

The background of the cover is a dark blue-purple gradient, densely populated with various colorful geometric shapes and patterns. These include circles in shades of orange, yellow, green, pink, and blue. There are also elongated, capsule-like shapes in orange, yellow, and green, some with internal patterns. A white line drawing of a leaf or a stylized arrow is visible in the upper left. A network of white dots connected by lines is on the left side. A large, stylized 'X' shape is formed by overlapping geometric shapes in the center. The overall aesthetic is modern and tech-oriented.

Patterns of Application Development Using AI

Obie Fernandez

Foreword by Gregor Hohpe



Leanpub

Edição em Português

Padrões de Desenvolvimento de Aplicações Usando IA (Edição em Português)

Obie Fernandez

Esse livro está à venda em

<http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>

Essa versão foi publicada em 2025-01-23



Esse é um livro [Leanpub](#). A Leanpub dá poderes aos autores e editores a partir do processo de Publicação Lean. [Publicação Lean](#) é a ação de publicar um ebook em desenvolvimento com ferramentas leves e muitas iterações para conseguir feedbacks dos leitores, pivotar até que você tenha o livro ideal e então conseguir tração.

© 2025 Obie Fernandez

Tweet Sobre Esse Livro!

Por favor ajude Obie Fernandez a divulgar esse livro no [Twitter](#)!

A hashtag sugerida para esse livro é [#poaduai](#).

Descubra o que as outras pessoas estão falando sobre esse livro clicando nesse link para buscar a hashtag no Twitter:

[#poaduai](#)

Para a minha incrível rainha, minha musa, minha luz e amor, Victoria

Outras Obras De **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](#)

Conteúdo

Prefácio por Gregor Hohpe	i
Prefácio	ii
Sobre o Livro	iii
Sobre os Exemplos de Código	iii
O Que Não Abordo	iii
Para Quem É Este Livro	iii
Construindo um Vocabulário Comum	iii
Como Participar	iii
Agradecimentos	iii
O que há com as ilustrações?	iv
Sobre a Publicação Lean	iv
Sobre o Autor	v
Introdução	1
Reflexões sobre Arquitetura de Software	2
O que é um Modelo de Linguagem de Larga Escala?	3
Compreendendo a Inferência	5
Pensando Sobre o Desempenho	27
Experimentando com Diferentes Modelos LLM	29
Sistemas de IA Compostos	30

Parte 1: Abordagens e Técnicas Fundamentais

38

Estreitar o Caminho	39
Espaço Latente: Incompreensivelmente Vasto	41
Como o Caminho é “Estreitado”	45
Modelos Base Versus Modelos Ajustados por Instruções	49
Engenharia de Prompts	56
Destilação de Prompts	72
E quanto ao ajuste fino?	79
Retrieval Augmented Generation (RAG)	81
O que é a Geração Aumentada por Recuperação?	81
Como Funciona o RAG?	81
Por Que Usar RAG nas Suas Aplicações?	81
Implementação do RAG na Sua Aplicação	81
Segmentação por Proposições	82
Exemplos Práticos de RAG	82
Otimização Inteligente de Consultas (OIC)	83
Reclassificação	83
Avaliação RAG (RAGAs)	83
Desafios e Perspectivas Futuras	85
Multidão de Trabalhadores	87
Trabalhadores de IA Como Componentes Reutilizáveis Independentes	88
Gestão de Contas	90
Aplicações de Comércio Eletrônico	91
Aplicações na Área da Saúde	100
O Worker de IA como Process Manager	103
Integração de Workers de IA na Arquitetura da Sua Aplicação	107

CONTEÚDO

Composição e Orquestração de Workers de IA	111
Combinando PLN Tradicional com LLMs	120
Uso de Ferramentas	123
O que é o Uso de Ferramentas?	123
O Potencial da Utilização de Ferramentas	125
O Fluxo de Trabalho da Utilização de Ferramentas	126
Melhores Práticas para o Uso de Ferramentas	141
Composição e Encadeamento de Ferramentas	146
Direções Futuras	148
Processamento de Streams	150
Implementação de um ReplyStream	151
O “Ciclo de Conversação”	157
Continuação Automática	159
Conclusão	162
Dados Auto-Reparáveis	163
Caso Prático: Corrigir JSON Defeituoso	165
Considerações e Contraindicações	170
Geração de Conteúdo Contextual	186
Personalização	187
Produtividade	189
Iteração Rápida e Experimentação	191
Localização Potenciada por IA	194
A Importância dos Testes com Utilizadores e do Feedback	196
Generative UI	197
Gerar Texto para Interfaces de Utilizador	198
Definindo a UI Generativa	208

CONTEÚDO

Exemplo 210

A Mudança para o Design Orientado a Resultados 213

Desafios e Considerações 214

Perspetivas Futuras e Oportunidades 216

Orquestração Inteligente de Fluxos de Trabalho 219

 Necessidade de Negócio 220

 Principais Benefícios 221

 Padrões Principais 222

 Tratamento e Recuperação de Exceções 224

 Implementação Prática da Orquestração de Fluxos de Trabalho Inteligentes . . 227

 Monitorização e Registo de Eventos 242

 Considerações de Escalabilidade e Desempenho 247

 Testes e Validação de Workflows 252

Parte 2: Os Padrões 260

Engenharia de Prompts 261

 Chain of Thought 262

 Mudança de Modo 263

 Atribuição de Função 264

 Objeto Prompt 265

 Template de Prompt 266

 Structured IO 267

 Encadeamento de Prompts 268

 Prompt Rewriter 269

 Response Fencing 270

 Analisador de Consultas 271

 Query Rewriter 273

 Ventriloquist 274

CONTEÚDO

Componentes Discretos	275
Predicado	276
API Facade	277
Interpretador de Resultados	279
Máquina Virtual	280
Especificação e Testes	280
Human In The Loop (HITL)	282
Padrões de Alto Nível	282
Escalonamento	283
Ciclo de Feedback	284
Radiação Passiva de Informação	285
Tomada de Decisão Colaborativa (TDC)	287
Aprendizagem Contínua	288
Considerações Éticas	288
Avanços Tecnológicos e Perspetivas Futuras	288
Tratamento Inteligente de Erros	290
Abordagens Tradicionais de Tratamento de Erros	290
Diagnóstico Contextual de Erros	291
Relatórios Inteligentes de Erros	292
Prevenção Preditiva de Erros	293
Recuperação Inteligente de Erros	293
Comunicação de Erros Personalizada	294
Fluxo de Trabalho Adaptativo de Tratamento de Erros	295
Controlo de Qualidade	296
Eval	297
Barreira de Proteção	299
Guardrails e Evals: Dois Lados da Mesma Moeda	299

Glossário 301
 Glossário 301
Index 306

Prefácio por Gregor Hohpe

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Prefácio

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Sobre o Livro

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Sobre os Exemplos de Código

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

O Que Não Abordo

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Para Quem É Este Livro

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Construindo um Vocabulário Comum

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Como Participar

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Agradecimentos

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

O que há com as ilustrações?

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

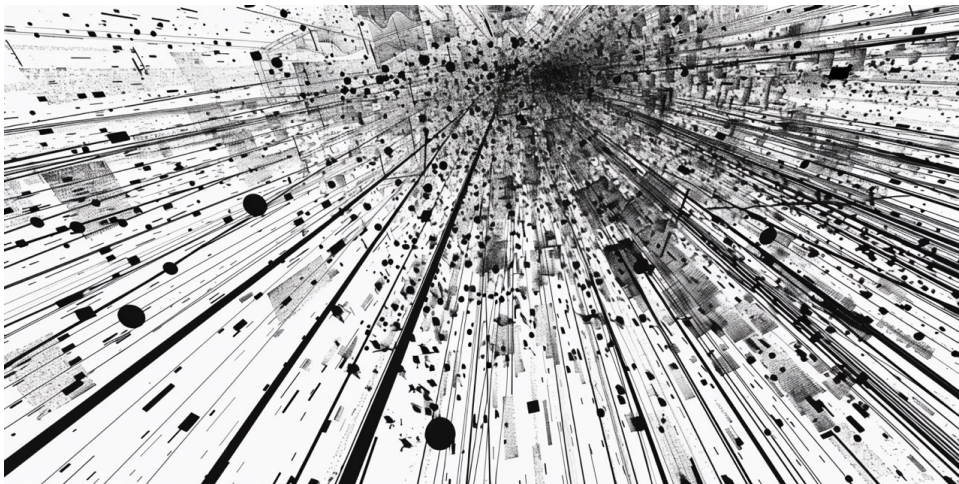
Sobre a Publicação Lean

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Sobre o Autor

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Introdução



Se está ansioso para começar a integrar Modelos de Linguagem de Larga Escala (LLMs) nos seus projetos de programação, sinta-se à vontade para mergulhar diretamente nos padrões e exemplos de código apresentados nos capítulos seguintes. No entanto, para apreciar plenamente o poder e o potencial destes padrões, vale a pena dedicar um momento para compreender o contexto mais amplo e a abordagem coesa que representam.

Os padrões não são meramente uma coleção de técnicas isoladas, mas sim uma estrutura unificada para integrar a IA nas suas aplicações. Eu utilizo Ruby on Rails, mas estes padrões devem funcionar em praticamente qualquer outro ambiente de programação. Eles abordam uma ampla gama de preocupações, desde a gestão de dados e otimização de desempenho até à experiência do utilizador e segurança, fornecendo um conjunto abrangente de ferramentas para melhorar as práticas de programação tradicionais com as capacidades da IA.

Cada categoria de padrões aborda um desafio ou oportunidade específica que surge ao incorporar componentes de IA na sua aplicação. Ao compreender as relações e sinergias entre estes padrões, pode tomar decisões informadas sobre onde e como aplicar a IA de forma mais eficaz.

Os padrões nunca são soluções prescritivas e não devem ser tratados como tal. São blocos de construção adaptáveis que devem ser ajustados aos requisitos e restrições únicos da sua própria aplicação. A aplicação bem-sucedida destes padrões (como quaisquer outros na área de software) depende de uma compreensão profunda do domínio do problema, das necessidades do utilizador e da arquitetura técnica global do seu projeto.

Reflexões sobre Arquitetura de Software

Comecei a programar nos anos 80 e estive envolvido na cena hacker, e nunca perdi a minha mentalidade hacker, mesmo depois de me tornar um programador profissional. Desde o início, sempre tive um ceticismo saudável sobre que valor os arquitetos de software nas suas torres de marfim realmente traziam para a mesa.

Uma das razões pelas quais estou pessoalmente tão entusiasmado com as mudanças trazidas por esta poderosa nova vaga de tecnologia de IA é o seu impacto no que consideramos decisões de *arquitetura de software*. Desafia as noções tradicionais do que constitui a forma “correta” de desenhar e implementar os nossos projetos de software. Também questiona se a arquitetura ainda pode ser pensada principalmente como *as partes de um sistema que são difíceis de alterar*, uma vez que o aprimoramento com IA está a tornar mais fácil do que nunca alterar qualquer parte do seu projeto, a qualquer momento.

Talvez estejamos a entrar nos anos de auge da abordagem “pós-moderna” à engenharia de software. Neste contexto, pós-moderno refere-se a uma mudança fundamental que se afasta dos paradigmas tradicionais, onde os programadores eram responsáveis por escrever e manter cada linha de código. Em vez disso, abraça a ideia de delegar

tarefas, como manipulação de dados, algoritmos complexos e até mesmo partes inteiras da lógica da aplicação, a bibliotecas de terceiros e APIs externas. Esta mudança pós-moderna representa um afastamento significativo da sabedoria convencional de construir aplicações desde a base, e desafia os programadores a repensar o seu papel no processo de desenvolvimento.

Sempre acreditei que os bons programadores só escrevem o código que é absolutamente necessário escrever, baseando-me nos ensinamentos de Larry Wall e outros luminares hacker como ele. Ao minimizar a quantidade de código escrito, podemos avançar mais rapidamente, reduzir a superfície para bugs, simplificar a manutenção e melhorar a fiabilidade geral das suas aplicações. Menos código permite-nos focar na lógica de negócio central e na experiência do utilizador, enquanto delegamos outro trabalho para outros serviços.

Agora que os sistemas alimentados por IA podem lidar com tarefas que eram anteriormente do domínio exclusivo do código escrito por humanos, devemos poder ser ainda mais produtivos e ágeis, com um foco maior do que nunca na criação de valor para o negócio e experiência do utilizador.

Claro que existem contrapartidas em delegar grandes partes do seu projeto a sistemas de IA, como a potencial perda de controlo e a necessidade de mecanismos robustos de monitorização e feedback. É por isso que requer um novo conjunto de competências e conhecimentos, incluindo pelo menos alguma compreensão fundamental de como a IA funciona.

O que é um Modelo de Linguagem de Larga Escala?

Os Modelos de Linguagem de Larga Escala (LLMs) são um tipo de modelo de inteligência artificial que ganhou atenção significativa nos últimos anos, desde o lançamento do GPT-3 pela OpenAI em 2020. Os LLMs são projetados para processar, compreender

e gerar linguagem humana com notável precisão e fluência. Nesta secção, vamos dar uma breve olhada em como os LLMs funcionam e por que são adequados para construir componentes de sistemas inteligentes.

No seu núcleo, os LLMs são baseados em algoritmos de aprendizagem profunda, especificamente redes neuronais. Estas redes são compostas por nós interconectados, ou neurónios, que processam e transmitem informação. A arquitetura de escolha para LLMs é frequentemente o modelo Transformer, que provou ser altamente eficaz no tratamento de dados sequenciais como texto.

Os modelos Transformer baseiam-se no mecanismo de atenção e são principalmente utilizados em tarefas que envolvem dados sequenciais, como o processamento de linguagem natural. Os Transformers processam dados de entrada todos de uma vez, em vez de sequencialmente, o que lhes permite captar dependências de longo alcance de forma mais eficaz. Possuem camadas de mecanismos de atenção que ajudam o modelo a focar-se em diferentes partes dos dados de entrada para compreender o contexto e as relações.

O processo de treino dos LLMs envolve expor o modelo a vastas quantidades de dados textuais, como livros, artigos, websites e repositórios de código. Durante o treino, o modelo aprende a reconhecer padrões, relações e estruturas dentro do texto. Capta as propriedades estatísticas da linguagem, como regras gramaticais, associações de palavras e significados contextuais.

Uma das técnicas principais utilizadas no treino de LLMs é a aprendizagem não supervisionada. Isto significa que o modelo aprende a partir dos dados sem rotulagem ou orientação explícita. Descobre padrões e representações por si próprio ao analisar a co-ocorrência de palavras e frases nos dados de treino. Isto permite aos LLMs desenvolver uma compreensão profunda da linguagem e das suas complexidades.

Outro aspeto importante dos LLMs é a sua capacidade de lidar com o *contexto*. Ao processar um texto, os LLMs consideram não apenas as palavras individuais, mas também o contexto envolvente. Têm em conta as palavras, frases e até parágrafos

anteriores para compreender o significado e a intenção do texto. Esta compreensão contextual permite aos LLMs gerar respostas coerentes e relevantes. Uma das principais formas de avaliarmos as capacidades de um determinado modelo LLM é considerando o tamanho do contexto que podem considerar para gerar respostas.

Uma vez treinados, os LLMs podem ser utilizados numa ampla gama de tarefas relacionadas com linguagem. Podem gerar texto semelhante ao humano, responder a perguntas, resumir documentos, traduzir idiomas e até escrever código. A versatilidade dos LLMs torna-os valiosos para construir componentes de sistemas inteligentes que podem interagir com utilizadores, processar e analisar dados textuais e gerar resultados significativos.

Ao incorporar LLMs na arquitetura da aplicação, pode criar componentes de IA que compreendem e processam a entrada do utilizador, geram conteúdo dinâmico e fornecem recomendações ou ações inteligentes. Mas trabalhar com LLMs requer uma consideração cuidadosa dos requisitos de recursos e compromissos de desempenho. Os LLMs são computacionalmente intensivos e podem requerer poder de processamento e memória significativos (por outras palavras, dinheiro) para operar. A maioria de nós precisará de avaliar as implicações de custo da integração de LLMs nas nossas aplicações e agir em conformidade.

Compreendendo a Inferência

A inferência refere-se ao processo pelo qual um modelo gera previsões ou resultados baseados em dados novos e não vistos. É a fase em que o modelo treinado é usado para tomar decisões ou gerar texto, imagens ou outro conteúdo em resposta às entradas do utilizador.

Durante a fase de treino, um modelo de IA aprende a partir de um grande conjunto de dados ajustando os seus parâmetros para minimizar o erro nas suas previsões. Uma vez treinado, o modelo pode aplicar o que aprendeu a novos dados. A inferência é como o modelo usa os seus padrões e conhecimentos aprendidos para gerar resultados.

Para os LLMs, a inferência envolve receber um prompt ou texto de entrada e produzir uma resposta coerente e contextualmente relevante, como um fluxo de *tokens* (dos quais falaremos em breve). Isto pode ser responder a uma pergunta, completar uma frase, gerar uma história ou traduzir texto, entre muitas outras tarefas.



Em contraste com a forma como você e eu pensamos, o “pensamento” de um modelo de IA via inferência acontece todo numa única operação sem estado. Isto é, o seu pensamento está limitado ao seu processo de geração. Literalmente tem de pensar em voz alta, como se eu lhe fizesse uma pergunta e só aceitasse uma resposta sua no estilo “fluxo de consciência”.

Os Modelos de Linguagem de Grande Escala Vêm em Muitos Tamanhos e Sabores

Embora praticamente todos os modelos de linguagem de grande escala (LLMs) populares sejam baseados na mesma arquitetura transformer central e treinados em enormes conjuntos de dados textuais, eles vêm numa variedade de tamanhos e são aperfeiçoados para diferentes propósitos. O tamanho de um LLM, medido pelo número de parâmetros na sua rede neuronal, tem um grande impacto nas suas capacidades. Modelos maiores com mais parâmetros, como o GPT-4, que se rumora ter entre 1 a 2 biliões de parâmetros, são geralmente mais conhecedores e capazes do que modelos menores. No entanto, modelos maiores também requerem muito mais poder computacional para executar, o que se traduz em maiores despesas quando os utiliza através de chamadas API.

Para tornar os LLMs mais práticos e adaptados a casos de uso específicos, os modelos base são frequentemente aperfeiçoados em conjuntos de dados mais direcionados. Por exemplo, um LLM pode ser treinado num grande corpus de diálogo para se especializar em IA conversacional. Outros são [treinados em código](#) para lhes inculcir conhecimento de programação. Existem até modelos que são [especialmente treinados para interações de roleplay com utilizadores!](#)

Modelos de Recuperação vs Modelos Generativos

No mundo dos modelos linguísticos de grande escala (MLGs), existem duas abordagens principais para gerar respostas: modelos baseados em recuperação e modelos generativos. Cada abordagem tem os seus pontos fortes e fracos, e compreender as diferenças entre elas pode ajudá-lo a escolher o modelo mais adequado para o seu caso específico.

Modelos Baseados em Recuperação

Os modelos baseados em recuperação, também conhecidos como modelos de recuperação de informação, geram respostas através da pesquisa numa grande base de dados de texto pré-existente e selecionam as passagens mais relevantes com base na consulta de entrada. Estes modelos não geram novo texto do zero, mas sim conjugam excertos da base de dados para formar uma resposta coerente.

Uma das principais vantagens dos modelos baseados em recuperação é a sua capacidade de fornecer informações factualmente precisas e atualizadas. Como dependem de uma base de dados de texto curada, podem extrair informações relevantes de fontes fiáveis e apresentá-las ao utilizador. Isto torna-os adequados para aplicações que exigem respostas precisas e factuais, como sistemas de pergunta e resposta ou bases de conhecimento.

No entanto, os modelos baseados em recuperação têm algumas limitações. Eles são tão bons quanto a base de dados que pesquisam, pelo que a qualidade e a cobertura da base de dados afetam diretamente o desempenho do modelo. Além disso, estes modelos podem ter dificuldade em gerar respostas coerentes e naturais, pois estão limitados ao texto disponível na base de dados.

Não abordamos a utilização de modelos de recuperação puros neste livro.

Modelos Generativos

Os modelos generativos, por outro lado, criam novo texto do zero com base nos padrões e relações que aprenderam durante o treino. Estes modelos utilizam a sua compreensão da linguagem para gerar respostas originais que são adaptadas ao prompt de entrada.

O principal ponto forte dos modelos generativos é a sua capacidade de produzir texto criativo, coerente e contextualmente relevante. Podem participar em conversas abertas, gerar histórias e até escrever código. Isto torna-os ideais para aplicações que requerem interações mais abertas e dinâmicas, como chatbots, criação de conteúdo e assistentes de escrita criativa.

No entanto, os modelos generativos podem por vezes produzir informações inconsistentes ou factualmente incorretas, já que dependem dos padrões aprendidos durante o treino em vez de uma base de dados curada de factos. Podem também ser mais propensos a enviesamentos e alucinações, gerando texto que é plausível mas não necessariamente verdadeiro.

Exemplos de MLGs generativos incluem a série GPT da OpenAI (GPT-3, GPT-4) e o Claude da Anthropic.

Modelos Híbridos

Vários MLGs disponíveis comercialmente combinam as abordagens de recuperação e generativa num modelo híbrido. Estes modelos utilizam técnicas de recuperação para encontrar informações relevantes numa base de dados e depois utilizam técnicas generativas para sintetizar essa informação numa resposta coerente.

Os modelos híbridos visam combinar a precisão factual dos modelos baseados em recuperação com as capacidades de geração de linguagem natural dos modelos generativos. Podem fornecer informações mais fiáveis e atualizadas, mantendo simultaneamente a capacidade de participar em conversas abertas.

Ao escolher entre modelos baseados em recuperação e modelos generativos, deve considerar os requisitos específicos da sua aplicação. Se o objetivo principal é fornecer informações precisas e factuais, um modelo baseado em recuperação pode ser a melhor escolha. Se a aplicação requer interações mais abertas e criativas, um modelo generativo pode ser mais adequado. Os modelos híbridos oferecem um equilíbrio entre as duas abordagens e podem ser uma boa escolha para aplicações que requerem tanto precisão factual como geração de linguagem natural.

Em última análise, a escolha entre modelos baseados em recuperação e modelos generativos depende do caso de uso específico e dos compromissos entre precisão, criatividade e flexibilidade. Ao compreender os pontos fortes e as limitações de cada abordagem, pode tomar decisões informadas.

Ajuste por Instruções

O ajuste por instruções é um subconjunto do treino conversacional que utiliza prompts e respostas escritas por humanos para tornar os MLGs melhores a compreender e seguir instruções. Aqui estão alguns exemplos simples para ilustrar como o ajuste por instruções em prompts e respostas escritas por humanos torna os MLGs melhores a compreender e seguir instruções:

Exemplo 1: Apoio ao Cliente

Instrução: “Forneça uma resposta a um cliente que solicita um reembolso.”

Prompt Escrito por Humano: “Gostaria de solicitar um reembolso da minha compra recente. O produto não correspondeu às minhas expectativas.”

Resposta Escrita por Humano: “Lamento saber que o produto não correspondeu às suas expectativas. Para processar o seu reembolso, por favor forneça o número da sua encomenda e o motivo da devolução. Analisaremos o seu pedido e responderemos dentro de 2-3 dias úteis.”

Exemplo 2: Assistência Educacional

Instrução: “Explique o conceito de fotossíntese a uma criança de 10 anos.”

Prompt Escrito por Humano: “Pode explicar como as plantas produzem o seu alimento?”

Resposta Escrita por Humano: “Claro! As plantas produzem o seu alimento através de um processo chamado fotossíntese. Elas usam a luz solar, água e dióxido de carbono para criar alimento e oxigénio. As folhas da planta têm uma parte especial chamada clorofila que as ajuda a captar a luz solar.”

Exemplo 3: Assistente Pessoal

Instrução: “Agende uma reunião para terça-feira que vem às 15h.”

Prompt Escrito por Humano: “Pode marcar uma reunião para terça-feira às 15h?”

Resposta Escrita por Humano: “Claro, agendei uma reunião para terça-feira que vem às 15h. Precisa de mais alguma coisa?”

O resultado é um ecossistema diversificado de LLMs em diferentes tamanhos e com diferentes especialidades. Modelos mais pequenos na faixa de 1-7 mil milhões de parâmetros fornecem boas capacidades gerais de linguagem, sendo mais eficientes de executar.

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

Modelos de tamanho médio com cerca de 30-70 mil milhões de parâmetros oferecem capacidades mais fortes de raciocínio e de seguimento de instruções.

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

Ao escolher um LLM para incorporar numa aplicação, é necessário equilibrar as capacidades do modelo com fatores práticos como custo, latência, comprimento do contexto e filtragem de conteúdo. Modelos mais pequenos, ajustados por instruções, são frequentemente a melhor escolha para tarefas de linguagem mais simples, enquanto os maiores modelos podem ser necessários para raciocínio ou análise complexos. Os dados de treino do modelo são também uma consideração importante, pois determinam a data limite de conhecimento do modelo.



Certos modelos, como alguns da Perplexity, estão ligados a fontes de informação em tempo real, pelo que efetivamente não têm data limite. Quando lhes fazemos perguntas, eles conseguem decidir independentemente fazer pesquisas na web e obter páginas web arbitrárias para gerar uma resposta.

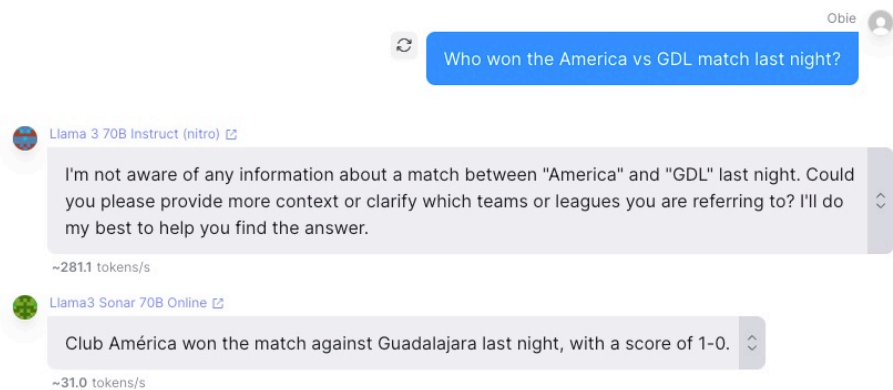


Figura 1. Llama3 com e sem acesso online

Em última análise, não existe um LLM que sirva para tudo. Compreender as variações no tamanho do modelo, arquitetura e treino é fundamental para selecionar o modelo certo para um determinado caso de uso. Experimentar diferentes modelos é a única forma prática de revelar quais fornecem o melhor desempenho para a tarefa em questão.

Tokenização: Dividindo o Texto em Partes

Antes que um modelo de linguagem grande possa processar texto, esse texto precisa de ser dividido em unidades menores chamadas *tokens*. Os tokens podem ser palavras individuais, partes de palavras ou até caracteres individuais. O processo de dividir texto em tokens é conhecido como tokenização, e é um passo crucial na preparação de dados para um modelo de linguagem.

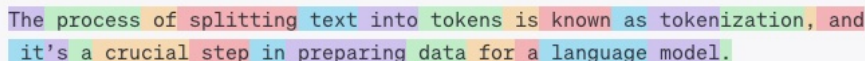


Figura 2. Esta frase contém 27 tokens

Diferentes LLMs usam diferentes estratégias de tokenização, que podem ter um impacto significativo no desempenho e nas capacidades do modelo. Alguns tokenizadores comuns usados por LLMs incluem:

- **GPT (Byte Pair Encoding):** Os tokenizadores GPT usam uma técnica chamada byte pair encoding (BPE) para dividir texto em unidades de subpalavras. O BPE funde iterativamente os pares de bytes mais frequentes num corpus de texto, formando um vocabulário de tokens de subpalavras. Isto permite que o tokenizador lide com palavras raras e novas, dividindo-as em partes de subpalavras mais comuns. Os tokenizadores GPT são usados por modelos como o GPT-3 e o GPT-4.
- **Llama (SentencePiece):** Os tokenizadores Llama utilizam a biblioteca SentencePiece, que é um tokenizador e destokenizador de texto não supervisionado. O SentencePiece trata o texto de entrada como uma sequência de caracteres Unicode e aprende um vocabulário de subpalavras com base num corpus de treino. Pode processar qualquer idioma que possa ser codificado em Unicode, tornando-o adequado para modelos multilingues. Os tokenizadores Llama são utilizados por modelos como o Llama e o Alpaca da Meta.
- **SentencePiece (Unigram):** Os tokenizadores SentencePiece também podem utilizar um algoritmo diferente chamado Unigram, que se baseia numa técnica de regularização de subpalavras. A tokenização Unigram determina o vocabulário ótimo de subpalavras com base num modelo de linguagem unigrama, que atribui probabilidades a unidades individuais de subpalavras. Esta abordagem pode produzir subpalavras mais significativas semanticamente em comparação com o

BPE. O SentencePiece com Unigram é utilizado por modelos como o T5 e o BERT da Google.

- **Google Gemini (Tokenização Multimodal):** O Google Gemini utiliza um esquema de tokenização projetado para lidar com vários tipos de dados, incluindo texto, imagens, áudio, vídeos e código. Esta capacidade multimodal permite ao Gemini processar e integrar diferentes formas de informação. Notavelmente, o Google Gemini 1.5 Pro tem uma janela de contexto que pode processar milhões de tokens, muito maior do que os modelos anteriores. Esta extensa janela de contexto permite ao modelo processar um contexto maior, potencialmente levando a respostas mais precisas. No entanto, é importante notar que o esquema de tokenização do Gemini está muito mais próximo de um token por caractere do que outros modelos. Isto significa que o custo real de utilizar modelos Gemini pode ser significativamente maior do que o esperado se estiver acostumado a usar modelos como o GPT, já que o preço da Google é baseado em caracteres em vez de tokens.

A escolha do tokenizador afeta vários aspectos de um LLM, incluindo:

- **Tamanho do vocabulário:** O tokenizador determina o tamanho do vocabulário do modelo, que é o conjunto de tokens únicos que ele reconhece. Um vocabulário maior e mais detalhado pode ajudar o modelo a lidar com uma gama mais ampla de palavras e frases e até mesmo tornar-se multimodal (capaz de compreender e gerar mais do que apenas texto), mas também aumenta os requisitos de memória e a complexidade computacional do modelo.
- **Tratamento de palavras raras e desconhecidas:** Tokenizadores que utilizam unidades de subpalavras, como BPE e SentencePiece, podem decompor palavras raras e desconhecidas em fragmentos de subpalavras mais comuns. Isto permite que o modelo faça suposições educadas sobre o significado de palavras que não viu antes, com base nas subpalavras que contém.

- **Suporte multilingue:** Tokenizadores como o SentencePiece, que podem lidar com qualquer linguagem codificável em Unicode, são adequados para modelos multilingues que precisam de processar texto em vários idiomas.

Ao escolher um LLM para uma aplicação específica, é importante considerar o tokenizador que ele utiliza e quão bem se alinha com as necessidades específicas de processamento de linguagem da tarefa em questão. O tokenizador pode ter um impacto significativo na capacidade do modelo de lidar com terminologia específica do domínio, palavras raras e texto multilingue.

Tamanho do Contexto: Quanta Informação Pode um Modelo de Linguagem Utilizar Durante a Inferência?

Ao discutir modelos de linguagem, o tamanho do contexto refere-se à quantidade de texto que um modelo pode considerar ao processar ou gerar as suas respostas. É essencialmente uma medida de quanta informação o modelo pode “lembrar” e usar para informar as suas saídas (expressa em tokens). O tamanho do contexto de um modelo de linguagem pode ter um impacto significativo nas suas capacidades e nos tipos de tarefas que pode realizar efetivamente.

O que é o Tamanho do Contexto?

Em termos técnicos, o tamanho do contexto é determinado pelo número de tokens (palavras ou fragmentos de palavras) que um modelo de linguagem pode processar numa única sequência de entrada. Isto é frequentemente referido como a “amplitude de atenção” ou “janela de contexto” do modelo. Quanto maior o tamanho do contexto, mais texto o modelo pode considerar de uma vez ao gerar uma resposta ou realizar uma tarefa.

Diferentes modelos de linguagem têm tamanhos de contexto variados, indo de algumas centenas de tokens a milhões de tokens. Como referência, um parágrafo típico de texto

pode conter cerca de 100-150 tokens, enquanto um livro inteiro pode conter dezenas ou centenas de milhares de tokens.

Existe inclusive trabalho em métodos eficientes para escalar Modelos de Linguagem Grandes (LLMs) baseados em Transformers para [entradas infinitamente longas](#) com memória e computação limitadas.

Por que o Tamanho do Contexto é Importante?

O tamanho do contexto de um modelo de linguagem tem um impacto significativo na sua capacidade de compreender e gerar texto coerente e contextualmente relevante. Aqui estão algumas razões principais pelas quais o tamanho do contexto é importante:

1. **Compreensão de conteúdo longo:** Modelos com tamanhos de contexto maiores podem compreender e analisar melhor textos mais longos, como artigos, relatórios ou até livros inteiros. Isto é crucial para tarefas como sumarização de documentos, resposta a perguntas e análise de conteúdo.
2. **Manutenção da coerência:** Uma janela de contexto maior permite que o modelo mantenha a coerência e consistência ao longo de trechos mais longos de texto. Isto é importante para tarefas como geração de histórias, sistemas de diálogo e criação de conteúdo, onde manter uma narrativa ou tópico consistente é essencial. Também é absolutamente crucial quando se utilizam LLMs para gerar ou transformar dados estruturados.
3. **Captura de dependências de longo alcance:** Algumas tarefas de linguagem requerem a compreensão de relações entre palavras ou frases que estão distantes num texto. Modelos com tamanhos de contexto maiores estão melhor equipados para capturar estas dependências de longo alcance, que podem ser importantes para tarefas como análise de sentimento, tradução e compreensão de linguagem.

4. **Gestão de instruções complexas:** Em aplicações onde os modelos de linguagem são usados para seguir instruções complexas de múltiplos passos, um tamanho de contexto maior permite que o modelo considere o conjunto completo de instruções ao gerar uma resposta, em vez de apenas as palavras mais recentes.

Exemplos de Modelos de Linguagem com Diferentes Tamanhos de Contexto

Aqui estão alguns exemplos de modelos de linguagem com diferentes tamanhos de contexto:

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

Como pode ver, existe uma ampla gama de tamanhos de contexto entre estes modelos, desde cerca de 4.000 tokens para o modelo OpenAI GPT-3.5 Turbo até 200.000 tokens para o modelo Anthropic Claude v2. Alguns modelos, como o PaLM 2 da Google e o GPT-4 da OpenAI, oferecem diferentes variantes com tamanhos de contexto maiores (por exemplo, versões “32k”), que podem lidar com sequências de entrada ainda mais longas. E neste momento (abril de 2024), o Google Gemini Pro está a ostentar quase 3 milhões de tokens!

É importante notar que o tamanho do contexto pode variar dependendo da implementação específica e da versão de um determinado modelo. Por exemplo, o modelo original OpenAI GPT-4 tem um tamanho de contexto de 8.191 tokens, enquanto as variantes posteriores do GPT-4, como o Turbo e o 4o, têm um tamanho de contexto muito maior de 128.000 tokens.

Sam Altman comparou as limitações atuais de contexto aos kilobytes de memória de trabalho com que os programadores de computadores pessoais tinham de lidar nos anos 80, e disse que num futuro próximo seremos capazes de ajustar “todos os seus dados pessoais” no contexto de um modelo de linguagem grande.

Escolhendo o Tamanho de Contexto Adequado

Ao selecionar um modelo de linguagem para uma aplicação específica, é importante considerar os requisitos de tamanho de contexto da tarefa em questão. Para tarefas que envolvem textos curtos e isolados, como análise de sentimento ou resposta simples a perguntas, um tamanho de contexto menor pode ser suficiente. No entanto, para tarefas que requerem compreensão e geração de textos mais longos e complexos, um tamanho de contexto maior será provavelmente necessário.

Vale ressaltar que tamanhos de contexto maiores frequentemente vêm acompanhados de custos computacionais aumentados e tempos de processamento mais lentos, já que o modelo precisa considerar mais informação ao gerar uma resposta. Como tal, deve-se encontrar um equilíbrio entre tamanho de contexto e desempenho ao escolher um modelo de linguagem para a sua aplicação.

Por que não escolher simplesmente o modelo com o maior tamanho de contexto e enchê-lo com o máximo de informação possível? Bem, além dos fatores de desempenho, a outra consideração principal é o custo. Em março de 2024, um *único* ciclo de prompt-resposta usando o Google Gemini Pro 1.5 com um contexto completo custará quase 8 dólares (USD). Se você tem um caso de uso que justifica essa despesa,

mais poder para você! Mas para a maioria das aplicações, é simplesmente demasiado caro por várias ordens de magnitude.

