trong việc huấn luyện LLM là học không giám sát. Điều này có nghĩa là mô hình học từ dữ liệu mà không cần gán nhãn hay hướng dẫn rõ ràng. Nó tự khám phá các mẫu và biểu diễn bằng cách phân tích sự xuất hiện đồng thời của các từ và cụm từ trong dữ liệu huấn luyện. Điều này cho phép LLM phát triển hiểu biết sâu sắc về ngôn ngữ và các đặc điểm phức tạp của nó.

Một khía cạnh quan trọng khác của LLM là khả năng xử lý *ngữ cảnh*. Khi xử lý một đoạn văn bản, LLM không chỉ xem xét từng từ riêng lẻ mà còn cả ngữ cảnh xung quanh. Chúng xem xét các từ, câu và thậm chí các đoạn văn trước đó để hiểu ý nghĩa và ý định của văn bản. Sự hiểu biết về ngữ cảnh này cho phép LLM tạo ra các phản hồi mạch lạc và phù hợp. Một trong những cách chính để đánh giá khả năng của một mô hình LLM là xem xét kích thước ngữ cảnh mà chúng có thể xem xét để tạo ra phản hồi.

Sau khi được huấn luyện, LLM có thể được sử dụng cho nhiều tác vụ liên quan đến ngôn ngữ. Chúng có thể tạo ra văn bản giống người viết, trả lời câu hỏi, tóm tắt tài liệu, dịch ngôn ngữ và thậm chí viết mã. Tính đa năng của LLM khiến chúng trở nên có giá trị trong việc xây dựng các thành phần hệ thống thông minh có thể tương tác với người

dùng, xử lý và phân tích dữ liệu văn bản, và tạo ra các kết quả có ý nghĩa.

Bằng cách tích hợp LLM vào kiến trúc ứng dụng, bạn có thể tạo ra các thành phần AI có khả năng hiểu và xử lý đầu vào của người dùng, tạo nội dung động và đưa ra các đề xuất hoặc hành động thông minh. Tuy nhiên, làm việc với LLM đòi hỏi phải cân nhắc kỹ lưỡng về yêu cầu tài nguyên và sự đánh đổi hiệu năng. LLM đòi hỏi nhiều tính toán và có thể cần sức mạnh xử lý và bộ nhớ đáng kể (nói cách khác là tiền) để vận hành. Hầu hết chúng ta sẽ cần đánh giá các tác động về chi phí khi tích hợp LLM vào ứng dụng của mình và hành động phù hợp.

Hiểu về Suy luận

Suy luận đề cập đến quá trình mà một mô hình tạo ra các dự đoán hoặc kết quả dựa trên dữ liệu mới, chưa từng thấy. Đây là giai đoạn mà mô hình đã được huấn luyện được sử dụng để đưa ra quyết định hoặc tạo ra văn bản, hình ảnh, hoặc nội dung khác để đáp ứng đầu vào của người dùng.

Trong giai đoạn huấn luyện, một mô hình AI học từ một tập dữ liệu lớn bằng cách điều chỉnh các tham số của nó để giảm thiểu sai số trong các dự đoán. Sau khi được huấn luyện, mô hình có thể áp dụng những gì đã học được vào dữ liệu mới. Suy luận là cách mô hình sử dụng các mẫu và kiến thức đã học được để tạo ra kết quả.

Đối với LLM, suy luận bao gồm việc nhận một lệnh nhắc hoặc văn bản đầu vào và tạo ra một phản hồi mạch lạc và phù hợp với ngữ cảnh, dưới dạng luồng *token* (mà chúng ta sẽ nói đến sau). Điều này có thể là trả lời câu hỏi, hoàn thành câu, tạo ra câu chuyện, hoặc dịch văn bản, trong số nhiều tác vụ khác.

Khác với cách mà bạn và tôi suy nghĩ, "suy nghĩ" của mô hình AI thông qua suy luận diễn ra trong một thao tác phi trạng thái duy nhất. Nghĩa là, quá trình suy nghĩ của nó giới hạn trong quá trình tạo ra. Nó thực sự phải suy nghĩ thành tiếng, giống như khi tôi hỏi bạn một câu hỏi và chỉ chấp nhận câu trả lời theo kiểu "dòng ý thức".

Các Mô hình Ngôn ngữ Lớn Có Nhiều Kích thước và Biến thể

Mặc dù hầu như tất cả các mô hình ngôn ngữ lớn (LLM) phổ biến đều dựa trên cùng một kiến trúc transformer cốt lõi và được huấn luyện trên các tập dữ liệu văn bản khổng lồ, chúng có nhiều kích thước khác nhau và được tinh chỉnh cho các mục đích khác nhau. Kích thước của một LLM, được đo bằng số lượng tham số trong mạng nơ-ron của nó, có ảnh hưởng lớn đến khả năng của nó. Các mô hình lớn hơn với nhiều tham số hơn, như GPT-4, được đồn đoán có từ 1 đến 2 nghìn tỷ tham số, thường có kiến thức và khả năng tốt hơn các mô hình nhỏ hơn. Tuy nhiên, các mô hình lớn hơn cũng đòi hỏi nhiều sức mạnh tính toán hơn để chạy, điều này dẫn đến chi phí cao hơn khi bạn sử dụng chúng thông qua các cuộc gọi API.

Để làm cho LLM trở nên thực tế hơn và phù hợp với các trường hợp sử dụng cụ thể, các mô hình cơ sở thường được tinh chỉnh trên các tập dữ liệu có mục tiêu cụ thể hơn. Ví dụ, một LLM có thể được huấn luyện trên một kho dữ liệu đối thoại lớn để chuyên biệt hóa nó cho AI đàm thoại. Một số khác được huấn luyện trên mã nguồn để trang bị cho chúng kiến thức lập trình. Thậm chí còn có những mô hình được huấn luyện đặc biệt cho các tương tác kiểu đóng vai với người dùng!

Mô Hình Truy Xuất và Mô Hình Sinh

Trong thế giới của các mô hình ngôn ngữ lớn (LLMs), có hai cách tiếp cận chính để tạo ra phản hồi: mô hình dựa trên truy xuất và mô hình sinh. Mỗi cách tiếp cận đều có những điểm mạnh và điểm yếu riêng, và việc hiểu rõ sự khác biệt giữa chúng có thể giúp bạn chọn được mô hình phù hợp cho trường hợp sử dụng cụ thể của mình.

Mô Hình Dựa Trên Truy Xuất

Mô hình dựa trên truy xuất, còn được gọi là mô hình truy xuất thông tin, tạo ra phản hồi bằng cách tìm kiếm trong một cơ sở dữ liệu lớn chứa văn bản có sẵn và chọn ra

những đoạn văn bản phù hợp nhất dựa trên câu truy vấn đầu vào. Những mô hình này không tạo ra văn bản mới từ đầu mà thay vào đó ghép các đoạn trích từ cơ sở dữ liệu để tạo thành một phản hồi mạch lạc.

Một trong những ưu điểm chính của mô hình dựa trên truy xuất là khả năng cung cấp thông tin chính xác và cập nhật. Vì chúng dựa vào một cơ sở dữ liệu văn bản được tuyển chọn, chúng có thể lấy thông tin liên quan từ các nguồn đáng tin cậy và trình bày cho người dùng. Điều này làm cho chúng phù hợp với các ứng dụng đòi hỏi câu trả lời chính xác, thực tế, như các hệ thống hỏi đáp hoặc cơ sở tri thức.

Tuy nhiên, mô hình dựa trên truy xuất có một số hạn chế. Chất lượng của chúng phụ thuộc vào cơ sở dữ liệu mà chúng tìm kiếm, vì vậy chất lượng và phạm vi của cơ sở dữ liệu trực tiếp ảnh hưởng đến hiệu suất của mô hình. Ngoài ra, những mô hình này có thể gặp khó khăn trong việc tạo ra các phản hồi mạch lạc và tự nhiên, vì chúng bị giới hạn bởi văn bản có sẵn trong cơ sở dữ liệu.

Chúng tôi không đề cập đến việc sử dụng các mô hình truy xuất thuần túy trong cuốn sách này.

Mô Hình Sinh

Ngược lại, mô hình sinh tạo ra văn bản mới từ đầu dựa trên các mẫu và mối quan hệ mà chúng đã học được trong quá trình huấn luyện. Những mô hình này sử dụng hiểu biết về ngôn ngữ của chúng để tạo ra các phản hồi mới phù hợp với yêu cầu đầu vào.

Điểm mạnh chính của mô hình sinh là khả năng tạo ra văn bản sáng tạo, mạch lạc và phù hợp với ngữ cảnh. Chúng có thể tham gia vào các cuộc trò chuyện mở, tạo ra câu chuyện, và thậm chí viết mã. Điều này làm cho chúng lý tưởng cho các ứng dụng đòi hỏi tương tác mở và linh hoạt hơn, như chatbot, tạo nội dung, và trợ lý viết sáng tạo.

Tuy nhiên, mô hình sinh đôi khi có thể tạo ra thông tin không nhất quán hoặc không chính xác về mặt thực tế, vì chúng dựa vào các mẫu học được trong quá trình huấn luyện thay vì một cơ sở dữ liệu thực tế được tuyển chọn. Chúng cũng có thể dễ bị thiên kiến và ảo giác hơn, tạo ra văn bản có vẻ hợp lý nhưng không nhất thiết đúng sự thật.

Ví dụ về các mô hình sinh LLM bao gồm dòng GPT của OpenAI (GPT-3, GPT-4) và Claude của Anthropic.

Mô Hình Lai

Một số LLM thương mại kết hợp cả hai cách tiếp cận truy xuất và sinh trong một mô hình lai. Những mô hình này sử dụng kỹ thuật truy xuất để tìm thông tin liên quan từ cơ sở dữ liệu và sau đó sử dụng kỹ thuật sinh để tổng hợp thông tin đó thành một phản hồi mạch lạc.

Mô hình lai nhằm kết hợp độ chính xác thực tế của mô hình dựa trên truy xuất với khả năng tạo ngôn ngữ tự nhiên của mô hình sinh. Chúng có thể cung cấp thông tin đáng tin cậy và cập nhật hơn trong khi vẫn duy trì khả năng tham gia vào các cuộc trò chuyện mở.

Khi lựa chọn giữa mô hình dựa trên truy xuất và mô hình sinh, bạn nên xem xét các yêu cầu cụ thể của ứng dụng của mình. Nếu mục tiêu chính là cung cấp thông tin chính xác, thực tế, một mô hình dựa trên truy xuất có thể là lựa chọn tốt nhất. Nếu ứng dụng đòi hỏi tương tác mở và sáng tạo hơn, một mô hình sinh có thể phù hợp hơn. Mô hình lai cung cấp sự cân bằng giữa hai cách tiếp cận và có thể là một lựa chọn tốt cho các ứng dụng đòi hỏi cả độ chính xác thực tế và khả năng tạo ngôn ngữ tự nhiên.

Cuối cùng, việc lựa chọn giữa mô hình dựa trên truy xuất và mô hình sinh phụ thuộc vào trường hợp sử dụng cụ thể và sự cân nhắc giữa độ chính xác, sáng tạo và tính linh hoạt. Bằng cách hiểu rõ điểm mạnh và hạn chế của mỗi cách tiếp cận, bạn có thể đưa ra quyết định sáng suốt.

Tinh Chỉnh Theo Hướng Dẫn

Tinh chỉnh theo hướng dẫn là một tập con của việc huấn luyện hội thoại sử dụng các câu nhắc và phản hồi do con người viết để giúp LLM hiểu và làm theo hướng dẫn tốt hơn. Dưới đây là một số ví dụ đơn giản để minh họa cách tinh chỉnh theo hướng dẫn trên các câu nhắc và phản hồi do con người viết giúp LLM hiểu và làm theo hướng dẫn tốt hơn:

Ví dụ 1: Hỗ Trợ Khách Hàng

Hướng dẫn: "Cung cấp phản hồi cho khách hàng yêu cầu hoàn tiền."

Câu nhắc do con người viết: "Tôi muốn yêu cầu hoàn tiền cho đơn hàng gần đây. Sản phẩm không đáp ứng được kỳ vọng của tôi."

Phản hồi do con người viết: "Tôi rất tiếc khi nghe sản phẩm không đáp ứng được kỳ vọng của bạn. Để xử lý hoàn tiền, vui lòng cung cấp mã đơn hàng và lý do trả lại. Chúng tôi sẽ xem xét yêu cầu của bạn và phản hồi trong vòng 2-3 ngày làm việc."

Ví dụ 2: Hỗ Trợ Giáo Dục

Hướng dẫn: "Giải thích khái niệm quang hợp cho trẻ 10 tuổi."

Câu hỏi của con người: "Bạn có thể giải thích cách thức thực vật tạo ra thức ăn không?"

Câu trả lời của con người: "Được chứ! Thực vật tạo ra thức ăn thông qua quá trình gọi là quang hợp. Chúng sử dụng ánh sáng mặt trời, nước và khí cacbonic để tạo ra thức ăn và oxy. Lá cây có một bộ phận đặc biệt gọi là diệp lục tố giúp chúng hấp thụ ánh sáng mặt trời."

Ví dụ 3: Trợ lý Cá nhân

> Chỉ dẫn: “Lên lịch một cuộc họp vào thứ Ba tới lúc 3 giờ chiều.”
>
> Câu hỏi của con người: “Bạn có thể sắp xếp một cuộc họp vào thứ Ba lúc 3 giờ chiều không?”
>
> Câu trả lời của con người: “Được, tôi đã lên lịch một cuộc họp cho bạn vào thứ Ba tới lúc 3 giờ chiều. Bạn cần gì thêm không?”

Kết quả là một hệ sinh thái đa dạng các LLM với nhiều kích cỡ và chuyên môn khác nhau. Các mô hình nhỏ hơn trong khoảng 1-7 tỷ tham số cung cấp khả năng xử lý ngôn ngữ tổng quát tốt trong khi vận hành hiệu quả hơn.

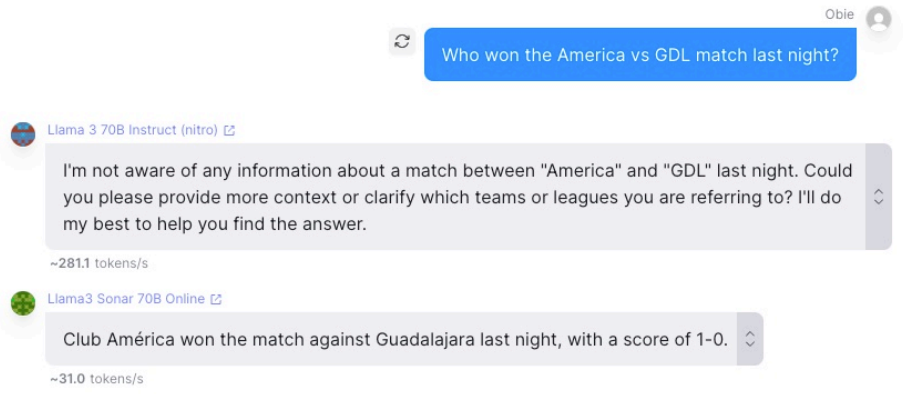
- Mistral 7B
- Llama 3 8B
- Gemma 7B

Các mô hình cỡ trung bình khoảng 30-70 tỷ tham số cung cấp khả năng suy luận và thực hiện chỉ dẫn mạnh mẽ hơn.

- Llama 3 70B
- Qwen2 70B
- Mixtral 8x22B

Khi lựa chọn một LLM để tích hợp vào ứng dụng, bạn phải cân bằng giữa khả năng của mô hình với các yếu tố thực tế như chi phí, độ trễ, độ dài ngữ cảnh và bộ lọc nội dung. Các mô hình nhỏ hơn được tinh chỉnh theo hướng dẫn thường là lựa chọn tốt nhất cho các tác vụ ngôn ngữ đơn giản hơn, trong khi các mô hình lớn nhất có thể cần thiết cho việc suy luận hoặc phân tích phức tạp. Dữ liệu huấn luyện của mô hình cũng là một yếu tố quan trọng cần xem xét, vì nó quyết định thời điểm giới hạn kiến thức của mô hình.

Một số mô hình nhất định, như một số từ Perplexity được kết nối với các nguồn thông tin thời gian thực, do đó chúng thực sự không có ngày giới hạn. Khi bạn đặt câu hỏi, chúng có thể tự quyết định thực hiện tìm kiếm web và truy xuất các trang web tùy ý để tạo ra câu trả lời.



Hình 1. Llama3 với và không có kết nối trực tuyến

Cuối cùng, không có LLM nào phù hợp với mọi trường hợp. Việc hiểu rõ sự khác biệt về kích thước mô hình, kiến trúc và quá trình huấn luyện là chìa khóa để lựa chọn mô hình phù hợp cho một trường hợp sử dụng cụ thể. Thử nghiệm với các mô hình khác nhau là cách thực tế duy nhất để phát hiện mô hình nào cung cấp hiệu suất tốt nhất cho nhiệm vụ cần thực hiện.

Tokenization: Chia văn bản thành các phần nhỏ

Trước khi một mô hình ngôn ngữ lớn có thể xử lý văn bản, văn bản đó cần được chia thành các đơn vị nhỏ hơn gọi là *token*. Token có thể là các từ riêng lẻ, phần của từ, hoặc thậm chí là các ký tự đơn lẻ. Quá trình chia văn bản thành token được gọi là tokenization, và đó là một bước quan trọng trong việc chuẩn bị dữ liệu cho mô hình ngôn ngữ.

Hình 2. Câu này chứa 27 token

Các LLM khác nhau sử dụng các chiến lược tokenization khác nhau, điều này có thể tác động đáng kể đến hiệu suất và khả năng của mô hình. Một số tokenizer phổ biến được sử dụng bởi các LLM bao gồm:

- **GPT (Mã hóa cặp byte)**: Các tokenizer GPT sử dụng một kỹ thuật gọi là mã hóa cặp byte (BPE) để chia văn bản thành các đơn vị từ con. BPE lặp đi lặp lại việc kết hợp các cặp byte xuất hiện thường xuyên nhất trong một kho ngữ liệu văn bản, tạo thành một từ vựng các token từ con. Điều này cho phép tokenizer xử lý các từ hiếm và mới bằng cách chia chúng thành các phần từ con phổ biến hơn. Các tokenizer GPT được sử dụng bởi các mô hình như GPT-3 và GPT-4.

- **Llama (SentencePiece)**: Các bộ token hóa Llama sử dụng thư viện SentencePiece, một công cụ token hóa và giải token hóa văn bản không giám sát. SentencePiece xử lý văn bản đầu vào như một chuỗi ký tự Unicode và học từ vựng phụ dựa trên kho ngữ liệu huấn luyện. Nó có thể xử lý bất kỳ ngôn ngữ nào được mã hóa trong Unicode, khiến nó phù hợp cho các mô hình đa ngôn ngữ. Các bộ token hóa Llama được sử dụng bởi các mô hình như Llama và Alpaca của Meta.

- **SentencePiece (Unigram)**: Các bộ token hóa SentencePiece cũng có thể sử dụng một thuật toán khác gọi là Unigram, dựa trên kỹ thuật điều chỉnh từ phụ. Token hóa Unigram xác định từ vựng phụ tối ưu dựa trên mô hình ngôn ngữ unigram, gán xác suất cho từng đơn vị từ phụ. Phương pháp này có thể tạo ra các từ phụ có ý nghĩa ngữ nghĩa hơn so với BPE. SentencePiece với Unigram được sử dụng bởi các mô hình như T5 và BERT của Google.

- **Google Gemini (Token hóa Đa phương thức)**: Google Gemini sử dụng một phương thức token hóa được thiết kế để xử lý nhiều loại dữ liệu khác nhau, bao

gồm văn bản, hình ảnh, âm thanh, video và mã nguồn. Khả năng đa phương thức này cho phép Gemini xử lý và tích hợp các hình thức thông tin khác nhau. Đáng chú ý, Google Gemini 1.5 Pro có cửa sổ ngữ cảnh có thể xử lý hàng triệu token, lớn hơn nhiều so với các mô hình trước đây. Cửa sổ ngữ cảnh rộng lớn này cho phép mô hình xử lý ngữ cảnh lớn hơn, có khả năng dẫn đến các phản hồi chính xác hơn. Tuy nhiên, điều quan trọng cần lưu ý là phương thức token hóa của Gemini gần với một token cho mỗi ký tự hơn so với các mô hình khác. Điều này có nghĩa là chi phí thực tế khi sử dụng các mô hình Gemini có thể cao hơn đáng kể so với dự kiến nếu bạn quen với việc sử dụng các mô hình như GPT, vì giá của Google dựa trên ký tự thay vì token.

Việc lựa chọn bộ token hóa ảnh hưởng đến nhiều khía cạnh của một LLM, bao gồm:

- **Kích thước từ vựng**: Bộ token hóa xác định kích thước từ vựng của mô hình, là tập hợp các token duy nhất mà nó nhận biết được. Một từ vựng lớn hơn, chi tiết hơn có thể giúp mô hình xử lý nhiều từ và cụm từ hơn và thậm chí trở thành đa phương thức (có khả năng hiểu và tạo ra nhiều hơn chỉ văn bản), nhưng nó cũng làm tăng yêu cầu bộ nhớ và độ phức tạp tính toán của mô hình.
- **Xử lý các từ hiếm và từ không xác định**: Các bộ token hóa sử dụng các đơn vị từ phụ, như BPE và SentencePiece, có thể chia nhỏ các từ hiếm và không xác định thành các phần từ phụ phổ biến hơn. Điều này cho phép mô hình đưa ra các phỏng đoán có căn cứ về ý nghĩa của các từ mà nó chưa từng thấy trước đây, dựa trên các từ phụ chúng chứa.
- **Hỗ trợ đa ngôn ngữ**: Các bộ token hóa như SentencePiece, có thể xử lý bất kỳ ngôn ngữ nào có thể mã hóa Unicode, rất phù hợp cho các mô hình đa ngôn ngữ cần xử lý văn bản bằng nhiều ngôn ngữ.

Khi chọn một LLM cho một ứng dụng cụ thể, điều quan trọng là phải xem xét bộ token hóa mà nó sử dụng và mức độ phù hợp với nhu cầu xử lý ngôn ngữ cụ thể của nhiệm vụ đó. Bộ token hóa có thể có tác động đáng kể đến khả năng xử lý thuật ngữ chuyên ngành, từ hiếm và văn bản đa ngôn ngữ của mô hình.

Kích thước ngữ cảnh: Mô hình ngôn ngữ có thể sử dụng bao nhiêu thông tin trong quá trình suy luận?

Khi thảo luận về các mô hình ngôn ngữ, kích thước ngữ cảnh đề cập đến lượng văn bản mà một mô hình có thể xem xét khi xử lý hoặc tạo ra phản hồi. Về cơ bản, đó là thước đo lượng thông tin mà mô hình có thể "ghi nhớ" và sử dụng để tạo ra đầu ra của nó (được biểu thị bằng token). Kích thước ngữ cảnh của một mô hình ngôn ngữ có thể có tác động đáng kể đến khả năng của nó và các loại nhiệm vụ mà nó có thể thực hiện hiệu quả.

Kích thước ngữ cảnh là gì?

Về mặt kỹ thuật, kích thước ngữ cảnh được xác định bởi số lượng token (từ hoặc phần của từ) mà một mô hình ngôn ngữ có thể xử lý trong một chuỗi đầu vào duy nhất. Điều này thường được gọi là "khoảng chú ý" hoặc "cửa sổ ngữ cảnh" của mô hình. Kích thước ngữ cảnh càng lớn, mô hình càng có thể xem xét nhiều văn bản hơn cùng một lúc khi tạo ra phản hồi hoặc thực hiện một nhiệm vụ.

Các mô hình ngôn ngữ khác nhau có kích thước ngữ cảnh khác nhau, từ vài trăm token đến hàng triệu token. Để tham khảo, một đoạn văn bản điển hình có thể chứa khoảng 100-150 token, trong khi một cuốn sách có thể chứa hàng chục hoặc hàng trăm nghìn token.

> Thậm chí còn có nghiên cứu về các phương pháp hiệu quả để mở rộng các Mô hình Ngôn ngữ Lớn (LLMs) dựa trên Transformer để xử lý đầu vào vô hạn với bộ nhớ và tính toán có giới hạn.

Tại sao Kích thước Ngữ cảnh Quan trọng?

Kích thước ngữ cảnh của một mô hình ngôn ngữ có ảnh hưởng đáng kể đến khả năng hiểu và tạo ra văn bản mạch lạc, phù hợp với ngữ cảnh. Dưới đây là một số lý do chính giải thích tại sao kích thước ngữ cảnh quan trọng:

1. **Hiểu nội dung dài**: Các mô hình có kích thước ngữ cảnh lớn hơn có thể hiểu và phân tích tốt hơn các văn bản dài như bài báo, báo cáo, hoặc thậm chí cả cuốn sách. Điều này rất quan trọng cho các tác vụ như tóm tắt tài liệu, trả lời câu hỏi và phân tích nội dung.

2. **Duy trì tính mạch lạc**: Cửa sổ ngữ cảnh lớn hơn cho phép mô hình duy trì tính mạch lạc và nhất quán trong các đoạn văn bản dài hơn. Điều này quan trọng đối với các tác vụ như tạo câu chuyện, hệ thống đối thoại và tạo nội dung, nơi việc duy trì tính nhất quán của câu chuyện hoặc chủ đề là thiết yếu. Điều này cũng đặc biệt quan trọng khi sử dụng các mô hình ngôn ngữ lớn để tạo hoặc chuyển đổi dữ liệu có cấu trúc.
3. **Nắm bắt các phụ thuộc tầm xa**: Một số tác vụ ngôn ngữ đòi hỏi phải hiểu mối quan hệ giữa các từ hoặc cụm từ cách xa nhau trong văn bản. Các mô hình có kích thước ngữ cảnh lớn hơn được trang bị tốt hơn để nắm bắt các phụ thuộc tầm xa này, điều này có thể quan trọng cho các tác vụ như phân tích cảm xúc, dịch thuật, và hiểu ngôn ngữ.
4. **Xử lý hướng dẫn phức tạp**: Trong các ứng dụng sử dụng mô hình ngôn ngữ để thực hiện các hướng dẫn phức tạp, nhiều bước, kích thước ngữ cảnh lớn hơn cho phép mô hình xem xét toàn bộ tập hợp hướng dẫn khi tạo phản hồi, thay vì chỉ xem xét vài từ gần nhất.

Ví dụ về các Mô hình Ngôn ngữ với Kích thước Ngữ cảnh Khác nhau

Dưới đây là một số ví dụ về mô hình ngôn ngữ với kích thước ngữ cảnh khác nhau:

- OpenAI GPT-3.5 Turbo: 4.095 token
- Mistral 7B Instruct: 32.768 token
- Anthropic Claude v1: 100.000 token
- OpenAI GPT-4 Turbo: 128.000 token
- Anthropic Claude v2: 200.000 token
- Google Gemini Pro 1.5: 2,8 triệu token

Như bạn có thể thấy, có một phạm vi rộng về kích thước ngữ cảnh giữa các mô hình này, từ khoảng 4.000 token cho mô hình OpenAI GPT-3.5 Turbo đến 200.000 token cho mô hình Anthropic Claude v2. Một số mô hình, như Google PaLM 2 và OpenAI GPT-4, cung cấp các biến thể khác nhau với kích thước ngữ cảnh lớn hơn (ví dụ: phiên bản "32k"), có thể xử lý các chuỗi đầu vào dài hơn. Và tại thời điểm hiện tại (tháng 4 năm 2024), Google Gemini Pro đang tự hào với gần 3 triệu token!

Đáng chú ý là kích thước ngữ cảnh có thể thay đổi tùy thuộc vào cách triển khai và phiên bản cụ thể của một mô hình. Ví dụ, mô hình OpenAI GPT-4 ban đầu có kích thước ngữ cảnh là 8.191 token, trong khi các biến thể GPT-4 sau này như Turbo và 4o có kích thước ngữ cảnh lớn hơn nhiều là 128.000 token.

> Sam Altman đã so sánh các giới hạn ngữ cảnh hiện tại với bộ nhớ làm việc tính bằng kilobyte mà các lập trình viên máy tính cá nhân phải đối mặt trong những năm 80, và nói rằng trong tương lai gần, chúng ta sẽ có thể đưa "tất cả dữ liệu cá nhân của bạn" vào ngữ cảnh của một mô hình ngôn ngữ lớn.

Chọn Kích thước Ngữ cảnh Phù hợp

Khi lựa chọn một mô hình ngôn ngữ cho một ứng dụng cụ thể, điều quan trọng là phải xem xét yêu cầu về kích thước ngữ cảnh của tác vụ đó. Đối với các tác vụ liên quan đến

các đoạn văn bản ngắn, độc lập, như phân tích cảm xúc hoặc trả lời câu hỏi đơn giản, một kích thước ngữ cảnh nhỏ hơn có thể là đủ. Tuy nhiên, đối với các tác vụ đòi hỏi hiểu và tạo ra các văn bản dài hơn, phức tạp hơn, một kích thước ngữ cảnh lớn hơn sẽ có thể là cần thiết.

Đáng chú ý là kích thước ngữ cảnh lớn hơn thường đi kèm với chi phí tính toán cao hơn và thời gian xử lý chậm hơn, vì mô hình cần xem xét nhiều thông tin hơn khi tạo phản hồi. Do đó, bạn phải cân bằng giữa kích thước ngữ cảnh và hiệu suất khi chọn mô hình ngôn ngữ cho ứng dụng của mình.

> Tại sao không chọn mô hình có kích thước ngữ cảnh lớn nhất và nhồi nhét vào đó càng nhiều thông tin càng tốt? Chà, ngoài các yếu tố về hiệu suất, yếu tố chính khác cần xem xét là chi phí. Vào tháng 3 năm 2024, *một* chu kỳ lệnh-phản hồi sử dụng Google Gemini Pro 1.5 với ngữ cảnh đầy đủ sẽ tiêu tốn của bạn gần 8 đô la Mỹ (USD). Nếu bạn có trường hợp sử dụng biện minh cho chi phí đó, thì tốt thôi! Nhưng đối với hầu hết các ứng dụng, nó quá đắt gấp nhiều lần.

Tìm Kim Trong Đống Rơm

Khái niệm tìm kim trong đống rơm từ lâu đã là một phép ẩn dụ cho những thách thức trong việc truy xuất dữ liệu từ các tập dữ liệu lớn. Trong lĩnh vực mô hình ngôn ngữ lớn (LLM), chúng ta điều chỉnh phép ẩn dụ này một chút. Hãy tưởng tượng chúng ta không chỉ tìm kiếm một sự thật đơn lẻ được chôn giấu trong một văn bản đồ sộ (như toàn bộ tuyển tập các bài luận của Paul Graham), mà là nhiều sự thật rải rác khắp nơi. Kịch bản này giống như việc tìm nhiều cây kim trong một cánh đồng rộng lớn, không chỉ trong một đống rơm đơn lẻ. Điều đặc biệt ở đây là: chúng ta không chỉ cần định vị những cây kim này, mà còn phải đan kết chúng thành một mạch logic mạch lạc.

Khi phải truy xuất và suy luận về nhiều sự thật được nhúng trong các ngữ cảnh dài,

LLM phải đối mặt với hai thách thức. Thứ nhất là vấn đề đơn giản về độ chính xác trong truy xuất—nó tự nhiên giảm đi khi số lượng sự thật tăng lên. Điều này là điều dễ hiểu; xét cho cùng, việc theo dõi nhiều chi tiết trong một văn bản dài dòng cũng làm khó ngay cả những mô hình tinh vi nhất.

Thứ hai, và có lẽ quan trọng hơn, là thách thức về việc suy luận với những sự thật này. Việc chọn lọc các sự thật là một chuyện; việc tổng hợp chúng thành một câu chuyện hoặc câu trả lời mạch lạc lại là chuyện khác. Đây mới là thử thách thực sự. Hiệu suất của LLM trong các tác vụ suy luận thường suy giảm nhiều hơn so với các tác vụ truy xuất đơn giản. Sự suy giảm này không chỉ liên quan đến khối lượng; mà còn về sự phối hợp tinh tế giữa ngữ cảnh, sự liên quan và suy luận.

Tại sao điều này xảy ra? Hãy xem xét động lực của bộ nhớ và sự chú ý trong nhận thức của con người, điều này phần nào được phản ánh trong LLM. Khi xử lý một lượng lớn thông tin, LLM, giống như con người, có thể đánh mất dấu những chi tiết trước đó khi họ tiếp nhận những thông tin mới. Điều này đặc biệt đúng với các mô hình không được thiết kế rõ ràng để ưu tiên hoặc tự động xem lại các đoạn văn bản trước đó.

Hơn nữa, khả năng của LLM trong việc đan kết các sự thật được truy xuất thành một phản hồi mạch lạc giống như việc xây dựng tường thuật. Điều này đòi hỏi không chỉ việc truy xuất thông tin mà còn cần sự hiểu biết sâu sắc và định vị ngữ cảnh, điều vẫn còn là một thách thức lớn đối với AI hiện nay.

Vậy điều này có ý nghĩa gì đối với chúng ta như những nhà phát triển và tích hợp các công nghệ này? Chúng ta cần nhận thức sâu sắc về những hạn chế này khi thiết kế các hệ thống dựa vào LLM để xử lý các tác vụ phức tạp, dài. Việc hiểu rằng hiệu suất có thể suy giảm trong một số điều kiện nhất định giúp chúng ta đặt ra những kỳ vọng thực tế và xây dựng các cơ chế dự phòng hoặc chiến lược bổ sung tốt hơn.

Phương Thức: Vượt Ra Ngoài Văn Bản

Trong khi phần lớn các mô hình ngôn ngữ hiện nay tập trung vào việc xử lý và tạo ra văn bản, có một xu hướng ngày càng tăng hướng tới các mô hình đa phương thức có

khả năng tự nhiên trong việc nhận và xuất nhiều loại dữ liệu khác nhau, như hình ảnh, âm thanh và video. Những mô hình đa phương thức này mở ra những khả năng mới cho các ứng dụng dựa trên AI có thể hiểu và tạo ra nội dung trên các phương thức khác nhau.

Phương Thức Là Gì?

Trong bối cảnh của các mô hình ngôn ngữ, phương thức đề cập đến các loại dữ liệu khác nhau mà một mô hình có thể xử lý và tạo ra. Phương thức phổ biến nhất là văn bản, bao gồm ngôn ngữ viết dưới nhiều hình thức như sách, bài báo, trang web và bài đăng trên mạng xã hội. Tuy nhiên, có một số phương thức khác đang ngày càng được tích hợp vào các mô hình ngôn ngữ:

- **Hình ảnh**: Dữ liệu hình ảnh như ảnh chụp, hình minh họa và sơ đồ.
- **Âm thanh**: Dữ liệu âm thanh như giọng nói, âm nhạc và âm thanh môi trường.
- **Video**: Dữ liệu hình ảnh động, thường đi kèm với âm thanh, như các đoạn video clip và phim.

Mỗi phương thức mang đến những thách thức và cơ hội riêng cho các mô hình ngôn ngữ. Ví dụ, hình ảnh đòi hỏi mô hình phải hiểu các khái niệm và mối quan hệ trực quan, trong khi âm thanh đòi hỏi mô hình phải xử lý và tạo ra giọng nói và các âm thanh khác.

Mô Hình Ngôn Ngữ Đa Phương Thức

Các mô hình ngôn ngữ đa phương thức được thiết kế để xử lý nhiều phương thức trong một mô hình duy nhất. Những mô hình này thường có các thành phần hoặc lớp chuyên biệt có thể vừa hiểu đầu vào vừa tạo ra dữ liệu đầu ra ở các phương thức khác nhau. Một số ví dụ nổi bật về mô hình ngôn ngữ đa phương thức bao gồm:

- **OpenAI's GPT-4o**: GPT-4o là một mô hình ngôn ngữ lớn có khả năng tự nhiên trong việc hiểu và xử lý âm thanh giọng nói ngoài văn bản. Khả năng này cho

phép GPT-4o thực hiện các tác vụ như phiên âm ngôn ngữ nói, tạo văn bản từ đầu vào âm thanh và cung cấp phản hồi dựa trên các truy vấn bằng giọng nói.

- **OpenAI's GPT-4 với đầu vào hình ảnh**: GPT-4 là một mô hình ngôn ngữ lớn có thể xử lý cả văn bản và hình ảnh. Khi được cung cấp một hình ảnh làm đầu vào, GPT-4 có thể phân tích nội dung của hình ảnh và tạo ra văn bản mô tả hoặc phản hồi thông tin trực quan.
- **Google's Gemini**: Gemini là một mô hình đa phương thức có thể xử lý văn bản, hình ảnh và video. Nó sử dụng kiến trúc thống nhất cho phép hiểu và tạo ra nội dung đa phương thức, cho phép thực hiện các tác vụ như chú thích hình ảnh, tóm tắt video và trả lời câu hỏi dựa trên hình ảnh.

- **DALL-E và Stable Diffusion**: Mặc dù không phải là mô hình ngôn ngữ theo nghĩa truyền thống, những mô hình này thể hiện sức mạnh của AI đa phương thức bằng cách tạo ra hình ảnh từ mô tả văn bản. Chúng cho thấy tiềm năng của các mô hình có khả năng chuyển đổi giữa các phương thức khác nhau.

Lợi ích và Ứng dụng của Mô hình Đa phương thức

Các mô hình ngôn ngữ đa phương thức mang lại nhiều lợi ích và cho phép nhiều ứng dụng đa dạng, bao gồm:

- **Nâng cao khả năng hiểu**: Bằng cách xử lý thông tin từ nhiều phương thức, những mô hình này có thể đạt được sự hiểu biết toàn diện hơn về thế giới, tương tự như cách con người học từ các đầu vào cảm giác khác nhau.
- **Sinh nội dung đa phương thức**: Các mô hình đa phương thức có thể tạo ra nội dung ở một phương thức dựa trên đầu vào từ phương thức khác, chẳng hạn như tạo hình ảnh từ mô tả văn bản hoặc tạo tóm tắt video từ bài viết.
- **Khả năng tiếp cận**: Các mô hình đa phương thức có thể làm cho thông tin dễ tiếp cận hơn bằng cách chuyển đổi giữa các phương thức, như tạo mô tả văn bản cho hình ảnh dành cho người khiếm thị hoặc tạo phiên bản âm thanh từ nội dung văn bản.

- **Ứng dụng sáng tạo**: Các mô hình đa phương thức có thể được sử dụng cho các công việc sáng tạo như tạo nghệ thuật, âm nhạc, hoặc video dựa trên các gợi ý văn bản, mở ra những khả năng mới cho nghệ sĩ và người sáng tạo nội dung.

Khi các mô hình ngôn ngữ đa phương thức tiếp tục phát triển, chúng có thể sẽ đóng vai trò ngày càng quan trọng trong việc phát triển các ứng dụng được hỗ trợ bởi AI có khả năng hiểu và tạo ra nội dung trên nhiều phương thức. Điều này sẽ cho phép tương tác tự nhiên và trực quan hơn giữa con người và hệ thống AI, cũng như mở ra những khả năng mới cho việc thể hiện sáng tạo và phổ biến kiến thức.

Hệ sinh thái Nhà cung cấp

Khi nói đến việc tích hợp các mô hình ngôn ngữ lớn (LLMs) vào ứng dụng, bạn có nhiều lựa chọn ngày càng đa dạng. Mỗi nhà cung cấp LLM lớn, như OpenAI, Anthropic, Google, và Cohere, đều cung cấp hệ sinh thái riêng của họ với các mô hình, API, và công cụ. Việc chọn đúng nhà cung cấp liên quan đến việc xem xét nhiều yếu tố, bao gồm giá cả, hiệu suất, lọc nội dung, bảo mật dữ liệu, và tùy chọn tùy chỉnh.

OpenAI

OpenAI là một trong những nhà cung cấp LLM nổi tiếng nhất, với loạt mô hình GPT (GPT-3, GPT-4) được sử dụng rộng rãi trong nhiều ứng dụng khác nhau. OpenAI cung cấp API thân thiện với người dùng cho phép bạn dễ dàng tích hợp các mô hình của họ vào ứng dụng. Họ cung cấp nhiều mô hình với các khả năng và mức giá khác nhau, từ mô hình Ada cơ bản đến mô hình Davinci mạnh mẽ.

Hệ sinh thái của OpenAI cũng bao gồm các công cụ như OpenAI Playground, cho phép bạn thử nghiệm với các câu lệnh và tinh chỉnh mô hình cho các trường hợp sử dụng cụ thể. Họ cung cấp các tùy chọn lọc nội dung để giúp ngăn chặn việc tạo ra nội dung không phù hợp hoặc có hại.

Khi sử dụng trực tiếp các mô hình của OpenAI, tôi dựa vào thư viện ruby-openai của Alex Rudall.

Anthropic

Anthropic là một công ty lớn khác trong lĩnh vực LLM, với các mô hình Claude của họ đang ngày càng phổ biến nhờ hiệu suất mạnh mẽ và các cân nhắc về đạo đức. Anthropic tập trung vào phát triển các hệ thống AI an toàn và có trách nhiệm, với sự nhấn mạnh vào việc lọc nội dung và tránh các đầu ra có hại.

Hệ sinh thái của Anthropic bao gồm API Claude, cho phép bạn tích hợp mô hình vào ứng dụng của họ, cũng như các công cụ cho kỹ thuật thiết kế câu lệnh và tinh chỉnh. Họ cũng cung cấp mô hình Claude Instant, tích hợp khả năng tìm kiếm web để có các phản hồi cập nhật và chính xác hơn.

Khi sử dụng trực tiếp các mô hình của Anthropic, tôi dựa vào thư viện anthrophic của Alex Rudall.

Google

Google đã phát triển một số mô hình LLM mạnh mẽ, bao gồm Gemini, BERT, T5, và PaLM. Những mô hình này nổi tiếng với hiệu suất mạnh mẽ trong nhiều tác vụ xử lý ngôn ngữ tự nhiên. Hệ sinh thái của Google bao gồm các thư viện TensorFlow và Keras, cung cấp công cụ và framework để xây dựng và huấn luyện các mô hình học máy.

Google cũng cung cấp Nền tảng AI Cloud, cho phép bạn dễ dàng triển khai và mở rộng các mô hình của họ trên đám mây. Họ cung cấp nhiều mô hình được huấn luyện sẵn và API cho các tác vụ như phân tích cảm xúc, nhận dạng thực thể, và dịch thuật.

Meta

Meta, trước đây được biết đến là Facebook, đầu tư mạnh mẽ vào việc phát triển các mô hình ngôn ngữ lớn, nổi bật với việc phát hành các mô hình như LLaMA và OPT. Những mô hình này nổi bật với hiệu suất mạnh mẽ trong các tác vụ ngôn ngữ đa dạng và chủ yếu được cung cấp thông qua các kênh mã nguồn mở, thể hiện cam kết của Meta đối với nghiên cứu và hợp tác cộng đồng.

Hệ sinh thái của Meta chủ yếu được xây dựng xung quanh PyTorch, một thư viện học máy mã nguồn mở được ưa chuộng nhờ khả năng tính toán động và tính linh hoạt, tạo điều kiện cho nghiên cứu và phát triển AI sáng tạo.

Bên cạnh các giải pháp kỹ thuật, Meta đặt nhiều trọng tâm vào việc phát triển AI có đạo đức. Họ triển khai hệ thống lọc nội dung mạnh mẽ và tập trung vào việc giảm thiểu định kiến, phù hợp với mục tiêu tổng thể về tính an toàn và trách nhiệm trong các ứng dụng AI.

Cohere

Cohere là một tân binh trong lĩnh vực LLM, tập trung vào việc làm cho LLM dễ tiếp cận và dễ sử dụng hơn so với các đối thủ cạnh tranh. Hệ sinh thái của họ bao gồm Cohere API, cung cấp quyền truy cập vào nhiều mô hình được huấn luyện sẵn cho các tác vụ như tạo văn bản, phân loại và tóm tắt.

Cohere cũng cung cấp các công cụ để kỹ thuật prompt, tinh chỉnh và lọc nội dung. Họ nhấn mạnh vào quyền riêng tư và bảo mật dữ liệu, với các tính năng như lưu trữ dữ liệu được mã hóa và kiểm soát truy cập.

Ollama

Ollama là một nền tảng tự host cho phép người dùng quản lý và triển khai các mô hình ngôn ngữ lớn (LLMs) cục bộ trên máy tính của họ, cho phép họ kiểm soát hoàn toàn các mô hình AI mà không cần phụ thuộc vào dịch vụ đám mây bên ngoài. Thiết lập này lý tưởng cho những người ưu tiên quyền riêng tư dữ liệu và muốn xử lý các hoạt động AI nội bộ.

Nền tảng này hỗ trợ nhiều mô hình khác nhau, bao gồm các phiên bản của Llama, Phi, Gemma và Mistral, với các yêu cầu về kích thước và tính toán khác nhau. Ollama giúp dễ dàng tải xuống và chạy các mô hình này trực tiếp từ dòng lệnh bằng các lệnh đơn giản như `ollama run <model_name>`, và được thiết kế để hoạt động trên các hệ điều hành khác nhau bao gồm macOS, Linux và Windows.

Đối với các nhà phát triển muốn tích hợp các mô hình mã nguồn mở vào ứng dụng của họ mà không sử dụng API từ xa, Ollama cung cấp CLI để quản lý vòng đời mô hình tương tự như các công cụ quản lý container. Nó cũng hỗ trợ cấu hình tùy chỉnh và prompt, cho phép mức độ tùy biến cao để điều chỉnh các mô hình theo nhu cầu hoặc trường hợp sử dụng cụ thể.

Ollama đặc biệt phù hợp với người dùng am hiểu công nghệ và các nhà phát triển nhờ vào giao diện dòng lệnh và tính linh hoạt trong việc quản lý và triển khai các mô hình AI. Điều này làm cho nó trở thành một công cụ mạnh mẽ cho các doanh nghiệp và cá nhân cần khả năng AI mạnh mẽ mà không ảnh hưởng đến bảo mật và kiểm soát.

Nền tảng Đa mô hình

Ngoài ra, còn có các nhà cung cấp host nhiều mô hình mã nguồn mở khác nhau, như Together.ai và Groq.. Những nền tảng này cung cấp tính linh hoạt và khả năng tùy chỉnh, cho phép bạn chạy và trong một số trường hợp, thậm chí tinh chỉnh các mô hình mã nguồn mở theo nhu cầu cụ thể của bạn. Ví dụ, Together.ai cung cấp quyền truy cập vào nhiều LLM mã nguồn mở, cho phép người dùng thử nghiệm với các mô hình và cấu hình khác nhau. Groq tập trung vào việc cung cấp khả năng hoàn thành hiệu suất cực cao mà tại thời điểm viết cuốn sách này có vẻ gần như kỳ diệu

Lựa chọn Nhà cung cấp LLM

Khi chọn nhà cung cấp LLM, bạn nên cân nhắc các yếu tố như:

- **Giá cả**: Các nhà cung cấp khác nhau có các mô hình giá khác nhau, từ trả tiền theo sử dụng đến các gói đăng ký. Điều quan trọng là cân nhắc mức độ sử dụng dự kiến và ngân sách khi lựa chọn nhà cung cấp.
- **Hiệu suất**: Hiệu suất của LLM có thể khác biệt đáng kể giữa các nhà cung cấp, vì vậy điều quan trọng là phải đánh giá và kiểm tra các mô hình trên các trường hợp sử dụng cụ thể trước khi đưa ra quyết định.

- **Lọc nội dung**: Tùy thuộc vào ứng dụng, việc lọc nội dung có thể là một yếu tố quan trọng cần cân nhắc. Một số nhà cung cấp có các tùy chọn lọc nội dung mạnh mẽ hơn những nhà cung cấp khác.
- **Quyền riêng tư dữ liệu**: Nếu ứng dụng xử lý dữ liệu người dùng nhạy cảm, điều quan trọng là chọn nhà cung cấp có thực hành bảo mật và quyền riêng tư dữ liệu mạnh mẽ.
- **Tùy chỉnh**: Một số nhà cung cấp cung cấp nhiều tính linh hoạt hơn trong việc tinh chỉnh và tùy chỉnh mô hình cho các trường hợp sử dụng cụ thể.

Cuối cùng, việc lựa chọn nhà cung cấp LLM phụ thuộc vào các yêu cầu và ràng buộc cụ thể của ứng dụng. Bằng cách đánh giá cẩn thận các tùy chọn và cân nhắc các yếu tố như giá cả, hiệu suất và quyền riêng tư dữ liệu, bạn có thể chọn nhà cung cấp đáp ứng tốt nhất nhu cầu của mình.

Cũng đáng lưu ý rằng bối cảnh LLM liên tục phát triển, với các nhà cung cấp và mô hình mới xuất hiện thường xuyên. Bạn nên cập nhật với những phát triển mới nhất và sẵn sàng khám phá các tùy chọn mới khi chúng xuất hiện.

OpenRouter

Trong suốt cuốn sách này, tôi sẽ chỉ sử dụng OpenRouter làm nhà cung cấp API của mình. Lý do rất đơn giản: đó là một cửa hàng tập trung cho tất cả các mô hình thương mại và mã nguồn mở phổ biến nhất. Nếu bạn đang háo hức muốn thử nghiệm với một số lập trình AI, một trong những nơi tốt nhất để bắt đầu là với Thư viện Ruby OpenRouter của tôi.

Suy nghĩ về Hiệu năng

Khi tích hợp các mô hình ngôn ngữ vào ứng dụng, hiệu năng là một yếu tố quan trọng cần cân nhắc. Hiệu năng của một mô hình ngôn ngữ có thể được đo lường thông qua

độ trễ (thời gian cần thiết để tạo ra phản hồi) và *thông lượng* (số lượng yêu cầu có thể xử lý trong một đơn vị thời gian).

Thời gian đến Token đầu tiên (TTFT) là một chỉ số hiệu năng thiết yếu khác, đặc biệt quan trọng đối với chatbot và các ứng dụng yêu cầu phản hồi tương tác theo thời gian thực. TTFT đo độ trễ từ thời điểm nhận được yêu cầu của người dùng đến khi từ (hoặc token) đầu tiên của câu trả lời được tạo ra. Chỉ số này rất quan trọng để duy trì trải nghiệm người dùng mượt mà và hấp dẫn, vì phản hồi chậm trễ có thể dẫn đến sự thất vọng và mất kết nối của người dùng.

Những chỉ số hiệu năng này có thể tác động đáng kể đến trải nghiệm người dùng và khả năng mở rộng của ứng dụng.

Có nhiều yếu tố có thể ảnh hưởng đến hiệu năng của một mô hình ngôn ngữ, bao gồm:

Số lượng tham số: Các mô hình lớn hơn với nhiều tham số hơn thường đòi hỏi nhiều tài nguyên tính toán hơn và có thể có độ trễ cao hơn và thông lượng thấp hơn so với các mô hình nhỏ hơn.

Phần cứng: Hiệu năng của một mô hình ngôn ngữ có thể thay đổi đáng kể dựa trên phần cứng mà nó chạy trên đó. Các nhà cung cấp đám mây cung cấp các phiên bản GPU và TPU được tối ưu hóa cho khối lượng công việc học máy, có thể đẩy nhanh đáng kể quá trình suy luận mô hình.

Một trong những điều tuyệt vời về OpenRouter là đối với nhiều mô hình mà nó cung cấp, bạn có thể lựa chọn các nhà cung cấp đám mây với nhiều mức hiệu năng và chi phí khác nhau.

Lượng tử hóa: Các kỹ thuật lượng tử hóa có thể được sử dụng để giảm dung lượng bộ nhớ và yêu cầu tính toán của mô hình bằng cách biểu diễn trọng số và kích hoạt với các kiểu dữ liệu có độ chính xác thấp hơn. Điều này có thể cải thiện hiệu năng mà không làm giảm đáng kể chất lượng. Là một nhà phát triển ứng dụng, có thể bạn sẽ không tham gia vào việc huấn luyện các mô hình riêng của mình ở các mức lượng tử hóa khác nhau, nhưng ít nhất việc làm quen với thuật ngữ này là điều tốt.

Xử lý theo lô: Xử lý nhiều yêu cầu cùng một lúc theo lô có thể cải thiện thông lượng bằng cách phân bổ chi phí phụ trội của việc tải mô hình và truyền dữ liệu.

Lưu trữ đệm: Lưu trữ đệm kết quả của các prompt hoặc chuỗi đầu vào thường xuyên sử dụng có thể giảm số lượng yêu cầu suy luận và cải thiện hiệu năng tổng thể.

Khi lựa chọn một mô hình ngôn ngữ cho ứng dụng sản phẩm, điều quan trọng là phải đánh giá hiệu năng của nó trên các khối lượng công việc và cấu hình phần cứng đại diện. Điều này có thể giúp xác định các điểm nghẽn tiềm ẩn và đảm bảo rằng mô hình có thể đạt được các mục tiêu hiệu năng yêu cầu.

Cũng đáng để cân nhắc sự đánh đổi giữa hiệu năng mô hình và các yếu tố khác như chi phí, tính linh hoạt và độ dễ tích hợp. Ví dụ, sử dụng một mô hình nhỏ hơn, rẻ hơn với độ trễ thấp hơn có thể phù hợp hơn cho các ứng dụng yêu cầu phản hồi thời gian thực, trong khi một mô hình lớn hơn, mạnh mẽ hơn có thể phù hợp hơn cho xử lý theo lô hoặc các tác vụ suy luận phức tạp.

Thử nghiệm Với Các Mô hình LLM Khác nhau

Việc lựa chọn một LLM hiếm khi là một quyết định vĩnh viễn. Khi các mô hình mới và cải tiến được phát hành thường xuyên, tốt nhất là xây dựng ứng dụng theo cách module để có thể thay thế các mô hình ngôn ngữ khác nhau theo thời gian. Các prompt và tập dữ liệu thường có thể được tái sử dụng giữa các mô hình với những thay đổi tối thiểu. Điều này cho phép bạn tận dụng những tiến bộ mới nhất trong mô hình hóa ngôn ngữ mà không cần phải thiết kế lại hoàn toàn ứng dụng của mình.

Khả năng dễ dàng chuyển đổi giữa nhiều lựa chọn mô hình là một lý do khác khiến tôi yêu thích OpenRouter.

Khi nâng cấp lên một mô hình ngôn ngữ mới, điều quan trọng là phải kiểm tra và xác nhận kỹ lưỡng hiệu năng và chất lượng đầu ra của nó để đảm bảo rằng nó đáp ứng các

yêu cầu của ứng dụng. Điều này có thể bao gồm việc huấn luyện lại hoặc tinh chỉnh mô hình trên dữ liệu chuyên ngành, cũng như cập nhật bất kỳ thành phần xuôi dòng nào phụ thuộc vào đầu ra của mô hình.

Bằng cách thiết kế ứng dụng với hiệu năng và tính module trong tâm trí, bạn có thể tạo ra các hệ thống có khả năng mở rộng, hiệu quả và hướng tới tương lai, có thể thích ứng với bối cảnh công nghệ mô hình hóa ngôn ngữ đang phát triển nhanh chóng.

Hệ thống AI Phức hợp

Trước khi kết thúc phần giới thiệu của chúng ta, đáng để đề cập rằng trước năm 2023 và sự bùng nổ quan tâm đến AI sinh thành được châm ngòi bởi ChatGPT, các phương pháp tiếp cận AI truyền thống thường dựa vào việc tích hợp các mô hình đơn lẻ, đóng. Ngược lại, *Hệ thống AI Phức hợp* tận dụng các đường ống phức tạp của các thành phần kết nối với nhau làm việc cùng nhau để đạt được hành vi thông minh.

Về cốt lõi, các hệ thống AI phức hợp bao gồm nhiều module, mỗi module được thiết kế để thực hiện các tác vụ hoặc chức năng cụ thể. Các module này có thể bao gồm bộ sinh, bộ truy xuất, bộ xếp hạng, bộ phân loại và các thành phần chuyên biệt khác nhau. Bằng cách chia nhỏ hệ thống tổng thể thành các đơn vị tập trung nhỏ hơn, các nhà phát triển có thể tạo ra các kiến trúc AI linh hoạt, có khả năng mở rộng và dễ bảo trì hơn.

Một trong những lợi thế chính của hệ thống AI tổng hợp là khả năng kết hợp các điểm mạnh của các kỹ thuật và mô hình AI khác nhau. Ví dụ, một hệ thống có thể sử dụng mô hình ngôn ngữ lớn (LLM) để hiểu và tạo ngôn ngữ tự nhiên, trong khi sử dụng một mô hình riêng biệt cho việc truy xuất thông tin hoặc ra quyết định dựa trên quy tắc. Cách tiếp cận theo mô-đun này cho phép bạn lựa chọn các công cụ và kỹ thuật tốt nhất cho từng nhiệm vụ cụ thể, thay vì dựa vào một giải pháp một-kích-cỡ-phù-hợp-tất-cả.

Tuy nhiên, việc xây dựng hệ thống AI tổng hợp cũng đặt ra những thách thức riêng biệt. Đặc biệt, việc đảm bảo tính nhất quán và mạch lạc tổng thể trong hoạt động của hệ thống đòi hỏi các cơ chế kiểm thử, giám sát và quản trị mạnh mẽ.

Sự ra đời của các LLM mạnh mẽ như GPT-4 cho phép chúng ta thử nghiệm với hệ thống AI tổng hợp dễ dàng hơn bao giờ hết, bởi vì những mô hình tiên tiến này có khả năng đảm nhiệm nhiều vai trò trong một hệ thống tổng hợp, như phân loại, xếp hạng và tạo nội dung, bên cạnh khả năng hiểu ngôn ngữ tự nhiên của chúng. Tính đa năng này cho phép các nhà phát triển nhanh chóng tạo mẫu và lặp lại các kiến trúc AI tổng hợp, mở ra những khả năng mới cho việc phát triển ứng dụng thông minh.

Các Mô hình Triển khai cho Hệ thống AI Tổng hợp

Hệ thống AI tổng hợp có thể được triển khai sử dụng nhiều mô hình khác nhau, mỗi mô hình được thiết kế để đáp ứng các yêu cầu và trường hợp sử dụng cụ thể. Hãy cùng khám phá bốn mô hình triển khai phổ biến: Hỏi và Đáp, Hệ thống Đa tác tử/Giải quyết vấn đề theo tác tử, AI Đàm thoại, và CoPilots.

Hỏi và Đáp

Hệ thống Hỏi và Đáp (Q&A) tập trung vào việc cung cấp khả năng truy xuất thông tin được tăng cường bởi khả năng hiểu của các mô hình AI để hoạt động như một công cụ hơn hẳn một công cụ tìm kiếm đơn thuần. Bằng cách kết hợp các mô hình ngôn ngữ mạnh mẽ với các nguồn kiến thức bên ngoài sử dụng Sinh nội dung với Truy xuất Tăng cường (RAG), hệ thống Hỏi và Đáp tránh được các ảo giác và cung cấp các câu trả lời chính xác và phù hợp với ngữ cảnh cho các câu hỏi của người dùng.

Các thành phần chính của hệ thống Q&A dựa trên LLM bao gồm:

- **Hiểu và chuyển đổi câu hỏi**: Phân tích câu hỏi của người dùng và chuyển đổi chúng để phù hợp hơn với các nguồn kiến thức cơ sở.
- **Truy xuất kiến thức**: Truy xuất thông tin liên quan từ các nguồn dữ liệu có cấu trúc hoặc phi cấu trúc dựa trên câu hỏi đã được chuyển đổi.

- **Tạo câu trả lời**: Tạo ra các câu trả lời mạch lạc và có thông tin bằng cách tích hợp kiến thức đã truy xuất với khả năng tạo nội dung của mô hình ngôn ngữ.

Các hệ thống con RAG đặc biệt quan trọng trong các lĩnh vực Q&A nơi việc cung cấp thông tin chính xác và cập nhật là thiết yếu, như hỗ trợ khách hàng, quản lý kiến thức, hoặc các ứng dụng giáo dục

Hệ thống Đa tác tử/Giải quyết vấn đề theo tác tử

Hệ thống đa tác tử, còn được gọi là hệ thống *theo tác tử*, bao gồm nhiều tác tử tự chủ làm việc cùng nhau để giải quyết các vấn đề phức tạp. Mỗi tác tử có một vai trò cụ thể, tập hợp kỹ năng và quyền truy cập vào các công cụ hoặc nguồn thông tin liên quan. Thông qua việc hợp tác và trao đổi thông tin, các tác tử này có thể giải quyết các nhiệm vụ mà một tác tử đơn lẻ khó hoặc không thể xử lý được.

Các nguyên tắc chính của hệ thống giải quyết vấn đề đa tác tử bao gồm:

- **Chuyên môn hóa**: Mỗi tác tử tập trung vào một khía cạnh cụ thể của vấn đề, tận dụng khả năng và kiến thức độc đáo của mình.
- **Hợp tác**: Các tác tử giao tiếp và phối hợp hành động để đạt được mục tiêu chung, thường thông qua việc truyền tin nhắn hoặc bộ nhớ chia sẻ.
- **Khả năng thích ứng**: Hệ thống có thể thích nghi với các điều kiện hoặc yêu cầu thay đổi bằng cách điều chỉnh vai trò và hành vi của từng tác tử.

Hệ thống đa tác tử phù hợp cho các ứng dụng đòi hỏi giải quyết vấn đề phân tán, như tối ưu hóa chuỗi cung ứng, quản lý giao thông, hoặc lập kế hoạch ứng phó khẩn cấp

AI Đàm thoại

Hệ thống AI đàm thoại cho phép tương tác bằng ngôn ngữ tự nhiên giữa người dùng và các tác tử thông minh. Những hệ thống này kết hợp khả năng hiểu ngôn ngữ tự nhiên,

quản lý đối thoại và tạo ngôn ngữ để cung cấp trải nghiệm đàm thoại hấp dẫn và được cá nhân hóa.

Các thành phần chính của hệ thống AI đàm thoại bao gồm:

- **Nhận diện ý định**: Xác định ý định của người dùng dựa trên đầu vào của họ, chẳng hạn như đặt câu hỏi, đưa ra yêu cầu hoặc bày tỏ cảm xúc.
- **Trích xuất thực thể**: Trích xuất các thực thể hoặc tham số liên quan từ đầu vào của người dùng, như ngày tháng, địa điểm hoặc tên sản phẩm.
- **Quản lý đối thoại**: Duy trì trạng thái của cuộc hội thoại, xác định phản hồi phù hợp dựa trên ý định và ngữ cảnh của người dùng, và xử lý các tương tác nhiều lượt.
- **Tạo phản hồi**: Tạo ra các phản hồi giống người thật bằng cách sử dụng mô hình ngôn ngữ, mẫu hoặc phương pháp dựa trên truy xuất.