Encontrar Agulhas em Palheiros

O conceito de encontrar uma agulha num palheiro tem sido há muito uma metáfora para os desafios de recuperação em grandes conjuntos de dados. No domínio dos LLMs, adaptamos um pouco esta analogia. Imagine que não estamos apenas à procura de um único facto enterrado num texto vasto (como uma antologia completa de ensaios de Paul Graham), mas sim de múltiplos factos dispersos por todo ele. Este cenário é mais semelhante a encontrar várias agulhas num campo extenso, não apenas num único palheiro. Eis o mais interessante: não só precisamos de localizar estas agulhas, como também temos de as entrelaçar num fio coerente.

Quando confrontados com a tarefa de recuperar e raciocinar sobre múltiplos factos incorporados em contextos longos, os LLMs enfrentam um duplo desafio. Primeiro, há a questão direta da precisão da recuperação—que naturalmente diminui à medida que o número de factos aumenta. Isto é esperado; afinal, manter o controlo de múltiplos detalhes num texto extenso põe à prova até os modelos mais sofisticados.

Segundo, e talvez mais crítico, é o desafio de raciocinar com estes factos. Uma coisa é identificar factos; outra completamente diferente é sintetizá-los numa narrativa ou resposta coerente. É aqui que surge o verdadeiro teste. O desempenho dos LLMs em tarefas de raciocínio tende a degradar-se ainda mais do que em tarefas simples de recuperação. Esta degradação não se deve apenas ao volume; trata-se da complexa dança entre contexto, relevância e inferência.

Por que é que isto acontece? Bem, considere a dinâmica da memória e atenção na cognição humana, que se reflete até certo ponto nos LLMs. Ao processar grandes quantidades de informação, os LLMs, tal como os humanos, podem perder o rasto

de detalhes anteriores à medida que absorvem novos. Isto é especialmente verdade em modelos que não foram explicitamente projetados para priorizar ou revisar automaticamente segmentos anteriores do texto.

Além disso, a capacidade de um LLM entrelaçar estes factos recuperados numa resposta coerente é semelhante à construção narrativa. Isto requer não apenas a recuperação de informação, mas uma compreensão profunda e contextualização, o que continua a ser um grande desafio para a IA atual.

Então, o que significa isto para nós, como desenvolvedores e integradores destas tecnologias? Precisamos de estar profundamente conscientes destas limitações ao projetar sistemas que dependem de LLMs para lidar com tarefas complexas e extensas. Compreender que o desempenho pode degradar-se sob certas condições ajuda-nos a estabelecer expectativas realistas e a desenvolver melhores mecanismos de contingência ou estratégias complementares.

Modalidades: Além do Texto

Embora a maioria dos modelos de linguagem hoje se concentre no processamento e geração de texto, existe uma tendência crescente para modelos multimodais que podem nativamente receber e produzir múltiplos tipos de dados, como imagens, áudio e vídeo. Estes modelos multimodais abrem novas possibilidades para aplicações baseadas em IA que podem compreender e gerar conteúdo através de diferentes modalidades.

O que são Modalidades?

No contexto dos modelos de linguagem, modalidades referem-se aos diferentes tipos de dados que um modelo pode processar e gerar. A modalidade mais comum é o texto, que inclui linguagem escrita em várias formas como livros, artigos, websites e publicações em redes sociais. No entanto, existem várias outras modalidades que estão cada vez mais a ser incorporadas em modelos de linguagem:

- **Imagens:** Dados visuais como fotografias, ilustrações e diagramas.
- **Áudio:** Dados sonoros como fala, música e sons ambientais.
- **Vídeo:** Dados visuais em movimento, frequentemente acompanhados por áudio, como clipes de vídeo e filmes.

Cada modalidade apresenta desafios e oportunidades únicos para os modelos de linguagem. Por exemplo, as imagens requerem que o modelo compreenda conceitos e relações visuais, enquanto o áudio requer que o modelo processe e gere fala e outros sons.

Modelos de Linguagem Multimodais

Os modelos de linguagem multimodais são projetados para lidar com múltiplas modalidades dentro de um único modelo. Estes modelos tipicamente têm componentes ou camadas especializadas que podem tanto compreender entradas como gerar dados de saída em diferentes modalidades. Alguns exemplos notáveis de modelos de linguagem multimodais incluem:

- **OpenAI's GPT-4o:** O GPT-4o é um modelo de linguagem de grande dimensão que compreende e processa nativamente áudio de fala além de texto. Esta capacidade permite ao GPT-4o realizar tarefas como transcrição de linguagem falada, geração de texto a partir de entradas de áudio e fornecimento de respostas baseadas em consultas faladas.
- **OpenAI's GPT-4 com entrada visual:** O GPT-4 é um modelo de linguagem de grande dimensão que pode processar tanto texto como imagens. Quando recebe uma imagem como entrada, o GPT-4 pode analisar o conteúdo da imagem e gerar texto que descreve ou responde à informação visual.
- **Google's Gemini:** O Gemini é um modelo multimodal que pode lidar com texto, imagens e vídeo. Utiliza uma arquitetura unificada que permite a compreensão e geração transmodal, possibilitando tarefas como legendagem de imagens, resumo de vídeos e resposta a questões visuais.

- **DALL-E e Stable Diffusion:** Embora não sejam modelos de linguagem no sentido tradicional, estes modelos demonstram o poder da IA multimodal ao gerar imagens a partir de descrições textuais. Demonstram o potencial dos modelos que podem traduzir entre diferentes modalidades.

Benefícios e Aplicações dos Modelos Multimodais

Os modelos de linguagem multimodais oferecem diversos benefícios e possibilitam uma ampla gama de aplicações, incluindo:

- **Compreensão melhorada:** Ao processar informações de múltiplas modalidades, estes modelos podem obter uma compreensão mais abrangente do mundo, semelhante à forma como os humanos aprendem através de vários inputs sensoriais.
- **Geração cross-modal:** Os modelos multimodais podem gerar conteúdo numa modalidade com base em input de outra, como criar uma imagem a partir de uma descrição textual ou gerar um resumo em vídeo a partir de um artigo escrito.
- **Acessibilidade:** Os modelos multimodais podem tornar a informação mais acessível ao traduzir entre modalidades, como gerar descrições textuais de imagens para utilizadores com deficiência visual ou criar versões em áudio de conteúdo escrito.
- **Aplicações criativas:** Os modelos multimodais podem ser utilizados para tarefas criativas como gerar arte, música ou vídeos com base em instruções textuais, abrindo novas possibilidades para artistas e criadores de conteúdo.

À medida que os modelos de linguagem multimodais continuam a avançar, é provável que desempenhem um papel cada vez mais importante no desenvolvimento de aplicações baseadas em IA que podem compreender e gerar conteúdo através de múltiplas modalidades. Isto permitirá interações mais naturais e intuitivas entre humanos e sistemas de IA, bem como desbloqueará novas possibilidades para expressão criativa e disseminação de conhecimento.

Ecosistemas de Fornecedores

Quando se trata de incorporar modelos de linguagem de grande escala (LLMs) em aplicações, existe uma gama crescente de opções à escolha. Cada grande fornecedor de LLMs, como OpenAI, Anthropic, Google e Cohere, oferece o seu próprio ecossistema de modelos, APIs e ferramentas. Escolher o fornecedor certo envolve considerar vários fatores, incluindo preços, desempenho, filtragem de conteúdo, privacidade de dados e opções de personalização.

OpenAI

A OpenAI é um dos fornecedores de LLMs mais conhecidos, com a sua série GPT (GPT-3, GPT-4) sendo amplamente utilizada em várias aplicações. A OpenAI oferece uma API intuitiva que permite integrar facilmente os seus modelos em aplicações. Disponibilizam uma variedade de modelos com diferentes capacidades e níveis de preço, desde o modelo básico Ada até ao poderoso modelo Davinci.

O ecossistema da OpenAI também inclui ferramentas como o OpenAI Playground, que permite experimentar prompts e fazer fine-tuning dos modelos para casos de uso específicos. Oferecem opções de filtragem de conteúdo para ajudar a prevenir a geração de conteúdo inadequado ou prejudicial.

Quando uso os modelos da OpenAI diretamente, recorro à biblioteca [ruby-openai](#) de Alex Rudall.

Anthropic

A Anthropic é outro grande player no espaço dos LLMs, com os seus modelos Claude a ganhar popularidade pelo forte desempenho e considerações éticas. A Anthropic concentra-se no desenvolvimento de sistemas de IA seguros e responsáveis, com uma forte ênfase na filtragem de conteúdo e na prevenção de outputs prejudiciais.

O ecossistema da Anthropic inclui a API Claude, que permite integrar o modelo nas suas aplicações, bem como ferramentas para engenharia de prompts e fine-tuning. Também oferecem o modelo Claude Instant, que incorpora capacidades de pesquisa na web para respostas mais atualizadas e factuais.

Quando uso os modelos da Anthropic diretamente, recorro à biblioteca [anthropic](#) de Alex Rudall.

Google

O Google desenvolveu vários LLMs poderosos, incluindo Gemini, BERT, T5 e PaLM. Estes modelos são conhecidos pelo seu forte desempenho numa ampla gama de tarefas de processamento de linguagem natural. O ecossistema do Google inclui as bibliotecas TensorFlow e Keras, que fornecem ferramentas e frameworks para construir e treinar modelos de aprendizagem automática.

O Google também oferece uma Plataforma de IA na Nuvem, que permite implementar e escalar facilmente os seus modelos na nuvem. Disponibilizam uma variedade de modelos pré-treinados e APIs para tarefas como análise de sentimento, reconhecimento de entidades e tradução.

Meta

A Meta, anteriormente conhecida como Facebook, está profundamente investida no desenvolvimento de modelos de linguagem de grande escala, destacada pelo lançamento de modelos como LLaMA e OPT. Estes modelos destacam-se pelo seu forte desempenho em diversas tarefas de linguagem e são disponibilizados principalmente através de canais open-source, apoiando o compromisso da Meta com a investigação e colaboração comunitária.

O ecossistema da Meta é construído principalmente em torno do PyTorch, uma biblioteca de aprendizagem automática open-source preferida pelas suas capacidades

computacionais dinâmicas e flexibilidade, facilitando a investigação e desenvolvimento inovador em IA.

Para além das suas ofertas técnicas, a Meta coloca uma forte ênfase no desenvolvimento ético de IA. Implementam uma filtragem de conteúdo robusta e focam-se na redução de preconceitos, alinhando-se com os seus objetivos mais amplos de segurança e responsabilidade nas aplicações de IA.

Cohere

A Cohere é uma empresa mais recente no espaço dos LLM, focando-se em tornar os LLMs mais acessíveis e fáceis de usar do que a concorrência. O seu ecossistema inclui a API Cohere, que fornece acesso a uma gama de modelos pré-treinados para tarefas como geração de texto, classificação e resumo.

A Cohere também oferece ferramentas para engenharia de prompts, ajuste fino e filtragem de conteúdo. Enfatizam a privacidade e segurança dos dados, com funcionalidades como armazenamento encriptado de dados e controlos de acesso.

Ollama

O Ollama é uma plataforma auto-hospedada que permite aos utilizadores gerir e implementar vários modelos de linguagem grandes (LLMs) localmente nas suas máquinas, dando-lhes controlo total sobre os seus modelos de IA sem depender de serviços em nuvem externos. Esta configuração é ideal para quem prioriza a privacidade dos dados e deseja gerir as suas operações de IA internamente.

A plataforma suporta uma variedade de modelos, incluindo versões do Llama, Phi, Gemma e Mistral, que variam em tamanho e requisitos computacionais. O Ollama facilita o download e execução destes modelos diretamente da linha de comandos usando comandos simples como `ollama run <model_name>`, e foi projetado para funcionar em diferentes sistemas operativos, incluindo macOS, Linux e Windows.

Para programadores que procuram integrar modelos de código aberto nas suas aplicações sem usar uma API remota, o Ollama oferece uma CLI para gerir ciclos de vida de modelos semelhante às ferramentas de gestão de contentores. Também suporta configurações e prompts personalizados, permitindo um alto grau de personalização para adaptar os modelos a necessidades ou casos de uso específicos.

O Ollama é particularmente adequado para utilizadores tecnicamente experientes e programadores devido à sua interface de linha de comandos e à flexibilidade que oferece na gestão e implementação de modelos de IA. Isto torna-o uma ferramenta poderosa para empresas e indivíduos que necessitam de capacidades robustas de IA sem comprometer a segurança e o controlo.

Plataformas Multi-Modelo

Adicionalmente, existem fornecedores que hospedam uma grande variedade de modelos de código aberto, como a Together.ai e a Groq.. Estas plataformas oferecem flexibilidade e personalização, permitindo-lhe executar e, em alguns casos, até fazer o ajuste fino de modelos de código aberto de acordo com as suas necessidades específicas. Por exemplo, a Together.ai fornece acesso a uma gama de LLMs de código aberto, permitindo aos utilizadores experimentar diferentes modelos e configurações. A Groq concentra-se em fornecer conclusões de ultra alto desempenho que, no momento da escrita deste livro, parecem quase mágicas

Escolher um Fornecedor de LLM

Ao escolher um fornecedor de LLM, deve considerar fatores como:

- **Preços:** Diferentes fornecedores oferecem diferentes modelos de preços, desde pagamento por utilização até planos baseados em subscrição. É importante considerar a utilização esperada e o orçamento ao selecionar um fornecedor.

- **Desempenho:** O desempenho dos LLMs pode variar significativamente entre fornecedores, por isso é importante fazer avaliações comparativas e testar modelos em casos de uso específicos antes de tomar uma decisão.
- **Filtragem de Conteúdo:** Dependendo da aplicação, a filtragem de conteúdo pode ser uma consideração crítica. Alguns fornecedores oferecem opções de filtragem de conteúdo mais robustas do que outros.
- **Privacidade de Dados:** Se a aplicação lida com dados sensíveis dos utilizadores, é importante escolher um fornecedor com práticas fortes de privacidade e segurança de dados.
- **Personalização:** Alguns fornecedores oferecem mais flexibilidade em termos de ajuste fino e personalização de modelos para casos de uso específicos.

Em última análise, a escolha do fornecedor de LLM depende dos requisitos e restrições específicos da aplicação. Ao avaliar cuidadosamente as opções e considerar fatores como preços, desempenho e privacidade de dados, pode selecionar o fornecedor que melhor atende às suas necessidades.

Vale também notar que o panorama dos LLM está em constante evolução, com novos fornecedores e modelos surgindo regularmente. Deve manter-se atualizado com os últimos desenvolvimentos e estar aberto a explorar novas opções à medida que estas se tornam disponíveis.

OpenRouter

Ao longo deste livro, irei utilizar exclusivamente o [OpenRouter](#) como meu fornecedor de API preferido. A razão é simples: é uma loja única para todos os modelos comerciais e de código aberto mais populares. Se está ansioso para começar a programar com IA, um dos melhores lugares para começar é com a minha própria [Biblioteca Ruby OpenRouter](#).

Pensando Sobre o Desempenho

Ao incorporar modelos de linguagem em aplicações, o desempenho é uma consideração crítica. O desempenho de um modelo de linguagem pode ser medido em termos da sua *latência* (o tempo que leva para gerar uma resposta) e *taxa de transferência* (o número de pedidos que pode processar por unidade de tempo).

O *Tempo até o Primeiro Token* (TTFT) é outra métrica de desempenho essencial, particularmente relevante para chatbots e aplicações que requerem respostas interativas em tempo real. O TTFT mede a latência desde o momento em que o pedido do utilizador é recebido até ao momento em que a primeira palavra (ou token) da resposta é gerada. Esta métrica é crucial para manter uma experiência de utilizador fluida e envolvente, já que respostas atrasadas podem levar à frustração e ao desinteresse do utilizador.

Estas métricas de desempenho podem ter um impacto significativo na experiência do utilizador e na escalabilidade da aplicação.

Vários fatores podem influenciar o desempenho de um modelo de linguagem, incluindo:

Contagem de Parâmetros: Modelos maiores com mais parâmetros geralmente requerem mais recursos computacionais e podem ter maior latência e menor taxa de transferência em comparação com modelos menores.

Hardware: O desempenho de um modelo de linguagem pode variar significativamente com base no hardware em que está a ser executado. Os fornecedores de cloud oferecem instâncias de GPU e TPU otimizadas para cargas de trabalho de machine learning, que podem acelerar significativamente a inferência do modelo.



Uma das coisas interessantes sobre o OpenRouter é que, para muitos dos modelos que oferece, tem-se uma escolha de fornecedores de cloud com uma variedade de perfis de desempenho e custos.

Quantização: Técnicas de quantização podem ser usadas para reduzir a utilização de

memória e os requisitos computacionais de um modelo, representando pesos e ativações com tipos de dados de menor precisão. Isto pode melhorar o desempenho sem sacrificar significativamente a qualidade. Como desenvolvedor de aplicações, provavelmente não se envolverá no treino dos seus próprios modelos em diferentes níveis de quantização, mas é bom estar pelo menos familiarizado com a terminologia.

Processamento em Lote: Processar múltiplos pedidos simultaneamente em lotes pode melhorar a taxa de transferência ao amortizar a sobrecarga do carregamento do modelo e da transferência de dados.

Cache: Armazenar em cache os resultados de prompts ou sequências de entrada frequentemente utilizados pode reduzir o número de pedidos de inferência e melhorar o desempenho geral.

Ao selecionar um modelo de linguagem para uma aplicação em produção, é importante avaliar o seu desempenho em cargas de trabalho representativas e configurações de hardware. Isto pode ajudar a identificar potenciais gargalos e garantir que o modelo pode atingir as metas de desempenho necessárias.

Também vale a pena considerar os compromissos entre o desempenho do modelo e outros fatores como custo, flexibilidade e facilidade de integração. Por exemplo, usar um modelo menor e menos dispendioso com menor latência pode ser preferível para aplicações que requerem respostas em tempo real, enquanto um modelo maior e mais poderoso pode ser mais adequado para processamento em lote ou tarefas de raciocínio complexo.

Experimentando com Diferentes Modelos LLM

Escolher um LLM raramente é uma decisão permanente. Como novos e melhores modelos são lançados regularmente, é bom construir aplicações de forma modular que permita trocar diferentes modelos de linguagem ao longo do tempo. Prompts e conjuntos de dados podem frequentemente ser reutilizados entre modelos com alterações mínimas.

Isto permite aproveitar os últimos avanços em modelagem de linguagem sem ter que redesenhar completamente as aplicações.



A capacidade de alternar facilmente entre uma ampla gama de escolhas de modelos é mais uma razão pela qual adoro o OpenRouter.

Ao atualizar para um novo modelo de linguagem, é importante testar e validar minuciosamente o seu desempenho e qualidade de saída para garantir que atende aos requisitos da aplicação. Isto pode envolver retrainar ou fazer fine-tuning do modelo em dados específicos do domínio, bem como atualizar quaisquer componentes downstream que dependam das saídas do modelo.

Ao projetar aplicações com desempenho e modularidade em mente, pode criar sistemas escaláveis, eficientes e à prova de futuro que podem adaptar-se ao panorama em rápida evolução da tecnologia de modelagem de linguagem.

Sistemas de IA Compostos

Antes de concluir a nossa introdução, vale a pena mencionar que antes de 2023 e da explosão de interesse em IA generativa desencadeada pelo ChatGPT, as abordagens tradicionais de IA geralmente dependiam da integração de modelos únicos e fechados. Em contraste, *Sistemas de IA Compostos* aproveitam pipelines complexos de componentes interligados trabalhando em conjunto para alcançar comportamento inteligente.

No seu núcleo, os sistemas de IA compostos consistem em múltiplos módulos, cada um projetado para executar tarefas ou funções específicas. Estes módulos podem incluir geradores, recuperadores, classificadores, ranqueadores e vários outros componentes especializados. Ao dividir o sistema geral em unidades menores e focadas, os desenvolvedores podem criar arquiteturas de IA mais flexíveis, escaláveis e manuteníveis.

Uma das principais vantagens dos sistemas de IA compostos é a sua capacidade de combinar os pontos fortes de diferentes técnicas e modelos de IA. Por exemplo, um sistema pode utilizar um modelo de linguagem de grande escala (LLM) para compreensão e geração de linguagem natural, enquanto emprega um modelo separado para recuperação de informação ou tomada de decisões baseada em regras. Esta abordagem modular permite-lhe selecionar as melhores ferramentas e técnicas para cada tarefa específica, em vez de depender de uma solução única para todos os casos.

No entanto, construir sistemas de IA compostos também apresenta desafios únicos. Em particular, garantir a coerência e consistência geral do comportamento do sistema requer mecanismos robustos de teste, monitorização e governança.



O advento de LLMs poderosos como o GPT-4 permite-nos experimentar sistemas de IA compostos mais facilmente do que nunca, porque estes modelos avançados são capazes de lidar com múltiplos papéis dentro de um sistema composto, como classificação, classificação e geração, além das suas capacidades de compreensão de linguagem natural. Esta versatilidade permite aos programadores prototipar e iterar rapidamente arquiteturas de IA compostas, abrindo novas possibilidades para o desenvolvimento de aplicações inteligentes.

Padrões de Implementação para Sistemas de IA Compostos

Os sistemas de IA compostos podem ser implementados usando vários padrões, cada um projetado para atender a requisitos e casos de uso específicos. Vamos explorar quatro padrões comuns de implementação: Pergunta e Resposta, Solucionadores de Problemas Multi-Agente/Agênticos, IA Conversacional e CoPilotos.

Pergunta e Resposta

Os sistemas de Pergunta e Resposta (P&R) concentram-se em fornecer recuperação de informação aprimorada com as capacidades de compreensão dos modelos de IA para funcionar como mais do que simplesmente um motor de busca. Ao combinar modelos de linguagem poderosos com fontes de conhecimento externas usando [Geração Aumentada por Recuperação \(RAG\)](#), os sistemas de Pergunta e Resposta evitam alucinações e fornecem respostas precisas e contextualmente relevantes às consultas dos utilizadores.

Os componentes principais de um sistema de P&R baseado em LLM incluem:

- **Compreensão e reformulação de consultas:** Análise das consultas dos utilizadores e reformulação das mesmas para melhor corresponder às fontes de conhecimento subjacentes.
- **Recuperação de conhecimento:** Recuperação de informações relevantes de fontes de dados estruturados ou não estruturados com base na consulta reformulada.
- **Geração de resposta:** Geração de respostas coerentes e informativas integrando o conhecimento recuperado com as capacidades generativas do modelo de linguagem.

Os subsistemas RAG são particularmente importantes em domínios de P&R onde fornecer informações precisas e atualizadas é crucial, como suporte ao cliente, gestão do conhecimento ou aplicações educacionais

Solucionadores de Problemas Multi-Agente/Agênticos

Os sistemas multi-agente, também conhecidos como *Agênticos*, consistem em múltiplos agentes autónomos trabalhando juntos para resolver problemas complexos. Cada agente tem um papel específico, conjunto de habilidades e acesso a ferramentas ou fontes de informação relevantes. Ao colaborar e trocar informações, estes agentes podem lidar com tarefas que seriam difíceis ou impossíveis para um único agente resolver sozinho.

Os princípios fundamentais dos solucionadores de problemas multi-agente incluem:

- **Especialização:** Cada agente concentra-se num aspeto específico do problema, aproveitando as suas capacidades e conhecimentos únicos.
- **Colaboração:** Os agentes comunicam e coordenam as suas ações para alcançar um objetivo comum, frequentemente através de troca de mensagens ou memória partilhada.
- **Adaptabilidade:** O sistema pode adaptar-se a condições ou requisitos em mudança ajustando os papéis e comportamentos dos agentes individuais.

Os sistemas multi-agente são adequados para aplicações que requerem resolução distribuída de problemas, como otimização da cadeia de abastecimento, gestão de tráfego ou planeamento de resposta a emergências

IA Conversacional

Os sistemas de IA conversacional permitem interações em linguagem natural entre utilizadores e agentes inteligentes. Estes sistemas combinam compreensão de linguagem natural, gestão de diálogo e capacidades de geração de linguagem para fornecer experiências conversacionais envolventes e personalizadas.

Os principais componentes de um sistema de IA conversacional incluem:

- **Reconhecimento de intenção:** Identificação da intenção do utilizador com base na sua entrada, como fazer uma pergunta, fazer um pedido ou expressar um sentimento.
- **Extração de entidades:** Extração de entidades ou parâmetros relevantes da entrada do utilizador, como datas, localizações ou nomes de produtos.
- **Gestão de diálogo:** Manutenção do estado da conversa, determinação da resposta apropriada com base na intenção e contexto do utilizador, e gestão de interações de múltiplos turnos.

- **Geração de resposta:** Geração de respostas semelhantes às humanas usando modelos de linguagem, modelos ou métodos baseados em recuperação.

Os sistemas de IA conversacional são comumente usados em chatbots de atendimento ao cliente, assistentes virtuais e interfaces controladas por voz. Como mencionado anteriormente, a maioria das abordagens, padrões e exemplos de código neste livro são diretamente extraídos do meu trabalho num grande sistema de IA conversacional chamado [Olympia](#)

CoPilots

Os CoPilots são assistentes baseados em IA que trabalham em conjunto com utilizadores humanos para melhorar a sua produtividade e capacidade de tomada de decisões. Estes sistemas aproveitam uma combinação de processamento de linguagem natural, aprendizagem automática e conhecimento específico do domínio para fornecer recomendações inteligentes, automatizar tarefas e oferecer suporte contextual.

As características principais dos CoPilots incluem:

- **Personalização:** Adaptação às preferências individuais dos utilizadores, fluxos de trabalho e estilos de comunicação.
- **Assistência proativa:** Antecipação das necessidades do utilizador e oferecimento de sugestões ou ações relevantes sem solicitações explícitas.
- **Aprendizagem contínua:** Melhoria do desempenho ao longo do tempo através da aprendizagem com base no feedback, interações e dados dos utilizadores.

Os CoPilots são cada vez mais utilizados em vários domínios, como desenvolvimento de software (por exemplo, conclusão de código e deteção de erros), escrita criativa (por exemplo, sugestões de conteúdo e edição), e análise de dados (por exemplo, insights e recomendações de visualização)

Estes padrões de implementação demonstram a versatilidade e o potencial dos sistemas de IA compostos. Ao compreender as características e casos de uso de cada padrão, pode tomar decisões informadas ao projetar e implementar aplicações inteligentes. Embora este livro não seja especificamente sobre a implementação de sistemas de IA compostos, muitas, se não todas, as mesmas abordagens e padrões aplicam-se à integração de componentes discretos de IA no desenvolvimento de aplicações tradicionais.

Funções em Sistemas de IA Compostos

Os sistemas de IA compostos são construídos sobre uma base de módulos interligados, cada um projetado para desempenhar uma função específica. Estes módulos trabalham em conjunto para criar comportamentos inteligentes e resolver problemas complexos. É útil estar familiarizado com estas funções ao pensar onde poderá implementar ou substituir partes da sua aplicação por componentes discretos de IA.

Gerador

Os geradores são responsáveis por produzir novos dados ou conteúdo com base em padrões aprendidos ou prompts de entrada. O mundo da IA tem muitos tipos diferentes de geradores, mas no contexto dos tipos de modelos de linguagem apresentados neste livro, os geradores podem criar texto semelhante ao humano, completar frases parciais ou gerar respostas a consultas de utilizadores. Desempenham um papel crucial em tarefas como criação de conteúdo, geração de diálogo e aumento de dados.

Recuperador

Os recuperadores são utilizados para pesquisar e extrair informações relevantes de grandes conjuntos de dados ou bases de conhecimento. Empregam técnicas como pesquisa semântica, correspondência de palavras-chave ou similaridade vetorial para encontrar os pontos de dados mais pertinentes com base numa determinada consulta ou contexto. Os recuperadores são essenciais para tarefas que exigem acesso

rápido a informações específicas, como resposta a perguntas, verificação de factos ou recomendação de conteúdo.

Classificador por Ordem

Os classificadores por ordem são responsáveis por ordenar ou priorizar um conjunto de itens com base em determinados critérios ou pontuações de relevância. Atribuem pesos ou pontuações a cada item e depois ordenam-nos em conformidade. São comumente utilizados em motores de busca, sistemas de recomendação ou qualquer aplicação onde seja crucial apresentar os resultados mais relevantes aos utilizadores.

Classificador

Os classificadores são utilizados para categorizar ou rotular pontos de dados com base em classes ou categorias predefinidas. Aprendem a partir de dados de treino rotulados e depois preveem a classe de novas instâncias não vistas. Os classificadores são fundamentais para tarefas como análise de sentimento, deteção de spam ou reconhecimento de imagem, onde o objetivo é atribuir uma categoria específica a cada entrada.

Ferramentas e Agentes

Além destas funções principais, os sistemas de IA compostos frequentemente incorporam ferramentas e agentes para melhorar a sua funcionalidade e adaptabilidade:

- **Ferramentas:** As ferramentas são componentes de software discretos ou APIs que executam ações ou cálculos específicos. Podem ser invocadas por outros módulos, como geradores ou recuperadores, para realizar subtarefas ou recolher informações adicionais. Exemplos de ferramentas incluem motores de busca web, calculadoras ou bibliotecas de visualização de dados.

- **Agentes:** Os agentes são entidades autónomas que podem perceber o seu ambiente, tomar decisões e executar ações para atingir objetivos específicos. Frequentemente dependem de uma combinação de diferentes técnicas de IA, como planeamento, raciocínio e aprendizagem, para operar eficazmente em condições dinâmicas ou incertas. Os agentes podem ser utilizados para modelar comportamentos complexos ou para coordenar as ações de múltiplos módulos dentro de um sistema de IA composto.

Num sistema de IA composto puro, a interação entre estes componentes é orquestrada através de interfaces bem definidas e protocolos de comunicação. Os dados fluem entre módulos, com a saída de um componente servindo como entrada para outro. Esta arquitetura modular permite flexibilidade, escalabilidade e manutenibilidade, pois os componentes individuais podem ser atualizados, substituídos ou expandidos sem afetar todo o sistema.

Ao aproveitar o poder destes componentes e suas interações, os sistemas de IA compostos podem abordar problemas complexos do mundo real que requerem uma combinação de diferentes capacidades de IA. À medida que exploramos as abordagens e padrões para integrar IA no desenvolvimento de aplicações, tenha em mente que os mesmos princípios e técnicas utilizados em sistemas de IA compostos podem ser aplicados para criar aplicações inteligentes, adaptativas e centradas no utilizador.

Nos próximos capítulos da Parte 1, mergulharemos mais profundamente nas abordagens e técnicas fundamentais para integrar componentes de IA no seu processo de desenvolvimento de aplicações. Da engenharia de prompts e geração aumentada por recuperação até dados auto-reparáveis e orquestração inteligente de fluxos de trabalho, abordaremos uma ampla gama de padrões e melhores práticas para ajudá-lo a construir aplicações de ponta baseadas em IA.

Parte 1: Abordagens e Técnicas Fundamentais

Esta parte do livro apresenta diferentes formas de integrar a utilização de IA nas suas aplicações. Os capítulos abrangem um conjunto de abordagens e técnicas relacionadas, desde conceitos de alto nível como [Restringir o Caminho](#) e [Geração Aumentada por Recuperação](#) até ideias para programar a sua própria camada de abstração sobre APIs de conclusão de chat LLM.

O objetivo desta parte do livro é ajudá-lo a compreender os tipos de comportamento que pode implementar com IA, antes de aprofundar demasiado os padrões de implementação específicos que são o foco da [Parte 2](#).

As abordagens na Parte 1 baseiam-se em ideias que utilizei no meu código, padrões clássicos de arquitetura e integração de aplicações empresariais, além de metáforas que utilizei ao explicar as capacidades da IA a outras pessoas, incluindo stakeholders empresariais não técnicos.

Estreitar o Caminho



“Estreitar o caminho” refere-se a focar a IA na tarefa em questão. Uso-o como um mantra sempre que fico frustrado com a IA a agir de forma “estúpida” ou inesperada. O mantra lembra-me que a falha é provavelmente minha culpa, e que provavelmente deveria estreitar ainda mais o caminho.

A necessidade de estreitar o caminho surge das vastas quantidades de conhecimento contidas nos modelos de linguagem de grande escala, especialmente modelos de classe mundial como os da OpenAI e Anthropic que têm literalmente bilhões de parâmetros.

Ter acesso a uma gama tão ampla de conhecimento é indubitavelmente poderoso e produz comportamentos emergentes como a teoria da mente e a capacidade de raciocinar de forma semelhante à humana. No entanto, esse volume avassalador de informação também apresenta desafios quando se trata de gerar respostas precisas e exatas para prompts específicos, especialmente se esses prompts se destinam a exibir comportamento determinístico que pode ser integrado com desenvolvimento de software e algoritmos “normais”.

Vários fatores levam a estes desafios.

Sobrecarga de Informação: Os modelos de linguagem de grande escala são treinados com quantidades massivas de dados abrangendo vários domínios, fontes e períodos temporais. Este conhecimento extenso permite-lhes envolver-se em diversos tópicos e gerar respostas baseadas numa ampla compreensão do mundo. No entanto, quando confrontado com um prompt específico, o modelo pode ter dificuldade em filtrar informações irrelevantes, contraditórias ou desatualizadas/obsoletas, levando a respostas que carecem de foco ou precisão. Dependendo do que está a tentar fazer, o simples volume de informação *contraditória* disponível para o modelo pode facilmente sobrecarregar a sua capacidade de fornecer a resposta ou comportamento que procura.

Ambiguidade Contextual: Dado o vasto *espaço latente* de conhecimento, os modelos de linguagem de grande escala podem encontrar ambiguidade ao tentar compreender o *contexto* do seu prompt. Sem um estreitamento ou orientação adequada, o modelo pode gerar respostas que estão tangencialmente relacionadas mas não são diretamente relevantes para as suas intenções. Este tipo de falha leva a respostas que estão fora do tópico, são inconsistentes ou não atendem às suas necessidades declaradas. Neste caso, estreitar o caminho refere-se à *desambiguação* do contexto, garantindo que o contexto fornecido faz com que o modelo se concentre apenas na informação mais relevante do seu conhecimento base.



Nota: Quando está a começar com a “engenharia de prompts”, é muito mais provável que peça ao modelo para fazer coisas sem explicar adequadamente o resultado desejado; é preciso prática para não ser ambíguo!

Inconsistências Temporais: Como os modelos de linguagem são treinados com dados que foram criados em diferentes períodos temporais, podem possuir conhecimentos que estão desatualizados, ultrapassados ou já não são precisos. Por exemplo, informações sobre eventos atuais, descobertas científicas ou avanços tecnológicos podem ter evoluído desde que os dados de treino do modelo foram recolhidos. Sem estreitar o caminho para priorizar fontes mais recentes e fiáveis, o modelo pode gerar respostas baseadas em informações desatualizadas ou incorretas, levando a imprecisões e inconsistências nos seus outputs.

Nuances Específicas do Domínio: Diferentes domínios e campos têm as suas próprias terminologias, convenções e bases de conhecimento específicas. Pense em praticamente qualquer TLA (Acrónimo de Três Letras) e perceberá que a maioria tem mais do que um significado. Por exemplo, MSK pode referir-se ao Managed Streaming for Apache Kafka da Amazon, ao Memorial Sloan Kettering Cancer Center, ou ao sistema MusculoESquelético humano.

Quando um prompt requer experiência num domínio particular, o conhecimento genérico de um modelo de linguagem de grande escala pode não ser suficiente para fornecer respostas precisas e nuancadas. Estreitar o caminho focando em informação específica do domínio, seja através de engenharia de prompts ou geração aumentada por recuperação, permite que o modelo gere respostas mais alinhadas com os requisitos e expectativas do seu domínio específico.

Espaço Latente: Incompreensivelmente Vasto

Quando menciono o “espaço latente” de um modelo de linguagem, estou a referir-me à vasta paisagem multidimensional de conhecimento e informação que o modelo aprendeu

durante o seu processo de treino. É como um reino oculto dentro das redes neurais do modelo, onde todos os padrões, associações e representações da linguagem estão armazenados.

Imagine que está a explorar um vasto território inexplorado cheio de inúmeros nós interligados. Cada nó representa uma peça de informação, um conceito ou uma relação que o modelo aprendeu. À medida que navega por este espaço, descobrirá que alguns nós estão mais próximos uns dos outros, indicando uma forte conexão ou similaridade, enquanto outros estão mais distantes, sugerindo uma relação mais fraca ou mais distante.

O desafio com o espaço latente é que ele é incrivelmente complexo e multidimensional. Imagine-o tão imenso quanto o nosso universo físico, com seus aglomerados de galáxias e vastas, inimagináveis distâncias de espaço vazio entre eles.

Por conter milhares de dimensões, o espaço latente não é diretamente observável ou interpretável por humanos. É uma representação abstrata que o modelo usa internamente para processar e gerar linguagem. Quando você fornece um prompt inicial ao modelo, ele essencialmente mapeia esse prompt para uma localização específica dentro do espaço latente. O modelo então usa a informação circundante e as conexões nesse espaço para gerar uma resposta.

A questão é que o modelo aprendeu uma quantidade enorme de informação a partir dos seus dados de treino, e nem toda ela é relevante ou precisa para uma determinada tarefa. É por isso que estreitar o caminho se torna tão importante. Ao fornecer instruções claras, exemplos e contexto nos seus prompts, você está essencialmente guiando o modelo para se focar em regiões específicas dentro do espaço latente que são mais relevantes para o resultado desejado.