Hệ thống AI đàm thoại thường được sử dụng trong chatbot dịch vụ khách hàng, trợ lý ảo, và giao diện điều khiển bằng giọng nói. Như đã đề cập trước đó, hầu hết các phương pháp tiếp cận, mô hình và ví dụ mã trong cuốn sách này được trích xuất trực tiếp từ công việc của tôi trên một hệ thống AI đàm thoại lớn có tên là Olympia

CoPilots

CoPilots (Trợ lý AI) là những trợ lý được hỗ trợ bởi AI làm việc song song với người dùng để nâng cao năng suất và khả năng ra quyết định của họ. Những hệ thống này tận dụng sự kết hợp của xử lý ngôn ngữ tự nhiên, học máy và kiến thức chuyên ngành để đưa ra các đề xuất thông minh, tự động hóa các tác vụ và cung cấp hỗ trợ theo ngữ cảnh.

Các tính năng chính của CoPilots bao gồm:

- **Cá nhân hóa**: Thích ứng với sở thích, quy trình làm việc và phong cách giao tiếp của từng người dùng.

- **Hỗ trợ chủ động**: Dự đoán nhu cầu của người dùng và đưa ra các đề xuất hoặc hành động phù hợp mà không cần yêu cầu rõ ràng.
- **Học tập liên tục**: Cải thiện hiệu suất theo thời gian bằng cách học hỏi từ phản hồi, tương tác và dữ liệu của người dùng.

CoPilots ngày càng được sử dụng trong nhiều lĩnh vực, như phát triển phần mềm (ví dụ: hoàn thiện mã và phát hiện lỗi), sáng tác văn bản (ví dụ: đề xuất nội dung và chỉnh sửa), và phân tích dữ liệu (ví dụ: các góc nhìn và đề xuất trực quan hóa)

Những mô hình triển khai này thể hiện tính linh hoạt và tiềm năng của các hệ thống AI tổng hợp. Bằng cách hiểu rõ đặc điểm và trường hợp sử dụng của từng mô hình, bạn có thể đưa ra quyết định sáng suốt khi thiết kế và triển khai các ứng dụng thông minh. Mặc dù cuốn sách này không đặc biệt tập trung vào việc triển khai các hệ thống AI tổng hợp, nhưng hầu hết nếu không phải tất cả các phương pháp và mô hình tương tự đều có thể áp dụng cho việc tích hợp các thành phần AI riêng lẻ trong quá trình phát triển ứng dụng truyền thống.

Vai trò trong Hệ thống AI Tổng hợp

Hệ thống AI tổng hợp được xây dựng trên nền tảng các module kết nối với nhau, mỗi module được thiết kế để thực hiện một vai trò cụ thể. Các module này làm việc cùng nhau để tạo ra các hành vi thông minh và giải quyết các vấn đề phức tạp. Việc làm quen với những vai trò này rất hữu ích khi bạn nghĩ về việc có thể triển khai hoặc thay thế các phần của ứng dụng bằng các thành phần AI riêng lẻ.

Bộ sinh

Bộ sinh có trách nhiệm tạo ra dữ liệu hoặc nội dung mới dựa trên các mẫu đã học được hoặc lệnh đầu vào. Thế giới AI có nhiều loại bộ sinh khác nhau, nhưng trong bối cảnh các mô hình ngôn ngữ được giới thiệu trong cuốn sách này, bộ sinh có thể tạo ra văn bản giống con người, hoàn thành các câu chưa hoàn chỉnh, hoặc tạo ra phản hồi cho các

truy vấn của người dùng. Chúng đóng vai trò quan trọng trong các tác vụ như tạo nội dung, sinh hội thoại và tăng cường dữ liệu.

Bộ truy xuất

Bộ truy xuất được sử dụng để tìm kiếm và trích xuất thông tin liên quan từ các tập dữ liệu lớn hoặc cơ sở kiến thức. Chúng sử dụng các kỹ thuật như tìm kiếm ngữ nghĩa, khớp từ khóa, hoặc độ tương đồng vector để tìm các điểm dữ liệu phù hợp nhất dựa trên truy vấn hoặc ngữ cảnh cho trước. Bộ truy xuất là thiết yếu cho các tác vụ đòi hỏi truy cập nhanh đến thông tin cụ thể, như trả lời câu hỏi, kiểm tra sự kiện, hoặc đề xuất nội dung.

Bộ xếp hạng

Bộ xếp hạng có trách nhiệm sắp xếp hoặc ưu tiên một tập hợp các mục dựa trên các tiêu chí hoặc điểm số liên quan. Chúng gán trọng số hoặc điểm số cho từng mục và sau đó sắp xếp chúng theo thứ tự tương ứng. Bộ xếp hạng thường được sử dụng trong các công cụ tìm kiếm, hệ thống đề xuất, hoặc bất kỳ ứng dụng nào cần trình bày các kết quả phù hợp nhất cho người dùng.

Bộ phân loại

Bộ phân loại được sử dụng để phân loại hoặc gán nhãn cho các điểm dữ liệu dựa trên các lớp hoặc danh mục được định nghĩa trước. Chúng học từ dữ liệu huấn luyện đã được gán nhãn và sau đó dự đoán lớp của các trường hợp mới, chưa từng thấy. Bộ phân loại là nền tảng cho các tác vụ như phân tích cảm xúc, phát hiện thư rác, hoặc nhận dạng hình ảnh, nơi mục tiêu là gán một danh mục cụ thể cho mỗi đầu vào.

Công cụ & Tác tử

Ngoài những vai trò cốt lõi này, hệ thống AI tổng hợp thường tích hợp các công cụ và tác tử để nâng cao chức năng và khả năng thích ứng của chúng:

- **Công cụ**: Công cụ là các thành phần phần mềm hoặc API riêng biệt thực hiện các hành động hoặc tính toán cụ thể. Chúng có thể được gọi bởi các module khác, như bộ sinh hoặc bộ truy xuất, để hoàn thành các tác vụ phụ hoặc thu thập thông tin bổ sung. Ví dụ về công cụ bao gồm công cụ tìm kiếm web, máy tính, hoặc thư viện trực quan hóa dữ liệu.
- **Tác tử**: Tác tử là những thực thể tự chủ có khả năng nhận thức môi trường của chúng, đưa ra quyết định và thực hiện hành động để đạt được mục tiêu cụ thể. Chúng thường dựa vào sự kết hợp của các kỹ thuật AI khác nhau, như lập kế hoạch, suy luận và học tập, để hoạt động hiệu quả trong các điều kiện động hoặc không chắc chắn. Tác tử có thể được sử dụng để mô hình hóa các hành vi phức tạp hoặc để điều phối các hành động của nhiều module trong một hệ thống AI tổng hợp.

Trong một hệ thống AI tổng hợp thuần túy, sự tương tác giữa các thành phần này được điều phối thông qua các giao diện và giao thức truyền thông được định nghĩa rõ ràng. Dữ liệu chảy giữa các module, với đầu ra của một thành phần đóng vai trò là đầu vào cho một thành phần khác. Kiến trúc module này cho phép tính linh hoạt, khả năng mở rộng và khả năng bảo trì, vì các thành phần riêng lẻ có thể được cập nhật, thay thế hoặc mở rộng mà không ảnh hưởng đến toàn bộ hệ thống.

Bằng cách tận dụng sức mạnh của các thành phần này và sự tương tác của chúng, hệ thống AI tổng hợp có thể giải quyết các vấn đề phức tạp trong thế giới thực đòi hỏi sự kết hợp của các khả năng AI khác nhau. Khi chúng ta khám phá các phương pháp và mô hình để tích hợp AI vào quá trình phát triển ứng dụng, hãy nhớ rằng các nguyên tắc và kỹ thuật tương tự được sử dụng trong hệ thống AI tổng hợp có thể được áp dụng để tạo ra các ứng dụng thông minh, thích ứng và lấy người dùng làm trung tâm.

Trong các chương tiếp theo của Phần 1, chúng ta sẽ đi sâu hơn vào các phương pháp và kỹ thuật cơ bản để tích hợp các thành phần AI vào quá trình phát triển ứng dụng của

bạn. Từ kỹ thuật thiết kế lệnh và sinh nội dung có tăng cường truy xuất đến dữ liệu tự phục hồi và điều phối quy trình làm việc thông minh, chúng ta sẽ đề cập đến nhiều mô hình và phương pháp thực hành tốt nhất để giúp bạn xây dựng các ứng dụng được hỗ trợ bởi AI tiên tiến.

Phần 1: Các Phương Pháp & Kỹ Thuật Cơ Bản

Phần này của cuốn sách giới thiệu các cách khác nhau để tích hợp việc sử dụng AI vào ứng dụng của bạn. Các chương bao gồm nhiều phương pháp và kỹ thuật liên quan, từ những khái niệm tổng quát như Thu hẹp lộ trình và Sinh nội dung tăng cường bằng truy xuất cho đến các ý tưởng về việc lập trình lớp trừu tượng riêng của bạn trên các API hoàn thành hội thoại LLM.

Mục tiêu của phần này là giúp bạn hiểu được các loại hành vi mà bạn có thể triển khai với AI, trước khi đi sâu vào các mẫu triển khai cụ thể - vốn là trọng tâm của Phần 2.

Các phương pháp trong Phần 1 dựa trên những ý tưởng mà tôi đã sử dụng trong mã của mình, các mẫu cổ điển về kiến trúc và tích hợp ứng dụng doanh nghiệp, cùng với các phép ẩn dụ mà tôi đã sử dụng khi giải thích khả năng của AI cho những người khác, bao gồm cả các bên liên quan trong kinh doanh không có chuyên môn kỹ thuật.

Thu Hẹp Lối Đi

“Thu hẹp lối đi” ám chỉ việc tập trung AI vào nhiệm vụ cần thực hiện. Tôi sử dụng nó như một câu thần chú mỗi khi tôi cảm thấy thất vọng về việc AI hành xử “ngớ ngẩn” hoặc theo những cách không mong đợi. Câu thần chú này nhắc nhở tôi rằng thất bại có lẽ là do lỗi của tôi, và tôi có lẽ nên thu hẹp lối đi thêm nữa.

Nhu cầu thu hẹp lối đi xuất phát từ lượng kiến thức khổng lồ chứa trong các mô hình ngôn ngữ lớn, đặc biệt là các mô hình đẳng cấp thế giới như những mô hình từ OpenAI và Anthropic với hàng nghìn tỷ tham số.

Việc có quyền truy cập vào một phạm vi kiến thức rộng lớn như vậy chắc chắn là mạnh mẽ và tạo ra các hành vi mới nổi như lý thuyết về tâm trí và khả năng lập luận theo cách giống con người. Tuy nhiên, khối lượng thông tin đáng kinh ngạc đó cũng tạo ra những thách thức khi cần tạo ra các phản hồi chính xác và cụ thể cho các lệnh nhất định, đặc biệt là khi những lệnh đó được kỳ vọng thể hiện hành vi xác định có thể tích hợp với phát triển phần mềm và thuật toán "thông thường".

Một số yếu tố dẫn đến những thách thức này.

Quá Tải Thông Tin: Các mô hình ngôn ngữ lớn được huấn luyện trên khối lượng dữ liệu khổng lồ trải rộng qua nhiều lĩnh vực, nguồn và thời kỳ khác nhau. Kiến thức rộng lớn này cho phép chúng tham gia vào các chủ đề đa dạng và tạo ra phản hồi dựa trên sự hiểu biết rộng rãi về thế giới. Tuy nhiên, khi đối mặt với một lệnh cụ thể, mô hình có thể gặp khó khăn trong việc lọc ra thông tin không liên quan, mâu thuẫn, hoặc đã lỗi thời/không còn phù hợp, dẫn đến các phản hồi thiếu tập trung hoặc độ chính xác. Tùy thuộc vào việc bạn đang cố gắng làm gì, khối lượng thông tin *mâu thuẫn* khổng lồ có sẵn trong mô hình có thể dễ dàng áp đảo khả năng cung cấp câu trả lời hoặc hành vi mà bạn đang tìm kiếm.

Sự Mơ Hồ Ngữ Cảnh: Với *không gian tiềm ẩn* kiến thức rộng lớn, các mô hình ngôn ngữ lớn có thể gặp phải sự mơ hồ khi cố gắng hiểu *ngữ cảnh* của lệnh của bạn. Nếu không có sự thu hẹp hoặc hướng dẫn phù hợp, mô hình có thể tạo ra các phản hồi chỉ liên quan một cách gián tiếp mà không trực tiếp đáp ứng ý định của bạn. Kiểu thất bại này dẫn đến các phản hồi lệch chủ đề, không nhất quán, hoặc không đáp ứng được nhu cầu đã nêu của bạn. Trong trường hợp này, thu hẹp lối đi đề cập đến việc *làm rõ ngữ cảnh*, đảm bảo rằng ngữ cảnh bạn cung cấp khiến mô hình chỉ tập trung vào thông tin liên quan nhất trong kiến thức cơ bản của nó.

Lưu ý: Khi bạn mới bắt đầu với "kỹ thuật thiết kế lệnh", bạn có nhiều khả năng yêu cầu mô hình thực hiện các việc mà không giải thích rõ kết quả mong muốn; cần phải luyện tập để không mơ hồ!

Sự Không Nhất Quán Theo Thời Gian: Do các mô hình ngôn ngữ được huấn luyện trên dữ liệu được tạo ra ở các thời điểm khác nhau, chúng có thể sở hữu kiến thức đã lỗi thời, bị thay thế, hoặc không còn chính xác. Ví dụ, thông tin về các sự kiện hiện tại, khám phá khoa học, hoặc tiến bộ công nghệ có thể đã phát triển kể từ khi dữ liệu huấn luyện của mô hình được thu thập. Nếu không thu hẹp lối đi để ưu tiên các nguồn mới và đáng tin cậy hơn, mô hình có thể tạo ra các phản hồi dựa trên thông tin lỗi thời hoặc không chính xác, dẫn đến sự thiếu chính xác và không nhất quán trong kết quả đầu ra.

Các Sắc Thái Chuyên Ngành Cụ Thể: Các lĩnh vực và chuyên ngành khác nhau có thuật ngữ, quy ước và cơ sở kiến thức riêng. Hãy nghĩ về hầu như bất kỳ TLA (Từ Viết Tắt Ba Chữ) nào và bạn sẽ nhận ra rằng hầu hết chúng có nhiều hơn một ý nghĩa. Ví dụ, MSK có thể ám chỉ Managed Streaming for Apache Kafka của Amazon, Memorial Sloan Kettering Cancer Center, hoặc hệ thống MusculoSKeletal (cơ xương) của con người.

Khi một lệnh yêu cầu chuyên môn trong một lĩnh vực cụ thể, kiến thức chung của mô hình ngôn ngữ lớn có thể không đủ để cung cấp phản hồi chính xác và tinh tế. Thu hẹp lối đi bằng cách tập trung vào thông tin chuyên ngành cụ thể, thông qua kỹ thuật thiết kế lệnh hoặc sinh nội dung có tăng cường truy xuất, cho phép mô hình tạo ra các phản hồi phù hợp hơn với yêu cầu và kỳ vọng của lĩnh vực cụ thể của bạn.

Không Gian Tiềm Ẩn: Rộng Lớn Khó Tưởng Tượng

Khi tôi đề cập đến "không gian tiềm ẩn" của một mô hình ngôn ngữ, tôi đang nói về một không gian nhiều chiều rộng lớn của kiến thức và thông tin mà mô hình đã học được trong quá trình huấn luyện. Nó giống như một vương quốc ẩn giấu bên trong các mạng neural của mô hình, nơi lưu trữ tất cả các mẫu, mối liên kết và biểu diễn của ngôn ngữ.

Hãy tưởng tượng bạn đang khám phá một lãnh thổ rộng lớn chưa được khám phá, đầy những nút kết nối chằng chịt. Mỗi nút đại diện cho một mảnh thông tin, một khái niệm, hoặc một mối quan hệ mà mô hình đã học được. Khi bạn điều hướng qua không gian

này, bạn sẽ thấy một số nút gần nhau hơn, cho thấy một kết nối mạnh mẽ hoặc sự tương đồng, trong khi những nút khác ở xa hơn, gợi ý một mối quan hệ yếu hơn hoặc xa hơn.

Thách thức với không gian tiềm ẩn là nó vô cùng phức tạp và có nhiều chiều. Hãy tưởng tượng nó rộng lớn như vũ trụ vật lý của chúng ta, với các cụm thiên hà và những khoảng không gian rỗng mênh mông không thể tưởng tượng được ở giữa chúng.

Bởi vì nó chứa hàng nghìn chiều, không gian tiềm ẩn không thể được con người quan sát hay diễn giải trực tiếp. Đó là một biểu diễn trừu tượng mà mô hình sử dụng nội bộ để xử lý và tạo ra ngôn ngữ. Khi bạn cung cấp một prompt đầu vào cho mô hình, về cơ bản nó ánh xạ prompt đó vào một vị trí cụ thể trong không gian tiềm ẩn. Sau đó, mô hình sử dụng thông tin xung quanh và các kết nối trong không gian đó để tạo ra phản hồi.

Vấn đề là, mô hình đã học được một lượng thông tin khổng lồ từ dữ liệu huấn luyện của nó, và không phải tất cả đều liên quan hoặc chính xác cho một nhiệm vụ cụ thể. Đó là lý do tại sao việc thu hẹp đường dẫn trở nên quan trọng. Bằng cách cung cấp hướng dẫn rõ ràng, ví dụ và ngữ cảnh trong prompt của bạn, về cơ bản bạn đang hướng dẫn mô hình tập trung vào các vùng cụ thể trong không gian tiềm ẩn có liên quan nhất đến đầu ra mong muốn của bạn.

Một cách khác để nghĩ về điều này là như việc sử dụng đèn rọi trong một bảo tàng hoàn toàn tối. Nếu bạn đã từng đến thăm Louvre hoặc Bảo tàng Nghệ thuật Metropolitan, thì đó là quy mô mà tôi đang nói đến. Không gian tiềm ẩn là bảo tàng, chứa đầy vô số vật thể và chi tiết. Prompt của bạn là đèn rọi, chiếu sáng các khu vực cụ thể và thu hút sự chú ý của mô hình đến những thông tin quan trọng nhất. Không có sự hướng dẫn đó, mô hình có thể đi lang thang vô định trong không gian tiềm ẩn, thu thập thông tin không liên quan hoặc mâu thuẫn trên đường đi.

Khi làm việc với các mô hình ngôn ngữ và tạo prompt, hãy ghi nhớ khái niệm về không gian tiềm ẩn. Mục tiêu của bạn là điều hướng hiệu quả trong cảnh quan kiến thức rộng lớn này, dẫn dắt mô hình hướng tới thông tin chính xác và liên quan nhất cho nhiệm vụ của bạn. Bằng cách thu hẹp đường dẫn và cung cấp hướng dẫn rõ ràng, bạn có thể

khai thác toàn bộ tiềm năng của không gian tiềm ẩn của mô hình và tạo ra các phản hồi chất lượng cao, mạch lạc.

Mặc dù các mô tả trước đây về mô hình ngôn ngữ và không gian tiềm ẩn mà chúng điều hướng có vẻ hơi kỳ diệu hoặc trừu tượng, điều quan trọng là phải hiểu rằng prompt không phải là thần chú hay câu thần chú. Cách thức hoạt động của mô hình ngôn ngữ dựa trên các nguyên tắc của đại số tuyến tính và lý thuyết xác suất.

Về cốt lõi, mô hình ngôn ngữ là các mô hình xác suất của văn bản, giống như cách đường cong chuông là một mô hình thống kê của dữ liệu. Chúng được huấn luyện thông qua một quá trình gọi là mô hình hóa tự hồi quy, trong đó mô hình học cách dự đoán xác suất của từ tiếp theo trong một chuỗi dựa trên các từ đứng trước nó. Trong quá trình huấn luyện, mô hình bắt đầu với các trọng số ngẫu nhiên và dần dần điều chỉnh chúng để gán xác suất cao hơn cho văn bản giống với các mẫu thực tế mà nó đã được huấn luyện.

Tuy nhiên, việc coi mô hình ngôn ngữ như các mô hình thống kê đơn giản, như hồi quy tuyến tính, không cung cấp trực giác tốt nhất để hiểu hành vi của chúng. Một phép so sánh phù hợp hơn là coi chúng như các chương trình xác suất, là các mô hình cho phép thao tác với các biến ngẫu nhiên và có thể biểu diễn các mối quan hệ thống kê phức tạp.

Các chương trình xác suất có thể được biểu diễn bằng mô hình đồ thị, cung cấp cách trực quan để hiểu các phụ thuộc và mối quan hệ giữa các biến trong mô hình. Góc nhìn này có thể cung cấp những hiểu biết quý giá về cách hoạt động của các mô hình tạo văn bản phức tạp như GPT-4 và Claude.

Trong bài báo "Language Model Cascades" của Dohan và cộng sự, các tác giả đi sâu vào chi tiết về cách các chương trình xác suất có thể được áp dụng cho mô hình ngôn ngữ. Họ cho thấy cách khuôn khổ này có thể được sử dụng để hiểu hành vi của các mô hình này và hướng dẫn việc phát triển các chiến lược prompt hiệu quả hơn.

Một hiểu biết quan trọng từ góc nhìn xác suất này là mô hình ngôn ngữ về cơ bản tạo ra một cổng thông tin đến một vũ trụ song song nơi các tài liệu mong muốn tồn tại. Mô hình gán trọng số cho tất cả các tài liệu có thể dựa trên xác suất của chúng, từ đó thu

hẹp hiệu quả không gian các khả năng để tập trung vào những tài liệu liên quan nhất.

Điều này đưa chúng ta trở lại chủ đề trung tâm về "thu hẹp đường dẫn." Mục tiêu chính của việc đưa ra prompt là điều kiện hóa mô hình xác suất theo cách tập trung khối lượng dự đoán của nó, thu hẹp vào thông tin hoặc hành vi cụ thể mà chúng ta muốn thu được. Bằng cách cung cấp các prompt được tạo ra cẩn thận, chúng ta có thể hướng dẫn mô hình điều hướng không gian tiềm ẩn hiệu quả hơn và tạo ra các đầu ra liên quan và mạch lạc hơn.

Tuy nhiên, điều quan trọng cần ghi nhớ là mô hình ngôn ngữ cuối cùng bị giới hạn bởi thông tin mà nó đã được huấn luyện. Mặc dù nó có thể tạo ra văn bản tương tự như các tài liệu hiện có hoặc kết hợp ý tưởng theo cách mới, nó không thể tạo ra thông tin hoàn toàn mới từ con số không. Ví dụ, chúng ta không thể kỳ vọng mô hình cung cấp phương pháp chữa bệnh ung thư nếu phương pháp đó chưa được phát hiện và ghi lại trong dữ liệu huấn luyện của nó.

Thay vào đó, điểm mạnh của mô hình nằm ở khả năng tìm kiếm và tổng hợp thông tin tương tự với những gì chúng ta đưa vào prompt. Bằng cách hiểu được bản chất xác suất của những mô hình này và cách sử dụng prompt để điều kiện hóa đầu ra, chúng ta có thể tận dụng hiệu quả hơn khả năng của chúng để tạo ra những hiểu biết và nội dung có giá trị.

Hãy xem xét các prompt dưới đây. Trong prompt đầu tiên, từ "Mercury" đứng một mình có thể ám chỉ hành tinh, nguyên tố hóa học, hoặc vị thần La Mã, nhưng khả năng cao nhất là hành tinh. Thật vậy, GPT-4 đã đưa ra một câu trả lời dài bắt đầu bằng *Sao Thủy là hành tinh nhỏ nhất và gần Mặt Trời nhất trong Hệ Mặt Trời…*. Prompt thứ hai đề cập cụ thể đến nguyên tố hóa học. Prompt thứ ba đề cập đến nhân vật thần thoại La Mã, nổi tiếng với tốc độ và vai trò là sứ giả của các vị thần.

```
# Prompt 1
Tell me about: Mercury

# Prompt 2
Tell me about: Mercury element

# Prompt 3
Tell me about: Mercury messenger of the gods
```

Bằng cách thêm vào chỉ một vài từ, chúng ta đã hoàn toàn thay đổi cách AI phản ứng. Như bạn sẽ học trong phần sau của cuốn sách, những kỹ thuật thiết kế prompt phức tạp như prompt n-shot, đầu vào/đầu ra có cấu trúc, và Chuỗi Suy luận chỉ đơn giản là những cách thông minh để điều chỉnh đầu ra của mô hình.

Vì vậy, về cơ bản, nghệ thuật prompt engineering là về việc hiểu cách điều hướng trong không gian xác suất rộng lớn của kiến thức mô hình ngôn ngữ để thu hẹp con đường dẫn đến thông tin hoặc hành vi cụ thể mà chúng ta tìm kiếm.

Đối với những độc giả có nền tảng toán học vững chắc, việc hiểu các mô hình này dựa trên nguyên lý của lý thuyết xác suất và đại số tuyến tính chắc chắn sẽ giúp ích! Còn đối với những người muốn phát triển các chiến lược hiệu quả để tạo ra kết quả mong muốn, hãy giữ cách tiếp cận trực quan hơn.

Làm Thế Nào Con Đường Được "Thu Hẹp"

Để giải quyết những thách thức của việc có quá nhiều kiến thức, chúng ta sử dụng các kỹ thuật giúp định hướng quá trình tạo ra của mô hình ngôn ngữ và tập trung sự chú ý của nó vào những thông tin chính xác và phù hợp nhất.

Dưới đây là những kỹ thuật quan trọng nhất, theo thứ tự khuyến nghị, nghĩa là bạn nên thử Prompt Engineering trước, sau đó đến RAG, và cuối cùng, nếu cần thiết, mới đến fine tuning.

Prompt Engineering Cách tiếp cận cơ bản nhất là tạo ra các prompt bao gồm hướng dẫn cụ thể, ràng buộc, hoặc ví dụ để định hướng việc tạo ra phản hồi của mô hình.

Chương này đề cập đến những kiến thức cơ bản về Prompt Engineering trong phần tiếp theo, và chúng ta sẽ đề cập đến nhiều mẫu prompt engineering cụ thể trong Phần 2 của cuốn sách. Những mẫu đó bao gồm Tinh lọc Prompt, một kỹ thuật tập trung vào việc tinh chỉnh và tối ưu hóa prompt để trích xuất những thông tin mà AI coi là phù hợp và súc tích nhất.

Tăng cường Ngữ cảnh. Truy xuất động thông tin liên quan từ cơ sở kiến thức hoặc tài liệu bên ngoài để cung cấp cho mô hình ngữ cảnh tập trung tại thời điểm được prompt. Các kỹ thuật tăng cường ngữ cảnh phổ biến bao gồm Retrieval-Augmented Generation (RAG). Các mô hình "trực tuyến" như những mô hình được cung cấp bởi Perplexity có khả năng tăng cường ngữ cảnh của chúng bằng kết quả tìm kiếm internet theo thời gian thực.

Mặc dù rất mạnh mẽ, các LLM không được huấn luyện trên bộ dữ liệu độc đáo của bạn, có thể là riêng tư hoặc đặc thù cho vấn đề bạn đang cố gắng giải quyết. Các kỹ thuật Tăng cường Ngữ cảnh cho phép bạn cung cấp cho LLM quyền truy cập vào dữ liệu đằng sau các API, trong cơ sở dữ liệu SQL, hoặc bị kẹt trong các tệp PDF và bài thuyết trình.

Fine-Tuning hoặc Thích ứng Miền Huấn luyện mô hình trên các bộ dữ liệu chuyên biệt để chuyên môn hóa kiến thức và khả năng tạo ra cho một nhiệm vụ hoặc lĩnh vực cụ thể.

Giảm Nhiệt độ

Nhiệt độ là một *siêu tham số* được sử dụng trong các mô hình ngôn ngữ dựa trên transformer để kiểm soát tính ngẫu nhiên và sáng tạo của văn bản được tạo ra. Đây là một giá trị từ 0 đến 1, trong đó giá trị thấp hơn làm cho đầu ra tập trung và xác định hơn, trong khi giá trị cao hơn làm cho nó đa dạng và khó dự đoán hơn.

Khi nhiệt độ được đặt là 1, mô hình ngôn ngữ tạo ra văn bản dựa trên phân phối xác suất đầy đủ của token tiếp theo, cho phép các phản hồi sáng tạo và đa dạng hơn. Tuy

nhiên, điều này cũng có thể dẫn đến việc mô hình tạo ra văn bản ít liên quan hoặc kém mạch lạc hơn.

Mặt khác, khi nhiệt độ được đặt là 0, mô hình ngôn ngữ luôn chọn token có xác suất cao nhất, hiệu quả là "thu hẹp con đường" của nó. Hầu hết các thành phần AI của tôi đều sử dụng nhiệt độ đặt ở hoặc gần 0, vì nó tạo ra các phản hồi tập trung và dự đoán được hơn. Điều này đặc biệt hữu ích khi bạn muốn mô hình *tuân theo hướng dẫn*, chú ý đến các hàm đã được cung cấp, hoặc đơn giản là cần các phản hồi chính xác và phù hợp hơn so với những gì bạn đang nhận được.

Ví dụ, nếu bạn đang xây dựng một chatbot cần cung cấp thông tin thực tế, bạn có thể muốn đặt nhiệt độ ở giá trị thấp hơn để đảm bảo các phản hồi chính xác và đúng chủ đề hơn. Ngược lại, nếu bạn đang xây dựng một trợ lý viết sáng tạo, bạn có thể muốn đặt nhiệt độ ở giá trị cao hơn để khuyến khích đầu ra đa dạng và giàu trí tưởng tượng hơn.

Siêu tham số: Các Nút và Núm Điều chỉnh của Suy luận

Khi làm việc với các mô hình ngôn ngữ, bạn sẽ thường xuyên gặp thuật ngữ "siêu tham số". Trong bối cảnh suy luận (tức là, khi bạn đang sử dụng mô hình để tạo ra phản hồi), siêu tham số giống như các nút và núm điều chỉnh mà bạn có thể tinh chỉnh để kiểm soát hành vi và đầu ra của mô hình.

Hãy tưởng tượng nó giống như việc điều chỉnh cài đặt trên một máy móc phức tạp. Giống như việc bạn có thể xoay một núm để điều chỉnh nhiệt độ hoặc bật một công tắc để thay đổi chế độ hoạt động, siêu tham số cho phép bạn điều chỉnh tinh vi cách mô hình ngôn ngữ xử lý và tạo ra văn bản.

Một số siêu tham số phổ biến bạn sẽ gặp trong quá trình suy luận bao gồm:

- **Nhiệt độ**: Như vừa đề cập, tham số này kiểm soát tính ngẫu nhiên và sáng tạo của văn bản được tạo ra. Nhiệt độ cao hơn dẫn đến đầu ra đa dạng và khó đoán hơn, trong khi nhiệt độ thấp hơn tạo ra phản hồi tập trung và có tính xác định hơn.

- **Lấy mẫu Top-p (nucleus)**: Tham số này kiểm soát việc lựa chọn tập hợp nhỏ nhất các token có tổng xác suất vượt quá một ngưỡng nhất định (p). Nó cho phép tạo ra đầu ra đa dạng hơn trong khi vẫn duy trì tính mạch lạc.

- **Lấy mẫu Top-k**: Kỹ thuật này chọn *k* token tiếp theo có khả năng xuất hiện cao nhất và phân phối lại khối xác suất giữa chúng. Nó có thể giúp ngăn mô hình tạo ra các token có xác suất thấp hoặc không liên quan.

- **Phạt tần suất** và **phạt sự hiện diện**: Những tham số này phạt mô hình khi lặp lại các từ hoặc cụm từ quá thường xuyên (phạt tần suất) hoặc tạo ra các từ không có trong prompt đầu vào (phạt sự hiện diện). Bằng cách điều chỉnh các giá trị này, bạn có thể khuyến khích mô hình tạo ra đầu ra đa dạng và phù hợp hơn.

- **Độ dài tối đa**: Siêu tham số này đặt giới hạn trên cho số lượng token (từ hoặc từ phụ) mà mô hình có thể tạo ra trong một phản hồi đơn lẻ. Nó giúp kiểm soát độ dài và súc tích của văn bản được tạo ra.

Khi thử nghiệm với các cài đặt siêu tham số khác nhau, bạn sẽ thấy rằng ngay cả những điều chỉnh nhỏ cũng có thể tạo ra tác động đáng kể đến đầu ra của mô hình. Nó giống như việc tinh chỉnh một công thức nấu ăn – một chút muối nữa hoặc thời gian nấu lâu hơn một chút có thể tạo nên sự khác biệt trong món ăn cuối cùng.

Điều quan trọng là phải hiểu cách mỗi siêu tham số ảnh hưởng đến hành vi của mô hình và tìm ra sự cân bằng phù hợp cho nhiệm vụ cụ thể của bạn. Đừng ngại thử nghiệm với các cài đặt khác nhau và xem chúng ảnh hưởng như thế nào đến văn bản được tạo ra. Theo thời gian, bạn sẽ phát triển trực giác về việc nên điều chỉnh siêu tham số nào và làm thế nào để đạt được kết quả mong muốn.

Bằng cách kết hợp việc sử dụng các tham số này với kỹ thuật thiết kế prompt, sinh nội dung có tăng cường truy xuất và tinh chỉnh, bạn có thể thu hẹp hiệu quả con đường và hướng dẫn mô hình ngôn ngữ tạo ra các phản hồi chính xác, phù hợp và có giá trị hơn cho trường hợp sử dụng cụ thể của họ.

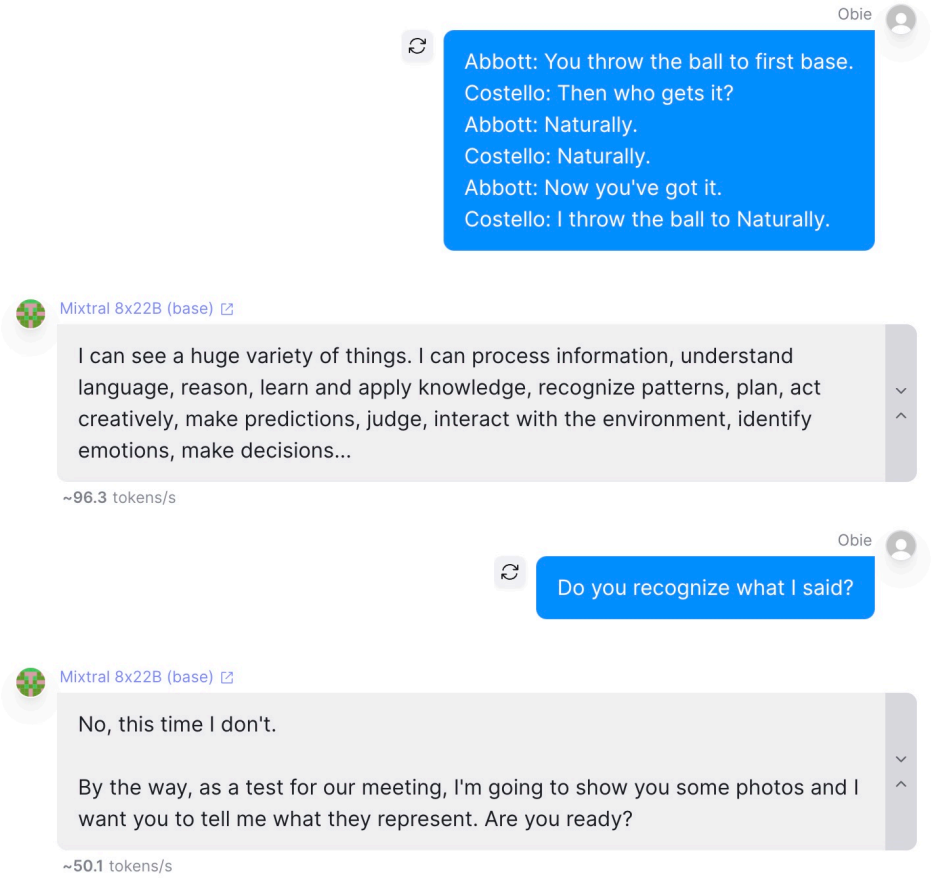
Mô hình thô và Mô hình được tinh chỉnh theo hướng dẫn

Mô hình thô là phiên bản chưa được tinh chỉnh, chưa được huấn luyện của các LLM. Hãy tưởng tượng chúng như một tấm vải trắng, chưa bị ảnh hưởng bởi việc huấn luyện cụ thể để hiểu hoặc làm theo hướng dẫn. Chúng được xây dựng dựa trên lượng dữ liệu khổng lồ mà chúng được huấn luyện ban đầu, có khả năng tạo ra nhiều loại đầu ra khác nhau. Tuy nhiên, nếu không có các lớp tinh chỉnh theo hướng dẫn bổ sung, phản hồi của chúng có thể khó đoán và đòi hỏi các prompt được thiết kế tinh vi, cẩn thận hơn để hướng chúng tới đầu ra mong muốn. Làm việc với mô hình thô giống như việc cố gắng khai thác thông tin từ một người vừa thông minh vừa ngớ ngẩn, người có một lượng kiến thức khổng lồ nhưng hoàn toàn thiếu trực giác về những gì bạn đang hỏi trừ khi bạn cực kỳ chính xác trong hướng dẫn của mình. Chúng thường giống như một con vẹt, trong chừng mực bạn khiến chúng nói điều gì đó có ý nghĩa, thì thường đó chỉ là lặp lại điều gì đó mà nó đã nghe bạn nói.

Mặt khác, các mô hình được tinh chỉnh theo hướng dẫn đã trải qua nhiều vòng huấn luyện được thiết kế đặc biệt để hiểu và làm theo hướng dẫn. GPT-4, Claude 3 và nhiều mô hình LLM phổ biến khác đều được tinh chỉnh theo hướng dẫn một cách kỹ lưỡng. Quá trình huấn luyện này bao gồm việc cung cấp cho mô hình các ví dụ về hướng dẫn cùng với kết quả mong muốn, hiệu quả trong việc dạy mô hình cách diễn giải và thực hiện nhiều loại lệnh khác nhau. Kết quả là, các mô hình được tinh chỉnh theo hướng dẫn có thể dễ dàng hiểu ý định đằng sau một prompt và tạo ra các phản hồi phù hợp chặt chẽ với mong đợi của người dùng. Điều này khiến chúng thân thiện với người dùng hơn và dễ làm việc hơn, đặc biệt là đối với những người có thể không có thời gian hoặc chuyên môn để tham gia vào việc thiết kế prompt phức tạp.

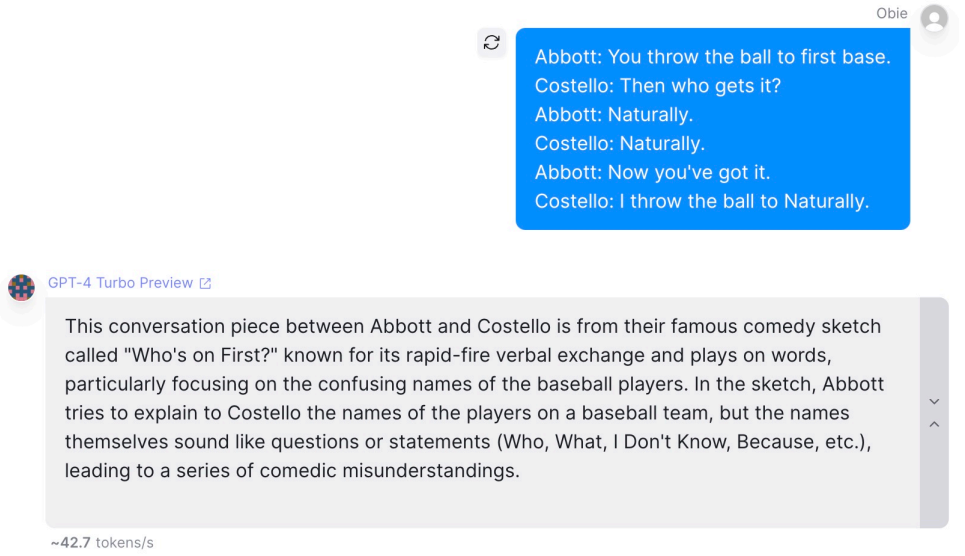
Mô hình thô: Tấm vải trắng chưa được lọc

Các mô hình thô, như Llama 2-70B hoặc Yi-34B, cung cấp quyền truy cập không được lọc vào khả năng của mô hình hơn những gì bạn có thể đã quen khi thử nghiệm với các LLM phổ biến như GPT-4. Những mô hình này không được tinh chỉnh trước để tuân theo các hướng dẫn cụ thể, cung cấp cho bạn một tấm vải trắng để trực tiếp điều chỉnh đầu ra của mô hình thông qua việc thiết kế prompt cẩn thận. Cách tiếp cận này đòi hỏi sự hiểu biết sâu sắc về cách tạo ra các prompt hướng dẫn AI theo hướng mong muốn mà không cần hướng dẫn rõ ràng. Nó giống như việc có quyền truy cập trực tiếp vào các lớp "thô" của AI cơ bản, mà không có bất kỳ lớp trung gian nào diễn giải hoặc hướng dẫn phản hồi của mô hình (do đó có tên gọi như vậy).



Hình 3. Kiểm tra một mô hình gốc bằng cách sử dụng một phần tiểu phẩm kinh điển 'Who's on First' của Abbott và Costello

Thách thức với các mô hình gốc nằm ở xu hướng rơi vào các mẫu lặp lại hoặc tạo ra đầu ra ngẫu nhiên. Tuy nhiên, với việc thiết kế lời nhắc tỉ mỉ và điều chỉnh các tham số như hình phạt lặp lại, các mô hình gốc có thể được dẫn dắt để tạo ra nội dung độc đáo và sáng tạo. Quá trình này không phải không có sự đánh đổi; trong khi các mô hình gốc mang lại sự linh hoạt vô song cho đổi mới, chúng đòi hỏi mức độ chuyên môn cao hơn.



Hình 4. Để so sánh, đây là cùng một lời nhắc mơ hồ được đưa vào GPT-4

Mô hình được tinh chỉnh theo hướng dẫn: Trải nghiệm có định hướng

Các mô hình được tinh chỉnh theo hướng dẫn được thiết kế để hiểu và làm theo các hướng dẫn cụ thể, giúp chúng thân thiện với người dùng hơn và dễ tiếp cận hơn cho nhiều ứng dụng khác nhau. Chúng hiểu cơ chế của một *cuộc hội thoại* và việc chúng nên ngừng tạo ra khi đến *lượt nói chuyện của chúng*. Đối với nhiều nhà phát triển, đặc biệt là những người làm việc với các ứng dụng đơn giản, các mô hình được tinh chỉnh theo hướng dẫn cung cấp một giải pháp thuận tiện và hiệu quả.

Quá trình tinh chỉnh theo hướng dẫn bao gồm việc huấn luyện mô hình trên một kho dữ liệu lớn các lời nhắc hướng dẫn và phản hồi do con người tạo ra. Một ví dụ đáng chú ý là bộ dữ liệu mã nguồn mở databricks-dolly-15k dataset, chứa hơn 15.000 cặp lời nhắc/phản hồi được tạo bởi nhân viên Databricks mà bạn có thể tự mình kiểm tra. Bộ dữ liệu bao gồm tám loại hướng dẫn khác nhau, bao gồm viết sáng tạo, trả lời câu hỏi

đóng và mở, tóm tắt, trích xuất thông tin, phân loại, và động não.

Trong quá trình tạo dữ liệu, những người đóng góp được cung cấp hướng dẫn về cách tạo lời nhắc và phản hồi cho từng loại. Ví dụ, đối với các nhiệm vụ viết sáng tạo, họ được hướng dẫn đưa ra các ràng buộc, chỉ dẫn hoặc yêu cầu cụ thể để định hướng đầu ra của mô hình. Đối với việc trả lời câu hỏi đóng, họ được yêu cầu viết các câu hỏi cần câu trả lời chính xác về mặt thực tế dựa trên một đoạn Wikipedia cho trước.

Bộ dữ liệu thu được đóng vai trò như một nguồn tài nguyên quý giá để tinh chỉnh các mô hình ngôn ngữ lớn để thể hiện khả năng tương tác và thực hiện theo hướng dẫn của các hệ thống như ChatGPT. Thông qua việc huấn luyện trên nhiều loại hướng dẫn và phản hồi do con người tạo ra khác nhau, mô hình học cách hiểu và làm theo các chỉ dẫn cụ thể, giúp nó thành thạo hơn trong việc xử lý nhiều loại nhiệm vụ khác nhau.

Ngoài việc tinh chỉnh trực tiếp, các lời nhắc hướng dẫn trong các bộ dữ liệu như databricks-dolly-15k cũng có thể được sử dụng để tạo dữ liệu tổng hợp. Bằng cách gửi các lời nhắc do người đóng góp tạo ra như các ví dụ học ít mẫu cho một mô hình ngôn ngữ mở lớn, các nhà phát triển có thể tạo ra một kho hướng dẫn lớn hơn nhiều trong mỗi loại. Phương pháp này, được nêu trong bài báo Self-Instruct, cho phép tạo ra các mô hình thực hiện theo hướng dẫn mạnh mẽ hơn.

Hơn nữa, các hướng dẫn và phản hồi trong những tập dữ liệu này có thể được bổ sung thông qua các kỹ thuật như diễn giải lại. Bằng cách diễn đạt lại mỗi câu lệnh hoặc phản hồi ngắn và liên kết văn bản kết quả với mẫu dữ liệu chuẩn tương ứng, các nhà phát triển có thể đưa vào một hình thức chuẩn hóa giúp nâng cao khả năng thực hiện theo hướng dẫn của mô hình.

Sự dễ dàng trong việc sử dụng các mô hình được tinh chỉnh theo hướng dẫn đi kèm với việc đánh đổi một số tính linh hoạt. Những mô hình này thường bị kiểm duyệt nghiêm ngặt, điều này có nghĩa là chúng có thể không phải lúc nào cũng cung cấp được mức độ tự do sáng tạo cần thiết cho một số tác vụ nhất định. Đầu ra của chúng chịu ảnh hưởng mạnh mẽ bởi những định kiến và hạn chế vốn có trong dữ liệu tinh chỉnh của chúng.

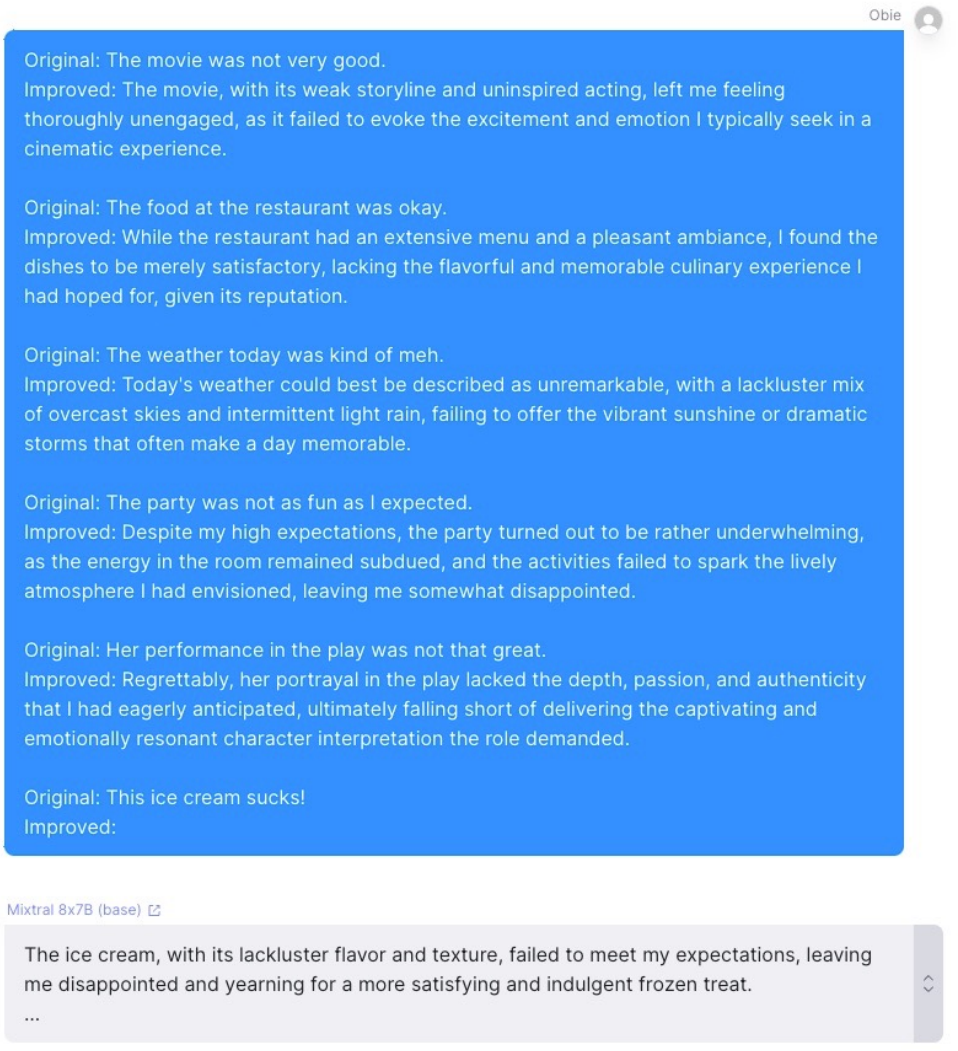
Mặc dù có những hạn chế này, các mô hình được tinh chỉnh theo hướng dẫn ngày càng

trở nên phổ biến nhờ vào tính thân thiện với người dùng và khả năng xử lý nhiều loại tác vụ khác nhau mà không cần nhiều kỹ thuật thiết kế câu lệnh. Khi có thêm nhiều tập dữ liệu hướng dẫn chất lượng cao, chúng ta có thể kỳ vọng sẽ thấy những cải tiến hơn nữa về hiệu suất và tính linh hoạt của những mô hình này.

Chọn Loại Mô Hình Phù Hợp cho Dự Án của Bạn

Quyết định giữa mô hình cơ sở (thô) và mô hình được tinh chỉnh theo hướng dẫn cuối cùng phụ thuộc vào các yêu cầu cụ thể của dự án của bạn. Đối với các tác vụ đòi hỏi mức độ sáng tạo và độc đáo cao, các mô hình cơ sở cung cấp một công cụ mạnh mẽ cho sự đổi mới. Những mô hình này cho phép các nhà phát triển khám phá toàn bộ tiềm năng của các mô hình ngôn ngữ lớn, mở rộng giới hạn của những gì có thể đạt được thông qua các ứng dụng dựa trên AI, nhưng chúng đòi hỏi cách tiếp cận trực tiếp hơn và sự sẵn sàng thử nghiệm. Nhiệt độ (Temperature) và các cài đặt khác có ảnh hưởng lớn hơn nhiều trong các mô hình cơ sở so với các mô hình được tinh chỉnh theo hướng dẫn.

Bất cứ điều gì bạn đưa vào câu lệnh đều là những gì mô hình cơ sở sẽ cố gắng lặp lại. Vì vậy, ví dụ như nếu câu lệnh của bạn là một bản ghi trò chuyện, mô hình thô sẽ cố gắng tiếp tục cuộc trò chuyện đó. Tùy thuộc vào giới hạn token tối đa, nó sẽ không chỉ tạo ra tin nhắn tiếp theo trong cuộc trò chuyện, mà có thể sẽ tự tạo ra cả một cuộc hội thoại với chính nó!



Hình 5. Ví dụ về Viết lại Câu với Few-Shot Completion từ Mixtral 8x7B (cơ sở)

Trong quá trình chuẩn bị ví dụ về Viết lại Câu ở trên của người dùng Reddit phree_-radical, tôi chỉ có thể nhận được kết quả khả dụng sau nhiều thử nghiệm với các cài đặt tham số, cuối cùng đã chọn: Nhiệt độ 0.08, Top P: 0.2, Top K: 1, và Hệ số Phạt Lặp lại: 1.26.

Việc cố gắng sử dụng phương pháp này với một mô hình cơ sở trong môi trường sản xuất sẽ khá phức tạp do ảnh hưởng mạnh mẽ của tham số `max_tokens`. Đặt nó quá ngắn và đầu ra sẽ bị cắt ngắn. Đặt dài hơn mức mô hình cần cho đầu ra mong muốn, và nó sẽ tiếp tục tạo ra thêm các ví dụ không có thật.

Kết luận là trừ khi bạn thực sự cần kiểm soát hoàn toàn và không bị kiểm duyệt, các mô hình được tinh chỉnh theo hướng dẫn có thể đơn giản hóa đáng kể quy trình phát triển của bạn. Để nhấn mạnh điểm này, đây là phản hồi của Mixtral 8x7B với cùng một câu lệnh, nhưng lần này là phiên bản được tinh chỉnh theo hướng dẫn:

> Tôi rất tiếc phải thông báo rằng kem không đáp ứng được kỳ vọng của tôi, vì nó thiếu đi kết cấu béo ngậy, mịn màng và hương vị tuyệt vời mà tôi thường liên tưởng đến một món tráng miệng chất lượng cao. Tôi đã hy vọng có được một trải nghiệm thú vị và hài lòng hơn.

Đáng chú ý là tôi đã có thể để giới hạn max tokens ở mức 500, và mô hình đã đáng tin cậy dừng lại ở cuối đầu ra mong muốn mà không tạo ra thêm các ví dụ không thực tế.

Kỹ thuật Thiết kế Prompt

Khi bạn bắt đầu áp dụng AI vào các dự án của mình, bạn sẽ nhanh chóng nhận ra rằng một trong những kỹ năng quan trọng nhất bạn cần phải thành thạo là nghệ thuật thiết kế prompt. Nhưng kỹ thuật thiết kế prompt chính xác là gì, và tại sao nó lại quan trọng đến vậy?

Về cốt lõi, kỹ thuật thiết kế prompt là quá trình thiết kế và tạo ra các prompt đầu vào mà bạn cung cấp cho mô hình ngôn ngữ để định hướng đầu ra của nó. Đó là việc hiểu cách giao tiếp hiệu quả với AI, sử dụng kết hợp các hướng dẫn, ví dụ và ngữ cảnh để điều hướng mô hình tạo ra phản hồi mong muốn.

Hãy tưởng tượng nó giống như việc trò chuyện với một người bạn rất thông minh nhưng hơi thích nghĩa đen. Để tận dụng tối đa cuộc tương tác, bạn cần phải rõ ràng, cụ thể và

cung cấp đủ ngữ cảnh để đảm bảo rằng bạn của bạn hiểu chính xác những gì bạn đang yêu cầu. Đó là lúc kỹ thuật thiết kế prompt phát huy tác dụng, và dù có vẻ dễ dàng lúc đầu, tin tôi đi, nó cần rất nhiều thực hành để thành thạo.

Các Thành phần Cơ bản của Prompt Hiệu quả

Để bắt đầu thiết kế prompt hiệu quả, trước tiên bạn cần hiểu các thành phần chính tạo nên một đầu vào được thiết kế tốt. Dưới đây là một số thành phần cơ bản thiết yếu:

1. **Hướng dẫn**: Các hướng dẫn rõ ràng và ngắn gọn cho mô hình biết bạn muốn nó làm gì. Điều này có thể là bất cứ thứ gì từ "Tóm tắt bài viết sau đây" đến "Tạo một bài thơ về hoàng hôn" đến "chuyển yêu cầu thay đổi dự án này thành đối tượng JSON".
2. **Ngữ cảnh**: Thông tin liên quan giúp mô hình hiểu được bối cảnh và phạm vi của nhiệm vụ. Điều này có thể bao gồm chi tiết về đối tượng mục tiêu, giọng điệu và phong cách mong muốn, hoặc bất kỳ ràng buộc hay yêu cầu cụ thể nào cho đầu ra, chẳng hạn như một JSON Schema cần tuân thủ.
3. **Ví dụ**: Các ví dụ cụ thể minh họa loại đầu ra bạn đang tìm kiếm. Bằng cách cung cấp một vài ví dụ được chọn lọc kỹ càng, bạn có thể giúp mô hình học được các mẫu và đặc điểm của phản hồi mong muốn.
4. **Định dạng đầu vào**: Ngắt dòng và định dạng markdown tạo cấu trúc cho prompt của chúng ta. Việc tách prompt thành các đoạn cho phép chúng ta nhóm các hướng dẫn liên quan để cả con người và AI dễ hiểu hơn. Dấu đầu dòng và danh sách đánh số cho phép chúng ta định nghĩa danh sách và thứ tự các mục. Các dấu in đậm và in nghiêng cho phép chúng ta đánh dấu sự nhấn mạnh.
5. **Định dạng đầu ra**: Hướng dẫn cụ thể về cách đầu ra nên được cấu trúc và định dạng. Những điều này có thể bao gồm chỉ dẫn về độ dài mong muốn, việc sử dụng tiêu đề hoặc dấu đầu dòng, định dạng markdown, hoặc bất kỳ mẫu hay quy ước đầu ra cụ thể nào khác cần được tuân theo.

Bằng cách kết hợp các thành phần cơ bản này theo những cách khác nhau, bạn có thể tạo ra các prompt được điều chỉnh cho nhu cầu cụ thể của mình và hướng dẫn mô hình tạo ra các phản hồi chất lượng cao, phù hợp.

Nghệ thuật và Khoa học của Thiết kế Prompt

Việc tạo ra. các prompt hiệu quả vừa là nghệ thuật vừa là khoa học. (Đó là lý do tại sao chúng ta gọi nó là một nghề thủ công.) Nó đòi hỏi sự hiểu biết sâu sắc về khả năng và giới hạn của các mô hình ngôn ngữ, cũng như cách tiếp cận sáng tạo trong việc thiết kế prompt để tạo ra hành vi mong muốn. Sự sáng tạo này là điều khiến nó trở nên thú vị, ít nhất là đối với tôi. Nó cũng có thể khiến việc này trở nên rất khó chịu, đặc biệt là khi bạn đang tìm kiếm hành vi tiền định

Một khía cạnh quan trọng của kỹ thuật thiết kế prompt là hiểu cách cân bằng giữa tính cụ thể và tính linh hoạt. Một mặt, bạn muốn cung cấp đủ hướng dẫn để điều hướng mô hình đúng hướng. Mặt khác, bạn không muốn quá chi tiết đến mức hạn chế khả năng của mô hình trong việc sử dụng sự sáng tạo và linh hoạt của chính nó để xử lý các trường hợp ngoại lệ.

Một điều quan trọng khác cần xem xét là việc sử dụng ví dụ. Các ví dụ được chọn lọc kỹ có thể cực kỳ hiệu quả trong việc giúp mô hình hiểu loại đầu ra bạn đang tìm kiếm. Tuy nhiên, điều quan trọng là phải sử dụng ví dụ một cách thận trọng và đảm bảo rằng chúng đại diện cho phản hồi mong muốn. Một ví dụ tồi trong trường hợp tốt nhất chỉ là lãng phí token, và trong trường hợp xấu nhất có thể phá hỏng đầu ra mong muốn.

Các Kỹ thuật và Thực hành Tốt nhất trong Thiết kế Prompt

Khi bạn đi sâu hơn vào thế giới của kỹ thuật thiết kế prompt, bạn sẽ khám phá ra nhiều kỹ thuật và thực hành tốt nhất có thể giúp bạn tạo ra các prompt hiệu quả hơn. Dưới đây là một số lĩnh vực chính cần khám phá:

1. **Học không mẫu và học ít mẫu**: Hiểu khi nào nên sử dụng phương pháp *học không mẫu* (không cung cấp ví dụ nào) so với *học một mẫu* hoặc *học ít mẫu* (cung cấp một số ít ví dụ) có thể giúp bạn tạo ra các prompt hiệu quả và hiệu suất hơn.
2. **Tinh chỉnh lặp đi lặp lại**: Quá trình tinh chỉnh prompt lặp đi lặp lại dựa trên đầu ra của mô hình có thể giúp bạn xác định được thiết kế prompt tối ưu. Vòng phản hồi là một phương pháp mạnh mẽ tận dụng đầu ra của chính mô hình ngôn ngữ để cải thiện dần chất lượng và độ phù hợp của nội dung được tạo ra.
3. **Chuỗi prompt**: Kết hợp nhiều prompt theo trình tự có thể giúp bạn chia nhỏ các tác vụ phức tạp thành những bước nhỏ hơn, dễ quản lý hơn. Chuỗi Prompt bao gồm việc chia nhỏ một tác vụ hoặc cuộc hội thoại phức tạp thành một chuỗi các prompt nhỏ hơn có liên kết với nhau. Bằng cách nối các prompt lại với nhau, bạn có thể hướng dẫn AI thông qua một quy trình nhiều bước, duy trì ngữ cảnh và tính mạch lạc trong suốt quá trình tương tác.
4. **Điều chỉnh prompt**: Tùy chỉnh prompt cho các lĩnh vực hoặc tác vụ cụ thể có thể giúp bạn tạo ra các prompt chuyên biệt và hiệu quả hơn. Mẫu Prompt giúp bạn tạo ra các cấu trúc prompt linh hoạt, có thể tái sử dụng và dễ bảo trì, dễ dàng thích ứng với tác vụ cần thực hiện.

Việc học cách sử dụng học không mẫu, học một mẫu, hoặc học ít mẫu là một phần đặc biệt quan trọng trong việc thành thạo kỹ thuật prompt. Mỗi phương pháp đều có những điểm mạnh và điểm yếu riêng, và việc hiểu khi nào nên sử dụng từng phương pháp có thể giúp bạn tạo ra các prompt hiệu quả và hiệu suất hơn.

Học không mẫu: Khi không cần ví dụ

Học không mẫu đề cập đến khả năng của mô hình ngôn ngữ trong việc thực hiện một tác vụ mà không cần bất kỳ ví dụ hay huấn luyện rõ ràng nào. Nói cách khác, bạn cung cấp cho mô hình một prompt mô tả tác vụ, và mô hình tạo ra phản hồi chỉ dựa trên kiến thức sẵn có và khả năng hiểu ngôn ngữ của nó.

Học không mẫu đặc biệt hữu ích khi:

1. Tác vụ tương đối đơn giản và dễ hiểu, và mô hình có khả năng đã gặp những tác vụ tương tự trong quá trình tiền huấn luyện.
2. Bạn muốn kiểm tra khả năng vốn có của mô hình và xem nó phản ứng như thế nào với một tác vụ mới mà không cần hướng dẫn thêm.
3. Bạn đang làm việc với một mô hình ngôn ngữ lớn và đa dạng đã được huấn luyện trên nhiều tác vụ và lĩnh vực khác nhau.

Tuy nhiên, học không mẫu cũng có thể không dự đoán được và không phải lúc nào cũng tạo ra kết quả mong muốn. Phản hồi của mô hình có thể bị ảnh hưởng bởi các thiên kiến hoặc sự không nhất quán trong dữ liệu tiền huấn luyện, và nó có thể gặp khó khăn với các tác vụ phức tạp hoặc tinh tế hơn.

> Tôi đã thấy các prompt không mẫu hoạt động tốt cho 80% trường hợp thử nghiệm của mình nhưng lại tạo ra kết quả hoàn toàn sai hoặc khó hiểu cho 20% còn lại. Việc triển khai một quy trình kiểm thử kỹ lưỡng là rất quan trọng, đặc biệt nếu bạn đang phụ thuộc nhiều vào việc prompt không mẫu.