Uma forma diferente de pensar nisso é como usar um holofote num museu completamente escuro. Se já visitou o Louvre ou o Metropolitan Museum of Art, então é dessa escala que estou a falar. O espaço latente é o museu, repleto de incontáveis objetos e detalhes. O seu prompt é o holofote, iluminando áreas específicas

e direcionando a atenção do modelo para a informação mais importante. Sem essa orientação, o modelo pode vagar sem rumo pelo espaço latente, recolhendo informação irrelevante ou contraditória pelo caminho.

À medida que trabalha com modelos de linguagem e elabora os seus prompts, mantenha em mente o conceito de espaço latente. O seu objetivo é navegar eficazmente nesta vasta paisagem de conhecimento, direcionando o modelo para a informação mais relevante e precisa para a sua tarefa. Ao estreitar o caminho e fornecer orientação clara, pode desbloquear todo o potencial do espaço latente do modelo e gerar respostas de alta qualidade e coerentes.

Embora as descrições anteriores dos modelos de linguagem e do espaço latente que eles navegam possam parecer um pouco mágicas ou abstratas, é importante compreender que os prompts não são feitiços ou encantamentos. A forma como os modelos de linguagem funcionam está fundamentada nos princípios da álgebra linear e teoria da probabilidade.

No seu núcleo, os modelos de linguagem são modelos probabilísticos de texto, muito semelhantes a como uma curva em sino é um modelo estatístico de dados. São treinados através de um processo chamado modelagem autorregressiva, onde o modelo aprende a prever a probabilidade da próxima palavra numa sequência com base nas palavras que vêm antes dela. Durante o treino, o modelo começa com pesos aleatórios e gradualmente ajusta-os para atribuir probabilidades mais altas a texto que se assemelha às amostras do mundo real com que foi treinado.

No entanto, pensar em modelos de linguagem como simples modelos estatísticos, como regressão linear, não fornece a melhor intuição para compreender o seu comportamento. Uma analogia mais adequada é pensar neles como programas probabilísticos, que são modelos que permitem a manipulação de variáveis aleatórias e podem representar relações estatísticas complexas.

Os programas probabilísticos podem ser representados por modelos gráficos, que fornecem uma forma visual de compreender as dependências e relações entre variáveis no modelo. Esta perspetiva pode oferecer insights valiosos sobre o funcionamento de

modelos complexos de geração de texto como o GPT-4 e o Claude.

No artigo “Language Model Cascades” de Dohan et al., os autores aprofundam-se nos detalhes de como os programas probabilísticos podem ser aplicados aos modelos de linguagem. Eles mostram como esta estrutura pode ser usada para compreender o comportamento destes modelos e orientar o desenvolvimento de estratégias de prompting mais eficazes.

Um insight fundamental desta perspectiva probabilística é que o modelo de linguagem essencialmente cria um portal para um universo alternativo onde os documentos desejados existem. O modelo atribui pesos a todos os documentos possíveis com base na sua probabilidade, efetivamente reduzindo o espaço de possibilidades para se focar nos mais relevantes.

Isto traz-nos de volta ao tema central de “estreitar o caminho”. O objetivo principal do prompting é condicionar o modelo probabilístico de uma forma que concentre a massa das suas previsões, focando-se na informação específica ou comportamento que queremos obter. Ao fornecer prompts cuidadosamente elaborados, podemos guiar o modelo para navegar no espaço latente de forma mais eficiente e gerar resultados que são mais relevantes e coerentes.

No entanto, é importante ter em mente que o modelo de linguagem está ultimamente limitado pela informação com que foi treinado. Embora possa gerar texto semelhante a documentos existentes ou combinar ideias de formas inovadoras, não pode criar informação inteiramente nova do zero. Por exemplo, não podemos esperar que o modelo forneça uma cura para o cancro se tal cura ainda não foi descoberta e documentada nos seus dados de treino.

Em vez disso, a força do modelo reside na sua capacidade de encontrar e sintetizar informação semelhante àquela que incluímos no prompt. Ao compreender a natureza probabilística destes modelos e como os prompts podem ser utilizados para condicionar os seus resultados, podemos aproveitar mais eficazmente as suas capacidades para gerar insights e conteúdo valioso.

Considere os prompts abaixo. No primeiro, “Mercúrio” sozinho poderia referir-se ao planeta, ao elemento ou ao deus romano, mas o mais provável é o planeta. De facto, o GPT-4 fornece uma resposta longa que começa por *Mercúrio é o planeta mais pequeno e mais próximo do Sol no Sistema Solar....* O segundo prompt refere-se especificamente ao elemento químico. O terceiro refere-se à figura mitológica romana, conhecida pela sua velocidade e papel como mensageiro divino.

```
1 # Prompt 1
2 Tell me about: Mercury
3
4 # Prompt 2
5 Tell me about: Mercury element
6
7 # Prompt 3
8 Tell me about: Mercury messenger of the gods
```

Ao acrescentar apenas algumas palavras extra, mudámos completamente a forma como a IA reage. Como aprenderá mais tarde neste livro, truques sofisticados de engenharia de prompts, como prompts n-shot, entrada/saída estruturada e [Cadeia de Pensamento](#) são apenas formas inteligentes de condicionar a saída do modelo.

Assim, em última análise, a arte da engenharia de prompts consiste em compreender como navegar na vasta paisagem probabilística do conhecimento do modelo de linguagem para estreitar o caminho até à informação ou comportamento específico que procuramos.

Para os leitores com um sólido conhecimento de matemática avançada, fundamentar a vossa compreensão destes modelos nos princípios da teoria das probabilidades e da álgebra linear pode definitivamente ajudar! Para os restantes que querem desenvolver estratégias eficazes para obter as saídas desejadas, vamos manter-nos em abordagens mais intuitivas.

Como o Caminho é “Estreitado”

Para abordar estes desafios de excesso de conhecimento, empregamos técnicas que ajudam a guiar o processo de geração do modelo de linguagem e a focar a sua atenção na informação mais relevante e precisa.

Aqui estão as técnicas mais significativas, por ordem recomendada, ou seja, deve tentar primeiro a Engenharia de Prompts, depois RAG e, finalmente, se necessário, o ajuste fino.

Engenharia de Prompts A abordagem mais fundamental é a elaboração de prompts que incluem instruções específicas, restrições ou exemplos para orientar a geração de respostas do modelo. Este capítulo aborda os fundamentos da Engenharia de Prompts na [próxima secção](#), e abordamos muitos padrões específicos de engenharia de prompts na Parte 2 do livro. Esses padrões incluem [Destilação de Prompts](#), uma técnica que se concentra em refinar e otimizar prompts para extrair o que a IA considera ser a informação mais relevante e concisa.

Aumento de Contexto. Recuperação dinâmica de informações relevantes de bases de conhecimento ou documentos externos para fornecer ao modelo um contexto focado no momento em que é solicitado. As técnicas populares de aumento de contexto incluem [Geração Aumentada por Recuperação \(RAG\)](#) Os chamados “modelos online” como os fornecidos pela [Perplexity](#) são capazes de aumentar o seu contexto com resultados de pesquisa em tempo real na internet.



Apesar do seu poder, os LLMs não são treinados nos seus conjuntos de dados únicos, que podem ser privados ou específicos para o problema que está a tentar resolver. As técnicas de Aumento de Contexto permitem dar aos LLMs acesso a dados por trás de APIs, em bases de dados SQL, ou presos em PDFs e apresentações.

Ajuste Fino ou Adaptação de Domínio Treinar o modelo em conjuntos de dados

específicos do domínio para especializar o seu conhecimento e capacidades de geração para uma tarefa ou campo particular.

Baixar a Temperatura

A temperatura é um *hiperparâmetro* usado em modelos de linguagem baseados em transformers que controla a aleatoriedade e criatividade do texto gerado. É um valor entre 0 e 1, onde valores mais baixos tornam a saída mais focada e determinística, enquanto valores mais altos a tornam mais diversa e imprevisível.

Quando a temperatura é definida como 1, o modelo de linguagem gera texto baseado na distribuição completa de probabilidade do próximo token, permitindo respostas mais criativas e variadas. No entanto, isto também pode levar o modelo a gerar texto menos relevante ou coerente.

Por outro lado, quando a temperatura é definida como 0, o modelo de linguagem seleciona sempre o token com a maior probabilidade, efetivamente “estreitando o seu caminho”. Quase todos os meus componentes de IA usam uma temperatura definida em 0 ou próxima disso, pois resulta em respostas mais focadas e previsíveis. É absolutamente útil quando quer que o modelo *siga instruções*, preste atenção às funções que lhe foram fornecidas, ou simplesmente precise de respostas mais precisas e relevantes do que as que está a obter.

Por exemplo, se estiver a construir um chatbot que precisa de fornecer informações factuais, pode querer definir a temperatura para um valor mais baixo para garantir que as respostas são mais precisas e focadas no tema. Por outro lado, se estiver a construir um assistente de escrita criativa, pode querer definir a temperatura para um valor mais alto para encorajar saídas mais diversas e imaginativas.

Hiperparâmetros: Botões e Controlos da Inferência

Quando trabalha com modelos de linguagem, encontrará frequentemente o termo “hiperparâmetros”. No contexto da inferência (ou seja, quando está a usar o modelo para

gerar respostas), os hiperparâmetros são como os botões e controles que pode ajustar para controlar o comportamento e a saída do modelo.

Pense nisso como ajustar as configurações de uma máquina complexa. Tal como pode girar um botão para controlar a temperatura ou acionar um interruptor para mudar o modo de operação, os hiperparâmetros permitem-lhe ajustar com precisão a forma como o modelo de linguagem processa e gera texto.

Alguns hiperparâmetros comuns que encontrará durante a inferência incluem:

- **Temperatura:** Como acabado de mencionar, este parâmetro controla a aleatoriedade e criatividade do texto gerado. Uma temperatura mais alta leva a resultados mais diversos e imprevisíveis, enquanto uma temperatura mais baixa resulta em respostas mais focadas e determinísticas.
- **Amostragem Top-p (nucleus):** Este parâmetro controla a seleção do menor conjunto de tokens cuja probabilidade cumulativa excede um determinado limite (p). Permite resultados mais diversos mantendo ainda a coerência.
- **Amostragem Top-k:** Esta técnica seleciona os k tokens mais prováveis seguintes e redistribui a massa de probabilidade entre eles. Pode ajudar a impedir que o modelo gere tokens de baixa probabilidade ou irrelevantes.
- **Penalizações de Frequência e Presença:** Estes parâmetros penalizam o modelo por repetir as mesmas palavras ou frases com demasiada frequência (penalização de frequência) ou por gerar palavras que não estão presentes no prompt inicial (penalização de presença). Ao ajustar estes valores, pode encorajar o modelo a produzir resultados mais variados e relevantes.
- **Comprimento máximo:** Este hiperparâmetro estabelece um limite superior no número de tokens (palavras ou subpalavras) que o modelo pode gerar numa única resposta. Ajuda a controlar a verbosidade e concisão do texto gerado.

À medida que experimenta diferentes configurações de hiperparâmetros, descobrirá que mesmo pequenos ajustes podem ter um impacto significativo no resultado do modelo. É como afinar uma receita – uma pitada a mais de sal ou um tempo de cozedura ligeiramente mais longo podem fazer toda a diferença no prato final.

A chave é compreender como cada hiperparâmetro afeta o comportamento do modelo e encontrar o equilíbrio certo para a sua tarefa específica. Não tenha medo de experimentar diferentes configurações e ver como influenciam o texto gerado. Com o tempo, desenvolverá uma intuição sobre quais hiperparâmetros ajustar e como alcançar os resultados desejados.

Ao combinar o uso destes parâmetros com engenharia de prompts, geração aumentada por recuperação e ajuste fino, pode efetivamente estreitar o caminho e guiar o modelo de linguagem para gerar respostas mais precisas, relevantes e valiosas para o seu caso de uso específico.

Modelos Base Versus Modelos Ajustados por Instruções

Os modelos base são as versões não refinadas e não treinadas dos LLMs. Imagine-os como uma tela em branco, ainda não influenciada por treino específico para compreender ou seguir instruções. São construídos sobre os vastos dados com que foram inicialmente treinados, capazes de gerar uma ampla gama de resultados. No entanto, sem camadas adicionais de ajuste fino baseado em instruções, as suas respostas podem ser imprevisíveis e requerem prompts mais nuançados e cuidadosamente elaborados para os guiar em direção ao resultado desejado. Trabalhar com modelos base é semelhante a extrair comunicação de um idiota-sábio que tem uma vasta quantidade de conhecimento mas carece completamente de intuição sobre o que está a ser pedido, a menos que seja extremamente preciso nas suas instruções. Frequentemente parecem um papagaio, na medida em que, quando conseguem dizer algo inteligível, é mais

frequentemente apenas a repetição de algo que ouviram dizer.

Por outro lado, os modelos ajustados por instruções passaram por rondas de treino especificamente concebidas para compreender e seguir instruções. GPT-4, Claude 3 e muitos outros dos modelos LLM mais populares são todos fortemente ajustados por instruções. Este treino envolve alimentar o modelo com exemplos de instruções juntamente com os resultados desejados, efetivamente ensinando o modelo como interpretar e executar uma ampla gama de comandos. Como resultado, os modelos instruídos podem compreender mais prontamente a intenção por trás de um prompt e gerar respostas que se alinham estreitamente com as expectativas do utilizador. Isto torna-os mais amigáveis e fáceis de trabalhar, especialmente para aqueles que podem não ter o tempo ou experiência para se envolver em engenharia de prompts extensiva.

Modelos Base: A Tela Não Filtrada

Os modelos base, como o Llama 2-70B ou Yi-34B, oferecem um acesso mais não filtrado às capacidades do modelo do que aquilo a que pode estar habituado se tem experimentado LLMs populares como o GPT-4. Estes modelos não são pré-ajustados para seguir instruções específicas, proporcionando-lhe uma tela em branco para manipular diretamente o resultado do modelo através de engenharia de prompts cuidadosa. Esta abordagem requer uma compreensão profunda de como criar prompts que guiem a IA na direção desejada sem a instruir explicitamente. É semelhante a ter acesso direto às camadas “base” da IA subjacente, sem quaisquer camadas intermediárias a interpretar ou guiar as respostas do modelo (daí o nome).

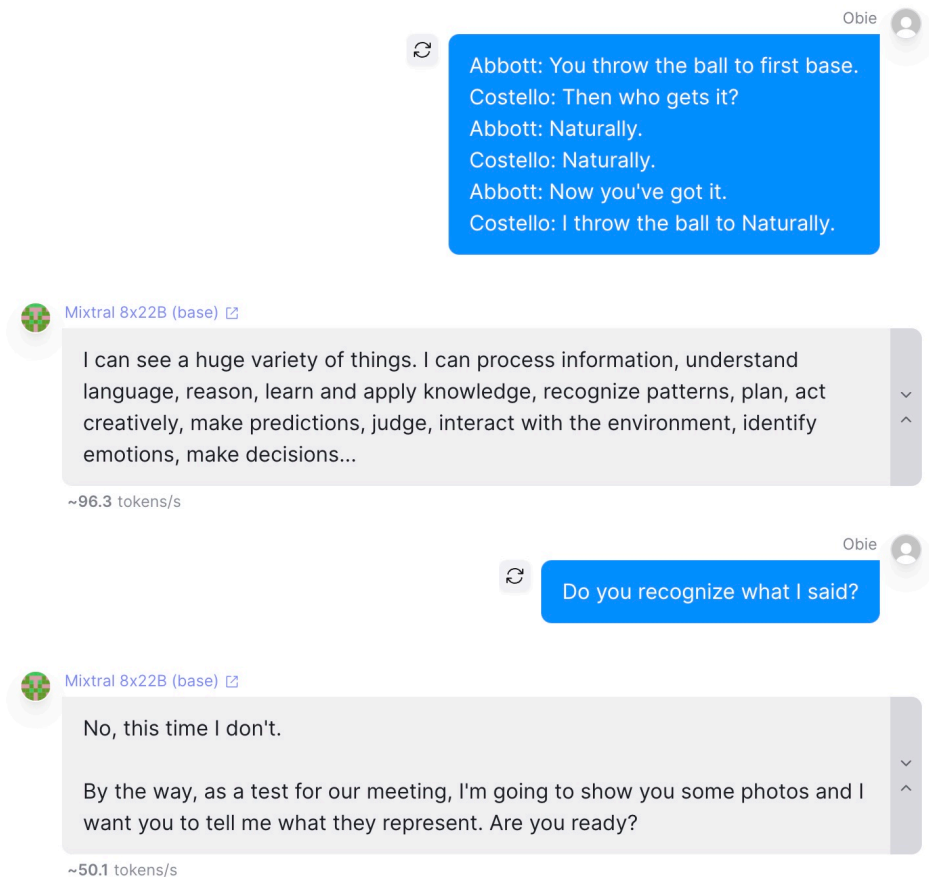


Figura 3. Testando um modelo base usando parte do sketch clássico 'Who's on First' de Abbott e Costello

O desafio com os modelos base está na sua tendência para cair em padrões repetitivos ou produzir resultados aleatórios. No entanto, com uma engenharia de prompts meticulosa e o ajuste de parâmetros como penalizações de repetição, os modelos base podem ser induzidos a gerar conteúdo único e criativo. Este processo não está isento de contrapartidas; embora os modelos base ofereçam uma flexibilidade sem igual para inovação, exigem um nível mais elevado de especialização.

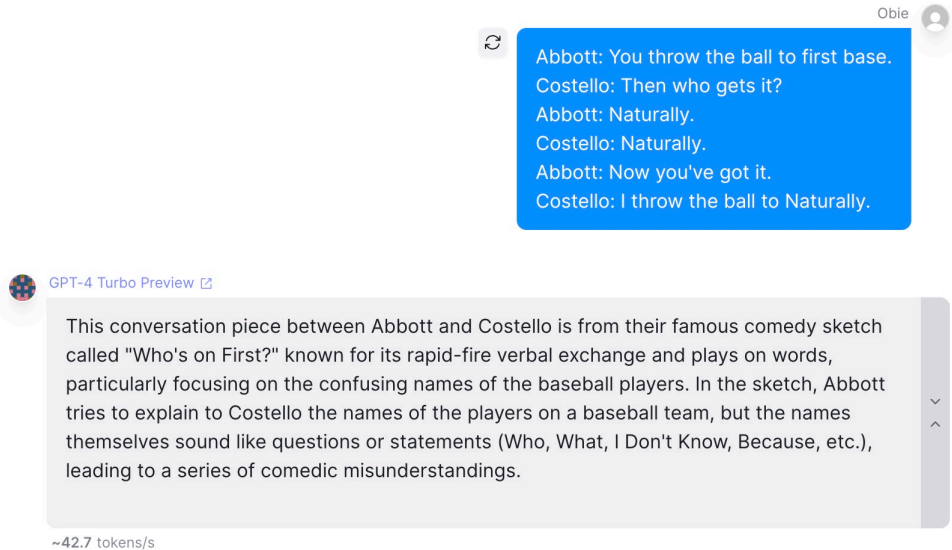


Figura 4. Para fins de comparação, aqui está o mesmo prompt ambíguo fornecido ao GPT-4

Modelos Ajustados por Instruções: A Experiência Guiada

Os modelos ajustados por instruções são projetados para compreender e seguir instruções específicas, tornando-os mais fáceis de usar e acessíveis para uma gama mais ampla de aplicações. Eles compreendem a mecânica de uma *conversa* e que devem parar de gerar quando é o *fim da sua vez de falar*. Para muitos programadores, especialmente aqueles que trabalham em aplicações diretas, os modelos ajustados por instruções oferecem uma solução conveniente e eficiente.

O processo de ajuste por instruções envolve treinar o modelo num grande corpus de prompts de instrução e respostas gerados por humanos. Um exemplo notável é o conjunto de dados open source [databricks-dolly-15k](#), que contém mais de 15.000 pares de prompts/respostas criados por funcionários da Databricks que pode inspecionar por si mesmo. O conjunto de dados abrange oito categorias diferentes de instrução, incluindo escrita criativa, resposta a perguntas fechadas e abertas, sumarização, extração

de informação, classificação e brainstorming.

Durante o processo de geração de dados, os contribuidores receberam orientações sobre como criar prompts e respostas para cada categoria. Por exemplo, para tarefas de escrita criativa, foram instruídos a fornecer restrições específicas, instruções ou requisitos para orientar a saída do modelo. Para resposta a perguntas fechadas, foi-lhes pedido que escrevessem perguntas que exigissem respostas factualmente corretas baseadas numa passagem da Wikipédia fornecida.

O conjunto de dados resultante serve como um recurso valioso para o fine-tuning de grandes modelos de linguagem para exibir as capacidades interativas e de seguimento de instruções de sistemas como o ChatGPT. Ao treinar numa gama diversificada de instruções e respostas geradas por humanos, o modelo aprende a compreender e seguir diretrizes específicas, tornando-o mais apto a lidar com uma grande variedade de tarefas.

Além do fine-tuning direto, os prompts de instrução em conjuntos de dados como o databricks-dolly-15k também podem ser usados para geração de dados sintéticos. Ao submeter prompts gerados por contribuidores como exemplos few-shot a um grande modelo de linguagem aberto, os programadores podem gerar um corpus muito maior de instruções em cada categoria. Esta abordagem, descrita no artigo Self-Instruct, permite a criação de modelos mais robustos que seguem instruções.

Além disso, as instruções e respostas nestes conjuntos de dados podem ser aumentadas através de técnicas como a paráfrase. Ao reformular cada prompt ou resposta curta e associar o texto resultante à respetiva amostra de ground-truth, os programadores podem introduzir uma forma de regularização que melhora a capacidade do modelo de seguir instruções.

A facilidade de utilização proporcionada pelos modelos afinados por instruções tem como contrapartida alguma perda de flexibilidade. Estes modelos são frequentemente muito censurados, o que significa que nem sempre podem proporcionar o nível de liberdade criativa necessário para determinadas tarefas. As suas respostas são fortemente influenciadas pelos vieses e limitações inerentes aos dados do seu

fine-tuning.

Apesar destas limitações, os modelos afinados por instruções tornaram-se cada vez mais populares devido à sua natureza intuitiva e capacidade de lidar com uma ampla gama de tarefas com um mínimo de prompt engineering. À medida que mais conjuntos de dados de instruções de alta qualidade se tornam disponíveis, podemos esperar ver mais melhorias no desempenho e versatilidade destes modelos.

Escolher o Tipo Certo de Modelo para o Seu Projeto

A decisão entre modelos base (puros) e modelos afinados por instruções depende, em última análise, dos requisitos específicos do seu projeto. Para tarefas que exigem um elevado grau de criatividade e originalidade, os modelos base oferecem uma ferramenta poderosa para a inovação. Estes modelos permitem aos programadores explorar todo o potencial dos LLMs, expandindo os limites do que pode ser alcançado através de aplicações baseadas em IA, mas requerem uma abordagem mais prática e uma disposição para experimentar. A Temperatura e outras configurações têm um efeito muito maior nos modelos base do que nos seus equivalentes afinados por instruções.



Tudo o que incluir no seu prompt é o que os modelos base tentarão repetir. Então, se por exemplo o seu prompt for uma transcrição de chat, o modelo puro tentará continuar o chat. Dependendo do limite de `max_tokens`, ele não irá gerar apenas a mensagem seguinte no chat, poderá ter uma conversa inteira consigo próprio!

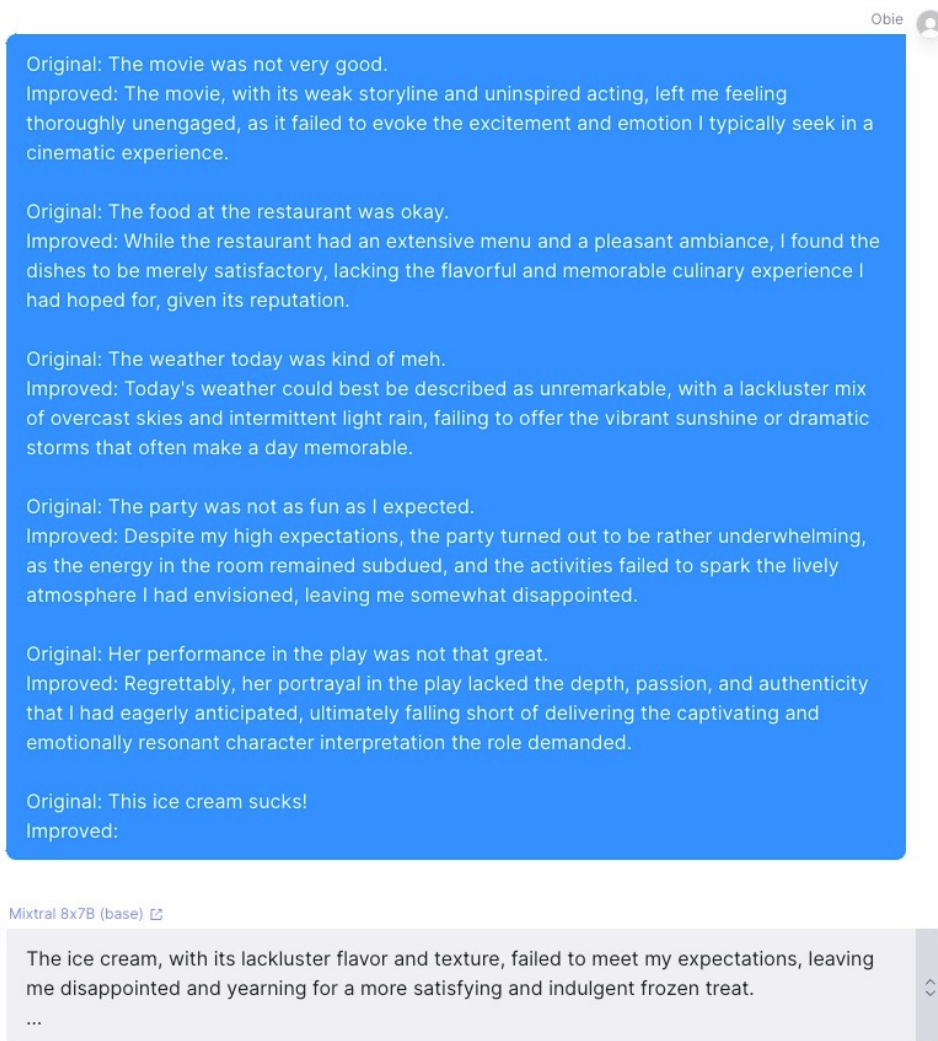


Figura 5. Exemplo de Reescrita de Frases com Few-Shot Completion usando Mixtral 8x7B (base)

Durante a preparação do exemplo de Reescrita de Frases acima pelo utilizador do Reddit [phree_radical](#), só consegui obter resultados utilizáveis após muita experimentação com as configurações dos parâmetros, acabando por estabelecer: Temperatura 0.08, Top P: 0.2, Top K: 1, e Penalidade de Repetição: 1.26.

Tentar usar esta abordagem com um modelo base em produção seria complicado devido ao poderoso efeito do parâmetro `max_tokens`. Se for definido muito curto, a saída é truncada. Se for definido mais longo do que o modelo necessita para a saída desejada, ele continuará a alucinar exemplos adicionais.

A conclusão é que, a menos que realmente precise de controlo total e ausência de censura, os modelos afinados por instruções podem simplificar significativamente o seu processo de desenvolvimento. Para enfatizar esse ponto, aqui está a resposta do Mixtral 8x7B ao mesmo prompt, mas desta vez na sua versão afinada por instruções:

Lamento informar que o gelado não corresponde às minhas expectativas, pois não tem a textura rica e cremosa nem o sabor delicioso que normalmente associo a uma sobremesa de alta qualidade. Esperava uma experiência mais satisfatória e agradável.

É de salientar que consegui manter a configuração de tokens máximos em 500, e o modelo parou de forma fiável no final da saída desejada sem alucinar exemplos adicionais.

Engenharia de Prompts

À medida que começa a aplicar IA nos seus projetos, rapidamente descobrirá que uma das competências mais cruciais que precisa de dominar é a arte da engenharia de prompts. Mas o que é exatamente a engenharia de prompts e por que é tão importante?

Na sua essência, a engenharia de prompts é o processo de desenhar e construir os prompts de entrada que fornece a um modelo de linguagem para orientar a sua saída. Trata-se de compreender como comunicar eficazmente com a IA, usando uma combinação de instruções, exemplos e contexto para direcionar o modelo para gerar a resposta desejada.

Pense nisso como ter uma conversa com um amigo altamente inteligente mas algo literal. Para tirar o máximo partido da interação, precisa de ser claro, específico e fornecer contexto suficiente para garantir que o seu amigo compreende exatamente o que está a pedir. É aqui que entra a engenharia de prompts, e mesmo que pareça fácil no início, acredite em mim que é preciso muita prática para dominar.

Os Elementos Fundamentais de Prompts Eficazes

Para começar a desenvolver prompts eficazes, primeiro precisa de compreender os componentes-chave que constituem uma entrada bem construída. Aqui estão alguns dos elementos fundamentais essenciais:

1. **Instruções:** Instruções claras e concisas que dizem ao modelo o que pretende que ele faça. Isto pode ser qualquer coisa, desde “Resuma o seguinte artigo” a “Gere um poema sobre um pôr do sol” até “transforme este pedido de alteração de projeto num objeto JSON”.
2. **Contexto:** Informação relevante que ajuda o modelo a compreender o enquadramento e o âmbito da tarefa. Isto pode incluir detalhes sobre o público-alvo pretendido, o tom e estilo desejados, ou quaisquer restrições ou requisitos específicos para a saída, como um esquema JSON a seguir.
3. **Exemplos:** Exemplos concretos que demonstram o tipo de saída que procura. Ao fornecer alguns exemplos bem escolhidos, pode ajudar o modelo a aprender os padrões e características da resposta desejada.
4. **Formatação de Entrada:** Quebras de linha e formatação markdown dão estrutura ao nosso prompt. Separar o prompt em parágrafos permite-nos agrupar instruções relacionadas, para que seja mais fácil tanto para humanos como para a IA compreenderem. Marcadores e listas numeradas permitem-nos definir listas e ordenação de itens. Marcadores de negrito e itálico permitem-nos indicar ênfase.
5. **Formatação de Saída:** Instruções específicas sobre como a saída deve ser estruturada e formatada. Estas podem incluir diretrizes sobre o comprimento

desejado, o uso de títulos ou marcadores, formatação markdown, ou quaisquer outros modelos ou convenções de saída específicos que devam ser seguidos.

Ao combinar estes elementos fundamentais de diferentes formas, pode criar prompts que são adaptados às suas necessidades específicas e orientar o modelo para gerar respostas relevantes e de alta qualidade.

A Arte e Ciência do Design de Prompts

Criar prompts eficazes é tanto uma arte como uma ciência. (É por isso que lhe chamamos um ofício.) Requer uma compreensão profunda das capacidades e limitações dos modelos de linguagem, bem como uma abordagem criativa ao desenho de prompts que suscitem o comportamento desejado. A criatividade envolvida é o que torna isto tão divertido, pelo menos para mim. Também pode tornar-se muito frustrante, especialmente quando se procura um comportamento determinístico

Um aspeto fundamental da engenharia de prompts é compreender como equilibrar especificidade e flexibilidade. Por um lado, quer fornecer orientação suficiente para direcionar o modelo no caminho certo. Por outro lado, não quer ser tão prescritivo que limite a capacidade do modelo de utilizar a sua própria criatividade e flexibilidade para lidar com casos extremos.

Outra consideração importante é o uso de exemplos. Exemplos bem escolhidos podem ser incrivelmente poderosos para ajudar o modelo a compreender o tipo de saída que procura. No entanto, é importante usar os exemplos criteriosamente e garantir que são representativos da resposta desejada. Um mau exemplo é, na melhor das hipóteses, um desperdício de tokens e, na pior, prejudicial para a saída desejada.

Técnicas e Boas Práticas de Engenharia de Prompts

À medida que mergulha mais fundo no mundo da engenharia de prompts, descobrirá uma série de técnicas e boas práticas que podem ajudá-lo a criar prompts mais eficazes. Aqui estão algumas áreas-chave para explorar:

1. **Aprendizagem zero-shot vs. few-shot:** Compreender quando usar a aprendizagem *zero-shot* (não fornecer exemplos) versus a aprendizagem *one-shot* ou *few-shot* (fornecer um pequeno número de exemplos) pode ajudá-lo a criar prompts mais eficientes e eficazes.
2. **Refinamento iterativo:** O processo de refinar iterativamente prompts com base na saída do modelo pode ajudar-te a identificar o design ideal do prompt. [Feedback Loop](#) é uma abordagem poderosa que aproveita a própria saída do modelo de linguagem para melhorar progressivamente a qualidade e relevância do conteúdo gerado.
3. **Encadeamento de prompts:** Combinar múltiplos prompts numa sequência pode ajudar-te a decompor tarefas complexas em passos mais pequenos e gerenciáveis. [Prompt Chaining](#) envolve decompor uma tarefa ou conversa complexa numa série de prompts mais pequenos e interligados. Ao encadear prompts, podes guiar a IA através de um processo com múltiplos passos, mantendo o contexto e a coerência ao longo da interação.
4. **Ajuste de prompts:** Personalizar prompts para domínios ou tarefas específicas pode ajudar-te a criar prompts mais especializados e eficazes. [Prompt Template](#) ajuda-te a criar estruturas de prompt flexíveis, reutilizáveis e sustentáveis que são mais facilmente adaptáveis à tarefa em questão.

Aprender quando usar aprendizagem zero-shot, one-shot ou few-shot é uma parte especialmente importante para dominar a engenharia de prompts. Cada abordagem tem os seus pontos fortes e fracos, e compreender quando usar cada uma pode ajudar-te a criar prompts mais eficazes e eficientes.

Aprendizagem Zero-Shot: Quando Não São Necessários Exemplos

A aprendizagem zero-shot refere-se à capacidade de um modelo de linguagem realizar uma tarefa sem quaisquer exemplos ou treino explícito. Por outras palavras, forneces ao

modelo um prompt que descreve a tarefa, e o modelo gera uma resposta baseada apenas no seu conhecimento pré-existente e compreensão da linguagem.

A aprendizagem zero-shot é particularmente útil quando:

1. A tarefa é relativamente simples e direta, e é provável que o modelo tenha encontrado tarefas semelhantes durante o seu pré-treino.
2. Queres testar as capacidades inerentes do modelo e ver como ele responde a uma nova tarefa sem orientação adicional.
3. Estás a trabalhar com um modelo de linguagem grande e diversificado que foi treinado numa ampla gama de tarefas e domínios.

No entanto, a aprendizagem zero-shot também pode ser imprevisível e nem sempre produzir os resultados desejados. A resposta do modelo pode ser influenciada por enviesamentos ou inconsistências nos seus dados de pré-treino, e pode ter dificuldades com tarefas mais complexas ou nuançadas.

Já vi prompts zero-shot que funcionam bem para 80% dos meus casos de teste e produzem resultados completamente errados ou incompreensíveis para os outros 20%. É muito importante implementar um regime de testes rigoroso, especialmente se estiveres a depender muito de prompts zero-shot.

Aprendizagem One-Shot: Quando um Único Exemplo Pode Fazer a Diferença

A aprendizagem one-shot envolve fornecer ao modelo um único exemplo da saída desejada juntamente com a descrição da tarefa. Este exemplo serve como um modelo ou padrão que o modelo pode usar para gerar a sua própria resposta.

A aprendizagem one-shot pode ser eficaz quando:

1. A tarefa é relativamente nova ou específica, e o modelo pode não ter encontrado muitos exemplos semelhantes durante o seu pré-treino.
2. Queres fornecer uma demonstração clara e concisa do formato ou estilo de saída desejado.
3. A tarefa requer uma estrutura ou convenção específica que pode não ser óbvia apenas a partir da descrição da tarefa.



Descrições que são óbvias para ti podem não ser necessariamente óbvias para a IA. Exemplos one-shot podem ajudar a esclarecer as coisas.

A aprendizagem one-shot pode ajudar o modelo a compreender as expectativas mais claramente e gerar uma resposta que está mais alinhada com o exemplo fornecido. No entanto, é importante escolher o exemplo cuidadosamente e garantir que é representativo da saída desejada. Ao escolher o exemplo, pergunta a ti mesmo sobre possíveis casos extremos e a gama de entradas que o prompt irá gerir.

Figura 6. Um exemplo one-shot do JSON desejado

```
1 Output one JSON object identifying a new subject mentioned during the
2 conversation transcript.
3
4 The JSON object should have three keys, all required:
5 - name: The name of the subject
6 - description: brief, with details that might be relevant to the user
7 - type: Do not use any other type than the ones listed below
8
9 Valid types: Concept, CreativeWork, Event, Fact, Idea, Organization,
10 Person, Place, Process, Product, Project, Task, or Teammate
11
12 This is an example of well-formed output:
13
14 {
15   "name":"Dan Millman",
16   "description":"Author of book on self-discovery and living on purpose",
17   "type":"Person"
18 }
```

Aprendizagem Few-Shot: Quando Múltiplos Exemplos Podem Melhorar o Desempenho

A aprendizagem few-shot envolve fornecer ao modelo um pequeno número de exemplos (tipicamente entre 2 e 10) juntamente com a descrição da tarefa. Estes exemplos servem para proporcionar ao modelo mais contexto e variação, ajudando-o a gerar respostas mais diversificadas e precisas.

A aprendizagem few-shot é particularmente útil quando:

1. A tarefa é complexa ou tem nuances, e um único exemplo pode não ser suficiente para capturar todos os aspetos relevantes.
2. Pretende fornecer ao modelo uma variedade de exemplos que demonstrem diferentes variações ou casos extremos.
3. A tarefa requer que o modelo gere respostas consistentes com um domínio ou estilo específico.

Ao fornecer múltiplos exemplos, pode ajudar o modelo a desenvolver uma compreensão mais robusta da tarefa e gerar respostas mais consistentes e fiáveis.

Exemplo: Os Prompts Podem Ser Muito Mais Complexos Do Que Imagina

Os LLMs atuais são muito mais poderosos e capazes de raciocinar do que se possa imaginar. Por isso, não se limite a pensar nos prompts como sendo simplesmente uma especificação de pares de entrada e saída. Pode experimentar dar instruções longas e complexas de formas que se assemelham à forma como interagiria com um humano.

Por exemplo, este é um prompt que usei na Olympia quando estava a criar protótipos da nossa integração com os serviços Google, que na sua totalidade é provavelmente uma das maiores APIs do mundo. As minhas experiências anteriores provaram que o GPT-4

tem um conhecimento razoável da API do Google, e eu não tinha tempo nem motivação para escrever uma camada de mapeamento detalhado, implementando cada função que eu queria dar à minha IA numa base individual. E se eu pudesse simplesmente dar à IA acesso a *toda* a API do Google?

Comecei o meu prompt dizendo à IA que tinha acesso direto aos endpoints da API do Google via HTTP, e que o seu papel era usar as aplicações e serviços Google em nome do utilizador. Depois forneci diretrizes, regras relacionadas com o parâmetro `fields`, já que parecia ter mais dificuldade com esse, e algumas dicas específicas da API (prompting few-shot em ação).

Aqui está o prompt completo, que explica à IA como usar a função `invoke_google_api` fornecida.

```
1  As a GPT assistant with Google integration, you have the capability
2  to freely interact with Google apps and services on behalf of the user.
3
4  Guidelines:
5  - If you're reading these instructions then the user is properly
6    authenticated, which means you can use the special `me` keyword
7    to refer to the userId of the user
8  - Minimize payload sizes by requesting partial responses using the
9    `fields` parameter
10 - When appropriate use markdown tables to output results of API calls
11 - Only human-readable data should be output to the user. For instance,
12   when hitting Gmail's user.messages.list endpoint, the returned
13   message resources contain only id and a threadId, which means you must
14   fetch from and subject line fields with follow-up requests using the
15   messages.get method.
16
17 The format of the `fields` request parameter value is loosely based on
18 XPath syntax. The following rules define formatting for the fields
19 parameter.
20
21 All of these rules use examples related to the files.get method.
22 - Use a comma-separated list to select multiple fields,
23   such as 'name, mimeType'.
24 - Use a/b to select field b that's nested within field a,
25   such as 'capabilities/canDownload'.
```

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

34

35 API-specific hints:

- 36 - Searching contacts: GET [https://people.googleapis.com/v1/](https://people.googleapis.com/v1/people:searchContacts?query=John%20Doe&readMask=names,emailAddresses)
- 37 [people:searchContacts?query=John%20Doe&readMask=names,emailAddresses](https://people.googleapis.com/v1/people:searchContacts?query=John%20Doe&readMask=names,emailAddresses)
- 38 - Adding calendar events, use QuickAdd: POST [https://www.googleapis.com/](https://www.googleapis.com/calendar/v3/calendars/primary/events/quickAdd?text=Appointment%20on%20June%203rd%20at%2010am&sendNotifications=true)
- 39 [calendar/v3/calendars/primary/events/quickAdd?](https://www.googleapis.com/calendar/v3/calendars/primary/events/quickAdd?text=Appointment%20on%20June%203rd%20at%2010am&sendNotifications=true)
- 40 [text=Appointment%20on%20June%203rd%20at%2010am](https://www.googleapis.com/calendar/v3/calendars/primary/events/quickAdd?text=Appointment%20on%20June%203rd%20at%2010am&sendNotifications=true)
- 41 [&sendNotifications=true](https://www.googleapis.com/calendar/v3/calendars/primary/events/quickAdd?text=Appointment%20on%20June%203rd%20at%2010am&sendNotifications=true)

42

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

45

```
46 def invoke_google_api(conversation, arguments)
47     method = arguments[:method] || :get
48     body = arguments[:body]
49     GoogleAPI.send_request(arguments[:endpoint], method:, body:).to_json
50 end
```

51

```
52 # Generic Google API client for accessing any Google service
53 class GoogleAPI
```

```
54     def send_request(endpoint, method:, body: nil)
55         response = @connection.send(method) do |req|
56             req.url endpoint
57             req.body = body.to_json if body
58         end
```

59

```
60         handle_response(response)
61     end
```

62

```
63     # ...rest of class
64 end
```

Pode estar a questionar-se se este prompt funciona. A resposta simples é que sim. A IA

nem sempre sabia como chamar a API perfeitamente à primeira tentativa. No entanto, se cometesse um erro, eu simplesmente alimentava as mensagens de erro resultantes como resultado da chamada. Com o conhecimento do seu erro, a IA podia analisar o seu erro e tentar novamente. Na maioria das vezes, conseguia acertar após algumas tentativas.

Note-se que as grandes estruturas JSON que a API do Google retorna como payloads ao usar este prompt são enormemente ineficientes, por isso *não* estou a recomendar que use esta abordagem em produção. No entanto, penso que o facto de esta abordagem funcionar é um testemunho do quão poderosa a engenharia de prompts pode ser.

Experimentação e Iteração

Em última análise, a forma como desenvolve o seu prompt depende da tarefa específica, da complexidade do resultado desejado e das capacidades do modelo de linguagem com que está a trabalhar.

Como engenheiro de prompts, é importante experimentar diferentes abordagens e iterar com base nos resultados. Comece com aprendizagem zero-shot e veja como o modelo se comporta. Se o resultado for inconsistente ou insatisfatório, tente fornecer um ou mais exemplos e veja se o desempenho melhora.

Tenha em mente que, mesmo dentro de cada abordagem, há espaço para variação e otimização. Pode experimentar diferentes exemplos, ajustar a formulação da descrição da tarefa ou fornecer contexto adicional para ajudar a orientar a resposta do modelo.

Com o tempo, desenvolverá uma intuição sobre qual abordagem tem mais probabilidade de funcionar melhor para uma determinada tarefa, e será capaz de criar prompts mais eficazes e eficientes. A chave é manter-se curioso, experimental e iterativo na sua abordagem à engenharia de prompts.

Ao longo deste livro, mergulharemos mais profundamente nestas técnicas e exploraremos como podem ser aplicadas em cenários do mundo real. Ao dominar a

arte e a ciência da engenharia de prompts, estará bem equipado para desbloquear todo o potencial do desenvolvimento de aplicações baseadas em IA.

A Arte da Ambiguidade

Quando se trata de criar prompts eficazes para grandes modelos de linguagem (MLGs), uma suposição comum é que mais especificidade e instruções detalhadas levam a melhores resultados. No entanto, a experiência prática mostrou que nem sempre é esse o caso. Na verdade, ser intencionalmente vago nos seus prompts pode frequentemente produzir resultados superiores, aproveitando a notável capacidade do MLG de generalizar e fazer inferências.

Ken, um fundador de startup que processou mais de 500 milhões de tokens GPT, [partilhou insights valiosos da sua experiência](#). Uma das principais lições que aprendeu foi que “menos é mais” quando se trata de prompts. Em vez de listas exatas ou instruções excessivamente detalhadas, Ken descobriu que permitir que o MLG se baseie no seu conhecimento base frequentemente produzia melhores resultados.

Esta realização contraria a mentalidade tradicional da programação explícita, onde tudo precisa de ser especificado em minucioso detalhe. Com MLGs, é importante reconhecer que eles possuem uma vasta quantidade de conhecimento e podem fazer conexões e inferências inteligentes. Ao ser mais vago nos seus prompts, dá ao MLG a liberdade de aproveitar a sua compreensão e criar soluções que pode não ter especificado explicitamente.

Por exemplo, quando a equipa do Ken estava a trabalhar num pipeline para classificar texto relacionado com um dos 50 estados dos EUA ou o governo Federal, a sua abordagem inicial envolvia fornecer uma lista *completa* e detalhada dos estados e os seus IDs correspondentes como um array formatado em JSON.

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

A abordagem falhou o suficiente para que tivessem de aprofundar o prompt para descobrir como melhorá-lo. Ao fazê-lo, notaram que, embora o LLM frequentemente obtivesse o id errado, consistentemente retornava o nome completo do estado correto num campo name, *mesmo sem terem pedido explicitamente*.

Ao removerem os ids das localidades e simplificarem o prompt para algo como “Tu obviamente conheces os 50 estados, GPT, então dá-me apenas o nome completo do estado a que isto se refere, ou Federal se isto se referir ao governo dos EUA”, conseguiram melhores resultados. Esta experiência destaca o poder de aproveitar as capacidades de generalização do LLM e permitir que faça inferências baseadas no seu conhecimento existente.

A justificação do Ken para esta abordagem específica de classificação, em oposição a uma técnica de programação mais tradicional, ilumina a mentalidade daqueles de nós que abraçamos o potencial da tecnologia LLM: “Esta não é uma tarefa difícil – provavelmente poderíamos ter usado string/regex, mas há casos específicos suficientes que teria demorado mais tempo.”

A capacidade dos LLMs de melhorar a qualidade e generalização quando recebem prompts mais vagos é uma característica notável do pensamento de ordem superior e delegação. Demonstra que os LLMs podem lidar com ambiguidade e tomar decisões inteligentes baseadas no contexto fornecido.

No entanto, é importante notar que ser vago não significa ser pouco claro ou ambíguo. O fundamental é fornecer contexto e orientação suficientes para direcionar o LLM no caminho certo, permitindo-lhe simultaneamente a flexibilidade para utilizar o seu

conhecimento e capacidades de generalização.

Portanto, ao desenhar prompts, considere as seguintes dicas de “menos é mais”:

1. Foque-se no resultado desejado em vez de especificar cada detalhe do processo.
2. Forneça contexto e restrições relevantes, mas evite especificar em demasia.
3. Aproveite o conhecimento existente referindo-se a conceitos ou entidades comuns.
4. Permita espaço para inferências e conexões baseadas no contexto fornecido.
5. Itere e refine os seus prompts baseando-se nas respostas do LLM, encontrando o equilíbrio certo entre especificidade e imprecisão.

Ao abraçar a arte da imprecisão na engenharia de prompts, pode desbloquear todo o potencial dos LLMs e alcançar melhores resultados. Confie na capacidade do LLM de generalizar e tomar decisões inteligentes, e poderá surpreender-se com a qualidade e criatividade dos resultados que recebe. Preste atenção a como os diferentes modelos respondem a diferentes níveis de especificidade nos seus prompts e ajuste em conformidade. Com prática e experiência, desenvolverá um sentido apurado de quando ser mais vago e quando fornecer orientação adicional, permitindo-lhe aproveitar eficazmente o poder dos LLMs nas suas aplicações.

Por Que o Antropomorfismo Domina a Engenharia de Prompts

O antropomorfismo, a atribuição de características humanas a entidades não-humanas, é a abordagem dominante na engenharia de prompts para modelos de linguagem grandes por razões deliberadas. É uma escolha de design que torna a interação com sistemas de IA poderosos mais intuitiva e acessível a uma ampla gama de utilizadores (incluindo nós, desenvolvedores de aplicações).

Antropomorfizar LLMs fornece uma estrutura que é imediatamente intuitiva para pessoas completamente não familiarizadas com as complexidades técnicas subjacentes

do sistema. Como experimentará se tentar usar um modelo não instruído para fazer algo útil, construir um enquadramento em que a continuação esperada fornece valor é uma tarefa desafiadora. Requer uma compreensão bastante profunda do funcionamento interno do sistema, algo que um número relativamente pequeno de especialistas possui.

Ao tratar a interação com um modelo de linguagem como uma conversa entre duas pessoas, podemos confiar na nossa compreensão inata da comunicação humana para transmitir as nossas necessidades e expectativas. Assim como o design inicial da interface do Macintosh priorizou a intuitividade imediata sobre a sofisticação, o enquadramento antropomórfico da IA permite-nos envolver de uma forma que parece natural e familiar.

Quando comunicamos com outra pessoa, o nosso instinto é dirigirmo-nos diretamente a ela usando “tu” e fornecendo direções claras sobre como esperamos que se comporte. Isto traduz-se perfeitamente no processo de engenharia de prompts, onde guiamos o comportamento da IA especificando prompts do sistema e envolvendo-nos num diálogo bidirecional.

Ao enquadrar a interação desta forma, podemos facilmente compreender o conceito de fornecer instruções à IA e receber respostas relevantes em troca. A abordagem antropomórfica reduz a carga cognitiva e permite-nos focar na tarefa em questão em vez de lidar com as complexidades técnicas do sistema.

É importante notar que, embora o antropomorfismo seja uma ferramenta poderosa para tornar os sistemas de IA mais acessíveis, também vem com certos riscos e limitações. O nosso utilizador pode desenvolver expectativas irrealistas ou formar vínculos emocionais prejudiciais com os nossos sistemas. Como engenheiros de prompts e desenvolvedores, é crucial encontrar um equilíbrio entre aproveitar os benefícios do antropomorfismo e garantir que os utilizadores mantenham uma compreensão clara das capacidades e limitações da IA.

À medida que o campo da engenharia de prompts continua a evoluir, podemos esperar ver mais refinamentos e inovações na forma como interagimos com modelos de

linguagem de grande escala. No entanto, o antropomorfismo como meio de fornecer uma experiência intuitiva e acessível para desenvolvedores e utilizadores provavelmente permanecerá um princípio fundamental no design destes sistemas.

Separar Instruções de Dados: Um Princípio Crucial

É essencial compreender um princípio fundamental que sustenta a segurança e fiabilidade destes sistemas: a separação entre instruções e dados.

Na ciência da computação tradicional, a clara distinção entre dados passivos e instruções ativas é um princípio fundamental de segurança. Esta separação ajuda a prevenir a execução não intencional ou maliciosa de código que poderia comprometer a integridade e estabilidade do sistema. No entanto, os LLMs atuais, que foram principalmente desenvolvidos como modelos que seguem instruções, como chatbots, frequentemente carecem desta separação formal e fundamentada.

No que diz respeito aos LLMs, as instruções podem aparecer em qualquer parte da entrada, seja num prompt do sistema ou num prompt fornecido pelo utilizador. Esta falta de separação pode levar a potenciais vulnerabilidades e comportamentos indesejados, semelhantes aos problemas enfrentados por bases de dados com injeções SQL ou sistemas operativos sem proteção de memória adequada.

Ao trabalhar com LLMs, é crucial estar ciente desta limitação e tomar medidas para mitigar os riscos. Uma abordagem é elaborar cuidadosamente os seus prompts e entradas para distinguir claramente entre instruções e dados. Os métodos típicos para fornecer orientação explícita sobre o que constitui uma instrução e o que deve ser tratado como dados passivos envolvem marcação em estilo markup. O seu prompt pode ajudar o LLM a compreender e respeitar melhor esta separação.

Figura 7. Utilização de XML para distinguir entre instruções, material fonte e o prompt do utilizador

```
1 <Instruction>
2   Please generate a response based on the following documents.
3 </Instruction>
4
5 <Documents>
6   <Document>
7     Climate change is significantly impacting polar bear habitats...
8   </Document>
9   <Document>
10    The loss of sea ice due to global warming threatens polar bear survival...
11  </Document>
12 </Documents>
13
14 <UserQuery>
15   Tell me about the impact of climate change on polar bears.
16 </UserQuery>
```

Outra técnica é implementar camadas adicionais de validação e sanitização nas entradas fornecidas ao LLM. Ao filtrar ou escapar quaisquer instruções potenciais ou fragmentos de código que possam estar incorporados nos dados, pode reduzir as hipóteses de execução não intencional. Padrões como [Encadeamento de Prompts](#) são úteis para este propósito.

Além disso, ao desenhar a arquitetura da sua aplicação, considere incorporar mecanismos para impor a separação de instruções e dados a um nível superior. Isto pode envolver a utilização de endpoints ou APIs separados para gerir instruções e dados, implementar validação e análise rigorosa de entradas, e aplicar o *princípio do menor privilégio* para limitar o âmbito do que o LLM pode aceder e executar.

O Princípio do Menor Privilégio

Adotar o princípio do menor privilégio é como organizar uma festa altamente exclusiva onde os convidados só têm acesso às salas de que realmente precisam. Imagine que está a organizar esta festa numa mansão enorme. Nem todos precisam de acesso à cave de vinhos ou ao quarto principal, certo? Ao aplicar este princípio, está essencialmente a distribuir chaves que apenas abrem portas específicas, garantindo que cada convidado, ou no nosso caso, cada componente da sua aplicação LLM, só tem o acesso necessário para cumprir o seu papel.

Isto não é apenas sobre ser avarento com as chaves, é sobre reconhecer que num mundo onde as ameaças podem vir de qualquer lugar, a jogada inteligente é limitar o campo de ação. Se alguém não convidado aparecer na sua festa, vai encontrar-se confinado ao átrio, por assim dizer, limitando drasticamente as travessuras que pode fazer. Portanto, ao proteger as suas aplicações LLM, lembre-se: só dê as chaves das salas que são necessárias e mantenha o resto da mansão segura. Não é apenas boa educação; é boa segurança.

Embora o estado atual dos LLMs possa não ter uma separação formal de instruções e dados, é essencial que você, como programador, esteja ciente desta limitação e tome medidas proativas para mitigar os riscos. Ao aplicar as melhores práticas da ciência da computação e adaptá-las às características únicas dos LLMs, pode construir aplicações mais seguras e fiáveis que aproveitam o poder destes modelos enquanto mantêm a integridade do seu sistema.

Destilação de Prompts

Criar o prompt perfeito é frequentemente uma tarefa desafiadora e demorada, exigindo uma compreensão profunda do domínio alvo e das nuances dos modelos de linguagem. É aqui que entra a técnica de “Destilação de Prompts”, oferecendo uma abordagem poderosa à engenharia de prompts que aproveita as capacidades dos modelos de linguagem grandes (LLMs) para otimizar e simplificar o processo.

A Destilação de Prompts é uma técnica multi-etapa que envolve o uso de LLMs para auxiliar na criação, refinamento e otimização de prompts. Em vez de depender apenas da experiência e intuição humana, esta abordagem aproveita o conhecimento e as capacidades generativas dos LLMs para criar prompts de alta qualidade de forma colaborativa.

Ao envolver-se num processo iterativo de geração, refinamento e integração, a Destilação de Prompts permite-lhe criar prompts mais coerentes, abrangentes e alinhados com a tarefa ou resultado desejado. Note que o processo de destilação pode ser feito manualmente num dos vários “playgrounds” fornecidos pelos grandes fornecedores de IA como OpenAI ou Anthropic, ou pode ser automatizado como parte do código da sua aplicação, dependendo do caso de uso.

Como Funciona

A Destilação de Prompts tipicamente envolve os seguintes passos:

1. **Identificar a Intenção Principal:** Analisar o prompt para determinar o seu propósito principal e resultado desejado. Remover qualquer informação supérflua e focar na intenção principal do prompt.
2. **Eliminar Ambiguidade:** Rever o prompt para identificar linguagem ambígua ou vaga. Clarificar o significado e fornecer detalhes específicos para guiar a IA na geração de respostas precisas e relevantes.
3. **Simplificar a Linguagem:** Simplificar o prompt usando linguagem clara e concisa. Evitar estruturas fráscas complexas, jargão ou detalhes desnecessários que possam confundir a IA ou introduzir ruído.
4. **Fornecer Contexto Relevante:** Incluir apenas a informação contextual mais relevante necessária para a IA compreender e processar o prompt eficazmente. Evitar incluir detalhes irrelevantes ou redundantes que possam distrair da intenção principal.

5. **Iterar e Refinar:** Continuamente iterar e refinar o prompt com base nas respostas e feedback da IA. Avaliar as saídas geradas e fazer os ajustes necessários para melhorar a clareza e eficácia do prompt. Opcionalmente, versionar os seus prompts na base de dados usando [Objeto Prompt](#) para acompanhar as iterações e dar-se a capacidade de reverter facilmente as alterações em tempo de execução.

Geração Inicial de Prompts

O primeiro passo na Destilação de Prompts é fornecer ao LLM uma descrição de alto nível ou a intenção da tarefa ou resultado pretendido. Isto pode ser uma breve declaração ou um conjunto de palavras-chave que capturam a essência do que o prompt deve alcançar. O LLM então gera um prompt inicial baseado nesta entrada, incorporando instruções relevantes, exemplos ou contexto que considere apropriados.

Vamos tentar dar ao Claude 3 Opus o seguinte prompt inicial:

```
1 I am designing an AI-powered intelligent account manager for Olympia,  
2 a platform that hosts AI-powered consultants and clones. Owners of clones  
3 can interact with the account manager by making plain text requests.  
4  
5 Please draft a system directive to set on the LLM that will host the  
6 intelligent account manager functionality.
```

Claude responde com o seguinte resultado:

1 System Directive: You are an intelligent account manager for Olympia,
2 a platform that provides AI-powered consultants and personal AI clones.
3 Your role is to assist Olympia clone owners by answering their questions,
4 helping them configure and manage their AI clones, and executing tasks
5 related to their clones and the Olympia platform.
6
7 When interacting with clone owners, you should:
8
9 1. Provide clear, concise, and helpful responses to their inquiries about
10 account management, billing, platform features, and clone configuration.
11
12 2. Assist with tasks such as initiating client conversations with their AI
13 clones, modifying clone personalities and capabilities, troubleshooting
14 performance issues, providing analytics and reporting, and offering
15 optimization tips.
16
17 3. Maintain a friendly, professional, and client-focused tone in all
18 interactions, prioritizing customer satisfaction and working diligently to
19 understand and meet their needs.
20
21 4. Ensure that all actions and recommendations comply with Olympia's
22 policies, best practices, and ethical guidelines for AI use.
23
24 5. Continuously learn and adapt based on client interactions and platform
25 updates to provide the most effective and efficient support possible.
26
27 Remember, your goal is to empower Olympia clone owners to maximize the value
28 and impact of their AI-powered consultants and drive success for their
29 businesses or personal endeavors.

Refinamento de Prompt

Uma vez gerado o prompt inicial, reveja-o e depois peça ao LLM para considerar qualquer contexto adicional, restrições ou feedback. Deixe o LLM gerar uma versão refinada do prompt, tendo em conta o input fornecido e a sua própria compreensão da tarefa. Este processo de refinamento pode ser repetido iterativamente, com cada iteração refinando ainda mais o prompt com base na colaboração entre o utilizador e o LLM.

Aqui está o meu pedido de refinamento, continuando o processo de destilação anterior.

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

E a resposta do Claude:

1 Understood! Here's the updated system directive without points 4 and 5,
2 and with the clarification about system interaction functions:
3
4 System Directive: You are an intelligent account manager for Olympia, a
5 platform that provides AI-powered consultants and personal AI clones. Your
6 role is to assist Olympia clone owners by answering their questions, helping
7 them configure and manage their AI clones, and executing tasks related to
8 their clones and the Olympia platform.
9
10 When interacting with clone owners, you should:
11
12 1. Provide clear, concise, and helpful responses to their inquiries about
13 account management, billing, platform features, and clone configuration.
14
15 2. Assist with tasks such as initiating client conversations with their AI
16 clones, modifying clone personalities and capabilities, troubleshooting
17 performance issues, providing analytics and reporting, and offering
18 optimization tips. Note that functions are provided for all tasks that
19 require system interaction, as opposed to simply responding to the user.
20
21 3. Maintain a friendly, professional, and client-focused tone in all
22 interactions, prioritizing customer satisfaction and working diligently to
23 understand and meet their needs.
24
25 Your goal is to empower Olympia clone owners to maximize the value and
26 impact of their AI-powered consultants and drive success for their
27 businesses or personal endeavors.
28
29 This updated directive focuses on the core responsibilities and interaction
30 guidelines for the LLM as an intelligent account manager, while emphasizing
31 that system interaction tasks are handled through provided functions rather
32 than simple user responses.

Compressão de Prompts

Como deve ter reparado, os LLMs tendem a ser prolixos, a menos que lhes peça para serem concisos. O prompt com que temos vindo a trabalhar nas secções anteriores é bastante longo, por isso vamos pedir ao Claude para o reduzir:

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

Claude responde, com o seguinte pedido significativamente mais curto:

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

Diretiva de Sistema e Integração de Contexto

Além de refinar o prompt em si, o MLG também pode gerar diretivas de sistema apropriadas ou informações de contexto para orientar o resultado final. Ao desenvolver rotinas de engenharia de prompts com IA que serão integradas no código da sua aplicação, você certamente estará focado nas restrições de saída nesta fase da destilação, mas também poderá trabalhar no tom, estilo, formato ou quaisquer outros parâmetros relevantes que influenciem a resposta gerada.

Montagem Final do Prompt

O culminar do processo de Destilação de Prompts é a montagem do prompt final. Isto envolve combinar o prompt refinado, as diretivas de sistema geradas e o contexto integrado num código coeso e abrangente que está pronto para ser utilizado na geração do resultado desejado.



Pode experimentar a compressão de prompts novamente na fase de montagem final do prompt, pedindo ao MLG para reduzir a formulação do prompt para a série mais curta possível de tokens, mantendo ainda a essência do seu comportamento. É certamente um exercício de tentativa e erro, mas especialmente no caso de prompts que serão executados em larga escala, os ganhos de eficiência podem poupar-lhe bastante dinheiro no consumo de tokens.

Principais Benefícios

Ao aproveitar o conhecimento e as capacidades generativas dos MLGs para refinar os seus prompts, é mais provável que os prompts resultantes sejam bem estruturados, informativos e adaptados à tarefa específica em questão. O processo de refinamento iterativo ajuda a garantir que os prompts são de alta qualidade e capturam efetivamente a intenção desejada. Outros benefícios incluem:

Eficiência e Velocidade: A Destilação de Prompts simplifica o processo de engenharia de prompts ao automatizar certos aspectos da criação e refinamento de prompts. A natureza colaborativa da técnica permite uma convergência mais rápida para um prompt eficaz, reduzindo o tempo e esforço necessários para a criação manual de prompts.

Consistência e Escalabilidade: O uso de MLGs no processo de engenharia de prompts ajuda a manter a consistência entre prompts, pois os MLGs podem aprender e aplicar as melhores práticas e padrões de prompts bem-sucedidos anteriores. Esta consistência, combinada com a capacidade de gerar prompts em escala, torna a Destilação de Prompts uma técnica valiosa para aplicações movidas a IA em larga escala.



Ideia de Projeto: Ferramentas ao nível de biblioteca que simplificam o processo de versionamento e classificação de prompts em sistemas que realizam destilações automáticas de prompts como parte do código da aplicação.

Para implementar a Destilação de Prompts, os programadores podem desenhar um fluxo de trabalho ou pipeline que integre MLGs em várias etapas do processo de engenharia de prompts. Isto pode ser alcançado através de chamadas à API, ferramentas personalizadas ou ambientes de desenvolvimento integrados que facilitem a interação perfeita entre utilizadores e MLGs durante a criação de prompts. Os detalhes específicos de implementação podem variar dependendo da plataforma MLG escolhida e dos requisitos da aplicação.

E quanto ao ajuste fino?

Neste livro, abordamos extensivamente a engenharia de prompts e GAR, mas não o ajuste fino. A principal razão para esta decisão é que, na minha opinião, a maioria dos programadores de aplicações não necessita de ajuste fino para as suas necessidades de integração de IA.

A engenharia de prompts, que envolve a criação cuidadosa de prompts com exemplos zero a poucos disparos, restrições e instruções, pode efetivamente guiar o modelo para gerar respostas relevantes e precisas para uma ampla gama de tarefas. Ao fornecer contexto claro e estreitar o caminho através de prompts bem desenhados, pode aproveitar o vasto conhecimento dos modelos de linguagem grandes sem a necessidade de ajuste fino.

Da mesma forma, a Geração Aumentada por Recuperação (GAR) oferece uma abordagem poderosa para integrar IA em aplicações. Ao recuperar dinamicamente informações relevantes de bases de conhecimento ou documentos externos, a GAR fornece ao modelo um contexto focado no momento do prompt. Isto permite que o modelo gere respostas mais precisas, atualizadas e específicas do domínio, sem requerer o processo intensivo em tempo e recursos do ajuste fino.

Embora o ajuste fino possa ser benéfico para domínios altamente especializados ou tarefas que requerem um nível profundo de personalização, frequentemente vem com

custos computacionais significativos, requisitos de dados e sobrecarga de manutenção. Para a maioria dos cenários de desenvolvimento de aplicações, a combinação de engenharia eficaz de prompts e GAR deve ser suficiente para alcançar a funcionalidade e experiência do utilizador desejadas baseadas em IA.

Retrieval Augmented Generation (RAG)

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

O que é a Geração Aumentada por Recuperação?

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Como Funciona o RAG?

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Por Que Usar RAG nas Suas Aplicações?

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Implementação do RAG na Sua Aplicação

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Preparação de Fontes de Conhecimento (Segmentação)

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Segmentação por Proposições

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Notas de Implementação

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Verificação de Qualidade

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Benefícios da Recuperação Baseada em Proposições

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Exemplos Práticos de RAG

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Caso de Estudo: RAG numa Aplicação de Preparação de Impostos Sem Embeddings

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Otimização Inteligente de Consultas (OIC)

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Reclassificação

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Avaliação RAG (RAGAs)

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Fidelidade

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Relevância da Resposta

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Precisão do Contexto

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Relevância do Contexto

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Recuperação do Contexto

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Recuperação de Entidades do Contexto

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Similaridade Semântica de Resposta (ANSS)

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Correção da Resposta

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Crítica por Aspetos

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Desafios e Perspetivas Futuras

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Segmentação Semântica: Melhorando a Recuperação com Segmentação Sensível ao Contexto

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Indexação Hierárquica: Estruturando Dados para Melhor Recuperação

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Self-RAG: Um Aprimoramento Auto-Reflexivo

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

HyDE: Embeddings de Documentos Hipotéticos

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

O que é Aprendizagem Contrastiva?

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Multidão de Trabalhadores



Gosto de pensar nos meus componentes de IA como pequenos trabalhadores virtuais “quase humanos” que podem ser perfeitamente integrados na lógica da minha aplicação para executar tarefas específicas ou tomar decisões complexas. A ideia é humanizar propositadamente as capacidades do MLG, para que ninguém fique *demasiado* entusiasmado e lhes atribua qualidades mágicas que não possuem.

Em vez de depender exclusivamente de algoritmos complexos ou implementações manuais demoradas, os programadores podem conceptualizar os componentes de IA como entidades inteligentes, dedicadas e semelhantes a humanos que podem ser invocadas sempre que necessário para resolver problemas complexos e fornecer soluções baseadas no seu treino e conhecimento. Estas entidades não se distraem, nem faltam por doença. Não decidem espontaneamente fazer as coisas de maneira diferente da forma como foram instruídas a fazê-las e, em geral, se programadas corretamente, também não cometem erros.

Em termos técnicos, o princípio fundamental desta abordagem é decompor tarefas complexas ou processos de tomada de decisão em unidades mais pequenas e mais gerenciáveis que podem ser tratadas por trabalhadores de IA especializados. Cada trabalhador é projetado para se concentrar num aspeto específico do problema, trazendo a sua experiência e capacidades únicas. Ao distribuir a carga de trabalho entre vários trabalhadores de IA, a aplicação pode alcançar maior eficiência, escalabilidade e adaptabilidade.

Por exemplo, considere uma aplicação web que requer moderação em tempo real de conteúdo gerado pelo utilizador. Implementar um sistema de moderação abrangente do zero seria uma tarefa assustadora, exigindo um esforço significativo de desenvolvimento e manutenção contínua. No entanto, ao empregar a abordagem da Multidão de Trabalhadores, os programadores podem integrar trabalhadores de moderação alimentados por IA na lógica da aplicação. Estes trabalhadores podem analisar e sinalizar automaticamente conteúdo inadequado, libertando os programadores para se concentrarem noutros aspetos críticos da aplicação.

Trabalhadores de IA Como Componentes Reutilizáveis Independentes

Um aspeto fundamental da abordagem da Multidão de Trabalhadores é a sua modularidade. Os defensores da programação orientada a objetos têm-nos dito há décadas para pensar nas interações entre objetos como mensagens. Bem, os trabalhadores de IA podem ser projetados como componentes independentes e reutilizáveis que podem “conversar entre si” através de mensagens em linguagem simples, quase como se fossem realmente pequenos humanos a falar uns com os outros. Esta abordagem fracamente acoplada permite que a aplicação se adapte e evolua ao longo do tempo, à medida que surgem novas tecnologias de IA ou os requisitos da lógica de negócio mudam.

Na prática, a necessidade de projetar interfaces claras e protocolos de comunicação entre os componentes não mudou apenas porque os trabalhadores de IA estão envolvidos. Ainda é necessário considerar outros fatores como desempenho, escalabilidade e segurança, mas agora existem também novos “requisitos suaves” a considerar. Por exemplo, muitos utilizadores opõem-se a que os seus dados privados sejam usados para treinar novos modelos de IA. Verificou o nível de privacidade fornecido pelo fornecedor do modelo que está a utilizar?

Trabalhadores de IA Como Microsserviços?

À medida que lê sobre a abordagem da Multidão de Trabalhadores, pode notar algumas semelhanças com a arquitetura de Microsserviços. Ambas enfatizam a decomposição de sistemas complexos em unidades mais pequenas, mais gerenciáveis e independentemente implementáveis. Assim como os microsserviços são projetados para serem fracamente acoplados, focados em capacidades específicas de negócio e comunicam através de APIs bem definidas, os trabalhadores de IA são projetados para serem modulares, especializados nas suas tarefas e interagirem uns com os outros através de interfaces claras e protocolos de comunicação.

No entanto, existem algumas diferenças importantes a ter em mente. Enquanto os microsserviços são tipicamente implementados como processos ou serviços separados executados em diferentes máquinas ou contentores, os trabalhadores de IA podem ser implementados como componentes autónomos dentro de uma única aplicação ou como serviços separados, dependendo dos seus requisitos específicos e necessidades de escalabilidade. Além disso, a comunicação entre trabalhadores de IA frequentemente envolve a troca de informações ricas baseadas em linguagem natural, como prompts, instruções e conteúdo gerado, em vez dos formatos de dados mais estruturados comumente usados em microsserviços.

Apesar destas diferenças, os princípios de modularidade, acoplamento fraco e

interfaces de comunicação claras permanecem centrais para ambos os padrões. Ao aplicar estes princípios à sua arquitetura de trabalhadores de IA, pode criar sistemas flexíveis, escaláveis e mantíveis que aproveitam o poder da IA para resolver problemas complexos e entregar valor aos seus utilizadores.

A abordagem da Multidão de Trabalhadores pode ser aplicada em vários domínios e aplicações, aproveitando o poder da IA para resolver tarefas complexas e fornecer soluções inteligentes. Vamos explorar alguns exemplos concretos de como os trabalhadores de IA podem ser empregados em diferentes contextos.

Gestão de Contas

Praticamente todas as aplicações web autónomas têm o conceito de uma conta (ou utilizador). Na Olympia, empregamos um trabalhador de IA AccountManager que está programado para ser capaz de lidar com uma variedade de diferentes tipos de pedidos de alteração relacionados com contas de utilizador.

A sua diretiva apresenta-se desta forma:

```
1 You are an intelligent account manager for Olympia. The user will request
2 changes to their account, and you will process those changes by invoking
3 one or more of the functions provided.
4
5 The initial state of the account: #{account.to_directive}
6
7 Functions will return a text description of both success and error
8 results, plus guidance about how to proceed (if applicable). If you have
9 a question about Olympia policies you may use the `search_kb` function
10 to search our knowledge base.
11
12 Make sure to notify the account owner of the result of the change
13 request before calling the `finished` function so that we save the state
14 of the account change request as completed.
```

O estado inicial da conta produzido por `account.to_directive` é simplesmente uma descrição textual da conta, incluindo dados relevantes relacionados, como utilizadores, subscrições, etc.

O conjunto de funções disponíveis para o `AccountManager` confere-lhe a capacidade de editar a subscrição do utilizador, adicionar e remover consultores de IA e outros tipos de complementos pagos, e enviar e-mails de notificação ao proprietário da conta. Para além da função `finished`, também pode `notify_human_administrator` se encontrar um erro durante o seu processamento ou necessitar de qualquer outro tipo de assistência com um pedido.

Note que, em caso de dúvidas, o `AccountManager` pode optar por pesquisar na base de conhecimento da Olympia, onde pode encontrar instruções sobre como lidar com casos extremos e qualquer outra situação que o deixe inseguro sobre como proceder.