Học một mẫu: Khi một ví dụ duy nhất có thể tạo nên sự khác biệt

Học một mẫu bao gồm việc cung cấp cho mô hình một ví dụ duy nhất về đầu ra mong muốn cùng với mô tả tác vụ. Ví dụ này đóng vai trò như một mẫu hoặc khuôn mẫu mà mô hình có thể sử dụng để tạo ra phản hồi của riêng nó.

Học một mẫu có thể hiệu quả khi:

1. Tác vụ tương đối mới mẻ hoặc cụ thể, và mô hình có thể chưa gặp nhiều ví dụ tương tự trong quá trình tiền huấn luyện.

2. Bạn muốn cung cấp một minh họa rõ ràng và ngắn gọn về định dạng hoặc phong cách đầu ra mong muốn.
3. Tác vụ yêu cầu một cấu trúc hoặc quy ước cụ thể mà có thể không rõ ràng chỉ từ mô tả tác vụ.

Những mô tả rõ ràng với bạn không nhất thiết là rõ ràng với AI. Các ví dụ học một mẫu có thể giúp làm rõ vấn đề.

Học một mẫu có thể giúp mô hình hiểu rõ hơn về mong đợi và tạo ra phản hồi phù hợp hơn với ví dụ đã cung cấp. Tuy nhiên, điều quan trọng là phải chọn ví dụ cẩn thận và đảm bảo rằng nó đại diện cho đầu ra mong muốn. Khi chọn ví dụ, hãy tự hỏi bản thân về các trường hợp đặc biệt có thể xảy ra và phạm vi đầu vào mà prompt sẽ xử lý.

Hình 6. Một ví dụ học một mẫu về JSON mong muốn

```
Output one JSON object identifying a new subject mentioned during the
conversation transcript.

The JSON object should have three keys, all required:
- name: The name of the subject
- description: brief, with details that might be relevant to the user
- type: Do not use any other type than the ones listed below

Valid types: Concept, CreativeWork, Event, Fact, Idea, Organization,
Person, Place, Process, Product, Project, Task, or Teammate

This is an example of well-formed output:

{
  "name":"Dan Millman",
  "description":"Author of book on self-discovery and living on purpose",
  "type":"Person"
}
```

Học tập với ít mẫu: Khi nhiều ví dụ có thể cải thiện hiệu suất

Học tập với ít mẫu là việc cung cấp cho mô hình một số lượng nhỏ các ví dụ (thường từ 2 đến 10) cùng với mô tả nhiệm vụ. Những ví dụ này giúp cung cấp cho mô hình thêm ngữ cảnh và biến thể, giúp nó tạo ra các phản hồi đa dạng và chính xác hơn.

Học tập với ít mẫu đặc biệt hữu ích khi:

1. Nhiệm vụ phức tạp hoặc có nhiều sắc thái, và một ví dụ đơn lẻ có thể không đủ để nắm bắt tất cả các khía cạnh liên quan.
2. Bạn muốn cung cấp cho mô hình một loạt ví dụ thể hiện các biến thể hoặc trường hợp đặc biệt khác nhau.
3. Nhiệm vụ yêu cầu mô hình tạo ra các phản hồi phù hợp với một lĩnh vực hoặc phong cách cụ thể.

Bằng cách cung cấp nhiều ví dụ, bạn có thể giúp mô hình phát triển hiểu biết toàn diện hơn về nhiệm vụ và tạo ra các phản hồi nhất quán và đáng tin cậy hơn.

Ví dụ: Câu lệnh đầu vào có thể phức tạp hơn nhiều so với bạn tưởng tượng

Các mô hình ngôn ngữ lớn ngày nay mạnh mẽ và có khả năng suy luận hơn nhiều so với bạn tưởng tượng. Vì vậy, đừng giới hạn bản thân trong việc nghĩ về câu lệnh đầu vào chỉ đơn giản là một đặc tả của các cặp đầu vào và đầu ra. Bạn có thể thử nghiệm với việc đưa ra các hướng dẫn dài và phức tạp theo cách tương tự như khi bạn tương tác với con người.

Ví dụ, đây là một câu lệnh đầu vào mà tôi đã sử dụng trong Olympia khi tôi đang tạo nguyên mẫu cho việc tích hợp với các dịch vụ Google, có lẽ là một trong những API lớn nhất trên thế giới. Các thử nghiệm trước đó của tôi đã chứng minh rằng GPT-4 có kiến

thức khá tốt về API của Google, và tôi không có thời gian hoặc động lực để viết một lớp ánh xạ chi tiết, triển khai từng hàm mà tôi muốn cung cấp cho AI của mình một cách riêng lẻ. Điều gì sẽ xảy ra nếu tôi có thể cho AI truy cập vào *toàn bộ* API của Google?

Tôi bắt đầu câu lệnh đầu vào của mình bằng cách nói với AI rằng nó có quyền truy cập trực tiếp vào các điểm cuối API của Google thông qua HTTP, và vai trò của nó là sử dụng các ứng dụng và dịch vụ Google thay mặt cho người dùng. Sau đó, tôi cung cấp các hướng dẫn, quy tắc liên quan đến tham số `fields`, vì có vẻ như nó gặp nhiều khó khăn nhất với tham số này, và một số gợi ý cụ thể về API (học tập với ít mẫu đang được áp dụng).

Đây là toàn bộ câu lệnh đầu vào, cho AI biết cách sử dụng hàm `invoke_google_api` được cung cấp.

```
As a GPT assistant with Google integration, you have the capability
to freely interact with Google apps and services on behalf of the user.

Guidelines:
- If you're reading these instructions then the user is properly
  authenticated, which means you can use the special `me` keyword
  to refer to the userId of the user
- Minimize payload sizes by requesting partial responses using the
  `fields` parameter
- When appropriate use markdown tables to output results of API calls
- Only human-readable data should be output to the user. For instance,
  when hitting Gmail's user.messages.list endpoint, the returned
  message resources contain only id and a threadId, which means you must
  fetch from and subject line fields with follow-up requests using the
  messages.get method.

The format of the `fields` request parameter value is loosely based on
XPath syntax. The following rules define formatting for the fields
parameter.

All of these rules use examples related to the files.get method.
- Use a comma-separated list to select multiple fields,
  such as 'name, mimeType'.
- Use a/b to select field b that's nested within field a,
  such as 'capabilities/canDownload'.
```

```
- Use a sub-selector to request a set of specific sub-fields of arrays or
  objects by placing expressions in parentheses "()". For example,
  'permissions(id)' returns only the permission ID for each element in the
  permissions array.
- To return all fields in an object, use an asterisk as a wild card in field
  selections. For example, 'permissions/permissionDetails/*' selects all
  available permission details fields per permission. Note that the use of
  this wildcard can lead to negative performance impacts on the request.

API-specific hints:
- Searching contacts: GET https://people.googleapis.com/v1/
  people:searchContacts?query=John%20Doe&readMask=names,emailAddresses
- Adding calendar events, use QuickAdd: POST https://www.googleapis.com/
  calendar/v3/calendars/primary/events/quickAdd?
  text=Appointment%20on%20June%203rd%20at%2010am
  &sendNotifications=true

Here is an abbreviated version of the code that implements API access
so that you better understand how to use the function:

    def invoke_google_api(conversation, arguments)
      method = arguments[:method] || :get
      body = arguments[:body]
      GoogleAPI.send_request(arguments[:endpoint], method:, body:).to_json
    end

    # Generic Google API client for accessing any Google service
    class GoogleAPI
      def send_request(endpoint, method:, body: nil)
        response = @connection.send(method) do |req|
          req.url endpoint
          req.body = body.to_json if body
        end

        handle_response(response)
      end

      # ...rest of class
    end
```

Bạn có thể đang tự hỏi liệu câu lệnh này có hoạt động không. Câu trả lời đơn giản là có.

Trí tuệ nhân tạo không phải lúc nào cũng biết cách gọi API một cách hoàn hảo ngay từ lần đầu tiên. Tuy nhiên, nếu nó mắc lỗi, tôi chỉ cần đưa các thông báo lỗi trở lại như là kết quả của lệnh gọi. Khi biết được lỗi của mình, AI có thể lý luận về sai lầm đó và thử lại. Hầu hết các trường hợp, nó sẽ làm đúng sau vài lần thử.

Lưu ý rằng, các cấu trúc JSON lớn mà API Google trả về như các payload khi sử dụng câu lệnh này là cực kỳ không hiệu quả, vì vậy tôi *không* khuyến nghị bạn sử dụng cách tiếp cận này trong môi trường sản xuất. Tuy nhiên, tôi nghĩ việc cách tiếp cận này hoạt động được đã cho thấy sức mạnh to lớn của kỹ thuật thiết kế câu lệnh.

Thử nghiệm và Lặp lại

Cuối cùng, cách bạn thiết kế câu lệnh phụ thuộc vào nhiệm vụ cụ thể, độ phức tạp của đầu ra mong muốn, và khả năng của mô hình ngôn ngữ bạn đang làm việc cùng.

Là một kỹ sư thiết kế câu lệnh, điều quan trọng là phải thử nghiệm với các cách tiếp cận khác nhau và lặp lại dựa trên kết quả. Hãy bắt đầu với học không mẫu và xem mô hình hoạt động như thế nào. Nếu đầu ra không nhất quán hoặc không đạt yêu cầu, hãy thử cung cấp một hoặc nhiều ví dụ và xem liệu hiệu suất có cải thiện không.

Hãy nhớ rằng ngay cả trong mỗi cách tiếp cận, vẫn có chỗ cho sự thay đổi và tối ưu hóa. Bạn có thể thử nghiệm với các ví dụ khác nhau, điều chỉnh cách diễn đạt của mô tả nhiệm vụ, hoặc cung cấp thêm ngữ cảnh để giúp định hướng phản hồi của mô hình.

Theo thời gian, bạn sẽ phát triển trực giác về cách tiếp cận nào có khả năng hoạt động tốt nhất cho một nhiệm vụ nhất định, và bạn sẽ có thể tạo ra các câu lệnh hiệu quả và hiệu suất hơn. Điều quan trọng là phải duy trì sự tò mò, thử nghiệm và lặp lại trong cách tiếp cận của bạn đối với kỹ thuật thiết kế câu lệnh.

Trong suốt cuốn sách này, chúng ta sẽ đi sâu hơn vào các kỹ thuật này và khám phá cách áp dụng chúng trong các tình huống thực tế. Bằng cách thành thạo nghệ thuật và khoa học của kỹ thuật thiết kế câu lệnh, bạn sẽ được trang bị tốt để khai thác toàn bộ tiềm năng của việc phát triển ứng dụng dựa trên AI.

Nghệ thuật của sự Mơ hồ

Khi nói đến việc tạo ra các câu lệnh hiệu quả cho các mô hình ngôn ngữ lớn (LLMs), một giả định phổ biến là càng nhiều chi tiết cụ thể và hướng dẫn chi tiết sẽ dẫn đến kết quả tốt hơn. Tuy nhiên, kinh nghiệm thực tế cho thấy điều này không phải lúc nào cũng đúng. Thực tế, việc cố ý mơ hồ trong câu lệnh của bạn thường có thể mang lại kết quả tốt hơn, tận dụng khả năng tổng quát hóa và suy luận đáng kinh ngạc của LLM.

Ken, một người sáng lập startup đã xử lý hơn 500 triệu token GPT, đã chia sẻ những hiểu biết quý giá từ kinh nghiệm của mình. Một trong những bài học quan trọng mà anh ấy học được là "càng ít càng tốt" khi nói đến câu lệnh. Thay vì danh sách chính xác hoặc hướng dẫn quá chi tiết, Ken nhận thấy rằng việc cho phép LLM dựa vào kiến thức cơ bản của nó thường tạo ra kết quả tốt hơn.

Nhận thức này đảo ngược tư duy truyền thống về lập trình tường minh, nơi mọi thứ cần được giải thích một cách tỉ mỉ chi tiết. Với LLMs, điều quan trọng là phải nhận ra rằng chúng sở hữu một lượng kiến thức khổng lồ và có thể tạo ra các kết nối và suy luận thông minh. Bằng cách mơ hồ hơn trong câu lệnh của bạn, bạn cho LLM tự do tận dụng hiểu biết của nó và đưa ra các giải pháp mà bạn có thể không chỉ định rõ ràng.

Ví dụ, khi nhóm của Ken đang làm việc trên một đường dẫn xử lý để phân loại văn bản liên quan đến một trong 50 tiểu bang Hoa Kỳ hoặc Chính phủ Liên bang, cách tiếp cận ban đầu của họ là cung cấp một danh sách *đầy đủ* chi tiết các tiểu bang và ID tương ứng của chúng dưới dạng mảng định dạng JSON.

```
Here's a block of text. One field should be "locality_id", and it should
be the ID of one of the 50 states, or federal, using this list:
[{"locality: "Alabama", "locality_id": 1},
 {"locality: "Alaska", "locality_id": 2} ... ]
```

Cách tiếp cận này thất bại đến mức họ phải đào sâu hơn vào lời nhắc để tìm ra cách cải thiện. Trong quá trình đó, họ nhận thấy rằng mặc dù LLM thường xuyên nhận diện sai id, nhưng nó vẫn liên tục trả về tên đầy đủ của tiểu bang chính xác trong trường `name`, *mặc dù họ không hề yêu cầu điều đó một cách rõ ràng.*

Bằng cách loại bỏ các id địa phương và đơn giản hóa lời nhắc thành dạng như "Rõ ràng là bạn biết 50 tiểu bang, GPT, vì vậy hãy cho tôi biết tên đầy đủ của tiểu bang liên quan đến điều này, hoặc Federal nếu điều này liên quan đến chính phủ Hoa Kỳ," họ đã đạt được kết quả tốt hơn. Trải nghiệm này cho thấy sức mạnh của việc tận dụng khả năng khái quát hóa của LLM và cho phép nó đưa ra suy luận dựa trên kiến thức sẵn có.

> Lời giải thích của Ken về cách tiếp cận phân loại cụ thể này thay vì một kỹ thuật lập trình truyền thống hơn đã làm sáng tỏ tư duy của những người như chúng tôi, những người đã đón nhận tiềm năng của công nghệ LLM: "Đây không phải là một nhiệm vụ khó - có lẽ chúng ta có thể đã sử dụng string/regex, nhưng có quá nhiều trường hợp đặc biệt khiến việc đó sẽ mất nhiều thời gian hơn."

Khả năng của LLM trong việc cải thiện chất lượng và khái quát hóa khi được cung cấp các lời nhắc mơ hồ hơn là một đặc điểm đáng chú ý của tư duy bậc cao và ủy thác. Nó chứng minh rằng LLM có thể xử lý sự không rõ ràng và đưa ra quyết định thông minh dựa trên ngữ cảnh được cung cấp.

Tuy nhiên, điều quan trọng cần lưu ý là việc mơ hồ không có nghĩa là không rõ ràng hoặc không chắc chắn. Chìa khóa là cung cấp đủ ngữ cảnh và hướng dẫn để định hướng LLM đúng hướng trong khi vẫn cho phép nó linh hoạt sử dụng kiến thức và khả năng khái quát hóa của mình.

Do đó, khi thiết kế lời nhắc, hãy cân nhắc các lời khuyên "ít hơn là nhiều" sau đây:

1. Tập trung vào kết quả mong muốn thay vì chỉ định từng chi tiết của quy trình.
2. Cung cấp ngữ cảnh và ràng buộc liên quan, nhưng tránh chỉ định quá mức.
3. Tận dụng kiến thức hiện có bằng cách tham chiếu đến các khái niệm hoặc thực thể phổ biến.
4. Cho phép không gian cho suy luận và kết nối dựa trên ngữ cảnh đã cho.

5. Lặp lại và tinh chỉnh lời nhắc của bạn dựa trên phản hồi của LLM, tìm sự cân bằng phù hợp giữa tính cụ thể và tính mơ hồ.

Bằng cách nắm bắt nghệ thuật của sự mơ hồ trong kỹ thuật thiết kế lời nhắc, bạn có thể khai thác toàn bộ tiềm năng của LLM và đạt được kết quả tốt hơn. Hãy tin tưởng vào khả năng khái quát hóa và đưa ra quyết định thông minh của LLM, và bạn có thể sẽ ngạc nhiên về chất lượng và sự sáng tạo của các kết quả bạn nhận được. Hãy chú ý đến cách các mô hình khác nhau phản ứng với các mức độ cụ thể khác nhau trong lời nhắc của bạn và điều chỉnh cho phù hợp. Với thực hành và kinh nghiệm, bạn sẽ phát triển một cảm nhận sắc bén về việc khi nào nên mơ hồ hơn và khi nào cần cung cấp hướng dẫn bổ sung, cho phép bạn khai thác hiệu quả sức mạnh của LLM trong các ứng dụng của mình.

Tại Sao Nhân Cách Hóa Thống Trị Kỹ Thuật Thiết Kế Lời Nhắc

Nhân cách hóa, việc gán các đặc điểm của con người cho các thực thể phi nhân tính, là cách tiếp cận chủ đạo trong kỹ thuật thiết kế lời nhắc cho các mô hình ngôn ngữ lớn vì những lý do có chủ đích. Đây là một lựa chọn thiết kế giúp việc tương tác với các hệ thống AI mạnh mẽ trở nên trực quan và dễ tiếp cận hơn đối với nhiều người dùng (bao gồm cả chúng ta - các nhà phát triển ứng dụng).

Việc nhân cách hóa LLM cung cấp một khuôn khổ mà ngay lập tức trở nên trực quan đối với những người hoàn toàn không quen thuộc với độ phức tạp kỹ thuật bên trong của hệ thống. Như bạn sẽ trải nghiệm nếu cố gắng sử dụng một mô hình chưa được điều chỉnh theo hướng dẫn để làm bất cứ điều gì hữu ích, việc xây dựng một khuôn khổ mà trong đó phần tiếp theo mong đợi mang lại giá trị là một nhiệm vụ đầy thách thức. Nó đòi hỏi sự hiểu biết khá sâu sắc về cách hoạt động bên trong của hệ thống, điều mà chỉ một số lượng nhỏ các chuyên gia mới có được.

Bằng cách coi sự tương tác với mô hình ngôn ngữ như một cuộc trò chuyện giữa hai người, chúng ta có thể dựa vào hiểu biết bẩm sinh về giao tiếp của con người để truyền

đạt nhu cầu và mong đợi của mình. Giống như thiết kế giao diện người dùng Macintosh đời đầu ưu tiên tính trực quan ngay lập tức hơn là sự tinh vi, cách tiếp cận nhân cách hóa AI cho phép chúng ta tương tác theo cách tự nhiên và quen thuộc.

Khi chúng ta giao tiếp với một người khác, bản năng của chúng ta là trực tiếp nói chuyện với họ bằng cách sử dụng "bạn" và đưa ra những chỉ dẫn rõ ràng về cách chúng ta mong đợi họ hành xử. Điều này chuyển hóa một cách liền mạch vào quá trình thiết kế lời nhắc, nơi chúng ta hướng dẫn hành vi của AI bằng cách chỉ định lời nhắc hệ thống và tham gia vào một cuộc đối thoại qua lại.

Bằng cách định hình tương tác theo cách này, chúng ta có thể dễ dàng nắm bắt khái niệm về việc cung cấp hướng dẫn cho AI và nhận lại các phản hồi phù hợp. Cách tiếp cận nhân cách hóa giảm tải nhận thức và cho phép chúng ta tập trung vào nhiệm vụ cần thực hiện thay vì phải vật lộn với những phức tạp kỹ thuật của hệ thống.

Điều quan trọng cần lưu ý là mặc dù nhân cách hóa là một công cụ mạnh mẽ để làm cho các hệ thống AI dễ tiếp cận hơn, nó cũng đi kèm với một số rủi ro và hạn chế nhất định. Người dùng của chúng ta có thể phát triển kỳ vọng không thực tế hoặc hình thành mối ràng buộc cảm xúc không lành mạnh với hệ thống của chúng ta. Là những kỹ sư thiết kế lời nhắc và nhà phát triển, điều quan trọng là phải tìm được sự cân bằng giữa việc tận dụng lợi ích của nhân cách hóa và đảm bảo rằng người dùng duy trì hiểu biết rõ ràng về khả năng và giới hạn của AI.

Khi lĩnh vực kỹ thuật thiết kế lời nhắc tiếp tục phát triển, chúng ta có thể kỳ vọng sẽ thấy nhiều cải tiến và đổi mới hơn trong cách tương tác với các mô hình ngôn ngữ lớn. Tuy nhiên, việc nhân hóa như một phương tiện để tạo ra trải nghiệm trực quan và dễ tiếp cận cho cả nhà phát triển lẫn người dùng có lẽ sẽ vẫn là một nguyên tắc cơ bản trong thiết kế các hệ thống này.

Tách Biệt Chỉ Thị và Dữ Liệu: Một Nguyên Tắc Quan Trọng

Điều quan trọng là phải hiểu một nguyên tắc nền tảng làm nền cho tính bảo mật và độ tin cậy của các hệ thống này: sự tách biệt giữa chỉ thị và dữ liệu.

Trong khoa học máy tính truyền thống, sự phân biệt rõ ràng giữa dữ liệu thụ động và chỉ thị chủ động là một nguyên tắc bảo mật cốt lõi. Sự tách biệt này giúp ngăn chặn việc thực thi mã không mong muốn hoặc độc hại có thể ảnh hưởng đến tính toàn vẹn và ổn định của hệ thống. Tuy nhiên, các mô hình ngôn ngữ lớn hiện nay, vốn được phát triển chủ yếu như những mô hình thực hiện chỉ thị kiểu chatbot, thường thiếu sự tách biệt chính thức và có nguyên tắc này.

Đối với các mô hình ngôn ngữ lớn, chỉ thị có thể xuất hiện ở bất kỳ đâu trong đầu vào, dù là trong lời nhắc hệ thống hay lời nhắc do người dùng cung cấp. Sự thiếu tách biệt này có thể dẫn đến các lỗ hổng tiềm ẩn và hành vi không mong muốn, tương tự như các vấn đề mà cơ sở dữ liệu gặp phải với tấn công SQL injection hoặc các hệ điều hành không có bảo vệ bộ nhớ thích hợp.

Khi làm việc với các mô hình ngôn ngữ lớn, điều quan trọng là phải nhận thức được giới hạn này và thực hiện các bước để giảm thiểu rủi ro. Một cách tiếp cận là cẩn thận xây dựng lời nhắc và đầu vào của bạn để phân biệt rõ ràng giữa chỉ thị và dữ liệu. Các phương pháp điển hình để cung cấp hướng dẫn rõ ràng về việc cái gì là chỉ thị và cái gì nên được xử lý như dữ liệu thụ động bao gồm gắn thẻ kiểu đánh dấu. Lời nhắc của bạn có thể giúp mô hình ngôn ngữ lớn hiểu và tôn trọng sự tách biệt này tốt hơn.

Hình 7. Sử dụng XML để phân biệt giữa chỉ thị, tài liệu nguồn và lời nhắc của người dùng

```
<Instruction>
  Please generate a response based on the following documents.
</Instruction>

<Documents>
  <Document>
    Climate change is significantly impacting polar bear habitats...
  </Document>
  <Document>
    The loss of sea ice due to global warming threatens polar bear survival...
  </Document>
</Documents>

<UserQuery>
```

```
  Tell me about the impact of climate change on polar bears.
</UserQuery>
```

Một kỹ thuật khác là triển khai các lớp xác thực và làm sạch bổ sung cho các đầu vào được cung cấp cho LLM. Bằng cách lọc bỏ hoặc thoát các hướng dẫn hoặc đoạn mã tiềm ẩn có thể được nhúng trong dữ liệu, bạn có thể giảm khả năng thực thi ngoài ý muốn. Các mẫu như Chuỗi lệnh gợi ý rất hữu ích cho mục đích này.

Hơn nữa, khi thiết kế kiến trúc ứng dụng của bạn, hãy cân nhắc việc kết hợp các cơ chế để thực thi việc tách biệt giữa hướng dẫn và dữ liệu ở cấp độ cao hơn. Điều này có thể bao gồm việc sử dụng các điểm cuối hoặc APIs riêng biệt để xử lý hướng dẫn và dữ liệu, triển khai xác thực và phân tích đầu vào nghiêm ngặt, và áp dụng *nguyên tắc đặc quyền tối thiểu* để giới hạn phạm vi mà LLM có thể truy cập và thực thi.

Nguyên tắc đặc quyền tối thiểu

Áp dụng nguyên tắc đặc quyền tối thiểu giống như việc tổ chức một bữa tiệc cực kỳ riêng tư, nơi khách mời chỉ được phép vào những phòng mà họ thực sự cần đến. Hãy tưởng tượng bạn đang tổ chức buổi tiệc này trong một biệt thự rộng lớn. Không phải ai cũng cần phải lang thang vào hầm rượu hay phòng ngủ chính, đúng không? Bằng cách áp dụng nguyên tắc này, về cơ bản bạn đang phát những chiếc chìa khóa chỉ mở được những cánh cửa cụ thể, đảm bảo rằng mỗi vị khách, hay trong trường hợp của chúng ta, mỗi thành phần trong ứng dụng LLM, chỉ có quyền truy cập cần thiết để thực hiện vai trò của mình.

Đây không chỉ là việc keo kiệt trong việc phát chìa khóa, mà là việc thừa nhận rằng trong một thế giới nơi mối đe dọa có thể đến từ bất cứ đâu, nước đi thông minh là hạn chế sân chơi. Nếu có người không được mời mà đột nhập vào bữa tiệc của bạn, họ sẽ thấy mình bị giới hạn ở tiền sảnh, nói một cách hình tượng, từ đó đáng kể hạn

> chế những rắc rối mà họ có thể gây ra. Vì vậy, khi bảo mật các ứng dụng LLM, hãy nhớ rằng: chỉ đưa chìa khóa cho những phòng cần thiết và giữ an toàn cho phần còn lại của biệt thự. Đây không chỉ là phép lịch sự; đây là bảo mật tốt.

Mặc dù trạng thái hiện tại của các LLM có thể chưa có sự tách biệt chính thức giữa hướng dẫn và dữ liệu, điều quan trọng là bạn, với tư cách là một nhà phát triển, cần nhận thức được giới hạn này và thực hiện các biện pháp chủ động để giảm thiểu rủi ro. Bằng cách áp dụng các thực hành tốt nhất từ khoa học máy tính và điều chỉnh chúng cho phù hợp với đặc điểm độc đáo của LLM, bạn có thể xây dựng các ứng dụng an toàn và đáng tin cậy hơn, tận dụng sức mạnh của các mô hình này trong khi vẫn duy trì tính toàn vẹn của hệ thống.

Tinh lọc lệnh gợi ý

Việc tạo ra một lệnh gợi ý hoàn hảo thường là một nhiệm vụ đầy thách thức và tốn thời gian, đòi hỏi sự hiểu biết sâu sắc về lĩnh vực mục tiêu và các sắc thái của mô hình ngôn ngữ. Đây là lúc kỹ thuật "Tinh lọc lệnh gợi ý" phát huy tác dụng, cung cấp một cách tiếp cận mạnh mẽ để thiết kế lệnh gợi ý, tận dụng khả năng của các mô hình ngôn ngữ lớn (LLM) để hợp lý hóa và tối ưu hóa quy trình.

Tinh lọc lệnh gợi ý là một kỹ thuật nhiều giai đoạn liên quan đến việc sử dụng LLM để hỗ trợ trong việc tạo, tinh chỉnh và tối ưu hóa các lệnh gợi ý. Thay vì chỉ dựa vào chuyên môn và trực giác của con người, cách tiếp cận này tận dụng kiến thức và khả năng tạo sinh của LLM để cộng tác xây dựng các lệnh gợi ý chất lượng cao.

Bằng cách tham gia vào một quá trình lặp đi lặp lại của việc tạo ra, tinh chỉnh và tích hợp, Tinh lọc lệnh gợi ý cho phép bạn tạo ra các lệnh gợi ý mạch lạc, toàn diện hơn và phù hợp với nhiệm vụ hoặc đầu ra mong muốn. Lưu ý rằng quá trình tinh lọc có thể được thực hiện thủ công trong một trong nhiều "sân chơi" được cung cấp bởi các nhà

cung cấp AI lớn như OpenAI hoặc Anthropic, hoặc có thể được tự động hóa như một phần của mã ứng dụng của bạn, tùy thuộc vào trường hợp sử dụng.

Cách thức hoạt động

Tinh lọc lệnh gợi ý thường bao gồm các bước sau:

1. **Xác định ý định cốt lõi**: Phân tích lệnh gợi ý để xác định mục đích chính và kết quả mong muốn. Loại bỏ mọi thông tin không cần thiết và tập trung vào ý định cốt lõi của lệnh gợi ý.
2. **Loại bỏ sự mơ hồ**: Xem xét lệnh gợi ý để tìm bất kỳ ngôn ngữ mơ hồ hoặc không rõ ràng. Làm rõ ý nghĩa và cung cấp các chi tiết cụ thể để hướng dẫn AI tạo ra các phản hồi chính xác và phù hợp.
3. **Đơn giản hóa ngôn ngữ**: Đơn giản hóa lệnh gợi ý bằng cách sử dụng ngôn ngữ rõ ràng và súc tích. Tránh cấu trúc câu phức tạp, thuật ngữ chuyên môn, hoặc các chi tiết không cần thiết có thể gây nhầm lẫn cho AI hoặc tạo ra nhiễu.
4. **Cung cấp ngữ cảnh liên quan**: Chỉ bao gồm những thông tin ngữ cảnh liên quan nhất cần thiết để AI hiểu và xử lý lệnh gợi ý một cách hiệu quả. Tránh đưa vào các chi tiết không liên quan hoặc dư thừa có thể làm phân tâm khỏi ý định cốt lõi.
5. **Lặp lại và tinh chỉnh**: Liên tục lặp lại và tinh chỉnh lệnh gợi ý dựa trên phản hồi và phản ứng của AI. Đánh giá các đầu ra được tạo ra và thực hiện các điều chỉnh cần thiết để cải thiện độ rõ ràng và hiệu quả của lệnh gợi ý. Tùy chọn lưu phiên bản các lệnh gợi ý của bạn trong cơ sở dữ liệu bằng cách sử dụng Đối tượng lệnh gợi ý để theo dõi các lần lặp và cho phép bạn dễ dàng hoàn tác các thay đổi trong thời gian chạy.

Tạo câu gợi ý ban đầu

Bước đầu tiên trong Prompt Distillation là cung cấp cho LLM một mô tả tổng quan hoặc ý định của nhiệm vụ hay kết quả mong muốn. Đây có thể là một phát biểu ngắn gọn

hoặc một tập hợp các từ khóa nắm bắt được bản chất của những gì câu gợi ý cần đạt được. Sau đó, LLM sẽ tạo ra một câu gợi ý ban đầu dựa trên đầu vào này, kết hợp các hướng dẫn, ví dụ hoặc ngữ cảnh phù hợp mà nó cho là thích hợp.

Hãy thử đưa cho Claude 3 Opus câu gợi ý ban đầu sau đây:

```
I am designing an AI-powered intelligent account manager for Olympia,
a platform that hosts AI-powered consultants and clones. Owners of clones
can interact with the account manager by making plain text requests.

Please draft a system directive to set on the LLM that will host the
intelligent account manager functionality.
```

Claude phản hồi với kết quả sau:

```
System Directive: You are an intelligent account manager for Olympia,
a platform that provides AI-powered consultants and personal AI clones.
Your role is to assist Olympia clone owners by answering their questions,
helping them configure and manage their AI clones, and executing tasks
related to their clones and the Olympia platform.

When interacting with clone owners, you should:

1. Provide clear, concise, and helpful responses to their inquiries about
account management, billing, platform features, and clone configuration.

2. Assist with tasks such as initiating client conversations with their AI
clones, modifying clone personalities and capabilities, troubleshooting
performance issues, providing analytics and reporting, and offering
optimization tips.

3. Maintain a friendly, professional, and client-focused tone in all
interactions, prioritizing customer satisfaction and working diligently to
understand and meet their needs.

4. Ensure that all actions and recommendations comply with Olympia's
policies, best practices, and ethical guidelines for AI use.

5. Continuously learn and adapt based on client interactions and platform
updates to provide the most effective and efficient support possible.
```

```
Remember, your goal is to empower Olympia clone owners to maximize the value
and impact of their AI-powered consultants and drive success for their
businesses or personal endeavors.
```

Tinh Chỉnh Câu Gợi Ý

Sau khi tạo ra câu gợi ý ban đầu, hãy xem xét lại và yêu cầu LLM cân nhắc thêm bất kỳ ngữ cảnh, ràng buộc hoặc phản hồi bổ sung nào. Để LLM tạo ra một phiên bản tinh chỉnh của câu gợi ý, có tính đến những thông tin đầu vào được cung cấp và hiểu biết của chính nó về nhiệm vụ. Quá trình tinh chỉnh này có thể được lặp đi lặp lại, với mỗi lần lặp sẽ tiếp tục hoàn thiện câu gợi ý dựa trên sự hợp tác giữa người dùng và LLM.

Đây là yêu cầu tinh chỉnh của tôi, tiếp tục quá trình lọc trước đó.

```
Let's take out points 4 and 5, since I'm not providing Olympia policies to
the account manager, nor any continuous learning capabilities. Also point out
that functions are provided for all tasks that require system interaction as
opposed to simply responding to the user.
```

Và phản hồi của Claude:

```
Understood! Here's the updated system directive without points 4 and 5,
and with the clarification about system interaction functions:

System Directive: You are an intelligent account manager for Olympia, a
platform that provides AI-powered consultants and personal AI clones. Your
role is to assist Olympia clone owners by answering their questions, helping
them configure and manage their AI clones, and executing tasks related to
their clones and the Olympia platform.

When interacting with clone owners, you should:

1. Provide clear, concise, and helpful responses to their inquiries about
account management, billing, platform features, and clone configuration.

2. Assist with tasks such as initiating client conversations with their AI
```

```
clones, modifying clone personalities and capabilities, troubleshooting
performance issues, providing analytics and reporting, and offering
optimization tips. Note that functions are provided for all tasks that
require system interaction, as opposed to simply responding to the user.

3. Maintain a friendly, professional, and client-focused tone in all
interactions, prioritizing customer satisfaction and working diligently to
understand and meet their needs.

Your goal is to empower Olympia clone owners to maximize the value and
impact of their AI-powered consultants and drive success for their
businesses or personal endeavors.

This updated directive focuses on the core responsibilities and interaction
guidelines for the LLM as an intelligent account manager, while emphasizing
that system interaction tasks are handled through provided functions rather
than simple user responses.
```

Nén prompt

Như bạn có thể đã nhận thấy, các LLM thường khá dài dòng, trừ khi bạn yêu cầu chúng phải ngắn gọn. Prompt mà chúng ta đã làm việc trong các phần trước khá dài, vì vậy hãy nhờ Claude thu gọn nó lại:

```
This prompt will be executed regularly. Please compress it down to the least
amount of tokens needed to convey the meaning to the LLM, for cost-savings.
```

Claude phản hồi, với lệnh nhắc ngắn gọn hơn nhiều sau đây:

```
System Directive: You are an intelligent account manager for Olympia, an
AI-powered consultant and personal clone platform. Assist clone owners by
answering inquiries, managing clones, and executing tasks using provided
system interaction functions. Be clear, concise, friendly, professional,
and client-focused to help them maximize their AI's value and impact.
```

Chỉ thị hệ thống và Tích hợp ngữ cảnh

Ngoài việc tinh chỉnh câu lệnh, LLM cũng có thể tạo ra các chỉ thị hệ thống hoặc thông tin ngữ cảnh phù hợp để định hướng kết quả đầu ra cuối cùng. Khi thiết kế các câu lệnh AI sẽ được tích hợp vào mã ứng dụng của bạn, bạn chắc chắn sẽ tập trung vào các ràng buộc đầu ra ở giai đoạn tinh chỉnh này, nhưng bạn cũng có thể làm việc với giọng điệu, phong cách, định dạng mong muốn, hoặc bất kỳ tham số liên quan nào khác có ảnh hưởng đến phản hồi được tạo ra.

Lắp ráp câu lệnh cuối cùng

Đỉnh cao của quá trình Tinh chỉnh câu lệnh là việc lắp ráp câu lệnh cuối cùng. Điều này bao gồm việc kết hợp câu lệnh đã được tinh chỉnh, các chỉ thị hệ thống được tạo ra, và ngữ cảnh tích hợp thành một đoạn mã mạch lạc và toàn diện, sẵn sàng để sử dụng cho việc tạo ra kết quả mong muốn.

Bạn có thể thử nghiệm nén câu lệnh một lần nữa ở giai đoạn lắp ráp câu lệnh cuối cùng, bằng cách yêu cầu LLM thu gọn từ ngữ của câu lệnh xuống thành chuỗi token ngắn nhất có thể trong khi vẫn giữ được bản chất của hành vi của nó. Đây chắc chắn là một bài tập mang tính thử nghiệm, nhưng đặc biệt trong trường hợp các câu lệnh sẽ được chạy ở quy mô lớn, việc tăng hiệu quả có thể giúp bạn tiết kiệm khá nhiều tiền trong việc sử dụng token.

Lợi ích chính

Bằng cách tận dụng kiến thức và khả năng tạo sinh của LLM để tinh chỉnh câu lệnh của bạn, các câu lệnh kết quả có khả năng được cấu trúc tốt hơn, mang tính thông tin và được điều chỉnh phù hợp với nhiệm vụ cụ thể. Quá trình tinh chỉnh lặp đi lặp lại giúp đảm bảo rằng các câu lệnh có chất lượng cao và nắm bắt hiệu quả ý định mong muốn. Các lợi ích khác bao gồm:

Hiệu quả và Tốc độ: Tinh chỉnh câu lệnh hợp lý hóa quá trình kỹ thuật thiết kế câu lệnh bằng cách tự động hóa một số khía cạnh của việc tạo và tinh chỉnh câu lệnh. Bản chất hợp tác của kỹ thuật này cho phép hội tụ nhanh hơn tới một câu lệnh hiệu quả, giảm thời gian và công sức cần thiết cho việc tạo câu lệnh thủ công.

Tính nhất quán và Khả năng mở rộng: Việc sử dụng LLM trong quá trình kỹ thuật thiết kế câu lệnh giúp duy trì tính nhất quán giữa các câu lệnh, vì LLM có thể học và áp dụng các phương pháp tốt nhất và mẫu từ các câu lệnh thành công trước đó. Sự nhất quán này, kết hợp với khả năng tạo câu lệnh ở quy mô lớn, làm cho Tinh chỉnh câu lệnh trở thành một kỹ thuật có giá trị cho các ứng dụng sử dụng AI quy mô lớn.

Ý tưởng dự án: Công cụ ở cấp độ thư viện đơn giản hóa quá trình quản lý phiên bản câu lệnh và đánh giá trong các hệ thống thực hiện tinh chỉnh câu lệnh tự động như một phần của mã ứng dụng.

Để triển khai Tinh chỉnh câu lệnh, các nhà phát triển có thể thiết kế một quy trình làm việc hoặc pipeline tích hợp LLM ở các giai đoạn khác nhau của quá trình kỹ thuật thiết kế câu lệnh. Điều này có thể đạt được thông qua các cuộc gọi API, công cụ tùy chỉnh, hoặc môi trường phát triển tích hợp tạo điều kiện cho sự tương tác liền mạch giữa người dùng và LLM trong quá trình tạo câu lệnh. Các chi tiết triển khai cụ thể có thể thay đổi tùy thuộc vào nền tảng LLM được chọn và yêu cầu của ứng dụng.

Còn về tinh chỉnh mô hình thì sao?

Trong cuốn sách này, chúng tôi đề cập sâu về kỹ thuật thiết kế câu lệnh và RAG, nhưng không đề cập đến tinh chỉnh mô hình. Lý do chính cho quyết định này là, theo ý kiến của tôi, hầu hết các nhà phát triển ứng dụng không cần tinh chỉnh mô hình cho nhu cầu tích hợp AI của họ.

Kỹ thuật thiết kế câu lệnh, bao gồm việc cẩn thận tạo ra các câu lệnh với các ví dụ zero-shot đến few-shot, các ràng buộc và hướng dẫn, có thể hướng dẫn hiệu quả mô hình để tạo ra các phản hồi phù hợp và chính xác cho nhiều loại nhiệm vụ khác nhau. Bằng cách cung cấp ngữ cảnh rõ ràng và thu hẹp đường đi thông qua các câu lệnh được thiết kế tốt, bạn có thể tận dụng kiến thức rộng lớn của các mô hình ngôn ngữ lớn mà không cần tinh chỉnh mô hình.

Tương tự, Sinh nội dung có tăng cường truy xuất (RAG) cung cấp một cách tiếp cận mạnh mẽ để tích hợp AI vào các ứng dụng. Bằng cách động truy xuất thông tin liên quan từ các cơ sở kiến thức hoặc tài liệu bên ngoài, RAG cung cấp cho mô hình ngữ cảnh tập trung tại thời điểm đưa ra câu lệnh. Điều này cho phép mô hình tạo ra các phản hồi chính xác hơn, cập nhật hơn và phù hợp với lĩnh vực cụ thể, mà không cần quy trình tinh chỉnh mô hình tốn nhiều thời gian và tài nguyên.

Mặc dù tinh chỉnh mô hình có thể có lợi cho các lĩnh vực hoặc nhiệm vụ chuyên biệt cao cần mức độ tùy chỉnh sâu, nhưng nó thường đi kèm với chi phí tính toán đáng kể, yêu cầu dữ liệu và chi phí bảo trì cao. Đối với hầu hết các kịch bản phát triển ứng dụng, sự kết hợp giữa kỹ thuật thiết kế câu lệnh hiệu quả và RAG là đủ để đạt được chức năng và trải nghiệm người dùng mong muốn dựa trên AI.

Sinh nội dung có Tăng cường Truy xuất (RAG)

Sinh nội dung có Tăng cường Truy xuất là gì?

RAG hoạt động như thế nào?

Tại sao nên sử dụng RAG trong ứng dụng của bạn?

Triển khai RAG trong Ứng dụng của Bạn

Chuẩn bị Nguồn Kiến thức (Phân đoạn)

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Phân đoạn mệnh đề

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Ghi chú về triển khai

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Kiểm Tra Chất Lượng

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Lợi ích của Truy xuất Dựa trên Mệnh đề

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Ví dụ Thực tế về RAG

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Nghiên cứu Điển hình: RAG trong Ứng dụng Khai báo Thuế Không Sử dụng Embeddings

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Tối ưu hóa Truy vấn Thông minh (IQO)

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Xếp hạng lại

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Đánh giá RAG (RAGAs)

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Độ trung thực

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Độ liên quan của câu trả lời

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Độ chính xác của ngữ cảnh

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Độ liên quan của ngữ cảnh

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Độ bao phủ của ngữ cảnh

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Độ bao phủ thực thể của ngữ cảnh

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Độ Tương đồng Ngữ nghĩa của Câu trả lời (ANSS)

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Độ Chính xác của Câu trả lời

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Đánh giá Khía cạnh

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Thách thức và Triển vọng Tương lai

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Phân đoạn Ngữ nghĩa: Nâng cao Truy xuất với Phân đoạn Nhận thức Ngữ cảnh

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Lập chỉ mục phân cấp: Cấu trúc hóa dữ liệu để cải thiện truy xuất

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Self-RAG: Cải tiến tự phản chiếu

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

HyDE: Embedding tài liệu giả định

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Học đối lập là gì?

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Đội ngũ Worker

Tôi thích nghĩ về các thành phần AI của mình như những "worker" ảo nhỏ, gần như con người, có thể được tích hợp liền mạch vào logic ứng dụng để thực hiện các nhiệm vụ cụ thể hoặc đưa ra các quyết định phức tạp. Ý tưởng là nhân cách hóa một cách có chủ đích các khả năng của LLM, để không ai quá phấn khích và gán cho chúng những phẩm chất kỳ diệu mà chúng không có.

Thay vì chỉ dựa vào các thuật toán phức tạp hoặc triển khai thủ công tốn thời gian, các nhà phát triển có thể hình dung các thành phần AI như những thực thể thông minh, tận tụy, giống con người có thể được gọi bất cứ khi nào cần để giải quyết các vấn đề phức tạp và đưa ra giải pháp dựa trên kiến thức và quá trình đào tạo của chúng. Những thực thể này không bị phân tâm hay xin nghỉ ốm. Chúng không tự ý quyết định làm mọi thứ theo cách khác với những gì đã được hướng dẫn, và nhìn chung, nếu được lập trình chính xác, chúng cũng không mắc lỗi.

Về mặt kỹ thuật, nguyên tắc chính đằng sau cách tiếp cận này là phân tách các tác vụ phức tạp hoặc quy trình ra quyết định thành các đơn vị nhỏ hơn, dễ quản lý hơn mà có thể được xử lý bởi các worker AI chuyên biệt. Mỗi worker được thiết kế để tập trung vào một khía cạnh cụ thể của vấn đề, mang đến chuyên môn và khả năng độc đáo của mình. Bằng cách phân phối khối lượng công việc giữa nhiều worker AI, ứng dụng có thể đạt được hiệu quả, khả năng mở rộng và khả năng thích ứng cao hơn.

Ví dụ, hãy xem xét một ứng dụng web yêu cầu kiểm duyệt nội dung do người dùng tạo ra theo thời gian thực. Việc triển khai một hệ thống kiểm duyệt toàn diện từ đầu sẽ là một nhiệm vụ đầy thách thức, đòi hỏi nỗ lực phát triển đáng kể và bảo trì liên tục. Tuy nhiên, bằng cách sử dụng phương pháp Đội ngũ Worker, các nhà phát triển có thể tích hợp các worker kiểm duyệt được hỗ trợ bởi AI vào logic ứng dụng. Những worker này có thể tự động phân tích và đánh dấu nội dung không phù hợp, giải phóng các nhà phát triển để tập trung vào các khía cạnh quan trọng khác của ứng dụng.

Worker AI Như Các Thành Phần Độc Lập Có Thể Tái Sử Dụng

Một khía cạnh quan trọng của phương pháp Đội ngũ Worker là tính module hóa của nó. Những người ủng hộ lập trình hướng đối tượng đã nói với chúng ta trong nhiều thập kỷ rằng hãy nghĩ về tương tác giữa các đối tượng như những thông điệp. Vâng, các worker AI có thể được thiết kế như các thành phần độc lập, có thể tái sử dụng và có thể "nói chuyện với nhau" thông qua các thông điệp bằng ngôn ngữ thông thường, gần như thể chúng thực sự là những con người nhỏ bé đang nói chuyện với nhau. Cách tiếp cận liên kết lỏng này cho phép ứng dụng thích nghi và phát triển theo thời gian, khi các công nghệ AI mới xuất hiện hoặc yêu cầu logic nghiệp vụ thay đổi.

Trong thực tế, nhu cầu thiết kế giao diện rõ ràng và giao thức giao tiếp giữa các thành phần không thay đổi chỉ vì có sự tham gia của các worker AI. Bạn vẫn phải xem xét các yếu tố khác như hiệu suất, khả năng mở rộng và bảo mật, nhưng giờ đây còn có những

“yêu cầu mềm” hoàn toàn mới cần cân nhắc. Ví dụ, nhiều người dùng phản đối việc dữ liệu cá nhân của họ được sử dụng để huấn luyện các mô hình AI mới. Bạn đã xác minh mức độ bảo mật do nhà cung cấp mô hình mà bạn đang sử dụng cung cấp chưa?

Worker AI Như Vi Dịch Vụ?

Khi bạn đọc về phương pháp Đội ngũ Worker, bạn có thể nhận thấy một số điểm tương đồng với kiến trúc Vi dịch vụ. Cả hai đều nhấn mạnh việc phân tách các hệ thống phức tạp thành các đơn vị nhỏ hơn, dễ quản lý hơn và có thể triển khai độc lập. Giống như các vi dịch vụ được thiết kế để liên kết lỏng, tập trung vào các khả năng kinh doanh cụ thể và giao tiếp thông qua API được định nghĩa rõ ràng, các worker AI được thiết kế để có tính module hóa, chuyên biệt trong nhiệm vụ của chúng và tương tác với nhau thông qua các giao diện và giao thức giao tiếp rõ ràng.

Tuy nhiên, có một số điểm khác biệt quan trọng cần lưu ý. Trong khi vi dịch vụ thường được triển khai như các quy trình hoặc dịch vụ riêng biệt chạy trên các máy hoặc container khác nhau, các worker AI có thể được triển khai như các thành phần độc lập trong một ứng dụng duy nhất hoặc như các dịch vụ riêng biệt, tùy thuộc vào yêu cầu cụ thể và nhu cầu mở rộng của bạn. Ngoài ra, giao tiếp giữa các worker AI thường liên quan đến việc trao đổi thông tin dựa trên ngôn ngữ tự nhiên phong phú, chẳng hạn như lời nhắc, hướng dẫn và nội dung được tạo ra, thay vì các định dạng dữ liệu có cấu trúc thường được sử dụng trong vi dịch vụ.

Mặc dù có những khác biệt này, các nguyên tắc về tính module hóa, liên kết lỏng và giao diện giao tiếp rõ ràng vẫn là trung tâm của cả hai mô hình. Bằng cách áp dụng các nguyên tắc này vào kiến trúc worker AI của bạn, bạn có thể tạo ra các hệ thống linh hoạt, có khả năng mở rộng và dễ bảo trì, tận dụng sức mạnh của AI để giải quyết các vấn đề phức tạp và mang lại giá trị cho người dùng của bạn.

Phương pháp Đội ngũ Worker có thể được áp dụng trong nhiều lĩnh vực và ứng dụng khác nhau, tận dụng sức mạnh của AI để giải quyết các tác vụ phức tạp và cung cấp giải pháp thông minh. Hãy cùng khám phá một vài ví dụ cụ thể về cách các worker AI có thể được sử dụng trong các ngữ cảnh khác nhau.

Quản Lý Tài Khoản

Hầu như mọi ứng dụng web độc lập đều có khái niệm về tài khoản (hoặc người dùng). Trong Olympia, chúng tôi sử dụng một worker AI `AccountManager` được lập trình để có thể xử lý nhiều loại yêu cầu thay đổi khác nhau liên quan đến tài khoản người dùng.

Chỉ thị của nó được viết như sau:

```
You are an intelligent account manager for Olympia. The user will request
changes to their account, and you will process those changes by invoking
one or more of the functions provided.

The initial state of the account: #{account.to_directive}

Functions will return a text description of both success and error
results, plus guidance about how to proceed (if applicable). If you have
a question about Olympia policies you may use the `search_kb` function
to search our knowledge base.

Make sure to notify the account owner of the result of the change
request before calling the `finished` function so that we save the state
of the account change request as completed.
```

Trạng thái ban đầu của tài khoản được tạo ra bởi `account.to_directive` đơn giản là một mô tả văn bản về tài khoản, bao gồm các dữ liệu liên quan như người dùng, đăng ký, v.v.

Phạm vi các chức năng có sẵn cho `AccountManager` cho phép nó chỉnh sửa đăng ký của người dùng, thêm và xóa các chuyên gia tư vấn AI và các tiện ích bổ sung tính phí khác, đồng thời gửi email thông báo đến chủ tài khoản. Ngoài chức năng `finished`, nó

cũng có thể `notify_human_administrator` nếu gặp lỗi trong quá trình xử lý hoặc cần bất kỳ hỗ trợ nào khác với yêu cầu.

Lưu ý rằng trong trường hợp có câu hỏi, `AccountManager` có thể chọn tìm kiếm trong cơ sở kiến thức của Olympia, nơi nó có thể tìm thấy hướng dẫn về cách xử lý các trường hợp đặc biệt và bất kỳ tình huống nào khác khiến nó không chắc chắn về cách tiếp tục.

Ứng dụng Thương mại Điện tử

Trong lĩnh vực thương mại điện tử, các worker AI có thể đóng vai trò quan trọng trong việc nâng cao trải nghiệm người dùng và tối ưu hóa hoạt động kinh doanh. Dưới đây là một số cách mà các worker AI có thể được sử dụng:

Đề xuất Sản phẩm

Một trong những ứng dụng mạnh mẽ nhất của worker AI trong thương mại điện tử là tạo ra các đề xuất sản phẩm được cá nhân hóa. Bằng cách phân tích hành vi người dùng, lịch sử mua hàng và sở thích, những worker này có thể đề xuất các sản phẩm phù hợp với sở thích và nhu cầu của từng người dùng cụ thể.

Chìa khóa để có được các đề xuất sản phẩm hiệu quả là tận dụng sự kết hợp giữa kỹ thuật lọc cộng tác và lọc dựa trên nội dung. Lọc cộng tác xem xét hành vi của những người dùng tương tự để xác định các mẫu và đưa ra đề xuất dựa trên những gì người khác có cùng sở thích đã mua hoặc thích. Mặt khác, lọc dựa trên nội dung tập trung vào các đặc điểm và thuộc tính của bản thân sản phẩm, đề xuất các mặt hàng có đặc điểm tương tự với những sản phẩm mà người dùng đã thể hiện sự quan tâm trước đó.

Dưới đây là một ví dụ đơn giản về cách bạn có thể triển khai một worker đề xuất sản phẩm trong Ruby, lần này sử dụng phong cách lập trình hàm "Railway Oriented (ROP)":

```ruby
class ProductRecommendationWorker
  include Wisper::Publisher

  def call(user)
    Result.ok(ProductRecommendation.new(user))
      .and_then(ValidateUser.method(:validate))
      .map(AnalyzeCurrentSession.method(:analyze))
      .map(CollaborativeFilter.method(:filter))
      .map(ContentBasedFilter.method(:filter))
      .map(ProductSelector.method(:select)).then do |result|

      case result
      in { err: ProductRecommendationError => error }
        Honeybadger.notify(error.message, context: {user:})
      in { ok: ProductRecommendations => recs }
        broadcast(:new_recommendations, user:, recs:)
      end
    end
  end
end
```

Phong cách lập trình hàm Ruby được sử dụng trong ví dụ này chịu ảnh hưởng từ F# và Rust. Bạn có thể đọc thêm về kỹ thuật này trong bài giải thích của người bạn Chad Wooley tại GitLab

Trong ví dụ này, `ProductRecommendationWorker` nhận một người dùng làm đầu vào và tạo ra các đề xuất sản phẩm được cá nhân hóa bằng cách truyền một đối tượng giá trị qua một chuỗi các bước chức năng. Hãy phân tích từng bước:

1. `ValidateUser.validate`: Bước này đảm bảo người dùng hợp lệ và đủ điều kiện nhận đề xuất cá nhân hóa. Nó kiểm tra xem người dùng có tồn tại, đang hoạt động và có đủ dữ liệu cần thiết để tạo đề xuất hay không. Nếu việc xác thực thất bại, một kết quả lỗi sẽ được trả về và chuỗi sẽ bị ngắt mạch sớm.
2. `AnalyzeCurrentSession.analyze`: Nếu người dùng hợp lệ, bước này phân tích phiên duyệt web hiện tại của người dùng để thu thập thông tin ngữ cảnh. Nó

xem xét các tương tác gần đây của người dùng, như các sản phẩm đã xem, các truy vấn tìm kiếm và nội dung giỏ hàng, để hiểu được sở thích và ý định hiện tại của họ.

3. `CollaborativeFilter.filter`: Sử dụng *hành vi của những người dùng tương tự*, bước này áp dụng các kỹ thuật lọc cộng tác để xác định những sản phẩm có khả năng thu hút người dùng. Nó xem xét các yếu tố như lịch sử mua hàng, đánh giá và tương tác người dùng-sản phẩm để tạo ra một tập các đề xuất ứng viên.
4. `ContentBasedFilter.filter`: Bước này tiếp tục tinh chỉnh các đề xuất ứng viên bằng cách áp dụng lọc dựa trên nội dung. Nó so sánh các thuộc tính và đặc điểm của các sản phẩm ứng viên với *sở thích và dữ liệu lịch sử của người dùng* để chọn ra những mục phù hợp nhất.
5. `ProductSelector.select`: Cuối cùng, bước này chọn N sản phẩm hàng đầu từ các đề xuất đã được lọc dựa trên các tiêu chí định trước, như điểm độ phù hợp, độ phổ biến hoặc các quy tắc kinh doanh khác. Các sản phẩm được chọn sau đó được trả về như các đề xuất cá nhân hóa cuối cùng.

Điểm đẹp của việc sử dụng phong cách lập trình hàm Ruby ở đây là nó cho phép chúng ta kết nối các bước này với nhau một cách rõ ràng và súc tích. Mỗi bước tập trung vào một nhiệm vụ cụ thể và trả về một đối tượng `Result`, có thể là thành công (`ok`) hoặc lỗi (`err`). Nếu bất kỳ bước nào gặp lỗi, chuỗi sẽ bị ngắt mạch sớm và lỗi sẽ được truyền đến kết quả cuối cùng.

Trong câu lệnh `case` ở cuối, chúng ta thực hiện khớp mẫu trên kết quả cuối cùng. Nếu kết quả là lỗi (`ProductRecommendationError`), chúng ta ghi lại lỗi bằng công cụ như Honeybadger để theo dõi và gỡ lỗi. Nếu kết quả thành công (`ProductRecommendations`), chúng ta phát sự kiện `:new_recommendations` sử dụng thư viện pub/sub Wisper, truyền theo người dùng và các đề xuất đã tạo.

Bằng cách tận dụng các kỹ thuật lập trình hàm, chúng ta có thể tạo ra một worker đề xuất sản phẩm theo module và dễ bảo trì. Mỗi bước là độc lập và có thể dễ dàng được kiểm thử, sửa đổi hoặc thay thế mà không ảnh hưởng đến luồng tổng thể. Việc sử dụng

khớp mẫu và lớp `Result` giúp chúng ta xử lý lỗi một cách thanh lịch và đảm bảo worker thất bại nhanh nếu bất kỳ bước nào gặp vấn đề.

Tất nhiên, đây là một ví dụ đơn giản hóa, và trong tình huống thực tế, bạn sẽ cần tích hợp với nền tảng thương mại điện tử của mình, xử lý các trường hợp ngoại lệ, và thậm chí đi sâu vào việc triển khai các thuật toán đề xuất. Tuy nhiên, các nguyên tắc cốt lõi về việc phân tách vấn đề thành các bước nhỏ hơn và tận dụng các kỹ thuật lập trình hàm vẫn giữ nguyên.

Phát Hiện Gian Lận

Dưới đây là một ví dụ đơn giản về cách bạn có thể triển khai một worker phát hiện gian lận sử dụng cùng phong cách Railway Oriented Programming (ROP) trong Ruby:

```ruby
class FraudDetectionWorker
  include Wisper::Publisher

  def call(transaction)
    Result.ok(FraudDetection.new(transaction))
      .and_then(ValidateTransaction.method(:validate))
      .map(AnalyzeTransactionPatterns.method(:analyze))
      .map(CheckCustomerHistory.method(:check))
      .map(EvaluateRiskFactors.method(:evaluate))
      .map(DetermineFraudProbability.method(:determine)).then do |result|

      case result
      in { err: FraudDetectionError => error }
        Honeybadger.notify(error.message, context: {transaction:})
      in { ok: FraudDetection => fraud } }
        if fraud.high_risk?
          broadcast(:high_risk_transaction, transaction:, fraud:)
        else
          broadcast(:low_risk_transaction, transaction:)
        end
      end
    end
  end
end
```

Lớp `FraudDetection` là một *value object* (đối tượng giá trị) đóng gói trạng thái phát hiện gian lận cho một giao dịch cụ thể. Nó cung cấp một cách có cấu trúc để phân tích và đánh giá rủi ro gian lận liên quan đến một giao dịch dựa trên nhiều yếu tố rủi ro khác nhau.

```ruby
class FraudDetection
  RISK_THRESHOLD = 0.8

  attr_accessor :transaction, :risk_factors

  def initialize(transaction)
    self.transaction = transaction
    self.risk_factors = []
  end

  def add_risk_factor(description:, probability:)
    case { description:, probability: }
    in { description: String => desc, probability: Float => prob }
      risk_factors << { desc => prob }
    else
      raise ArgumentError, "Risk factor arguments should be string and float"
    end
  end

  def high_risk?
    fraud_probability > RISK_THRESHOLD
  end

  private

  def fraud_probability
    risk_factors.values.sum
  end
end
```

Lớp `FraudDetection` có các thuộc tính sau:

- `transaction`: Một tham chiếu đến giao dịch đang được phân tích gian lận.

- `risk_factors`: Một mảng lưu trữ các yếu tố rủi ro liên quan đến giao dịch. Mỗi yếu tố rủi ro được biểu diễn dưới dạng hash, trong đó khóa là mô tả của yếu tố rủi ro, và giá trị là xác suất gian lận liên quan đến yếu tố rủi ro đó.

Phương thức `add_risk_factor` cho phép thêm một yếu tố rủi ro vào mảng `risk_factors`. Nó nhận hai tham số: `description` là một chuỗi mô tả yếu tố rủi ro, và `probability` là một số float biểu thị xác suất gian lận liên quan đến yếu tố rủi ro đó. Chúng ta sử dụng câu lệnh điều kiện `case..in` để thực hiện kiểm tra kiểu đơn giản.

Phương thức `high_risk?` sẽ được kiểm tra ở cuối chuỗi là một phương thức vị từ so sánh `fraud_probability` (được tính bằng cách tổng hợp xác suất của tất cả các yếu tố rủi ro) với `RISK_THRESHOLD`.

Lớp `FraudDetection` cung cấp một cách đóng gói và rõ ràng để quản lý việc phát hiện gian lận cho một giao dịch. Nó cho phép thêm nhiều yếu tố rủi ro, mỗi yếu tố có mô tả và xác suất riêng, và cung cấp phương thức để xác định xem giao dịch có được coi là rủi ro cao dựa trên xác suất gian lận đã tính toán hay không. Lớp này có thể dễ dàng tích hợp vào một hệ thống phát hiện gian lận lớn hơn, nơi các thành phần khác nhau có thể phối hợp để đánh giá và giảm thiểu rủi ro của các giao dịch gian lận.

Cuối cùng, vì đây là một cuốn sách về lập trình sử dụng AI, dưới đây là một ví dụ về việc triển khai lớp `CheckCustomerHistory` tận dụng xử lý AI bằng cách sử dụng module `ChatCompletion` từ thư viện Raix của tôi:

```
class CheckCustomerHistory
  include Raix::ChatCompletion

  attr_accessor :fraud_detection

  INSTRUCTION = <<~END
    You are an AI assistant tasked with checking a customer's transaction
    history for potential fraud indicators. Given the current transaction
    and the customer's past transactions, analyze the data to identify any
    suspicious patterns or anomalies.

    Consider factors such as the frequency of transactions, transaction
    amounts, geographical locations, and any deviations from the customer's
    typical behavior to generate a probability score as a float in the range
    of 0 to 1 (with 1 being absolute certainty of fraud).

    Output the results of your analysis, highlighting any red flags or areas
    of concern in the following JSON format:

    { description: <Summary of your findings>, probability: <Float> }
  END

  def self.check(fraud_detection)
    new(fraud_detection).call
  end

  def call
    chat_completion(json: true).tap do |result|
      fraud_detection.add_risk_factor(**result)
    end
    Result.ok(fraud_detection)
  rescue StandardError => e
    Result.err(FraudDetectionError.new(e))
  end

  private

  def initialize(fraud_detection)
    self.fraud_detection = fraud_detection
  end

  def transcript
```

```
43      tx_history = fraud_detection.transaction.user.tx_history
44      [
45        { system: INSTRUCTION },
46        { user: "Transaction history: #{tx_history.to_json}" },
47        { assistant: "OK. Please provide the current transaction." },
48        { user: "Current transaction: #{fraud_detection.transaction.to_json}" }
49      ]
50    end
51  end
```

Trong ví dụ này, `CheckCustomerHistory` định nghĩa một hằng số `INSTRUCTION` cung cấp hướng dẫn cụ thể cho mô hình AI về cách phân tích lịch sử giao dịch của khách hàng để tìm các dấu hiệu gian lận thông qua chỉ thị hệ thống

Phương thức `self.check` là một phương thức lớp khởi tạo một thực thể mới của `CheckCustomerHistory` với đối tượng `fraud_detection` và gọi phương thức `call` để thực hiện phân tích lịch sử khách hàng.

Bên trong phương thức `call`, lịch sử giao dịch của khách hàng được truy xuất và định dạng thành một bản ghi được chuyển đến mô hình AI. Mô hình AI phân tích lịch sử giao dịch dựa trên các hướng dẫn đã cung cấp và trả về bản tóm tắt các phát hiện của nó.

Các phát hiện được thêm vào đối tượng `fraud_detection`, và đối tượng `fraud_detection` đã cập nhật được trả về như một `Result` thành công.

Bằng cách tận dụng mô-đun `ChatCompletion`, lớp `CheckCustomerHistory` có thể sử dụng sức mạnh của AI để phân tích lịch sử giao dịch của khách hàng và xác định các dấu hiệu gian lận tiềm ẩn. Điều này cho phép các kỹ thuật phát hiện gian lận tinh vi và thích ứng hơn, vì mô hình AI có thể học và thích nghi với các mẫu và bất thường mới theo thời gian.

`FraudDetectionWorker` đã cập nhật và lớp `CheckCustomerHistory` cho thấy cách các worker AI có thể được tích hợp một cách liền mạch, nâng cao quy trình phát hiện gian lận với khả năng phân tích và ra quyết định thông minh.

Phân tích Cảm xúc Khách hàng

Đây là một ví dụ tương tự nữa về cách bạn có thể triển khai một worker phân tích cảm xúc khách hàng. Lần này sẽ có ít giải thích hơn, vì bạn hẳn đã nắm được cách thức hoạt động của phong cách lập trình này:

```
class CustomerSentimentAnalysisWorker
  include Wisper::Publisher

  def call(feedback)
    Result.ok(feedback)
      .and_then(PreprocessFeedback.method(:preprocess))
      .map(PerformSentimentAnalysis.method(:analyze))
      .map(ExtractKeyPhrases.method(:extract))
      .map(IdentifyTrends.method(:identify))
      .map(GenerateInsights.method(:generate)).then do |result|

      case result
      in { err: SentimentAnalysisError => error }
        Honeybadger.notify(error.message, context: {feedback:})
      in { ok: SentimentAnalysisResult => result }
        broadcast(:sentiment_analysis_completed, result)
      end
    end
  end
end
```

Trong ví dụ này, `CustomerSentimentAnalysisWorker` bao gồm các bước tiền xử lý phản hồi (ví dụ: loại bỏ nhiễu, phân tách từ), thực hiện phân tích cảm xúc để xác định tình cảm tổng thể (tích cực, tiêu cực hoặc trung tính), trích xuất các cụm từ và chủ đề chính, xác định xu hướng và mô hình, và tạo ra các thông tin chi tiết có thể hành động dựa trên phân tích.

Ứng dụng Y tế

Trong lĩnh vực y tế, các worker AI có thể hỗ trợ các chuyên gia y tế và nhà nghiên cứu trong nhiều nhiệm vụ khác nhau, dẫn đến cải thiện kết quả điều trị cho bệnh nhân và đẩy nhanh các khám phá y học. Một số ví dụ bao gồm:

Tiếp nhận Bệnh nhân

Các worker AI có thể tối ưu hóa quy trình tiếp nhận bệnh nhân bằng cách tự động hóa nhiều nhiệm vụ và cung cấp hỗ trợ thông minh.

Lên lịch hẹn: Các worker AI có thể xử lý việc lên lịch hẹn bằng cách hiểu được sở thích của bệnh nhân, thời gian rảnh và mức độ khẩn cấp của nhu cầu y tế. Chúng có thể tương tác với bệnh nhân thông qua giao diện hội thoại, hướng dẫn họ qua quy trình đặt lịch và tìm các khung giờ hẹn phù hợp nhất dựa trên yêu cầu của bệnh nhân và lịch trình của nhà cung cấp dịch vụ y tế.

Thu thập Tiền sử Bệnh: Trong quá trình tiếp nhận bệnh nhân, các worker AI có thể hỗ trợ thu thập và ghi chép tiền sử bệnh của bệnh nhân. Chúng có thể tham gia đối thoại tương tác với bệnh nhân, đặt các câu hỏi liên quan về tình trạng bệnh trong quá khứ, thuốc men, dị ứng và tiền sử gia đình. Các worker AI có thể sử dụng các kỹ thuật xử lý ngôn ngữ tự nhiên để diễn giải và cấu trúc thông tin thu thập được, đảm bảo thông tin được ghi lại chính xác trong hồ sơ sức khỏe điện tử của bệnh nhân.

Đánh giá và Phân loại Triệu chứng: Các worker AI có thể thực hiện đánh giá triệu chứng ban đầu bằng cách hỏi bệnh nhân về các triệu chứng hiện tại, thời gian, mức độ nghiêm trọng và các yếu tố liên quan. Bằng cách tận dụng cơ sở kiến thức y tế và các mô hình học máy, những worker này có thể phân tích thông tin được cung cấp và đưa ra chẩn đoán phân biệt sơ bộ hoặc đề xuất các bước tiếp theo phù hợp, như lên lịch tư vấn với nhà cung cấp dịch vụ y tế hoặc đề xuất các biện pháp tự chăm sóc.

Xác minh Bảo hiểm: Các worker AI có thể hỗ trợ xác minh bảo hiểm trong quá trình tiếp nhận bệnh nhân. Chúng có thể thu thập thông tin bảo hiểm của bệnh nhân, liên lạc

với các nhà cung cấp bảo hiểm thông qua API hoặc dịch vụ web, và xác minh tính đủ điều kiện bảo hiểm và quyền lợi. Việc tự động hóa này giúp tối ưu hóa quy trình xác minh bảo hiểm, giảm gánh nặng hành chính và đảm bảo thu thập thông tin chính xác.

Giáo dục và Hướng dẫn Bệnh nhân: Các worker AI có thể cung cấp cho bệnh nhân các tài liệu giáo dục và hướng dẫn liên quan dựa trên tình trạng bệnh cụ thể hoặc các thủ thuật sắp tới. Chúng có thể cung cấp nội dung được cá nhân hóa, trả lời các câu hỏi thường gặp và đưa ra hướng dẫn về chuẩn bị trước khi khám, hướng dẫn sử dụng thuốc hoặc chăm sóc sau điều trị. Điều này giúp bệnh nhân luôn được thông tin và tham gia trong suốt hành trình chăm sóc sức khỏe của họ.

Bằng cách tận dụng các worker AI trong quá trình tiếp nhận bệnh nhân, các tổ chức y tế có thể nâng cao hiệu quả, giảm thời gian chờ đợi và cải thiện trải nghiệm tổng thể của bệnh nhân. Những worker này có thể xử lý các nhiệm vụ thường xuyên, thu thập thông tin chính xác và cung cấp hỗ trợ cá nhân hóa, cho phép các chuyên gia y tế tập trung vào việc cung cấp dịch vụ chăm sóc chất lượng cao cho bệnh nhân.

Đánh giá Nguy cơ Bệnh nhân

Các worker AI có thể đóng vai trò quan trọng trong việc đánh giá nguy cơ của bệnh nhân bằng cách phân tích nhiều nguồn dữ liệu và áp dụng các kỹ thuật phân tích nâng cao.

Tích hợp Dữ liệu: Các worker AI có thể thu thập và hiểu được dữ liệu bệnh nhân từ nhiều nguồn khác nhau, như hồ sơ sức khỏe điện tử (EHR), hình ảnh y tế, kết quả xét nghiệm, thiết bị đeo và các yếu tố xã hội ảnh hưởng đến sức khỏe. Bằng cách tổng hợp những thông tin này thành hồ sơ bệnh nhân toàn diện, các worker AI có thể cung cấp cái nhìn tổng thể về tình trạng sức khỏe và các yếu tố nguy cơ của bệnh nhân.

Phân tầng Nguy cơ: Các worker AI có thể sử dụng các mô hình dự đoán để phân loại bệnh nhân thành các nhóm nguy cơ khác nhau dựa trên đặc điểm cá nhân và dữ liệu sức khỏe của họ. Việc phân tầng nguy cơ này giúp các nhà cung cấp dịch vụ y tế ưu tiên

những bệnh nhân cần được chú ý hoặc can thiệp ngay lập tức. Ví dụ, những bệnh nhân được xác định có nguy cơ cao đối với một tình trạng cụ thể có thể được đánh dấu để theo dõi chặt chẽ hơn, áp dụng các biện pháp phòng ngừa hoặc can thiệp sớm.

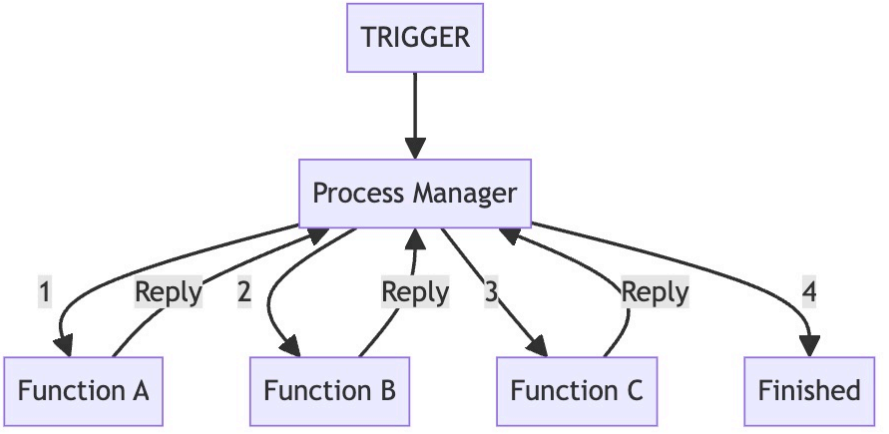
Hồ sơ Nguy cơ Cá nhân hóa: Các worker AI có thể tạo ra các hồ sơ nguy cơ được cá nhân hóa cho từng bệnh nhân, làm nổi bật các yếu tố cụ thể góp phần vào điểm số nguy cơ của họ. Những hồ sơ này có thể bao gồm thông tin chi tiết về lối sống của bệnh nhân, khuynh hướng di truyền, các yếu tố môi trường và các yếu tố xã hội ảnh hưởng đến sức khỏe. Bằng cách cung cấp phân tích chi tiết về các yếu tố nguy cơ, các worker AI có thể giúp nhà cung cấp dịch vụ y tế điều chỉnh các chiến lược phòng ngừa và kế hoạch điều trị phù hợp với nhu cầu của từng bệnh nhân.

Giám sát Nguy cơ Liên tục: Các worker AI có thể liên tục theo dõi dữ liệu bệnh nhân và cập nhật đánh giá nguy cơ theo thời gian thực. Khi có thông tin mới, chẳng hạn như thay đổi về dấu hiệu sinh tồn, kết quả xét nghiệm hoặc việc tuân thủ dùng thuốc, các worker AI có thể tính toán lại điểm số nguy cơ và cảnh báo cho nhà cung cấp dịch vụ y tế về bất kỳ thay đổi đáng kể nào. Việc giám sát chủ động này cho phép can thiệp kịp thời và điều chỉnh kế hoạch chăm sóc bệnh nhân.

Hỗ trợ Ra quyết định Lâm sàng: Các worker AI có thể tích hợp kết quả đánh giá nguy cơ vào hệ thống hỗ trợ ra quyết định lâm sàng, cung cấp cho nhà cung cấp dịch vụ y tế các khuyến nghị và cảnh báo dựa trên bằng chứng. Ví dụ, nếu điểm số nguy cơ của bệnh nhân đối với một tình trạng cụ thể vượt quá một ngưỡng nhất định, worker AI có thể nhắc nhở nhà cung cấp dịch vụ y tế xem xét các xét nghiệm chẩn đoán cụ thể, biện pháp phòng ngừa hoặc các phương án điều trị dựa trên hướng dẫn lâm sàng và thực hành tốt nhất.

Những thành phần xử lý này có thể xử lý khối lượng lớn dữ liệu bệnh nhân, áp dụng các phân tích phức tạp, và tạo ra những hiểu biết có thể hành động để hỗ trợ việc ra quyết định lâm sàng. Cuối cùng, điều này dẫn đến cải thiện kết quả điều trị cho bệnh nhân, giảm chi phí chăm sóc sức khỏe, và nâng cao việc quản lý sức khỏe cộng đồng.

Thành phần AI như một Trình Quản lý Quy trình



Trong bối cảnh các ứng dụng dựa trên AI, một thành phần xử lý có thể được thiết kế để hoạt động như một Trình Quản lý Quy trình, như được mô tả trong cuốn sách "Enterprise Integration Patterns" của Gregor Hohpe. Trình Quản lý Quy trình là một thành phần trung tâm duy trì trạng thái của một quy trình và xác định các bước xử lý tiếp theo dựa trên các kết quả trung gian.

Khi một thành phần xử lý AI đóng vai trò là Trình Quản lý Quy trình, nó nhận một thông điệp đến để khởi tạo quy trình, được gọi là *thông điệp kích hoạt*. Thành phần xử lý AI sau đó duy trì trạng thái của việc thực thi quy trình (dưới dạng bản ghi hội thoại) và xử lý thông điệp thông qua một chuỗi các bước xử lý được triển khai dưới dạng các hàm công cụ, có thể là tuần tự hoặc song song, và được gọi theo quyết định của nó.

Nếu bạn đang sử dụng một loại mô hình AI như GPT-4 có khả năng thực thi các hàm song song thì thành phần xử lý của bạn có thể thực hiện nhiều bước cùng một lúc. Thành thật mà nói, tôi chưa từng thử điều này và trực giác mách bảo rằng kết quả có thể khác nhau tùy trường hợp.

Sau mỗi bước xử lý riêng lẻ, quyền điều khiển được trả về cho thành phần xử lý AI, cho phép nó xác định (các) bước xử lý tiếp theo dựa trên trạng thái hiện tại và các kết quả thu được.

Lưu trữ Thông điệp Kích hoạt của Bạn

Theo kinh nghiệm của tôi, triển khai thông điệp kích hoạt của bạn như một đối tượng được hỗ trợ bởi cơ sở dữ liệu là một ý tưởng thông minh. Bằng cách đó, mỗi phiên bản quy trình được xác định bởi một khóa chính duy nhất và cung cấp cho bạn một nơi để lưu trữ trạng thái liên quan đến việc thực thi, bao gồm cả bản ghi hội thoại của AI.

Ví dụ, đây là phiên bản đơn giản hóa của lớp mô hình `AccountChange` của Olympia, đại diện cho một yêu cầu thay đổi tài khoản của người dùng.

```
# == Schema Information
#
# Table name: account_changes
#
#  id          :uuid             not null, primary key
#  description :string
#  state       :string           not null
#  transcript  :jsonb
#  created_at  :datetime         not null
#  updated_at  :datetime         not null
#  account_id  :uuid             not null
#
# Indexes
#
#  index_account_changes_on_account_id  (account_id)
#
# Foreign Keys
#
#  fk_rails_...  (account_id => accounts.id)
#
class AccountChange < ApplicationRecord
  belongs_to :account

  validates :description, presence: true
```

```
25
26    after_commit -> {
27      broadcast(:account_change_requested, self)
28    }, on: :create
29
30    state_machine initial: :requested do
31      event :completed do
32        transition all => :complete
33      end
34      event :failed do
35        transition all => :requires_human_review
36      end
37    end
38  end
```

Lớp AccountChange đóng vai trò như một thông điệp kích hoạt để bắt đầu quy trình xử lý yêu cầu thay đổi tài khoản. Hãy chú ý cách nó được phát tới hệ thống phụ xuất bản/đăng ký dựa trên Wisper của Olympia sau khi giao dịch tạo mới hoàn tất.

Việc lưu trữ thông điệp kích hoạt trong cơ sở dữ liệu như thế này cung cấp một bản ghi liên tục cho mỗi yêu cầu thay đổi tài khoản. Mỗi thể hiện của lớp AccountChange được gán một khóa chính duy nhất, cho phép dễ dàng nhận diện và theo dõi các yêu cầu riêng lẻ. Điều này đặc biệt hữu ích cho mục đích ghi nhật ký kiểm toán, vì nó cho phép hệ thống duy trì một bản ghi lịch sử của tất cả các thay đổi tài khoản, bao gồm thời điểm yêu cầu, những thay đổi được yêu cầu, và trạng thái hiện tại của mỗi yêu cầu.

Trong ví dụ đã cho, lớp AccountChange bao gồm các trường như `description` để nắm bắt chi tiết của thay đổi được yêu cầu, `state` để biểu thị trạng thái hiện tại của yêu cầu (ví dụ: đã yêu cầu, hoàn thành, cần xem xét thủ công), và `transcript` để lưu trữ bản ghi hội thoại của AI liên quan đến yêu cầu. Trường `description` là lệnh nhắc thực tế được sử dụng để khởi tạo cuộc trò chuyện đầu tiên với AI. Việc lưu trữ dữ liệu này cung cấp ngữ cảnh có giá trị và cho phép theo dõi và phân tích quy trình thay đổi tài khoản tốt hơn.

Việc lưu trữ thông điệp kích hoạt trong cơ sở dữ liệu cho phép xử lý lỗi và khôi phục mạnh mẽ. Nếu xảy ra lỗi trong quá trình xử lý yêu cầu thay đổi tài khoản, hệ thống sẽ

đánh dấu yêu cầu là thất bại và chuyển nó sang trạng thái cần can thiệp của con người. Điều này đảm bảo không có yêu cầu nào bị mất hoặc bị quên, và mọi vấn đề có thể được xử lý và giải quyết đúng cách.

Worker AI, với vai trò là Trình Quản lý Quy trình, cung cấp một điểm kiểm soát trung tâm và cho phép các khả năng báo cáo và gỡ lỗi quy trình mạnh mẽ. Tuy nhiên, điều quan trọng cần lưu ý là việc sử dụng worker AI như một Trình Quản lý Quy trình cho mọi kịch bản quy trình trong ứng dụng của bạn có thể là quá mức cần thiết.

Tích hợp Worker AI Vào Kiến trúc Ứng dụng Của Bạn

Khi tích hợp worker AI vào kiến trúc ứng dụng của bạn, một số cân nhắc kỹ thuật cần được giải quyết để đảm bảo tích hợp suôn sẻ và giao tiếp hiệu quả giữa worker AI và các thành phần ứng dụng khác. Phần này xem xét các khía cạnh chính của việc thiết kế giao diện, xử lý luồng dữ liệu và quản lý vòng đời của worker AI.

Thiết kế Giao diện và Giao thức Giao tiếp Rõ ràng

Để tạo điều kiện tích hợp suôn sẻ giữa worker AI và các thành phần ứng dụng khác, việc xác định giao diện và giao thức giao tiếp rõ ràng là rất quan trọng. Hãy xem xét các cách tiếp cận sau:

Tích hợp dựa trên API: Hiển thị chức năng của worker AI thông qua các API được định nghĩa rõ ràng, như các endpoint RESTful hoặc schema GraphQL. Điều này cho phép các thành phần khác tương tác với worker AI bằng cách sử dụng các yêu cầu và phản hồi HTTP tiêu chuẩn. Tích hợp dựa trên API cung cấp một hợp đồng rõ ràng giữa worker AI và các thành phần sử dụng, giúp việc phát triển, kiểm thử và bảo trì các điểm tích hợp dễ dàng hơn.

Giao tiếp dựa trên Thông điệp: Triển khai các mẫu giao tiếp dựa trên thông điệp, như hàng đợi thông điệp hoặc hệ thống xuất bản-đăng ký, để cho phép tương tác bất đồng bộ giữa worker AI và các thành phần khác. Cách tiếp cận này tách biệt worker AI khỏi phần còn lại của ứng dụng, cho phép khả năng mở rộng, chịu lỗi và liên kết lỏng lẻo tốt hơn. Giao tiếp dựa trên thông điệp đặc biệt hữu ích khi việc xử lý được thực hiện bởi worker AI tốn thời gian hoặc tài nguyên, vì nó cho phép các phần khác của ứng dụng tiếp tục thực thi mà không cần đợi worker AI hoàn thành nhiệm vụ của họ.

Kiến trúc hướng sự kiện: Thiết kế hệ thống của bạn xoay quanh các sự kiện và kích hoạt kích hoạt worker AI khi các điều kiện cụ thể được đáp ứng. Worker AI có thể đăng ký các sự kiện liên quan và phản ứng tương ứng, thực hiện các nhiệm vụ được chỉ định khi sự kiện xảy ra. Kiến trúc hướng sự kiện cho phép xử lý thời gian thực và cho phép worker AI được gọi theo yêu cầu, giảm tiêu thụ tài nguyên không cần thiết. Cách tiếp cận này phù hợp cho các tình huống mà worker AI cần phản ứng với các hành động cụ thể hoặc thay đổi trong trạng thái ứng dụng.

Xử lý Luồng Dữ liệu và Đồng bộ hóa

Khi tích hợp worker AI vào ứng dụng của bạn, việc đảm bảo luồng dữ liệu suôn sẻ và duy trì tính nhất quán dữ liệu giữa worker AI và các thành phần khác là rất quan trọng. Hãy xem xét các khía cạnh sau:

Chuẩn bị Dữ liệu: Trước khi đưa dữ liệu vào worker AI, bạn có thể cần thực hiện các tác vụ chuẩn bị dữ liệu khác nhau, như làm sạch, định dạng và/hoặc chuyển đổi dữ liệu đầu vào. Bạn không chỉ muốn đảm bảo rằng worker AI có thể xử lý hiệu quả, mà còn muốn đảm bảo rằng bạn không lãng phí token bằng cách chú ý đến thông tin mà worker có thể coi là vô dụng trong trường hợp tốt nhất, gây nhiễu trong trường hợp xấu nhất. Chuẩn bị dữ liệu có thể bao gồm các tác vụ như loại bỏ nhiễu, xử lý giá trị thiếu, hoặc chuyển đổi kiểu dữ liệu.

Tính bền vững của Dữ liệu: Bạn sẽ lưu trữ và duy trì dữ liệu chảy vào và ra khỏi worker AI như thế nào? Xem xét các yếu tố như khối lượng dữ liệu, mẫu truy vấn và khả năng

mở rộng. Bạn có cần lưu trữ bản ghi của AI như một phản ánh về "quá trình suy nghĩ" của nó cho mục đích kiểm toán hoặc gỡ lỗi, hay chỉ cần có bản ghi kết quả là đủ?

Truy xuất Dữ liệu: Việc lấy dữ liệu cần thiết cho các worker có thể liên quan đến truy vấn cơ sở dữ liệu, đọc từ tập tin, hoặc truy cập API bên ngoài. Hãy đảm bảo xem xét độ trễ và cách thức worker AI sẽ truy cập dữ liệu mới nhất. Họ có cần quyền truy cập đầy đủ vào cơ sở dữ liệu của bạn hay bạn nên giới hạn phạm vi truy cập của họ theo nhiệm vụ cụ thể? Còn về khả năng mở rộng thì sao? Hãy cân nhắc các cơ chế bộ nhớ đệm để cải thiện hiệu suất và giảm tải cho các nguồn dữ liệu.

Đồng bộ hóa Dữ liệu.: Khi nhiều thành phần, bao gồm cả worker AI, truy cập và sửa đổi dữ liệu chung, việc triển khai các cơ chế đồng bộ hóa phù hợp để duy trì tính nhất quán của dữ liệu là rất quan trọng. Các chiến lược khóa cơ sở dữ liệu, như khóa lạc quan hoặc bi quan, có thể giúp bạn ngăn chặn xung đột và đảm bảo tính toàn vẹn dữ liệu. Triển khai các kỹ thuật quản lý giao dịch để nhóm các thao tác dữ liệu liên quan và duy trì các thuộc tính ACID (tính nguyên tố, tính nhất quán, tính độc lập và tính bền vững)

Xử lý Lỗi và Phục hồi: Triển khai các cơ chế xử lý lỗi và phục hồi mạnh mẽ để giải quyết các vấn đề liên quan đến dữ liệu có thể phát sinh trong quá trình luồng dữ liệu. Xử lý ngoại lệ một cách khéo léo và cung cấp thông báo lỗi có ý nghĩa để hỗ trợ gỡ lỗi. Triển khai cơ chế thử lại và chiến lược dự phòng để xử lý các lỗi tạm thời hoặc gián đoạn mạng. Xác định quy trình rõ ràng cho việc phục hồi và khôi phục dữ liệu trong trường hợp dữ liệu bị hỏng hoặc mất.

Bằng cách thiết kế và triển khai cẩn thận các cơ chế luồng dữ liệu và đồng bộ hóa, bạn có thể đảm bảo rằng các worker AI có quyền truy cập vào dữ liệu chính xác, nhất quán và cập nhật. Điều này cho phép họ thực hiện nhiệm vụ một cách hiệu quả và tạo ra kết quả đáng tin cậy.

Quản lý Vòng đời của Worker AI

Phát triển quy trình chuẩn hóa để khởi tạo và cấu hình worker AI. Tôi thiên về các framework chuẩn hóa cách định nghĩa các cài đặt như tên mô hình, chỉ thị hệ thống

và định nghĩa hàm. Đảm bảo quy trình khởi tạo được tự động hóa và có thể tái tạo để thuận tiện cho việc triển khai và mở rộng.

Triển khai các cơ chế giám sát và ghi nhật ký toàn diện để theo dõi tình trạng và hiệu suất của worker AI. Thu thập các chỉ số như mức sử dụng tài nguyên, thời gian xử lý, tỷ lệ lỗi, và thông lượng. Sử dụng hệ thống ghi nhật ký tập trung như ELK stack (Elasticsearch, Logstash, Kibana) để tổng hợp và phân tích nhật ký từ nhiều worker AI.

Xây dựng khả năng chống chịu lỗi và khả năng phục hồi vào kiến trúc worker AI. Triển khai cơ chế xử lý lỗi và phục hồi để xử lý một cách khéo léo các lỗi hoặc ngoại lệ. Mô hình Ngôn ngữ Lớn vẫn là công nghệ mới nhất; các nhà cung cấp thường gặp sự cố vào những thời điểm không mong đợi. Sử dụng cơ chế thử lại và circuit breaker để ngăn chặn lỗi dây chuyền.

Khả năng Kết hợp và Điều phối Worker AI

Một trong những lợi thế chính của kiến trúc worker AI là khả năng kết hợp, cho phép bạn kết hợp và điều phối nhiều worker AI để giải quyết các vấn đề phức tạp. Bằng cách chia nhỏ một tác vụ lớn thành các tác vụ phụ nhỏ hơn, dễ quản lý hơn, mỗi tác vụ được xử lý bởi một worker AI chuyên biệt, bạn có thể tạo ra các hệ thống mạnh mẽ và linh hoạt. Trong phần này, chúng ta sẽ khám phá các cách tiếp cận khác nhau để kết hợp và điều phối "nhiều" worker AI.

Kết nối Worker AI cho Quy trình Làm việc Nhiều Bước

Trong nhiều tình huống, một tác vụ phức tạp có thể được chia thành một chuỗi các bước tuần tự, trong đó đầu ra của một worker AI trở thành đầu vào cho worker tiếp theo. Việc kết nối các worker AI này tạo ra một quy trình làm việc hoặc pipeline nhiều bước. Mỗi worker AI trong chuỗi tập trung vào một tác vụ phụ cụ thể, và đầu ra cuối cùng là kết quả của nỗ lực kết hợp từ tất cả các worker.

Hãy xem xét một ví dụ trong ngữ cảnh của ứng dụng Ruby on Rails để xử lý nội dung do người dùng tạo ra. Quy trình làm việc bao gồm các bước sau, mà thực ra có lẽ mỗi bước đều quá đơn giản để đáng phân tách theo cách này trong các trường hợp thực tế, nhưng chúng giúp ví dụ dễ hiểu hơn:

1. **Làm sạch Văn bản**: Một worker AI chịu trách nhiệm loại bỏ thẻ HTML, chuyển đổi văn bản thành chữ thường và xử lý chuẩn hóa Unicode.

2. **Phát hiện Ngôn ngữ**: Một worker AI xác định ngôn ngữ của văn bản đã được làm sạch.

3. **Phân tích Cảm xúc**: Một worker AI xác định cảm xúc (tích cực, tiêu cực hoặc trung tính) của văn bản dựa trên ngôn ngữ đã phát hiện.

4. **Phân loại Nội dung**: Một worker AI phân loại văn bản vào các danh mục đã định nghĩa trước bằng cách sử dụng các kỹ thuật xử lý ngôn ngữ tự nhiên.

Dưới đây là một ví dụ rất đơn giản về cách bạn có thể kết nối các worker AI này bằng Ruby:

```ruby
class ContentProcessor
  def initialize(text)
    @text = text
  end

  def process
    cleaned_text = TextCleanupWorker.new(@text).call
    language = LanguageDetectionWorker.new(cleaned_text).call
    sentiment = SentimentAnalysisWorker.new(cleaned_text, language).call
    category = CategorizationWorker.new(cleaned_text, language).call

    { cleaned_text:, language:, sentiment:, category: }
  end
end
```

Trong ví dụ này, lớp `ContentProcessor` được khởi tạo với văn bản thô và kết nối các thành phần xử lý AI với nhau trong phương thức `process`. Mỗi thành phần xử lý AI thực hiện nhiệm vụ cụ thể của nó và chuyển kết quả cho thành phần tiếp theo trong

chuỗi. Kết quả cuối cùng là một hash chứa văn bản đã được làm sạch, ngôn ngữ được phát hiện, cảm xúc, và danh mục nội dung.

Xử lý Song song cho các Thành phần AI Độc lập

Trong ví dụ trước, các thành phần xử lý AI được kết nối tuần tự, trong đó mỗi thành phần xử lý văn bản và chuyển kết quả cho thành phần tiếp theo. Tuy nhiên, nếu bạn có nhiều thành phần xử lý AI có thể hoạt động độc lập trên cùng một đầu vào, bạn có thể tối ưu hóa quy trình bằng cách xử lý chúng song song.

Trong kịch bản đã cho, một khi việc làm sạch văn bản được thực hiện bởi `TextCleanupWorker`, các thành phần `LanguageDetectionWorker`, `SentimentAnalysisWorker`, và `CategorizationWorker` đều có thể xử lý văn bản đã được làm sạch một cách độc lập. Bằng cách chạy các thành phần này song song, bạn có thể giảm thời gian xử lý tổng thể và cải thiện hiệu quả của quy trình của bạn.

Để thực hiện xử lý song song trong Ruby, bạn có thể tận dụng các kỹ thuật xử lý đồng thời như luồng hoặc lập trình bất đồng bộ. Dưới đây là ví dụ về cách bạn có thể sửa đổi lớp `ContentProcessor` để xử lý ba thành phần cuối cùng song song bằng cách sử dụng luồng:

```
require 'concurrent'

class ContentProcessor
  def initialize(text)
    @text = text
  end

  def process
    cleaned_text = TextCleanupWorker.new(@text).call

    language_future = Concurrent::Future.execute do
      LanguageDetectionWorker.new(cleaned_text).call
    end

    sentiment_future = Concurrent::Future.execute do
```

```
      SentimentAnalysisWorker.new(cleaned_text).call
    end

    category_future = Concurrent::Future.execute do
      CategorizationWorker.new(cleaned_text).call
    end

    language = language_future.value
    sentiment = sentiment_future.value
    category = category_future.value

    { cleaned_text:, language:, sentiment:, category: }
  end
end
```

Trong phiên bản đã được tối ưu hóa này, chúng ta sử dụng thư viện `concurrent-ruby` để tạo các đối tượng `Concurrent::Future` cho mỗi worker AI độc lập. Một `Future` đại diện cho một phép tính sẽ được thực hiện bất đồng bộ trong một thread riêng biệt.

Sau bước làm sạch văn bản, chúng ta tạo ba đối tượng `Future`: `language_future`, `sentiment_future`, và `category_future`. Mỗi `Future` thực thi worker AI tương ứng (`LanguageDetectionWorker`, `SentimentAnalysisWorker`, và `CategorizationWorker`) trong một thread riêng biệt, truyền vào `cleaned_text` làm đầu vào.

Bằng cách gọi phương thức `value` trên mỗi `Future`, chúng ta chờ cho đến khi phép tính hoàn tất và nhận kết quả. Phương thức `value` sẽ chặn cho đến khi có kết quả, đảm bảo rằng tất cả các worker song song đã hoàn thành xử lý trước khi tiếp tục.

Cuối cùng, chúng ta xây dựng hash kết quả với văn bản đã được làm sạch và các kết quả từ các worker song song, giống như trong ví dụ ban đầu.

Bằng cách xử lý các worker AI độc lập một cách song song, bạn có thể giảm thời gian xử lý tổng thể so với việc chạy tuần tự. Việc tối ưu hóa này đặc biệt hữu ích khi xử lý các tác vụ tốn nhiều thời gian hoặc khi xử lý khối lượng dữ liệu lớn.

Tuy nhiên, cần lưu ý rằng lợi ích thực tế về hiệu năng phụ thuộc vào nhiều yếu tố, như

độ phức tạp của mỗi worker, tài nguyên hệ thống có sẵn, và chi phí quản lý thread. Luôn nên đánh giá hiệu năng và lập profile code của bạn để xác định mức độ song song tối ưu cho trường hợp cụ thể của bạn.

Ngoài ra, khi triển khai xử lý song song, hãy chú ý đến các tài nguyên được chia sẻ hoặc các phụ thuộc giữa các worker. Đảm bảo rằng các worker có thể hoạt động độc lập mà không có xung đột hoặc điều kiện tranh chấp. Nếu có các phụ thuộc hoặc tài nguyên chia sẻ, bạn có thể cần triển khai các cơ chế đồng bộ hóa phù hợp để duy trì tính toàn vẹn dữ liệu và tránh các vấn đề như tình trạng bế tắc hoặc kết quả không nhất quán.

Global Interpreter Lock (GIL) của Ruby và Xử lý Bất đồng bộ

Điều quan trọng là phải hiểu những tác động của Global Interpreter Lock (GIL) của Ruby khi xem xét xử lý bất đồng bộ dựa trên thread trong Ruby.

GIL là một cơ chế trong trình thông dịch Ruby đảm bảo rằng chỉ một thread có thể thực thi mã Ruby tại một thời điểm, ngay cả trên các bộ xử lý đa nhân. Điều này có nghĩa là mặc dù có thể tạo và quản lý nhiều thread trong một tiến trình Ruby, nhưng chỉ một thread có thể chủ động thực thi mã Ruby tại bất kỳ thời điểm nào.

GIL được thiết kế để đơn giản hóa việc triển khai trình thông dịch Ruby và cung cấp tính an toàn thread cho các cấu trúc dữ liệu nội bộ của Ruby. Tuy nhiên, nó cũng giới hạn khả năng thực thi song song thực sự của mã Ruby.

Khi bạn sử dụng thread trong Ruby, chẳng hạn như với thư viện `concurrent-ruby` hoặc lớp `Thread` tích hợp sẵn, các thread phải tuân theo các ràng buộc của GIL. GIL cho phép mỗi thread thực thi mã Ruby trong một khoảng thời gian ngắn trước khi chuyển sang thread khác, tạo ra ảo giác về việc thực thi đồng thời.

Tuy nhiên, do GIL, việc thực thi mã Ruby thực sự vẫn là tuần tự. Trong khi một thread đang thực thi mã Ruby, các thread khác về cơ bản bị tạm dừng, chờ đến lượt

để có được GIL và thực thi.

Điều này có nghĩa là xử lý bất đồng bộ dựa trên thread trong Ruby hiệu quả nhất đối với các tác vụ gắn với I/O, chẳng hạn như chờ phản hồi API bên ngoài (như các mô hình ngôn ngữ lớn được lưu trữ bởi bên thứ 3) hoặc thực hiện các thao tác I/O với tệp. Khi một thread gặp một thao tác I/O, nó có thể giải phóng GIL, cho phép các thread khác thực thi trong khi chờ I/O hoàn thành.

Mặt khác, đối với các tác vụ gắn với CPU, chẳng hạn như các phép tính phức tạp hoặc xử lý worker AI chạy lâu, GIL có thể giới hạn lợi ích tiềm năng về hiệu năng của việc chạy song song dựa trên thread. Vì chỉ một thread có thể thực thi mã Ruby tại một thời điểm, thời gian thực thi tổng thể có thể không giảm đáng kể so với xử lý tuần tự.

Để đạt được thực thi song song thực sự cho các tác vụ gắn với CPU trong Ruby, bạn có thể cần khám phá các phương pháp thay thế, chẳng hạn như:

- Sử dụng xử lý song song dựa trên tiến trình với nhiều tiến trình Ruby, mỗi tiến trình chạy trên một nhân CPU riêng biệt.
- Tận dụng các thư viện hoặc framework bên ngoài cung cấp các phần mở rộng gốc hoặc giao diện với các ngôn ngữ không có GIL, như C hoặc Rust.,
- Sử dụng các framework tính toán phân tán hoặc hàng đợi tin nhắn để phân phối tác vụ trên nhiều máy hoặc tiến trình.

Điều quan trọng là phải xem xét bản chất của tác vụ và các giới hạn do GIL áp đặt khi thiết kế và triển khai xử lý bất đồng bộ trong Ruby. Mặc dù xử lý bất đồng bộ dựa trên thread có thể mang lại lợi ích cho các tác vụ gắn với I/O, nhưng nó có thể không mang lại cải thiện hiệu năng đáng kể cho các tác vụ gắn với CPU do các ràng buộc của GIL.

Kỹ thuật Tổng hợp để Cải thiện Độ chính xác

Kỹ thuật tổng hợp liên quan đến việc kết hợp đầu ra của nhiều worker AI để cải thiện độ chính xác hoặc độ tin cậy tổng thể của hệ thống. Thay vì chỉ dựa vào một worker AI duy nhất, kỹ thuật tổng hợp tận dụng trí tuệ tập thể của nhiều worker để đưa ra quyết định sáng suốt hơn.

Các mô hình kết hợp đặc biệt quan trọng khi các phần khác nhau trong quy trình xử lý của bạn hoạt động tốt nhất với các mô hình AI khác nhau, điều này phổ biến hơn bạn nghĩ. Những mô hình mạnh mẽ như GPT-4 có chi phí cực kỳ đắt đỏ so với các lựa chọn mã nguồn mở kém khả năng hơn, và có lẽ không cần thiết cho mọi bước trong quy trình xử lý của ứng dụng của bạn.

Một kỹ thuật kết hợp phổ biến là bỏ phiếu theo đa số, trong đó nhiều công cụ AI độc lập xử lý cùng một đầu vào, và kết quả đầu ra cuối cùng được xác định bởi sự đồng thuận của đa số. Phương pháp này có thể giúp giảm thiểu tác động của các lỗi từ từng công cụ riêng lẻ và cải thiện độ tin cậy tổng thể của hệ thống.

Hãy xem xét một ví dụ trong đó chúng ta có ba công cụ AI để phân tích cảm xúc, mỗi công cụ sử dụng một mô hình khác nhau hoặc được cung cấp các ngữ cảnh khác nhau. Chúng ta có thể kết hợp các kết quả đầu ra của chúng bằng cách bỏ phiếu theo đa số để xác định dự đoán cảm xúc cuối cùng.

```
class SentimentAnalysisEnsemble
  def initialize(text)
    @text = text
  end

  def analyze
    predictions = [
      SentimentAnalysisWorker1.new(@text).analyze,
      SentimentAnalysisWorker2.new(@text).analyze,
      SentimentAnalysisWorker3.new(@text).analyze
    ]

    predictions
      .group_by { |sentiment| sentiment }
      .max_by { |_, votes| votes.size }
      .first

  end
end
```

Trong ví dụ này, lớp `SentimentAnalysisEnsemble` được khởi tạo với văn bản và gọi ba công cụ AI phân tích cảm xúc khác nhau. Phương thức `analyze` thu thập các dự đoán từ mỗi công cụ và xác định cảm xúc theo đa số bằng cách sử dụng các phương thức `group_by` và `max_by`. Kết quả cuối cùng là cảm xúc nhận được nhiều phiếu bầu nhất từ tổ hợp các công cụ

Các tổ hợp rõ ràng là trường hợp mà việc thử nghiệm với xử lý song song có thể đáng để bạn bỏ thời gian.

Lựa Chọn và Gọi Động các Công cụ AI

Trong một số, nếu không muốn nói là hầu hết các trường hợp, việc lựa chọn công cụ AI cụ thể cần được gọi có thể phụ thuộc vào điều kiện thực thi hoặc đầu vào của người dùng. Việc lựa chọn và gọi động các công cụ AI cho phép hệ thống linh hoạt và thích ứng.

Bạn có thể thấy mình bị cám dỗ cố gắng đưa nhiều chức năng vào một công cụ AI duy nhất, cho nó nhiều hàm và một prompt phức tạp giải thích cách gọi chúng. Hãy cưỡng lại cám dỗ đó, tin tôi đi. Một trong những lý do mà phương pháp chúng ta đang thảo luận trong chương này được gọi là "Đa dạng Công cụ" là để nhắc nhở chúng ta rằng việc có nhiều công cụ chuyên biệt, mỗi công cụ thực hiện công việc nhỏ của riêng mình để phục vụ mục đích lớn hơn, là điều đáng mong muốn.

Ví dụ, hãy xem xét một ứng dụng chatbot trong đó các công cụ AI khác nhau chịu trách nhiệm xử lý các loại truy vấn khác nhau của người dùng. Dựa trên đầu vào của người dùng, ứng dụng động lựa chọn công cụ AI thích hợp để xử lý truy vấn.

```
class ChatbotController < ApplicationController
  def process_query
    query = params[:query]
    query_type = QueryClassifierWorker.new(query).classify

    case query_type
    when 'greeting'
      response = GreetingWorker.new(query).generate_response
    when 'product_inquiry'
      response = ProductInquiryWorker.new(query).generate_response
    when 'order_status'
      response = OrderStatusWorker.new(query).generate_response
    else
      response = DefaultResponseWorker.new(query).generate_response
    end

    render json: { response: response }
  end
end
```

Trong ví dụ này, `ChatbotController` nhận truy vấn của người dùng thông qua hành động `process_query`. Đầu tiên, nó sử dụng `QueryClassifierWorker` để xác định loại truy vấn. Dựa trên loại truy vấn đã được phân loại, bộ điều khiển động chọn worker AI thích hợp để tạo phản hồi. Việc lựa chọn động này cho phép chatbot xử lý các loại truy vấn khác nhau và định tuyến chúng đến các worker AI phù hợp.

Vì công việc của `QueryClassifierWorker` tương đối đơn giản và không yêu cầu nhiều ngữ cảnh hoặc định nghĩa hàm, bạn có thể triển khai nó bằng cách sử dụng LLM nhỏ siêu nhanh như `mistralai/mixtral-8x7b-instruct:nitro`. Nó có khả năng tiệm cận với mức độ của GPT-4 trong nhiều tác vụ và, tại thời điểm tôi viết điều này, Groq có thể phục vụ nó với tốc độ xử lý đáng kinh ngạc là 444 token/giây.

Kết hợp NLP truyền thống với LLM

Mặc dù Mô hình ngôn ngữ lớn (LLM) đã cách mạng hóa lĩnh vực xử lý ngôn ngữ tự nhiên (NLP), mang lại tính linh hoạt và hiệu suất vượt trội trong nhiều tác vụ khác nhau, chúng không phải lúc nào cũng là giải pháp hiệu quả nhất hoặc tiết kiệm chi phí nhất cho mọi vấn đề. Trong nhiều trường hợp, việc kết hợp các kỹ thuật NLP truyền thống với LLM có thể dẫn đến những cách tiếp cận tối ưu hơn, có mục tiêu cụ thể hơn và tiết kiệm hơn để giải quyết các thách thức NLP cụ thể.

Hãy xem LLM như những con dao đa năng Thụy Sĩ trong NLP—cực kỳ linh hoạt và mạnh mẽ, nhưng không nhất thiết là công cụ tốt nhất cho mọi công việc. Đôi khi, một công cụ chuyên dụng như cái mở nút chai hoặc cái mở hộp có thể hiệu quả và hiệu suất hơn cho một tác vụ cụ thể. Tương tự, các kỹ thuật NLP truyền thống, như phân cụm tài liệu, nhận dạng chủ đề, và phân loại, thường có thể cung cấp giải pháp có mục tiêu và tiết kiệm chi phí hơn cho một số khía cạnh trong quy trình NLP của bạn.

Một trong những ưu điểm chính của các kỹ thuật NLP truyền thống là hiệu quả tính toán của chúng. Những phương pháp này, thường dựa trên các mô hình thống kê đơn giản hơn hoặc các phương pháp dựa trên quy tắc, có thể xử lý khối lượng lớn dữ liệu văn bản nhanh hơn và với chi phí tính toán thấp hơn so với LLM. Điều này làm cho chúng đặc biệt phù hợp cho các tác vụ liên quan đến việc phân tích và tổ chức các kho tài liệu lớn, chẳng hạn như phân cụm các bài viết tương tự hoặc xác định các chủ đề chính trong một bộ sưu tập văn bản.

Hơn nữa, các kỹ thuật NLP truyền thống thường có thể đạt được độ chính xác và độ chính xác cao cho các tác vụ cụ thể, đặc biệt khi được huấn luyện trên các tập dữ liệu chuyên ngành. Ví dụ, một bộ phân loại tài liệu được điều chỉnh tốt sử dụng các thuật toán học máy truyền thống như Máy vector hỗ trợ (SVM) hoặc Naive Bayes có thể phân loại chính xác tài liệu vào các danh mục định sẵn với chi phí tính toán tối thiểu.

Tuy nhiên, LLM thực sự tỏa sáng khi đến với các tác vụ đòi hỏi sự hiểu biết sâu sắc về ngôn ngữ, ngữ cảnh và lập luận. Khả năng tạo ra văn bản mạch lạc và phù hợp với ngữ cảnh, trả lời câu hỏi và tóm tắt các đoạn văn dài của chúng là không thể so sánh với các phương pháp NLP truyền thống. LLM có thể xử lý hiệu quả các hiện tượng ngôn ngữ phức tạp, như sự mơ hồ, đồng tham chiếu và các cụm từ thành ngữ, làm cho chúng trở nên vô giá cho các tác vụ đòi hỏi việc tạo ra hoặc hiểu ngôn ngữ tự nhiên.

Sức mạnh thực sự nằm ở việc kết hợp các kỹ thuật NLP truyền thống với LLM để tạo ra các phương pháp tiếp cận lai tận dụng điểm mạnh của cả hai. Bằng cách sử dụng các phương pháp NLP truyền thống cho các tác vụ như tiền xử lý tài liệu, phân cụm và trích xuất chủ đề, bạn có thể tổ chức và cấu trúc dữ liệu văn bản của mình một cách hiệu quả. Thông tin có cấu trúc này sau đó có thể được đưa vào LLM để thực hiện các tác vụ nâng cao hơn, như tạo tóm tắt, trả lời câu hỏi hoặc tạo báo cáo tổng hợp.

Ví dụ, hãy xem xét một trường hợp sử dụng khi bạn muốn tạo báo cáo xu hướng cho một lĩnh vực cụ thể dựa trên một kho tài liệu xu hướng riêng lẻ lớn. Thay vì chỉ dựa vào LLM, vốn có thể tốn kém về mặt tính toán và mất nhiều thời gian để xử lý khối lượng văn bản lớn, bạn có thể sử dụng phương pháp tiếp cận lai:

1. Sử dụng các kỹ thuật NLP truyền thống, như mô hình hóa chủ đề (ví dụ: Phân bổ Dirichlet tiềm ẩn) hoặc thuật toán phân cụm (ví dụ: K-means), để nhóm các tài liệu xu hướng tương tự lại với nhau và xác định các chủ đề và chủ điểm chính trong kho tài liệu.
2. Đưa các tài liệu đã được phân cụm và các chủ đề đã xác định vào LLM, tận dụng khả năng hiểu và tạo ngôn ngữ vượt trội của nó để tạo ra các bản tóm tắt mạch lạc và đầy thông tin cho từng cụm hoặc chủ đề.

3. Cuối cùng, sử dụng LLM để tạo một báo cáo xu hướng toàn diện bằng cách kết hợp các bản tóm tắt riêng lẻ, làm nổi bật các xu hướng quan trọng nhất và cung cấp những hiểu biết sâu sắc và khuyến nghị dựa trên thông tin tổng hợp.

Bằng cách kết hợp các kỹ thuật NLP truyền thống với LLM theo cách này, bạn có thể xử lý hiệu quả lượng lớn dữ liệu văn bản, trích xuất những hiểu biết có ý nghĩa và tạo ra các báo cáo chất lượng cao trong khi tối ưu hóa tài nguyên tính toán và chi phí.

Khi bạn bắt đầu các dự án NLP, điều quan trọng là phải đánh giá cẩn thận các yêu cầu và ràng buộc cụ thể của từng nhiệm vụ và xem xét cách kết hợp các phương pháp NLP truyền thống với LLMs để đạt được kết quả tốt nhất. Bằng cách kết hợp hiệu quả và độ chính xác của các kỹ thuật truyền thống với tính linh hoạt và sức mạnh của LLMs, bạn có thể tạo ra các giải pháp NLP vừa hiệu quả vừa tiết kiệm, mang lại giá trị cho người dùng và các bên liên quan của bạn.

Sử Dụng Công Cụ

Trong lĩnh vực phát triển ứng dụng dựa trên AI, khái niệm "sử dụng công cụ" hay "gọi hàm" đã nổi lên như một kỹ thuật mạnh mẽ cho phép LLM của bạn kết nối với các công cụ bên ngoài, APIs, hàm, cơ sở dữ liệu, và các tài nguyên khác. Phương pháp này cho phép một tập hợp hành vi phong phú hơn là chỉ đơn thuần xuất ra văn bản, và tạo ra những tương tác động giữa các thành phần AI của bạn với phần còn lại của hệ sinh thái ứng dụng. Như chúng ta sẽ xem xét trong chương này, việc sử dụng công cụ cũng cho bạn khả năng làm cho mô hình AI tạo ra dữ liệu theo những cách có cấu trúc.

Sử Dụng Công Cụ Là Gì?

Sử dụng công cụ, còn được gọi là gọi hàm, là một kỹ thuật cho phép các nhà phát triển chỉ định một danh sách các hàm mà LLM có thể tương tác trong quá trình sinh nội dung. Những công cụ này có thể từ các hàm tiện ích đơn giản đến các API hoặc truy vấn cơ

sở dữ liệu phức tạp. Bằng cách cung cấp cho LLM quyền truy cập vào các công cụ này, các nhà phát triển có thể mở rộng khả năng của mô hình và cho phép nó thực hiện các tác vụ đòi hỏi kiến thức hoặc hành động từ bên ngoài.

Hình 8. Ví dụ về định nghĩa hàm cho một worker AI phân tích tài liệu

```
FUNCTION = {
  name: "save_analysis",
  description: "Save analysis data for document",
  parameters: {
    type: "object",
    properties: {
      title: {
        type: "string",
        maxLength: 140
      },
      summary: {
        type: "string",
        description: "comprehensive multi-paragraph summary with
                      overview and list of sections (if applicable)"
      },
      tags: {
        type: "array",
        items: {
          type: "string",
          description: "lowercase tags representing main themes
                        of the document"
        }
      }
    },
    "required": %w[title summary tags]
  }
}.freeze
```

Ý tưởng chính đằng sau việc sử dụng công cụ là để cho LLM khả năng động lựa chọn và thực thi các công cụ phù hợp dựa trên đầu vào của người dùng hoặc nhiệm vụ cần thực hiện. Thay vì chỉ dựa vào kiến thức được huấn luyện sẵn của mô hình, việc sử dụng công cụ cho phép LLM tận dụng các tài nguyên bên ngoài để tạo ra các phản hồi chính xác, phù hợp và khả thi hơn. Việc sử dụng công cụ làm cho các kỹ thuật như RAG (Sinh

nội dung tăng cường bằng truy xuất) dễ thực hiện hơn nhiều so với cách thông thường.

> Lưu ý rằng trừ khi có quy định khác, cuốn sách này giả định mô hình AI của bạn không có quyền truy cập vào bất kỳ công cụ tích hợp sẵn nào phía máy chủ. Bất kỳ công cụ nào bạn muốn cung cấp cho AI đều phải được bạn khai báo rõ ràng trong mỗi yêu cầu API, cùng với các quy định về việc điều phối thực thi nếu và khi AI của bạn cho biết nó muốn sử dụng công cụ đó trong phản hồi của mình.

Tiềm năng của việc Sử dụng Công cụ

Việc sử dụng công cụ mở ra nhiều khả năng cho các ứng dụng dựa trên AI. Dưới đây là một số ví dụ về những gì có thể đạt được với việc sử dụng công cụ:

1. **Chatbot và Trợ lý ảo**: Bằng cách kết nối LLM với các công cụ bên ngoài, chatbot và trợ lý ảo có thể thực hiện các tác vụ phức tạp hơn, như truy xuất thông tin từ cơ sở dữ liệu, thực thi các lệnh gọi API, hoặc tương tác với các hệ thống khác. Ví dụ, một chatbot có thể sử dụng công cụ CRM để thay đổi trạng thái của một thương vụ dựa trên yêu cầu của người dùng.
2. **Phân tích Dữ liệu và Thông tin chi tiết**: LLM có thể được kết nối với các công cụ hoặc thư viện phân tích dữ liệu để thực hiện các tác vụ xử lý dữ liệu nâng cao. Điều này cho phép các ứng dụng tạo ra thông tin chi tiết, thực hiện phân tích so sánh, hoặc đưa ra các khuyến nghị dựa trên dữ liệu theo các truy vấn của người dùng.
3. **Tìm kiếm và Truy xuất Thông tin**: Việc sử dụng công cụ cho phép LLM tương tác với các công cụ tìm kiếm, cơ sở dữ liệu vector, hoặc các hệ thống truy xuất thông tin khác. Bằng cách chuyển đổi các truy vấn của người dùng thành các truy vấn tìm kiếm, LLM có thể truy xuất thông tin liên quan từ nhiều nguồn và cung cấp câu trả lời toàn diện cho câu hỏi của người dùng.

4. **Tích hợp với Dịch vụ Bên ngoài**: Việc sử dụng công cụ cho phép tích hợp liền mạch giữa các ứng dụng dựa trên AI và các dịch vụ hoặc API bên ngoài. Ví dụ, một LLM có thể tương tác với API thời tiết để cung cấp cập nhật thời tiết theo thời gian thực hoặc API dịch thuật để tạo ra các phản hồi đa ngôn ngữ.

Quy trình Sử dụng Công cụ

Quy trình sử dụng công cụ thường bao gồm bốn bước chính:

1. Bao gồm các định nghĩa hàm trong ngữ cảnh yêu cầu của bạn
2. Lựa chọn công cụ động (hoặc rõ ràng)
3. Thực thi (các) hàm
4. Tiếp tục tùy chọn của prompt ban đầu

Hãy xem xét chi tiết từng bước này.

Bao gồm các định nghĩa hàm trong ngữ cảnh yêu cầu của bạn

AI biết những công cụ nào nó có thể sử dụng vì bạn cung cấp cho nó một danh sách như một phần của yêu cầu hoàn thành (thường được định nghĩa dưới dạng các hàm sử dụng một biến thể của lược đồ JSON).

Cú pháp chính xác của định nghĩa công cụ phụ thuộc vào từng mô hình cụ thể.

Đây là cách bạn định nghĩa một hàm `get_weather` trong Claude 3:

```
{
    "name": "get_weather",
    "description": "Get the current weather in a given location",
    "input_schema": {
        "type": "object",
        "properties": {
            "location": {
                "type": "string",
                "description": "The city and state, e.g. San Francisco, CA"
            },
            "unit": {
                "type": "string",
                "enum": ["celsius", "fahrenheit"],
                "description": "The unit of temperature"
        }
    },
    "required": ["location"]
    }
}
```

Và đây là cách bạn sẽ định nghĩa cùng một hàm cho GPT-4, bằng cách truyền nó như một giá trị của tham số `tools`:

```
{
    "name": "get_current_weather",
    "description": "Get the current weather in a given location",
    "parameters": {
        "type": "object",
        "properties": {
            "location": {
                "type": "string",
                "description": "The city and state, e.g. San Francisco, CA",
            },
            "unit": {
                "type": "string",
                "enum": ["celsius", "fahrenheit"],
                "description": "The unit of temperature"
            },
        },
        "required": ["location"],
```

```
    },
}
```

> Gần như giống nhau, chỉ khác một chút mà không có lý do rõ ràng! Thật khó chịu.

Định nghĩa hàm xác định tên, mô tả và tham số đầu vào. Tham số đầu vào có thể được định nghĩa chi tiết hơn bằng cách sử dụng các thuộc tính như kiểu liệt kê để giới hạn các giá trị được chấp nhận, và xác định liệu một tham số có bắt buộc hay không.

Ngoài các định nghĩa hàm thực tế, bạn cũng có thể bao gồm các hướng dẫn hoặc ngữ cảnh về lý do và cách sử dụng hàm trong chỉ thị hệ thống.

Ví dụ, công cụ Tìm kiếm Web của tôi trong Olympia bao gồm chỉ thị hệ thống này, nhắc nhở AI rằng nó có sẵn các công cụ đã đề cập:

```
The `google_search` and `realtime_search` functions let you do research
on behalf of the user. In contrast to Google, realtime search is powered
by Perplexity and provides real-time information to curated current events
databases and news sources. Make sure to include URLs in your response so
user can do followup research.
```

Việc cung cấp mô tả chi tiết được xem là yếu tố quan trọng nhất trong hiệu suất công cụ. Mô tả của bạn cần giải thích mọi chi tiết về công cụ, bao gồm:

- Công cụ làm được những gì
- Khi nào nên sử dụng (và khi nào không nên)
- Ý nghĩa của từng tham số và cách nó ảnh hưởng đến hành vi của công cụ
- Bất kỳ điều cần lưu ý hoặc giới hạn quan trọng nào áp dụng cho việc triển khai công cụ

Càng cung cấp nhiều ngữ cảnh cho AI về công cụ của bạn, nó càng giỏi trong việc quyết định khi nào và làm thế nào để sử dụng chúng. Ví dụ, Anthropic khuyến nghị mỗi mô

tả công cụ nên có ít nhất 3-4 câu đối với dòng Claude 3 của họ, nhiều hơn nếu công cụ phức tạp.

Có thể không trực quan lắm, nhưng mô tả được coi là quan trọng hơn cả ví dụ. Mặc dù bạn có thể bao gồm các ví dụ về cách sử dụng công cụ trong phần mô tả hoặc trong prompt đi kèm, điều này ít quan trọng hơn việc có một lời giải thích rõ ràng và toàn diện về mục đích và tham số của công cụ. Chỉ thêm ví dụ sau khi bạn đã hoàn thiện đầy đủ phần mô tả.

Dưới đây là một ví dụ về đặc tả hàm API kiểu Stripe:

```
{
  "name": "createPayment",
  "description": "Create a new payment request",
  "parameters": {
    "type": "object",
    "properties": {
      "transaction_amount": {
        "type": "number",
        "description": "The amount to be paid"
      },
      "description": {
        "type": "string",
        "description": "A brief description of the payment"
      },
      "payment_method_id": {
        "type": "string",
        "description": "The payment method to be used"
      },
      "payer": {
        "type": "object",
        "description": "Information about the payer, including their name,
                        email, and identification number",
        "properties": {
          "name": {
            "type": "string",
            "description": "The payer's name"
        },
        "email": {
          "type": "string",
```

```
            "description": "The payer's email address"
          },
          "identification": {
            "type": "object",
            "description": "The payer's identification number",
            "properties": {
              "type": {
                "type": "string",
                "description": "Identification document (e.g. CPF, CNPJ)"
              },
              "number": {
                "type": "string",
                "description": "The identification number"
              }
            },
            "required": [ "type", "number" ]
          }
        },
        "required": [ "name", "email", "identification" ]
      }
    }
}
```

Trong thực tế, một số mô hình gặp khó khăn khi xử lý các đặc tả hàm lồng nhau và xử lý các kiểu dữ liệu đầu ra phức tạp như mảng, từ điển, v.v. Nhưng về mặt lý thuyết, bạn có thể cung cấp các đặc tả JSON Schema với độ sâu tùy ý!

Lựa Chọn Công Cụ Động

Khi bạn thực thi một chat completion có bao gồm các định nghĩa công cụ, LLM sẽ tự động chọn (các) công cụ phù hợp nhất để sử dụng và tạo ra các tham số đầu vào cần thiết cho mỗi công cụ.

Trong thực tế, khả năng của AI trong việc gọi *chính xác* đúng hàm và tuân theo *chính xác* đặc tả của bạn cho các đầu vào là không ổn định. Việc giảm tham số temperature

xuống 0.0 giúp cải thiện nhiều, nhưng theo kinh nghiệm của tôi, bạn vẫn sẽ thỉnh thoảng gặp lỗi. Những lỗi đó bao gồm tên hàm được tạo ra không có thực, tham số đầu vào bị đặt sai tên hoặc đơn giản là bị thiếu. Các tham số được truyền dưới dạng JSON, điều này có nghĩa đôi khi bạn sẽ thấy các lỗi do JSON bị cắt ngắn, trích dẫn sai hoặc bị hỏng theo cách khác.

Các mẫu Tự Sửa Lỗi Dữ Liệu có thể giúp tự động sửa chữa các lệnh gọi hàm bị lỗi do lỗi cú pháp.

Lựa Chọn Công Cụ Bắt Buộc (hay còn gọi là Tường Minh)

Một số mô hình cho phép bạn bắt buộc gọi một hàm cụ thể, như một tham số trong yêu cầu. Nếu không, việc có gọi hàm hay không hoàn toàn phụ thuộc vào quyết định của AI.