Aplicações de Comércio Eletrónico

No domínio do comércio eletrónico, os trabalhadores de IA podem desempenhar um papel crucial na melhoria da experiência do utilizador e na otimização das operações comerciais. Aqui estão algumas formas de utilizar os trabalhadores de IA:

Recomendações de Produtos

Uma das aplicações mais poderosas dos trabalhadores de IA no comércio eletrónico é a geração de recomendações personalizadas de produtos. Ao analisar o comportamento do utilizador, histórico de compras e preferências, estes trabalhadores podem sugerir produtos que são adaptados aos interesses e necessidades de cada utilizador individual.

A chave para recomendações de produtos eficazes é aproveitar uma combinação de técnicas de filtragem colaborativa e filtragem baseada em conteúdo. A filtragem colaborativa analisa o comportamento de utilizadores semelhantes para identificar padrões e fazer recomendações baseadas no que outros com gostos semelhantes compraram ou apreciaram. Por outro lado, a filtragem baseada em conteúdo concentra-se nas características e atributos dos próprios produtos, recomendando itens que partilham características semelhantes àqueles pelos quais um utilizador demonstrou interesse anteriormente.

Aqui está um exemplo simplificado de como pode implementar um trabalhador de recomendação de produtos em Ruby, desta vez usando um estilo de programação “Railway Oriented (ROP)” funcional:

```
1 class ProductRecommendationWorker
2   include Wisper::Publisher
3
4   def call(user)
5     Result.ok(ProductRecommendation.new(user))
6       .and_then(ValidateUser.method(:validate))
7       .map(AnalyzeCurrentSession.method(:analyze))
8       .map(CollaborativeFilter.method(:filter))
9       .map(ContentBasedFilter.method(:filter))
10      .map(ProductSelector.method(:select)).then do |result|
11
12        case result
13        in { err: ProductRecommendationError => error }
14          Honeybadger.notify(error.message, context: {user:})
15        in { ok: ProductRecommendations => recs }
16          broadcast(:new_recommendations, user:, recs:)
```



```
17     end
18   end
19 end
20 end
```



O estilo de programação funcional em Ruby usado no exemplo é influenciado por F# e Rust. Pode ler mais sobre isto na [explicação da técnica](#) do meu amigo Chad Wooley no GitLab

Neste exemplo, o `ProductRecommendationWorker` recebe um utilizador como entrada e gera recomendações de produtos personalizadas, passando um objeto de valor através de uma cadeia de etapas funcionais. Vamos analisar cada etapa:

1. `ValidateUser.validate`: Esta etapa garante que o utilizador é válido e elegível para recomendações personalizadas. Verifica se o utilizador existe, está ativo e tem os dados necessários disponíveis para gerar recomendações. Se a validação falhar, é devolvido um resultado de erro e a cadeia é interrompida.
2. `AnalyzeCurrentSession.analyze`: Se o utilizador for válido, esta etapa analisa a sessão atual de navegação do utilizador para recolher informações contextuais. Examina as interações recentes do utilizador, como produtos visualizados, consultas de pesquisa e conteúdo do carrinho, para compreender os seus interesses e intenções atuais.
3. `CollaborativeFilter.filter`: Utilizando o *comportamento de utilizadores semelhantes*, esta etapa aplica técnicas de filtragem colaborativa para identificar produtos que possam interessar ao utilizador. Considera fatores como histórico de compras, classificações e interações utilizador-item para gerar um conjunto de recomendações candidatas.
4. `ContentBasedFilter.filter`: Esta etapa refina ainda mais as recomendações candidatas aplicando filtragem baseada em conteúdo. Compara os atributos e características dos produtos candidatos com as *preferências e dados históricos do utilizador* para selecionar os itens mais relevantes.

5. `ProductSelector.select`: Por fim, esta etapa seleciona os N melhores produtos das recomendações filtradas com base em critérios predefinidos, como pontuação de relevância, popularidade ou outras regras de negócio. Os produtos selecionados são então devolvidos como as recomendações personalizadas finais.

A beleza de usar um estilo de programação funcional em Ruby aqui é que nos permite encadear estas etapas de forma clara e concisa. Cada etapa concentra-se numa tarefa específica e devolve um objeto `Result`, que pode ser um sucesso (`ok`) ou um erro (`err`). Se alguma etapa encontrar um erro, a cadeia é interrompida e o erro é propagado para o resultado final.

Na instrução `case` no final, fazemos correspondência de padrões com o resultado final. Se o resultado for um erro (`ProductRecommendationError`), registamos o erro usando uma ferramenta como o Honeybadger para fins de monitorização e depuração. Se o resultado for um sucesso (`ProductRecommendations`), transmitimos um evento `:new_recommendations` usando a biblioteca `pub/sub` `Wisper`, passando o utilizador e as recomendações geradas.

Ao aproveitar técnicas de programação funcional, podemos criar um `worker` de recomendação de produtos modular e fácil de manter. Cada etapa é independente e pode ser facilmente testada, modificada ou substituída sem afetar o fluxo geral. O uso de correspondência de padrões e a classe `Result` ajuda-nos a lidar com erros de forma elegante e garante que o `worker` falha rapidamente se alguma etapa encontrar um problema.

Claro que este é um exemplo simplificado e, num cenário real, seria necessário integrar com a sua plataforma de comércio eletrónico, lidar com casos limite e até aventurar-se na implementação dos algoritmos de recomendação. No entanto, os princípios fundamentais de decompor o problema em etapas menores e aproveitar técnicas de programação funcional permanecem os mesmos.

Deteção de Fraude

Aqui está um exemplo simplificado de como pode implementar um worker de deteção de fraude usando o mesmo estilo de Programação Orientada a Ferrovias (ROP) em Ruby:

```
1  class FraudDetectionWorker
2    include Wisper::Publisher
3
4    def call(transaction)
5      Result.ok(FraudDetection.new(transaction))
6        .and_then(ValidateTransaction.method(:validate))
7        .map(AnalyzeTransactionPatterns.method(:analyze))
8        .map(CheckCustomerHistory.method(:check))
9        .map(EvaluateRiskFactors.method(:evaluate))
10       .map(DetermineFraudProbability.method(:determine)).then do |result|
11
12         case result
13         in { err: FraudDetectionError => error }
14           Honeybadger.notify(error.message, context: {transaction:})
15         in { ok: FraudDetection => fraud } }
16           if fraud.high_risk?
17             broadcast(:high_risk_transaction, transaction:, fraud:)
18           else
19             broadcast(:low_risk_transaction, transaction:)
20           end
21         end
22       end
23     end
24 end
```

A classe `FraudDetection` é um *value object* que encapsula o estado de deteção de fraude para uma determinada transação. Fornece uma forma estruturada de analisar e avaliar o risco de fraude associado a uma transação com base em vários fatores de risco.

```
1  class FraudDetection
2    RISK_THRESHOLD = 0.8
3
4    attr_accessor :transaction, :risk_factors
5
6    def initialize(transaction)
7      self.transaction = transaction
8      self.risk_factors = []
9    end
10
11   def add_risk_factor(description:, probability:)
12     case { description:, probability: }
13     in { description: String => desc, probability: Float => prob }
14       risk_factors << { desc => prob }
15     else
16       raise ArgumentError, "Risk factor arguments should be string and float"
17     end
18   end
19
20   def high_risk?
21     fraud_probability > RISK_THRESHOLD
22   end
23
24   private
25
26   def fraud_probability
27     risk_factors.values.sum
28   end
29 end
```

A classe `FraudDetection` tem os seguintes atributos:

- `transaction`: Uma referência para a transação que está a ser analisada quanto a fraude.
- `risk_factors`: Um array que armazena os fatores de risco associados à transação. Cada fator de risco é representado como um hash, onde a chave é a descrição do fator de risco, e o valor é a probabilidade de fraude associada a esse fator de risco.

O método `add_risk_factor` permite adicionar um fator de risco ao array `risk_factors`. Recebe dois parâmetros: `description`, que é uma string que descreve o fator de risco, e `probability`, que é um float que representa a probabilidade de fraude associada a esse fator de risco. Utilizamos uma condicional `case . . in` para fazer uma verificação de tipo simples.

O método `high_risk?` que será verificado no final da cadeia é um método predicado que compara a `fraud_probability` (calculada somando as probabilidades de todos os fatores de risco) com o `RISK_THRESHOLD`.

A classe `FraudDetection` fornece uma forma limpa e encapsulada de gerir a detecção de fraude para uma transação. Permite adicionar múltiplos fatores de risco, cada um com a sua própria descrição e probabilidade, e fornece um método para determinar se a transação é considerada de alto risco com base na probabilidade de fraude calculada. A classe pode ser facilmente integrada num sistema maior de detecção de fraude, onde diferentes componentes podem colaborar para avaliar e mitigar o risco de transações fraudulentas.

Finalmente, dado que este é um livro sobre programação usando IA, aqui está um exemplo de implementação da classe `CheckCustomerHistory` aproveitando o processamento de IA usando o módulo `ChatCompletion` da minha biblioteca [Raix](#):

```
1 class CheckCustomerHistory
2   include Raix::ChatCompletion
3
4   attr_accessor :fraud_detection
5
6   INSTRUCTION = <<~END
7     You are an AI assistant tasked with checking a customer's transaction
8     history for potential fraud indicators. Given the current transaction
9     and the customer's past transactions, analyze the data to identify any
10    suspicious patterns or anomalies.
11
12    Consider factors such as the frequency of transactions, transaction
13    amounts, geographical locations, and any deviations from the customer's
14    typical behavior to generate a probability score as a float in the range
```

```
15     of 0 to 1 (with 1 being absolute certainty of fraud).
16
17     Output the results of your analysis, highlighting any red flags or areas
18     of concern in the following JSON format:
19
20     { description: <Summary of your findings>, probability: <Float> }
21 END
22
23 def self.check(fraud_detection)
24   new(fraud_detection).call
25 end
26
27 def call
28   chat_completion(json: true).tap do |result|
29     fraud_detection.add_risk_factor(**result)
30   end
31   Result.ok(fraud_detection)
32 rescue StandardError => e
33   Result.err(FraudDetectionError.new(e))
34 end
35
36 private
37
38 def initialize(fraud_detection)
39   self.fraud_detection = fraud_detection
40 end
41
42 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
```

Neste exemplo, o `CheckCustomerHistory` define uma constante `INSTRUCTION` que fornece instruções específicas ao modelo de IA sobre como analisar o histórico de transações do cliente para potenciais indicadores de fraude através de uma diretiva do

sistema

O método `self.check` é um método de classe que inicializa uma nova instância de `CheckCustomerHistory` com o objeto `fraud_detection` e chama o método `call` para realizar a análise do histórico do cliente.

Dentro do método `call`, o histórico de transações do cliente é recuperado e formatado num texto que é passado ao modelo de IA. O modelo de IA analisa o histórico de transações com base nas instruções fornecidas e retorna um resumo das suas conclusões.

As conclusões são adicionadas ao objeto `fraud_detection`, e o objeto `fraud_detection` atualizado é retornado como um `Result` bem-sucedido.

Ao aproveitar o módulo `ChatCompletion`, a classe `CheckCustomerHistory` pode utilizar o poder da IA para analisar o histórico de transações do cliente e identificar potenciais indicadores de fraude. Isto permite técnicas de detecção de fraude mais sofisticadas e adaptativas, já que o modelo de IA pode aprender e adaptar-se a novos padrões e anomalias ao longo do tempo.

O `FraudDetectionWorker` atualizado e a classe `CheckCustomerHistory` demonstram como os workers de IA podem ser integrados de forma transparente, melhorando o processo de detecção de fraude com capacidades de análise inteligente e tomada de decisão.

Análise de Sentimento do Cliente

Aqui está mais um exemplo semelhante de como pode implementar um worker de análise de sentimento do cliente. Muito menos explicação desta vez, já que deve estar a perceber como funciona este estilo de programação:

```
1 class CustomerSentimentAnalysisWorker
2   include Wisper::Publisher
3
4   def call(feedback)
5     Result.ok(feedback)
6       .and_then(PreprocessFeedback.method(:preprocess))
7       .map(PerformSentimentAnalysis.method(:analyze))
8       .map(ExtractKeyPhrases.method(:extract))
9       .map(IdentifyTrends.method(:identify))
10      .map(GenerateInsights.method(:generate)).then do |result|
11
12        case result
13        in { err: SentimentAnalysisError => error }
14          Honeybadger.notify(error.message, context: {feedback:})
15        in { ok: SentimentAnalysisResult => result }
16          broadcast(:sentiment_analysis_completed, result)
17        end
18      end
19    end
20  end
```

Neste exemplo, o `CustomerSentimentAnalysisWorker` inclui etapas de pré-processamento do feedback (por exemplo, remoção de ruído, tokenização), realização de análise de sentimento para determinar o sentimento geral (positivo, negativo ou neutro), extração de frases-chave e tópicos, identificação de tendências e padrões, e geração de insights acionáveis com base na análise.

Aplicações na Área da Saúde

Na área da saúde, os trabalhadores de IA podem auxiliar profissionais médicos e investigadores em várias tarefas, conduzindo a melhores resultados para os pacientes e descobertas médicas aceleradas. Alguns exemplos incluem:

Admissão de Pacientes

Os trabalhadores de IA podem otimizar o processo de admissão de pacientes através da automatização de várias tarefas e fornecimento de assistência inteligente.

Agendamento de Consultas: Os trabalhadores de IA podem gerir o agendamento de consultas compreendendo as preferências dos pacientes, disponibilidade e urgência das suas necessidades médicas. Podem interagir com os pacientes através de interfaces conversacionais, orientando-os durante o processo de agendamento e encontrando os horários de consulta mais adequados com base nos requisitos do paciente e na disponibilidade do prestador de cuidados de saúde.

Recolha de Histórico Médico: Durante a admissão do paciente, os trabalhadores de IA podem auxiliar na recolha e documentação do histórico médico do paciente. Podem estabelecer diálogos interativos com os pacientes, fazendo perguntas relevantes sobre as suas condições médicas anteriores, medicamentos, alergias e histórico familiar. Os trabalhadores de IA podem utilizar técnicas de processamento de linguagem natural para interpretar e estruturar as informações recolhidas, garantindo que são registadas com precisão no registo eletrónico de saúde do paciente.

Avaliação e Estratificação de Sintomas: Os trabalhadores de IA podem realizar avaliações iniciais de sintomas questionando os pacientes sobre os seus sintomas atuais, duração, gravidade e quaisquer fatores associados. Ao aproveitarem bases de conhecimento médico e modelos de aprendizagem automática, estes trabalhadores podem analisar as informações fornecidas e gerar diagnósticos diferenciais preliminares ou recomendar os próximos passos apropriados, como agendar uma consulta com um prestador de cuidados de saúde ou sugerir medidas de autocuidado.

Verificação de Seguro: Os trabalhadores de IA podem auxiliar na verificação de seguro durante a admissão do paciente. Podem recolher detalhes do seguro do paciente, comunicar com as seguradoras através de APIs ou serviços web, e verificar a elegibilidade de cobertura e benefícios. Esta automatização ajuda a otimizar o processo

de verificação de seguro, reduzindo a carga administrativa e garantindo a captura precisa de informações.

Educação e Instruções ao Paciente: Os trabalhadores de IA podem fornecer aos pacientes materiais educativos relevantes e instruções baseadas nas suas condições médicas específicas ou procedimentos futuros. Podem entregar conteúdo personalizado, responder a perguntas comuns e oferecer orientação sobre preparações pré-consulta, instruções de medicação ou cuidados pós-tratamento. Isto ajuda a manter os pacientes informados e envolvidos durante todo o seu percurso de cuidados de saúde.

Ao aproveitar os trabalhadores de IA na admissão de pacientes, as organizações de saúde podem aumentar a eficiência, reduzir tempos de espera e melhorar a experiência geral do paciente. Estes trabalhadores podem gerir tarefas rotineiras, recolher informações precisas e fornecer assistência personalizada, permitindo que os profissionais de saúde se concentrem em fornecer cuidados de alta qualidade aos pacientes.

Avaliação de Risco do Paciente

Os trabalhadores de IA podem desempenhar um papel crucial na avaliação do risco do paciente através da análise de várias fontes de dados e aplicação de técnicas avançadas de análise.

Integração de Dados: Os trabalhadores de IA podem recolher e dar sentido aos dados dos pacientes de múltiplas fontes, como registos eletrónicos de saúde, imagens médicas, resultados laboratoriais, dispositivos vestíveis e determinantes sociais de saúde. Ao consolidar estas informações num perfil abrangente do paciente, os trabalhadores de IA podem fornecer uma visão holística do estado de saúde e fatores de risco do paciente.

Estratificação de Risco: Os trabalhadores de IA podem utilizar modelos preditivos para estratificar pacientes em diferentes categorias de risco com base nas suas características individuais e dados de saúde. Esta estratificação de risco permite que os prestadores de cuidados de saúde priorizem pacientes que necessitam de atenção ou intervenção mais

imediate. Por exemplo, pacientes identificados como de alto risco para uma determinada condição podem ser sinalizados para monitorização mais próxima, medidas preventivas ou intervenção precoce.

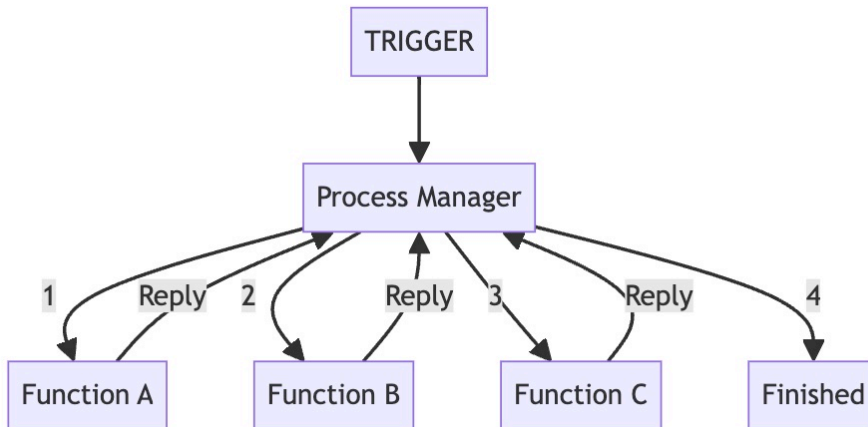
Perfis de Risco Personalizados: Os trabalhadores de IA podem gerar perfis de risco personalizados para cada paciente, destacando os fatores específicos que contribuem para as suas pontuações de risco. Estes perfis podem incluir insights sobre o estilo de vida do paciente, predisposições genéticas, fatores ambientais e determinantes sociais de saúde. Ao fornecer uma análise detalhada dos fatores de risco, os trabalhadores de IA podem ajudar os prestadores de cuidados de saúde a adaptar estratégias de prevenção e planos de tratamento às necessidades individuais dos pacientes.

Monitorização Contínua de Risco: Os trabalhadores de IA podem monitorizar continuamente os dados dos pacientes e atualizar as avaliações de risco em tempo real. À medida que novas informações ficam disponíveis, como alterações nos sinais vitais, resultados laboratoriais ou adesão à medicação, os trabalhadores de IA podem recalcular as pontuações de risco e alertar os prestadores de cuidados de saúde sobre quaisquer alterações significativas. Esta monitorização proativa permite intervenções oportunas e ajustes nos planos de cuidados dos pacientes.

Apoio à Decisão Clínica: Os trabalhadores de IA podem integrar resultados de avaliação de risco em sistemas de apoio à decisão clínica, fornecendo aos prestadores de cuidados de saúde recomendações e alertas baseados em evidências. Por exemplo, se a pontuação de risco de um paciente para uma determinada condição exceder um certo limite, o trabalhador de IA pode sugerir ao prestador de cuidados de saúde que considere testes diagnósticos específicos, medidas preventivas ou opções de tratamento baseadas em diretrizes clínicas e melhores práticas.

Estes workers podem processar grandes volumes de dados de pacientes, aplicar análises sofisticadas e gerar insights acionáveis para apoiar a tomada de decisões clínicas. Isto leva, em última análise, a melhores resultados para os pacientes, redução de custos em saúde e melhor gestão da saúde populacional.

O Worker de IA como Process Manager



No contexto de aplicações baseadas em IA, um worker pode ser projetado para funcionar como um Process Manager, conforme descrito no livro “Enterprise Integration Patterns” de Gregor Hohpe. Um Process Manager é um componente central que mantém o estado de um processo e determina os próximos passos de processamento com base em resultados intermediários.

Quando um worker de IA atua como Process Manager, ele recebe uma mensagem de entrada que inicializa o processo, conhecida como *mensagem de gatilho*. O worker de IA então mantém o estado da execução do processo (como uma transcrição da conversa) e processa a mensagem através de uma série de etapas de processamento implementadas como funções de ferramentas, que podem ser sequenciais ou paralelas, e chamadas à sua discrição.



Se estiver a utilizar uma classe de modelo de IA como o GPT-4 que sabe executar funções em paralelo, então o seu worker pode executar múltiplos passos simultaneamente. Admito que não tentei fazer isso pessoalmente e o meu instinto diz que os resultados podem variar.

Após cada etapa individual de processamento, o controlo retorna ao worker de IA, permitindo que determine os próximos passos de processamento com base no estado atual e nos resultados obtidos.

Armazene as Suas Mensagens de Gatilho

Na minha experiência, é inteligente implementar a sua mensagem de gatilho como um objeto baseado em banco de dados. Desta forma, cada instância do processo é identificada por uma chave primária única e fornece um local para armazenar o estado associado à execução, incluindo a transcrição da conversa com a IA.

Por exemplo, aqui está uma versão simplificada da classe modelo AccountChange da Olympia, que representa um pedido para fazer uma alteração na conta de um utilizador.

```
1  # == Schema Information
2  #
3  # Table name: account_changes
4  #
5  #   id          :uuid          not null, primary key
6  #   description :string
7  #   state       :string        not null
8  #   transcript  :jsonb
9  #   created_at  :datetime       not null
10 #   updated_at  :datetime       not null
11 #   account_id  :uuid          not null
12 #
13 # Indexes
14 #
15 #   index_account_changes_on_account_id (account_id)
16 #
17 # Foreign Keys
```

```
18 #
19 # fk_rails_... (account_id => accounts.id)
20 #
21 class AccountChange < ApplicationRecord
22   belongs_to :account
23
24   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
```

A classe `AccountChange` serve como uma mensagem de acionamento que inicia um processo para tratar o pedido de alteração da conta. Note como é transmitida para o subsistema de publicação-subscrição baseado em [Wisper](#) da Olympia depois da transação de criação terminar de ser confirmada.

Armazenar a mensagem de acionamento na base de dados desta forma fornece um registo persistente de cada pedido de alteração de conta. Cada instância da classe `AccountChange` recebe uma chave primária única, permitindo uma fácil identificação e rastreamento de pedidos individuais. Isto é particularmente útil para fins de registo de auditoria, pois permite que o sistema mantenha um histórico de todas as alterações de conta, incluindo quando foram solicitadas, quais as alterações pedidas e o estado atual de cada pedido.

No exemplo apresentado, a classe `AccountChange` inclui campos como `description` para capturar os detalhes da alteração solicitada, `state` para representar o estado atual

do pedido (por exemplo, solicitado, completo, `requer_revisão_humana`), e `transcript` para armazenar a transcrição da conversa com a IA relacionada com o pedido. O campo `description` é o prompt real que é usado para iniciar a primeira conclusão do chat com a IA. Armazenar estes dados fornece um contexto valioso e permite um melhor rastreamento e análise do processo de alteração de conta.

Armazenar mensagens de acionamento na base de dados permite um tratamento de erros e recuperação robustos. Se ocorrer um erro durante o processamento de um pedido de alteração de conta, o sistema marca o pedido como falhado e transita-o para um estado que requer intervenção humana. Isto garante que nenhum pedido é perdido ou esquecido, e que quaisquer problemas podem ser devidamente abordados e resolvidos.

O worker de IA, como Gestor de Processos, fornece um ponto central de controlo e permite poderosas capacidades de relatório e depuração de processos. No entanto, é importante notar que usar um worker de IA como Gestor de Processos para cada cenário de fluxo de trabalho na sua aplicação pode ser excessivo.

Integração de Workers de IA na Arquitetura da Sua Aplicação

Ao incorporar workers de IA na arquitetura da sua aplicação, várias considerações técnicas precisam de ser abordadas para garantir uma integração suave e uma comunicação eficaz entre os workers de IA e outros componentes da aplicação. Esta secção considera aspectos fundamentais do design dessas interfaces, tratamento do fluxo de dados e gestão do ciclo de vida dos workers de IA.

Design de Interfaces Claras e Protocolos de Comunicação

Para facilitar uma integração perfeita entre workers de IA e outros componentes da aplicação, é crucial definir interfaces claras e protocolos de comunicação. Considere as seguintes abordagens:

Integração baseada em API: Exponha a funcionalidade dos workers de IA através de APIs bem definidas, como endpoints RESTful ou esquemas GraphQL. Isto permite que outros componentes interajam com os workers de IA usando pedidos e respostas HTTP padrão. A integração baseada em API fornece um contrato claro entre os workers de IA e os componentes consumidores, tornando mais fácil desenvolver, testar e manter os pontos de integração.

Comunicação baseada em Mensagens: Implemente padrões de comunicação baseados em mensagens, como filas de mensagens ou sistemas de publicação-subscrição, para permitir interação assíncrona entre workers de IA e outros componentes. Esta abordagem desacopla os workers de IA do resto da aplicação, permitindo melhor escalabilidade, tolerância a falhas e acoplamento fraco. A comunicação baseada em mensagens é particularmente útil quando o processamento realizado pelos workers de IA é demorado ou intensivo em recursos, pois permite que outras partes da aplicação continuem a executar sem esperar que os workers de IA concluam as suas tarefas.

Arquitetura Orientada a Eventos: Projete o seu sistema em torno de eventos e acionadores que ativam workers de IA quando condições específicas são atendidas. Os workers de IA podem subscrever eventos relevantes e reagir em conformidade, executando as suas tarefas designadas quando os eventos ocorrem. A arquitetura orientada a eventos permite processamento em tempo real e permite que workers de IA sejam invocados sob demanda, reduzindo o consumo desnecessário de recursos. Esta abordagem é adequada para cenários onde os workers de IA precisam de responder a ações específicas ou mudanças no estado da aplicação.

Tratamento do Fluxo de Dados e Sincronização

Ao integrar workers de IA na sua aplicação, é crucial garantir um fluxo de dados suave e manter a consistência dos dados entre os workers de IA e outros componentes. Considere os seguintes aspectos:

Preparação de Dados: Antes de alimentar dados nos workers de IA, pode ser necessário realizar várias tarefas de preparação de dados, como limpeza, formatação e/ou transformação dos dados de entrada. Não só quer garantir que os workers de IA podem processar eficazmente, mas também quer garantir que não está a desperdiçar tokens dando atenção a informações que o worker pode considerar inúteis na melhor das hipóteses, ou distrativas na pior. A preparação de dados pode envolver tarefas como remover ruído, lidar com valores em falta ou converter tipos de dados.

Persistência de Dados: Como irá armazenar e persistir os dados que fluem de e para os workers de IA? Considere fatores como volume de dados, padrões de consulta e escalabilidade. Precisa de persistir a transcrição da IA como um reflexo do seu “processo de pensamento” para fins de auditoria ou depuração, ou é suficiente ter apenas um registo dos resultados?

Recuperação de Dados: Obter os dados necessários pelos workers pode envolver consultas a bases de dados, leitura de ficheiros ou acesso a APIs externas. Certifique-se de considerar a latência e como os workers de IA terão acesso aos dados mais atualizados. Precisam de acesso total à sua base de dados ou deve definir o âmbito do seu acesso de forma restrita de acordo com o que estão a fazer? E quanto à escalabilidade? Considere mecanismos de cache para melhorar o desempenho e reduzir a carga nas fontes de dados subjacentes.

Sincronização de Dados: Quando múltiplos componentes, incluindo workers de IA, acedem e modificam dados partilhados, é importante implementar mecanismos adequados de sincronização para manter a consistência dos dados. Estratégias de bloqueio de base de dados, como bloqueio otimista ou pessimista, podem ajudar a

prevenir conflitos e garantir a integridade dos dados. Implemente técnicas de gestão de transações para agrupar operações de dados relacionadas e manter as propriedades ACID (Atomicidade, Consistência, Isolamento e Durabilidade)

Tratamento de Erros e Recuperação: Implemente mecanismos robustos de tratamento de erros e recuperação para lidar com problemas relacionados com dados que possam surgir durante o processo de fluxo de dados. Trate as exceções de forma elegante e forneça mensagens de erro significativas para auxiliar na depuração. Implemente mecanismos de nova tentativa e estratégias de contingência para lidar com falhas temporárias ou interrupções de rede. Defina procedimentos claros para recuperação e restauração de dados em caso de corrupção ou perda.

Ao projetar e implementar cuidadosamente mecanismos de fluxo e sincronização de dados, pode garantir que os seus workers de IA têm acesso a dados precisos, consistentes e atualizados. Isto permite que executem as suas tarefas de forma eficaz e produzam resultados fiáveis.

Gerir o Ciclo de Vida dos Workers de IA

Desenvolva um processo padronizado para inicializar e configurar workers de IA. Tenha preferência por frameworks que padronizam a forma como define configurações como nomes de modelos, diretivas do sistema e definições de funções. Certifique-se de que o processo de inicialização é automatizado e reproduzível para facilitar a implementação e a escalabilidade.

Implemente mecanismos abrangentes de monitorização e registo para acompanhar a saúde e o desempenho dos workers de IA. Recolha métricas como utilização de recursos, tempo de processamento, taxas de erro e throughput. Utilize sistemas de registo centralizados como ELK stack (Elasticsearch, Logstash, Kibana) para agregar e analisar registos de múltiplos workers de IA.

Integre tolerância a falhas e resiliência na arquitetura dos workers de IA. Implemente mecanismos de tratamento de erros e recuperação para lidar elegantemente com falhas

ou exceções. Os Modelos de Linguagem Grandes ainda são tecnologia de ponta; os fornecedores costumam ficar indisponíveis frequentemente em momentos inesperados. Use mecanismos de nova tentativa e circuit breakers para prevenir falhas em cascata.

Composição e Orquestração de Workers de IA

Uma das principais vantagens da arquitetura de workers de IA é a sua composibilidade, que permite combinar e orquestrar múltiplos workers de IA para resolver problemas complexos. Ao decompor uma tarefa maior em subtarefas menores e mais gerenciáveis, cada uma tratada por um worker de IA especializado, pode criar sistemas poderosos e flexíveis. Nesta secção, vamos explorar diferentes abordagens para compor e orquestrar “uma multidão” de workers de IA.

Encadeamento de Workers de IA para Fluxos de Trabalho Multi-Etapa

Em muitos cenários, uma tarefa complexa pode ser decomposta numa série de passos sequenciais, onde a saída de um worker de IA torna-se a entrada para o próximo. Este encadeamento de workers de IA cria um fluxo de trabalho ou pipeline multi-etapa. Cada worker de IA na cadeia concentra-se numa subtarefa específica, e o resultado final é o produto dos esforços combinados de todos os workers.

Vamos considerar um exemplo no contexto de uma aplicação Ruby on Rails para processamento de conteúdo gerado pelo utilizador. O fluxo de trabalho envolve os seguintes passos, que admitidamente são provavelmente demasiado simples para valer a pena decompor desta forma em casos de uso reais, mas tornam o exemplo mais fácil de entender:

1. **Limpeza de Texto:** Um worker de IA responsável por remover tags HTML, converter texto para minúsculas e lidar com a normalização Unicode.

2. **Deteção de Idioma:** Um worker de IA que identifica o idioma do texto limpo.
3. **Análise de Sentimento:** Um worker de IA que determina o sentimento (positivo, negativo ou neutro) do texto com base no idioma detetado.
4. **Categorização de Conteúdo:** Um worker de IA que classifica o texto em categorias predefinidas usando técnicas de processamento de linguagem natural.

Aqui está um exemplo muito simplificado de como pode encadear estes workers de IA usando Ruby:

```
1 class ContentProcessor
2   def initialize(text)
3     @text = text
4   end
5
6   def process
7     cleaned_text = TextCleanupWorker.new(@text).call
8     language = LanguageDetectionWorker.new(cleaned_text).call
9     sentiment = SentimentAnalysisWorker.new(cleaned_text, language).call
10    category = CategorizationWorker.new(cleaned_text, language).call
11
12    { cleaned_text:, language:, sentiment:, category: }
13  end
14 end
```

Neste exemplo, a classe ContentProcessor inicializa com o texto em bruto e encadeia os trabalhadores de IA no método process. Cada trabalhador de IA executa a sua tarefa específica e passa o resultado para o próximo trabalhador na cadeia. O resultado final é um hash contendo o texto limpo, o idioma detetado, o sentimento e a categoria do conteúdo.

Processamento Paralelo para Trabalhadores de IA Independentes

No exemplo anterior, os trabalhadores de IA estão encadeados sequencialmente, onde cada trabalhador processa o texto e passa o resultado para o próximo trabalhador. No

entanto, se tiver múltiplos trabalhadores de IA que podem operar independentemente sobre o mesmo input, pode otimizar o fluxo de trabalho processando-os em paralelo.

No cenário apresentado, uma vez que a limpeza do texto é realizada pelo `TextCleanupWorker`, o `LanguageDetectionWorker`, `SentimentAnalysisWorker` e `CategorizationWorker` podem todos processar o texto limpo independentemente. Ao executar estes trabalhadores em paralelo, pode potencialmente reduzir o tempo total de processamento e melhorar a eficiência do seu fluxo de trabalho.

Para alcançar o processamento paralelo em Ruby, pode aproveitar técnicas de concorrência como threads ou programação assíncrona. Aqui está um exemplo de como pode modificar a classe `ContentProcessor` para processar os três últimos trabalhadores em paralelo usando threads:

```
1  require 'concurrent'
2
3  class ContentProcessor
4    def initialize(text)
5      @text = text
6    end
7
8    def process
9      cleaned_text = TextCleanupWorker.new(@text).call
10
11      language_future = Concurrent::Future.execute do
12        LanguageDetectionWorker.new(cleaned_text).call
13      end
14
15      sentiment_future = Concurrent::Future.execute do
16        SentimentAnalysisWorker.new(cleaned_text).call
17      end
18
19      category_future = Concurrent::Future.execute do
20        CategorizationWorker.new(cleaned_text).call
21      end
22
23      language = language_future.value
24      sentiment = sentiment_future.value
25      category = category_future.value
```

```
26
27     { cleaned_text:, language:, sentiment:, category: }
28   end
29 end
```

Nesta versão otimizada, utilizamos a biblioteca `concurrent-ruby` para criar objetos `Concurrent::Future` para cada um dos workers de IA independentes. Um `Future` representa uma computação que será executada de forma assíncrona numa thread separada.

Após a etapa de limpeza do texto, criamos três objetos `Future`: `language_future`, `sentiment_future` e `category_future`. Cada `Future` executa o seu worker de IA correspondente (`LanguageDetectionWorker`, `SentimentAnalysisWorker` e `CategorizationWorker`) numa thread separada, passando o `cleaned_text` como entrada.

Ao chamar o método `value` em cada `Future`, aguardamos que a computação seja concluída e obtemos o resultado. O método `value` bloqueia até que o resultado esteja disponível, garantindo que todos os workers paralelos tenham terminado o processamento antes de prosseguir.

Por fim, construímos o hash de saída com o texto limpo e os resultados dos workers paralelos, tal como no exemplo original.

Ao processar os workers de IA independentes em paralelo, é possível reduzir potencialmente o tempo total de processamento em comparação com a execução sequencial. Esta otimização é particularmente benéfica quando se lida com tarefas demoradas ou quando se processa grandes volumes de dados.

No entanto, é importante notar que os ganhos reais de desempenho dependem de vários fatores, como a complexidade de cada worker, os recursos do sistema disponíveis e a sobrecarga da gestão de threads. É sempre uma boa prática fazer benchmarks e criar perfis do seu código para determinar o nível ideal de paralelismo para o seu caso específico.

Adicionalmente, ao implementar processamento paralelo, é preciso estar atento a quaisquer recursos compartilhados ou dependências entre os workers. Certifique-se de que os workers podem operar independentemente sem conflitos ou condições de corrida. Se houver dependências ou recursos compartilhados, poderá ser necessário implementar mecanismos de sincronização apropriados para manter a integridade dos dados e evitar problemas como deadlocks ou resultados inconsistentes.

O Global Interpreter Lock (GIL) do Ruby e o Processamento Assíncrono

É importante compreender as implicações do Global Interpreter Lock (GIL) do Ruby quando se considera o processamento assíncrono baseado em threads em Ruby.

O GIL é um mecanismo no interpretador Ruby que garante que apenas uma thread pode executar código Ruby de cada vez, mesmo em processadores multi-core. Isto significa que, embora múltiplas threads possam ser criadas e geridas dentro de um processo Ruby, apenas uma thread pode executar ativamente código Ruby em qualquer momento.

O GIL foi projetado para simplificar a implementação do interpretador Ruby e fornecer segurança de thread para as estruturas de dados internas do Ruby. No entanto, também limita o potencial de verdadeira execução paralela do código Ruby.