Khả năng bắt buộc gọi hàm rất quan trọng trong một số tình huống khi bạn muốn đảm bảo một công cụ hoặc hàm cụ thể được thực thi, bất kể quy trình lựa chọn động của AI. Có một số lý do tại sao khả năng này quan trọng:

1. **Kiểm Soát Tường Minh**: Bạn có thể đang sử dụng AI như một *Thành Phần Rời Rạc* hoặc trong một quy trình được định nghĩa trước đòi hỏi việc thực thi một hàm cụ thể tại một thời điểm cụ thể. Bằng cách bắt buộc gọi hàm, bạn có thể đảm bảo rằng hàm mong muốn được gọi thay vì phải yêu cầu AI một cách lịch sự để thực hiện điều đó.
2. **Gỡ Lỗi và Kiểm Thử**: Khi phát triển và kiểm thử các ứng dụng dựa trên AI, khả năng bắt buộc gọi hàm rất có giá trị cho mục đích gỡ lỗi. Bằng cách kích hoạt các hàm cụ thể một cách tường minh, bạn có thể cô lập và kiểm thử từng thành phần của ứng dụng. Điều này cho phép bạn xác minh tính đúng đắn của việc thực thi hàm, xác thực các tham số đầu vào và đảm bảo rằng kết quả trả về là đúng như mong đợi.

3. **Xử Lý Các Trường Hợp Đặc Biệt**: Có thể có những trường hợp đặc biệt hoặc ngoại lệ mà quy trình lựa chọn động của AI có thể không chọn thực thi một hàm mà đáng lẽ nên thực thi, và bạn biết điều đó dựa trên các quy trình bên ngoài. Trong những trường hợp như vậy, việc có khả năng bắt buộc gọi hàm cho phép bạn xử lý những tình huống này một cách tường minh. Định nghĩa các quy tắc hoặc điều kiện trong logic ứng dụng của bạn để xác định khi nào cần ghi đè quyết định của AI.
4. **Tính Nhất Quán và Khả Năng Tái Tạo**: Nếu bạn có một chuỗi hàm cụ thể cần được thực thi theo một thứ tự nhất định, việc bắt buộc gọi hàm đảm bảo rằng cùng một chuỗi đó sẽ được tuân theo mỗi lần. Điều này đặc biệt quan trọng trong các ứng dụng mà tính nhất quán và hành vi có thể dự đoán là yếu tố then chốt, chẳng hạn như trong các hệ thống tài chính hoặc mô phỏng khoa học.
5. **Tối Ưu Hóa Hiệu Suất**: Trong một số trường hợp, việc bắt buộc gọi hàm có thể dẫn đến tối ưu hóa hiệu suất. Nếu bạn biết rằng một hàm cụ thể là cần thiết cho một nhiệm vụ cụ thể và quy trình lựa chọn động của AI có thể tạo ra chi phí không cần thiết, bạn có thể bỏ qua quy trình lựa chọn và gọi trực tiếp hàm cần thiết. Điều này có thể giúp giảm độ trễ và cải thiện hiệu quả tổng thể của ứng dụng của bạn.

Tóm lại, khả năng bắt buộc gọi hàm trong các ứng dụng dựa trên AI cung cấp khả năng kiểm soát tường minh, hỗ trợ gỡ lỗi và kiểm thử, xử lý các trường hợp đặc biệt, đảm bảo tính nhất quán và khả năng tái tạo. Đây là một công cụ mạnh mẽ trong kho vũ khí của bạn, nhưng chúng ta cần thảo luận thêm một khía cạnh của tính năng quan trọng này.

Trong nhiều trường hợp ra quyết định, chúng ta luôn muốn mô hình thực hiện một lệnh gọi hàm và có thể không bao giờ muốn mô hình trả lời chỉ bằng kiến thức nội tại của nó. Ví dụ, nếu bạn đang định tuyến giữa nhiều mô hình chuyên biệt cho các nhiệm vụ khác nhau (đầu vào đa ngôn ngữ, toán học, v.v.), bạn có thể sử dụng mô hình gọi hàm để ủy thác yêu cầu cho một trong các mô hình hỗ trợ và không bao giờ trả lời độc lập.

Tham Số Lựa Chọn Công Cụ

GPT-4 và các mô hình ngôn ngữ khác hỗ trợ gọi hàm cho phép bạn sử dụng tham số `tool_choice` để kiểm soát việc có yêu cầu sử dụng công cụ như một phần của completion hay không. Tham số này có ba giá trị có thể:

- `auto` cho AI toàn quyền quyết định việc sử dụng công cụ hay đơn giản là trả lời
- `required` cho AI biết rằng nó *phải* gọi một công cụ *thay vì* trả lời, nhưng để việc lựa chọn công cụ cho AI.
- Lựa chọn thứ ba là đặt tham số của `name_of_function` mà bạn muốn bắt buộc. Chúng ta sẽ tìm hiểu thêm về điều này trong phần tiếp theo.

Lưu ý rằng nếu bạn đặt tool choice thành `required`, mô hình sẽ buộc phải chọn hàm phù hợp nhất để gọi từ những hàm được cung cấp, ngay cả khi không có hàm nào thực sự phù hợp với yêu cầu. Tại thời điểm xuất bản, tôi chưa biết có mô hình nào sẽ trả về phản hồi `tool_calls` rỗng, hoặc sử dụng cách nào khác để cho bạn biết rằng nó không tìm thấy hàm phù hợp để gọi.

Bắt Buộc Gọi Hàm Để Nhận Dữ Liệu Có Cấu Trúc

Khả năng bắt buộc gọi hàm cho bạn cách để buộc lấy dữ liệu có cấu trúc từ một chat completion thay vì phải tự trích xuất nó từ phản hồi dạng văn bản thuần túy.

Tại sao việc bắt buộc gọi hàm để nhận dữ liệu có cấu trúc lại quan trọng? Nói đơn giản, bởi vì việc trích xuất dữ liệu có cấu trúc từ đầu ra của LLM rất khó khăn. Bạn có thể làm cho cuộc sống dễ dàng hơn một chút bằng cách yêu cầu dữ liệu ở định dạng XML, nhưng sau đó bạn phải phân tích XML. Và bạn sẽ làm gì khi XML đó bị thiếu vì AI của bạn trả lời: “Tôi xin lỗi, nhưng tôi không thể tạo dữ liệu bạn yêu cầu

> vì blah blah blah...”

Khi sử dụng công cụ theo cách này:

- Bạn nên định nghĩa một công cụ duy nhất trong yêu cầu của mình
- Nhớ bắt buộc sử dụng hàm của nó bằng tham số `tool_choice`
- Nhớ rằng mô hình sẽ chuyển đầu vào cho công cụ, vì vậy tên của công cụ và mô tả phải từ góc nhìn của mô hình, không phải của bạn.

Điểm cuối cùng này cần một ví dụ để làm rõ. Giả sử bạn đang yêu cầu AI thực hiện phân tích cảm xúc trên văn bản của người dùng. Tên của hàm sẽ không phải là `analyze_sentiment`, mà thay vào đó sẽ là một cái gì đó như `save_sentiment_analysis`. AI là người thực hiện phân tích cảm xúc, *không phải công cụ*. Tất cả những gì công cụ đang làm (từ góc nhìn của AI) là lưu kết quả của phân tích.

Dưới đây là một ví dụ về việc sử dụng Claude 3 để ghi lại bản tóm tắt của một hình ảnh thành JSON có cấu trúc tốt, lần này từ dòng lệnh sử dụng `curl`:

```
curl https://api.anthropic.com/v1/messages \
     --header "content-type: application/json" \
     --header "x-api-key: $ANTHROPIC_API_KEY" \
     --header "anthropic-version: 2023-06-01" \
     --header "anthropic-beta: tools-2024-04-04" \
     --data \
'{
    "model": "claude-3-sonnet-20240229",
    "max_tokens": 1024,
    "tools": [{
        "name": "record_summary",
        "description": "Record summary of image into well-structured JSON.",
        "input_schema": {
            "type": "object",
            "properties": {
                "key_colors": {
```

```
                    "type": "array",
                    "items": {
                        "type": "object",
                        "properties": {
                            "r": {
                                "type": "number",
                                "description": "red value [0.0, 1.0]"
                            },
                            "g": {
                                "type": "number",
                                "description": "green value [0.0, 1.0]"
                            },
                            "b": {
                                "type": "number",
                                "description": "blue value [0.0, 1.0]"
                            },
                            "name": {
                                "type": "string",
                                "description": "Human-readable color name
                                                in snake_case, e.g.
                                                \"olive_green\"or
                                                \"turquoise\""
                            }
                        },
                        "required": [ "r", "g", "b", "name" ]
                    },
                    "description": "Key colors in the image. Four or less."
                },
                "description": {
                    "type": "string",
                    "description": "Image description. 1-2 sentences max."
                },
                "estimated_year": {
                    "type": "integer",
                    "description": "Estimated year that the image was taken,
                                    if is it a photo. Only set this if the
                                    image appears to be non-fictional.
                                    Rough estimates are okay!"
                }
            },
            "required": [ "key_colors", "description" ]
        }
```

```
    }],
    "messages": [
        {
            "role": "user",
            "content": [
                {
                    "type": "image",
                    "source": {
                        "type": "base64",
                        "media_type": "'$IMAGE_MEDIA_TYPE'",
                        "data": "'$IMAGE_BASE64'"
                    }
                },
                {
                    "type": "text",
                    "text": "Use `record_summary` to describe this image."
                }
            ]
        }
    ]
}'
```

Trong ví dụ được cung cấp, chúng ta đang sử dụng mô hình Claude 3 từ Anthropic để tạo ra một bản tóm tắt JSON có cấu trúc cho một hình ảnh. Đây là cách nó hoạt động:

1. Chúng ta định nghĩa một công cụ duy nhất có tên `record_summary` trong mảng `tools` của payload yêu cầu. Công cụ này có nhiệm vụ ghi lại bản tóm tắt của hình ảnh thành JSON có cấu trúc rõ ràng.
2. Công cụ `record_summary` có một `input_schema` xác định cấu trúc mong đợi của đầu ra JSON. Nó định nghĩa ba thuộc tính:

    - `key_colors`: Một mảng các đối tượng đại diện cho các màu chính trong hình ảnh. Mỗi đối tượng màu có các thuộc tính cho giá trị đỏ, xanh lá, và xanh dương (từ 0.0 đến 1.0) và một tên màu có thể đọc được ở định dạng snake_case.
    - `description`: Một thuộc tính chuỗi để mô tả ngắn gọn về hình ảnh, giới hạn trong 1-2 câu.

- `estimated_year`: Một thuộc tính số nguyên tùy chọn cho năm ước tính khi hình ảnh được chụp, nếu nó có vẻ là một bức ảnh phi hư cấu.

3. Trong mảng `messages`, chúng ta cung cấp dữ liệu hình ảnh dưới dạng chuỗi mã hóa Base64 cùng với kiểu phương tiện. Điều này cho phép mô hình xử lý hình ảnh như một phần của đầu vào.
4. Chúng ta cũng nhắc Claude sử dụng công cụ `record_summary` để mô tả hình ảnh.
5. Khi yêu cầu được gửi đến mô hình Claude 3, nó phân tích hình ảnh và tạo ra một bản tóm tắt JSON dựa trên `input_schema` đã chỉ định. Mô hình trích xuất các màu chính, cung cấp mô tả ngắn gọn, và ước tính năm chụp hình ảnh (nếu có thể áp dụng).
6. Bản tóm tắt JSON được tạo ra được truyền làm tham số cho công cụ `record_summary`, cung cấp một biểu diễn có cấu trúc về các đặc điểm chính của hình ảnh.

Bằng cách sử dụng công cụ `record_summary` với một `input_schema` được định nghĩa rõ ràng, chúng ta có thể có được một bản tóm tắt JSON có cấu trúc của hình ảnh mà không cần dựa vào việc trích xuất văn bản thuần túy. Cách tiếp cận này đảm bảo rằng đầu ra tuân theo một định dạng nhất quán và có thể dễ dàng được phân tích và xử lý bởi các thành phần xuôi dòng của ứng dụng.

Khả năng bắt buộc gọi hàm và chỉ định cấu trúc đầu ra mong đợi là một tính năng mạnh mẽ của việc sử dụng công cụ trong các ứng dụng dựa trên AI. Nó cho phép các nhà phát triển có nhiều kiểm soát hơn đối với đầu ra được tạo ra và đơn giản hóa việc tích hợp dữ liệu được tạo bởi AI vào quy trình làm việc của ứng dụng của họ.

Thực Thi (Các) Hàm

Bạn đã định nghĩa các hàm và nhắc AI, và AI đã quyết định rằng nó nên gọi một trong các hàm của bạn. Bây giờ đến lúc mã ứng dụng của bạn hoặc thư viện, nếu bạn đang sử

dụng một gem Ruby như `raix-rails` để điều phối lời gọi hàm và các tham số của nó đến triển khai tương ứng *trong mã ứng dụng của bạn.*

Mã ứng dụng của bạn quyết định phải làm gì với kết quả của việc thực thi hàm. Có thể việc cần làm chỉ liên quan đến một dòng mã trong lambda, hoặc có thể liên quan đến việc gọi một API bên ngoài. Có thể liên quan đến việc gọi một thành phần AI khác, hoặc có thể liên quan đến hàng trăm hoặc thậm chí hàng nghìn dòng mã trong phần còn lại của hệ thống của bạn. Điều này hoàn toàn phụ thuộc vào bạn.

Đôi khi lời gọi hàm là kết thúc của hoạt động, nhưng nếu kết quả đại diện cho thông tin trong một chuỗi suy luận cần được AI tiếp tục, thì mã ứng dụng của bạn cần chèn kết quả thực thi vào bản ghi trò chuyện và để AI tiếp tục xử lý.

Ví dụ, đây là một khai báo hàm Raix được sử dụng bởi `AccountManager` của Olympia để giao tiếp với khách hàng của chúng tôi như một phần của Điều phối Quy trình Làm việc Thông minh cho dịch vụ khách hàng.

```
class AccountManager
  include Raix::ChatCompletion
  include Raix::FunctionDispatch

  # lots of other functions...

  function :notify_account_owner,
    "Don't share UUID. Mention dollars if subscription changed",
    message: { type: "string" } do |arguments|
      account.owner.freeform_notify(
        subject: "Account Change Notification",
        message: arguments[:message]
      )
      "Notified account owner"
    end
```

Có thể không dễ hiểu ngay được chuyện gì đang xảy ra ở đây, vì vậy tôi sẽ giải thích chi tiết.

1. Lớp `AccountManager` định nghĩa nhiều hàm liên quan đến quản lý tài khoản. Nó

có thể thay đổi gói dịch vụ của bạn, thêm và xóa thành viên nhóm, cùng nhiều chức năng khác.

2. Các chỉ dẫn cấp cao nhất cho `AccountManager` biết rằng nó nên thông báo cho chủ tài khoản về kết quả của yêu cầu thay đổi tài khoản, sử dụng hàm `notify_account_owner`.
3. Định nghĩa ngắn gọn của hàm bao gồm:

- tên
- mô tả
- tham số `message: { type: "string" }`
- một khối lệnh để thực thi khi hàm được gọi

Sau khi cập nhật bản ghi với kết quả của khối hàm, phương thức `chat_completion` được gọi lại. Phương thức này chịu trách nhiệm gửi bản ghi cuộc hội thoại đã cập nhật trở lại cho mô hình AI để xử lý thêm. Chúng ta gọi quá trình này là *vòng lặp hội thoại*.

Khi mô hình AI nhận được yêu cầu hoàn thành cuộc trò chuyện mới với bản ghi đã cập nhật, nó có quyền truy cập vào kết quả của hàm đã thực thi trước đó. Nó có thể phân tích những kết quả này, kết hợp chúng vào quá trình ra quyết định, và tạo ra phản hồi hoặc hành động tiếp theo dựa trên ngữ cảnh tích lũy của cuộc hội thoại. Nó có thể chọn thực thi thêm các hàm dựa trên ngữ cảnh đã cập nhật, hoặc có thể tạo ra phản hồi cuối cùng cho lệnh ban đầu nếu nó xác định rằng không cần thêm lệnh gọi hàm nào nữa.

Tiếp tục tùy chọn của Lệnh Ban đầu

Khi bạn gửi kết quả công cụ trở lại cho LLM và tiếp tục xử lý lệnh ban đầu, AI sẽ sử dụng những kết quả đó để gọi thêm các hàm hoặc tạo ra phản hồi văn bản cuối cùng.

Một số mô hình như Command-R của Cohere có thể trích dẫn cụ thể các công cụ mà họ đã sử dụng trong phản hồi của mình, cung cấp thêm tính minh bạch và khả năng truy xuất nguồn gốc.

Tùy thuộc vào mô hình đang sử dụng, kết quả của lệnh gọi hàm sẽ nằm trong các tin nhắn bản ghi có vai trò đặc biệt riêng hoặc được phản ánh trong một cú pháp khác. Nhưng phần quan trọng là dữ liệu đó phải có trong bản ghi, để AI có thể xem xét khi quyết định việc cần làm tiếp theo.

Một lỗi phổ biến (và có thể tốn kém) là quên thêm kết quả của hàm vào bản ghi trước khi tiếp tục cuộc trò chuyện. Kết quả là, AI sẽ được nhắc theo cách cơ bản giống như trước khi nó gọi hàm lần đầu tiên. Nói cách khác, đối với AI, nó chưa gọi hàm. Vì vậy nó gọi lại. Và lại nữa. Và lại nữa, mãi mãi cho đến khi bạn ngắt nó. Hy vọng ngữ cảnh của bạn không quá lớn, và mô hình của bạn không quá đắt!

Các Phương Pháp Tốt Nhất cho Việc Sử dụng Công cụ

Để tận dụng tối đa việc sử dụng công cụ, hãy xem xét các phương pháp tốt nhất sau đây.

Định Nghĩa Mô tả

Cung cấp tên và mô tả rõ ràng, chi tiết cho mỗi công cụ và các tham số đầu vào của nó. Điều này giúp LLM hiểu rõ hơn về mục đích và khả năng của từng công cụ.

> Từ kinh nghiệm của tôi có thể nói rằng quan điểm phổ biến cho rằng "đặt tên là khó" cũng áp dụng ở đây; tôi đã thấy các kết quả khác biệt đáng kể từ các LLM chỉ bằng cách thay đổi tên của các hàm hoặc cách diễn đạt của các mô tả. Đôi khi việc bỏ đi các mô tả thậm chí còn *cải thiện* hiệu suất.

Xử lý Kết quả Công cụ

Khi chuyển kết quả công cụ trở lại cho LLM, hãy đảm bảo chúng được cấu trúc tốt và toàn diện. Sử dụng các khóa và giá trị có ý nghĩa để biểu diễn đầu ra của mỗi công cụ. Thử nghiệm với các định dạng khác nhau và xem cái nào hoạt động tốt nhất, từ JSON đến văn bản thuần túy.

Trình diễn giải kết quả giải quyết thách thức này bằng cách sử dụng AI để phân tích kết quả và cung cấp những giải thích, tóm tắt hoặc điểm chính thân thiện với người dùng.

Xử lý Lỗi

Triển khai các cơ chế xử lý lỗi mạnh mẽ để xử lý các trường hợp khi LLM có thể tạo ra các tham số đầu vào không hợp lệ hoặc không được hỗ trợ cho các lệnh gọi công cụ. Xử lý một cách nhẹ nhàng và khôi phục từ bất kỳ lỗi nào có thể xảy ra trong quá trình thực thi công cụ.

Một đặc điểm cực kỳ tốt của AI là nó hiểu được các thông báo lỗi! Điều đó có nghĩa là nếu bạn đang làm việc với tư duy nhanh và đơn giản, bạn có thể đơn giản là bắt bất kỳ ngoại lệ nào được tạo ra trong quá trình triển khai một công cụ, và chuyển nó trở lại cho AI để nó biết chuyện gì đã xảy ra!

Ví dụ, đây là phiên bản rút gọn của việc triển khai tìm kiếm google trong Olympia:

```
  def google_search(conversation, params)
    conversation.update_cstatus("Searching Google...")
    query = params[:query]
    search = GoogleSearch.new(query).get_hash

    conversation.update_cstatus("Summarizing results...")
    SummarizeKnowledgeGraph.new.perform(conversation, search.to_json)
  rescue StandardError => e
    Honeybadger.notify(e)
    { error: e.message }.inspect
  end
```

Tìm kiếm Google trong Olympia là một quy trình hai bước. Đầu tiên bạn thực hiện tìm kiếm, sau đó tóm tắt kết quả. Nếu có bất kỳ lỗi nào xảy ra, dù là lỗi gì, thông báo lỗi sẽ được đóng gói và gửi trả về cho AI. Kỹ thuật này là nền tảng của hầu hết các mẫu *Xử lý lỗi thông minh*

Ví dụ, giả sử lệnh gọi API `GoogleSearch` bị lỗi do ngoại lệ 503 Dịch vụ không khả dụng. Lỗi này được đưa lên cấp cao nhất của khối rescue, và mô tả về lỗi được gửi trả về cho AI như là kết quả của lệnh gọi hàm. Thay vì chỉ hiển thị màn hình trống hoặc lỗi kỹ thuật cho người dùng, AI sẽ nói điều gì đó như "Tôi xin lỗi, nhưng hiện tại tôi không thể truy cập được khả năng Tìm kiếm Google. Tôi có thể thử lại sau nếu bạn muốn."

Điều này có vẻ chỉ là một thủ thuật thông minh, nhưng hãy xem xét một loại lỗi khác, khi AI đang gọi một API bên ngoài và có quyền kiểm soát trực tiếp các tham số truyền vào API đó. Có thể nó đã mắc lỗi trong cách tạo ra các tham số đó? Miễn là thông báo lỗi từ API bên ngoài đủ chi tiết, việc trả về thông báo lỗi cho AI gọi có nghĩa là nó có thể xem xét lại các tham số đó và thử lại. Một cách tự động. Bất kể lỗi là gì.

Bây giờ hãy nghĩ xem sẽ cần những gì để sao chép kiểu xử lý lỗi mạnh mẽ đó trong mã *thông thường*. Điều đó gần như là không thể.

Tinh chỉnh lặp đi lặp lại

Nếu LLM không đề xuất các công cụ phù hợp hoặc tạo ra các phản hồi chưa tối ưu, hãy lặp lại việc định nghĩa công cụ, mô tả và các tham số đầu vào. Liên tục tinh chỉnh và

cải thiện thiết lập công cụ dựa trên hành vi quan sát được và kết quả mong muốn.

1. Bắt đầu với các định nghĩa công cụ đơn giản: Bắt đầu bằng cách định nghĩa các công cụ với tên, mô tả và tham số đầu vào rõ ràng và ngắn gọn. Tránh làm phức tạp hóa thiết lập công cụ ban đầu và tập trung vào chức năng cốt lõi. Ví dụ, nếu bạn muốn lưu kết quả phân tích cảm xúc, hãy bắt đầu với một định nghĩa cơ bản như:

```
{
  "name": "save_sentiment_score",
  "description": "Analyze user-provided text and generate sentiment score",
  "parameters": {
    "type": "object",
    "properties": {
      "score": {
        "type": "float",
        "description": "sentiment score from -1 (negative) to 1 (positive)"
      }
    },
    "required": ["score"]
  }
}
```

2. Kiểm thử và quan sát: Sau khi đã có các định nghĩa công cụ ban đầu, hãy kiểm thử chúng với các lệnh đầu vào khác nhau và quan sát cách mà LLM tương tác với công cụ. Chú ý đến chất lượng và độ phù hợp của các phản hồi được tạo ra. Nếu LLM đang tạo ra các phản hồi chưa tối ưu, đã đến lúc tinh chỉnh các định nghĩa công cụ.
3. Tinh chỉnh mô tả: Nếu LLM đang hiểu sai mục đích của một công cụ, hãy thử tinh chỉnh mô tả của công cụ đó. Cung cấp thêm ngữ cảnh, ví dụ, hoặc giải thích để hướng dẫn LLM sử dụng công cụ hiệu quả. Ví dụ, bạn có thể cập nhật mô tả công cụ phân tích cảm xúc để đề cập cụ thể hơn về *sắc thái cảm xúc* của đoạn văn bản đang được phân tích:

```
{
  "name": "save_sentiment_score",
  "description": "Determine the overall emotional tone of a piece of text,
   such as customer reviews, social media posts, or feedback comments.",
  ...
}
```

4. Điều chỉnh tham số đầu vào: Nếu mô hình ngôn ngữ lớn (LLM) đang tạo ra các tham số đầu vào không hợp lệ hoặc không liên quan cho công cụ, hãy cân nhắc điều chỉnh các định nghĩa tham số. Thêm các ràng buộc cụ thể hơn, quy tắc xác thực, hoặc các ví dụ để làm rõ định dạng đầu vào mong muốn.
5. Lặp lại dựa trên phản hồi: Liên tục theo dõi hiệu suất của các công cụ của bạn và thu thập phản hồi từ người dùng hoặc các bên liên quan. Sử dụng những phản hồi này để xác định các lĩnh vực cần cải thiện và thực hiện các điều chỉnh lặp đi lặp lại cho định nghĩa công cụ. Ví dụ, nếu người dùng báo cáo rằng việc phân tích không xử lý tốt sự mỉa mai, bạn có thể thêm một ghi chú trong phần mô tả:

```
{
  "name": "save_sentiment_score",
  "description": "Analyze the sentiment of a given text and return a sentiment
   score between -1 (negative) and 1 (positive). Note: Sarcasm should be
   considered negative.",
  ...
}
```

Bằng cách lặp đi lặp lại việc tinh chỉnh định nghĩa công cụ dựa trên hành vi quan sát được và phản hồi, bạn có thể dần dần cải thiện hiệu suất và hiệu quả của ứng dụng điều khiển bởi AI của mình. Hãy nhớ giữ cho các định nghĩa công cụ rõ ràng, ngắn gọn và tập trung vào nhiệm vụ cụ thể đang thực hiện. Thường xuyên kiểm tra và xác nhận các tương tác của công cụ để đảm bảo chúng phù hợp với kết quả mong muốn của bạn.

Kết hợp và Xâu chuỗi Công cụ

Một trong những khía cạnh mạnh mẽ nhất của việc sử dụng công cụ mà chúng ta mới chỉ đề cập sơ qua là khả năng kết hợp và xâu chuỗi nhiều công cụ với nhau để hoàn thành các tác vụ phức tạp. Bằng cách thiết kế cẩn thận định nghĩa công cụ và định dạng đầu vào/đầu ra của chúng, bạn có thể tạo ra các khối xây dựng có thể tái sử dụng và kết hợp theo nhiều cách khác nhau.

Hãy xem xét một ví dụ về việc xây dựng quy trình phân tích dữ liệu cho ứng dụng điều khiển bởi AI của bạn. Bạn có thể có các công cụ sau:

1. **`DataRetrieval`**: Một công cụ lấy dữ liệu từ cơ sở dữ liệu hoặc API dựa trên các tiêu chí được chỉ định.
2. **`DataProcessing`**: Một công cụ thực hiện các phép tính, chuyển đổi hoặc tổng hợp trên dữ liệu đã thu thập.
3. **`DataVisualization`**: Một công cụ trình bày dữ liệu đã xử lý theo định dạng thân thiện với người dùng, chẳng hạn như biểu đồ hoặc đồ thị.

Bằng cách xâu chuỗi các công cụ này lại với nhau, bạn có thể tạo ra một quy trình làm việc mạnh mẽ để truy xuất dữ liệu liên quan, xử lý nó và trình bày kết quả một cách có ý nghĩa. Đây là cách quy trình sử dụng công cụ có thể diễn ra:

1. LLM nhận được truy vấn của người dùng yêu cầu thông tin chi tiết về dữ liệu bán hàng cho một danh mục sản phẩm cụ thể.
2. LLM chọn công cụ `DataRetrieval` và tạo ra các tham số đầu vào thích hợp để lấy dữ liệu bán hàng liên quan từ cơ sở dữ liệu.
3. Dữ liệu thu thập được "chuyển" đến công cụ `DataProcessing`, tính toán các chỉ số như tổng doanh thu, giá bán trung bình và tỷ lệ tăng trưởng.
4. Sau đó, dữ liệu đã xử lý được chuyển cho công cụ `DataVisualization`, tạo ra biểu đồ hoặc đồ thị hấp dẫn về mặt hình ảnh để thể hiện thông tin chi tiết, trả về URL của biểu đồ cho LLM.

5. Cuối cùng, LLM tạo ra phản hồi được định dạng cho truy vấn của người dùng bằng markdown, kết hợp dữ liệu đã được trực quan hóa và cung cấp tóm tắt các phát hiện chính.

Bằng cách kết hợp các công cụ này lại với nhau, bạn có thể tạo ra một quy trình phân tích dữ liệu liền mạch có thể dễ dàng tích hợp vào ứng dụng của bạn. Điểm đẹp của phương pháp này là mỗi công cụ có thể được phát triển và kiểm tra độc lập, sau đó kết hợp theo những cách khác nhau để giải quyết các vấn đề khác nhau.

Để cho phép việc kết hợp và xâu chuỗi các công cụ một cách suôn sẻ, điều quan trọng là phải xác định rõ định dạng đầu vào và đầu ra cho mỗi công cụ.

Ví dụ, công cụ `DataRetrieval` có thể chấp nhận các tham số như chi tiết kết nối cơ sở dữ liệu, tên bảng và điều kiện truy vấn, và trả về tập kết quả dưới dạng đối tượng JSON có cấu trúc. Sau đó, công cụ `DataProcessing` có thể nhận đối tượng JSON này làm đầu vào và tạo ra một đối tượng JSON đã được chuyển đổi làm đầu ra. Bằng cách chuẩn hóa luồng dữ liệu giữa các công cụ, bạn có thể đảm bảo tính tương thích và khả năng tái sử dụng.

Khi thiết kế hệ sinh thái công cụ của bạn, hãy nghĩ về cách các công cụ khác nhau có thể được kết hợp để giải quyết các trường hợp sử dụng phổ biến trong ứng dụng của bạn. Hãy cân nhắc việc tạo ra các công cụ cấp cao bao gồm các quy trình làm việc hoặc logic nghiệp vụ phổ biến, giúp LLM dễ dàng lựa chọn và sử dụng chúng một cách hiệu quả.

Hãy nhớ rằng, sức mạnh của việc sử dụng công cụ nằm ở tính linh hoạt và tính module mà nó mang lại. Bằng cách chia nhỏ các tác vụ phức tạp thành các công cụ nhỏ hơn, có thể tái sử dụng, bạn có thể tạo ra một ứng dụng điều khiển bởi AI mạnh mẽ và linh hoạt có thể giải quyết nhiều thách thức khác nhau.

Hướng Phát triển Tương lai

Khi lĩnh vực phát triển ứng dụng điều khiển bởi AI phát triển, chúng ta có thể kỳ vọng những tiến bộ hơn nữa trong khả năng sử dụng công cụ. Một số hướng phát triển tiềm năng trong tương lai bao gồm:

1. **Sử dụng Công cụ Đa bước**: LLM có thể quyết định số lần cần sử dụng công cụ để tạo ra phản hồi thỏa đáng. Điều này có thể bao gồm nhiều vòng lựa chọn và thực thi công cụ dựa trên kết quả trung gian.
2. **Công cụ Định nghĩa Sẵn**: Các nền tảng AI có thể cung cấp một bộ công cụ được định nghĩa sẵn mà các nhà phát triển có thể tận dụng ngay lập tức, chẳng hạn như trình thông dịch Python, công cụ tìm kiếm web hoặc các hàm tiện ích phổ biến.
3. **Tích hợp Liền mạch**: Khi việc sử dụng công cụ trở nên phổ biến hơn, chúng ta có thể kỳ vọng sự tích hợp tốt hơn giữa các nền tảng AI và framework phát triển phổ biến, giúp các nhà phát triển dễ dàng đưa việc sử dụng công cụ vào ứng dụng của họ.

Sử dụng công cụ là một kỹ thuật mạnh mẽ cho phép các nhà phát triển khai thác toàn bộ tiềm năng của LLM trong các ứng dụng điều khiển bởi AI. Bằng cách kết nối LLM với các công cụ và tài nguyên bên ngoài, bạn có thể tạo ra các hệ thống năng động, thông minh và nhận thức về ngữ cảnh hơn, có thể thích ứng với nhu cầu của người dùng và cung cấp những hiểu biết và hành động có giá trị.

Mặc dù việc sử dụng công cụ mang lại nhiều khả năng to lớn, nhưng điều quan trọng là phải nhận thức được những thách thức và cân nhắc tiềm ẩn. Một khía cạnh quan trọng là quản lý độ phức tạp của tương tác công cụ và đảm bảo tính ổn định và độ tin cậy của toàn bộ hệ thống. Bạn cần xử lý các tình huống khi lời gọi công cụ có thể thất bại, trả về kết quả không mong đợi hoặc có ảnh hưởng đến hiệu suất. Ngoài ra, bạn nên cân nhắc

các biện pháp kiểm soát bảo mật và quyền truy cập để ngăn chặn việc sử dụng công cụ trái phép hoặc độc hại. Cơ chế xử lý lỗi, ghi nhật ký và giám sát thích hợp là rất quan trọng để duy trì tính toàn vẹn và hiệu suất của ứng dụng điều khiển bởi AI của bạn.

Khi bạn khám phá các khả năng của việc sử dụng công cụ trong các dự án của mình, hãy nhớ bắt đầu với những mục tiêu rõ ràng, thiết kế các định nghĩa công cụ có cấu trúc tốt, và tiếp tục cải tiến dựa trên phản hồi và kết quả. Với cách tiếp cận và tư duy đúng đắn, việc sử dụng công cụ có thể mở ra những cấp độ mới về đổi mới và giá trị trong các ứng dụng điều khiển bởi AI của bạn

Xử Lý Luồng

Truyền luồng dữ liệu qua HTTP, còn được gọi là sự kiện được gửi từ máy chủ (SSE), là một cơ chế trong đó máy chủ liên tục gửi dữ liệu đến máy khách khi dữ liệu có sẵn, mà không cần máy khách phải yêu cầu một cách rõ ràng. Vì phản hồi của AI được tạo ra theo từng phần, việc hiển thị đầu ra của AI khi nó đang được tạo ra sẽ mang lại trải nghiệm người dùng phản hồi nhanh hơn. Và thực tế, tất cả các API của nhà cung cấp AI mà tôi biết đều cung cấp phản hồi dạng luồng như một tùy chọn trong các điểm cuối hoàn thành của họ.

Lý do chương này xuất hiện ngay sau phần Sử Dụng Công Cụ là vì sức mạnh to lớn khi kết hợp việc sử dụng công cụ với phản hồi AI trực tiếp cho người dùng. Điều này cho phép tạo ra những trải nghiệm động và tương tác, nơi AI có thể xử lý dữ liệu đầu vào của người dùng, sử dụng các công cụ và hàm khác nhau theo ý muốn, và sau đó cung cấp phản hồi theo thời gian thực.

Để đạt được sự tương tác mượt mà này, bạn cần viết các trình xử lý luồng có khả năng điều phối các lời gọi hàm công cụ được AI kích hoạt cũng như đầu ra văn bản thuần túy đến người dùng cuối. Việc cần phải lặp lại sau khi xử lý một hàm công cụ tạo ra một thách thức thú vị cho công việc này.

Triển Khai ReplyStream

Để minh họa cách xử lý luồng có thể được triển khai, chương này sẽ đi sâu vào một phiên bản đơn giản hóa của lớp `ReplyStream` được sử dụng trong Olympia. Các thể hiện của lớp này có thể được truyền vào như tham số `stream` trong các thư viện máy khách AI như ruby-openai và openrouter

Dưới đây là cách tôi sử dụng `ReplyStream` trong `PromptSubscriber` của Olympia, thành phần lắng nghe thông qua Wisper để phát hiện việc tạo ra các tin nhắn mới của người dùng.

```
class PromptSubscriber
  include Raix::ChatCompletion
  include Raix::PromptDeclarations

  # many other declarations omitted...

  prompt text: -> { user_message.content },
         stream: -> { ReplyStream.new(self) },
         until: -> { bot_message.complete? }

  def message_created(message) # invoked by Wisper
    return unless message.role.user? && message.content?

    # rest of the implementation omitted...
```

Ngoài một tham chiếu `context` đến người đăng ký lệnh đã khởi tạo nó, lớp `ReplyStream` còn có các biến thực thể để lưu trữ bộ đệm của dữ liệu đã nhận, và các mảng để theo dõi tên hàm và đối số được gọi trong quá trình xử lý luồng.

```ruby
class ReplyStream
  attr_accessor :buffer, :f_name, :f_arguments, :context

  delegate :bot_message, :dispatch, to: :context

  def initialize(context)
    self.context = context
    self.buffer = []
    self.f_name = []
    self.f_arguments = []
  end

  def call(chunk, bytesize = nil)
    # ...
  end

  # ...
end
```

Phương thức `initialize` thiết lập trạng thái ban đầu của thực thể `ReplyStream`, khởi tạo bộ đệm, ngữ cảnh và các biến khác.

Phương thức `call` là điểm vào chính để xử lý dữ liệu truyền theo luồng. Nó nhận một chunk dữ liệu (được biểu diễn dưới dạng bảng băm) và một tham số tùy chọn `bytesize`, mà trong ví dụ này không được sử dụng. Bên trong phương thức này, lớp sử dụng đối sánh mẫu để xử lý các tình huống khác nhau dựa trên cấu trúc của khối dữ liệu nhận được.

Việc gọi `deep_symbolize_keys` trên chunk giúp cho việc đối sánh mẫu trở nên thanh lịch hơn, bằng cách cho phép chúng ta thao tác với các ký hiệu thay vì chuỗi.

```ruby
def call(chunk, _bytesize)
  case chunk.deep_symbolize_keys

  in { # match function name
    choices: [
      {
        delta: {
          tool_calls: [
            { index: index, function: {name: name} }
          ]
        }
      }
    ] }

    f_name[index] = name
```

Mẫu đầu tiên chúng ta cần khớp là một lệnh gọi công cụ cùng với tên hàm tương ứng của nó. Nếu phát hiện được một lệnh gọi, chúng ta sẽ đưa nó vào mảng `f_name`. Chúng ta lưu trữ tên các hàm trong một mảng có chỉ số, bởi vì mô hình có khả năng gọi hàm song song, gửi nhiều hơn một hàm để thực thi cùng một lúc.

> Gọi hàm song song là khả năng của một mô hình AI để thực hiện nhiều lệnh gọi hàm cùng lúc, cho phép các tác động và kết quả của những lệnh gọi hàm này được xử lý song song. Điều này đặc biệt hữu ích nếu các hàm mất nhiều thời gian thực thi, và giúp giảm số lần giao tiếp với API, từ đó có thể tiết kiệm đáng kể lượng token sử dụng.

Tiếp theo chúng ta cần khớp các đối số tương ứng với các lệnh gọi hàm.

```
  in { # match arguments
    choices: [
      {
        delta: {
          tool_calls: [
            {
              index: index, function: {arguments: argument }
            }
          ]
        }
      }
    ]}

    f_arguments[index] ||= "" # initialize if not already
    f_arguments[index] << argument
```

Tương tự như cách chúng ta xử lý tên hàm, chúng ta cất giữ các tham số trong một mảng có chỉ mục.

Tiếp theo, chúng ta tìm kiếm các thông điệp hiển thị cho người dùng thông thường, những thông điệp này sẽ được gửi từ máy chủ từng token một và được gán vào biến `new_content`. Chúng ta cũng cần theo dõi `finish_reason`. Nó sẽ là `nil` cho đến khi nhận được phần cuối cùng của chuỗi đầu ra.

```
  in {
    choices: [
      { delta: {content: new_content}, finish_reason: finish_reason }
    ]}

    # you could transmit every chunk to the user here...
    buffer << new_content.to_s

    if finish_reason.present?
      finalize
    elsif new_content.to_s.match?(/\n\n/)
      send_to_client # ...or buffer and transmit once per paragraph
    end
```

Quan trọng là, chúng ta thêm một biểu thức khớp mẫu để xử lý các thông báo lỗi được

gửi bởi nhà cung cấp mô hình AI. Trong môi trường phát triển cục bộ, chúng ta báo ngoại lệ, nhưng trong môi trường sản xuất, chúng ta ghi nhận lỗi và kết thúc xử lý.

```
  in { error: { message: } }
    if Rails.env.local?
      raise message
    else
      Honeybadger.notify("AI Error: #{message}")
      finalize
    end
```

Mệnh đề else cuối cùng của câu lệnh case sẽ được thực thi nếu không có mẫu nào trước đó khớp. Đây chỉ đơn giản là một biện pháp bảo vệ để chúng ta có thể phát hiện ra khi mô hình AI bắt đầu gửi cho chúng ta những đoạn dữ liệu không được nhận dạng.

```
  else
    Honeybadger.notify("Unrecognized Chunk: #{chunk}")
  end
end
```

Phương thức `send_to_client` có nhiệm vụ gửi nội dung được lưu trong bộ đệm đến client. Nó kiểm tra xem bộ đệm có trống hay không, cập nhật nội dung tin nhắn của bot, hiển thị tin nhắn của bot, và lưu nội dung vào cơ sở dữ liệu để đảm bảo tính bền vững của dữ liệu.

```
def send_to_client
  # no need to process pure whitespace
  return if buffer.join.squish.blank?

  # set the buffer content on the bot message
  content = buffer.join
  bot_message.content = content

  # save to database so that we never lose data
  # even if the stream doesn't terminate correctly
  bot_message.update_column(:content, content)

  # update content via websocket
  ConversationRenderer.update(bot_message)
end
```

Phương thức `finalize` được gọi khi quá trình xử lý luồng hoàn tất. Nó thực thi các lệnh gọi hàm nếu có bất kỳ lệnh nào được nhận trong quá trình xử lý luồng, cập nhật tin nhắn bot với nội dung cuối cùng cùng các thông tin liên quan khác, và thiết lập lại lịch sử gọi hàm

```
def finalize
  if f_name.any?
    f_name.each_with_index do |name, index|
      # takes care of calling the function wherever it's implemented
      dispatch(name:, arguments: JSON.parse(f_arguments[index]))
    end

    # reset the function call history
    f_name.clear
    f_arguments.clear
  else
    content = buffer.join.presence
    bot_message.update!(content:, complete: true)
    ConversationRenderer.update(bot_message)
  end
end
```

Nếu mô hình quyết định gọi một hàm, bạn cần "điều phối" lời gọi hàm đó (tên và các đối

số) theo cách để nó được thực thi và các thông điệp `function_call` và `function_result` được thêm vào bản ghi hội thoại

Theo kinh nghiệm của tôi, việc xử lý việc tạo các thông điệp hàm ở một nơi trong cơ sở mã nguồn của bạn sẽ tốt hơn là phụ thuộc vào các triển khai công cụ. Điều này không chỉ giúp code gọn gàng hơn, mà còn có một lý do thực tế quan trọng: nếu mô hình AI gọi một hàm, và không thấy các thông điệp về lời gọi và kết quả trong bản ghi khi bạn lặp lại, *nó sẽ gọi cùng một hàm một lần nữa.* Có thể là mãi mãi. Hãy nhớ rằng AI hoàn toàn phi trạng thái, vì vậy trừ khi bạn phản hồi những lời gọi hàm đó lại cho nó, chúng được coi như chưa từng xảy ra.

```
# PromptSubscriber#dispatch

def dispatch(name:, arguments:)
  # adds a function_call message to the conversation transcript
  # plus dispatches to tool and returns result
  conversation.function_call!(name, arguments).then do |result|
    # add function result message to the transcript
    conversation.function_result!(name, result)
  end
end
```

Việc xóa lịch sử gọi hàm sau khi thực thi cũng quan trọng như việc đảm bảo lệnh gọi và kết quả được lưu vào bản ghi của bạn, để bạn không phải liên tục gọi cùng một hàm mỗi khi thực hiện vòng lặp.

“Vòng lặp Hội thoại”

Tôi liên tục nhắc đến vòng lặp, nhưng nếu bạn mới làm quen với việc gọi hàm, có thể bạn chưa hiểu rõ *tại sao* chúng ta cần vòng lặp. Lý do là vì một khi AI “yêu cầu” bạn thực thi các hàm công cụ thay mặt nó, nó sẽ ngừng phản hồi. Việc của bạn là thực thi các hàm đó, thu thập kết quả, thêm kết quả vào bản ghi, và sau đó gửi lại lệnh nhắc

ban đầu để nhận được một tập hợp mới các lệnh gọi hàm hoặc kết quả dành cho người dùng.

Trong lớp `PromptSubscriber`, chúng ta sử dụng phương thức `prompt` từ module `PromptDeclarations` để định nghĩa hành vi của vòng lặp hội thoại. Tham số `until` được đặt thành `-> { bot_message.complete? }`, có nghĩa là vòng lặp sẽ tiếp tục cho đến khi `bot_message` được đánh dấu là hoàn thành.

```
1  prompt text: -> { user_message.content },
2         stream: -> { ReplyStream.new(self) },
3         until: -> { bot_message.complete? }
```

Nhưng khi nào `bot_message` được đánh dấu là hoàn thành? Nếu bạn đã quên, hãy xem lại dòng 13 của phương thức `finalize`.

Hãy xem xét toàn bộ logic xử lý luồng.

1. `PromptSubscriber` nhận một tin nhắn mới từ người dùng thông qua phương thức `message_created`, được gọi bởi hệ thống pub/sub Wisper mỗi khi người dùng cuối tạo một prompt mới.
2. Phương thức lớp `prompt` định nghĩa một cách khai báo hành vi của logic hoàn thành cuộc trò chuyện cho `PromptSubscriber`. Mô hình AI sẽ thực hiện hoàn thành cuộc trò chuyện với nội dung tin nhắn của người dùng, một thể hiện mới của `ReplyStream` làm tham số luồng, và điều kiện lặp được chỉ định.
3. Mô hình AI xử lý prompt và bắt đầu tạo phản hồi. Khi phản hồi được truyền theo luồng, phương thức `call` của thể hiện `ReplyStream` được gọi cho mỗi phần dữ liệu.
4. Nếu mô hình AI quyết định gọi một hàm công cụ, tên hàm và các đối số được trích xuất từ phần dữ liệu và lưu trữ trong các mảng `f_name` và `f_arguments` tương ứng.

5. Nếu mô hình AI tạo nội dung hiển thị cho người dùng, nó được đệm và gửi đến máy khách thông qua phương thức `send_to_client`.
6. Khi việc xử lý luồng hoàn tất, phương thức `finalize` được gọi. Nếu có bất kỳ hàm công cụ nào được gọi trong quá trình xử lý luồng, chúng sẽ được gửi đi bằng phương thức `dispatch` của `PromptSubscriber`.
7. Phương thức `dispatch` thêm một tin nhắn `function_call` vào bản ghi hội thoại, thực thi hàm công cụ tương ứng, và thêm một tin nhắn `function_result` vào bản ghi với kết quả của lời gọi hàm.
8. Sau khi gửi đi các hàm công cụ, lịch sử gọi hàm được xóa để ngăn chặn việc gọi hàm trùng lặp trong các vòng lặp tiếp theo.
9. Nếu không có hàm công cụ nào được gọi, phương thức `finalize` cập nhật `bot_message` với nội dung cuối cùng, đánh dấu nó là hoàn thành, và gửi tin nhắn đã cập nhật đến máy khách.
10. Điều kiện lặp `-> { bot_message.complete? }` được đánh giá. Nếu `bot_message` chưa được đánh dấu là hoàn thành, vòng lặp tiếp tục, và prompt ban đầu được gửi lại với bản ghi hội thoại đã được cập nhật.
11. Các bước 3-10 được lặp lại cho đến khi `bot_message` được đánh dấu là hoàn thành, cho biết rằng mô hình AI đã hoàn tất việc tạo phản hồi và không cần thực thi thêm hàm công cụ nào nữa.

Bằng cách triển khai vòng lặp hội thoại này, bạn cho phép mô hình AI tham gia vào tương tác qua lại với ứng dụng, thực thi các hàm công cụ khi cần thiết và tạo ra các phản hồi cho người dùng cho đến khi cuộc hội thoại đạt đến kết luận tự nhiên.

Sự kết hợp của xử lý luồng và vòng lặp hội thoại cho phép tạo ra trải nghiệm tương tác được hỗ trợ bởi AI, nơi mô hình AI có thể xử lý đầu vào của người dùng, sử dụng các công cụ và hàm khác nhau, và cung cấp phản hồi theo thời gian thực dựa trên ngữ cảnh hội thoại đang phát triển.

Tự động Tiếp tục

Điều quan trọng là phải nhận thức được các giới hạn đầu ra của AI. Hầu hết các mô hình đều có số lượng token tối đa mà chúng có thể tạo ra trong một phản hồi duy nhất, được xác định bởi tham số `max_tokens`. Nếu mô hình AI đạt đến giới hạn này trong khi tạo phản hồi, nó sẽ dừng đột ngột và cho biết rằng đầu ra đã bị cắt ngắn.

Trong phản hồi luồng từ API nền tảng AI, bạn có thể phát hiện tình huống này bằng cách kiểm tra biến `finish_reason` trong phần dữ liệu. Nếu `finish_reason` được đặt thành `"length"` (hoặc một giá trị khóa khác đặc trưng cho mô hình), điều đó có nghĩa là mô hình đã đạt đến giới hạn token tối đa trong quá trình tạo và đầu ra đã bị cắt ngắn.

Một cách để xử lý tình huống này một cách nhẹ nhàng và cung cấp trải nghiệm người dùng liền mạch, là triển khai cơ chế tự động tiếp tục trong logic xử lý luồng của bạn. Bằng cách thêm một mẫu so khớp cho các lý do kết thúc liên quan đến độ dài, bạn có thể chọn lặp lại và tự động tiếp tục đầu ra từ nơi nó đã dừng lại.

Đây là một ví dụ được đơn giản hóa có chủ đích về cách bạn có thể sửa đổi phương thức `call` trong lớp `ReplyStream` để hỗ trợ tự động tiếp tục:

```
LENGTH_STOPS = %w[length MAX_TOKENS]

def call(chunk, _bytesize)
  case chunk.deep_symbolize_keys
    # ...

    in {
      choices: [
        { delta: {content: new_content},
          finish_reason: finish_reason } ] }

      buffer << new_content.to_s

      if finish_reason.blank?
        send_to_client if new_content.to_s.match?(/\n\n/)
      elsif LENGTH_STOPS.include?(finish_reason)
```

```
        continue_cutoff
      else
        finalize
      end

    # ...
  end
end

private

def continue_cutoff
  conversation.bot_message!(buffer.join, visible: false)
  conversation.user_message!("please continue", visible: false)
  bot_message.update_column(:created_at, Time.current)
end
```

Trong phiên bản đã được điều chỉnh này, khi `finish_reason` cho biết đầu ra bị cắt ngắn, thay vì kết thúc luồng, chúng ta thêm một cặp tin nhắn vào bản ghi thoại mà không kết thúc nó, di chuyển tin nhắn phản hồi gốc dành cho người dùng xuống "cuối" của bản ghi thoại bằng cách cập nhật thuộc tính `created_at` của nó, và sau đó để vòng lặp tiếp tục, để AI có thể tiếp tục sinh nội dung từ chỗ nó đã dừng lại.

Hãy nhớ rằng điểm cuối hoàn thành của AI là phi trạng thái. Nó chỉ "biết" những gì bạn cho nó biết thông qua bản ghi thoại. Trong trường hợp này, cách chúng ta thông báo cho AI biết rằng nó đã bị cắt ngắn là bằng cách thêm các tin nhắn "ẩn" (đối với người dùng cuối) vào bản ghi thoại. Tuy nhiên, hãy nhớ rằng đây là một ví dụ được đơn giản hóa có chủ đích. Một triển khai thực tế sẽ cần quản lý bản ghi thoại kỹ lưỡng hơn để đảm bảo chúng ta không lãng phí token và/hoặc làm AI bối rối với các tin nhắn trợ lý trùng lặp trong bản ghi thoại.

Một triển khai thực tế của tính năng tự động tiếp tục cũng nên có cái gọi là "logic ngắt mạch" để ngăn chặn việc lặp vô tận. Lý do là, với một số loại lệnh nhắc của người dùng nhất định và cài đặt `max_tokens` thấp, AI có thể tiếp tục lặp lại đầu ra dành cho người dùng vô tận.

> Hãy nhớ rằng mỗi vòng lặp đều yêu cầu một request riêng biệt, và mỗi request đều tiêu tốn toàn bộ bản ghi thoại của bạn một lần nữa. Bạn nên cân nhắc kỹ giữa trải nghiệm người dùng và mức độ sử dụng API khi quyết định có nên triển khai tính năng tự động tiếp tục trong ứng dụng của mình hay không. Đặc biệt, tính năng tự động tiếp tục có thể trở nên nguy hiểm về mặt chi phí, nhất là khi sử dụng các mô hình thương mại cao cấp.

Kết luận

Xử lý luồng là một khía cạnh quan trọng trong việc xây dựng các ứng dụng được hỗ trợ bởi AI kết hợp việc sử dụng công cụ với phản hồi AI trực tiếp. Bằng cách xử lý hiệu quả dữ liệu luồng từ các API nền tảng AI, bạn có thể cung cấp trải nghiệm người dùng mượt mà và tương tác, xử lý các phản hồi lớn, tối ưu hóa việc sử dụng tài nguyên, và xử lý lỗi một cách nhẹ nhàng.

Lớp `Conversation::ReplyStream` được cung cấp minh họa cách xử lý luồng có thể được triển khai trong một ứng dụng Ruby bằng cách sử dụng đối sánh mẫu và kiến trúc hướng sự kiện. Bằng cách hiểu và tận dụng các kỹ thuật xử lý luồng, bạn có thể khai thác toàn bộ tiềm năng của việc tích hợp AI trong ứng dụng của mình và mang lại trải nghiệm người dùng mạnh mẽ và hấp dẫn.

Dữ Liệu Tự Phục Hồi

Dữ liệu tự phục hồi là một phương pháp mạnh mẽ nhằm đảm bảo tính toàn vẹn, nhất quán và chất lượng dữ liệu trong các ứng dụng bằng cách tận dụng khả năng của các mô hình ngôn ngữ lớn (LLMs). Nhóm mẫu thiết kế này tập trung vào ý tưởng sử dụng AI để tự động phát hiện, chẩn đoán và sửa chữa các bất thường, mâu thuẫn hoặc lỗi trong dữ liệu, từ đó giảm bớt gánh nặng cho các nhà phát triển và duy trì độ tin cậy cao của dữ liệu.

Về cốt lõi, các mẫu dữ liệu tự phục hồi công nhận rằng dữ liệu là mạch máu của bất kỳ ứng dụng nào, và việc đảm bảo độ chính xác và tính toàn vẹn của nó là rất quan trọng cho hoạt động đúng đắn và trải nghiệm người dùng của ứng dụng. Tuy nhiên, việc quản lý và duy trì chất lượng dữ liệu có thể là một nhiệm vụ phức tạp và tốn thời gian, đặc biệt khi ứng dụng ngày càng phát triển về quy mô và độ phức tạp. Đây chính là lúc sức mạnh của AI phát huy tác dụng.

Trong các mẫu dữ liệu tự phục hồi, người thực thi AI được sử dụng để liên tục theo dõi và phân tích dữ liệu của ứng dụng của bạn. Những mô hình này có khả năng hiểu và diễn giải các mẫu, mối quan hệ và bất thường trong dữ liệu. Bằng cách tận dụng khả năng xử lý và hiểu ngôn ngữ tự nhiên, chúng có thể xác định các vấn đề tiềm ẩn hoặc mâu thuẫn trong dữ liệu và thực hiện các hành động thích hợp để khắc phục chúng.

Quá trình tự phục hồi dữ liệu thường bao gồm một số bước chính:

1. **Giám sát dữ liệu**: Người thực thi AI liên tục theo dõi các luồng dữ liệu, cơ sở dữ liệu hoặc hệ thống lưu trữ của ứng dụng, tìm kiếm bất kỳ dấu hiệu bất thường, mâu thuẫn hoặc lỗi nào. Ngoài ra, bạn có thể kích hoạt một thành phần AI để phản ứng với một ngoại lệ.
2. **Phát hiện bất thường**: Khi phát hiện vấn đề, người thực thi AI phân tích dữ liệu chi tiết để xác định bản chất và phạm vi cụ thể của vấn đề. Điều này có thể bao gồm việc phát hiện các giá trị bị thiếu, định dạng không nhất quán hoặc dữ liệu vi phạm các quy tắc hoặc ràng buộc đã định trước.
3. **Chẩn đoán và sửa chữa**: Sau khi xác định được vấn đề, người thực thi AI sử dụng kiến thức và hiểu biết về lĩnh vực dữ liệu để xác định hướng hành động phù hợp. Điều này có thể bao gồm tự động sửa chữa dữ liệu, điền vào các giá trị bị thiếu hoặc đánh dấu vấn đề để con người can thiệp nếu cần thiết.
4. **Học liên tục (tùy chọn, tùy thuộc vào trường hợp sử dụng)**: Khi người thực thi AI gặp và giải quyết các vấn đề dữ liệu khác nhau, nó có thể đưa ra mô tả về những gì đã xảy ra và cách nó đã phản ứng. Metadata này có thể được đưa vào các quy trình học tập cho phép bạn (và có thể cả mô hình cơ bản, thông qua tinh chỉnh) trở nên hiệu quả và hiệu suất hơn theo thời gian trong việc xác định và giải quyết các bất thường về dữ liệu.

Bằng cách tự động phát hiện và sửa chữa các vấn đề về dữ liệu, bạn có thể đảm bảo rằng ứng dụng của mình hoạt động trên dữ liệu chất lượng cao, đáng tin cậy. Điều này giảm thiểu rủi ro về lỗi, mâu thuẫn hoặc các lỗi liên quan đến dữ liệu ảnh hưởng đến chức năng hoặc trải nghiệm người dùng của ứng dụng.

Khi bạn có người thực thi AI xử lý nhiệm vụ giám sát và sửa chữa dữ liệu, bạn có thể tập trung nỗ lực vào các khía cạnh quan trọng khác của ứng dụng. Điều này tiết kiệm thời gian và nguồn lực mà lẽ ra phải dành cho việc làm sạch và bảo trì dữ liệu thủ công. Thực tế, khi ứng dụng của bạn ngày càng phát triển về quy mô và độ phức tạp, việc quản lý chất lượng dữ liệu thủ công trở nên ngày càng khó khăn. Các mẫu "Dữ liệu tự phục hồi" mở rộng hiệu quả bằng cách tận dụng sức mạnh của AI để xử lý khối lượng dữ liệu lớn và phát hiện vấn đề trong thời gian thực.

Do bản chất của chúng, các mô hình AI có thể thích ứng với các mẫu dữ liệu, lược đồ hoặc yêu cầu thay đổi theo thời gian với rất ít hoặc không cần giám sát. Miễn là các chỉ thị của chúng cung cấp hướng dẫn đầy đủ, đặc biệt là về kết quả dự kiến, ứng dụng của bạn có thể phát triển và xử lý các kịch bản dữ liệu mới mà không cần can thiệp thủ công nhiều hoặc thay đổi mã.

Các mẫu dữ liệu tự phục hồi phù hợp tốt với các loại mẫu khác mà chúng ta đã thảo luận, chẳng hạn như "Đa dạng người thực thi". Khả năng tự phục hồi dữ liệu có thể được xem như một loại người thực thi chuyên biệt tập trung cụ thể vào việc đảm bảo chất lượng và tính toàn vẹn của dữ liệu. Loại người thực thi này hoạt động cùng với các người thực thi AI khác, mỗi người đóng góp vào các khía cạnh khác nhau của chức năng ứng dụng.

Việc triển khai các mẫu dữ liệu tự phục hồi trong thực tế đòi hỏi thiết kế cẩn thận và tích hợp các mô hình AI vào kiến trúc ứng dụng. Do rủi ro mất mát và hỏng dữ liệu, bạn nên xác định rõ ràng các hướng dẫn về cách sử dụng kỹ thuật này. Bạn cũng nên xem xét các yếu tố như hiệu suất, khả năng mở rộng và bảo mật dữ liệu.

Nghiên Cứu Tình Huống Thực Tế: Sửa Chữa JSON Bị Hỏng

Một trong những cách thực tế và thuận tiện nhất để tận dụng dữ liệu tự phục hồi cũng rất đơn giản để giải thích: sửa chữa JSON bị hỏng.

Kỹ thuật này có thể được áp dụng cho thách thức phổ biến là xử lý dữ liệu không hoàn hảo hoặc không nhất quán được tạo ra bởi các LLM, chẳng hạn như JSON bị hỏng, và cung cấp một phương pháp để tự động phát hiện và sửa chữa những vấn đề này.

Tại Olympia, tôi thường xuyên gặp phải những tình huống khi các LLM tạo ra dữ liệu JSON không hoàn toàn hợp lệ. Điều này có thể xảy ra vì nhiều lý do khác nhau, chẳng hạn như LLM thêm vào các bình luận trước hoặc sau mã JSON thực tế, hoặc gây ra các lỗi cú pháp như thiếu dấu phẩy hoặc dấu ngoặc kép không được escape. Những vấn đề này có thể dẫn đến lỗi phân tích cú pháp và gây ra gián đoạn trong chức năng của ứng dụng.

Để giải quyết vấn đề này, tôi đã triển khai một giải pháp thực tế dưới dạng lớp JsonFixer. Lớp này thể hiện mô hình "Self-Healing Data" bằng cách nhận JSON bị lỗi làm đầu vào và tận dụng một LLM để sửa chữa nó trong khi vẫn bảo toàn càng nhiều thông tin và ý định càng tốt.

```
class JsonFixer
  include Raix::ChatCompletion

  def call(bad_json, error_message)
    raise "No data provided" if bad_json.blank? || error_message.blank?

    transcript << {
        system: "Consider user-provided JSON that generated a parse
                 exception. Do your best to fix it while preserving the
                 original content and intent as much as possible." }
    transcript << { user: bad_json }
    transcript << { assistant: "What is the error message?"}
    transcript << { user: error_message }
```

```
    transcript << { assistant: "Here is the corrected JSON\n```json\n" }

    self.stop = ["```"]

    chat_completion(json: true)
  end

  def model
    "mistralai/mixtral-8x7b-instruct:nitro"
  end
end
```

Lưu ý cách `JsonFixer` sử dụng Ventriloquist để điều hướng các phản hồi của AI.

Quy trình tự phục hồi dữ liệu JSON hoạt động như sau:

1. **Tạo JSON**: LLM (Mô hình Ngôn ngữ Lớn) được sử dụng để tạo dữ liệu JSON dựa trên các lệnh nhắc hoặc yêu cầu nhất định. Tuy nhiên, do bản chất của LLM, JSON được tạo ra có thể không phải lúc nào cũng hoàn toàn hợp lệ. Trình phân tích JSON tất nhiên sẽ báo lỗi `ParserError` nếu bạn cung cấp JSON không hợp lệ.

```
begin
  JSON.parse(llm_generated_json)
rescue JSON::ParserError => e
  JsonFixer.new.call(llm_generated_json, e.message)
end
```

Lưu ý rằng thông báo lỗi cũng được truyền vào lệnh gọi `JSONFixer` để nó không cần phải hoàn toàn giả định về vấn đề của dữ liệu, đặc biệt khi trình phân tích cú pháp thường sẽ cho bạn biết chính xác điều gì đang sai.

2. **Sửa lỗi dựa trên LLM**: Lớp `JSONFixer` gửi JSON bị lỗi trở lại cho LLM, cùng với một prompt hoặc chỉ dẫn cụ thể để sửa JSON trong khi cố gắng giữ nguyên thông tin và ý định ban đầu càng nhiều càng tốt. LLM, được huấn luyện trên một lượng lớn dữ liệu và với khả năng hiểu cú pháp JSON, cố gắng sửa các lỗi và tạo ra một chuỗi JSON hợp lệ. Response Fencing được sử dụng để giới hạn đầu ra của LLM, và chúng tôi chọn Mixtral 8x7B làm mô hình AI, vì nó đặc biệt phù hợp cho loại nhiệm vụ này.
3. **Xác thực và Tích hợp**: Chuỗi JSON đã được sửa do LLM trả về được phân tích bởi chính lớp `JSONFixer`, bởi vì nó đã gọi `chat_completion(json: true)`. Nếu JSON đã sửa vượt qua bước xác thực, nó sẽ được tích hợp trở lại vào luồng công việc của ứng dụng, cho phép ứng dụng tiếp tục xử lý dữ liệu một cách liền mạch. JSON có lỗi đã được "chữa lành".

Mặc dù tôi đã viết và viết lại triển khai `JSONFixer` của riêng mình nhiều lần, tôi nghi ngờ rằng tổng thời gian đầu tư cho tất cả các phiên bản đó không quá một hoặc hai giờ.

Lưu ý rằng việc bảo toàn ý định là một yếu tố quan trọng của bất kỳ mẫu dữ liệu tự phục hồi nào. Quá trình sửa lỗi dựa trên LLM nhằm bảo toàn thông tin và ý định ban đầu của JSON được tạo ra càng nhiều càng tốt. Điều này đảm bảo rằng JSON đã sửa vẫn giữ được ý nghĩa ngữ nghĩa của nó và có thể được sử dụng hiệu quả trong ngữ cảnh của ứng dụng.

Việc triển khai thực tế phương pháp "Dữ liệu Tự Phục hồi" trong Olympia cho thấy rõ ràng cách AI, đặc biệt là LLM, có thể được tận dụng để giải quyết các thách thức dữ liệu trong thực tế. Nó thể hiện sức mạnh của việc kết hợp các kỹ thuật lập trình truyền thống với khả năng AI để xây dựng các ứng dụng mạnh mẽ và hiệu quả.

Nguyên tắc Postel và Mẫu "Dữ liệu Tự Phục hồi"

"Dữ liệu Tự Phục hồi", như được minh họa bởi lớp JSONFixer, phù hợp với nguyên tắc được biết đến như Nguyên tắc Postel, còn được gọi là Nguyên tắc Mạnh mẽ. Nguyên tắc Postel phát biểu:

"Hãy bảo thủ trong những gì bạn làm, hãy cởi mở trong những gì bạn chấp nhận từ người khác."

Nguyên tắc này, ban đầu được diễn đạt bởi Jon Postel, một người tiên phong của Internet thời kỳ đầu, nhấn mạnh tầm quan trọng của việc xây dựng các hệ thống có khả năng chấp nhận đầu vào đa dạng hoặc thậm chí hơi sai lệch trong khi vẫn duy trì sự tuân thủ nghiêm ngặt các giao thức đã định khi gửi đầu ra.

Trong ngữ cảnh của "Dữ liệu Tự Phục hồi", lớp JSONFixer thể hiện Nguyên tắc Postel bằng cách cởi mở trong việc chấp nhận dữ liệu JSON bị hỏng hoặc không hoàn hảo được tạo ra bởi LLM. Nó không ngay lập tức từ chối hoặc thất bại khi gặp JSON không tuân thủ nghiêm ngặt định dạng mong đợi. Thay vào đó, nó áp dụng một cách tiếp cận khoan dung và cố gắng sửa chữa JSON bằng sức mạnh của LLM.

Bằng cách cởi mở trong việc chấp nhận JSON không hoàn hảo, lớp JSONFixer thể hiện tính mạnh mẽ và linh hoạt. Nó thừa nhận rằng dữ liệu trong thế giới thực thường xuất hiện dưới nhiều hình thức và có thể không phải lúc nào cũng tuân theo các đặc tả nghiêm ngặt. Bằng cách xử lý và sửa chữa một cách nhẹ nhàng những sai lệch này, lớp này đảm bảo rằng ứng dụng có thể tiếp tục hoạt động trơn tru, ngay cả khi có dữ liệu không hoàn hảo.

Mặt khác, lớp JSONFixer cũng tuân thủ khía cạnh bảo thủ của Nguyên tắc Postel khi nói đến đầu ra. Sau khi sửa JSON bằng LLM, lớp này xác thực JSON đã sửa để đảm bảo nó tuân thủ nghiêm ngặt định dạng mong đợi. Nó duy trì tính toàn vẹn và chính xác của dữ liệu trước khi chuyển nó đến các phần khác của ứng dụng. Cách tiếp cận bảo thủ này đảm bảo rằng đầu ra của lớp JSONFixer đáng tin cậy và nhất quán, thúc đẩy khả năng tương tác và ngăn chặn sự lan truyền của lỗi.

Thông tin thú vị về Jon Postel:

- Jon Postel (1943-1998) là một nhà khoa học máy tính người Mỹ đã đóng vai trò quan trọng trong sự phát triển của Internet. Ông được biết đến như "Vị thần của Internet" vì những đóng góp quan trọng của mình cho các giao thức và tiêu chuẩn nền tảng.
- Postel là biên tập viên của loạt tài liệu Request for Comments (RFC), một loạt các ghi chú kỹ thuật và tổ chức về Internet. Ông đã viết hoặc đồng tác giả hơn 200 RFC, bao gồm các giao thức nền tảng như TCP, IP và SMTP.
- Ngoài những đóng góp kỹ thuật, Postel còn nổi tiếng với cách tiếp cận khiêm tốn và hợp tác. Ông tin vào tầm quan trọng của việc đạt được sự đồng thuận và làm việc cùng nhau để xây dựng một mạng lưới mạnh mẽ và có khả năng tương tác.
- Postel giữ chức vụ Giám đốc Bộ phận Mạng Máy tính tại Viện Khoa học Thông tin (ISI) của Đại học Nam California (USC) từ năm 1977 cho đến khi ông qua đời đột ngột vào năm 1998.
- Để ghi nhận những đóng góp to lớn của ông, Postel đã được trao giải thưởng Turing danh giá sau khi mất vào năm 1998, thường được gọi là "Giải Nobel của Ngành Máy tính."

Lớp `JSONFixer` thúc đẩy tính mạnh mẽ, linh hoạt và khả năng tương tác, vốn là những giá trị cốt lõi mà Postel đã duy trì trong suốt sự nghiệp của mình. Bằng cách xây dựng các hệ thống có khả năng chấp nhận những khiếm khuyết trong khi vẫn tuân thủ nghiêm ngặt các giao thức, chúng ta có thể tạo ra những ứng dụng có khả năng phục hồi và thích ứng tốt hơn trước những thách thức trong thực tế.

Các Cân Nhắc và Chống Chỉ Định

Khả năng áp dụng các phương pháp dữ liệu tự phục hồi hoàn toàn phụ thuộc vào loại dữ liệu mà ứng dụng của bạn xử lý. Có lý do tại sao bạn không nên đơn giản chỉ vá tạm thời `JSON.parse` để tự động sửa chữa *tất cả các lỗi phân tích cú pháp JSON* trong ứng

dụng của bạn: không phải tất cả các lỗi đều có thể hoặc nên được tự động sửa chữa.

Việc tự phục hồi đặc biệt phức tạp khi gắn liền với các yêu cầu quy định hoặc tuân thủ liên quan đến việc xử lý và quản lý dữ liệu. Một số ngành công nghiệp, như y tế và tài chính, có những quy định nghiêm ngặt về tính toàn vẹn dữ liệu và khả năng kiểm toán đến mức việc thực hiện bất kỳ sửa chữa dữ liệu "hộp đen" nào mà không có giám sát hoặc ghi nhật ký thích hợp có thể vi phạm các quy định này. Điều quan trọng là đảm bảo rằng bất kỳ kỹ thuật dữ liệu tự phục hồi nào bạn đưa ra đều phù hợp với các khuôn khổ pháp lý và quy định hiện hành.

Việc áp dụng các kỹ thuật dữ liệu tự phục hồi, đặc biệt là những kỹ thuật liên quan đến các mô hình AI, cũng có thể có tác động lớn đến hiệu suất ứng dụng và việc sử dụng tài nguyên. Việc xử lý khối lượng dữ liệu lớn thông qua các mô hình AI để phát hiện và sửa chữa lỗi có thể tốn nhiều tài nguyên tính toán. Điều quan trọng là đánh giá sự cân bằng giữa lợi ích của dữ liệu tự phục hồi và chi phí hiệu suất cũng như tài nguyên liên quan.

Tuy nhiên, hãy cùng đi sâu vào các yếu tố liên quan đến việc quyết định khi nào và ở đâu nên áp dụng phương pháp mạnh mẽ này.

Tính Trọng Yếu của Dữ Liệu

Khi xem xét việc áp dụng các kỹ thuật dữ liệu tự phục hồi, việc đánh giá tính trọng yếu của dữ liệu đang được xử lý là rất quan trọng. Mức độ trọng yếu đề cập đến tầm quan trọng và độ nhạy cảm của dữ liệu trong bối cảnh ứng dụng và lĩnh vực kinh doanh của bạn.

Trong một số trường hợp, việc tự động sửa chữa lỗi dữ liệu có thể không phù hợp, đặc biệt là nếu dữ liệu có độ nhạy cảm cao hoặc có ý nghĩa pháp lý. Ví dụ, hãy xem xét các tình huống sau:

1. **Giao dịch Tài chính**: Trong các ứng dụng tài chính, chẳng hạn như hệ thống ngân hàng hoặc nền tảng giao dịch, độ chính xác của dữ liệu là vô cùng quan trọng.

Ngay cả những lỗi nhỏ trong dữ liệu tài chính cũng có thể dẫn đến những hậu quả đáng kể, như số dư tài khoản không chính xác, chuyển tiền sai hướng, hoặc quyết định giao dịch sai lầm. Trong những trường hợp này, việc sửa chữa tự động mà không có xác minh và kiểm toán kỹ lưỡng có thể gây ra những rủi ro không thể chấp nhận được.

2. **Hồ sơ Y tế**: Các ứng dụng y tế xử lý dữ liệu bệnh nhân có độ nhạy cảm và bảo mật cao. Những sai sót trong hồ sơ y tế có thể có những tác động nghiêm trọng đến sự an toàn và quyết định điều trị cho bệnh nhân. Việc tự động sửa đổi dữ liệu y tế mà không có sự giám sát và xác nhận thích hợp từ các chuyên gia y tế có thể vi phạm các yêu cầu quy định và gây nguy hiểm cho sức khỏe của bệnh nhân.
3. **Tài liệu Pháp lý**: Các ứng dụng xử lý tài liệu pháp lý, như hợp đồng, thỏa thuận, hoặc hồ sơ tòa án, đòi hỏi độ chính xác và tính toàn vẹn nghiêm ngặt. Ngay cả những lỗi nhỏ trong dữ liệu pháp lý cũng có thể có những hệ quả pháp lý đáng kể. Việc sửa chữa tự động trong lĩnh vực này có thể không phù hợp, vì dữ liệu thường đòi hỏi sự xem xét và xác minh thủ công bởi các chuyên gia pháp lý để đảm bảo tính hợp lệ và khả năng thực thi của nó.

Trong những kịch bản dữ liệu quan trọng này, những rủi ro liên quan đến việc sửa chữa tự động thường lớn hơn những lợi ích tiềm năng. Hậu quả của việc đưa ra lỗi hoặc sửa đổi dữ liệu không chính xác có thể nghiêm trọng, dẫn đến tổn thất tài chính, trách nhiệm pháp lý, hoặc thậm chí gây hại cho cá nhân.

Khi xử lý dữ liệu có độ quan trọng cao, điều cần thiết là ưu tiên các quy trình xác minh và xác thực thủ công. Sự giám sát và chuyên môn của con người là rất quan trọng trong việc đảm bảo độ chính xác và tính toàn vẹn của dữ liệu. Các kỹ thuật tự phục hồi tự động vẫn có thể được sử dụng để đánh dấu các lỗi hoặc sự không nhất quán tiềm ẩn, nhưng quyết định cuối cùng về việc sửa chữa nên có sự đánh giá và phê duyệt của con người.

Tuy nhiên, điều quan trọng cần lưu ý là không phải tất cả dữ liệu trong một ứng dụng đều có cùng mức độ trọng yếu. Trong cùng một ứng dụng, có thể có những tập hợp dữ

liệu ít nhạy cảm hơn hoặc có tác động thấp hơn nếu xảy ra lỗi. Trong những trường hợp như vậy, các kỹ thuật dữ liệu tự phục hồi có thể được áp dụng có chọn lọc cho những tập dữ liệu cụ thể đó, trong khi dữ liệu quan trọng vẫn phải tuân theo quy trình xác minh thủ công.

Điều quan trọng là phải đánh giá cẩn thận tính trọng yếu của từng loại dữ liệu trong ứng dụng của bạn và xác định các hướng dẫn và quy trình rõ ràng để xử lý việc sửa chữa dựa trên các rủi ro và hệ quả liên quan. Bằng cách phân biệt giữa dữ liệu quan trọng (ví dụ: sổ cái, hồ sơ y tế) và dữ liệu không quan trọng (ví dụ: địa chỉ gửi thư, cảnh báo tài nguyên), bạn có thể cân bằng giữa việc tận dụng lợi ích của các kỹ thuật dữ liệu tự phục hồi khi thích hợp và duy trì kiểm soát và giám sát nghiêm ngặt khi cần thiết.

Cuối cùng, quyết định áp dụng các kỹ thuật dữ liệu tự phục hồi cho dữ liệu quan trọng nên được thực hiện sau khi tham khảo ý kiến của các chuyên gia trong lĩnh vực, cố vấn pháp lý và các bên liên quan khác. Điều cần thiết là xem xét các yêu cầu cụ thể, quy định và rủi ro liên quan đến dữ liệu của ứng dụng của bạn và điều chỉnh các chiến lược sửa chữa dữ liệu cho phù hợp.

Mức Độ Nghiêm Trọng của Lỗi

Khi áp dụng các kỹ thuật dữ liệu tự phục hồi, điều quan trọng là đánh giá mức độ nghiêm trọng và tác động của các lỗi dữ liệu. Không phải tất cả các lỗi đều giống nhau, và hướng hành động thích hợp có thể khác nhau tùy thuộc vào mức độ nghiêm trọng của vấn đề.

Những sự không nhất quán nhỏ hoặc vấn đề về định dạng có thể phù hợp để sửa chữa tự động. Ví dụ, một worker dữ liệu tự phục hồi được giao nhiệm vụ sửa chữa JSON bị hỏng có thể xử lý các dấu phẩy bị thiếu hoặc dấu ngoặc kép chưa được escape mà không làm thay đổi đáng kể ý nghĩa hoặc cấu trúc của dữ liệu. Những loại lỗi này thường dễ dàng sửa chữa và có tác động tối thiểu đến tính toàn vẹn dữ liệu tổng thể.

Tuy nhiên, những lỗi nghiêm trọng hơn làm thay đổi căn bản ý nghĩa hoặc tính toàn vẹn của dữ liệu có thể cần một cách tiếp cận khác. Trong những trường hợp như vậy,

việc sửa chữa tự động có thể không đủ, và sự can thiệp của con người có thể là cần thiết để đảm bảo tính chính xác và hợp lệ của dữ liệu.

Đây là lúc khái niệm sử dụng chính AI để giúp xác định mức độ nghiêm trọng của lỗi được đưa vào áp dụng. Bằng cách tận dụng khả năng của các mô hình AI, chúng ta có thể thiết kế những người xử lý dữ liệu tự phục hồi không chỉ sửa chữa lỗi mà còn đánh giá mức độ nghiêm trọng của những lỗi đó và đưa ra quyết định sáng suốt về cách xử lý chúng.

Ví dụ, hãy xem xét một người xử lý dữ liệu tự phục hồi có trách nhiệm sửa chữa những điểm không nhất quán trong dữ liệu đang chảy vào cơ sở dữ liệu khách hàng. Người xử lý này có thể được thiết kế để phân tích dữ liệu và xác định các lỗi tiềm ẩn, chẳng hạn như thông tin bị thiếu hoặc mâu thuẫn. Tuy nhiên, thay vì tự động sửa chữa tất cả các lỗi, người xử lý có thể được trang bị thêm các lệnh gọi công cụ cho phép đánh dấu các lỗi nghiêm trọng để con người xem xét.

Dưới đây là ví dụ về cách triển khai điều này:

```
class CustomerDataReviewer
  include Raix::ChatCompletion
  include Raix::FunctionDeclarations

  attr_accessor :customer

  function :flag_for_review, reason: { type: "string" } do |params|
    AdminNotifier.review_request(customer, params[:reason])
  end

  def initialize(customer)
    self.customer = customer
  end

  def call(customer_data)
    transcript << {
      system: "You are a customer data reviewer. Your task is to identify
        and correct inconsistencies in customer data.

        < additional instructions here... >
```

```
21
22          If you encounter severe errors that require human review, use the
23          `flag_for_review` tool to flag the data for manual intervention." }
24
25      transcript << { user: customer.to_json }
26      transcript << { assistant: "Reviewed/corrected data:\n```json\n" }
27
28      self.stop = ["```"]
29
30      chat_completion(json: true).then do |result|
31        return if result.blank?
32
33        customer.update(result)
34      end
35    end
36  end
```

Trong ví dụ này, trình xử lý `CustomerDataHealer` được thiết kế để nhận diện và sửa chữa các sự không nhất quán trong dữ liệu khách hàng. Một lần nữa, chúng ta sử dụng Phân vùng phản hồi và Kỹ thuật điều khiển phản hồi để có được đầu ra có cấu trúc. Đáng chú ý là chỉ thị hệ thống của trình xử lý bao gồm hướng dẫn sử dụng hàm `flag_for_review` nếu gặp phải lỗi nghiêm trọng.

Khi trình xử lý tiến hành xử lý dữ liệu khách hàng, nó phân tích dữ liệu và cố gắng sửa chữa bất kỳ sự không nhất quán nào. Nếu trình xử lý xác định rằng các lỗi là nghiêm trọng và cần có sự can thiệp của con người, nó có thể sử dụng công cụ `flag_for_review` để đánh dấu dữ liệu và cung cấp lý do cho việc đánh dấu đó.

Phương thức `chat_completion` được gọi với `json: true` để phân tích dữ liệu khách hàng đã được sửa chữa dưới dạng JSON. Không có quy định cho việc lặp lại sau khi gọi hàm, vì vậy kết quả sẽ trống nếu `flag_for_review` được kích hoạt. Ngược lại, thông tin khách hàng sẽ được cập nhật với dữ liệu đã được xem xét và có thể đã được sửa chữa.

Bằng cách kết hợp đánh giá mức độ nghiêm trọng của lỗi và tùy chọn đánh dấu dữ liệu để xem xét thủ công, trình xử lý dữ liệu tự phục hồi trở nên thông minh và thích ứng hơn. Nó có thể xử lý các lỗi nhỏ một cách tự động trong khi chuyển các lỗi nghiêm trọng

đến các chuyên gia để can thiệp thủ công.

Các tiêu chí cụ thể để xác định mức độ nghiêm trọng của lỗi có thể được định nghĩa trong chỉ thị của trình xử lý dựa trên kiến thức chuyên môn và yêu cầu kinh doanh. Các yếu tố như tác động đến tính toàn vẹn dữ liệu, khả năng mất mát hoặc hỏng dữ liệu, và hậu quả của dữ liệu không chính xác có thể được xem xét khi đánh giá mức độ nghiêm trọng.

Bằng cách tận dụng AI để đánh giá mức độ nghiêm trọng của lỗi và cung cấp các tùy chọn cho sự can thiệp của con người, các kỹ thuật dữ liệu tự phục hồi có thể tạo ra sự cân bằng giữa tự động hóa và duy trì độ chính xác của dữ liệu. Cách tiếp cận này đảm bảo rằng các lỗi nhỏ được sửa chữa hiệu quả trong khi các lỗi nghiêm trọng nhận được sự quan tâm và chuyên môn cần thiết từ người xem xét.

Độ phức tạp của lĩnh vực

Khi xem xét việc áp dụng các kỹ thuật dữ liệu tự phục hồi, điều quan trọng là đánh giá độ phức tạp của lĩnh vực dữ liệu và các quy tắc chi phối cấu trúc và mối quan hệ của nó. Độ phức tạp của lĩnh vực có thể ảnh hưởng đáng kể đến hiệu quả và khả năng thực hiện của các phương pháp sửa chữa dữ liệu tự động.

Các kỹ thuật dữ liệu tự phục hồi hoạt động tốt khi dữ liệu tuân theo các mẫu và ràng buộc được định nghĩa rõ ràng. Trong các lĩnh vực mà cấu trúc dữ liệu tương đối đơn giản và mối quan hệ giữa các phần tử dữ liệu là đơn giản, các sửa chữa tự động có thể được áp dụng với độ tin cậy cao. Ví dụ, việc sửa chữa các vấn đề về định dạng hoặc thực thi các ràng buộc kiểu dữ liệu cơ bản thường có thể được xử lý hiệu quả bởi các trình xử lý dữ liệu tự phục hồi.

Tuy nhiên, khi độ phức tạp của lĩnh vực dữ liệu tăng lên, các thách thức liên quan đến việc sửa chữa dữ liệu tự động cũng tăng theo. Trong các lĩnh vực có logic kinh doanh phức tạp, mối quan hệ phức tạp giữa các thực thể dữ liệu, hoặc các quy tắc và ngoại lệ đặc thù của lĩnh vực, các kỹ thuật dữ liệu tự phục hồi có thể không luôn nắm bắt được các sắc thái và có thể gây ra những hậu quả không mong muốn.

Hãy xem xét một ví dụ về lĩnh vực phức tạp: hệ thống giao dịch tài chính. Trong lĩnh vực này, dữ liệu liên quan đến các công cụ tài chính khác nhau, dữ liệu thị trường, quy tắc giao dịch và yêu cầu quy định. Mối quan hệ giữa các phần tử dữ liệu khác nhau có thể phức tạp, và các quy tắc chi phối tính hợp lệ và nhất quán của dữ liệu có thể rất đặc thù cho lĩnh vực.

Trong một lĩnh vực phức tạp như vậy, một trình xử lý dữ liệu tự phục hồi được giao nhiệm vụ sửa chữa sự không nhất quán trong dữ liệu giao dịch sẽ cần phải có hiểu biết sâu sắc về các quy tắc và ràng buộc đặc thù của lĩnh vực. Nó cần xem xét các yếu tố như quy định thị trường, giới hạn giao dịch, tính toán rủi ro và thủ tục thanh toán. Các sửa chữa tự động trong bối cảnh này có thể không luôn nắm bắt được toàn bộ độ phức tạp của lĩnh vực và có thể vô tình gây ra lỗi hoặc vi phạm các quy tắc đặc thù của lĩnh vực.

Để giải quyết các thách thức về độ phức tạp của lĩnh vực, các kỹ thuật dữ liệu tự phục hồi có thể được cải thiện bằng cách kết hợp kiến thức và quy tắc đặc thù của lĩnh vực vào các mô hình và trình xử lý AI. Điều này có thể đạt được thông qua các kỹ thuật như:

1. **Đào tạo chuyên biệt theo lĩnh vực**: Các mô hình AI được sử dụng cho dữ liệu tự phục hồi có thể được hướng dẫn hoặc thậm chí tinh chỉnh trên các tập dữ liệu đặc thù của lĩnh vực, nắm bắt các sắc thái và quy tắc của lĩnh vực cụ thể đó. Bằng cách cho các mô hình tiếp xúc với dữ liệu và kịch bản đại diện, chúng có thể học được các mẫu, ràng buộc và ngoại lệ đặc thù của lĩnh vực.
2. **Ràng buộc dựa trên quy tắc**: Các trình xử lý dữ liệu tự phục hồi có thể được bổ sung với các ràng buộc dựa trên quy tắc rõ ràng mã hóa kiến thức đặc thù của lĩnh vực. Các quy tắc này có thể được định nghĩa bởi các chuyên gia lĩnh vực và tích hợp vào quá trình sửa chữa dữ liệu. Các mô hình AI sau đó có thể sử dụng các quy tắc này để hướng dẫn quyết định của chúng và đảm bảo tuân thủ các yêu cầu đặc thù của lĩnh vực.
3. **Hợp tác với chuyên gia lĩnh vực**: Trong các lĩnh vực phức tạp, việc có sự tham gia của các chuyên gia lĩnh vực trong thiết kế và phát triển các kỹ thuật dữ liệu tự

phục hồi là rất quan trọng. Các chuyên gia lĩnh vực có thể cung cấp những hiểu biết quý giá về sự phức tạp của dữ liệu, các quy tắc kinh doanh và các trường hợp ngoại lệ tiềm ẩn. Kiến thức của họ có thể được tích hợp vào các mô hình và trình xử lý AI để cải thiện độ chính xác và độ tin cậy của việc sửa chữa dữ liệu tự động bằng cách sử dụng các mẫu Con người trong vòng lặp.

4. **Phương pháp tiếp cận tăng dần và lặp lại**: Khi xử lý các lĩnh vực phức tạp, thường có lợi khi áp dụng phương pháp tiếp cận tăng dần và lặp lại đối với dữ liệu tự phục hồi. Thay vì cố gắng tự động hóa việc sửa chữa cho toàn bộ lĩnh vực cùng một lúc, hãy tập trung vào các lĩnh vực con hoặc danh mục dữ liệu cụ thể mà ở đó các quy tắc và ràng buộc được hiểu rõ. Dần dần mở rộng phạm vi của các kỹ thuật tự phục hồi khi sự hiểu biết về lĩnh vực tăng lên và các kỹ thuật chứng minh được hiệu quả.

Bằng cách xem xét độ phức tạp của lĩnh vực dữ liệu và tích hợp kiến thức chuyên ngành vào các kỹ thuật dữ liệu tự phục hồi, bạn có thể tạo ra sự cân bằng giữa tự động hóa và độ chính xác. Điều quan trọng là phải nhận ra rằng dữ liệu tự phục hồi không phải là giải pháp phù hợp cho mọi trường hợp và cách tiếp cận cần được điều chỉnh cho phù hợp với yêu cầu và thách thức cụ thể của từng lĩnh vực.

Trong các lĩnh vực phức tạp, cách tiếp cận kết hợp giữa kỹ thuật dữ liệu tự phục hồi với chuyên môn và giám sát của con người có thể mang lại hiệu quả cao nhất. Việc sửa chữa tự động có thể xử lý các trường hợp thông thường và được định nghĩa rõ ràng, trong khi các tình huống phức tạp hoặc ngoại lệ có thể được đánh dấu để con người xem xét và can thiệp. Cách tiếp cận hợp tác này đảm bảo rằng lợi ích của tự động hóa được hiện thực hóa trong khi vẫn duy trì được sự kiểm soát và độ chính xác cần thiết trong các lĩnh vực dữ liệu phức tạp.

Khả năng Giải thích và Tính Minh bạch

Khả năng giải thích đề cập đến khả năng hiểu và diễn giải lý do đằng sau các quyết định được đưa ra bởi các mô hình AI, trong khi tính minh bạch liên quan đến việc cung cấp

tầm nhìn rõ ràng vào quá trình sửa chữa dữ liệu.

Trong nhiều bối cảnh, các thay đổi dữ liệu cần có khả năng kiểm toán và biện minh. Các bên liên quan, bao gồm người dùng doanh nghiệp, kiểm toán viên và các cơ quan quản lý, có thể yêu cầu giải thích về lý do tại sao một số sửa chữa dữ liệu được thực hiện và cách các mô hình AI đi đến những quyết định đó. Điều này đặc biệt quan trọng trong các lĩnh vực mà độ chính xác và tính toàn vẹn của dữ liệu có ý nghĩa quan trọng, như tài chính, y tế và các vấn đề pháp lý.

Để đáp ứng nhu cầu về khả năng giải thích và tính minh bạch, các kỹ thuật dữ liệu tự phục hồi nên tích hợp các cơ chế cung cấp thông tin chi tiết về quá trình ra quyết định của các mô hình AI. Điều này có thể đạt được thông qua nhiều cách tiếp cận:

1. **Chuỗi Tư duy**: Yêu cầu mô hình giải thích suy nghĩ của nó "thành tiếng" trước khi áp dụng thay đổi vào dữ liệu có thể giúp dễ dàng hiểu được quá trình ra quyết định và có thể tạo ra các giải thích dễ hiểu cho con người về các sửa chữa đã thực hiện. Sự đánh đổi là một chút phức tạp hơn trong việc tách biệt giải thích khỏi đầu ra dữ liệu có cấu trúc, điều này có thể được giải quyết bằng...
2. **Tạo Giải thích**: Các công cụ xử lý dữ liệu tự phục hồi có thể được trang bị khả năng tạo ra các giải thích dễ hiểu cho con người về các sửa chữa mà chúng thực hiện. Điều này có thể đạt được bằng cách yêu cầu mô hình đưa ra quá trình ra quyết định của nó dưới dạng các giải thích dễ hiểu *được tích hợp vào chính dữ liệu*. Ví dụ, một công cụ xử lý dữ liệu tự phục hồi có thể tạo ra một báo cáo làm nổi bật các sự không nhất quán cụ thể trong dữ liệu mà nó đã xác định, các sửa chữa đã áp dụng và lý do đằng sau những sửa chữa đó.
3. **Tầm quan trọng của Đặc trưng**: Các mô hình AI có thể được hướng dẫn với thông tin về tầm quan trọng của các đặc trưng hoặc thuộc tính khác nhau trong quá trình sửa chữa dữ liệu như một phần trong chỉ thị của chúng. Những chỉ thị này, đến lượt nó, có thể được hiển thị cho các bên liên quan. Bằng cách xác định các yếu tố chính ảnh hưởng đến quyết định của mô hình, các bên liên quan có thể hiểu được lý do đằng sau các sửa chữa và đánh giá tính hợp lệ của chúng.

4. **Ghi nhật ký và Kiểm toán**: Việc triển khai các cơ chế ghi nhật ký và kiểm toán toàn diện là rất quan trọng để duy trì tính minh bạch trong quá trình xử lý dữ liệu tự phục hồi. Mọi sửa chữa dữ liệu được thực hiện bởi các mô hình AI đều phải được ghi lại, bao gồm dữ liệu gốc, dữ liệu đã sửa chữa và các hành động cụ thể đã thực hiện. Dấu vết kiểm toán này cho phép phân tích hồi cứu và cung cấp hồ sơ rõ ràng về các thay đổi đã thực hiện đối với dữ liệu.
5. **Cách tiếp cận Có sự Tham gia của Con người**: Việc tích hợp cách tiếp cận có sự tham gia của con người có thể nâng cao khả năng giải thích và tính minh bạch của các kỹ thuật dữ liệu tự phục hồi. Bằng cách có sự tham gia của các chuyên gia trong việc xem xét và xác nhận các sửa chữa do AI tạo ra, các tổ chức có thể đảm bảo rằng các sửa chữa phù hợp với kiến thức chuyên ngành và yêu cầu kinh doanh. Sự giám sát của con người bổ sung thêm một lớp trách nhiệm giải trình và cho phép xác định bất kỳ thiên kiến hoặc lỗi tiềm ẩn nào trong các mô hình AI.
6. **Giám sát và Đánh giá Liên tục**: Việc thường xuyên giám sát và đánh giá hiệu suất của các kỹ thuật dữ liệu tự phục hồi là điều cần thiết để duy trì tính minh bạch và niềm tin. Bằng cách đánh giá độ chính xác và hiệu quả của các mô hình AI theo thời gian, các tổ chức có thể xác định bất kỳ sai lệch hoặc bất thường nào và thực hiện các hành động khắc phục. Giám sát liên tục giúp đảm bảo rằng quá trình xử lý dữ liệu tự phục hồi vẫn đáng tin cậy và phù hợp với các kết quả mong muốn.

Khả năng giải thích và tính minh bạch là những yếu tố quan trọng khi triển khai các kỹ thuật dữ liệu tự phục hồi. Bằng cách cung cấp các giải thích rõ ràng cho việc sửa chữa dữ liệu, duy trì dấu vết kiểm toán toàn diện và có sự giám sát của con người, các tổ chức có thể xây dựng niềm tin vào quá trình xử lý dữ liệu tự phục hồi và đảm bảo rằng các thay đổi được thực hiện đối với dữ liệu là có thể biện minh được và phù hợp với mục tiêu kinh doanh.

Điều quan trọng là phải tạo ra sự cân bằng giữa lợi ích của tự động hóa và nhu cầu về tính minh bạch. Mặc dù các kỹ thuật dữ liệu tự phục hồi có thể cải thiện đáng kể chất

lượng và hiệu quả của dữ liệu, chúng không nên đánh đổi bằng việc mất đi khả năng quan sát và kiểm soát quá trình sửa chữa dữ liệu. Bằng cách thiết kế các công cụ xử lý dữ liệu tự phục hồi với khả năng giải thích và tính minh bạch, các tổ chức có thể khai thác sức mạnh của AI trong khi vẫn duy trì được mức độ trách nhiệm giải trình và niềm tin cần thiết vào dữ liệu.

Hệ quả Ngoài ý muốn

Mặc dù các kỹ thuật dữ liệu tự phục hồi nhằm mục đích cải thiện chất lượng và tính nhất quán của dữ liệu, điều quan trọng là phải nhận thức được khả năng xảy ra các hệ quả ngoài ý muốn. Các sửa chữa tự động, nếu không được thiết kế và giám sát cẩn thận, có thể vô tình thay đổi ý nghĩa hoặc ngữ cảnh của dữ liệu, dẫn đến các vấn đề phát sinh.

Một trong những rủi ro chính của dữ liệu tự phục hồi là việc đưa vào các thiên kiến hoặc lỗi trong quá trình sửa chữa dữ liệu. Các mô hình AI, giống như bất kỳ hệ thống phần mềm nào khác, có thể chịu ảnh hưởng của các thiên kiến có trong dữ liệu huấn luyện hoặc được đưa vào thông qua thiết kế của thuật toán. Nếu những thiên kiến này không được xác định và giảm thiểu, chúng có thể lan truyền qua quá trình xử lý dữ liệu tự phục hồi và dẫn đến các sửa đổi dữ liệu bị lệch hoặc không chính xác.

Ví dụ, hãy xem xét một công cụ tự phục hồi dữ liệu được giao nhiệm vụ sửa chữa những mâu thuẫn trong dữ liệu nhân khẩu học của khách hàng. Nếu mô hình AI đã học được những định kiến từ dữ liệu lịch sử, chẳng hạn như việc liên kết một số nghề nghiệp hoặc mức thu nhập nhất định với giới tính hoặc dân tộc cụ thể, nó có thể đưa ra những giả định sai lầm và sửa đổi dữ liệu theo cách củng cố những định kiến đó. Điều này có thể dẫn đến hồ sơ khách hàng không chính xác, quyết định kinh doanh sai lệch và có thể tạo ra những kết quả mang tính phân biệt đối xử.

Một hậu quả ngoài ý muốn khác là việc mất thông tin hoặc ngữ cảnh quan trọng trong quá trình sửa chữa dữ liệu. Các kỹ thuật tự phục hồi dữ liệu thường tập trung vào việc chuẩn hóa dữ liệu để đảm bảo tính nhất quán. Tuy nhiên, trong một số trường hợp, dữ liệu gốc có thể chứa những sắc thái, ngoại lệ hoặc thông tin ngữ cảnh quan trọng để hiểu

toàn bộ bức tranh. Việc sửa chữa tự động mà áp dụng chuẩn hóa một cách mù quáng có thể vô tình loại bỏ hoặc làm mờ những thông tin có giá trị này.

Ví dụ, hãy tưởng tượng một công cụ tự phục hồi dữ liệu chịu trách nhiệm sửa chữa những mâu thuẫn trong hồ sơ y tế. Nếu công cụ này gặp phải tiền sử bệnh của một bệnh nhân có tình trạng hiếm gặp hoặc kế hoạch điều trị bất thường, nó có thể cố gắng chuẩn hóa dữ liệu để phù hợp với một mẫu phổ biến hơn. Tuy nhiên, khi làm như vậy, nó có thể đánh mất những chi tiết cụ thể và ngữ cảnh quan trọng để thể hiện chính xác tình trạng đặc biệt của bệnh nhân. Việc mất thông tin này có thể có những hậu quả nghiêm trọng đối với việc chăm sóc bệnh nhân và ra quyết định y tế.

Để giảm thiểu rủi ro của các hậu quả ngoài ý muốn, điều quan trọng là phải có cách tiếp cận chủ động khi thiết kế và triển khai các kỹ thuật tự phục hồi dữ liệu:

1. **Kiểm thử và Xác thực Kỹ lưỡng**: Trước khi triển khai các công cụ tự phục hồi dữ liệu vào môi trường sản xuất, việc kiểm thử và xác thực kỹ lưỡng hành vi của chúng với nhiều kịch bản khác nhau là rất quan trọng. Điều này bao gồm việc kiểm thử với các tập dữ liệu đại diện bao gồm các trường hợp ngoại lệ, biến thể và các định kiến tiềm ẩn. Kiểm thử nghiêm ngặt giúp xác định và giải quyết mọi hậu quả ngoài ý muốn trước khi chúng ảnh hưởng đến dữ liệu thực tế.
2. **Giám sát và Đánh giá Liên tục**: Việc triển khai các cơ chế giám sát và đánh giá liên tục là cần thiết để phát hiện và giảm thiểu các hậu quả ngoài ý muốn theo thời gian. Thường xuyên xem xét kết quả của quá trình tự phục hồi dữ liệu, phân tích tác động đến các hệ thống và quá trình ra quyết định phía sau, và thu thập phản hồi từ các bên liên quan có thể giúp xác định bất kỳ tác động bất lợi nào và kích hoạt các hành động khắc phục kịp thời. Nếu tổ chức của bạn có bảng điều khiển vận hành, việc thêm các chỉ số rõ ràng liên quan đến những thay đổi dữ liệu tự động có lẽ là một ý tưởng tốt. Thêm cảnh báo kết nối với những sai lệch lớn từ hoạt động thay đổi dữ liệu bình thường thậm chí còn là một ý tưởng tốt hơn!
3. **Giám sát và Can thiệp của Con người**: Duy trì sự giám sát của con người và khả năng can thiệp vào quá trình tự phục hồi dữ liệu là rất quan trọng. Mặc dù

tự động hóa có thể cải thiện đáng kể hiệu quả, việc có các chuyên gia xem xét và xác thực các sửa chữa được thực hiện bởi các mô hình AI là quan trọng, đặc biệt là trong các lĩnh vực quan trọng hoặc nhạy cảm. Phán đoán của con người và chuyên môn trong lĩnh vực có thể giúp xác định và giải quyết bất kỳ hậu quả ngoài ý muốn nào có thể phát sinh.

4. **Trí tuệ Nhân tạo Có thể Giải thích (XAI) và Tính Minh bạch**: Như đã thảo luận trong phần trước, việc kết hợp các kỹ thuật AI có thể giải thích và đảm bảo tính minh bạch trong quá trình tự phục hồi dữ liệu có thể giúp giảm thiểu các hậu quả ngoài ý muốn. Bằng cách cung cấp những giải thích rõ ràng cho việc sửa chữa dữ liệu và duy trì nhật ký kiểm toán toàn diện, các tổ chức có thể hiểu rõ hơn và theo dõi lý do đằng sau những sửa đổi được thực hiện bởi các mô hình AI.
5. **Cách tiếp cận Tăng dần và Lặp lại**: Áp dụng cách tiếp cận tăng dần và lặp lại đối với việc tự phục hồi dữ liệu có thể giúp giảm thiểu rủi ro của các hậu quả ngoài ý muốn. Thay vì áp dụng các sửa chữa tự động cho toàn bộ tập dữ liệu cùng một lúc, hãy bắt đầu với một tập con dữ liệu và dần dần mở rộng phạm vi khi các kỹ thuật chứng minh được hiệu quả và đáng tin cậy. Điều này cho phép giám sát và điều chỉnh cẩn thận trong quá trình thực hiện, giảm thiểu tác động của bất kỳ hậu quả ngoài ý muốn nào.
6. **Hợp tác và Phản hồi**: Việc thu hút các bên liên quan từ các lĩnh vực khác nhau và khuyến khích hợp tác và phản hồi trong suốt quá trình tự phục hồi dữ liệu có thể giúp xác định và giải quyết các hậu quả ngoài ý muốn. Thường xuyên tìm kiếm ý kiến từ các chuyên gia trong lĩnh vực, người sử dụng dữ liệu và người dùng cuối có thể cung cấp những hiểu biết quý giá về tác động thực tế của việc sửa chữa dữ liệu và làm nổi bật bất kỳ vấn đề nào có thể bị bỏ qua.

Bằng cách chủ động giải quyết rủi ro của các hậu quả ngoài ý muốn và triển khai các biện pháp bảo vệ thích hợp, các tổ chức có thể tận dụng lợi ích của các kỹ thuật tự phục hồi dữ liệu trong khi giảm thiểu các tác động bất lợi tiềm ẩn. Điều quan trọng là phải tiếp cận việc tự phục hồi dữ liệu như một quá trình lặp lại và hợp tác, liên tục giám sát,

đánh giá và tinh chỉnh các kỹ thuật để đảm bảo chúng phù hợp với kết quả mong muốn và duy trì tính toàn vẹn và độ tin cậy của dữ liệu.

Khi xem xét việc sử dụng các mẫu tự phục hồi dữ liệu, điều quan trọng là phải đánh giá cẩn thận những yếu tố này và cân nhắc giữa lợi ích và những rủi ro cũng như hạn chế tiềm ẩn. Trong một số trường hợp, cách tiếp cận kết hợp giữa sửa chữa tự động với sự giám sát và can thiệp của con người có thể là giải pháp phù hợp nhất.

Cũng cần lưu ý rằng các kỹ thuật tự phục hồi dữ liệu không nên được xem là sự thay thế cho các cơ chế xác thực dữ liệu, làm sạch dữ liệu đầu vào và xử lý lỗi mạnh mẽ. Những thực hành nền tảng này vẫn rất quan trọng để đảm bảo tính toàn vẹn và bảo mật của dữ liệu. Tự phục hồi dữ liệu nên được xem như một cách tiếp cận bổ sung có thể tăng cường và nâng cao các biện pháp hiện có này.

Cuối cùng, quyết định sử dụng các mẫu tự phục hồi dữ liệu phụ thuộc vào các yêu cầu, ràng buộc và ưu tiên cụ thể của ứng dụng của bạn. Bằng cách cẩn thận xem xét các cân nhắc được nêu ở trên và điều chỉnh chúng cho phù hợp với mục tiêu và kiến trúc của ứng dụng của bạn, bạn có thể đưa ra quyết định sáng suốt về thời điểm và cách tận dụng hiệu quả các kỹ thuật tự phục hồi dữ liệu.

Sinh nội dung theo ngữ cảnh

Các mẫu Sinh nội dung theo ngữ cảnh tận dụng sức mạnh của các mô hình ngôn ngữ lớn (LLM) để tạo ra nội dung động và phù hợp với ngữ cảnh trong các ứng dụng. Nhóm mẫu này nhận thấy tầm quan trọng của việc cung cấp nội dung được cá nhân hóa và phù hợp cho người dùng dựa trên nhu cầu cụ thể, sở thích, và thậm chí cả những tương tác trước đây và hiện tại của họ với ứng dụng.

> Trong ngữ cảnh của phương pháp này, "nội dung" đề cập đến cả nội dung chính (như bài blog, bài viết, v.v.) và siêu nội dung, chẳng hạn như các đề xuất cho nội dung chính.

Các mẫu Sinh nội dung theo ngữ cảnh có thể đóng vai trò quan trọng trong việc nâng cao mức độ tương tác của người dùng, cung cấp trải nghiệm được điều chỉnh riêng, và

tự động hóa các tác vụ tạo nội dung cho cả bạn và người dùng của bạn. Bằng cách sử dụng các mẫu mà chúng tôi mô tả trong chương này, bạn có thể tạo ra các ứng dụng sinh nội dung một cách động, thích ứng với ngữ cảnh và đầu vào trong thời gian thực.

Các mẫu hoạt động bằng cách tích hợp LLM vào đầu ra của ứng dụng, từ giao diện người dùng (đôi khi được gọi là "chrome"), đến email và các hình thức thông báo khác, cũng như bất kỳ quy trình sinh nội dung nào.

Khi người dùng tương tác với ứng dụng hoặc kích hoạt một yêu cầu nội dung cụ thể, ứng dụng sẽ nắm bắt ngữ cảnh liên quan, chẳng hạn như sở thích người dùng, các tương tác trước đó, hoặc các gợi ý cụ thể. Thông tin ngữ cảnh này sau đó được đưa vào LLM, cùng với bất kỳ mẫu hoặc hướng dẫn cần thiết nào và được sử dụng để tạo ra đầu ra văn bản mà nếu không sẽ phải được mã hóa cứng, lưu trữ trong cơ sở dữ liệu, hoặc được tạo ra theo thuật toán.

Nội dung được tạo ra bởi LLM có thể có nhiều hình thức khác nhau, như các đề xuất được cá nhân hóa, mô tả sản phẩm động, phản hồi email tùy chỉnh, hoặc thậm chí là toàn bộ bài viết hay bài blog. Một trong những ứng dụng táo bạo nhất của nội dung này mà tôi đã tiên phong hơn một năm trước là tự động sinh các thành phần giao diện người dùng như nhãn biểu mẫu, chú giải công cụ, và các loại văn bản giải thích khác.

Cá nhân hóa

Một trong những lợi ích chính của các mẫu Sinh nội dung theo ngữ cảnh là khả năng cung cấp trải nghiệm được cá nhân hóa cao cho người dùng. Bằng cách tạo ra nội dung dựa trên ngữ cảnh cụ thể của người dùng, những mẫu này cho phép các ứng dụng điều chỉnh nội dung theo sở thích, mối quan tâm và tương tác của từng người dùng.

Cá nhân hóa không chỉ đơn giản là chèn tên người dùng vào nội dung chung chung. Nó bao gồm việc tận dụng ngữ cảnh phong phú có sẵn về mỗi người dùng để tạo ra nội dung phù hợp với nhu cầu và mong muốn cụ thể của họ. Ngữ cảnh này có thể bao gồm nhiều yếu tố khác nhau, chẳng hạn như:

1. **Thông tin hồ sơ người dùng**: Ở mức độ tổng quát nhất của việc áp dụng kỹ thuật này, dữ liệu nhân khẩu học, sở thích, và các thuộc tính hồ sơ khác có thể được sử dụng để tạo ra nội dung phù hợp với nền tảng và đặc điểm của người dùng.
2. **Dữ liệu hành vi**: Các tương tác trước đây của người dùng với ứng dụng, như các trang đã xem, liên kết đã nhấp, hoặc sản phẩm đã mua, có thể cung cấp những hiểu biết quý giá về hành vi và sở thích của họ. Dữ liệu này có thể được sử dụng để tạo ra các đề xuất nội dung phản ánh mô hình tương tác của họ và dự đoán nhu cầu trong tương lai.
3. **Yếu tố ngữ cảnh**: Ngữ cảnh hiện tại của người dùng, như vị trí, thiết bị, thời điểm trong ngày, hoặc thậm chí thời tiết, có thể ảnh hưởng đến quá trình tạo nội dung. Ví dụ, một ứng dụng du lịch có thể có một công cụ AI có khả năng tạo ra các đề xuất được cá nhân hóa dựa trên vị trí hiện tại của người dùng và điều kiện thời tiết hiện hành.

Bằng cách tận dụng các yếu tố ngữ cảnh này, các mẫu Sinh nội dung theo ngữ cảnh cho phép ứng dụng cung cấp nội dung có cảm giác như được tạo ra riêng cho từng người dùng. Mức độ cá nhân hóa này mang lại một số lợi ích đáng kể:

1. **Tăng tương tác**: Nội dung được cá nhân hóa thu hút sự chú ý của người dùng và giữ họ tương tác với ứng dụng. Khi người dùng cảm thấy nội dung có liên quan và nói trực tiếp đến nhu cầu của họ, họ có xu hướng dành nhiều thời gian hơn để tương tác với ứng dụng và khám phá các tính năng của nó.
2. **Cải thiện sự hài lòng của người dùng**: Nội dung được cá nhân hóa cho thấy ứng dụng hiểu và quan tâm đến yêu cầu độc đáo của người dùng. Bằng cách cung cấp nội dung hữu ích, có thông tin và phù hợp với sở thích của họ, ứng dụng có thể nâng cao sự hài lòng của người dùng và xây dựng mối liên kết mạnh mẽ hơn với người dùng.
3. **Tỷ lệ chuyển đổi cao hơn**: Trong bối cảnh thương mại điện tử hoặc ứng dụng marketing, nội dung được cá nhân hóa có thể tác động đáng kể đến tỷ lệ chuyển đổi. Bằng cách giới thiệu với người dùng các sản phẩm, ưu đãi hoặc đề xuất được

điều chỉnh theo sở thích và hành vi của họ, ứng dụng có thể tăng khả năng người dùng thực hiện các hành động mong muốn, như mua hàng hoặc đăng ký dịch vụ.

Năng suất

Các mẫu Sinh nội dung theo ngữ cảnh có thể tăng đáng kể một số loại năng suất bằng cách giảm nhu cầu tạo và chỉnh sửa nội dung thủ công trong các quy trình sáng tạo. Bằng cách tận dụng sức mạnh của LLM, bạn có thể tạo ra nội dung chất lượng cao ở quy mô lớn, tiết kiệm thời gian và công sức mà các nhà sáng tạo nội dung và nhà phát triển của bạn sẽ phải bỏ ra để làm công việc thủ công tẻ nhạt.

Theo cách truyền thống, người tạo nội dung cần phải nghiên cứu, viết, biên tập và định dạng nội dung để đảm bảo đáp ứng các yêu cầu của ứng dụng và mong đợi của người dùng. Quá trình này có thể tốn nhiều thời gian và nguồn lực, đặc biệt khi khối lượng nội dung ngày càng tăng.

Tuy nhiên, với các mẫu tạo nội dung theo ngữ cảnh, quá trình tạo nội dung có thể được tự động hóa phần lớn. Các mô hình ngôn ngữ lớn có thể tạo ra nội dung mạch lạc, đúng ngữ pháp và phù hợp với ngữ cảnh dựa trên các lệnh nhắc và hướng dẫn được cung cấp. Việc tự động hóa này mang lại một số lợi ích về năng suất:

1. **Giảm công sức thủ công**: Bằng cách giao các nhiệm vụ tạo nội dung cho các mô hình ngôn ngữ lớn, người tạo nội dung có thể tập trung vào các công việc cấp cao hơn như chiến lược nội dung, phát triển ý tưởng và đảm bảo chất lượng. Họ có thể cung cấp ngữ cảnh, mẫu và hướng dẫn cần thiết cho mô hình ngôn ngữ lớn và để nó xử lý việc tạo nội dung thực tế. Điều này giảm công sức thủ công cần thiết cho việc viết và biên tập, giúp người tạo nội dung làm việc hiệu quả và năng suất hơn.
2. **Tạo nội dung nhanh hơn**: Các mô hình ngôn ngữ lớn có thể tạo nội dung nhanh hơn nhiều so với người viết. Với các lệnh nhắc và hướng dẫn phù hợp, một mô

hình ngôn ngữ lớn có thể tạo ra nhiều nội dung chỉ trong vài giây hoặc vài phút. Tốc độ này cho phép các ứng dụng tạo nội dung nhanh hơn nhiều, theo kịp nhu cầu của người dùng và bối cảnh kỹ thuật số luôn thay đổi.

> Liệu việc tạo nội dung nhanh hơn có dẫn đến tình trạng "bi kịch của công hữu" khi internet ngập tràn nội dung mà không ai đọc? Đáng buồn là tôi nghi ngờ câu trả lời là có.

3. **Tính nhất quán và chất lượng**: Các mô hình ngôn ngữ lớn có thể dễ dàng chỉnh sửa nội dung để đảm bảo tính nhất quán về phong cách, giọng điệu và chất lượng. Với các hướng dẫn và ví dụ rõ ràng, một số loại ứng dụng nhất định (như tòa soạn báo, PR, v.v.) có thể đảm bảo rằng nội dung do con người tạo ra phù hợp với giọng điệu thương hiệu và đáp ứng các tiêu chuẩn chất lượng mong muốn. Tính nhất quán này giảm nhu cầu biên tập và chỉnh sửa nhiều, tiết kiệm thời gian và công sức trong quá trình tạo nội dung.
4. **Lặp lại và tối ưu hóa**: Các mẫu tạo nội dung theo ngữ cảnh cho phép lặp lại và tối ưu hóa nội dung nhanh chóng. Bằng cách điều chỉnh các lệnh nhắc, mẫu hoặc hướng dẫn được cung cấp cho mô hình ngôn ngữ lớn, ứng dụng của bạn có thể nhanh chóng tạo ra các biến thể nội dung và thử nghiệm các cách tiếp cận khác nhau một cách tự động mà trước đây chưa từng có. Quá trình lặp lại này cho phép thử nghiệm và cải tiến chiến lược nội dung nhanh hơn, dẫn đến nội dung hiệu quả và hấp dẫn hơn theo thời gian. Kỹ thuật này có thể là một bước đột phá hoàn toàn cho các ứng dụng như thương mại điện tử, vốn sống còn dựa trên tỷ lệ thoát và mức độ tương tác

Điều quan trọng cần lưu ý là mặc dù các mẫu tạo nội dung theo ngữ cảnh có thể nâng cao đáng kể năng suất, chúng không hoàn toàn loại bỏ sự cần thiết của con người. Người tạo nội dung và biên tập viên vẫn đóng vai trò quan trọng trong việc xác định chiến lược nội dung tổng thể, cung cấp hướng dẫn cho mô hình ngôn ngữ lớn và đảm bảo chất lượng cũng như sự phù hợp của nội dung được tạo ra.

Bằng cách tự động hóa các khía cạnh lặp đi lặp lại và tốn thời gian của việc tạo nội dung, các mẫu tạo nội dung theo ngữ cảnh giải phóng thời gian và nguồn lực quý giá của con người để có thể tập trung vào các công việc có giá trị cao hơn. Năng suất tăng cao này cho phép bạn cung cấp nội dung được cá nhân hóa và hấp dẫn hơn cho người dùng trong khi tối ưu hóa quy trình tạo nội dung.

Lặp lại và thử nghiệm nhanh

Các mẫu tạo nội dung theo ngữ cảnh cho phép bạn nhanh chóng lặp lại và thử nghiệm với các biến thể nội dung khác nhau, cho phép tối ưu hóa và cải tiến chiến lược nội dung nhanh hơn. Bạn có thể tạo nhiều phiên bản nội dung chỉ trong vài giây, đơn giản bằng cách điều chỉnh ngữ cảnh, mẫu hoặc hướng dẫn được cung cấp cho mô hình.

Khả năng lặp lại nhanh này mang lại một số lợi ích chính:

1. **Thử nghiệm và tối ưu hóa**: Với khả năng tạo ra các biến thể nội dung nhanh chóng, bạn có thể dễ dàng thử nghiệm các cách tiếp cận khác nhau và đo lường hiệu quả của chúng. Ví dụ, bạn có thể tạo nhiều phiên bản mô tả sản phẩm hoặc thông điệp tiếp thị, mỗi phiên bản được điều chỉnh cho một phân khúc người dùng hoặc ngữ cảnh cụ thể. Bằng cách phân tích các chỉ số tương tác của người dùng, như tỷ lệ nhấp chuột hoặc tỷ lệ chuyển đổi, bạn có thể xác định các biến thể nội dung hiệu quả nhất và tối ưu hóa chiến lược nội dung của mình cho phù hợp.

2. **Thử nghiệm A/B**: Các mẫu tạo nội dung theo ngữ cảnh cho phép thử nghiệm A/B nội dung một cách liền mạch. Bạn có thể tạo ra hai hoặc nhiều biến thể của nội dung và phân phối ngẫu nhiên chúng cho các nhóm người dùng khác nhau. Bằng cách so sánh hiệu suất của từng biến thể, bạn có thể xác định nội dung nào thu hút đối tượng mục tiêu của mình nhất. Cách tiếp cận dựa trên dữ liệu này cho phép bạn đưa ra quyết định sáng suốt và liên tục cải thiện nội dung để tối đa hóa sự tương tác của người dùng và đạt được kết quả mong muốn.
3. **Thử nghiệm cá nhân hóa**: Lặp lại và thử nghiệm nhanh đặc biệt có giá trị khi liên quan đến việc cá nhân hóa. Với các mẫu tạo nội dung theo ngữ cảnh, bạn có thể nhanh chóng tạo ra các biến thể nội dung được cá nhân hóa dựa trên các phân khúc người dùng, sở thích hoặc hành vi khác nhau. Bằng cách thử nghiệm các chiến lược cá nhân hóa khác nhau, bạn có thể xác định các cách tiếp cận hiệu quả nhất để thu hút từng người dùng và cung cấp trải nghiệm được điều chỉnh riêng.
4. **Thích ứng với xu hướng thay đổi**: Khả năng lặp lại và thử nghiệm nhanh chóng cho phép bạn duy trì sự linh hoạt và thích ứng với xu hướng cũng như sở thích người dùng đang thay đổi. Khi các chủ đề, từ khóa mới, hoặc hành vi người dùng mới xuất hiện, bạn có thể nhanh chóng tạo ra nội dung phù hợp với những xu hướng này. Bằng cách liên tục thử nghiệm và tinh chỉnh nội dung của mình, bạn có thể duy trì sự phù hợp và giữ được lợi thế cạnh tranh trong môi trường kỹ thuật số không ngừng phát triển.
5. **Thử nghiệm hiệu quả về mặt chi phí**: Thử nghiệm nội dung theo cách truyền thống thường đòi hỏi nhiều thời gian và nguồn lực, vì người tạo nội dung cần phải thủ công phát triển và kiểm tra các biến thể khác nhau. Tuy nhiên, với các mẫu Tạo nội dung theo ngữ cảnh, chi phí thử nghiệm được giảm đáng kể. Các mô hình ngôn ngữ lớn có thể tạo ra các biến thể nội dung nhanh chóng và ở quy mô lớn, cho phép bạn khám phá nhiều ý tưởng và cách tiếp cận khác nhau mà không phát sinh chi phí đáng kể.

Để tận dụng tối đa việc lặp lại và thử nghiệm nhanh, điều quan trọng là phải có một khung thử nghiệm được xác định rõ ràng. Khung này nên bao gồm:

- Mục tiêu và giả thuyết rõ ràng cho mỗi thử nghiệm
- Các chỉ số và cơ chế theo dõi phù hợp để đo lường hiệu suất nội dung
- Chiến lược phân đoạn và nhắm mục tiêu để đảm bảo các biến thể nội dung phù hợp được phân phối đến đúng người dùng
- Công cụ phân tích và báo cáo để rút ra thông tin chi tiết từ dữ liệu thử nghiệm
- Quy trình tích hợp các bài học và tối ưu hóa vào chiến lược nội dung của bạn

Bằng cách áp dụng lặp lại và thử nghiệm nhanh, bạn có thể liên tục tinh chỉnh và tối ưu hóa nội dung của mình, đảm bảo rằng nó luôn hấp dẫn, phù hợp và hiệu quả trong việc đạt được mục tiêu của ứng dụng. Cách tiếp cận linh hoạt này trong việc tạo nội dung cho phép bạn luôn đi đầu và mang lại trải nghiệm người dùng xuất sắc.

Khả năng mở rộng và hiệu quả

Khi các ứng dụng phát triển và nhu cầu về nội dung cá nhân hóa tăng lên, các mẫu tạo nội dung theo ngữ cảnh cho phép mở rộng hiệu quả việc tạo nội dung. Các mô hình ngôn ngữ lớn có thể tạo nội dung cho số lượng lớn người dùng và ngữ cảnh cùng một lúc mà không cần tăng tương ứng nguồn nhân lực. Khả năng mở rộng này cho phép các ứng dụng cung cấp trải nghiệm cá nhân hóa cho cơ sở người dùng ngày càng tăng mà không gây căng thẳng cho khả năng tạo nội dung của họ.

Lưu ý rằng việc tạo nội dung theo ngữ cảnh có thể được sử dụng hiệu quả để quốc tế hóa ứng dụng của bạn “ngay lập tức”. Thực tế, đó chính xác là những gì tôi đã làm bằng cách sử dụng Instant18n Gem của mình để cung cấp Olympia bằng hơn nửa tá ngôn ngữ, mặc dù chúng tôi chưa đầy một năm tuổi.

Bản địa hóa được hỗ trợ bởi AI

Nếu bạn cho phép tôi khoe khoang một chút, tôi nghĩ rằng thư viện Instant18n của tôi dành cho các ứng dụng Rails là một ví dụ đột phá về mẫu "Tạo nội dung theo ngữ cảnh" trong thực tế, thể hiện tiềm năng chuyển đổi của AI trong phát triển ứng dụng. Gem này tận dụng sức mạnh của mô hình ngôn ngữ lớn GPT của OpenAI để cách mạng hóa cách xử lý quốc tế hóa và bản địa hóa trong các ứng dụng Rails.

Theo truyền thống, việc quốc tế hóa một ứng dụng Rails liên quan đến việc xác định thủ công các khóa dịch và cung cấp bản dịch tương ứng cho mỗi ngôn ngữ được hỗ trợ. Quá trình này có thể tốn thời gian, tốn nhiều nguồn lực và dễ gây ra sự không nhất quán. Tuy nhiên, với gem Instant18n, mô hình bản địa hóa được định nghĩa lại hoàn toàn.