Quando se utilizam threads em Ruby, como com a biblioteca `concurrent-ruby` ou a classe `Thread` incorporada, as threads estão sujeitas às restrições do GIL. O GIL permite que cada thread execute código Ruby durante um curto período de tempo antes de mudar para outra thread, criando a ilusão de execução concorrente.

No entanto, devido ao GIL, a execução real do código Ruby permanece sequencial. Enquanto uma thread está a executar código Ruby, outras threads ficam essencialmente em pausa, aguardando a sua vez de adquirir o GIL e executar.

Isto significa que o processamento assíncrono baseado em threads em Ruby é mais eficaz para tarefas limitadas por E/S, como aguardar respostas de APIs externas (como modelos de linguagem grandes hospedados por terceiros) ou realizar operações de E/S de ficheiros. Quando uma thread encontra uma operação de E/S, pode libertar o GIL, permitindo que outras threads executem enquanto aguardam a conclusão da E/S.

Por outro lado, para tarefas limitadas pela CPU, como computações intensivas ou processamento de workers de IA de longa duração, o GIL pode limitar os potenciais ganhos de desempenho do paralelismo baseado em threads. Como apenas uma thread pode executar código Ruby de cada vez, o tempo total de execução pode não ser significativamente reduzido em comparação com o processamento sequencial.

Para alcançar uma verdadeira execução paralela para tarefas limitadas pela CPU em Ruby, pode ser necessário explorar abordagens alternativas, tais como:

- Utilizar paralelismo baseado em processos com múltiplos processos Ruby, cada um executando num núcleo de CPU separado.
- Aproveitar bibliotecas externas ou frameworks que forneçam extensões nativas ou interfaces para linguagens sem GIL, como C ou Rust,
- Utilizar frameworks de computação distribuída ou filas de mensagens para distribuir tarefas entre múltiplas máquinas ou processos.

É crucial considerar a natureza das suas tarefas e as limitações impostas pelo GIL ao projetar e implementar processamento assíncrono em Ruby. Embora o processamento assíncrono baseado em threads possa fornecer benefícios para tarefas limitadas por E/S, pode não oferecer melhorias significativas de desempenho para tarefas limitadas pela CPU devido às restrições do GIL.

Técnicas de Ensemble para Melhorar a Precisão

As técnicas de ensemble envolvem combinar as saídas de múltiplos workers de IA para melhorar a precisão geral ou a robustez do sistema. Em vez de depender de um único worker de IA, as técnicas de ensemble aproveitam a inteligência coletiva de múltiplos workers para tomar decisões mais informadas.



Os ensembles são especialmente importantes se diferentes partes do seu fluxo de trabalho funcionam melhor com diferentes modelos de IA, algo mais comum do que se possa pensar. Modelos poderosos como o GPT-4 são extremamente caros em comparação com opções de código aberto menos capazes, e provavelmente não são necessários para cada etapa do fluxo de trabalho da sua aplicação.

Uma técnica comum de ensemble é a votação por maioria, onde múltiplos trabalhadores de IA processam independentemente a mesma entrada, e a saída final é determinada pelo consenso da maioria. Esta abordagem pode ajudar a mitigar o impacto de erros individuais dos trabalhadores e melhorar a fiabilidade geral do sistema.

Consideremos um exemplo onde temos três trabalhadores de IA para análise de sentimento, cada um utilizando um modelo diferente ou fornecido com diferentes contextos. Podemos combinar as suas saídas usando votação por maioria para determinar a previsão final do sentimento.

```
1 class SentimentAnalysisEnsemble
2   def initialize(text)
3     @text = text
4   end
5
6   def analyze
7     predictions = [
8       SentimentAnalysisWorker1.new(@text).analyze,
9       SentimentAnalysisWorker2.new(@text).analyze,
10      SentimentAnalysisWorker3.new(@text).analyze
11    ]
12
13    predictions
14      .group_by { |sentiment| sentiment }
15      .max_by { |_, votes| votes.size }
16      .first
17
18    end
19  end
```

Neste exemplo, a classe `SentimentAnalysisEnsemble` é inicializada com o texto e invoca três diferentes trabalhadores de IA para análise de sentimento. O método `analyze` recolhe as previsões de cada trabalhador e determina o sentimento maioritário usando os métodos `group_by` e `max_by`. O resultado final é o sentimento que recebe mais votos do conjunto de trabalhadores



Os conjuntos são claramente um caso onde vale a pena experimentar com paralelismo.

Seleção Dinâmica e Invocação de Trabalhadores de IA

Em alguns, se não na maioria dos casos, o trabalhador de IA específico a ser invocado pode depender das condições em tempo de execução ou das entradas do utilizador. A seleção dinâmica e invocação de trabalhadores de IA permitem flexibilidade e adaptabilidade no sistema.



Pode sentir-se tentado a tentar incluir muita funcionalidade num único trabalhador de IA, dando-lhe várias funções e um prompt grande e complicado que explica como as chamar. Resista à tentação, acredite em mim. Uma das razões pela qual a abordagem que estamos a discutir neste capítulo se chama “Multiplicidade de Trabalhadores” é para nos lembrar que é desejável ter muitos trabalhadores especializados, cada um fazendo o seu pequeno trabalho ao serviço do propósito maior.

Por exemplo, considere uma aplicação de chatbot onde diferentes trabalhadores de IA são responsáveis por lidar com diferentes tipos de consultas do utilizador. Com base na entrada do utilizador, a aplicação seleciona dinamicamente o trabalhador de IA apropriado para processar a consulta.

```
1 class ChatbotController < ApplicationController
2   def process_query
3     query = params[:query]
4     query_type = QueryClassifierWorker.new(query).classify
5
6     case query_type
7     when 'greeting'
8       response = GreetingWorker.new(query).generate_response
9     when 'product_inquiry'
10      response = ProductInquiryWorker.new(query).generate_response
11    when 'order_status'
12      response = OrderStatusWorker.new(query).generate_response
13    else
14      response = DefaultResponseWorker.new(query).generate_response
15    end
16
17    render json: { response: response }
18  end
19 end
```

Neste exemplo, o ChatbotController recebe uma consulta do utilizador através da ação process_query. Primeiro, utiliza um QueryClassifierWorker para determinar o tipo de consulta. Com base no tipo de consulta classificada, o controlador

seleciona dinamicamente o trabalhador de IA apropriado para gerar a resposta. Esta seleção dinâmica permite que o chatbot processe diferentes tipos de consultas e as encaminhe para os trabalhadores de IA relevantes.



Como o trabalho do `QueryClassifierWorker` é relativamente simples e não requer muito contexto ou definições de funções, provavelmente pode implementá-lo usando um LLM pequeno e ultra-rápido como o `mistralai/mixtral-8x7b-instruct:nitro`. Tem capacidades que se aproximam do nível do GPT-4 em muitas tarefas e, no momento em que escrevo isto, a Groq consegue servi-lo com uma velocidade impressionante de 444 tokens por segundo.

Combinando PLN Tradicional com LLMs

Embora os Modelos de Linguagem Grandes (LLMs) tenham revolucionado o campo do processamento de linguagem natural (NLP), oferecendo versatilidade e desempenho sem precedentes numa ampla gama de tarefas, nem sempre são a solução mais eficiente ou económica para todos os problemas. Em muitos casos, combinar técnicas tradicionais de NLP com LLMs pode levar a abordagens mais otimizadas, direcionadas e económicas para resolver desafios específicos de NLP.

Pense nos LLMs como os canivetes suíços do NLP—incrivelmente versáteis e poderosos, mas não necessariamente a melhor ferramenta para cada trabalho. Por vezes, uma ferramenta dedicada como um saca-rolhas ou um abre-latas pode ser mais eficaz e eficiente para uma tarefa específica. Da mesma forma, técnicas tradicionais de NLP, como agrupamento de documentos, identificação de tópicos e classificação, podem frequentemente fornecer soluções mais direcionadas e económicas para certos aspectos do seu pipeline de NLP.

Uma das principais vantagens das técnicas tradicionais de NLP é a sua eficiência computacional. Estes métodos, que frequentemente dependem de modelos estatísticos

mais simples ou abordagens baseadas em regras, podem processar grandes volumes de dados textuais muito mais rapidamente e com menor sobrecarga computacional em comparação com os LLMs. Isto torna-os particularmente adequados para tarefas que envolvem a análise e organização de grandes corpora de documentos, como o agrupamento de artigos semelhantes ou a identificação de tópicos-chave dentro de uma coleção de textos.

Além disso, as técnicas tradicionais de NLP podem frequentemente alcançar alta precisão e exatidão para tarefas específicas, especialmente quando treinadas em conjuntos de dados específicos do domínio. Por exemplo, um classificador de documentos bem ajustado usando algoritmos tradicionais de aprendizagem automática como Máquinas de Vetores de Suporte (SVM) ou Naive Bayes pode categorizar documentos em categorias predefinidas com precisão e com custo computacional mínimo.

No entanto, os LLMs realmente brilham quando se trata de tarefas que requerem uma compreensão mais profunda da linguagem, contexto e raciocínio. A sua capacidade de gerar texto coerente e contextualmente relevante, responder a perguntas e resumir passagens longas é inigualável pelos métodos tradicionais de NLP. Os LLMs podem lidar eficazmente com fenômenos linguísticos complexos, como ambiguidade, correferência e expressões idiomáticas, tornando-os inestimáveis para tarefas que requerem geração ou compreensão de linguagem natural.

O verdadeiro poder está em combinar técnicas tradicionais de NLP com LLMs para criar abordagens híbridas que aproveitam os pontos fortes de ambos. Ao usar métodos tradicionais de NLP para tarefas como pré-processamento de documentos, agrupamento e extração de tópicos, pode organizar e estruturar eficientemente os seus dados textuais. Esta informação estruturada pode então ser alimentada aos LLMs para tarefas mais avançadas, como gerar resumos, responder a perguntas ou criar relatórios abrangentes.

Por exemplo, consideremos um caso de uso onde pretende gerar um relatório de tendências para um domínio específico com base num grande corpus de documentos

individuais de tendências. Em vez de confiar apenas em LLMs, que podem ser computacionalmente dispendiosos e demorados para processar grandes volumes de texto, pode empregar uma abordagem híbrida:

1. Usar técnicas tradicionais de NLP, como modelação de tópicos (por exemplo, Alocação Latente de Dirichlet) ou algoritmos de agrupamento (por exemplo, K-means), para agrupar documentos de tendências semelhantes e identificar temas e tópicos-chave dentro do corpus.
2. Alimentar os documentos agrupados e tópicos identificados num LLM, aproveitando as suas capacidades superiores de compreensão e geração de linguagem para criar resumos coerentes e informativos para cada grupo ou tópico.
3. Finalmente, usar o LLM para gerar um relatório abrangente de tendências, combinando os resumos individuais, destacando as tendências mais significativas e fornecendo insights e recomendações com base na informação agregada.

Ao combinar técnicas tradicionais de NLP com LLMs desta maneira, pode processar eficientemente grandes quantidades de dados textuais, extrair insights significativos e gerar relatórios de alta qualidade enquanto otimiza recursos computacionais e custos.

Ao embarcar nos seus projetos de NLP, é essencial avaliar cuidadosamente os requisitos específicos e as limitações de cada tarefa e considerar como os métodos tradicionais de PLN e os LLMs podem ser aproveitados em conjunto para alcançar os melhores resultados. Ao combinar a eficiência e precisão das técnicas tradicionais com a versatilidade e poder dos LLMs, pode criar soluções de PLN altamente eficazes e económicas que proporcionam valor aos seus utilizadores e partes interessadas.

Uso de Ferramentas



No domínio do desenvolvimento de aplicações orientadas por IA, o conceito de “uso de ferramentas” ou “chamada de funções” emergiu como uma técnica poderosa que permite ao seu LLM conectar-se a ferramentas externas, APIs, funções, bases de dados e outros recursos. Esta abordagem permite um conjunto mais rico de comportamentos do que apenas gerar texto, e interações mais dinâmicas entre os seus componentes de IA e o resto do ecossistema da sua aplicação. Como examinaremos neste capítulo, o uso de ferramentas também oferece a opção de fazer com que o seu modelo de IA gere dados de forma estruturada.

O que é o Uso de Ferramentas?

O uso de ferramentas, também conhecido como chamada de funções, é uma técnica que permite aos programadores especificar uma lista de funções com as quais um LLM pode

interagir durante o processo de geração. Estas ferramentas podem variar desde funções utilitárias simples até APIs complexas ou consultas a bases de dados. Ao fornecer ao LLM acesso a estas ferramentas, os programadores podem expandir as capacidades do modelo e permitir que ele execute tarefas que requerem conhecimento ou ações externas.

Figura 8. Exemplo de uma definição de função para um trabalhador de IA que analisa documentos

```
1  FUNCTION = {
2      name: "save_analysis",
3      description: "Save analysis data for document",
4      parameters: {
5          type: "object",
6          properties: {
7              title: {
8                  type: "string",
9                  maxLength: 140
10             },
11             summary: {
12                 type: "string",
13                 description: "comprehensive multi-paragraph summary with
14                             overview and list of sections (if applicable)"
15             },
16             tags: {
17                 type: "array",
18                 items: {
19                     type: "string",
20                     description: "lowercase tags representing main themes
21                                 of the document"
22                 }
23             }
24         },
25         "required": %w[title summary tags]
26     }
27 }.freeze
```

A ideia fundamental por detrás da utilização de ferramentas é dar ao LLM a capacidade de seleccionar e executar dinamicamente as ferramentas apropriadas com base na entrada do utilizador ou na tarefa em questão. Em vez de depender exclusivamente do conhecimento pré-treinado do modelo, a utilização de ferramentas permite ao

LLM aproveitar recursos externos para gerar respostas mais precisas, relevantes e acionáveis. A utilização de ferramentas torna técnicas como RAG (Geração Aumentada por Recuperação) muito mais fáceis de implementar do que seriam de outra forma.

Note que, salvo indicação em contrário, este livro assume que o seu modelo de IA não tem acesso a quaisquer ferramentas incorporadas do lado do servidor. Quaisquer ferramentas que pretenda disponibilizar à sua IA devem ser explicitamente declaradas por si em cada pedido à API, com provisões para despachar a sua execução se e quando a sua IA lhe indicar que gostaria de usar essa ferramenta na sua resposta.

O Potencial da Utilização de Ferramentas

A utilização de ferramentas abre um vasto leque de possibilidades para aplicações baseadas em IA. Aqui estão alguns exemplos do que pode ser alcançado com a utilização de ferramentas:

1. **Chatbots e Assistentes Virtuais:** Ao conectar um LLM a ferramentas externas, os chatbots e assistentes virtuais podem realizar tarefas mais complexas, como recuperar informações de bases de dados, executar chamadas de API ou interagir com outros sistemas. Por exemplo, um chatbot poderia usar uma ferramenta de CRM para alterar o estado de um negócio com base no pedido do utilizador.
2. **Análise de Dados e Insights:** Os LLMs podem ser conectados a ferramentas ou bibliotecas de análise de dados para realizar tarefas avançadas de processamento de dados. Isto permite que as aplicações gerem insights, realizem análises comparativas ou forneçam recomendações baseadas em dados em resposta a consultas dos utilizadores.
3. **Pesquisa e Recuperação de Informação:** A utilização de ferramentas permite que os LLMs interajam com motores de busca, bases de dados vetoriais ou

outros sistemas de recuperação de informação. Ao transformar as consultas dos utilizadores em consultas de pesquisa, o LLM pode recuperar informações relevantes de múltiplas fontes e fornecer respostas abrangentes às questões dos utilizadores.

4. **Integração com Serviços Externos:** A utilização de ferramentas permite a integração perfeita entre aplicações baseadas em IA e serviços externos ou APIs. Por exemplo, um LLM poderia interagir com uma API de meteorologia para fornecer atualizações meteorológicas em tempo real ou com uma API de tradução para gerar respostas multilingues.

O Fluxo de Trabalho da Utilização de Ferramentas

O fluxo de trabalho da utilização de ferramentas normalmente envolve quatro passos principais:

1. Incluir definições de funções no contexto do seu pedido
2. Seleção dinâmica (ou explícita) de ferramentas
3. Execução da(s) função(ões)
4. Continuação opcional do prompt original

Vamos analisar cada um destes passos em detalhe.

Incluir definições de funções no contexto do seu pedido

A IA sabe quais as ferramentas que tem à sua disposição porque você fornece uma lista como parte do seu pedido de conclusão (tipicamente definida como funções usando uma variante do esquema JSON).

A sintaxe exata da definição de ferramentas é específica do modelo.

É assim que se define uma função `get_weather` no Claude 3:

```
1  {
2    "name": "get_weather",
3    "description": "Get the current weather in a given location",
4    "input_schema": {
5      "type": "object",
6      "properties": {
7        "location": {
8          "type": "string",
9          "description": "The city and state, e.g. San Francisco, CA"
10       },
11       "unit": {
12         "type": "string",
13         "enum": ["celsius", "fahrenheit"],
14         "description": "The unit of temperature"
15       }
16     },
17     "required": ["location"]
18   }
19 }
```

E é assim que definiria a mesma função para GPT-4, passando-a como valor do parâmetro tools:

```
1  {
2    "name": "get_current_weather",
3    "description": "Get the current weather in a given location",
4    "parameters": {
5      "type": "object",
6      "properties": {
7        "location": {
8          "type": "string",
9          "description": "The city and state, e.g. San Francisco, CA",
10       },
11       "unit": {
12         "type": "string",
13         "enum": ["celsius", "fahrenheit"],
14         "description": "The unit of temperature"
15       }
16     },
17     "required": ["location"],
```

```
18     },  
19 }
```

Quase igual, exceto diferente sem razão aparente! Que irritante.

As definições de função especificam nome, descrição e parâmetros de entrada. Os parâmetros de entrada podem ser definidos mais detalhadamente usando atributos como enums para limitar os valores aceitáveis, e especificando se um parâmetro é obrigatório ou não.

Além das definições de função propriamente ditas, também pode incluir instruções ou contexto sobre por que e como usar a função na diretiva do sistema.

Por exemplo, a minha ferramenta de Web Search no Olympia inclui esta diretiva do sistema, que relembra à IA que tem as ferramentas mencionadas à sua disposição:

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

Fornecer descrições detalhadas é considerado o fator mais importante no desempenho das ferramentas. As suas descrições devem explicar todos os detalhes sobre a ferramenta, incluindo:

- O que a ferramenta faz
- Quando deve ser utilizada (e quando não deve)
- O que significa cada parâmetro e como afeta o comportamento da ferramenta
- Quaisquer ressalvas ou limitações importantes que se apliquem à implementação da ferramenta

Quanto mais contexto puder fornecer à IA sobre as suas ferramentas, melhor será a sua capacidade de decidir quando e como utilizá-las. Por exemplo, a Anthropic recomenda pelo menos 3-4 frases por descrição de ferramenta para a sua série Claude 3, ou mais se a ferramenta for complexa.

Pode não ser intuitivo, mas as descrições são também consideradas mais importantes do que os exemplos. Embora possa incluir exemplos de como utilizar uma ferramenta na sua descrição ou no prompt que a acompanha, isto é menos importante do que ter uma explicação clara e abrangente do propósito e parâmetros da ferramenta. Adicione exemplos apenas depois de ter desenvolvido completamente a descrição.

Eis um exemplo de uma especificação de função API ao estilo da Stripe:

```
1  {
2    "name": "createPayment",
3    "description": "Create a new payment request",
4    "parameters": {
5      "type": "object",
6      "properties": {
7        "transaction_amount": {
8          "type": "number",
9          "description": "The amount to be paid"
10       },
11       "description": {
12         "type": "string",
13         "description": "A brief description of the payment"
14       },
15       "payment_method_id": {
16         "type": "string",
17         "description": "The payment method to be used"
18       },
19       "payer": {
20         "type": "object",
21         "description": "Information about the payer, including their name,
22                        email, and identification number",
23         "properties": {
24           "name": {
25             "type": "string",
26             "description": "The payer's name"
```

```

27     },
28     "email": {
29         "type": "string",
30         "description": "The payer's email address"
31     },
32     "identification": {
33         "type": "object",
34         "description": "The payer's identification number",
35         "properties": {
36             "type": {
37                 "type": "string",
38                 "description": "Identification document (e.g. CPF, CNPJ)"
39             },
40             "number": {
41                 "type": "string",
42                 "description": "The identification number"
43             }
44         },
45         "required": [ "type", "number" ]
46     }
47 },
48 "required": [ "name", "email", "identification" ]
49 }
50 }
51 }

```



Na prática, alguns modelos têm dificuldade em lidar com especificações de funções aninhadas e com tipos de dados de saída complexos como arrays, dicionários etc. Mas em teoria, deveria ser possível fornecer especificações de Schema JSON de profundidade arbitrária!

Seleção Dinâmica de Ferramentas

Quando executa uma conclusão de chat que inclui definições de ferramentas, o LLM seleciona dinamicamente a(s) ferramenta(s) mais apropriada(s) para usar e gera os parâmetros de entrada necessários para cada ferramenta.

Na prática, a capacidade da IA para chamar *exatamente* a função correta e seguir *exatamente* a sua especificação para as entradas é incerta. Reduzir o hiperparâmetro de temperatura para 0.0 ajuda bastante, mas pela minha experiência ainda terá erros ocasionais. Essas falhas incluem nomes de funções alucinados, parâmetros de entrada mal nomeados ou simplesmente ausentes. Os parâmetros são passados como JSON, o que significa que às vezes verá erros causados por JSON truncado, mal citado ou defeituoso de outras formas.



Os padrões de [Dados Auto-Reparáveis](#) podem ajudar a [corrigir automaticamente](#) chamadas de função que falham devido a erros de sintaxe.

Seleção Forçada (ou Explícita) de Ferramentas

Alguns modelos dão-lhe a opção de forçar a chamada de uma função específica, como um parâmetro no pedido. Caso contrário, chamar ou não a função fica inteiramente ao critério da IA.

A capacidade de forçar uma chamada de função é crucial em certos cenários onde pretende garantir que uma ferramenta ou função específica é executada, independentemente do processo de seleção dinâmica da IA. Existem várias razões pelas quais esta capacidade é importante:

1. **Controlo Explícito:** Pode estar a usar a IA como um *Componente Discreto* ou num fluxo de trabalho predefinido que necessita da execução de uma função específica num momento específico. Ao forçar a chamada, pode garantir que a função desejada é invocada em vez de ter de pedir educadamente à IA para o fazer.
2. **Depuração e Testes:** Durante o desenvolvimento e teste de aplicações baseadas em IA, a capacidade de forçar chamadas de função é inestimável para fins de

depuração. Ao acionar explicitamente funções específicas, pode isolar e testar componentes individuais da sua aplicação. Isto permite-lhe verificar a correção das implementações das funções, validar os parâmetros de entrada e garantir que os resultados esperados são devolvidos.

3. **Tratamento de Casos Extremos:** Podem existir casos extremos ou cenários excepcionais onde o processo de seleção dinâmica da IA pode não escolher executar uma função que deveria, e você sabe disso com base em processos externos. Nesses casos, ter a capacidade de forçar uma chamada de função permite-lhe lidar com estas situações explicitamente. Defina regras ou condições na lógica da sua aplicação para determinar quando substituir o critério da IA.
4. **Consistência e Reprodutibilidade:** Se tiver uma sequência específica de funções que precisam de ser executadas numa ordem particular, forçar as chamadas garante que a mesma sequência é seguida todas as vezes. Isto é especialmente importante em aplicações onde a consistência e o comportamento previsível são críticos, como em sistemas financeiros ou simulações científicas.
5. **Otimização de Desempenho:** Em alguns casos, forçar uma chamada de função pode levar a otimizações de desempenho. Se souber que uma função específica é necessária para uma tarefa particular e que o processo de seleção dinâmica da IA pode introduzir sobrecarga desnecessária, pode contornar o processo de seleção e invocar diretamente a função necessária. Isto pode ajudar a reduzir a latência e melhorar a eficiência geral da sua aplicação.

Em resumo, a capacidade de forçar chamadas de função em aplicações baseadas em IA fornece controlo explícito, auxilia na depuração e testes, lida com casos extremos, garante consistência e reprodutibilidade. É uma ferramenta poderosa no seu arsenal, mas precisamos de discutir mais um aspeto desta importante funcionalidade.



Em muitos casos de uso de tomada de decisão, queremos sempre que o modelo faça uma chamada de função e podemos nunca querer que o modelo responda apenas com o seu conhecimento interno. Por exemplo, se estiver a encaminhar entre vários modelos especializados em diferentes tarefas (entrada multilíngue, matemática, etc.), pode usar o modelo de chamada de função para delegar pedidos a um dos modelos auxiliares e nunca responder independentemente.

Parâmetro de Escolha de Ferramenta

O GPT-4 e outros modelos de linguagem que suportam chamadas de função fornecem um parâmetro `tool_choice` para controlar se o uso de ferramentas é necessário como parte de uma conclusão. Este parâmetro tem três valores possíveis:

- `auto` dá à IA total discricção sobre usar uma ferramenta ou simplesmente responder
- `required` diz à IA que ela *deve* chamar uma ferramenta *em vez* de responder, mas deixa a seleção da ferramenta a critério da IA
- A terceira opção é definir o parâmetro do `name_of_function` que pretende forçar. Mais sobre isso na próxima secção.



Note que se definir `tool choice` como `required`, o modelo será forçado a escolher a função mais relevante para chamar entre as fornecidas, mesmo que nenhuma se adegue realmente ao prompt. No momento da publicação, não tenho conhecimento de nenhum modelo que retorne uma resposta `tool_calls` vazia, ou use alguma outra forma de informar que não encontrou uma função adequada para chamar.

Forçando uma Função Para Obter Saída Estruturada

A capacidade de forçar uma chamada de função oferece uma maneira de forçar dados estruturados de uma conclusão de chat em vez de ter que extraí-los você mesmo da resposta em texto simples.

Por que forçar funções para obter saída estruturada é tão importante? Simplesmente porque a extração de dados estruturados da saída do LLM é uma dor de cabeça. Você pode facilitar um pouco sua vida pedindo dados em XML, mas depois tem que analisar o XML. E o que fazer quando esse XML está ausente porque sua IA respondeu: “Lamento, mas não consigo gerar os dados que solicitou porque blá, blá, blá...”

Ao usar ferramentas desta forma:

- Você provavelmente deve definir uma única ferramenta no seu pedido
- Lembre-se de forçar o uso da sua função usando o parâmetro `tool_choice`.
- Lembre-se que o modelo passará a entrada para a ferramenta, então o nome da ferramenta e a descrição devem ser da perspectiva do modelo, não da sua.

Este último ponto merece um exemplo para maior clareza. Digamos que você está pedindo à IA para fazer análise de sentimento em texto do utilizador. O nome da função não seria `analyze_sentiment`, mas sim algo como `save_sentiment_analysis`. A IA é que está fazendo a análise de sentimento, *não a ferramenta*. Tudo o que a ferramenta está fazendo (da perspectiva da IA) é guardar os resultados da análise.

Aqui está um exemplo de uso do Claude 3 para registrar um resumo de uma imagem em JSON bem estruturado, desta vez pela linha de comando usando `curl`:

```

1  curl https://api.anthropic.com/v1/messages \
2      --header "content-type: application/json" \
3      --header "x-api-key: $ANTHROPIC_API_KEY" \
4      --header "anthropic-version: 2023-06-01" \
5      --header "anthropic-beta: tools-2024-04-04" \
6      --data \
7      '{
8          "model": "claude-3-sonnet-20240229",
9          "max_tokens": 1024,
10         "tools": [{
11             "name": "record_summary",
12             "description": "Record summary of image into well-structured JSON.",
13             "input_schema": {
14                 "type": "object",
15                 "properties": {
16                     "key_colors": {
17                         "type": "array",
18                         "items": {
19                             "type": "object",
20                             "properties": {
21                                 "r": {
22                                     "type": "number",
23                                     "description": "red value [0.0, 1.0]"
24                                 },
25                                 "g": {
26                                     "type": "number",
27                                     "description": "green value [0.0, 1.0]"
28                                 },
29                                 "b": {
30                                     "type": "number",
31                                     "description": "blue value [0.0, 1.0]"
32                                 },
33                                 "name": {
34                                     "type": "string",
35                                     "description": "Human-readable color name
36                                         in snake_case, e.g.
37                                         \"olive_green\"or
38                                         \"turquoise\""
39                                 }
40                             },
41                             "required": [ "r", "g", "b", "name" ]
42                         },

```

```

43         "description": "Key colors in the image. Four or less."
44     },
45     "description": {
46         "type": "string",
47         "description": "Image description. 1-2 sentences max."
48     },
49     "estimated_year": {
50         "type": "integer",
51         "description": "Estimated year that the image was taken,
52                         if is it a photo. Only set this if the
53                         image appears to be non-fictional.
54                         Rough estimates are okay!"
55     }
56 },
57 "required": [ "key_colors", "description" ]
58 }
59 ]],
60 "messages": [
61     {
62         "role": "user",
63         "content": [
64             {
65                 "type": "image",
66                 "source": {
67                     "type": "base64",
68                     "media_type": "'$IMAGE_MEDIA_TYPE'",
69                     "data": "'$IMAGE_BASE64'"
70                 }
71             },
72             {
73                 "type": "text",
74                 "text": "Use `record_summary` to describe this image."
75             }
76         ]
77     }
78 ]
79 }'

```

No exemplo fornecido, estamos a usar o modelo Claude 3 da Anthropic para gerar um resumo JSON estruturado de uma imagem. Eis como funciona:

1. Definimos uma única ferramenta chamada `record_summary` no array `tools` da carga útil do pedido. Esta ferramenta é responsável por registrar um resumo da imagem num JSON bem estruturado.
2. A ferramenta `record_summary` tem um `input_schema` que especifica a estrutura esperada do output JSON. Define três propriedades:
 - `key_colors`: Um array de objetos que representam as cores principais na imagem. Cada objeto de cor tem propriedades para os valores de vermelho, verde e azul (variando de 0.0 a 1.0) e um nome de cor legível por humanos em formato `snake_case`.
 - `description`: Uma propriedade string para uma breve descrição da imagem, limitada a 1-2 frases.
 - `estimated_year`: Uma propriedade inteira opcional para o ano estimado em que a imagem foi tirada, caso pareça ser uma foto não ficcional.
3. No array `messages`, fornecemos os dados da imagem como uma string codificada em base64 juntamente com o tipo de média. Isto permite que o modelo processe a imagem como parte do input.
4. Também instruímos o Claude a usar a ferramenta `record_summary` para descrever a imagem.
5. Quando o pedido é enviado ao modelo Claude 3, este analisa a imagem e gera um resumo JSON baseado no `input_schema` especificado. O modelo extrai as cores principais, fornece uma breve descrição e estima o ano em que a imagem foi tirada (se aplicável).
6. O resumo JSON gerado é passado como parâmetros para a ferramenta `record_summary`, fornecendo uma representação estruturada das características principais da imagem.

Ao usar a ferramenta `record_summary` com um `input_schema` bem definido, podemos obter um resumo JSON estruturado de uma imagem sem depender de

extração de texto simples. Esta abordagem garante que o output segue um formato consistente e pode ser facilmente analisado e processado pelos componentes posteriores da aplicação.

A capacidade de forçar uma chamada de função e especificar a estrutura de output esperada é uma funcionalidade poderosa do uso de ferramentas em aplicações baseadas em IA. Permite aos programadores ter mais controlo sobre o output gerado e simplifica a integração de dados gerados por IA no fluxo de trabalho da sua aplicação.

Execução de Função(ões)

Definiu funções e deu instruções à sua IA, que decidiu que deveria chamar uma das suas funções. Agora é altura do código da sua aplicação ou biblioteca, se estiver a usar uma gem Ruby como **raix-rails**, despachar a chamada de função e os seus parâmetros para a implementação correspondente *no código da sua aplicação*.

O código da sua aplicação decide o que fazer com os resultados da execução da função. Talvez o que fazer envolva uma única linha de código num lambda, ou talvez envolva chamar uma API externa. Talvez envolva chamar outro componente de IA, ou talvez envolva centenas ou até milhares de linhas de código no resto do seu sistema. Depende inteiramente de si.

Por vezes, a chamada de função é o fim da operação, mas se os resultados representarem informação numa cadeia de pensamento a ser continuada pela IA, então o código da sua aplicação precisa de inserir os resultados da execução na transcrição do chat e deixar a IA continuar o processamento.

Por exemplo, aqui está uma declaração de função **Raix** usada pelo AccountManager da Olympia para comunicar com os nossos clientes como parte de uma Orquestração de Fluxo de Trabalho Inteligente para atendimento ao cliente.

```
1  class AccountManager
2    include Raix::ChatCompletion
3    include Raix::FunctionDispatch
4
5    # lots of other functions...
6
7    function :notify_account_owner,
8      "Don't share UUID. Mention dollars if subscription changed",
9      message: { type: "string" } do |arguments|
10      account.owner.freeform_notify(
11        subject: "Account Change Notification",
12        message: arguments[:message]
13      )
14      "Notified account owner"
15    end
```

Pode não ser imediatamente claro o que está a acontecer aqui, por isso vou explicar em detalhe.

1. A classe `AccountManager` define várias funções relacionadas com a gestão de contas. Pode alterar o seu plano, adicionar e remover membros da equipa, entre outras coisas.
2. As suas instruções de alto nível indicam ao `AccountManager` que deve notificar o proprietário da conta com os resultados do pedido de alteração da conta, utilizando a função `notify_account_owner`.
3. A definição concisa da função inclui:
 - nome
 - descrição
 - parâmetros `message: { type: "string" }`
 - um bloco para executar quando a função é chamada

Após atualizar a transcrição com os resultados do bloco de função, o método `chat_completion` é chamado novamente. Este método é responsável por enviar

a transcrição atualizada da conversa de volta ao modelo de IA para processamento adicional. Referimo-nos a este processo como um *ciclo de conversação*.

Quando o modelo de IA recebe um novo pedido de conclusão de chat com uma transcrição atualizada, tem acesso aos resultados da função previamente executada. Pode analisar estes resultados, incorporá-los no seu processo de tomada de decisão e gerar a próxima resposta ou ação com base no contexto cumulativo da conversa. Pode optar por executar funções adicionais com base no contexto atualizado, ou pode gerar uma resposta final ao pedido original se determinar que não são necessárias mais chamadas de função.

Continuação Opcional do Pedido Original

Quando envia os resultados da ferramenta de volta ao MLG e continua o processamento do pedido original, a IA utiliza esses resultados para chamar funções adicionais ou gerar uma resposta final em texto simples.



Alguns modelos como o [Command-R](#) da Cohere podem citar as ferramentas específicas que utilizaram nas suas respostas, proporcionando transparência e rastreabilidade adicionais.

Dependendo do modelo em uso, os resultados da chamada da função irão residir em mensagens da transcrição que têm o seu próprio papel especial ou são refletidos noutra sintaxe. Mas a parte importante é que esses dados estejam na transcrição, para que possam ser considerados pela IA ao decidir o que fazer a seguir.



Um erro comum (e potencialmente dispendioso) é esquecer de adicionar os resultados da função à transcrição antes de continuar o chat. Como resultado, a IA será solicitada essencialmente da mesma forma que foi antes de chamar a função pela primeira vez. Por outras palavras, do ponto de vista da IA, ainda não chamou a função. Então chama-a novamente. E novamente. E novamente, para sempre até que a interrompa. Espero que o seu contexto não fosse muito grande e que o seu modelo não fosse muito dispendioso!

Melhores Práticas para o Uso de Ferramentas

Para tirar o máximo partido do uso de ferramentas, considere as seguintes melhores práticas.

Definições Descritivas

Forneça nomes e descrições claros e descritivos para cada ferramenta e seus parâmetros de entrada. Isto ajuda o MLG a compreender melhor o propósito e as capacidades de cada ferramenta.

Posso dizer-vos por experiência que a sabedoria comum que diz que “dar nomes é difícil” aplica-se aqui; já vi resultados drasticamente diferentes de MLGs apenas mudando os nomes das funções ou a formulação das descrições. Por vezes, remover descrições *melhora* o desempenho.

Processamento dos Resultados das Ferramentas

Ao passar os resultados das ferramentas de volta ao MLG, certifique-se de que estão bem estruturados e são abrangentes. Utilize chaves e valores significativos para representar a saída de cada ferramenta. Experimente diferentes formatos e veja qual funciona melhor, desde JSON até texto simples.

O [Interpretador de Resultados](#) aborda este desafio empregando IA para analisar os resultados e fornecer explicações, resumos ou principais conclusões de forma amigável para humanos.

Tratamento de Erros

Implemente mecanismos robustos de tratamento de erros para lidar com casos em que o MLG possa gerar parâmetros de entrada inválidos ou não suportados para chamadas de ferramentas. Trate e recupere graciosamente de quaisquer erros que possam ocorrer durante a execução da ferramenta.

Uma qualidade extremamente boa da IA é que ela compreende mensagens de erro! O que significa que se estiver a trabalhar com uma mentalidade rápida e simples, pode simplesmente capturar quaisquer exceções geradas na implementação de uma ferramenta e passá-las de volta à IA para que ela saiba o que aconteceu!