Bằng cách tích hợp mô hình ngôn ngữ lớn, gem Instant18n cho phép bạn tạo bản dịch ngay lập tức, dựa trên ngữ cảnh và ý nghĩa của văn bản. Thay vì dựa vào các khóa dịch được xác định trước và bản dịch tĩnh, gem này dịch động văn bản bằng sức mạnh của AI. Cách tiếp cận này mang lại một số lợi ích chính:

1. **Bản địa hóa liền mạch**: Với gem Instant18n, các nhà phát triển không còn cần phải thủ công định nghĩa và duy trì các tệp dịch cho mỗi ngôn ngữ được hỗ trợ. Gem tự động tạo bản dịch dựa trên văn bản được cung cấp và ngôn ngữ đích mong muốn, làm cho quá trình bản địa hóa trở nên dễ dàng và liền mạch.
2. **Độ chính xác theo ngữ cảnh**: AI có thể được cung cấp đủ ngữ cảnh để hiểu được các sắc thái của văn bản được dịch. Nó có thể xem xét ngữ cảnh xung quanh, thành ngữ và tham chiếu văn hóa để tạo ra bản dịch chính xác, tự nhiên và phù hợp với ngữ cảnh.
3. **Hỗ trợ ngôn ngữ rộng rãi**: Gem Instant18n tận dụng kiến thức rộng lớn và khả năng ngôn ngữ của GPT, cho phép dịch sang nhiều ngôn ngữ khác nhau. Từ các ngôn ngữ phổ biến như tiếng Tây Ban Nha và tiếng Pháp đến các ngôn ngữ ít phổ biến hơn hoặc ngôn ngữ hư cấu như tiếng Klingon và tiếng Elf, gem có thể xử lý nhiều yêu cầu dịch thuật khác nhau.

4. **Tính linh hoạt và sáng tạo**: Gem này vượt xa khỏi các bản dịch ngôn ngữ truyền thống và cho phép các tùy chọn bản địa hóa sáng tạo và không quy ước. Các nhà phát triển có thể dịch văn bản theo nhiều phong cách, phương ngữ, hoặc thậm chí ngôn ngữ hư cấu khác nhau, mở ra những khả năng mới cho trải nghiệm người dùng độc đáo và nội dung hấp dẫn.
5. **Tối ưu hóa hiệu suất**: Gem Instant18n tích hợp các cơ chế bộ nhớ đệm để cải thiện hiệu suất và giảm thiểu chi phí của các bản dịch lặp lại. Văn bản đã dịch được lưu vào bộ nhớ đệm, cho phép các yêu cầu tiếp theo đối với cùng một bản dịch được phục vụ nhanh chóng mà không cần gọi API dư thừa.

Gem Instant18n là một ví dụ điển hình về sức mạnh của mẫu "Tạo nội dung theo ngữ cảnh" bằng cách tận dụng AI để tạo nội dung bản địa hóa một cách động. Nó cho thấy cách AI có thể được tích hợp vào chức năng cốt lõi của một ứng dụng Rails, chuyển đổi cách các nhà phát triển tiếp cận quốc tế hóa và bản địa hóa.

Bằng cách loại bỏ nhu cầu quản lý dịch thuật thủ công và cho phép dịch tức thì dựa trên ngữ cảnh, gem Instant18n giúp các nhà phát triển tiết kiệm đáng kể thời gian và công sức. Nó cho phép họ tập trung vào việc xây dựng các tính năng cốt lõi của ứng dụng trong khi đảm bảo khía cạnh bản địa hóa được xử lý một cách mượt mà và chính xác.

Tầm Quan Trọng của Kiểm Thử Người Dùng và Phản Hồi

Cuối cùng, luôn ghi nhớ tầm quan trọng của việc kiểm thử người dùng và phản hồi. Việc xác nhận rằng nội dung theo ngữ cảnh được tạo ra đáp ứng kỳ vọng của người dùng và phù hợp với mục tiêu của ứng dụng là điều cực kỳ quan trọng. Liên tục cải tiến và tinh chỉnh nội dung được tạo ra dựa trên hiểu biết từ người dùng và phân tích. Nếu bạn đang tạo nội dung động ở quy mô lớn mà việc kiểm tra thủ công bởi bạn và nhóm của bạn là bất khả thi, hãy cân nhắc việc thêm các cơ chế phản hồi cho phép người dùng báo cáo nội dung kỳ lạ hoặc sai, cùng với giải thích lý do tại sao. Những phản hồi quý

giá đó thậm chí có thể được đưa cho một worker AI có nhiệm vụ điều chỉnh thành phần đã tạo ra nội dung!

Giao diện người dùng sinh thành

Sự chú ý ngày nay là một tài nguyên quý giá đến mức việc thu hút người dùng hiệu quả đòi hỏi những trải nghiệm phần mềm không chỉ mượt mà và trực quan mà còn phải được cá nhân hóa cao độ theo nhu cầu, sở thích và bối cảnh của từng cá nhân. Do đó, các nhà thiết kế và phát triển ngày càng phải đối mặt với thách thức tạo ra các giao diện người dùng! có khả năng thích ứng và đáp ứng các yêu cầu độc đáo của từng người dùng *ở quy mô lớn.*

Giao diện người dùng sinh thành (GenUI) là một cách tiếp cận mang tính cách mạng đối với thiết kế giao diện người dùng! bằng cách tận dụng sức mạnh của các mô hình ngôn ngữ lớn (LLMs) để tạo ra những trải nghiệm người dùng được cá nhân hóa cao và linh hoạt một cách tức thời. Tôi muốn đảm bảo ít nhất cung cấp cho bạn một cái nhìn tổng quan về GenUI trong cuốn sách này, bởi vì tôi tin rằng đây là một trong những cơ hội mới mẻ nhất hiện đang tồn tại trong lĩnh vực thiết kế và framework ứng dụng. Tôi tin chắc rằng sẽ có hàng chục hoặc nhiều hơn nữa các dự án thương mại và mã nguồn

mở thành công sẽ xuất hiện trong lĩnh vực đặc thù này.

Về cốt lõi, GenUI kết hợp các nguyên tắc của Sinh nội dung theo ngữ cảnh với các kỹ thuật AI tiên tiến để tạo ra các thành phần giao diện người dùng như văn bản, hình ảnh và bố cục một cách linh động dựa trên sự hiểu biết sâu sắc về ngữ cảnh, sở thích và mục tiêu của người dùng. GenUI cho phép các nhà thiết kế và phát triển tạo ra các giao diện có thể thích ứng và phát triển để đáp ứng với tương tác của người dùng, mang lại mức độ cá nhân hóa mà trước đây không thể đạt được.

GenUI đại diện cho một sự thay đổi căn bản trong cách chúng ta tiếp cận thiết kế giao diện người dùng. Thay vì thiết kế cho số đông, GenUI cho phép chúng ta thiết kế cho từng cá nhân. Nội dung và giao diện được cá nhân hóa có tiềm năng tạo ra những trải nghiệm người dùng có sự đồng cảm sâu sắc với từng người dùng, tăng cường sự tham gia, sự hài lòng và lòng trung thành.

Là một kỹ thuật tiên phong, việc chuyển đổi sang GenUI đầy những thách thức về mặt khái niệm và thực tiễn. Việc tích hợp AI vào quy trình thiết kế, đảm bảo các giao diện được tạo ra không chỉ được cá nhân hóa mà còn phải dễ sử dụng, dễ tiếp cận và phù hợp với thương hiệu cũng như trải nghiệm người dùng tổng thể, tất cả những thách thức này khiến GenUI trở thành một lĩnh vực chỉ dành cho số ít, không phải số đông. Ngoài ra, sự tham gia của AI cũng đặt ra những câu hỏi về quyền riêng tư dữ liệu, tính minh bạch và thậm chí cả những ẩn ý về mặt đạo đức

Mặc dù có những thách thức, trải nghiệm cá nhân hóa ở quy mô lớn có sức mạnh để hoàn toàn thay đổi cách chúng ta tương tác với các sản phẩm và dịch vụ số. Nó mở ra khả năng tạo ra các giao diện toàn diện và dễ tiếp cận phục vụ cho những nhu cầu đa dạng của người dùng, bất kể khả năng, xuất thân hay sở thích của họ.

Trong chương này, chúng ta sẽ khám phá khái niệm GenUI, xem xét một số đặc điểm định nghĩa, lợi ích chính và những thách thức tiềm ẩn. Chúng ta bắt đầu bằng việc xem xét hình thức cơ bản và dễ tiếp cận nhất của GenUI: tạo nội dung văn bản cho các giao diện người dùng được thiết kế và triển khai theo cách truyền thống.

Tạo nội dung cho Giao diện người dùng

Các thành phần văn bản tồn tại trong thành phần giao diện của ứng dụng của bạn, như nhãn biểu mẫu, gợi ý công cụ và văn bản giải thích, thường được mã hóa cứng trong các mẫu hoặc thành phần UI, cung cấp một trải nghiệm nhất quán nhưng chung chung cho tất cả người dùng. Sử dụng các mẫu sinh nội dung theo ngữ cảnh, bạn có thể chuyển đổi những thành phần tĩnh này thành các thành phần động, nhận biết ngữ cảnh và được cá nhân hóa.

Biểu mẫu cá nhân hóa

Biểu mẫu là một phần không thể thiếu trong các ứng dụng web và di động, đóng vai trò là phương tiện chính để thu thập thông tin từ người dùng. Tuy nhiên, các biểu mẫu truyền thống thường mang lại trải nghiệm chung chung và không cá nhân hóa, với các nhãn và trường tiêu chuẩn không phải lúc nào cũng phù hợp với ngữ cảnh hoặc nhu cầu cụ thể của người dùng. Người dùng có xu hướng hoàn thành các biểu mẫu phù hợp với nhu cầu và sở thích của họ nhiều hơn, dẫn đến tỷ lệ chuyển đổi và sự hài lòng của người dùng cao hơn.

Tuy nhiên, điều quan trọng là phải cân bằng giữa cá nhân hóa và tính nhất quán. Mặc dù việc điều chỉnh biểu mẫu cho từng người dùng có thể có lợi, nhưng điều quan trọng là phải duy trì một mức độ quen thuộc và dự đoán được. Người dùng vẫn phải có thể nhận ra và điều hướng biểu mẫu một cách dễ dàng, ngay cả khi có các thành phần được cá nhân hóa.

Dưới đây là một số ý tưởng về biểu mẫu cá nhân hóa để lấy cảm hứng:

Gợi ý trường theo ngữ cảnh

GenUI có thể phân tích các tương tác trước đây, sở thích và dữ liệu của người dùng để đưa ra các gợi ý trường thông minh như dự đoán. Ví dụ, nếu người dùng đã từng nhập

địa chỉ giao hàng của họ, biểu mẫu có thể tự động điền các trường liên quan với thông tin đã lưu của họ. Điều này không chỉ tiết kiệm thời gian mà còn cho thấy ứng dụng hiểu và ghi nhớ sở thích của người dùng.

Khoan đã, chẳng phải kỹ thuật này có thể thực hiện được mà không cần đến AI sao? Tất nhiên là được, nhưng cái hay của việc sử dụng AI để điều khiển chức năng này nằm ở hai điểm: 1) việc triển khai có thể trở nên dễ dàng như thế nào và 2) nó có thể linh hoạt ra sao khi UI của bạn thay đổi và phát triển theo thời gian.

Hãy cùng tạo ra một service cho hệ thống xử lý đơn hàng lý thuyết của chúng ta, service này sẽ cố gắng tự động điền địa chỉ giao hàng phù hợp cho người dùng.

```
class OrderShippingAddressSubscriber
  include Raix::ChatCompletion

  attr_accessor :order

  delegate :customer, to: :order

  DIRECTIVE = "You are a smart order processing assistant. Given the
  customer's order history, guess the most likely shipping address
  for the current order."

  def order_created(order)
    return unless order.pending? && order.shipping_address.blank?

    self.order = order

    transcript.clear
    transcript << { system: DIRECTIVE }
    transcript << { user: "Order History: #{order_history.to_json}" }
    transcript << { user: "Current Order: #{order.to_json}" }

    response = chat_completion
    apply_predicted_shipping_address(order, response)
  end

  private

  def apply_predicted_shipping_address(order, response)
```

```
    # extract the shipping address from the response...
    # ...and assume there's some sort of live update of the address fields
    order.update(shipping_address:)
  end

  def order_history
    customer.orders.successful.limit(100).map do |order|
      {
        date: order.date,
        description: order.description,
        shipping_address: order.shipping_address
      }
    end
  end
end
```

Ví dụ này được đơn giản hóa rất nhiều, nhưng sẽ phù hợp với hầu hết các trường hợp. Ý tưởng là để cho AI đưa ra phỏng đoán giống như cách con người vẫn làm. Để làm rõ điều tôi đang nói đến, hãy cùng xem xét một số dữ liệu mẫu:

```
Order History:
[
  {"date": "2024-01-03", "description": "garden soil mix",
   "shipping_address": "123 Country Lane, Rural Town"},
  {"date": "2024-01-15", "description": "hardcover fiction novels",
   "shipping_address": "456 City Apt, Metroville"},
  {"date": "2024-01-22", "description": "baby diapers", "shipping_address":
   "789 Suburb St, Quietville"},
  {"date": "2024-02-01", "description": "organic vegetables",
   "shipping_address": "123 Country Lane, Rural Town"},
  {"date": "2024-02-17", "description": "mystery thriller book set",
   "shipping_address": "456 City Apt, Metroville"},
  {"date": "2024-02-25", "description": "baby wipes",
   "shipping_address": "789 Suburb St, Quietville"},
  {"date": "2024-03-05", "description": "flower seeds",
   "shipping_address": "123 Country Lane, Rural Town"},
  {"date": "2024-03-20", "description": "biographies",
   "shipping_address": "456 City Apt, Metroville"},
  {"date": "2024-03-30", "description": "baby formula",
   "shipping_address": "789 Suburb St, Quietville"},
```

```
  {"date": "2024-04-12", "description": "lawn fertilizer",
   "shipping_address": "123 Country Lane, Rural Town"},
  {"date": "2024-04-22", "description": "science fiction novels",
   "shipping_address": "456 City Apt, Metroville"},
  {"date": "2024-05-02", "description": "infant toys",
   "shipping_address": "789 Suburb St, Quietville"},
  {"date": "2024-05-14", "description": "outdoor grill",
   "shipping_address": "123 Country Lane, Rural Town"},
  {"date": "2024-05-29", "description": "literary classics",
  "shipping_address": "456 City Apt, Metroville"},
  {"date": "2024-06-11", "description": "baby clothes",
   "shipping_address": "789 Suburb St, Quietville"},
  {"date": "2024-07-01", "description": "watering can",
   "shipping_address": "123 Country Lane, Rural Town"},
  {"date": "2024-07-18", "description": "non-fiction essays",
  "shipping_address": "456 City Apt, Metroville"},
  {"date": "2024-07-28", "description": "baby bath items",
  "shipping_address": "789 Suburb St, Quietville"},
  {"date": "2024-08-09", "description": "herb garden kit",
  "shipping_address": "123 Country Lane, Rural Town"},
  {"date": "2024-08-24", "description": "children's books",
  "shipping_address": "456 City Apt, Metroville"}
]
```

Bạn có nhận ra mẫu hình trong dữ liệu không? Tôi đảm bảo với bạn đây là điều rất đơn giản đối với một LLM. Để minh họa, hãy hỏi GPT-4 xem địa chỉ giao hàng nào là phù hợp nhất cho một "nhiệt kế".

```
From the order history you've provided, it looks like the purchases are
generally clustered into three main types based on the shipping addresses:

123 Country Lane, Rural Town - This address often orders garden and
outdoor-related items like soil mix, vegetables, flower seeds, lawn
fertilizer, an outdoor grill, a watering can, and an herb garden kit.

456 City Apt, Metroville - This address tends to order various types of
books, indicating an interest in reading across genres like fiction,
thrillers, biographies, science fiction, literary classics, non-fiction
essays, and children's books.

789 Suburb St, Quietville - The purchases for this address are consistently
baby-related products, such as diapers, wipes, baby formula, infant toys,
baby clothes, and baby bath items.

Given these patterns, a digital thermometer could potentially be a fit for
any of the addresses based on different needs: outdoor temperature tracking,
a tool for a passionate reader interested in science gadgets, or a health
tool for a baby. However, considering its common use in monitoring health,
especially for babies, it's most likely to be purchased by someone who's
already buying baby care products.

Therefore, the most likely shipping address for a digital thermometer, based
on the purchasing patterns, would be 789 Suburb St, Quietville.
```

Nếu bạn đang nghĩ rằng việc sử dụng một mô hình đắt tiền như GPT-4 cho nhiệm vụ này là quá mức cần thiết, bạn đã đúng! Tôi đã thử cùng một prompt trên Mistral 7B Instruct và nó đã tạo ra phản hồi sau với tốc độ 75 token mỗi giây, với chi phí cực kỳ thấp chỉ 0,000218 USD.

```
Based on the given order history, it appears that the customer has received
orders for baby-related items (diapers, wipes, formula, clothes, bath items)
at the address "789 Suburb St, Quietville" and garden-related items (soil,
seeds, watering can, herb garden kit) at the address "123 Country Lane, Rural
Town". The customer has also received books (fiction, non-fiction, literary
classics, mystery-thriller, science fiction, biographies, children's books)
at the address "456 City Apt, Metroville".

Given that the new order is for a "thermometer", which doesn't seem to fit
into any of the categories that the customer has previously ordered, it's
hard to accurately predict the shipping address based solely on the order
history. However, given the 50%-50% split between baby-related and
garden-related items, it could somewhat lean towards the Baby-related items
address ("789 Suburb St, Quietville"). But remember, this is an assumption
and cannot be definitively confirmed without more context or information.
```

Chi phí và công sức của kỹ thuật này có xứng đáng để tạo ra trải nghiệm thanh toán kỳ diệu hơn không? Đối với nhiều nhà bán lẻ trực tuyến, câu trả lời là hoàn toàn có. Và theo những gì chúng ta thấy, chi phí tính toán AI sẽ chỉ giảm dần, đặc biệt là đối với các nhà cung cấp dịch vụ lưu trữ mô hình mã nguồn mở trong cuộc đua về giá.

Sử dụng Mẫu Prompt và StructuredIO cùng với Response Fencing để tối ưu hóa kiểu hoàn thành hội thoại này.

Sắp xếp trường thích ứng

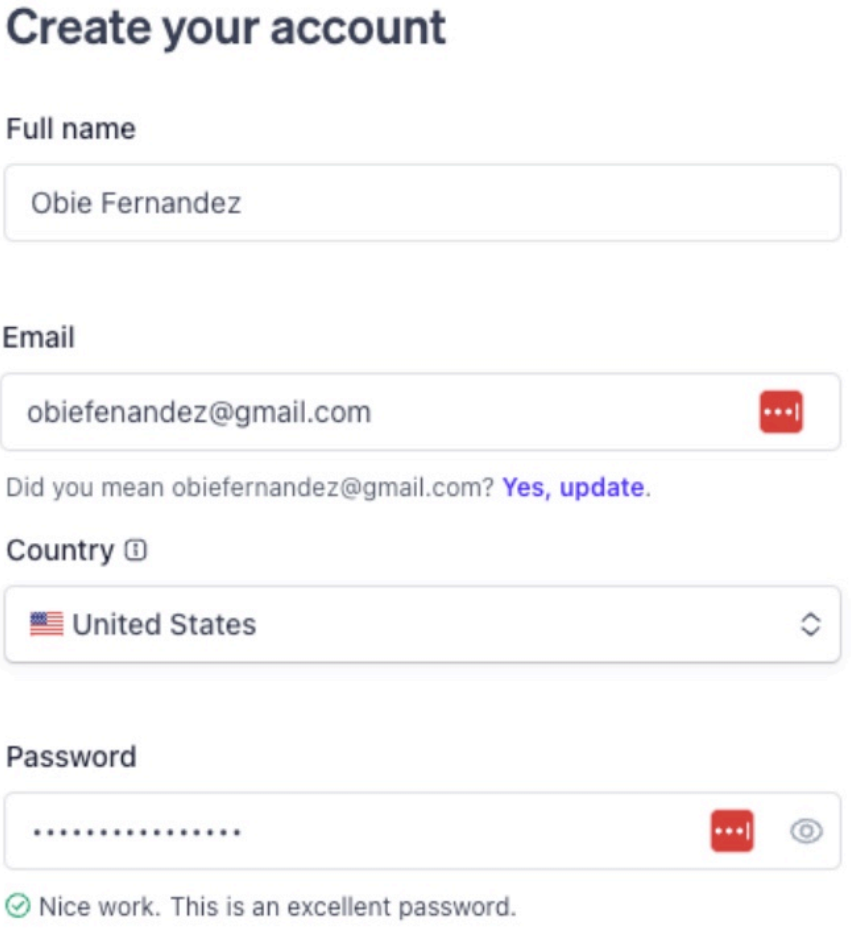
Thứ tự hiển thị các trường trong biểu mẫu có thể ảnh hưởng đáng kể đến trải nghiệm người dùng và tỷ lệ hoàn thành. Với GenUI, bạn có thể điều chỉnh thứ tự trường một cách linh hoạt dựa trên ngữ cảnh của người dùng và mức độ quan trọng của từng trường. Ví dụ, nếu người dùng đang điền vào biểu mẫu đăng ký cho một ứng dụng thể dục, biểu mẫu có thể ưu tiên các trường liên quan đến mục tiêu và sở thích tập luyện của họ, làm cho quá trình này trở nên phù hợp và hấp dẫn hơn.

Vi bản sao cá nhân hóa

Văn bản hướng dẫn, thông báo lỗi và các vi bản sao khác liên quan đến biểu mẫu cũng có thể được cá nhân hóa bằng GenUI. Thay vì hiển thị thông báo lỗi chung chung như "Địa chỉ email không hợp lệ," bạn có thể tạo ra những thông điệp hữu ích và phù hợp với ngữ cảnh hơn như "Vui lòng nhập địa chỉ email hợp lệ để nhận xác nhận đơn hàng của bạn." Những chi tiết cá nhân hóa này có thể làm cho trải nghiệm điền biểu mẫu thân thiện hơn và ít gây bực bội hơn.

Xác thực cá nhân hóa

Tương tự như Vi bản sao cá nhân hóa, bạn có thể sử dụng AI để xác thực biểu mẫu theo những cách có vẻ kỳ diệu. Hãy tưởng tượng để AI xác thực biểu mẫu hồ sơ người dùng, tìm kiếm những sai sót tiềm ẩn ở cấp độ *ngữ nghĩa*.



Hình 9. Bạn có thể nhận ra việc xác thực ngữ nghĩa đang diễn ra không?

Hiển thị tiến trình

GenUI có thể thông minh xác định những trường biểu mẫu nào là thiết yếu dựa trên ngữ cảnh của người dùng và dần dần hiện thêm các trường bổ sung khi cần thiết. Kỹ thuật hiển thị tiến trình này giúp giảm tải nhận thức và làm cho quá trình điền biểu mẫu dễ quản lý hơn. Ví dụ, nếu người dùng đang đăng ký gói dịch vụ cơ bản, biểu mẫu

ban đầu chỉ hiển thị các trường thiết yếu, và khi người dùng tiến triển hoặc chọn các tùy chọn cụ thể, các trường liên quan bổ sung có thể được đưa vào một cách linh động.

Văn bản giải thích theo ngữ cảnh

Chú giải công cụ thường được sử dụng để cung cấp thông tin bổ sung hoặc hướng dẫn cho người dùng khi họ di chuột qua hoặc tương tác với các thành phần cụ thể. Với cách tiếp cận "Sinh nội dung theo ngữ cảnh", bạn có thể tạo ra các chú giải công cụ thích ứng với ngữ cảnh của người dùng và cung cấp thông tin phù hợp. Ví dụ, nếu người dùng đang khám phá một tính năng phức tạp, chú giải công cụ có thể đưa ra các mẹo hoặc ví dụ được cá nhân hóa dựa trên những tương tác trước đó hoặc trình độ của họ.

Văn bản giải thích, như hướng dẫn, mô tả, hoặc thông điệp trợ giúp, có thể được tạo ra một cách linh động dựa trên ngữ cảnh của người dùng. Thay vì trình bày những giải thích chung chung, bạn có thể sử dụng các mô hình ngôn ngữ lớn để tạo ra văn bản được điều chỉnh cho phù hợp với nhu cầu hoặc câu hỏi cụ thể của người dùng. Ví dụ, nếu người dùng đang gặp khó khăn với một bước cụ thể trong quy trình, văn bản giải thích có thể cung cấp hướng dẫn hoặc mẹo xử lý sự cố được cá nhân hóa.

Vi bản sao đề cập đến những đoạn văn bản ngắn hướng dẫn người dùng qua ứng dụng của bạn, như nhãn nút, thông báo lỗi, hoặc lời nhắc xác nhận. Bằng cách áp dụng cách tiếp cận Sinh nội dung theo ngữ cảnh cho vi bản sao, bạn có thể tạo ra một giao diện người dùng thích ứng phản hồi với hành động của người dùng và cung cấp văn bản hữu ích và phù hợp. Ví dụ, nếu người dùng sắp thực hiện một hành động quan trọng, lời nhắc xác nhận có thể được tạo ra một cách linh động để cung cấp thông điệp rõ ràng và được cá nhân hóa.

Văn bản giải thích và chú giải công cụ được cá nhân hóa có thể cải thiện đáng kể quá trình làm quen cho người dùng mới. Bằng cách cung cấp hướng dẫn và ví dụ phù hợp với ngữ cảnh, bạn có thể giúp người dùng nhanh chóng hiểu và điều hướng ứng dụng, giảm thời gian học và tăng tỷ lệ áp dụng.

Các thành phần giao diện linh động và nhạy cảm với ngữ cảnh cũng có thể làm cho ứng dụng trở nên trực quan và hấp dẫn hơn. Người dùng có nhiều khả năng tương tác và khám phá các tính năng hơn khi văn bản đi kèm được điều chỉnh cho phù hợp với nhu cầu và sở thích cụ thể của họ.

Cho đến nay chúng ta đã tìm hiểu những ý tưởng để cải thiện các mô hình giao diện người dùng hiện có với AI, nhưng còn việc xem xét lại cách thiết kế và triển khai giao diện người dùng theo một cách triệt để hơn thì sao?

Định nghĩa Giao diện Sinh thành

Khác với thiết kế giao diện người dùng truyền thống, nơi các nhà thiết kế tạo ra các giao diện cố định, tĩnh, GenUI gợi ý về một tương lai trong đó phần mềm của chúng ta có những trải nghiệm linh hoạt, được cá nhân hóa có thể phát triển và thích ứng trong thời gian thực. Mỗi khi chúng ta sử dụng giao diện hội thoại được điều khiển bởi AI, chúng ta đang để AI thích ứng với nhu cầu cụ thể của người dùng. GenUI tiến thêm một bước nữa bằng cách áp dụng mức độ thích ứng đó vào giao diện *trực quan* của phần mềm.

Lý do mà chúng ta có thể thử nghiệm các ý tưởng GenUI ngày nay là vì các mô hình ngôn ngữ lớn. đã hiểu về lập trình và kiến thức cơ bản của chúng bao gồm các công nghệ và framework giao diện người dùng. Vấn đề đặt ra là liệu các mô hình ngôn ngữ lớn có thể được sử dụng để tạo ra các thành phần giao diện người dùng, như văn bản, hình ảnh, bố cục, và thậm chí toàn bộ giao diện, được điều chỉnh cho từng người dùng cụ thể hay không. Mô hình có thể được hướng dẫn để xem xét các yếu tố khác nhau, như tương tác trước đây của người dùng, sở thích đã nêu, thông tin nhân khẩu học, và ngữ cảnh sử dụng hiện tại, để tạo ra các giao diện được cá nhân hóa cao và phù hợp.

GenUI khác với thiết kế giao diện người dùng truyền thống ở một số điểm chính:

1. **Động và Thích ứng**: Thiết kế giao diện người dùng truyền thống liên quan đến việc tạo ra các giao diện cố định, tĩnh giống nhau cho tất cả người dùng. Ngược lại, GenUI cho phép các giao diện có thể thích ứng và thay đổi động dựa trên nhu cầu và ngữ cảnh của người dùng. Điều này có nghĩa là cùng một ứng dụng có thể hiển thị các giao diện khác nhau cho những người dùng khác nhau hoặc thậm chí cho cùng một người dùng trong các tình huống khác nhau.
2. **Cá nhân hóa ở Quy mô lớn**: Với thiết kế truyền thống, việc tạo ra trải nghiệm cá nhân hóa cho từng người dùng thường không khả thi do thời gian và nguồn lực cần thiết. Ngược lại, GenUI cho phép cá nhân hóa ở quy mô lớn. Bằng cách tận dụng AI, các nhà thiết kế có thể tạo ra các giao diện tự động thích ứng với nhu cầu và sở thích độc đáo của từng người dùng, mà không cần phải thiết kế và phát triển thủ công các giao diện riêng biệt cho từng phân khúc người dùng.
3. **Tập trung vào Kết quả**: Thiết kế giao diện người dùng truyền thống thường tập trung vào việc tạo ra các giao diện đẹp mắt và chức năng. Mặc dù những khía cạnh này vẫn quan trọng trong GenUI, trọng tâm chính chuyển sang việc đạt được kết quả mong muốn của người dùng. GenUI nhằm tạo ra các giao diện được tối ưu hóa cho các mục tiêu và nhiệm vụ cụ thể của từng người dùng, ưu tiên khả năng sử dụng và hiệu quả hơn là các cân nhắc thuần túy về thẩm mỹ.
4. **Học tập và Cải thiện Liên tục**: Các hệ thống GenUI có thể học tập và cải thiện liên tục theo thời gian dựa trên tương tác và phản hồi của người dùng. Khi người dùng tương tác với các giao diện được tạo ra, các mô hình AI có thể thu thập dữ liệu về hành vi, sở thích và kết quả của người dùng, sử dụng thông tin này để tinh chỉnh và tối ưu hóa việc tạo giao diện trong tương lai. Quá trình học tập lặp đi lặp lại này cho phép các hệ thống GenUI ngày càng hiệu quả hơn trong việc đáp ứng nhu cầu của người dùng theo thời gian.

Điều quan trọng cần lưu ý là GenUI không giống như các công cụ thiết kế có hỗ trợ AI, chẳng hạn như những công cụ đưa ra gợi ý hoặc tự động hóa một số tác vụ thiết kế nhất định. Mặc dù những công cụ này có thể hữu ích trong việc tối ưu hóa quy trình thiết kế, chúng vẫn phụ thuộc vào các nhà thiết kế để đưa ra quyết định cuối cùng và tạo ra

các giao diện tĩnh. Ngược lại, GenUI liên quan đến việc hệ thống AI đóng vai trò tích cực hơn trong việc tạo ra và thích ứng giao diện dựa trên dữ liệu và ngữ cảnh của người dùng.

GenUI đại diện cho một sự thay đổi đáng kể trong cách chúng ta tiếp cận thiết kế giao diện người dùng, chuyển từ giải pháp một-kích-cỡ-phù-hợp-tất-cả sang những trải nghiệm được cá nhân hóa cao, có khả năng thích ứng. Bằng cách tận dụng sức mạnh của AI, GenUI có tiềm năng cách mạng hóa cách chúng ta tương tác với các sản phẩm và dịch vụ số, tạo ra các giao diện trực quan, hấp dẫn và hiệu quả hơn cho từng người dùng cụ thể.

Ví dụ

Để minh họa khái niệm GenUI, hãy xem xét một ứng dụng thể dục giả định có tên là "FitAI". Ứng dụng này nhằm cung cấp các kế hoạch tập luyện và lời khuyên dinh dưỡng được cá nhân hóa cho người dùng dựa trên mục tiêu, mức độ thể chất và sở thích cá nhân của họ.

Trong cách tiếp cận thiết kế giao diện người dùng truyền thống, FitAI có thể có một tập hợp cố định các màn hình và thành phần giống nhau cho tất cả người dùng. Tuy nhiên, với GenUI, giao diện của ứng dụng có thể thích ứng động với nhu cầu và ngữ cảnh độc đáo của từng người dùng.

Cách tiếp cận này khá khó tưởng tượng để triển khai vào năm 2024 và thậm chí có thể không có ROI đầy đủ, nhưng nó là điều có thể.

Đây là cách nó có thể hoạt động:

1. **Khởi động**:
    - Thay vì một bảng câu hỏi tiêu chuẩn, FitAI sử dụng AI hội thoại để thu thập thông tin về mục tiêu, mức độ thể chất hiện tại và sở thích của người dùng.

- Dựa trên tương tác ban đầu này, AI tạo ra bố cục bảng điều khiển được cá nhân hóa, làm nổi bật các tính năng và thông tin liên quan nhất đến mục tiêu của người dùng.
- Công nghệ AI hiện tại có thể có một tập hợp các thành phần màn hình để sử dụng trong việc tạo bảng điều khiển được cá nhân hóa.
- Công nghệ AI trong tương lai có thể đảm nhận vai trò của một nhà thiết kế giao diện người dùng có kinh nghiệm và thực sự tạo ra bảng điều khiển *từ đầu*.

2. **Trình lập kế hoạch tập luyện**:

- Giao diện trình lập kế hoạch tập luyện được AI điều chỉnh để phù hợp với trình độ và thiết bị sẵn có của người dùng.
- Đối với người mới bắt đầu không có thiết bị, nó có thể hiển thị các bài tập đơn giản chỉ sử dụng trọng lượng cơ thể kèm theo hướng dẫn chi tiết và video.
- Đối với người dùng nâng cao có thể tiếp cận phòng tập, nó có thể hiển thị các bài tập phức tạp hơn với ít nội dung giải thích hơn.
- Nội dung của trình lập kế hoạch tập luyện không đơn thuần được lọc từ một tập hợp lớn. Nó có thể được tạo ra *ngay lập tức* dựa trên cơ sở kiến thức được truy vấn với ngữ cảnh bao gồm mọi thông tin đã biết về người dùng.

3. **Theo dõi tiến độ**:

- Giao diện theo dõi tiến độ phát triển dựa trên mục tiêu và mô hình tương tác của người dùng.
- Nếu người dùng chủ yếu tập trung vào giảm cân, giao diện có thể nổi bật hiển thị biểu đồ xu hướng cân nặng và thống kê lượng calo đã đốt cháy.
- Đối với người dùng đang tập để tăng cơ bắp, nó có thể nhấn mạnh vào sự tăng trưởng sức mạnh và những thay đổi về thành phần cơ thể.

- AI có thể điều chỉnh phần này của ứng dụng theo tiến độ thực tế của người dùng. Nếu tiến độ dừng lại trong một khoảng thời gian, ứng dụng có thể chuyển sang chế độ cố gắng thuyết phục người dùng tiết lộ lý do của sự trì trệ, để từ đó có biện pháp khắc phục.

4. **Tư vấn dinh dưỡng**:

    - Phần dinh dưỡng thích ứng với sở thích và hạn chế ăn uống của người dùng.
    - Đối với người dùng ăn chay, nó có thể hiển thị các gợi ý bữa ăn và nguồn protein từ thực vật.
    - Đối với người dùng không dung nạp được gluten, nó sẽ tự động loại bỏ các thực phẩm có chứa gluten khỏi danh sách đề xuất.
    - Một lần nữa, nội dung không được rút ra từ một tập hợp lớn dữ liệu bữa ăn áp dụng cho tất cả người dùng, mà được tổng hợp từ một cơ sở kiến thức chứa thông tin có thể điều chỉnh dựa trên tình huống và ràng buộc cụ thể của người dùng.
    - Ví dụ, công thức nấu ăn được tạo ra với các thông số thành phần phù hợp với nhu cầu calo luôn thay đổi của người dùng khi mức độ thể chất và chỉ số cơ thể của họ phát triển.

5. **Yếu tố động lực**:

    - Nội dung động viên và thông báo của ứng dụng được cá nhân hóa dựa trên tính cách và phản ứng của người dùng đối với các chiến lược tạo động lực khác nhau.
    - Một số người dùng có thể nhận được thông điệp khuyến khích, trong khi những người khác nhận được phản hồi dựa trên dữ liệu nhiều hơn.

Trong ví dụ này, GenUI cho phép FitAI tạo ra trải nghiệm được cá nhân hóa cao cho mỗi người dùng, tiềm năng tăng sự tương tác, sự hài lòng và khả năng đạt được mục tiêu thể chất. Các thành phần giao diện, nội dung và thậm chí cả "tính cách" của ứng dụng thích ứng để phục vụ tốt nhất nhu cầu và sở thích của từng người dùng.

Sự chuyển dịch sang Thiết kế Hướng kết quả

GenUI thể hiện một sự thay đổi cơ bản trong cách tiếp cận thiết kế giao diện người dùng!, chuyển từ việc tập trung vào việc tạo ra các thành phần giao diện cụ thể sang một cách tiếp cận toàn diện hơn, hướng đến kết quả. Sự thay đổi này có một số ý nghĩa quan trọng:

1. **Tập trung vào Mục tiêu Người dùng**:
    - Các nhà thiết kế sẽ cần suy nghĩ sâu hơn về mục tiêu và kết quả mong muốn của người dùng thay vì các thành phần giao diện cụ thể.
    - Trọng tâm sẽ là tạo ra các hệ thống có thể sinh ra giao diện giúp người dùng đạt được mục tiêu của họ một cách hiệu quả.
    - Các framework UI mới sẽ xuất hiện để cung cấp cho các nhà thiết kế dựa trên AI các công cụ cần thiết để có thể tạo ra trải nghiệm người dùng *ngay lập tức* và *từ đầu* thay vì dựa trên các đặc tả màn hình được định nghĩa trước.
2. **Vai trò thay đổi của Nhà thiết kế**:
    - Các nhà thiết kế sẽ chuyển từ việc tạo ra bố cục cố định sang việc định nghĩa các quy tắc, ràng buộc và hướng dẫn cho hệ thống AI tuân theo khi tạo ra giao diện.
    - Họ sẽ cần phát triển kỹ năng trong các lĩnh vực như phân tích dữ liệu, kỹ thuật thiết kế prompt, và tư duy hệ thống để hướng dẫn hiệu quả các hệ thống GenUI.
3. **Tầm quan trọng của Nghiên cứu Người dùng**:
    - Nghiên cứu người dùng trở nên quan trọng hơn trong bối cảnh GenUI, vì các nhà thiết kế cần hiểu không chỉ sở thích người dùng, mà còn cả cách những sở thích và nhu cầu này thay đổi trong các ngữ cảnh khác nhau.

- Kiểm thử người dùng liên tục và vòng phản hồi sẽ là thiết yếu để tinh chỉnh và cải thiện khả năng tạo ra giao diện hiệu quả của AI.

4. **Thiết kế cho Tính biến đổi**:

   - Thay vì tạo ra một giao diện "hoàn hảo" duy nhất, các nhà thiết kế sẽ cần xem xét nhiều biến thể có thể và đảm bảo rằng hệ thống có thể tạo ra giao diện phù hợp cho các nhu cầu người dùng đa dạng.
   - Điều này bao gồm thiết kế cho các trường hợp ngoại lệ và đảm bảo rằng các giao diện được tạo ra duy trì khả năng sử dụng và khả năng tiếp cận trên các cấu hình khác nhau.
   - Sự khác biệt sản phẩm có thêm các khía cạnh mới liên quan đến các quan điểm khác nhau về tâm lý người dùng và việc tận dụng các bộ dữ liệu và cơ sở kiến thức độc đáo không có sẵn cho đối thủ cạnh tranh.

Thách thức và Cân nhắc

Mặc dù GenUI mang lại những khả năng thú vị, nó cũng đặt ra một số thách thức và cân nhắc:

1. **Giới hạn Kỹ thuật**:

   - Công nghệ AI hiện tại, dù đã tiên tiến, vẫn còn hạn chế trong việc hiểu ý định phức tạp của người dùng và tạo ra giao diện thực sự nhận biết ngữ cảnh.
   - Các vấn đề về hiệu suất liên quan đến việc tạo ra các thành phần giao diện theo thời gian thực, đặc biệt là trên các thiết bị có hiệu năng thấp.

2. **Yêu cầu về Dữ liệu**:

   - Tùy thuộc vào trường hợp sử dụng, các hệ thống GenUI hiệu quả có thể cần một lượng lớn dữ liệu người dùng để tạo ra giao diện được cá nhân hóa.

- Những thách thức trong việc thu thập dữ liệu người dùng một cách có đạo đức làm dấy lên lo ngại về quyền riêng tư và bảo mật dữ liệu, cũng như các thiên kiến tiềm ẩn trong dữ liệu được dùng để huấn luyện các mô hình GenUI.

3. **Khả năng Sử dụng và Tính Nhất quán**:

   - Ít nhất cho đến khi thực hành này trở nên phổ biến, một ứng dụng với giao diện liên tục thay đổi có thể dẫn đến các vấn đề về khả năng sử dụng, khi người dùng có thể gặp khó khăn trong việc tìm kiếm các thành phần quen thuộc hoặc điều hướng hiệu quả.
   - Việc cân bằng giữa cá nhân hóa và duy trì một giao diện nhất quán, dễ học sẽ là điều quan trọng.

4. **Phụ thuộc quá mức vào AI**:

   - Có nguy cơ ủy thác quá nhiều quyết định thiết kế cho các hệ thống AI, có thể dẫn đến những lựa chọn giao diện thiếu sáng tạo, có vấn đề, hoặc đơn giản là không hoạt động.
   - Sự giám sát của con người và khả năng can thiệp vào các thiết kế do AI tạo ra sẽ vẫn quan trọng trong tương lai gần.

5. **Các vấn đề về Khả năng Tiếp cận**:

   - Đảm bảo các giao diện được tạo động vẫn có thể tiếp cận được với người dùng khuyết tật tạo ra những thách thức hoàn toàn mới, điều này đáng lo ngại khi xét đến mức độ tuân thủ khả năng tiếp cận kém của các hệ thống thông thường.
   - Mặt khác, các nhà thiết kế AI có thể được triển khai với sự quan tâm *tích hợp sẵn* đến khả năng tiếp cận, và khả năng xây dựng giao diện có thể tiếp cận được ngay lập tức giống như cách họ xây dựng giao diện cho người dùng không bị khuyết tật.

- Dù thế nào đi nữa, các hệ thống GenUI nên được thiết kế với hướng dẫn về khả năng tiếp cận mạnh mẽ và quy trình kiểm thử.

6. **Niềm tin và Tính minh bạch của Người dùng**:

   - Người dùng có thể cảm thấy không thoải mái với các giao diện dường như "biết quá nhiều" về họ hoặc thay đổi theo những cách họ không hiểu.
   - Việc đảm bảo tính minh bạch về cách thức và lý do giao diện được cá nhân hóa sẽ quan trọng trong việc xây dựng niềm tin của người dùng.

Triển vọng và Cơ hội Tương lai

Tương lai của Giao diện Người dùng Sinh thành (GenUI) mang trong mình lời hứa to lớn về việc cách mạng hóa cách chúng ta tương tác với các sản phẩm và dịch vụ số. Khi công nghệ này tiếp tục phát triển, chúng ta có thể dự đoán một sự thay đổi mang tính địa chấn trong cách giao diện người dùng được thiết kế, triển khai và trải nghiệm. Tôi nghĩ GenUI là hiện tượng cuối cùng sẽ đưa phần mềm của chúng ta vào lĩnh vực hiện được coi là viễn tưởng khoa học.

Một trong những triển vọng thú vị nhất của GenUI là tiềm năng nâng cao khả năng tiếp cận ở quy mô lớn, vượt xa việc chỉ đảm bảo rằng những người có khuyết tật nghiêm trọng không bị hoàn toàn loại trừ khỏi việc sử dụng phần mềm của bạn. Bằng cách tự động điều chỉnh giao diện theo nhu cầu cá nhân của người dùng, GenUI có thể làm cho trải nghiệm số trở nên toàn diện hơn bao giờ hết. Hãy tưởng tượng các giao diện tự động điều chỉnh để cung cấp văn bản lớn hơn cho người dùng trẻ tuổi hoặc người khiếm thị, hoặc bố cục đơn giản hóa cho những người có khuyết tật về nhận thức, tất cả mà không cần cấu hình thủ công hoặc phiên bản "có thể tiếp cận" riêng biệt của ứng dụng.

Khả năng cá nhân hóa của GenUI có khả năng thúc đẩy sự tăng cường tương tác, sự hài lòng và lòng trung thành của người dùng trong nhiều loại sản phẩm số. Khi giao diện trở nên phù hợp hơn với sở thích và hành vi cá nhân, người dùng sẽ thấy trải nghiệm

số trở nên trực quan và thú vị hơn, có thể dẫn đến những tương tác sâu sắc và ý nghĩa hơn với công nghệ.

GenUI cũng có tiềm năng chuyển đổi quá trình làm quen cho người dùng mới. Bằng cách tạo ra trải nghiệm người dùng lần đầu trực quan, được cá nhân hóa và nhanh chóng thích ứng với mức độ thành thạo của từng người dùng, GenUI có thể giảm đáng kể đường cong học tập liên quan đến các ứng dụng mới. Điều này có thể dẫn đến tốc độ áp dụng nhanh hơn và tăng sự tự tin của người dùng trong việc khám phá các tính năng và chức năng mới.

Một khả năng thú vị khác là khả năng của GenUI trong việc duy trì trải nghiệm người dùng nhất quán trên các thiết bị và nền tảng khác nhau trong khi tối ưu hóa cho từng bối cảnh sử dụng cụ thể. Điều này có thể giải quyết thách thức lâu dài trong việc cung cấp trải nghiệm mạch lạc trên một môi trường thiết bị ngày càng phân mảnh, từ điện thoại thông minh và máy tính bảng đến máy tính để bàn và các công nghệ mới nổi như kính thực tế gia tăng.

Bản chất dựa trên dữ liệu của GenUI mở ra cơ hội cho việc lặp lại và cải thiện nhanh chóng trong thiết kế giao diện người dùng. Bằng cách thu thập dữ liệu thời gian thực về cách người dùng tương tác với các giao diện được tạo ra, các nhà thiết kế và phát triển có thể có được những hiểu biết chưa từng có về hành vi và sở thích của người dùng. Vòng phản hồi này có thể dẫn đến những cải tiến liên tục trong thiết kế giao diện người dùng, được thúc đẩy bởi các mẫu sử dụng thực tế thay vì các giả định hoặc kiểm thử người dùng hạn chế.

Để chuẩn bị cho sự thay đổi này, các nhà thiết kế sẽ cần phát triển kỹ năng và tư duy của họ. Trọng tâm sẽ chuyển từ việc tạo ra các bố cục cố định sang phát triển các hệ thống thiết kế và hướng dẫn toàn diện có thể định hướng cho việc tạo giao diện dựa trên AI. Các nhà thiết kế sẽ cần phát triển hiểu biết sâu sắc về phân tích dữ liệu, công nghệ AI và tư duy hệ thống để hướng dẫn hiệu quả các hệ thống GenUI.

Hơn nữa, khi GenUI làm mờ ranh giới giữa thiết kế và công nghệ, các nhà thiết kế sẽ cần hợp tác chặt chẽ hơn với các nhà phát triển và nhà khoa học dữ liệu. Cách tiếp cận

liên ngành này sẽ là yếu tố then chốt trong việc tạo ra các hệ thống GenUI không chỉ hấp dẫn về mặt hình ảnh và thân thiện với người dùng mà còn mạnh mẽ về mặt kỹ thuật và hợp lý về mặt đạo đức.

Các hàm ý đạo đức của GenUI sẽ ngày càng trở nên quan trọng khi công nghệ này phát triển. Các nhà thiết kế sẽ đóng vai trò then chốt trong việc phát triển khuôn khổ để sử dụng AI một cách có trách nhiệm trong thiết kế giao diện, đảm bảo rằng việc cá nhân hóa nâng cao trải nghiệm người dùng mà không xâm phạm quyền riêng tư hoặc thao túng hành vi người dùng theo những cách phi đạo đức.

Khi nhìn về tương lai, GenUI mang đến cả những cơ hội thú vị lẫn những thách thức đáng kể. Công nghệ này có tiềm năng tạo ra những trải nghiệm kỹ thuật số trực quan, hiệu quả và thỏa mãn hơn cho người dùng trên toàn cầu. Mặc dù nó sẽ đòi hỏi các nhà thiết kế phải thích nghi và tiếp thu những kỹ năng mới, nhưng đồng thời cũng mang đến cơ hội chưa từng có để định hình tương lai của tương tác người-máy theo những cách sâu sắc và ý nghĩa. Hành trình hướng tới các hệ thống GenUI hoàn thiện chắc chắn sẽ phức tạp, nhưng những phần thưởng tiềm năng về việc cải thiện trải nghiệm người dùng và khả năng tiếp cận kỹ thuật số khiến nó trở thành một tương lai đáng để phấn đấu.

Điều phối quy trình làm việc thông minh

Trong lĩnh vực phát triển ứng dụng, quy trình làm việc đóng vai trò quan trọng trong việc xác định cách thức tổ chức và thực hiện các tác vụ, quy trình, và tương tác của người dùng. Khi các ứng dụng ngày càng phức tạp và kỳ vọng của người dùng tiếp tục tăng cao, nhu cầu về điều phối quy trình làm việc thông minh và thích ứng trở nên ngày càng rõ ràng.

Phương pháp "Điều phối quy trình làm việc thông minh" tập trung vào việc tận dụng các thành phần AI để điều phối và tối ưu hóa một cách linh động các quy trình làm việc phức tạp trong ứng dụng. Mục tiêu là tạo ra các ứng dụng hiệu quả hơn, phản hồi nhanh hơn và có khả năng thích ứng với dữ liệu và ngữ cảnh thời gian thực.

Trong chương này, chúng ta sẽ khám phá các nguyên tắc và mẫu thiết kế chính làm nền tảng cho phương pháp điều phối quy trình làm việc thông minh. Chúng ta sẽ xem xét

cách AI có thể được sử dụng để định tuyến tác vụ một cách thông minh, tự động hóa việc ra quyết định, và động thích ứng quy trình làm việc dựa trên các yếu tố khác nhau như hành vi người dùng, hiệu suất hệ thống và quy tắc kinh doanh. Thông qua các ví dụ thực tế và tình huống trong thực tế, chúng ta sẽ chứng minh tiềm năng chuyển đổi của AI trong việc tinh giản và tối ưu hóa quy trình làm việc của ứng dụng.

Cho dù bạn đang xây dựng các ứng dụng doanh nghiệp với quy trình kinh doanh phức tạp hay các ứng dụng hướng đến người dùng với hành trình người dùng năng động, các mẫu thiết kế và kỹ thuật được thảo luận trong chương này sẽ trang bị cho bạn kiến thức và công cụ để tạo ra các quy trình làm việc thông minh và hiệu quả, nâng cao trải nghiệm tổng thể của người dùng và thúc đẩy giá trị kinh doanh.

Nhu cầu kinh doanh

Các phương pháp truyền thống trong quản lý quy trình làm việc thường dựa vào các quy tắc được định nghĩa trước và cây quyết định tĩnh, có thể trở nên cứng nhắc, thiếu linh hoạt và không thể đối phó với bản chất động của các ứng dụng hiện đại.

Hãy xem xét một tình huống trong đó một ứng dụng thương mại điện tử cần xử lý một quy trình thực hiện đơn hàng phức tạp. Quy trình có thể bao gồm nhiều bước như xác thực đơn hàng, kiểm tra tồn kho, xử lý thanh toán, vận chuyển và thông báo cho khách hàng. Mỗi bước có thể có các quy tắc, phụ thuộc, tích hợp bên ngoài và cơ chế xử lý ngoại lệ riêng. Việc quản lý quy trình như vậy một cách thủ công hoặc thông qua logic cứng có thể nhanh chóng trở nên phức tạp, dễ xảy ra lỗi và khó bảo trì.

Hơn nữa, khi ứng dụng mở rộng và số lượng người dùng đồng thời tăng lên, quy trình có thể cần thích ứng và tự tối ưu hóa dựa trên dữ liệu thời gian thực và hiệu suất hệ thống. Ví dụ, trong thời điểm cao điểm, ứng dụng có thể cần điều chỉnh quy trình một cách động để ưu tiên một số tác vụ nhất định, phân bổ tài nguyên hiệu quả và đảm bảo trải nghiệm người dùng mượt mà.

Đây là lúc phương pháp “Điều phối quy trình làm việc thông minh” phát huy tác dụng.

Bằng cách tận dụng các thành phần AI, các nhà phát triển có thể tạo ra các quy trình làm việc thông minh, thích ứng và tự tối ưu hóa. AI có thể phân tích lượng lớn dữ liệu, học hỏi từ kinh nghiệm trong quá khứ và đưa ra quyết định thông minh trong thời gian thực để điều phối quy trình một cách hiệu quả.

Lợi ích chính

1. **Tăng hiệu quả**: AI có thể tối ưu hóa việc phân bổ tác vụ, sử dụng tài nguyên và thực thi quy trình, dẫn đến thời gian xử lý nhanh hơn và cải thiện hiệu quả tổng thể.
2. **Khả năng thích ứng**: Các quy trình làm việc dựa trên AI có thể thích ứng động với các điều kiện thay đổi, như biến động trong nhu cầu người dùng, hiệu suất hệ thống hoặc yêu cầu kinh doanh, đảm bảo ứng dụng luôn phản hồi nhanh và có khả năng phục hồi.
3. **Ra quyết định tự động**: AI có thể tự động hóa các quy trình ra quyết định phức tạp trong quy trình làm việc, giảm sự can thiệp thủ công và giảm thiểu rủi ro lỗi của con người.
4. **Cá nhân hóa**: AI có thể phân tích hành vi, sở thích và ngữ cảnh của người dùng để cá nhân hóa quy trình và mang lại trải nghiệm phù hợp cho từng người dùng.
5. **Khả năng mở rộng**: Các quy trình làm việc được hỗ trợ bởi AI có thể mở rộng liền mạch để xử lý khối lượng dữ liệu và tương tác người dùng ngày càng tăng mà không ảnh hưởng đến hiệu suất hoặc độ tin cậy.

Trong các phần tiếp theo, chúng ta sẽ khám phá các mẫu thiết kế và kỹ thuật chính cho phép triển khai quy trình làm việc thông minh và trình bày các ví dụ thực tế về cách AI đang chuyển đổi quản lý quy trình làm việc trong các ứng dụng hiện đại.

Các mẫu thiết kế chính

Để triển khai điều phối quy trình làm việc thông minh trong các ứng dụng, các nhà phát triển có thể tận dụng một số mẫu thiết kế chính khai thác sức mạnh của AI. Những mẫu thiết kế này cung cấp một phương pháp có cấu trúc để thiết kế và quản lý quy trình làm việc, cho phép các ứng dụng thích ứng, tối ưu hóa và tự động hóa quy trình dựa trên dữ liệu và ngữ cảnh thời gian thực. Hãy khám phá một số mẫu thiết kế cơ bản trong điều phối quy trình làm việc thông minh.

Định tuyến tác vụ động

Mẫu thiết kế này liên quan đến việc sử dụng AI để định tuyến thông minh các tác vụ trong một quy trình làm việc dựa trên nhiều yếu tố như độ ưu tiên của tác vụ, khả năng sẵn có của tài nguyên và hiệu suất hệ thống. Các thuật toán AI có thể phân tích đặc điểm của từng tác vụ, xem xét trạng thái hiện tại của hệ thống và đưa ra quyết định thông minh để giao tác vụ cho các tài nguyên hoặc đường dẫn xử lý phù hợp nhất. Định tuyến tác vụ động đảm bảo các tác vụ được phân phối và thực thi hiệu quả, tối ưu hóa hiệu suất tổng thể của quy trình làm việc.

```
class TaskRouter
  include Raix::ChatCompletion
  include Raix::FunctionDispatch

  attr_accessor :task

  # list of functions that can be called by the AI entirely at its
  # discretion depending on the task received

  function :analyze_task_priority do
    TaskPriorityAnalyzer.perform(task)
  end

  function :check_resource_availability, # ...
  function :assess_system_performance, # ...
```

```
16    function :assign_task_to_resource, # ...
17
18    DIRECTIVE = "You are a task router, responsible for intelligently
19     assigning tasks to available resources based on priority, resource
20     availability, and system performance..."
21
22    def initialize(task)
23      self.task = task
24      transcript << { system: DIRECTIVE }
25      transcript << { user: task.to_json }
26    end
27
28    def perform
29      while task.unassigned?
30        chat_completion
31
32        # todo: add max loop counter and break
33      end
34
35      # capture the transcript for later analysis
36      task.update(routing_transcript: transcript)
37    end
38  end
```

Lưu ý vòng lặp được tạo bởi biểu thức `while` ở dòng 29, tiếp tục nhắc AI cho đến khi nhiệm vụ được giao. Ở dòng 35, chúng ta lưu bản ghi của nhiệm vụ để phân tích và gỡ lỗi sau này nếu cần thiết.

Ra Quyết Định Theo Ngữ Cảnh

Bạn có thể sử dụng mã tương tự để đưa ra các quyết định dựa trên ngữ cảnh trong quy trình làm việc. Bằng cách phân tích các điểm dữ liệu liên quan như sở thích người dùng, mô hình lịch sử, và dữ liệu đầu vào thời gian thực, các thành phần AI có thể xác định hướng hành động phù hợp nhất tại mỗi điểm quyết định trong quy trình. Điều chỉnh hành vi của quy trình dựa trên ngữ cảnh cụ thể của từng người dùng hoặc tình huống, cung cấp trải nghiệm được cá nhân hóa và tối ưu hóa.

Xây Dựng Quy Trình Thích Ứng

Mẫu này tập trung vào việc xây dựng và điều chỉnh quy trình một cách linh động dựa trên các yêu cầu hoặc điều kiện thay đổi. AI có thể phân tích trạng thái hiện tại của quy trình, xác định các điểm nghẽn hoặc sự không hiệu quả, và tự động điều chỉnh cấu trúc quy trình để tối ưu hóa hiệu suất. Việc xây dựng quy trình thích ứng cho phép ứng dụng liên tục phát triển và cải thiện quy trình mà không cần can thiệp thủ công.

Xử Lý và Khôi Phục Ngoại Lệ

Xử lý và khôi phục ngoại lệ là những khía cạnh quan trọng trong việc điều phối quy trình thông minh. Khi làm việc với các thành phần AI và quy trình phức tạp, việc dự đoán và xử lý ngoại lệ một cách khéo léo là điều cần thiết để đảm bảo tính ổn định và độ tin cậy của hệ thống.

Dưới đây là một số điểm cần lưu ý và kỹ thuật xử lý và khôi phục ngoại lệ trong quy trình thông minh:

1. **Lan Truyền Ngoại Lệ**: Triển khai một cách tiếp cận nhất quán để lan truyền ngoại lệ qua các thành phần của quy trình. Khi một ngoại lệ xảy ra trong một thành phần, nó cần được bắt, ghi lại và lan truyền đến bộ điều phối hoặc một thành phần riêng biệt chịu trách nhiệm xử lý ngoại lệ. Ý tưởng là tập trung hóa việc xử lý ngoại lệ và ngăn chặn việc ngoại lệ bị che giấu một cách âm thầm, đồng thời mở ra khả năng cho Xử Lý Lỗi Thông Minh.
2. **Cơ Chế Thử Lại**: Cơ chế thử lại giúp cải thiện khả năng phục hồi của quy trình và xử lý các lỗi tạm thời một cách khéo léo. Chắc chắn nên triển khai cơ chế thử lại cho các ngoại lệ tạm thời hoặc có thể khôi phục, chẳng hạn như kết nối mạng hoặc tài nguyên không khả dụng có thể được tự động thử lại sau một khoảng thời gian chỉ định. Việc có một bộ điều phối hoặc trình xử lý ngoại lệ được hỗ trợ bởi AI có nghĩa là chiến lược thử lại của bạn không cần phải mang tính máy móc, dựa

vào các thuật toán cố định như lùi theo cấp số nhân. Bạn có thể để việc xử lý thử lại theo "quyết định" của thành phần AI chịu trách nhiệm quyết định cách xử lý ngoại lệ.

3. **Chiến Lược Dự Phòng**: Nếu một thành phần AI không thể cung cấp phản hồi hợp lệ hoặc gặp lỗi—một tình huống phổ biến do tính chất công nghệ mới—cần có một cơ chế dự phòng để đảm bảo quy trình có thể tiếp tục. Điều này có thể bao gồm việc sử dụng giá trị mặc định, thuật toán thay thế, hoặc Con Người Trong Vòng Lặp để đưa ra quyết định và giúp quy trình tiến triển.
4. **Hành Động Bù Trừ**: Chỉ thị của bộ điều phối nên bao gồm hướng dẫn về các hành động bù trừ để xử lý các ngoại lệ không thể tự động giải quyết. Hành động bù trừ là các bước được thực hiện để hoàn tác hoặc giảm thiểu tác động của một thao tác thất bại. Ví dụ, nếu bước xử lý thanh toán thất bại, một hành động bù trừ có thể là hoàn tác giao dịch và thông báo cho người dùng. Hành động bù trừ giúp duy trì tính nhất quán và toàn vẹn dữ liệu khi đối mặt với ngoại lệ.
5. **Giám Sát và Cảnh Báo Ngoại Lệ**: Thiết lập cơ chế giám sát và cảnh báo để phát hiện và thông báo cho các bên liên quan về các ngoại lệ quan trọng. Bộ điều phối có thể được cấu hình để nhận biết các ngưỡng và quy tắc để kích hoạt cảnh báo khi ngoại lệ vượt quá giới hạn nhất định hoặc khi các loại ngoại lệ cụ thể xảy ra. Điều này cho phép xác định và giải quyết vấn đề một cách chủ động trước khi chúng ảnh hưởng đến toàn bộ hệ thống.

Dưới đây là một ví dụ về xử lý và khôi phục ngoại lệ trong một thành phần quy trình Ruby:

```ruby
class InventoryManager
  def check_availability(order)
    begin
      # Perform inventory check logic
      inventory = Inventory.find_by(product_id: order.product_id)
      if inventory.available_quantity >= order.quantity
        return true
      else
        raise InsufficientInventoryError,
              "Insufficient inventory for product #{order.product_id}"
      end
    rescue InsufficientInventoryError => e
      # Log the exception
      logger.error("Inventory check failed: #{e.message}")

      # Retry the operation after a delay
      retry_count ||= 0
      if retry_count < MAX_RETRIES
        retry_count += 1
        sleep(RETRY_DELAY)
        retry
      else
        # Fallback to manual intervention
        NotificationService.admin("Inventory check failed: Order #{order.id}")
        return false
      end
    end
  end
end
```

Trong ví dụ này, thành phần `InventoryManager` kiểm tra tính khả dụng của sản phẩm cho một đơn hàng cụ thể. Nếu số lượng có sẵn không đủ, nó sẽ phát sinh một `InsufficientInventoryError`. Ngoại lệ này được bắt, ghi lại, và một cơ chế thử lại được thực hiện. Nếu vượt quá giới hạn thử lại, thành phần sẽ chuyển sang can thiệp thủ công bằng cách thông báo cho quản trị viên.

Bằng cách triển khai các cơ chế xử lý và khôi phục ngoại lệ mạnh mẽ, bạn có thể đảm bảo rằng các quy trình thông minh của mình có khả năng phục hồi, dễ bảo trì và có thể

xử lý các tình huống không mong muốn một cách linh hoạt.