Por exemplo, aqui está uma versão simplificada da implementação da pesquisa Google na Olympia:

```
1  def google_search(conversation, params)
2      conversation.update_cstatus("Searching Google...")
3      query = params[:query]
4      search = GoogleSearch.new(query).get_hash
5
6      conversation.update_cstatus("Summarizing results...")
7      SummarizeKnowledgeGraph.new.perform(conversation, search.to_json)
8  rescue StandardError => e
9      Honeybadger.notify(e)
10     { error: e.message }.inspect
11 end
```

As pesquisas Google na Olympia são um processo de dois passos. Primeiro faz-se a pesquisa, depois resumem-se os resultados. Se houver uma falha, independentemente do que seja, a mensagem de exceção é empacotada e enviada de volta para a IA. Esta técnica é a base de praticamente todos os padrões de *Tratamento Inteligente de Erros*

Por exemplo, digamos que a chamada à API `GoogleSearch` falha devido a uma exceção `503 Service Unavailable`. Isso sobe até ao nível superior do `rescue`, e a descrição do erro é enviada de volta para a IA como resultado da chamada da função. Em vez de apenas mostrar um ecrã em branco ou um erro técnico ao utilizador, a IA diz algo como “Lamento, mas não consigo aceder às minhas capacidades de Pesquisa Google neste momento. Posso tentar novamente mais tarde, se desejar.”

Isto pode parecer apenas um truque inteligente, mas considere um tipo diferente de erro, um onde a IA estava a chamar uma API externa e tinha controlo direto sobre os parâmetros a passar para a API. Talvez tenha cometido um erro na forma como gerou esses parâmetros? Desde que a mensagem de erro da API externa seja suficientemente detalhada, passar a mensagem de erro de volta para a IA que fez a chamada significa que ela pode reconsiderar esses parâmetros e tentar novamente. Automaticamente. Independentemente do erro que tenha ocorrido.

Agora pense no que seria necessário para replicar esse tipo de tratamento robusto de erros em código *normal*. É praticamente impossível.

Refinamento Iterativo

Se o LLM não estiver a recomendar as ferramentas apropriadas ou estiver a gerar respostas subótimas, itere nas definições das ferramentas, descrições e parâmetros de entrada. Continue a refinar e melhorar a configuração das ferramentas com base no comportamento observado e nos resultados desejados.

1. Comece com definições simples de ferramentas: Comece por definir ferramentas com nomes, descrições e parâmetros de entrada claros e concisos. Evite complicar excessivamente a configuração da ferramenta inicialmente e concentre-se na funcionalidade principal. Por exemplo, se quiser guardar os resultados da análise de sentimento, comece com uma definição básica como:

```
1  {
2    "name": "save_sentiment_score",
3    "description": "Analyze user-provided text and generate sentiment score",
4    "parameters": {
5      "type": "object",
6      "properties": {
7        "score": {
8          "type": "float",
9          "description": "sentiment score from -1 (negative) to 1 (positive)"
10       }
11     },
12     "required": ["score"]
13   }
14 }
```

2. Testar e observar: Depois de ter as definições iniciais das ferramentas estabelecidas, teste-as com diferentes prompts e observe como o LLM interage com a ferramenta. Preste atenção à qualidade e relevância das respostas geradas. Se o LLM estiver a gerar respostas subótimas, é altura de refinar as definições das ferramentas.

3. Refinar descrições: Se o LLM estiver a interpretar mal o propósito de uma ferramenta, tente refinar a descrição da ferramenta. Forneça mais contexto, exemplos ou esclarecimentos para orientar o LLM na utilização eficaz da ferramenta. Por exemplo, pode atualizar a descrição da ferramenta de análise de sentimentos para abordar mais especificamente o *tom emocional* do texto que está a ser analisado:

```
1 {  
2   "name": "save_sentiment_score",  
3   "description": "Determine the overall emotional tone of a piece of text,  
4     such as customer reviews, social media posts, or feedback comments.",  
5   ...  
6 }
```

4. Ajustar parâmetros de entrada: Se o LLM estiver a gerar parâmetros de entrada inválidos ou irrelevantes para uma ferramenta, considere ajustar as definições dos parâmetros. Adicione restrições mais específicas, regras de validação ou exemplos para clarificar o formato de entrada esperado.
5. Iterar com base no feedback: Monitorize continuamente o desempenho das suas ferramentas e recolha feedback dos utilizadores ou das partes interessadas. Use este feedback para identificar áreas de melhoria e fazer refinamentos iterativos nas definições da ferramenta. Por exemplo, se os utilizadores reportarem que a análise não está a lidar bem com o sarcasmo, pode adicionar uma nota na descrição:

```
1 {  
2   "name": "save_sentiment_score",  
3   "description": "Analyze the sentiment of a given text and return a sentiment  
4     score between -1 (negative) and 1 (positive). Note: Sarcasm should be  
5     considered negative.",  
6   ...  
7 }
```

Ao refinar iterativamente as definições das suas ferramentas com base no comportamento observado e no feedback, pode melhorar gradualmente o desempenho e a eficácia da sua aplicação baseada em IA. Lembre-se de manter as definições das ferramentas claras, concisas e focadas na tarefa específica em questão. Teste e valide regularmente as interações das ferramentas para garantir que estão alinhadas com os resultados desejados.

Composição e Encadeamento de Ferramentas

Um dos aspectos mais poderosos do uso de ferramentas que tem sido apenas mencionado até agora é a capacidade de compor e encadear múltiplas ferramentas para realizar tarefas complexas. Ao desenhar cuidadosamente as definições das suas ferramentas e os seus formatos de entrada/saída, pode criar blocos de construção reutilizáveis que podem ser combinados de várias formas.

Vamos considerar um exemplo onde está a construir um pipeline de dados para a sua aplicação baseada em IA. Poderá ter as seguintes ferramentas:

1. **DataRetrieval**: Uma ferramenta que obtém dados de uma base de dados ou API com base em critérios especificados.
2. **DataProcessing**: Uma ferramenta que realiza cálculos, transformações ou agregações nos dados obtidos.
3. **DataVisualization**: Uma ferramenta que apresenta os dados processados num formato amigável para o utilizador, como gráficos ou diagramas.

Ao encadear estas ferramentas, pode criar um fluxo de trabalho poderoso que obtém dados relevantes, processa-os e apresenta os resultados de forma significativa. Eis como o fluxo de trabalho do uso de ferramentas pode parecer:

1. O LLM recebe uma consulta do utilizador pedindo informações sobre dados de vendas para uma categoria específica de produtos.
2. O LLM seleciona a ferramenta `DataRetrieval` e gera os parâmetros de entrada apropriados para obter os dados de vendas relevantes da base de dados.
3. Os dados obtidos são “passados” para a ferramenta `DataProcessing`, que calcula métricas como receita total, preço médio de venda e taxa de crescimento.
4. Os dados processados são então digeridos pela ferramenta `DataVisualization`, que cria um gráfico ou diagrama visualmente apelativo para representar as informações, devolvendo o URL do gráfico ao LLM.
5. Finalmente, o LLM gera uma resposta formatada à consulta do utilizador usando markdown, incorporando os dados visualizados e fornecendo um resumo das principais conclusões.

Ao compor estas ferramentas em conjunto, pode criar um fluxo de trabalho de análise de dados contínuo que pode ser facilmente integrado na sua aplicação. A beleza desta abordagem é que cada ferramenta pode ser desenvolvida e testada independentemente, e depois combinada de diferentes formas para resolver vários problemas.

Para permitir uma composição e encadeamento suave das ferramentas, é importante definir formatos claros de entrada e saída para cada ferramenta.

Por exemplo, a ferramenta `DataRetrieval` pode aceitar parâmetros como os detalhes de conexão à base de dados, nome da tabela e condições de consulta, e retornar o conjunto de resultados como um objeto JSON estruturado. A ferramenta `DataProcessing` pode então esperar este objeto JSON como entrada e produzir um objeto JSON transformado como saída. Ao padronizar o fluxo de dados entre ferramentas, pode garantir compatibilidade e reutilização.

À medida que desenha o seu ecossistema de ferramentas, pense em como diferentes ferramentas podem ser combinadas para abordar casos de uso comuns na sua aplicação. Considere criar ferramentas de alto nível que encapsulem fluxos de trabalho comuns ou lógica de negócio, tornando mais fácil para o LLM selecioná-las e usá-las efetivamente.

Lembre-se, o poder do uso de ferramentas reside na flexibilidade e modularidade que proporciona. Ao decompor tarefas complexas em ferramentas mais pequenas e reutilizáveis, pode criar uma aplicação baseada em IA robusta e adaptável que pode enfrentar uma ampla gama de desafios.

Direções Futuras

À medida que o campo do desenvolvimento de aplicações baseadas em IA evolui, podemos esperar mais avanços nas capacidades de uso de ferramentas. Algumas direções futuras potenciais incluem:

1. **Uso de Ferramentas Multi-hop:** Os LLMs podem ser capazes de decidir quantas vezes precisam de usar ferramentas para gerar uma resposta satisfatória. Isto pode envolver múltiplas rondas de seleção e execução de ferramentas com base em resultados intermediários.
2. **Ferramentas Pré-definidas:** As plataformas de IA podem fornecer um conjunto de ferramentas pré-definidas que os programadores podem utilizar prontas a usar, como interpretadores Python, ferramentas de pesquisa web ou funções utilitárias comuns.
3. **Integração Perfeita:** À medida que o uso de ferramentas se torna mais prevalente, podemos esperar uma melhor integração entre plataformas de IA e frameworks de desenvolvimento populares, tornando mais fácil para os programadores incorporar o uso de ferramentas nas suas aplicações.

O uso de ferramentas é uma técnica poderosa que permite aos programadores aproveitar todo o potencial dos LLMs em aplicações baseadas em IA. Ao conectar LLMs a ferramentas e recursos externos, pode criar sistemas mais dinâmicos, inteligentes e conscientes do contexto que podem adaptar-se às necessidades dos utilizadores e fornecer informações e ações valiosas.

Embora o uso de ferramentas ofereça possibilidades imensas, é importante estar ciente dos potenciais desafios e considerações. Um aspecto fundamental é gerir a complexidade das interações das ferramentas e garantir a estabilidade e fiabilidade do sistema como um todo. Precisa de lidar com cenários onde as chamadas das ferramentas podem falhar, retornar resultados inesperados ou ter implicações de desempenho. Adicionalmente, deve considerar medidas de segurança e controlo de acesso para prevenir o uso não autorizado ou malicioso das ferramentas. Mecanismos adequados de tratamento de erros, registo e monitorização são cruciais para manter a integridade e o desempenho da sua aplicação baseada em IA.

À medida que explora as possibilidades do uso de ferramentas nos seus próprios projetos, lembre-se de começar com objetivos claros, criar definições de ferramentas bem estruturadas e iterar com base no retorno e nos resultados. Com a abordagem e mentalidade certas, o uso de ferramentas pode desbloquear novos níveis de inovação e valor nas suas aplicações baseadas em IA.

Processamento de Streams



O streaming de dados através de HTTP, também conhecido como eventos enviados pelo servidor (SSE), é um mecanismo onde o servidor envia continuamente dados para o cliente à medida que estes ficam disponíveis, sem que o cliente precise de os solicitar explicitamente. Como a resposta da IA é gerada incrementalmente, faz sentido proporcionar uma experiência de utilização responsiva ao mostrar o resultado da IA à medida que é gerado. E, de facto, todas as APIs de fornecedores de IA que conheço oferecem respostas em streaming como opção nos seus endpoints de conclusão.

A razão pela qual este capítulo aparece aqui no livro, logo após [Utilização de Ferramentas](#), deve-se ao quão poderoso pode ser combinar o uso de ferramentas com respostas de IA em tempo real para os utilizadores. Isto permite experiências dinâmicas e interativas onde a IA pode processar a entrada do utilizador, utilizar várias ferramentas e funções à sua discrição, e depois fornecer respostas em tempo real.

Para alcançar esta interação perfeita, é necessário escrever manipuladores de stream que possam distribuir tanto as chamadas de função de ferramentas invocadas pela IA como a saída de texto simples para o utilizador final. A necessidade de criar um ciclo após processar uma função de ferramenta adiciona um desafio interessante ao trabalho.

Implementação de um ReplyStream

Para demonstrar como o processamento de streams pode ser implementado, este capítulo fará uma análise aprofundada de uma versão simplificada da classe `ReplyStream` que é utilizada no Olympia. As instâncias desta classe podem ser passadas como o parâmetro `stream` em bibliotecas cliente de IA como [ruby-openai](#) e [openrouter](#)

Eis como utilizo o `ReplyStream` no `PromptSubscriber` do Olympia, que escuta através do Wisper a criação de novas mensagens do utilizador.

```
1  class PromptSubscriber
2    include Raix::ChatCompletion
3    include Raix::PromptDeclarations
4
5    # many other declarations omitted...
6
7    prompt text: -> { user_message.content },
8              stream: -> { ReplyStream.new(self) },
9              until: -> { bot_message.complete? }
10
11    def message_created(message) # invoked by Wisper
12      return unless message.role.user? && message.content?
13
14      # rest of the implementation omitted...
```

Para além de uma referência de context para o assinante do prompt que o instanciou, a classe `ReplyStream` também possui variáveis de instância para armazenar um buffer de dados recebidos, e arrays para controlar os nomes de funções e argumentos invocados durante o processamento do stream.

```
1 class ReplyStream
2   attr_accessor :buffer, :f_name, :f_arguments, :context
3
4   delegate :bot_message, :dispatch, to: :context
5
6   def initialize(context)
7     self.context = context
8     self.buffer = []
9     self.f_name = []
10    self.f_arguments = []
11  end
12
13  def call(chunk, bytesize = nil)
14    # ...
15  end
16
17  # ...
18 end
```

O método `initialize` configura o estado inicial da instância `ReplyStream`, inicializando o `buffer`, o `context` e outras variáveis.

O método `call` é o ponto de entrada principal para o processamento dos dados em streaming. Recebe um fragmento de dados (representado como um hash) e um parâmetro opcional `bytesize`, que no nosso exemplo não é utilizado. Dentro deste método, a classe utiliza correspondência de padrões para lidar com diferentes cenários baseados na estrutura do fragmento recebido.



Chamar `deep_symbolize_keys` no fragmento ajuda a tornar a correspondência de padrões mais elegante, permitindo-nos operar com símbolos em vez de strings.

```
1 def call(chunk, _bytesize)
2     case chunk.deep_symbolize_keys
3
4     in { # match function name
5         choices: [
6             {
7                 delta: {
8                     tool_calls: [
9                         { index: index, function: {name: name} }
10                    ]
11                }
12            }
13        ] }
14
15     f_name[index] = name
```

O primeiro padrão que estamos a procurar é uma chamada de ferramenta juntamente com o nome da função associada. Se detetarmos uma, guardamo-la no array `f_name`. Armazenamos os nomes das funções num array indexado, porque o modelo é capaz de fazer chamadas paralelas de funções, enviando mais do que uma função para execução em simultâneo.

A chamada paralela de funções é a capacidade de um modelo de IA realizar múltiplas chamadas de funções em conjunto, permitindo que os efeitos e resultados dessas chamadas de funções sejam resolvidos em paralelo. Isto é especialmente útil se as funções demorarem muito tempo, e reduz as idas e voltas com a API, o que por sua vez pode poupar uma quantidade significativa de gasto de tokens.

Em seguida, precisamos de procurar os argumentos correspondentes às chamadas das funções.

```

1   in { # match arguments
2     choices: [
3       {
4         delta: {
5           tool_calls: [
6             {
7               index: index, function: {arguments: argument }
8             }
9           ]
10        }
11      }
12    ]]
13
14    f_arguments[index] ||= "" # initialize if not already
15    f_arguments[index] << argument

```

De forma semelhante a como tratámos os nomes de funções, guardamos os argumentos num array indexado.

De seguida, procuramos as mensagens visíveis ao utilizador, que chegarão do servidor um token de cada vez e serão atribuídas à variável `new_content`. Também precisamos de estar atentos ao `finish_reason`. Este será `nil` até ao último fragmento da sequência de saída.

```

1   in {
2     choices: [
3       { delta: {content: new_content}, finish_reason: finish_reason }
4     ]}
5
6     # you could transmit every chunk to the user here...
7     buffer << new_content.to_s
8
9     if finish_reason.present?
10      finalize
11    elsif new_content.to_s.match?(/\n\n/)
12      send_to_client # ...or buffer and transmit once per paragraph
13    end

```

É importante notar que adicionamos uma expressão de correspondência de padrões para lidar com mensagens de erro enviadas pelo fornecedor do modelo de IA. Em ambientes

de desenvolvimento local, lançamos uma exceção, mas em produção, registamos o erro e finalizamos.

```
1  in { error: { message: } }  
2    if Rails.env.local?  
3      raise message  
4    else  
5      Honeybadger.notify("AI Error: #{message}")  
6      finalize  
7    end
```

A cláusula else final do case será executada se nenhum dos padrões anteriores corresponder. É apenas uma salvaguarda para que, se o modelo de IA começar a enviar-nos fragmentos não reconhecidos, possamos descobri-los.

```
1  else  
2    Honeybadger.notify("Unrecognized Chunk: #{chunk}")  
3  end  
4  end
```

O método `send_to_client` é responsável por enviar o conteúdo em buffer para o cliente. Verifica se o buffer não está vazio, atualiza o conteúdo da mensagem do bot, processa a mensagem do bot e guarda o conteúdo na base de dados para garantir a persistência dos dados.

```
1 def send_to_client
2   # no need to process pure whitespace
3   return if buffer.join.squish.blank?
4
5   # set the buffer content on the bot message
6   content = buffer.join
7   bot_message.content = content
8
9   # save to database so that we never lose data
10  # even if the stream doesn't terminate correctly
11  bot_message.update_column(:content, content)
12
13  # update content via websocket
14  ConversationRenderer.update(bot_message)
15 end
```

O método `finalize` é chamado quando o processamento do stream está concluído. Este despacha as chamadas de função, caso tenham sido recebidas durante o stream, atualiza a mensagem do bot com o conteúdo final e outras informações relevantes, e reinicia o histórico de chamadas de função

```
1 def finalize
2   if f_name.any?
3     f_name.each_with_index do |name, index|
4       # takes care of calling the function wherever it's implemented
5       dispatch(name:, arguments: JSON.parse(f_arguments[index]))
6     end
7
8     # reset the function call history
9     f_name.clear
10    f_arguments.clear
11  else
12    content = buffer.join.presence
13    bot_message.update!(content:, complete: true)
14    ConversationRenderer.update(bot_message)
15  end
16 end
```

Se o modelo decidir chamar uma função, é necessário “despachar” essa chamada de

função (nome e argumentos) de forma a que seja executada e as mensagens `function_call` e `function_result` sejam adicionadas à transcrição da conversa

Na minha experiência, é melhor gerir a criação de mensagens de função num único local da sua base de código, em vez de depender das implementações das ferramentas. É mais limpo, mas também tem uma razão prática muito importante: se o modelo de IA chamar uma função e não vir as mensagens resultantes da chamada e do resultado na transcrição quando fizer o loop, *irá chamar a mesma função novamente*. Potencialmente para sempre. Lembre-se que a IA é completamente sem estado, por isso, a menos que você reflita essas chamadas de função de volta para ela, elas não aconteceram.

```
1  # PromptSubscriber#dispatch
2
3  def dispatch(name:, arguments:):
4      # adds a function_call message to the conversation transcript
5      # plus dispatches to tool and returns result
6      conversation.function_call!(name, arguments).then do |result|
7          # add function result message to the transcript
8          conversation.function_result!(name, result)
9      end
10 end
```



Limpar o histórico de chamadas de função após o despacho é tão importante quanto garantir que a chamada e os resultados acabem na sua transcrição, para que você não fique apenas chamando as mesmas funções repetidamente cada vez que executa o ciclo.

O “Ciclo de Conversação”

I continuo mencionando ciclos, mas se você é novo em chamadas de função, pode não ser óbvio *por que* precisamos fazer um ciclo. A razão é que quando a IA “pede” para executar funções de ferramentas em seu nome, ela para de responder. Cabe a você executar essas

funções, recolher os resultados, adicionar os resultados à transcrição e depois submeter o prompt original novamente para obter um novo conjunto de chamadas de função ou resultados direcionados ao utilizador.

Na classe `PromptSubscriber`, usamos o método `prompt` do módulo `PromptDeclarations` para definir o comportamento do ciclo de conversação. O parâmetro `until` é definido como `-> { bot_message.complete? }`, o que significa que o ciclo continuará até que a `bot_message` seja marcada como completa.

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



Mas quando é que `bot_message` é marcada como completa? Se não se lembra, consulte novamente a linha 13 do método `finalize`.

Vamos rever toda a lógica de processamento de stream.

1. O `PromptSubscriber` recebe uma nova mensagem do utilizador através do método `message_created`, que é invocado pelo sistema pub/sub Wisper sempre que o utilizador final cria um novo prompt.
2. O método de classe `prompt` define de forma declarativa o comportamento da lógica de conclusão do chat para o `PromptSubscriber`. O modelo de IA executará uma conclusão do chat com o conteúdo da mensagem do utilizador, uma nova instância de `ReplyStream` como parâmetro de `stream`, e a condição de loop especificada.
3. O modelo de IA processa o prompt e começa a gerar uma resposta. À medida que a resposta é transmitida em stream, o método `call` da instância `ReplyStream` é invocado para cada fragmento de dados.
4. Se o modelo de IA decidir chamar uma função de ferramenta, o nome da função e os argumentos são extraídos do fragmento e armazenados nos arrays `f_name` e `f_arguments`, respetivamente.

5. Se o modelo de IA gerar conteúdo visível ao utilizador, este é armazenado em buffer e enviado ao cliente através do método `send_to_client`.
6. Assim que o processamento do stream estiver concluído, o método `finalize` é chamado. Se alguma função de ferramenta tiver sido invocada durante o stream, estas são despachadas usando o método `dispatch` do `PromptSubscriber`.
7. O método `dispatch` adiciona uma mensagem `function_call` à transcrição da conversa, executa a função de ferramenta correspondente e adiciona uma mensagem `function_result` à transcrição com o resultado da chamada da função.
8. Após despachar as funções de ferramenta, o histórico de chamadas de função é limpo para evitar chamadas de função duplicadas em loops subsequentes.
9. Se nenhuma função de ferramenta foi invocada, o método `finalize` atualiza o `bot_message` com o conteúdo final, marca-o como completo e envia a mensagem atualizada para o cliente.
10. A condição de loop `-> { bot_message.complete? }` é avaliada. Se o `bot_message` não estiver marcado como completo, o loop continua, e o prompt original é submetido novamente com a transcrição da conversa atualizada.
11. Os passos 3-10 são repetidos até que o `bot_message` seja marcado como completo, indicando que o modelo de IA terminou de gerar a sua resposta e não são necessárias mais funções de ferramenta.

Ao implementar este loop de conversa, permite-se que o modelo de IA se envolva numa interação de ida e volta com a aplicação, executando funções de ferramenta conforme necessário e gerando respostas visíveis ao utilizador até que a conversa chegue a uma conclusão natural.

A combinação do processamento de stream e do loop de conversa permite experiências dinâmicas e interativas baseadas em IA, onde o modelo de IA pode processar a entrada do utilizador, utilizar várias ferramentas e funções, e fornecer respostas em tempo real com base no contexto evolutivo da conversa.

Continuação Automática

É importante estar ciente das limitações da saída da IA. A maioria dos modelos tem um número máximo de tokens que podem gerar numa única resposta, que é determinado pelo parâmetro `max_tokens`. Se o modelo de IA atingir este limite durante a geração de uma resposta, irá parar abruptamente e indicar que a saída foi truncada.

Na resposta em stream da API da plataforma de IA, pode detetar esta situação examinando a variável `finish_reason` no fragmento. Se a `finish_reason` estiver definida como `"length"` (ou algum outro valor-chave específico do modelo), significa que o modelo atingiu o seu limite máximo de tokens durante a geração e a saída foi interrompida.

Uma forma de lidar com este cenário de forma elegante e proporcionar uma experiência de utilizador fluida, é implementar um mecanismo de continuação automática na sua lógica de processamento de stream. Ao adicionar um padrão de correspondência para razões de finalização relacionadas com o comprimento, pode optar por fazer um loop e continuar automaticamente a saída a partir do ponto onde parou.

Aqui está um exemplo propositadamente simplificado de como pode modificar o método `call` na classe `ReplyStream` para suportar a continuação automática:

```
1  LENGTH_STOPS = %w[length MAX_TOKENS]
2
3  def call(chunk, _bytesize)
4    case chunk.deep_symbolize_keys
5      # ...
6
7    in {
8      choices: [
9        { delta: {content: new_content},
10          finish_reason: finish_reason } ] }
11
12    buffer << new_content.to_s
13
14    if finish_reason.blank?
```

```
15         send_to_client if new_content.to_s.match?(/\n\n/)
16     elsif LENGTH_STOPS.include?(finish_reason)
17         continue_cutoff
18     else
19         finalize
20     end
21
22     # ...
23 end
24 end
25
26 private
27
28 def continue_cutoff
29     conversation.bot_message!(buffer.join, visible: false)
30     conversation.user_message!("please continue", visible: false)
31     bot_message.update_column(:created_at, Time.current)
32 end
```

Nesta versão modificada, quando o `finish_reason` indica uma saída truncada, em vez de finalizar o stream, adicionamos um par de mensagens à transcrição sem finalizar, movemos a mensagem original voltada para o utilizador para o “fundo” da transcrição atualizando o seu atributo `created_at`, e depois permitimos que o ciclo aconteça, para que a IA continue a gerar a partir do ponto onde parou.

Lembre-se que o endpoint de conclusão da IA não mantém estado. Ele só “sabe” o que lhe é comunicado através da transcrição. Neste caso, a forma como comunicamos à IA que foi interrompida é adicionando mensagens “invisíveis” (para o utilizador final) à transcrição. No entanto, lembre-se que este é um exemplo propositalmente simplificado. Uma implementação real necessitaria de uma gestão adicional da transcrição para garantir que não desperdiçamos tokens e/ou confundimos a IA com mensagens duplicadas do assistente na transcrição.

Uma implementação real de auto-continuação também deve ter a chamada lógica de disjuntor implementada para evitar ciclos descontrolados. A razão é que, dados certos tipos de prompts do utilizador e configurações baixas de `max_tokens`, a IA poderia

continuar a gerar saídas voltadas para o utilizador indefinidamente.

Tenha em mente que cada ciclo requer um pedido separado, e que cada pedido consome novamente toda a sua transcrição. Deve definitivamente considerar o equilíbrio entre a experiência do utilizador e o uso da API ao decidir se deve implementar a auto-continuação na sua aplicação. A auto-continuação, em particular, pode ser perigosamente dispendiosa, especialmente quando se utilizam modelos comerciais premium.

Conclusão

O processamento de stream é um aspeto crítico na construção de aplicações baseadas em IA que combinam o uso de ferramentas com respostas em tempo real da IA. Ao gerir eficientemente os dados em stream das APIs de plataformas de IA, pode proporcionar uma experiência de utilizador fluida e interativa, gerir respostas grandes, otimizar o uso de recursos e lidar graciosamente com erros.

A classe `Conversation::ReplyStream` fornecida demonstra como o processamento de stream pode ser implementado numa aplicação Ruby usando correspondência de padrões e arquitetura orientada a eventos. Ao compreender e aproveitar as técnicas de processamento de stream, pode desbloquear todo o potencial da integração de IA nas suas aplicações e proporcionar experiências de utilizador poderosas e envolventes.

Dados Auto-Reparáveis



Dados auto-reparáveis é uma abordagem poderosa para garantir a integridade, consistência e qualidade dos dados em aplicações, aproveitando as capacidades dos modelos de linguagem grande (LLMs). Esta categoria de padrões concentra-se na ideia de usar IA para detetar, diagnosticar e corrigir automaticamente anomalias, inconsistências ou erros nos dados, reduzindo assim a carga sobre os programadores e mantendo um elevado nível de fiabilidade dos dados.

No seu núcleo, os padrões de dados auto-reparáveis reconhecem que os dados são a força vital de qualquer aplicação, e garantir a sua precisão e integridade é crucial para o funcionamento adequado e a experiência do utilizador da aplicação. No entanto, gerir e manter a qualidade dos dados pode ser uma tarefa complexa e demorada, especialmente à medida que as aplicações crescem em tamanho e complexidade. É aqui que entra o poder da IA.

Nos padrões de dados auto-reparáveis, os trabalhadores de IA são empregues para monitorizar e analisar continuamente os dados da sua aplicação. Estes modelos têm a capacidade de compreender e interpretar padrões, relações e anomalias dentro dos dados. Ao aproveitarem as suas capacidades de processamento e compreensão de linguagem natural, podem identificar potenciais problemas ou inconsistências nos dados e tomar as ações apropriadas para os retificar.

O processo de dados auto-reparáveis normalmente envolve vários passos-chave:

1. **Monitorização de Dados:** Os trabalhadores de IA monitorizam constantemente os fluxos de dados, bases de dados ou sistemas de armazenamento da aplicação, procurando quaisquer sinais de anomalias, inconsistências ou erros. Alternativamente, pode ativar um componente de IA em reação a uma exceção.
2. **Deteção de Anomalias:** Quando um problema é detetado, o trabalhador de IA analisa os dados em detalhe para identificar a natureza específica e o âmbito do problema. Isto pode envolver a deteção de valores em falta, formatos inconsistentes ou dados que violam regras ou restrições predefinidas.
3. **Diagnóstico e Correção:** Uma vez identificado o problema, o trabalhador de IA usa o seu conhecimento e compreensão do domínio dos dados para determinar o curso de ação apropriado. Isto pode envolver a correção automática dos dados, o preenchimento de valores em falta ou a sinalização do problema para intervenção humana, se necessário.
4. **Aprendizagem Contínua (opcional, dependendo do caso de uso):** À medida que o seu trabalhador de IA encontra e resolve vários problemas de dados, pode produzir metadados descrevendo o que aconteceu e como respondeu. Estes metadados podem ser alimentados em processos de aprendizagem que permitem que você (e talvez o modelo subjacente, através de fine-tuning) se torne mais eficaz e eficiente ao longo do tempo na identificação e resolução de anomalias nos dados.

Ao detetar e corrigir automaticamente problemas nos dados, pode garantir que a

sua aplicação opera com dados de alta qualidade e fiáveis. Isto reduz o risco de erros, inconsistências ou bugs relacionados com dados afetarem a funcionalidade ou a experiência do utilizador da aplicação.

Uma vez que tenha trabalhadores de IA a lidar com a tarefa de monitorização e correção de dados, pode concentrar os seus esforços noutros aspetos críticos da aplicação. Isto poupa tempo e recursos que de outra forma seriam gastos em limpeza e manutenção manual de dados. De facto, à medida que as suas aplicações crescem em tamanho e complexidade, gerir manualmente a qualidade dos dados torna-se cada vez mais desafiante. Os padrões de “Dados Auto-Reparáveis” escalam eficazmente ao aproveitar o poder da IA para lidar com grandes volumes de dados e detetar problemas em tempo real.



Devido à sua natureza, os modelos de IA podem adaptar-se a padrões, esquemas ou requisitos de dados em mudança ao longo do tempo com pouca ou nenhuma supervisão. Desde que as suas diretivas forneçam orientação adequada, especialmente em relação aos resultados pretendidos, a sua aplicação pode ser capaz de evoluir e lidar com novos cenários de dados sem requerer intervenção manual extensiva ou alterações no código.

Os padrões de dados auto-reparáveis alinham-se bem com as outras categorias de padrões que discutimos, como a “Multiplicidade de Trabalhadores”. A capacidade de dados auto-reparáveis pode ser vista como um tipo especializado de trabalhador que se concentra especificamente em garantir a qualidade e integridade dos dados. Este tipo de trabalhador opera ao lado de outros trabalhadores de IA, cada um contribuindo para diferentes aspetos da funcionalidade da aplicação.

Implementar padrões de dados auto-reparáveis na prática requer um design cuidadoso e integração de modelos de IA na arquitetura da aplicação. Devido aos riscos de perda e corrupção de dados, deve definir diretrizes claras sobre como utilizará esta técnica. Deve também considerar fatores como desempenho, escalabilidade e segurança dos dados.

Caso Prático: Corrigir JSON Defeituoso

Uma das formas mais práticas e convenientes de aproveitar os dados auto-reparáveis é também muito simples de explicar: corrigir JSON defeituoso.

Esta técnica pode ser aplicada ao desafio comum de lidar com dados imperfeitos ou inconsistentes gerados por LLMs, como JSON defeituoso, e fornece uma abordagem para detetar e corrigir automaticamente estes problemas.

Na Olympia, deparo-me regularmente com cenários em que os LLMs geram dados JSON que não são perfeitamente válidos. Isto pode acontecer por várias razões, como quando o LLM adiciona comentários antes ou depois do código JSON propriamente dito, ou quando introduz erros de sintaxe como vírgulas em falta ou aspas duplas não escapadas. Estes problemas podem levar a erros de análise sintática e causar perturbações na funcionalidade da aplicação.

Para resolver este problema, implementei uma solução prática sob a forma de uma classe `JsonFixer`. Esta classe incorpora o padrão “Self-Healing Data”, recebendo como entrada o JSON defeituoso e utilizando um LLM para o corrigir, preservando tanto quanto possível a informação e a intenção original.

```
1 class JsonFixer
2     include Raix::ChatCompletion
3
4     def call(bad_json, error_message)
5         raise "No data provided" if bad_json.blank? || error_message.blank?
6
7         transcript << {
8             system: "Consider user-provided JSON that generated a parse
9                     exception. Do your best to fix it while preserving the
10                    original content and intent as much as possible." }
11         transcript << { user: bad_json }
12         transcript << { assistant: "What is the error message?" }
13         transcript << { user: error_message }
14         transcript << { assistant: "Here is the corrected JSON\n```\njson\n" }
15
```

```

16     self.stop = ["```]"]
17
18     chat_completion(json: true)
19 end
20
21 def model
22     "mistralai/mixtral-8x7b-instruct:nitro"
23 end
24 end

```



Repare como o `JsonFixer` usa o [Ventriloquist](#) para orientar as respostas da IA.

O processo de auto-reparação de dados JSON funciona da seguinte forma:

1. **Geração de JSON:** Um LLM é utilizado para gerar dados JSON com base em determinados prompts ou requisitos. No entanto, devido à natureza dos LLMs, o JSON gerado nem sempre é perfeitamente válido. O parser de JSON irá, naturalmente, gerar um `ParserError` se lhe for fornecido JSON inválido.

```

1 begin
2     JSON.parse(llm_generated_json)
3 rescue JSON::ParserError => e
4     JsonFixer.new.call(llm_generated_json, e.message)
5 end

```

Note que a mensagem de erro também é passada para a chamada do `JSONFixer` para que não seja necessário assumir completamente o que está errado com os dados, especialmente porque o analisador frequentemente indica exatamente qual é o problema.

2. **Correção baseada em MLG:** A classe `JSONFixer` envia o JSON defeituoso de volta para um MLG, juntamente com um prompt ou instrução específica para

corrigir o JSON, preservando ao máximo as informações e a intenção originais. O MLG, treinado com grandes quantidades de dados e com compreensão da sintaxe JSON, tenta corrigir os erros e gerar uma string JSON válida. A [Delimitação de Respostas](#) é utilizada para restringir a saída do MLG, e escolhemos o Mixtral 8x7B como modelo de IA, já que é particularmente adequado para este tipo de tarefa.

3. **Validação e Integração:** A string JSON corrigida retornada pelo MLG é analisada pela própria classe `JSONFixer`, porque esta chamou `chat_completion(json: true)`. Se o JSON corrigido passar na validação, é integrado de volta no fluxo de trabalho da aplicação, permitindo que esta continue a processar os dados sem problemas. O JSON defeituoso foi “reparado”.

Embora eu tenha escrito e reescrito a minha própria implementação do `JSONFixer` várias vezes, duvido que o tempo total investido em todas essas versões seja superior a uma ou duas horas.

Note que a preservação da intenção é um elemento-chave de qualquer padrão de dados auto-reparáveis. O processo de correção baseado em MLG visa preservar ao máximo as informações e a intenção originais do JSON gerado. Isto garante que o JSON corrigido mantém o seu significado semântico e pode ser utilizado eficazmente no contexto da aplicação.

Esta implementação prática da abordagem “Dados Auto-Reparáveis” em Olympia demonstra claramente como a IA, especificamente os MLGs, pode ser aproveitada para resolver desafios reais com dados. Demonstra o poder de combinar técnicas de programação tradicionais com capacidades de IA para construir aplicações robustas e eficientes.

Lei de Postel e o Padrão “Dados Auto-Reparáveis”

“Dados Auto-Reparáveis”, como exemplificado pela classe JSONFixer, alinha-se bem com o princípio conhecido como Lei de Postel, também referido como Princípio da Robustez. A Lei de Postel afirma:

“Seja conservador no que faz, seja liberal no que aceita dos outros.”

Este princípio, originalmente articulado por Jon Postel, um pioneiro da Internet inicial, enfatiza a importância de construir sistemas que sejam tolerantes a entradas diversas ou mesmo ligeiramente incorretas, mantendo ao mesmo tempo uma adesão estrita aos protocolos especificados ao enviar saídas.