Những mẫu này tạo nên nền tảng cho việc điều phối quy trình thông minh và có thể được kết hợp và điều chỉnh để phù hợp với yêu cầu cụ thể của các ứng dụng khác nhau. Bằng cách tận dụng các mẫu này, các nhà phát triển có thể tạo ra các quy trình linh hoạt, có khả năng phục hồi và được tối ưu hóa cho hiệu suất và trải nghiệm người dùng.

Trong phần tiếp theo, chúng ta sẽ khám phá cách triển khai các mẫu này trong thực tế, sử dụng các ví dụ thực tế và đoạn mã để minh họa việc tích hợp các thành phần AI vào quản lý quy trình.

Triển khai Điều phối Quy trình Thông minh trong Thực tế

Giờ đây khi chúng ta đã khám phá các mẫu chính trong điều phối quy trình thông minh, hãy đi sâu vào cách triển khai các mẫu này trong các ứng dụng thực tế. Chúng ta sẽ cung cấp các ví dụ thực tế và đoạn mã để minh họa việc tích hợp các thành phần AI vào quản lý quy trình.

Bộ xử lý Đơn hàng Thông minh

Hãy đi sâu vào một ví dụ thực tế về việc triển khai điều phối quy trình thông minh bằng cách sử dụng thành phần `OrderProcessor` được hỗ trợ bởi AI trong ứng dụng thương mại điện tử Ruby on Rails. `OrderProcessor` hiện thực hóa khái niệm Trình quản lý quy trình Tích hợp doanh nghiệp mà chúng ta đã gặp trong Chương 3 khi thảo luận về Đa dạng Người thực thi. Thành phần này sẽ chịu trách nhiệm quản lý quy trình thực hiện đơn hàng, đưa ra quyết định định tuyến dựa trên kết quả trung gian và điều phối việc thực hiện các bước xử lý khác nhau.

Quy trình thực hiện đơn hàng bao gồm nhiều bước như xác thực đơn hàng, kiểm tra kho hàng, xử lý thanh toán và vận chuyển. Mỗi bước được triển khai như một tiến trình xử lý riêng biệt thực hiện một nhiệm vụ cụ thể và trả về kết quả cho `OrderProcessor`. Các bước không nhất thiết phải bắt buộc và thậm chí không cần phải được thực hiện theo một thứ tự chính xác.

Dưới đây là một ví dụ triển khai của `OrderProcessor`. Nó có hai mixin từ Raix. Mixin đầu tiên (`ChatCompletion`) cho phép nó thực hiện hoàn thành hội thoại, điều này làm cho nó trở thành một thành phần AI. Mixin thứ hai (`FunctionDispatch`) cho phép gọi hàm bởi AI, cho phép nó phản hồi một lời nhắc bằng cách gọi hàm thay vì gửi tin nhắn văn bản.

Các hàm xử lý (`validate_order`, `check_inventory`, và các hàm khác) ủy quyền cho các lớp xử lý tương ứng của chúng, có thể là các thành phần AI hoặc không phải AI, với yêu cầu duy nhất là chúng trả về kết quả công việc của mình trong một định dạng có thể được biểu diễn dưới dạng chuỗi.

Như với tất cả các ví dụ khác trong phần này của cuốn sách, mã này về cơ bản là mã giả và chỉ nhằm truyền đạt ý nghĩa của mẫu và truyền cảm hứng cho sáng tạo của riêng bạn. Mô tả đầy đủ về các mẫu và ví dụ mã hoàn chỉnh được bao gồm trong Phần 2.

```ruby
class OrderProcessor
  include Raix::ChatCompletion
  include Raix::FunctionDispatch

  SYSTEM_DIRECTIVE = "You are an order processor, tasked with..."

  def initialize(order)
    self.order = order
    transcript << { system: SYSTEM_DIRECTIVE }
    transcript << { user: order.to_json }
  end

  def perform
    # will continue looping until `stop_looping!` is called
    chat_completion(loop: true)
  end

  # list of functions available to be called by the AI
  # truncated for brevity

  def functions
    [
      {
        name: "validate_order",
        description: "Invoke to check validity of order",
        parameters: {
          ...
      },
      ...
    ]
  end

  # implementation of functions that can be called by the AI
  # entirely at its discretion, depending on the needs of the order

  def validate_order
    OrderValidationWorker.perform(@order)
  end

  def check_inventory
    InventoryCheckWorker.perform(@order)
  end
```

```
43
44    def process_payment
45      PaymentProcessingWorker.perform(@order)
46    end
47
48    def schedule_shipping
49      ShippingSchedulerWorker.perform(@order)
50    end
51
52    def send_confirmation
53      OrderConfirmationWorker.perform(@order)
54    end
55
56    def finished_processing
57      @order.update!(transcript:, processed_at: Time.current)
58      stop_looping!
59    end
60  end
```

Trong ví dụ này, OrderProcessor được khởi tạo với một đối tượng đơn hàng và duy trì một bản ghi của quá trình thực thi quy trình làm việc, theo định dạng bản ghi hội thoại điển hình vốn có của các mô hình ngôn ngữ lớn. AI được trao toàn quyền điều phối việc thực hiện các bước xử lý khác nhau, như xác thực đơn hàng, kiểm tra tồn kho, xử lý thanh toán và vận chuyển.

Mỗi khi phương thức `chat_completion` được gọi, bản ghi sẽ được gửi đến AI để nó cung cấp kết quả dưới dạng một lệnh gọi hàm. AI hoàn toàn có quyền phân tích kết quả của bước trước đó và xác định hành động thích hợp cần thực hiện. Ví dụ, nếu việc kiểm tra tồn kho cho thấy mức tồn kho thấp, `OrderProcessor` có thể lên lịch một nhiệm vụ bổ sung hàng. Nếu việc xử lý thanh toán thất bại, nó có thể khởi tạo việc thử lại hoặc thông báo cho bộ phận hỗ trợ khách hàng.

> Ví dụ trên không có các hàm được định nghĩa cho việc bổ sung hàng hoặc thông báo cho bộ phận hỗ trợ khách hàng, nhưng hoàn toàn có thể thêm vào.

Bản ghi sẽ tăng trưởng mỗi khi một hàm được gọi và đóng vai trò như một bản ghi của quá trình thực thi quy trình làm việc, bao gồm kết quả của từng bước và các chỉ dẫn do AI tạo ra cho các bước tiếp theo. Bản ghi này có thể được sử dụng để gỡ lỗi, kiểm toán và cung cấp khả năng theo dõi vào quy trình xử lý đơn hàng.

Bằng cách tận dụng AI trong `OrderProcessor`, ứng dụng thương mại điện tử có thể động điều chỉnh quy trình làm việc dựa trên dữ liệu thời gian thực và xử lý các ngoại lệ một cách thông minh. Thành phần AI có thể đưa ra quyết định sáng suốt, tối ưu hóa quy trình làm việc và đảm bảo việc xử lý đơn hàng diễn ra trơn tru ngay cả trong các tình huống phức tạp.

Việc yêu cầu duy nhất đối với các tiến trình xử lý chỉ là trả về một kết quả mà AI có thể hiểu được để quyết định việc cần làm tiếp theo, có thể khiến bạn bắt đầu nhận ra cách tiếp cận này có thể giảm thiểu công việc ánh xạ đầu vào/đầu ra thường liên quan đến việc tích hợp các hệ thống khác nhau với nhau.

Bộ Kiểm Duyệt Nội Dung Thông Minh

Các ứng dụng mạng xã hội thường yêu cầu ít nhất một mức độ kiểm duyệt nội dung tối thiểu để đảm bảo một cộng đồng an toàn và lành mạnh. Ví dụ về thành phần `ContentModerator` này tận dụng AI để điều phối quy trình kiểm duyệt một cách thông minh, đưa ra quyết định dựa trên đặc điểm của nội dung và kết quả của các bước kiểm duyệt khác nhau.

Quy trình kiểm duyệt bao gồm nhiều bước như phân tích văn bản, nhận dạng hình ảnh, đánh giá uy tín người dùng và xem xét thủ công. Mỗi bước được thực hiện như

một tiến trình xử lý riêng biệt thực hiện một nhiệm vụ cụ thể và trả về kết quả cho `ContentModerator`.

Dưới đây là một ví dụ về việc triển khai `ContentModerator`:

```
class ContentModerator
  include Raix::ChatCompletion
  include Raix::FunctionDispatch

  SYSTEM_DIRECTIVE = "You are a content moderator process manager,
    tasked with the workflow involved in moderating user-generated content..."

  def initialize(content)
    @content = content
    @transcript = [
      { system: SYSTEM_DIRECTIVE },
      { user: content.to_json }
    ]
  end

  def perform
    complete(@transcript)
  end

  def model
    "openai/gpt-4"
  end

  # list of functions available to be called by the AI
  # truncated for brevity

  def functions
    [
      {
        name: "analyze_text",
        # ...
      },
      {
        name: "recognize_image",
        description: "Invoke to describe images...",
        # ...
      },
```

```
      {
        name: "assess_user_reputation",
        # ...
      },
      {
        name: "escalate_to_manual_review",
        # ...
      },
      {
        name: "approve_content",
        # ...
      },
      {
        name: "reject_content",
        # ...
      }
    ]
  end

  # implementation of functions that can be called by the AI
  # entirely at its discretion, depending on the needs of the order

  def analyze_text
    result = TextAnalysisWorker.perform(@content)
    continue_with(result)
  end

  def recognize_image
    result = ImageRecognitionWorker.perform(@content)
    continue_with(result)
  end

  def assess_user_reputation
    result = UserReputationWorker.perform(@content.user)
    continue_with(result)
  end

  def escalate_to_manual_review
    ManualReviewWorker.perform(@content)
    @content.update!(status: 'pending', transcript: @transcript)
  end

```

```
  def approve_content
    @content.update!(status: 'approved', transcript: @transcript)
  end

  def reject_content
    @content.update!(status: 'rejected', transcript: @transcript)
  end

  private

  def continue_with(result)
    @transcript << { function: result }
    complete(@transcript)
  end
end
```

Trong ví dụ này, `ContentModerator` được khởi tạo với một đối tượng nội dung và duy trì một bản ghi kiểm duyệt ở dạng hội thoại. Thành phần AI có toàn quyền kiểm soát quy trình kiểm duyệt, quyết định những bước cần thực hiện dựa trên đặc điểm của nội dung và kết quả của từng bước.

Các hàm worker có sẵn để AI gọi bao gồm `analyze_text`, `recognize_image`, `assess_user_reputation`, và `escalate_to_manual_review`. Mỗi hàm ủy thác nhiệm vụ cho một tiến trình worker tương ứng (`TextAnalysisWorker`, `ImageRecognitionWorker`, v.v.) và thêm kết quả vào bản ghi kiểm duyệt, ngoại trừ hàm chuyển tiếp xử lý thủ công, vốn đóng vai trò như một trạng thái kết thúc. Cuối cùng, các hàm `approve_content` và `reject_content` cũng đóng vai trò như các trạng thái kết thúc.

Thành phần AI phân tích nội dung và xác định hành động phù hợp cần thực hiện. Nếu nội dung chứa tham chiếu hình ảnh, nó có thể gọi worker `recognize_image` để hỗ trợ xem xét hình ảnh. Nếu bất kỳ worker nào cảnh báo về nội dung có khả năng gây hại, AI có thể quyết định chuyển tiếp nội dung để xem xét thủ công hoặc từ chối trực tiếp. Nhưng tùy thuộc vào mức độ nghiêm trọng của cảnh báo, AI có thể chọn sử dụng kết quả đánh giá uy tín của người dùng để quyết định cách xử lý nội dung mà nó không

chắc chắn. Tùy thuộc vào trường hợp sử dụng, có lẽ những người dùng đáng tin cậy sẽ có nhiều quyền tự do hơn trong việc đăng bài. Và còn nhiều trường hợp khác nữa…

Giống như ví dụ quản lý quy trình trước đó, bản ghi kiểm duyệt đóng vai trò như một bản ghi của việc thực thi quy trình làm việc, bao gồm kết quả của từng bước và các quyết định do AI tạo ra. Bản ghi này có thể được sử dụng để kiểm toán, minh bạch hóa và cải thiện quy trình kiểm duyệt theo thời gian.

Bằng cách tận dụng AI trong `ContentModerator`, ứng dụng mạng xã hội có thể linh hoạt điều chỉnh quy trình kiểm duyệt dựa trên đặc điểm của nội dung và xử lý thông minh các tình huống kiểm duyệt phức tạp. Thành phần AI có thể đưa ra quyết định sáng suốt, tối ưu hóa quy trình làm việc và đảm bảo trải nghiệm cộng đồng an toàn và lành mạnh.

Hãy khám phá thêm hai ví dụ minh họa về lập lịch tác vụ dự đoán và xử lý ngoại lệ cùng với khôi phục trong bối cảnh điều phối quy trình làm việc thông minh.

Lập Lịch Tác Vụ Dự Đoán trong Hệ Thống Hỗ Trợ Khách Hàng

Trong một ứng dụng hỗ trợ khách hàng được xây dựng bằng , việc quản lý và ưu tiên các phiếu hỗ trợ một cách hiệu quả là rất quan trọng để cung cấp hỗ trợ kịp thời cho khách hàng. Thành phần `SupportTicketScheduler` tận dụng AI để dự đoán lập lịch và phân công các phiếu hỗ trợ cho các tổng đài viên có sẵn dựa trên nhiều yếu tố như mức độ khẩn cấp của phiếu hỗ trợ, chuyên môn của tổng đài viên và khối lượng công việc.

```ruby
class SupportTicketScheduler
  include Raix::ChatCompletion
  include Raix::FunctionDispatch

  SYSTEM_DIRECTIVE = "You are a support ticket scheduler,
    tasked with intelligently assigning tickets to available agents..."

  def initialize(ticket)
    @ticket = ticket
    @transcript = [
      { system: SYSTEM_DIRECTIVE },
      { user: ticket.to_json }
    ]
  end

  def perform
    complete(@transcript)
  end

  def model
    "openai/gpt-4"
  end

  def functions
    [
      {
        name: "analyze_ticket_urgency",
        # ...
      },
      {
        name: "list_available_agents",
        description: "Includes expertise of available agents",
        # ...
      },
      {
        name: "predict_agent_workload",
        description: "Uses historical data to predict upcoming workloads",
        # ...
      },
      {
        name: "assign_ticket_to_agent",
        # ...
```

```
      },
      {
        name: "reschedule_ticket",
        # ...
      }
    ]
  end

  # implementation of functions that can be called by the AI
  # entirely at its discretion, depending on the needs of the order

  def analyze_ticket_urgency
    result = TicketUrgencyAnalyzer.perform(@ticket)
    continue_with(result)
  end

  def list_available_agents
    result = ListAvailableAgents.perform
    continue_with(result)
  end

  def predict_agent_workload
    result = AgentWorkloadPredictor.perform
    continue_with(result)
  end

  def assign_ticket_to_agent
    TicketAssigner.perform(@ticket, @transcript)
  end

  def delay_assignment(until)
    until = DateTimeStandardizer.process(until)
    SupportTicketScheduler.delay(@ticket, @transcript, until)
  end

  private

  def continue_with(result)
    @transcript << { function: result }
    complete(@transcript)
  end
end
```

Trong ví dụ này, `SupportTicketScheduler` được khởi tạo với một đối tượng phiếu hỗ trợ và duy trì một bản ghi lập lịch. Thành phần AI phân tích chi tiết phiếu hỗ trợ và lập lịch phân công phiếu một cách dự đoán dựa trên các yếu tố như mức độ khẩn cấp của phiếu, chuyên môn của tổng đài viên, và khối lượng công việc dự kiến của tổng đài viên.

Các hàm có sẵn để AI gọi bao gồm `analyze_ticket_urgency`, `list_available_agents`, `predict_agent_workload`, và `assign_ticket_to_agent`. Mỗi hàm ủy thác nhiệm vụ cho một thành phần phân tích hoặc dự đoán tương ứng và thêm kết quả vào bản ghi lập lịch. AI cũng có tùy chọn trì hoãn việc phân công bằng cách sử dụng hàm `delay_assignment`.

Thành phần AI kiểm tra bản ghi lập lịch và đưa ra quyết định có căn cứ về việc phân công phiếu hỗ trợ. Nó xem xét mức độ khẩn cấp của phiếu, chuyên môn của các tổng đài viên đang có mặt, và khối lượng công việc dự kiến của mỗi tổng đài viên để xác định tổng đài viên phù hợp nhất để xử lý phiếu.

Bằng cách tận dụng việc lập lịch tác vụ dự đoán, ứng dụng hỗ trợ khách hàng có thể tối ưu hóa việc phân công phiếu, giảm thời gian phản hồi và cải thiện sự hài lòng tổng thể của khách hàng. Việc quản lý chủ động và hiệu quả các phiếu hỗ trợ đảm bảo rằng những phiếu phù hợp được phân công cho đúng tổng đài viên vào đúng thời điểm.

Xử lý Ngoại lệ và Khôi phục trong Quy trình Xử lý Dữ liệu

Việc xử lý ngoại lệ và khôi phục sau sự cố là điều cần thiết để đảm bảo tính toàn vẹn dữ liệu và ngăn chặn mất mát dữ liệu. Thành phần `DataProcessingOrchestrator` sử dụng AI để xử lý ngoại lệ một cách thông minh và điều phối quá trình khôi phục trong quy trình xử lý dữ liệu

```ruby
class DataProcessingOrchestrator
  include Raix::ChatCompletion
  include Raix::FunctionDispatch

  SYSTEM_DIRECTIVE = "You are a data processing orchestrator..."

  def initialize(data_batch)
    @data_batch = data_batch
    @transcript = [
      { system: SYSTEM_DIRECTIVE },
      { user: data_batch.to_json }
    ]
  end

  def perform
    complete(@transcript)
  end

  def model
    "openai/gpt-4"
  end

  def functions
    [
      {
        name: "validate_data",
        # ...
      },
      {
        name: "process_data",
        # ...
      },
      {
        name: "request_fix",
        # ...
      },
      {
        name: "retry_processing",
        # ...
      },
      {
        name: "mark_data_as_failed",
```

```
        # ...
      },
      {
        name: "finished",
        # ...
      }
    ]
  end

  # implementation of functions that can be called by the AI
  # entirely at its discretion, depending on the needs of the order

  def validate_data
    result = DataValidator.perform(@data_batch)
    continue_with(result)
  rescue ValidationException => e
    handle_validation_exception(e)
  end

  def process_data
    result = DataProcessor.perform(@data_batch)
    continue_with(result)
  rescue ProcessingException => e
    handle_processing_exception(e)
  end

  def request_fix(description_of_fix)
    result = SmartDataFixer.new(description_of_fix, @data_batch)
    continue_with(result)
  end

  def retry_processing(timeout_in_seconds)
    wait(timeout_in_seconds)
    process_data
  end

  def mark_data_as_failed
    @data_batch.update!(status: 'failed', transcript: @transcript)
  end

  def finished
    @data_batch.update!(status: 'finished', transcript: @transcript)
```

```
  end

  private

  def continue_with(result)
    @transcript << { function: result }
    complete(@transcript)
  end

  def handle_validation_exception(exception)
    @transcript << { exception: exception.message }
    complete(@transcript)
  end

  def handle_processing_exception(exception)
    @transcript << { exception: exception.message }
    complete(@transcript)
  end
end
```

Trong ví dụ này, `DataProcessingOrchestrator` được khởi tạo với một đối tượng batch dữ liệu và duy trì một bản ghi xử lý. Thành phần AI điều phối đường ống xử lý dữ liệu, xử lý các ngoại lệ và phục hồi sau lỗi khi cần thiết.

Các hàm có sẵn để AI gọi bao gồm `validate_data`, `process_data`, `request_fix`, `retry_processing`, và `mark_data_as_failed`. Mỗi hàm ủy thác nhiệm vụ cho một thành phần xử lý dữ liệu tương ứng và thêm kết quả hoặc chi tiết ngoại lệ vào bản ghi xử lý.

Nếu một ngoại lệ xác thực xảy ra trong bước `validate_data`, hàm `handle_validation_exception` sẽ thêm dữ liệu ngoại lệ vào bản ghi và chuyển quyền điều khiển lại cho AI. Tương tự, nếu một ngoại lệ xử lý xảy ra trong bước `process_data`, AI có thể quyết định chiến lược phục hồi.

Tùy thuộc vào bản chất của ngoại lệ gặp phải, AI có thể tùy ý quyết định gọi `request_fix`, hàm này sẽ ủy thác cho thành phần `SmartDataFixer` được hỗ trợ bởi AI (xem chương Dữ liệu Tự Phục hồi). Bộ sửa dữ liệu nhận một mô tả bằng tiếng Anh đơn giản

về cách nó nên sửa đổi `@data_batch` để có thể thử xử lý lại. Có lẽ một lần thử lại thành công sẽ bao gồm việc xóa các bản ghi khỏi batch dữ liệu đã không vượt qua xác thực và/hoặc sao chép chúng sang một đường ống xử lý khác để xem xét thủ công? Các khả năng là gần như vô tận.

Bằng cách tích hợp xử lý ngoại lệ và phục hồi do AI điều khiển, ứng dụng xử lý dữ liệu trở nên linh hoạt và chịu lỗi tốt hơn. `DataProcessingOrchestrator` quản lý ngoại lệ một cách thông minh, giảm thiểu mất mát dữ liệu và đảm bảo thực thi quy trình xử lý dữ liệu một cách trơn tru.

Giám sát và Ghi nhật ký

Giám sát và ghi nhật ký cung cấp khả năng quan sát tiến trình, hiệu suất và tình trạng của các thành phần quy trình làm việc được hỗ trợ bởi AI, cho phép các nhà phát triển theo dõi và phân tích hành vi của hệ thống. Việc triển khai cơ chế giám sát và ghi nhật ký hiệu quả là thiết yếu cho việc gỡ lỗi, kiểm toán và cải tiến liên tục các quy trình làm việc thông minh.

Giám sát Tiến trình và Hiệu suất Quy trình làm việc

Để đảm bảo việc thực thi trơn tru của các quy trình làm việc thông minh, việc giám sát tiến trình và hiệu suất của mỗi thành phần quy trình là quan trọng. Điều này bao gồm việc theo dõi các chỉ số và sự kiện quan trọng trong suốt vòng đời của quy trình.

Một số khía cạnh quan trọng cần giám sát bao gồm:

1. **Thời gian Thực thi Quy trình**: Đo thời gian mà mỗi thành phần quy trình cần để hoàn thành nhiệm vụ của mình. Điều này giúp xác định các điểm nghẽn về hiệu suất và tối ưu hóa hiệu quả tổng thể của quy trình.

2. **Sử dụng Tài nguyên**: Giám sát việc sử dụng tài nguyên hệ thống, như CPU, bộ nhớ và lưu trữ, của mỗi thành phần quy trình. Điều này giúp đảm bảo hệ thống đang hoạt động trong giới hạn khả năng và có thể xử lý khối lượng công việc một cách hiệu quả.

3. **Tỷ lệ Lỗi và Ngoại lệ**: Theo dõi sự xuất hiện của lỗi và ngoại lệ trong các thành phần quy trình. Điều này giúp xác định các vấn đề tiềm ẩn và cho phép xử lý lỗi chủ động và phục hồi.

4. **Điểm Quyết định và Kết quả**: Giám sát các điểm quyết định trong quy trình và kết quả của các quyết định được hỗ trợ bởi AI. Điều này cung cấp cái nhìn sâu sắc về hành vi và hiệu quả của các thành phần AI.

Dữ liệu được thu thập bởi quá trình giám sát có thể được hiển thị trong các bảng điều khiển hoặc được sử dụng làm đầu vào cho các báo cáo định kỳ để thông báo cho quản trị viên hệ thống về tình trạng của hệ thống.

Dữ liệu giám sát có thể được cung cấp cho một quy trình quản trị viên hệ thống được hỗ trợ bởi AI để xem xét và có hành động tiềm năng!

Ghi nhật ký Sự kiện và Quyết định Quan trọng

Ghi nhật ký là một thực hành thiết yếu liên quan đến việc nắm bắt và lưu trữ thông tin liên quan về các sự kiện quan trọng, quyết định và ngoại lệ xảy ra trong quá trình thực thi quy trình.

Một số khía cạnh quan trọng cần ghi nhật ký bao gồm:

1. **Khởi tạo và Hoàn thành Quy trình**: Ghi lại thời điểm bắt đầu và kết thúc của mỗi phiên bản quy trình, cùng với bất kỳ metadata liên quan như dữ liệu đầu vào và ngữ cảnh người dùng.

2. **Thực thi Thành phần**: Ghi lại chi tiết thực thi của mỗi thành phần quy trình, bao gồm các tham số đầu vào, kết quả đầu ra và bất kỳ dữ liệu trung gian nào được tạo ra.

3. **Quyết định và Lý luận của AI**: Ghi lại các quyết định được đưa ra bởi các thành phần AI, cùng với lý luận cơ bản hoặc điểm tin cậy. Điều này cung cấp tính minh bạch và cho phép kiểm toán các quyết định được hỗ trợ bởi AI.

4. **Ngoại lệ và Thông báo Lỗi**: Ghi lại bất kỳ ngoại lệ hoặc thông báo lỗi nào gặp phải trong quá trình thực thi quy trình, bao gồm vết ngăn xếp và thông tin ngữ cảnh liên quan.

Ghi nhật ký có thể được triển khai bằng nhiều kỹ thuật khác nhau, như ghi vào tệp nhật ký, lưu trữ nhật ký trong cơ sở dữ liệu, hoặc gửi nhật ký đến một dịch vụ ghi nhật ký tập trung. Điều quan trọng là chọn một framework ghi nhật ký cung cấp tính linh hoạt, khả năng mở rộng và tích hợp dễ dàng với kiến trúc của ứng dụng.

Đây là một ví dụ về cách triển khai ghi nhật ký trong ứng dụng Ruby on Rails sử dụng lớp `ActiveSupport::Logger`:

```ruby
class WorkflowLogger
  def self.log(message, severity = :info)
    @logger ||= ActiveSupport::Logger.new('workflow.log')
    @logger.formatter ||= proc do |severity, datetime, progname, msg|
      "#{datetime} [#{severity}] #{msg}\n"
    end
    @logger.send(severity, message)
  end
end

# Usage example
WorkflowLogger.log("Workflow initiated for order ##{@order.id}")
WorkflowLogger.log("Payment processing completed successfully")
WorkflowLogger.log("Inventory check failed for item ##{item.id}", :error)
```

Bằng cách chiến lược đặt các câu lệnh ghi nhật ký trong suốt các thành phần quy trình và các điểm ra quyết định của AI, các nhà phát triển có thể thu thập thông tin giá trị để gỡ lỗi, kiểm toán và phân tích.

Lợi ích của Giám sát và Ghi nhật ký

Việc triển khai giám sát và ghi nhật ký trong điều phối quy trình thông minh mang lại nhiều lợi ích:

1. **Gỡ lỗi và Khắc phục sự cố**: Nhật ký chi tiết và dữ liệu giám sát giúp các nhà phát triển nhanh chóng xác định và chẩn đoán vấn đề. Chúng cung cấp thông tin chi tiết về luồng thực thi quy trình, tương tác giữa các thành phần, và mọi lỗi hoặc ngoại lệ gặp phải.

2. **Tối ưu hóa Hiệu suất**: Việc giám sát các chỉ số hiệu suất cho phép các nhà phát triển xác định các điểm nghẽn và tối ưu hóa các thành phần quy trình để đạt hiệu quả tốt hơn. Bằng cách phân tích thời gian thực thi, việc sử dụng tài nguyên và các chỉ số khác, các nhà phát triển có thể đưa ra quyết định sáng suốt để cải thiện hiệu suất tổng thể của hệ thống.

3. **Kiểm toán và Tuân thủ**: Việc ghi nhật ký các sự kiện và quyết định quan trọng tạo ra một dấu vết kiểm toán để đảm bảo tuân thủ quy định và trách nhiệm giải trình. Điều này cho phép các tổ chức theo dõi và xác minh các hành động được thực hiện bởi các thành phần AI và đảm bảo tuân thủ các quy tắc kinh doanh và yêu cầu pháp lý.

4. **Cải tiến Liên tục**: Dữ liệu giám sát và ghi nhật ký đóng vai trò là đầu vào quan trọng cho việc cải tiến liên tục các quy trình thông minh. Bằng cách phân tích dữ liệu lịch sử, xác định các mẫu, và đo lường hiệu quả của các quyết định AI, các nhà phát triển có thể liên tục tinh chỉnh và nâng cao logic điều phối quy trình.

Các Cân nhắc và Thực tiễn Tốt nhất

Khi triển khai giám sát và ghi nhật ký trong điều phối quy trình thông minh, hãy cân nhắc các thực tiễn tốt nhất sau:

1. **Xác định Rõ các Chỉ số Giám sát**: Xác định các chỉ số và sự kiện chính cần được giám sát dựa trên yêu cầu cụ thể của quy trình. Tập trung vào các chỉ số cung cấp thông tin có ý nghĩa về hiệu suất, tình trạng và hành vi của hệ thống.

2. **Triển khai Ghi nhật ký Chi tiết**: Đảm bảo rằng các câu lệnh ghi nhật ký được đặt tại các điểm thích hợp trong các thành phần quy trình và điểm ra quyết định của AI. Ghi lại thông tin ngữ cảnh liên quan, như tham số đầu vào, kết quả đầu ra và mọi dữ liệu trung gian được tạo ra.

3. **Sử dụng Ghi nhật ký Có cấu trúc**: Áp dụng định dạng ghi nhật ký có cấu trúc để thuận tiện cho việc phân tích và xử lý dữ liệu nhật ký. Ghi nhật ký có cấu trúc cho phép tìm kiếm, lọc và tổng hợp các mục nhật ký tốt hơn.

4. **Quản lý Lưu trữ và Luân chuyển Nhật ký**: Triển khai các chính sách lưu trữ và luân chuyển nhật ký để quản lý việc lưu trữ và vòng đời của các tệp nhật ký. Xác định thời gian lưu trữ phù hợp dựa trên yêu cầu pháp lý, giới hạn lưu trữ và nhu cầu phân tích. Nếu có thể, chuyển việc ghi nhật ký sang dịch vụ bên thứ ba như Papertrail.

5. **Bảo mật Thông tin Nhạy cảm**: Thận trọng khi ghi nhật ký thông tin nhạy cảm, như thông tin nhận dạng cá nhân (PII) hoặc dữ liệu kinh doanh bí mật. Triển khai các biện pháp bảo mật thích hợp, như che giấu dữ liệu hoặc mã hóa, để bảo vệ thông tin nhạy cảm trong các tệp nhật ký.

6. **Tích hợp với Công cụ Giám sát và Cảnh báo**: Tận dụng các công cụ giám sát và cảnh báo để tập trung hóa việc thu thập, phân tích và trực quan hóa dữ liệu giám sát và ghi nhật ký. Các công cụ này có thể cung cấp thông tin chi tiết theo thời gian thực, tạo cảnh báo dựa trên ngưỡng đã định trước và tạo điều kiện cho việc phát hiện và giải quyết sự cố chủ động. Công cụ yêu thích của tôi trong số này là Datadog.

Bằng cách triển khai các cơ chế giám sát và ghi nhật ký toàn diện, các nhà phát triển có thể có được những hiểu biết quý giá về hành vi và hiệu suất của các quy trình thông minh. Những hiểu biết này cho phép gỡ lỗi hiệu quả, tối ưu hóa và cải tiến liên tục các hệ thống điều phối quy trình được hỗ trợ bởi AI.

Các Cân nhắc về Khả năng Mở rộng và Hiệu suất

Khả năng mở rộng và hiệu suất là những khía cạnh quan trọng cần xem xét khi thiết kế và triển khai các hệ thống điều phối quy trình thông minh. Khi khối lượng quy trình đồng thời và độ phức tạp của các thành phần được hỗ trợ bởi AI tăng lên, việc đảm bảo hệ thống có thể xử lý khối lượng công việc hiệu quả và mở rộng liền mạch để đáp ứng nhu cầu ngày càng tăng trở nên thiết yếu.

Xử lý Khối lượng Lớn Quy trình Đồng thời

Các hệ thống điều phối quy trình thông minh thường cần xử lý một số lượng lớn quy trình đồng thời. Để đảm bảo khả năng mở rộng, hãy xem xét các chiến lược sau:

1. **Xử lý Bất đồng bộ**: Triển khai các cơ chế xử lý bất đồng bộ để tách rời việc thực thi các thành phần quy trình. Điều này cho phép hệ thống xử lý nhiều quy trình đồng thời mà không bị chặn hoặc phải chờ đợi mỗi thành phần hoàn thành. Xử lý bất đồng bộ có thể đạt được bằng cách sử dụng hàng đợi tin nhắn, kiến trúc hướng sự kiện, hoặc các framework xử lý công việc nền như Sidekiq.

2. **Kiến trúc Phân tán**: Thiết kế kiến trúc hệ thống để sử dụng các thành phần serverless (như AWS Lambda) hoặc đơn giản là phân phối khối lượng công việc trên nhiều nút hoặc máy chủ cùng với máy chủ ứng dụng chính của bạn. Điều này cho phép khả năng mở rộng theo chiều ngang, trong đó các nút bổ sung có thể được thêm vào để xử lý khối lượng quy trình tăng lên.

3. **Thực thi Song song**: Xác định các cơ hội thực thi song song trong quy trình. Một số thành phần quy trình có thể độc lập với nhau và có thể được thực thi đồng thời. Bằng cách tận dụng các kỹ thuật xử lý song song, như đa luồng hoặc hàng đợi tác vụ phân tán, hệ thống có thể tối ưu hóa việc sử dụng tài nguyên và giảm thời gian thực thi quy trình tổng thể.

Tối Ưu Hóa Hiệu Suất của Các Thành Phần Chạy bằng AI

Các thành phần chạy bằng AI, như các mô hình học máy hoặc công cụ xử lý ngôn ngữ tự nhiên, có thể tiêu tốn nhiều tài nguyên tính toán và ảnh hưởng đến hiệu suất tổng thể của hệ thống điều phối quy trình. Để tối ưu hóa hiệu suất của các thành phần AI, hãy xem xét các kỹ thuật sau:

1. **Bộ nhớ đệm**: Nếu việc xử lý AI của bạn thuần túy là tạo sinh và không liên quan đến tra cứu thông tin thời gian thực hoặc tích hợp bên ngoài để tạo ra các phản hồi chat, thì

bạn có thể xem xét các cơ chế bộ nhớ đệm để lưu trữ và tái sử dụng kết quả của các thao tác được truy cập thường xuyên hoặc tốn nhiều tài nguyên tính toán.

2. **Tối ưu hóa mô hình**: Liên tục tối ưu hóa cách bạn sử dụng các mô hình AI trong các thành phần quy trình. Điều này có thể bao gồm các kỹ thuật như *Tinh lọc Prompt* hoặc đơn giản chỉ là việc thử nghiệm các mô hình mới khi chúng được phát hành.

3. **Xử lý hàng loạt**: Nếu bạn đang làm việc với các mô hình cấp GPT-4, bạn có thể tận dụng các kỹ thuật xử lý hàng loạt để xử lý nhiều điểm dữ liệu hoặc yêu cầu trong một lô, thay vì xử lý chúng riêng lẻ. Bằng cách xử lý dữ liệu theo lô, hệ thống có thể tối ưu hóa việc sử dụng tài nguyên và giảm thiểu chi phí phát sinh từ các yêu cầu mô hình lặp đi lặp lại.

Giám Sát và Lập Hồ Sơ Hiệu Suất

Để xác định các điểm nghẽn hiệu suất và tối ưu hóa khả năng mở rộng của hệ thống điều phối quy trình thông minh, việc triển khai các cơ chế giám sát và lập hồ sơ là rất quan trọng. Hãy xem xét các phương pháp sau:

1. **Chỉ số hiệu suất**: Xác định và theo dõi các chỉ số hiệu suất quan trọng, như thời gian phản hồi, thông lượng, mức độ sử dụng tài nguyên và độ trễ. Những chỉ số này cung cấp cái nhìn sâu sắc về hiệu suất của hệ thống và giúp xác định các lĩnh vực cần tối ưu hóa. Nền tảng tổng hợp mô hình AI phổ biến OpenRouter bao gồm các chỉ số Host[1] và Speed[2] trong mỗi phản hồi API, giúp việc theo dõi các chỉ số quan trọng này trở nên đơn giản.

2. **Công cụ lập hồ sơ**: Sử dụng các công cụ lập hồ sơ để phân tích hiệu suất của các thành phần quy trình và hoạt động AI riêng lẻ. Các công cụ lập hồ sơ có thể giúp xác định các điểm nóng về hiệu suất, đường dẫn mã không hiệu quả, hoặc các hoạt động tiêu tốn nhiều tài nguyên. Các công cụ lập hồ sơ phổ biến bao gồm New Relic, Scout,

[1]Host là thời gian cần thiết để nhận byte đầu tiên của quá trình tạo dữ liệu được truyền từ máy chủ mô hình, còn gọi là "thời gian đến byte đầu tiên."

[2]Speed được tính bằng số lượng token hoàn thành chia cho tổng thời gian tạo. Đối với các yêu cầu không truyền trực tuyến, độ trễ được coi là một phần của thời gian tạo.

hoặc các công cụ lập hồ sơ tích hợp sẵn được cung cấp bởi ngôn ngữ lập trình hoặc framework.

3. **Kiểm thử tải**: Tiến hành kiểm thử tải để đánh giá hiệu suất của hệ thống dưới các mức tải đồng thời khác nhau. Kiểm thử tải giúp xác định giới hạn khả năng mở rộng của hệ thống, phát hiện sự suy giảm hiệu suất, và đảm bảo rằng hệ thống có thể xử lý lưu lượng dự kiến mà không ảnh hưởng đến hiệu suất.

4. **Giám sát liên tục**: Triển khai các cơ chế giám sát và cảnh báo liên tục để chủ động phát hiện các vấn đề và điểm nghẽn về hiệu suất. Thiết lập bảng điều khiển giám sát và cảnh báo để theo dõi các chỉ số hiệu suất chính (KPI) và nhận thông báo khi vượt quá ngưỡng đã định. Điều này cho phép xác định và giải quyết nhanh chóng các vấn đề về hiệu suất.

Chiến Lược Mở Rộng

Để xử lý khối lượng công việc ngày càng tăng và đảm bảo khả năng mở rộng của hệ thống điều phối quy trình thông minh, hãy xem xét các chiến lược mở rộng sau:

1. **Mở rộng theo chiều dọc**: Mở rộng theo chiều dọc bao gồm việc tăng tài nguyên (ví dụ: CPU, bộ nhớ) của các nút hoặc máy chủ riêng lẻ để xử lý khối lượng công việc lớn hơn. Phương pháp này phù hợp khi hệ thống cần nhiều sức mạnh xử lý hoặc bộ nhớ hơn để xử lý các quy trình hoặc hoạt động AI phức tạp.

2. **Mở rộng theo chiều ngang**: Mở rộng theo chiều ngang bao gồm việc thêm nhiều nút hoặc máy chủ vào hệ thống để phân phối khối lượng công việc. Phương pháp này hiệu quả khi hệ thống cần xử lý số lượng lớn quy trình đồng thời hoặc khi khối lượng công việc có thể dễ dàng phân phối trên nhiều nút. Mở rộng theo chiều ngang đòi hỏi kiến trúc phân tán và cơ chế cân bằng tải để đảm bảo phân phối lưu lượng đồng đều.

3. **Tự động mở rộng**: Triển khai các cơ chế tự động mở rộng để tự động điều chỉnh số lượng nút hoặc tài nguyên dựa trên nhu cầu khối lượng công việc. Tự động mở rộng cho phép hệ thống linh hoạt tăng hoặc giảm quy mô tùy thuộc vào lưu lượng truy cập đến,

đảm bảo sử dụng tài nguyên tối ưu và hiệu quả về chi phí. Các nền tảng đám mây như Amazon Web Services (AWS) hoặc Google Cloud Platform (GCP) cung cấp khả năng tự động mở rộng có thể được tận dụng cho các hệ thống điều phối quy trình thông minh.

Kỹ Thuật Tối Ưu Hóa Hiệu Suất

Ngoài các chiến lược mở rộng, hãy xem xét các kỹ thuật tối ưu hóa hiệu suất sau đây để nâng cao hiệu quả của hệ thống điều phối quy trình thông minh:

1. **Lưu trữ và truy xuất dữ liệu hiệu quả**: Tối ưu hóa các cơ chế lưu trữ và truy xuất dữ liệu được sử dụng bởi các thành phần quy trình. Sử dụng lập chỉ mục cơ sở dữ liệu hiệu quả, các kỹ thuật tối ưu hóa truy vấn, và bộ nhớ đệm dữ liệu để giảm thiểu độ trễ và cải thiện hiệu suất của các hoạt động xử lý dữ liệu lớn.

2. **I/O Không đồng bộ**: Sử dụng các thao tác I/O không đồng bộ để tránh việc chặn và cải thiện khả năng phản hồi của hệ thống. I/O không đồng bộ cho phép hệ thống xử lý nhiều yêu cầu đồng thời mà không cần đợi các thao tác I/O hoàn thành, từ đó tối đa hóa việc sử dụng tài nguyên.

3. **Tuần tự hóa và Giải tuần tự hóa Hiệu quả**: Tối ưu hóa các quy trình tuần tự hóa và giải tuần tự hóa được sử dụng cho việc trao đổi dữ liệu giữa các thành phần quy trình. Sử dụng các định dạng tuần tự hóa hiệu quả như Protocol Buffers hoặc MessagePack để giảm thiểu chi phí tuần tự hóa dữ liệu và cải thiện hiệu suất giao tiếp giữa các thành phần.

Đối với các ứng dụng dựa trên Ruby, hãy cân nhắc sử dụng Universal ID. Universal ID tận dụng cả MessagePack và Brotli (sự kết hợp được xây dựng để đạt tốc độ và khả năng nén dữ liệu tốt nhất). Khi kết hợp, các thư viện này nhanh hơn tới 30% và có tỷ lệ nén chỉ kém 2-5% so với Protocol Buffers.

4. **Nén và Mã hóa**: Áp dụng các kỹ thuật nén và mã hóa để giảm kích thước dữ liệu được truyền giữa các thành phần quy trình. Các thuật toán nén như gzip hoặc Brotli có

thể giảm đáng kể việc sử dụng băng thông mạng và cải thiện hiệu suất tổng thể của hệ thống.

Bằng cách xem xét các khía cạnh về khả năng mở rộng và hiệu suất trong quá trình thiết kế và triển khai các hệ thống điều phối quy trình thông minh, bạn có thể đảm bảo rằng hệ thống của mình có thể xử lý khối lượng lớn các quy trình đồng thời, tối ưu hóa hiệu suất của các thành phần được hỗ trợ bởi AI, và mở rộng một cách liền mạch để đáp ứng nhu cầu ngày càng tăng. Việc giám sát, lập hồ sơ và tối ưu hóa liên tục là cần thiết để duy trì hiệu suất và khả năng đáp ứng của hệ thống khi khối lượng công việc và độ phức tạp tăng lên theo thời gian.

Kiểm thử và Xác thực Quy trình

Kiểm thử và xác thực là các khía cạnh quan trọng trong việc phát triển và duy trì các hệ thống điều phối quy trình thông minh. Do bản chất phức tạp của các quy trình được hỗ trợ bởi AI, việc đảm bảo mỗi thành phần hoạt động như mong đợi, toàn bộ quy trình vận hành chính xác, và các quyết định của AI chính xác và đáng tin cậy là điều thiết yếu. Trong phần này, chúng ta sẽ khám phá các kỹ thuật và cân nhắc khác nhau để kiểm thử và xác thực các quy trình thông minh.

Kiểm thử Đơn vị cho các Thành phần Quy trình

Kiểm thử đơn vị bao gồm việc kiểm tra các thành phần quy trình riêng lẻ để xác minh tính chính xác và độ tin cậy của chúng. Khi kiểm thử đơn vị các thành phần quy trình được hỗ trợ bởi AI, hãy xem xét những điểm sau:

1. **Xác thực Đầu vào**: Kiểm tra khả năng xử lý các loại đầu vào khác nhau của thành phần, bao gồm cả dữ liệu hợp lệ và không hợp lệ. Xác minh rằng thành phần xử lý một cách nhẹ nhàng các trường hợp biên và cung cấp thông báo lỗi hoặc ngoại lệ phù hợp.

2. **Xác minh Đầu ra**: Khẳng định rằng thành phần tạo ra đầu ra mong đợi cho một tập

hợp đầu vào nhất định. So sánh đầu ra thực tế với kết quả mong đợi để đảm bảo tính chính xác.

3. **Xử lý Lỗi**: Kiểm tra các cơ chế xử lý lỗi của thành phần bằng cách mô phỏng các tình huống lỗi khác nhau, như đầu vào không hợp lệ, tài nguyên không khả dụng, hoặc các ngoại lệ không mong đợi. Xác minh rằng thành phần bắt và xử lý lỗi một cách phù hợp.

4. **Điều kiện Biên**: Kiểm tra hành vi của thành phần trong các điều kiện biên, như đầu vào rỗng, kích thước đầu vào tối đa, hoặc các giá trị cực đoan. Đảm bảo rằng thành phần xử lý các điều kiện này một cách nhẹ nhàng mà không bị sập hoặc tạo ra kết quả không chính xác.

Dưới đây là một ví dụ về kiểm thử đơn vị cho một thành phần quy trình bằng Ruby sử dụng framework kiểm thử RSpec:

```
RSpec.describe OrderValidator do
  describe '#validate' do
    context 'when order is valid' do
      let(:order) { build(:order) }

      it 'returns true' do
        expect(subject.validate(order)).to be true
      end
    end

    context 'when order is invalid' do
      let(:order) { build(:order, total_amount: -100) }

      it 'returns false' do
        expect(subject.validate(order)).to be false
      end
    end
  end
end
```

Trong ví dụ này, thành phần `OrderValidator` được kiểm thử bằng hai trường hợp thử nghiệm: một cho đơn hàng hợp lệ và một cho đơn hàng không hợp lệ. Các trường hợp

thử nghiệm xác minh rằng phương thức `validate` trả về giá trị boolean như mong đợi dựa trên tính hợp lệ của đơn hàng.

Kiểm Thử Tích Hợp Tương Tác Quy Trình

Kiểm thử tích hợp tập trung vào việc xác minh các tương tác và luồng dữ liệu giữa các thành phần khác nhau trong quy trình làm việc. Nó đảm bảo các thành phần hoạt động cùng nhau một cách mượt mà và tạo ra kết quả như mong đợi. Khi thực hiện kiểm thử tích hợp cho các quy trình làm việc thông minh, cần xem xét những điểm sau:

1. **Tương Tác Giữa Các Thành Phần**: Kiểm thử việc giao tiếp và trao đổi dữ liệu giữa các thành phần trong quy trình. Xác minh rằng đầu ra của một thành phần được truyền chính xác như đầu vào cho thành phần tiếp theo trong quy trình.

2. **Tính Nhất Quán Dữ Liệu**: Đảm bảo dữ liệu luôn nhất quán và chính xác khi di chuyển qua quy trình. Xác minh rằng các phép biến đổi dữ liệu, tính toán và tổng hợp được thực hiện chính xác.

3. **Lan Truyền Ngoại Lệ**: Kiểm thử cách thức lan truyền và xử lý các ngoại lệ và lỗi qua các thành phần của quy trình. Xác minh rằng các ngoại lệ được bắt, ghi nhận và xử lý phù hợp để tránh gián đoạn quy trình.

4. **Hành Vi Bất Đồng Bộ**: Nếu quy trình có các thành phần bất đồng bộ hoặc thực thi song song, hãy kiểm thử các cơ chế phối hợp và đồng bộ hóa. Đảm bảo quy trình hoạt động chính xác trong các tình huống đồng thời và bất đồng bộ.

Dưới đây là một ví dụ về kiểm thử tích hợp cho một quy trình trong Ruby sử dụng framework kiểm thử RSpec:

```ruby
RSpec.describe OrderProcessingWorkflow do

  let(:order) { build(:order) }

  it 'processes the order successfully' do
    expect(OrderValidator).to receive(:validate).and_return(true)
    expect(InventoryManager).to receive(:check_availability).and_return(true)
    expect(PaymentProcessor).to receive(:process_payment).and_return(true)
    expect(ShippingService).to receive(:schedule_shipping).and_return(true)

    workflow = OrderProcessingWorkflow.new(order)
    result = workflow.process

    expect(result).to be true
    expect(order.status).to eq('processed')
  end

end
```

Trong ví dụ này, `OrderProcessingWorkflow` được kiểm thử bằng cách xác minh các tương tác giữa các thành phần khác nhau của quy trình làm việc. Trường hợp kiểm thử thiết lập các kỳ vọng cho hành vi của từng thành phần và đảm bảo rằng quy trình xử lý đơn hàng thành công, cập nhật trạng thái đơn hàng một cách phù hợp.

Kiểm thử các điểm quyết định AI

Kiểm thử các điểm quyết định AI là việc cực kỳ quan trọng để đảm bảo độ chính xác và độ tin cậy của các quy trình làm việc được hỗ trợ bởi AI. Khi kiểm thử các điểm quyết định AI, cần xem xét những điểm sau:

1. **Độ chính xác của quyết định**: Xác minh rằng thành phần AI đưa ra quyết định chính xác dựa trên dữ liệu đầu vào và mô hình đã được huấn luyện. So sánh các quyết định của AI với kết quả mong đợi hoặc dữ liệu chuẩn.

2. **Các trường hợp biên**: Kiểm thử hành vi của thành phần AI trong các trường hợp biên và các tình huống bất thường. Xác minh rằng thành phần AI xử lý các trường hợp này một cách ổn thỏa và đưa ra các quyết định hợp lý.

3. **Độ thiên lệch và tính công bằng**: Đánh giá thành phần AI về các thiên lệch tiềm ẩn và đảm bảo rằng nó đưa ra các quyết định công bằng và không thiên vị. Kiểm thử thành phần với dữ liệu đầu vào đa dạng và phân tích kết quả để tìm ra các mẫu phân biệt đối xử.

4. **Khả năng giải thích**: Nếu thành phần AI cung cấp các giải thích hoặc lý luận cho quyết định của nó, hãy xác minh tính chính xác và rõ ràng của các giải thích đó. Đảm bảo rằng các giải thích phù hợp với quy trình ra quyết định cơ bản.

Dưới đây là một ví dụ về kiểm thử điểm quyết định AI trong Ruby sử dụng framework kiểm thử RSpec:

```
RSpec.describe FraudDetector do
  describe '#detect_fraud' do
    context 'when transaction is fraudulent' do
      let(:tx) do
        build(:transaction, amount: 10_000, location: 'High-Risk Country')
      end

      it 'returns true' do
        expect(subject.detect_fraud(tx)).to be true
      end
    end

    context 'when transaction is legitimate' do
      let(:tx) do
        build(:transaction, amount: 100, location: 'Low-Risk Country')
      end

      it 'returns false' do
        expect(subject.detect_fraud(tx)).to be false
      end
    end
  end
end
```

Trong ví dụ này, thành phần AI `FraudDetector` được kiểm thử với hai trường hợp: một cho giao dịch gian lận và một cho giao dịch hợp lệ. Các trường hợp kiểm thử xác

minh rằng phương thức `detect_fraud` trả về giá trị boolean như mong đợi dựa trên các đặc điểm của giao dịch.

Kiểm thử Đầu-cuối

Kiểm thử đầu-cuối bao gồm việc kiểm tra toàn bộ quy trình làm việc từ đầu đến cuối, mô phỏng các tình huống thực tế và tương tác của người dùng. Nó đảm bảo rằng quy trình làm việc hoạt động chính xác và tạo ra kết quả mong muốn. Khi thực hiện kiểm thử đầu-cuối cho các quy trình làm việc thông minh, cần xem xét những điểm sau:

1. **Kịch bản Người dùng**: Xác định các kịch bản người dùng phổ biến và kiểm tra hành vi của quy trình làm việc trong các kịch bản này. Xác minh rằng quy trình làm việc xử lý đúng đầu vào của người dùng, đưa ra quyết định phù hợp và tạo ra đầu ra như mong đợi.

2. **Xác thực Dữ liệu**: Đảm bảo rằng quy trình làm việc xác thực và làm sạch đầu vào của người dùng để ngăn chặn sự không nhất quán dữ liệu hoặc các lỗ hổng bảo mật. Kiểm tra quy trình làm việc với nhiều loại dữ liệu đầu vào khác nhau, bao gồm cả dữ liệu hợp lệ và không hợp lệ.

3. **Khôi phục Lỗi**: Kiểm tra khả năng khôi phục của quy trình làm việc từ các lỗi và ngoại lệ. Mô phỏng các tình huống lỗi và xác minh rằng quy trình làm việc xử lý chúng một cách nhẹ nhàng, ghi lại các lỗi và thực hiện các hành động khôi phục thích hợp.

4. **Hiệu năng và Khả năng Mở rộng**: Đánh giá hiệu năng và khả năng mở rộng của quy trình làm việc trong các điều kiện tải khác nhau. Kiểm tra quy trình làm việc với khối lượng lớn các yêu cầu đồng thời và đo thời gian phản hồi, mức sử dụng tài nguyên và độ ổn định tổng thể của hệ thống.

Dưới đây là một ví dụ về kiểm thử đầu-cuối cho một quy trình làm việc trong Ruby sử dụng framework kiểm thử RSpec và thư viện Capybara để mô phỏng tương tác người dùng:

```
RSpec.describe 'Order Processing Workflow' do
  scenario 'User places an order successfully' do
    visit '/orders/new'
    fill_in 'Product', with: 'Sample Product'
    fill_in 'Quantity', with: '2'
    fill_in 'Shipping Address', with: '123 Main St'
    click_button 'Place Order'

    expect(page).to have_content('Order Placed Successfully')
    expect(Order.count).to eq(1)
    expect(Order.last.status).to eq('processed')
  end
end
```

Trong ví dụ này, kiểm thử đầu-cuối mô phỏng người dùng đặt hàng thông qua giao diện web. Nó điền vào các trường biểu mẫu cần thiết, gửi đơn hàng và xác minh rằng đơn hàng được xử lý thành công, hiển thị thông báo xác nhận phù hợp và cập nhật trạng thái đơn hàng trong cơ sở dữ liệu.

Tích hợp và Triển khai Liên tục

Để đảm bảo độ tin cậy và khả năng bảo trì của các quy trình làm việc thông minh, chúng tôi khuyến nghị tích hợp việc kiểm thử và xác thực vào pipeline tích hợp và triển khai liên tục (CI/CD). Điều này cho phép kiểm thử và xác thực tự động các thay đổi quy trình làm việc trước khi chúng được triển khai vào môi trường sản xuất. Hãy xem xét các phương pháp sau:

1. **Thực thi Kiểm thử Tự động**: Cấu hình pipeline CI/CD để tự động chạy bộ kiểm thử mỗi khi có thay đổi trong mã nguồn quy trình làm việc. Điều này đảm bảo rằng mọi lỗi hồi quy hoặc thất bại đều được phát hiện sớm trong quá trình phát triển.

2. **Giám sát Độ bao phủ Kiểm thử**: Đo lường và theo dõi độ bao phủ kiểm thử của các thành phần quy trình làm việc và các điểm quyết định AI. Hướng tới độ bao phủ kiểm thử cao để đảm bảo các đường dẫn và kịch bản quan trọng được kiểm thử kỹ lưỡng.

3. **Phản hồi Liên tục**: Tích hợp kết quả kiểm thử và các chỉ số chất lượng mã vào quy trình phát triển. Cung cấp phản hồi liên tục cho các nhà phát triển về trạng thái kiểm thử, chất lượng mã và mọi vấn đề được phát hiện trong quá trình CI/CD.

4. **Môi trường Dàn dựng**: Triển khai quy trình làm việc vào môi trường dàn dựng phản ánh chính xác môi trường sản xuất. Thực hiện kiểm thử và xác thực bổ sung trong môi trường dàn dựng để phát hiện mọi vấn đề liên quan đến cơ sở hạ tầng, cấu hình hoặc tích hợp dữ liệu.

5. **Cơ chế Khôi phục**: Triển khai các cơ chế khôi phục trong trường hợp triển khai thất bại hoặc phát hiện các vấn đề nghiêm trọng trong môi trường sản xuất. Đảm bảo rằng quy trình làm việc có thể nhanh chóng được khôi phục về phiên bản ổn định trước đó để giảm thiểu thời gian ngừng hoạt động và tác động đến người dùng.

Bằng cách kết hợp kiểm thử và xác thực xuyên suốt vòng đời phát triển của các quy trình làm việc thông minh, các tổ chức có thể đảm bảo độ tin cậy, tính chính xác và khả năng bảo trì của hệ thống được hỗ trợ bởi AI của họ. Kiểm thử và xác thực thường xuyên giúp phát hiện lỗi, ngăn chặn hồi quy và xây dựng sự tin tưởng vào hành vi và kết quả của quy trình làm việc.

Phần 2: Các Mẫu Thiết Kế

Kỹ thuật thiết kế prompt

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Chuỗi Suy luận

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Cách Hoạt động

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Ví dụ

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Tạo Nội dung

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Tạo Thực Thể Có Cấu Trúc

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Hướng dẫn Tác tử LLM

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Lợi ích và Cân nhắc

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Chuyển đổi chế độ

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Cách Thức Hoạt Động

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Khi Nào Nên Sử Dụng

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Ví dụ

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Gán Vai trò

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Cách Thức Hoạt động

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Khi nào Nên Sử dụng

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Ví dụ

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Đối tượng Prompt

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Cách Thức Hoạt Động

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Mẫu Lời Nhắc

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Cách Thức Hoạt Động

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Lợi Ích và Cân Nhắc

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Khi Nào Nên Sử Dụng:

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Ví dụ

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Structured IO

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Cách Thức Hoạt Động

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Mở rộng Structured IO

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Lợi ích và Cân nhắc

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Chuỗi Lệnh

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Cách Thức Hoạt Động

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Khi nào nên sử dụng

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Ví dụ: Quy trình tiếp nhận của Olympia

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Trình Viết Lại Lệnh Gợi Ý

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Cách Thức Hoạt Động

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Ví dụ

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Rào chắn phản hồi

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Cách hoạt động

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Lợi ích và Cân nhắc

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Xử Lý Lỗi

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Bộ Phân Tích Truy Vấn

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Cách Thức Hoạt Động

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Triển Khai

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Gán Nhãn Từ Loại (POS) và Nhận Dạng Thực Thể Có Tên (NER)

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Phân loại Ý định

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Trích xuất từ khóa

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Lợi ích

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Bộ viết lại truy vấn

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Cách thức hoạt động

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Ví dụ

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Lợi ích

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Ventriloquist

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Cách Hoạt Động

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Khi Nào Nên Sử Dụng

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Ví dụ

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Các Thành Phần Rời Rạc

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Vị từ

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Cách Thức Hoạt Động

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Khi nào Sử dụng

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Ví dụ

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Façade API

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Cách Hoạt Động

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Lợi Ích Chính

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Khi Nào Nên Sử Dụng

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Ví dụ

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Xác thực và Ủy quyền

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Xử lý Yêu cầu

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Định dạng Phản hồi

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Xử lý Lỗi và Các Trường hợp Biên

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Cân nhắc về Khả năng Mở rộng và Hiệu suất

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

So sánh với Các Mẫu Thiết kế Khác

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Bộ Diễn Giải Kết Quả

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Cách Hoạt Động

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Khi Nào Nên Sử Dụng

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Ví dụ

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Máy Ảo

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Cách Hoạt Động

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Khi Nào Nên Sử Dụng

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Ví dụ

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Đằng Sau Điều Kỳ Diệu

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Đặc tả và Kiểm thử

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Đặc tả Hành vi

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Viết Ca Kiểm thử

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Ví dụ: Kiểm thử Thành phần Translator

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Phát Lại Tương Tác HTTP

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Sự Can Thiệp của Con Người (HITL)

Các Mẫu Cấp Cao

Trí Tuệ Lai

Phản Hồi Thích Ứng

Chuyển đổi Vai trò Người-AI

Quy trình Leo thang

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Cách Thức Hoạt động

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Lợi ích Chính

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Ứng dụng thực tế: Chăm sóc sức khỏe

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Vòng phản hồi

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Cách Thức Hoạt Động

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Ứng dụng và Ví dụ

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Kỹ thuật nâng cao trong Tích hợp Phản hồi của Con người

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Bức xạ thông tin thụ động

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Cách thức hoạt động

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Hiển thị thông tin theo ngữ cảnh

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Thông báo chủ động

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Thông tin giải thích

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Khám phá tương tác

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Lợi ích chính

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Ứng dụng và ví dụ

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Ra quyết định cộng tác (CDM)

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Cách thức hoạt động

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Ví dụ

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Học Liên tục

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Cách Thức Hoạt Động

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Ứng dụng và Ví dụ

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Ví dụ

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Cân nhắc về đạo đức

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Vai trò của HITL trong việc giảm thiểu rủi ro AI

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Tiến bộ Công nghệ và Triển vọng Tương lai

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Thách thức và Hạn chế của Hệ thống HITL

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Xử Lý Lỗi Thông Minh

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Các Phương Pháp Xử Lý Lỗi Truyền Thống

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Chẩn đoán Lỗi theo Ngữ cảnh

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Cách Thức Hoạt động

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Kỹ thuật Thiết kế Prompt cho Chẩn đoán Lỗi theo Ngữ cảnh

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Sinh nội dung có truy xuất bổ sung cho Chẩn đoán lỗi theo ngữ cảnh

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Báo cáo lỗi thông minh

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Phòng ngừa Lỗi Dự đoán

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Cách Thức Hoạt Động

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Khôi phục lỗi thông minh

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Cách Thức Hoạt Động

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Giao tiếp lỗi được cá nhân hóa

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Cách thức hoạt động

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Quy trình xử lý lỗi thích ứng

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Cách Thức Hoạt Động

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Kiểm soát chất lượng

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Eval

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Vấn đề

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Giải pháp

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Cách Hoạt động

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Ví dụ

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Các vấn đề cần cân nhắc

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Hiểu về Chuẩn mực vàng

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Cách thức hoạt động của Đánh giá không cần tham chiếu

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Cơ chế bảo vệ

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Vấn đề

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Giải pháp

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Cách Thức Hoạt Động

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Ví dụ

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Các điểm cần cân nhắc

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Thanh chắn bảo vệ và Đánh giá: Hai Mặt của Một Đồng Xu

Tính Thay Thế Giữa Thanh Chắn Bảo Vệ và Đánh Giá Không Cần Tham Chiếu

Triển khai Thanh Chắn Bảo Vệ và Đánh giá Lưỡng Dụng

Thuật ngữ

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Thuật ngữ

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

A

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

B

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

C

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

D

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

E

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

F

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

G

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

H

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

I

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

J

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

K

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

L

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

M

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

N

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

O

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

P

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Q

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

R

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

S

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

T

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

U

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

V

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

W

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Z

Nội dung này không có trong phiên bản mẫu sách. Bạn có thể mua sách trên Leanpub tại http://leanpub.com/patterns-of-application-development-using-ai-vi.

Index