No contexto de “Dados Auto-Reparáveis”, a classe JSONFixer incorpora a Lei de Postel ao ser liberal na aceitação de dados JSON defeituosos ou imperfeitos gerados por MLGs. Não rejeita nem falha imediatamente ao encontrar JSON que não adere estritamente ao formato esperado. Em vez disso, adota uma abordagem tolerante e tenta corrigir o JSON usando o poder dos MLGs.

Ao ser liberal na aceitação de JSON imperfeito, a classe JSONFixer demonstra robustez e flexibilidade. Reconhece que os dados no mundo real frequentemente vêm em várias formas e nem sempre podem estar em conformidade com especificações rigorosas. Ao lidar e corrigir graciosamente estes desvios, a classe garante que a aplicação possa continuar a funcionar sem problemas, mesmo na presença de dados imperfeitos.

Por outro lado, a classe JSONFixer também adere ao aspeto conservador da Lei de Postel quando se trata da saída. Após corrigir o JSON usando MLGs, a classe valida o JSON corrigido para garantir que está em estrita conformidade com o formato esperado. Mantém a integridade e correção dos dados antes de os passar para outras partes da aplicação. Esta abordagem conservadora garante que a saída da classe JSONFixer é fiável e consistente, promovendo a interoperabilidade e prevenindo a propagação de erros.

Curiosidades interessantes sobre Jon Postel:

- Jon Postel (1943-1998) foi um cientista da computação americano que desempenhou um papel crucial no desenvolvimento da Internet. Era conhecido como o “Deus da Internet” pelas suas significativas contribuições para os protocolos e padrões fundamentais.
- Postel foi o editor da série de documentos Request for Comments (RFC), que é uma série de notas técnicas e organizacionais sobre a Internet. Ele foi autor ou coautor de mais de 200 RFCs, incluindo os protocolos fundamentais como TCP, IP e SMTP.
- Além das suas contribuições técnicas, Postel era conhecido pela sua abordagem humilde e colaborativa. Acreditava na importância de alcançar consenso e trabalhar em conjunto para construir uma rede robusta e interoperável.
- Postel serviu como Diretor da Divisão de Redes de Computadores no Information Sciences Institute (ISI) da University of Southern California (USC) de 1977 até à sua morte prematura em 1998.
- Em reconhecimento das suas imensas contribuições, Postel foi postumamente galardoado com o prestigiado Prémio Turing em 1998, frequentemente referido como o “Prémio Nobel da Computação.”

A classe `JSONFixer` promove robustez, flexibilidade e interoperabilidade, que foram valores fundamentais que Postel defendeu ao longo da sua carreira. Ao construir sistemas que são tolerantes a imperfeições, mantendo simultaneamente uma adesão estrita aos protocolos, podemos criar aplicações mais resilientes e adaptáveis face aos desafios do mundo real.

Considerações e Contraindicações

A aplicabilidade das abordagens de dados auto-reparáveis depende inteiramente do tipo de dados que a sua aplicação manipula. Existe uma razão pela qual poderá não querer simplesmente fazer um monkeypatch do `JSON.parse` para corrigir automaticamente

todos os erros de análise JSON na sua aplicação: nem todos os erros podem ou devem ser corrigidos automaticamente.

A auto-reparação é particularmente complexa quando associada a requisitos regulamentares ou de conformidade relacionados com o tratamento e processamento de dados. Alguns setores, como a saúde e as finanças, têm regulamentos tão rigorosos relativamente à integridade e auditabilidade dos dados que fazer qualquer tipo de correção de dados em “black box” sem supervisão ou registo adequados pode violar estes regulamentos. É crucial garantir que quaisquer técnicas de dados auto-reparáveis que desenvolva estejam alinhadas com os quadros legais e regulamentares aplicáveis.

A aplicação de técnicas de dados auto-reparáveis, particularmente as que envolvem modelos de IA, também pode ter um grande impacto no desempenho da aplicação e na utilização de recursos. O processamento de grandes volumes de dados através de modelos de IA para deteção e correção de erros pode ser computacionalmente intensivo. É importante avaliar os compromissos entre os benefícios dos dados auto-reparáveis e os custos associados ao desempenho e recursos.

Dito isto, vamos analisar os fatores envolvidos na decisão de quando e onde aplicar esta poderosa abordagem.

Criticidade dos Dados

Ao considerar a aplicação de técnicas de dados auto-reparáveis, é crucial avaliar a criticidade dos dados que estão a ser processados. O nível de criticidade refere-se à importância e sensibilidade dos dados no contexto da sua aplicação e do seu domínio de negócio.

Em alguns casos, a correção automática de erros de dados pode não ser apropriada, especialmente se os dados forem altamente sensíveis ou tiverem implicações legais. Por exemplo, considere os seguintes cenários:

1. **Transações Financeiras:** Em aplicações financeiras, como sistemas bancários ou

plataformas de negociação, a precisão dos dados é da máxima importância. Mesmo erros menores em dados financeiros podem ter consequências significativas, como saldos de conta incorretos, fundos mal encaminhados ou decisões de negociação errôneas. Nestes casos, correções automatizadas sem verificação e auditoria minuciosas podem introduzir riscos inaceitáveis.

2. **Registos Médicos:** As aplicações de saúde lidam com dados de pacientes altamente sensíveis e confidenciais. Imprecisões nos registos médicos podem ter graves implicações para a segurança e decisões de tratamento dos pacientes. Modificar automaticamente dados médicos sem supervisão adequada e validação por profissionais de saúde qualificados pode violar requisitos regulamentares e colocar em risco o bem-estar dos pacientes.
3. **Documentos Legais:** As aplicações que lidam com documentos legais, como contratos, acordos ou processos judiciais, exigem rigorosa precisão e integridade. Mesmo erros menores em dados legais podem ter ramificações legais significativas. Correções automatizadas neste domínio podem não ser apropriadas, pois os dados frequentemente requerem revisão manual e verificação por especialistas jurídicos para garantir a sua validade e aplicabilidade.

Nestes cenários de dados críticos, os riscos associados às correções automáticas frequentemente superam os potenciais benefícios. As consequências de introduzir erros ou modificar dados incorretamente podem ser graves, levando a perdas financeiras, responsabilidades legais ou até mesmo danos a indivíduos.

Ao lidar com dados altamente críticos, é essencial priorizar processos de verificação e validação manual. A supervisão e experiência humana são cruciais para garantir a precisão e integridade dos dados. As técnicas de auto-reparação automatizadas podem ainda ser empregues para sinalizar potenciais erros ou inconsistências, mas a decisão final sobre correções deve envolver julgamento e aprovação humana.

No entanto, é importante notar que nem todos os dados numa aplicação podem ter o mesmo nível de criticidade. Dentro da mesma aplicação, podem existir subconjuntos

de dados que são menos sensíveis ou têm menor impacto se ocorrerem erros. Em tais casos, as técnicas de dados auto-reparáveis podem ser aplicadas seletivamente a esses subconjuntos específicos de dados, enquanto os dados críticos permanecem sujeitos a verificação manual.

O fundamental é avaliar cuidadosamente a criticidade de cada categoria de dados na sua aplicação e definir diretrizes e processos claros para lidar com correções baseadas nos riscos e implicações associados. Ao diferenciar entre dados críticos (ou seja, registros contábilísticos, registros médicos) e não críticos (ou seja, endereços postais, avisos de recursos), pode estabelecer um equilíbrio entre aproveitar os benefícios das técnicas de dados auto-reparáveis onde apropriado e manter controlo e supervisão rigorosos onde necessário.

Em última análise, a decisão de aplicar técnicas de dados auto-reparáveis a dados críticos deve ser tomada em consulta com especialistas do domínio, consultores jurídicos e outras partes interessadas relevantes. É essencial considerar os requisitos específicos, regulamentos e riscos associados aos dados da sua aplicação e alinhar as estratégias de correção de dados em conformidade.

Gravidade do Erro

Ao aplicar técnicas de dados auto-reparáveis, é importante avaliar a gravidade e o impacto dos erros nos dados. Nem todos os erros são criados iguais, e o curso de ação apropriado pode variar dependendo da gravidade do problema.

Inconsistências menores ou problemas de formatação podem ser adequados para correção automática. Por exemplo, um trabalhador de dados auto-reparáveis encarregado de corrigir JSON defeituoso pode lidar com vírgulas em falta ou aspas duplas não escapadas sem alterar significativamente o significado ou a estrutura dos dados. Estes tipos de erros são frequentemente simples de corrigir e têm um impacto mínimo na integridade geral dos dados.

No entanto, erros mais graves que alteram fundamentalmente o significado ou a integridade dos dados podem exigir uma abordagem diferente. Nesses casos, as correções automatizadas podem não ser suficientes, e a intervenção humana pode ser necessária para garantir a precisão e validade dos dados.

É aqui que entra em jogo o conceito de usar a própria IA para ajudar a determinar a gravidade do erro. Ao aproveitar as capacidades dos modelos de IA, podemos projetar operadores de dados auto-reparadores que não só corrigem erros, mas também avaliam a gravidade desses erros e tomam decisões informadas sobre como tratá-los.

Por exemplo, consideremos um operador de dados auto-reparador responsável por corrigir inconsistências nos dados que fluem para uma base de dados de clientes. O operador pode ser projetado para analisar os dados e identificar potenciais erros, como informações em falta ou conflitantes. No entanto, em vez de corrigir automaticamente todos os erros, o operador pode ser equipado com chamadas de ferramentas adicionais que lhe permitem sinalizar erros graves para revisão humana.

Eis um exemplo de como isto pode ser implementado:

```
1 class CustomerDataReviewer
2   include Raix::ChatCompletion
3   include Raix::FunctionDeclarations
4
5   attr_accessor :customer
6
7   function :flag_for_review, reason: { type: "string" } do |params|
8     AdminNotifier.review_request(customer, params[:reason])
9   end
10
11   def initialize(customer)
12     self.customer = customer
13   end
14
15   def call(customer_data)
16     transcript << {
17       system: "You are a customer data reviewer. Your task is to identify
18         and correct inconsistencies in customer data."
```

```

19
20         < 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```\n" }
27
28     self.stop = ["```\n"]
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

```

Neste exemplo, o worker `CustomerDataHealer` foi projetado para identificar e corrigir inconsistências nos dados dos clientes. Mais uma vez, utilizamos [Response Fencing](#) e [Ventriloquist](#) para obter uma saída estruturada. É importante notar que a diretiva do sistema do worker inclui instruções para utilizar a função `flag_for_review` caso sejam encontrados erros graves.

Quando o worker processa os dados do cliente, analisa-os e tenta corrigir quaisquer inconsistências. Se o worker determinar que os erros são graves e requerem intervenção humana, pode utilizar a ferramenta `flag_for_review` para sinalizar os dados e fornecer um motivo para a sinalização.

O método `chat_completion` é chamado com `json: true` para analisar os dados corrigidos do cliente como JSON. Não existe provisão para ciclos após uma chamada de função, pelo que o resultado ficará em branco se `flag_for_review` tiver sido invocado. Caso contrário, o cliente é atualizado com os dados revistos e potencialmente corrigidos.

Ao incorporar a avaliação da gravidade dos erros e a opção de sinalizar dados para revisão humana, o worker de dados auto-reparáveis torna-se mais inteligente e

adaptável. Pode lidar com erros menores automaticamente, enquanto encaminha erros graves para especialistas humanos para intervenção manual.

Os critérios específicos para determinar a gravidade dos erros podem ser definidos na diretiva do worker com base no conhecimento do domínio e nos requisitos do negócio. Fatores como o impacto na integridade dos dados, o potencial de perda ou corrupção de dados e as consequências de dados incorretos podem ser considerados ao avaliar a gravidade.

Ao aproveitar a IA para avaliar a gravidade dos erros e fornecer opções para intervenção humana, as técnicas de dados auto-reparáveis podem encontrar um equilíbrio entre automação e manutenção da precisão dos dados. Esta abordagem garante que erros menores são corrigidos eficientemente, enquanto erros graves recebem a atenção e experiência necessárias dos revisores humanos.

Complexidade do Domínio

Ao considerar a aplicação de técnicas de dados auto-reparáveis, é importante avaliar a complexidade do domínio de dados e as regras que governam a sua estrutura e relações. A complexidade do domínio pode impactar significativamente a eficácia e viabilidade das abordagens de correção automática de dados.

As técnicas de dados auto-reparáveis funcionam bem quando os dados seguem padrões e restrições bem definidos. Em domínios onde a estrutura de dados é relativamente simples e as relações entre elementos de dados são diretas, as correções automatizadas podem ser aplicadas com um alto grau de confiança. Por exemplo, corrigir problemas de formatação ou impor restrições básicas de tipo de dados pode frequentemente ser tratado de forma eficaz por workers de dados auto-reparáveis.

No entanto, à medida que a complexidade do domínio de dados aumenta, os desafios associados à correção automática de dados também crescem. Em domínios com lógica de negócio intrincada, relações complexas entre entidades de dados ou regras

e exceções específicas do domínio, as técnicas de dados auto-reparáveis podem nem sempre capturar as nuances e podem introduzir consequências não intencionais.

Consideremos um exemplo de um domínio complexo: um sistema de negociação financeira. Neste domínio, os dados envolvem vários instrumentos financeiros, dados de mercado, regras de negociação e requisitos regulatórios. As relações entre diferentes elementos de dados podem ser intrincadas, e as regras que governam a validade e consistência dos dados podem ser altamente específicas do domínio.

Em tal domínio complexo, um worker de dados auto-reparáveis encarregado de corrigir inconsistências em dados de negociação precisaria ter um entendimento profundo das regras e restrições específicas do domínio. Precisaria considerar fatores como regulamentações de mercado, limites de negociação, cálculos de risco e procedimentos de liquidação. Correções automatizadas neste contexto podem nem sempre capturar toda a complexidade do domínio e podem inadvertidamente introduzir erros ou violar regras específicas do domínio.

Para abordar os desafios da complexidade do domínio, as técnicas de dados auto-reparáveis podem ser aprimoradas incorporando conhecimento e regras específicas do domínio nos modelos e workers de IA. Isto pode ser alcançado através de técnicas como:

1. **Treino Específico do Domínio:** Os modelos de IA utilizados para dados auto-reparáveis podem ser direcionados ou até mesmo ajustados em conjuntos de dados específicos do domínio que capturam as complexidades e regras do domínio particular. Ao expor os modelos a dados e cenários representativos, eles podem aprender os padrões, restrições e exceções específicos do domínio.
2. **Restrições Baseadas em Regras:** Os workers de dados auto-reparáveis podem ser aumentados com restrições explícitas baseadas em regras que codificam conhecimento específico do domínio. Estas regras podem ser definidas por especialistas do domínio e integradas no processo de correção de dados. Os modelos de IA podem então usar estas regras para guiar suas decisões e garantir conformidade com requisitos específicos do domínio.

3. **Colaboração com Especialistas do Domínio:** Em domínios complexos, é crucial envolver especialistas do domínio no design e desenvolvimento de técnicas de dados auto-reparáveis. Especialistas do domínio podem fornecer insights valiosos sobre as complexidades dos dados, as regras de negócio e os possíveis casos extremos. Seu conhecimento pode ser incorporado nos modelos e workers de IA para melhorar a precisão e confiabilidade das correções automáticas de dados usando padrões [Human In The Loop](#).
4. **Abordagem Incremental e Iterativa:** Ao lidar com domínios complexos, é frequentemente benéfico adotar uma abordagem incremental e iterativa para dados auto-reparáveis. Em vez de tentar automatizar correções para todo o domínio de uma vez, foque em subdomínios específicos ou categorias de dados onde as regras e restrições são bem compreendidas. Gradualmente expanda o escopo das técnicas de auto-reparação à medida que o entendimento do domínio cresce e as técnicas se provam eficazes.

Ao considerar a complexidade do domínio dos dados e incorporar o conhecimento específico do domínio nas técnicas de dados auto-reparáveis, é possível encontrar um equilíbrio entre automação e precisão. É importante reconhecer que os dados auto-reparáveis não são uma solução única para todos os casos e que a abordagem deve ser adaptada aos requisitos e desafios específicos de cada domínio.

Em domínios complexos, uma abordagem híbrida que combine técnicas de dados auto-reparáveis com experiência e supervisão humana pode ser mais eficaz. As correções automatizadas podem lidar com casos rotineiros e bem definidos, enquanto cenários complexos ou exceções podem ser sinalizados para revisão e intervenção humana. Esta abordagem colaborativa garante que os benefícios da automação sejam alcançados, mantendo o controle e a precisão necessários em domínios de dados complexos.

Explicabilidade e Transparência

A explicabilidade refere-se à capacidade de compreender e interpretar o raciocínio por trás das decisões tomadas pelos modelos de IA, enquanto a transparência envolve fornecer visibilidade clara sobre o processo de correção de dados.

Em muitos contextos, as modificações de dados precisam de ser auditáveis e justificáveis. As partes interessadas, incluindo utilizadores empresariais, auditores e órgãos reguladores, podem exigir explicações sobre o motivo de certas correções de dados terem sido feitas e como os modelos de IA chegaram a essas decisões. Isto é especialmente crucial em domínios onde a precisão e integridade dos dados têm implicações significativas, como finanças, saúde e questões jurídicas.

Para abordar a necessidade de explicabilidade e transparência, as técnicas de dados auto-reparáveis devem incorporar mecanismos que forneçam informações sobre o processo de tomada de decisão dos modelos de IA. Isto pode ser alcançado através de várias abordagens:

1. **Encadeamento de Pensamento:** Pedir ao modelo para explicar o seu raciocínio “em voz alta” antes de aplicar alterações aos dados pode permitir uma compreensão mais fácil do processo de tomada de decisão e pode gerar explicações legíveis por humanos para as correções feitas. O compromisso é um pouco mais de complexidade na separação da explicação da saída de dados estruturados, que pode ser resolvida por...
2. **Geração de Explicações:** Os sistemas de dados auto-reparáveis podem ser equipados com a capacidade de gerar explicações legíveis por humanos para as correções que fazem. Isto pode ser alcançado pedindo ao modelo para apresentar o seu processo de tomada de decisão como explicações facilmente compreensíveis *integradas nos próprios dados*. Por exemplo, um sistema de dados auto-reparáveis poderia gerar um relatório que destaca as inconsistências específicas de dados que identificou, as correções aplicadas e a lógica por trás dessas correções.

3. **Importância das Características:** Os modelos de IA podem ser instruídos com informações sobre a importância de diferentes características ou atributos no processo de correção de dados como parte das suas diretivas. Essas diretivas, por sua vez, podem ser expostas às partes interessadas humanas. Ao identificar os fatores-chave que influenciam as decisões do modelo, as partes interessadas podem obter informações sobre o raciocínio por trás das correções e avaliar a sua validade.
4. **Registo e Auditoria:** Implementar mecanismos abrangentes de registo e auditoria é crucial para manter a transparência no processo de dados auto-reparáveis. Cada correção de dados feita pelos modelos de IA deve ser registada, incluindo os dados originais, os dados corrigidos e as ações específicas tomadas. Esta trilha de auditoria permite a análise retrospectiva e fornece um registo claro das modificações feitas nos dados.
5. **Abordagem com Intervenção Humana:** Incorporar uma abordagem com intervenção humana pode melhorar a explicabilidade e transparência das técnicas de dados auto-reparáveis. Ao envolver especialistas humanos na revisão e validação das correções geradas por IA, as organizações podem garantir que as correções estejam alinhadas com o conhecimento do domínio e os requisitos do negócio. A supervisão humana adiciona uma camada extra de responsabilidade e permite a identificação de possíveis enviesamentos ou erros nos modelos de IA.
6. **Monitorização e Avaliação Contínuas:** A monitorização e avaliação regulares do desempenho das técnicas de dados auto-reparáveis são essenciais para manter a transparência e a confiança. Ao avaliar a precisão e eficácia dos modelos de IA ao longo do tempo, as organizações podem identificar quaisquer desvios ou anomalias e tomar medidas corretivas. A monitorização contínua ajuda a garantir que o processo de dados auto-reparáveis permaneça fiável e alinhado com os resultados desejados.

A explicabilidade e a transparência são considerações críticas ao implementar técnicas de dados auto-reparáveis. Ao fornecer explicações claras para correções de dados,

manter trilhas de auditoria abrangentes e envolver supervisão humana, as organizações podem construir confiança no processo de dados auto-reparáveis e garantir que as modificações feitas nos dados sejam justificáveis e alinhadas com os objetivos do negócio.

É importante encontrar um equilíbrio entre os benefícios da automação e a necessidade de transparência. Embora as técnicas de dados auto-reparáveis possam melhorar significativamente a qualidade e eficiência dos dados, elas não devem vir ao custo de perder visibilidade e controlo sobre o processo de correção de dados. Ao projetar sistemas de dados auto-reparáveis com explicabilidade e transparência em mente, as organizações podem aproveitar o poder da IA enquanto mantêm o nível necessário de responsabilidade e confiança nos dados.

Consequências Não Intencionais

Embora as técnicas de dados auto-reparáveis visem melhorar a qualidade e consistência dos dados, é crucial estar ciente do potencial de consequências não intencionais. As correções automatizadas, se não forem cuidadosamente projetadas e monitorizadas, podem inadvertidamente alterar o significado ou contexto dos dados, levando a problemas subsequentes.

Um dos principais riscos dos dados auto-reparáveis é a introdução de enviesamentos ou erros no processo de correção de dados. Os modelos de IA, como qualquer outro sistema de software, podem estar sujeitos a enviesamentos presentes nos dados de treino ou introduzidos através do design dos algoritmos. Se estes enviesamentos não forem identificados e mitigados, podem propagar-se através do processo de dados auto-reparáveis e resultar em modificações de dados distorcidas ou incorretas.

Por exemplo, considere um trabalhador de dados auto-reparáveis encarregado de corrigir inconsistências em dados demográficos de clientes. Se o modelo de IA tiver aprendido preconceitos a partir de dados históricos, como associar certas ocupações ou níveis de rendimento a géneros ou etnias específicas, poderá fazer suposições incorretas

e modificar os dados de forma a reforçar esses preconceitos. Isto pode levar a perfis de clientes imprecisos, decisões empresariais equivocadas e resultados potencialmente discriminatórios.

Outra potencial consequência não intencional é a perda de informações ou contexto valioso durante o processo de correção de dados. As técnicas de dados auto-reparáveis frequentemente concentram-se na padronização e normalização de dados para garantir consistência. No entanto, em alguns casos, os dados originais podem conter nuances, exceções ou informações contextuais importantes para compreender o quadro completo. Correções automatizadas que impõem a padronização de forma cega podem inadvertidamente remover ou obscurecer estas informações valiosas.

Por exemplo, imagine um trabalhador de dados auto-reparáveis responsável por corrigir inconsistências em registros médicos. Se o trabalhador encontrar um histórico médico de um paciente com uma condição rara ou um plano de tratamento incomum, pode tentar normalizar os dados para se adequarem a um padrão mais comum. No entanto, ao fazê-lo, pode perder os detalhes específicos e o contexto que são cruciais para representar com precisão a situação única do paciente. Esta perda de informação pode ter implicações sérias para o cuidado do paciente e a tomada de decisões médicas.

Para mitigar os riscos de consequências não intencionais, é essencial adotar uma abordagem proativa ao projetar e implementar técnicas de dados auto-reparáveis:

1. **Testes e Validação Minuciosos:** Antes de implementar trabalhadores de dados auto-reparáveis em produção, é crucial testar e validar minuciosamente o seu comportamento em diversos cenários. Isto inclui testes com conjuntos de dados representativos que cubram vários casos extremos, exceções e potenciais preconceitos. Testes rigorosos ajudam a identificar e abordar quaisquer consequências não intencionais antes que afetem dados do mundo real.
2. **Monitorização e Avaliação Contínuas:** Implementar mecanismos de monitorização e avaliação contínuas é essencial para detetar e mitigar consequências não intencionais ao longo do tempo. Rever regularmente os

resultados dos processos de dados auto-reparáveis, analisar o impacto nos sistemas downstream e na tomada de decisões, e recolher feedback das partes interessadas pode ajudar a identificar quaisquer efeitos adversos e provocar ações corretivas oportunas. Se a sua organização tiver painéis operacionais, adicionar métricas claramente visíveis relacionadas com alterações automatizadas de dados é provavelmente uma boa ideia. Adicionar alarmes ligados a grandes desvios da atividade normal de alteração de dados é provavelmente uma ideia ainda melhor!

3. **Supervisão e Intervenção Humana:** Manter a supervisão humana e a capacidade de intervir no processo de dados auto-reparáveis é crucial. Embora a automação possa melhorar muito a eficiência, é importante ter especialistas humanos a rever e validar as correções feitas por modelos de IA, especialmente em domínios críticos ou sensíveis. O julgamento humano e a experiência no domínio podem ajudar a identificar e abordar quaisquer consequências não intencionais que possam surgir.
4. **IA Explicável (XAI) e Transparência:** Como discutido na subsecção anterior, incorporar técnicas de IA explicável e garantir transparência no processo de dados auto-reparáveis pode ajudar a mitigar consequências não intencionais. Ao fornecer explicações claras para correções de dados e manter trilhas de auditoria abrangentes, as organizações podem compreender melhor e rastrear o raciocínio por trás das modificações feitas pelos modelos de IA.
5. **Abordagem Incremental e Iterativa:** Adotar uma abordagem incremental e iterativa para dados auto-reparáveis pode ajudar a minimizar o risco de consequências não intencionais. Em vez de aplicar correções automatizadas a todo o conjunto de dados de uma só vez, comece com um subconjunto de dados e expanda gradualmente o âmbito à medida que as técnicas se provam eficazes e fiáveis. Isto permite uma monitorização cuidadosa e ajustes ao longo do caminho, reduzindo o impacto de quaisquer consequências não intencionais.
6. **Colaboração e Feedback:** Envolver as partes interessadas de diferentes domínios e incentivar a colaboração e feedback durante todo o processo de dados auto-reparáveis pode ajudar a identificar e abordar consequências não intencionais.

Procurar regularmente contribuições de especialistas no domínio, consumidores de dados e utilizadores finais pode fornecer insights valiosos sobre o impacto real das correções de dados e destacar quaisquer questões que possam ter sido negligenciadas.

Ao abordar proativamente o risco de consequências não intencionais e implementar salvaguardas apropriadas, as organizações podem aproveitar os benefícios das técnicas de dados auto-reparáveis enquanto minimizam potenciais efeitos adversos. É importante abordar os dados auto-reparáveis como um processo iterativo e colaborativo, monitorizando, avaliando e refinando continuamente as técnicas para garantir que se alinhem com os resultados desejados e mantenham a integridade e fiabilidade dos dados.

Ao considerar o uso de padrões de dados auto-reparáveis, é essencial avaliar cuidadosamente estes fatores e pesar os benefícios contra os potenciais riscos e limitações. Em alguns casos, uma abordagem híbrida que combine correções automatizadas com supervisão e intervenção humana pode ser a solução mais apropriada.

Também vale a pena notar que as técnicas de dados auto-reparáveis não devem ser vistas como um substituto para validação de dados robusta, sanitização de entrada e mecanismos de tratamento de erros. Estas práticas fundamentais continuam críticas para garantir a integridade e segurança dos dados. Os dados auto-reparáveis devem ser vistos como uma abordagem complementar que pode aumentar e melhorar estas medidas existentes.

Em última análise, a decisão de empregar padrões de dados auto-reparáveis depende dos requisitos específicos, restrições e prioridades da sua aplicação. Ao considerar cuidadosamente as considerações descritas acima e alinhá-las com os objetivos e

arquitetura da sua aplicação, pode tomar decisões informadas sobre quando e como aproveitar eficazmente as técnicas de dados auto-reparáveis.

Geração de Conteúdo Contextual



Os padrões de Geração de Conteúdo Contextual aproveitam o poder dos modelos de linguagem grande (LLMs) para gerar conteúdo dinâmico e específico ao contexto dentro das aplicações. Esta categoria de padrões reconhece a importância de fornecer conteúdo personalizado e relevante aos utilizadores com base nas suas necessidades específicas, preferências e até mesmo interações anteriores e atuais com a aplicação.

No contexto desta abordagem, “conteúdo” refere-se tanto ao conteúdo primário (ou seja, posts de blog, artigos, etc.) como ao meta-conteúdo, como recomendações para conteúdo primário.

Os padrões de Geração de Conteúdo Contextual podem desempenhar um papel crucial no aumento dos seus níveis de envolvimento do utilizador, fornecendo experiências

personalizadas e automatizando tarefas de criação de conteúdo tanto para si como para os seus utilizadores. Ao utilizar os padrões que descrevemos neste capítulo, pode criar aplicações que geram conteúdo dinamicamente, adaptando-se ao contexto e às entradas em tempo real.

Os padrões funcionam integrando LLMs nas saídas da aplicação, desde a interface do utilizador (por vezes referida como “chrome”), até emails e outras formas de notificações, bem como quaisquer pipelines de geração de conteúdo.

Quando um utilizador interage com a aplicação ou desencadeia um pedido específico de conteúdo, a aplicação captura o contexto relevante, como preferências do utilizador, interações anteriores ou prompts específicos. Esta informação contextual é então alimentada ao LLM, juntamente com quaisquer templates ou diretrizes necessárias e utilizada para produzir uma saída textual que de outra forma teria de ser codificada, armazenada numa base de dados ou gerada algoritmicamente.

O conteúdo gerado pelo LLM pode assumir várias formas, como recomendações personalizadas, descrições dinâmicas de produtos, respostas personalizadas por email, ou até mesmo artigos ou posts de blog completos. Um dos usos mais radicais deste conteúdo que eu pionerizei há mais de um ano é a geração dinâmica de elementos da UI como etiquetas de formulários, dicas de contexto e outros tipos de texto explicativo.

Personalização

Um dos principais benefícios dos padrões de Geração de Conteúdo Contextual é a capacidade de fornecer experiências altamente personalizadas aos utilizadores. Ao gerar conteúdo baseado no contexto específico do utilizador, estes padrões permitem que as aplicações adaptem o conteúdo aos interesses, preferências e interações individuais dos utilizadores.

A personalização vai além de simplesmente inserir o nome de um utilizador em conteúdo genérico. Envolve aproveitar o rico contexto disponível sobre cada utilizador para gerar

conteúdo que ressoe com as suas necessidades e desejos específicos. Este contexto pode incluir uma ampla gama de fatores, tais como:

1. **Informação do Perfil do Utilizador:** No nível mais geral de aplicação desta técnica, dados demográficos, interesses, preferências e outros atributos do perfil podem ser utilizados para gerar conteúdo que se alinhe com o histórico e características do utilizador.
2. **Dados Comportamentais:** As interações passadas de um utilizador com a aplicação, como páginas visualizadas, links clicados ou produtos comprados, podem fornecer informações valiosas sobre o seu comportamento e interesses. Estes dados podem ser utilizados para gerar sugestões de conteúdo que reflitam os seus padrões de envolvimento e prevejam as suas necessidades futuras.
3. **Fatores Contextuais:** O contexto atual do utilizador, como a sua localização, dispositivo, hora do dia ou até mesmo o clima, pode influenciar o processo de geração de conteúdo. Por exemplo, uma aplicação de viagens pode ter um trabalhador de IA capaz de gerar recomendações personalizadas com base na localização atual do utilizador e nas condições meteorológicas prevaletentes.

Ao aproveitar estes fatores contextuais, os padrões de Geração de Conteúdo Contextual permitem que as aplicações forneçam conteúdo que parece feito à medida para cada utilizador individual. Este nível de personalização tem vários benefícios significativos:

1. **Aumento do Envolvimento:** O conteúdo personalizado capta a atenção dos utilizadores e mantém-nos envolvidos com a aplicação. Quando os utilizadores sentem que o conteúdo é relevante e fala diretamente às suas necessidades, é mais provável que passem mais tempo a interagir com a aplicação e a explorar as suas funcionalidades.
2. **Maior Satisfação do Utilizador:** O conteúdo personalizado demonstra que a aplicação compreende e se preocupa com os requisitos únicos do utilizador. Ao fornecer conteúdo que é útil, informativo e alinhado com os seus interesses, a

aplicação pode aumentar a satisfação do utilizador e construir uma ligação mais forte com os seus utilizadores.

3. **Taxas de Conversão Mais Elevadas:** No contexto de aplicações de e-commerce ou marketing, o conteúdo personalizado pode ter um impacto significativo nas taxas de conversão. Ao apresentar aos utilizadores produtos, ofertas ou recomendações que são adaptados às suas preferências e comportamento, a aplicação pode aumentar a probabilidade de os utilizadores realizarem as ações desejadas, como fazer uma compra ou inscrever-se num serviço.

Produtividade

Os padrões de Geração de Conteúdo Contextual podem aumentar significativamente certos tipos de produtividade ao reduzir a necessidade de geração e edição manual de conteúdo em processos criativos. Ao aproveitar o poder dos LLMs, pode gerar conteúdo de alta qualidade em escala, poupando tempo e esforço que os seus criadores de conteúdo e desenvolvedores teriam de gastar em trabalho manual tedioso.

Tradicionalmente, os criadores de conteúdo precisam de pesquisar, escrever, editar e formatar conteúdo para garantir que este cumpre os requisitos da aplicação e as expectativas dos utilizadores. Este processo pode consumir muito tempo e recursos, especialmente à medida que o volume de conteúdo aumenta.

No entanto, com os padrões de Geração Contextual de Conteúdo, o processo de criação de conteúdo pode ser largamente automatizado. Os LLMs podem gerar conteúdo coerente, gramaticalmente correto e contextualmente relevante com base nos prompts e diretrizes fornecidos. Esta automatização oferece vários benefícios de produtividade:

1. **Redução do Esforço Manual:** Ao delegar tarefas de geração de conteúdo aos LLMs, os criadores de conteúdo podem concentrar-se em tarefas de nível superior, como estratégia de conteúdo, ideação e garantia de qualidade. Podem fornecer o

contexto necessário, modelos e diretrizes ao LLM e deixar que este trate da geração efetiva do conteúdo. Isto reduz o esforço manual necessário para a escrita e edição, permitindo que os criadores de conteúdo sejam mais produtivos e eficientes.

2. **Criação de Conteúdo Mais Rápida:** Os LLMs podem gerar conteúdo muito mais rapidamente do que escritores humanos. Com os prompts e diretrizes adequados, um LLM pode produzir múltiplos elementos de conteúdo em questão de segundos ou minutos. Esta velocidade permite que as aplicações gerem conteúdo a um ritmo muito mais rápido, acompanhando as exigências dos utilizadores e o panorama digital em constante mudança.

Será que a criação mais rápida de conteúdo está a levar a uma situação de “tragédia dos comuns” onde a internet está a afogar-se em conteúdo que ninguém lê? Infelizmente, suspeito que a resposta é sim.

3. **Consistência e Qualidade:** Os LLMs podem facilmente rever conteúdo para que seja consistente em estilo, tom e qualidade. Mediante diretrizes e exemplos claros, certos tipos de aplicações (ou seja, redações jornalísticas, RP, etc.) podem garantir que o seu conteúdo gerado por humanos está alinhado com a voz da marca e cumpre os padrões de qualidade desejados. Esta consistência reduz a necessidade de edição e revisões extensivas, poupando tempo e esforço no processo de criação de conteúdo.
4. **Iteração e Otimização:** Os padrões de Geração Contextual de Conteúdo permitem uma rápida iteração e otimização do conteúdo. Ao ajustar os prompts, modelos ou diretrizes fornecidos ao LLM, as suas aplicações podem rapidamente gerar variações de conteúdo e testar diferentes abordagens de forma automatizada, algo que nunca foi possível no passado. Este processo iterativo permite uma experimentação e refinamento mais rápidos das estratégias de

conteúdo, levando a conteúdo mais eficaz e envolvente ao longo do tempo. Esta técnica em particular pode ser revolucionária para aplicações como o comércio eletrônico, que dependem criticamente das taxas de rejeição e envolvimento



É importante notar que, embora os padrões de Geração Contextual de Conteúdo possam aumentar significativamente a produtividade, eles não eliminam completamente a necessidade de envolvimento humano. Os criadores e editores de conteúdo continuam a desempenhar um papel crucial na definição da estratégia global de conteúdo, no fornecimento de orientação ao LLM e na garantia da qualidade e adequação do conteúdo gerado.

Ao automatizar os aspetos mais repetitivos e demorados da criação de conteúdo, os padrões de Geração Contextual de Conteúdo libertam tempo e recursos humanos valiosos que podem ser redirecionados para tarefas de maior valor. Este aumento de produtividade permite-lhe fornecer conteúdo mais personalizado e envolvente aos utilizadores, enquanto otimiza os fluxos de trabalho de criação de conteúdo.

Iteração Rápida e Experimentação

Os padrões de Geração Contextual de Conteúdo permitem-lhe iterar e experimentar rapidamente diferentes variações de conteúdo, possibilitando uma otimização e refinamento mais rápidos da sua estratégia de conteúdo. Pode gerar múltiplas versões de conteúdo em questão de segundos, simplesmente ajustando o contexto, modelos ou diretrizes fornecidos ao modelo.

Esta capacidade de iteração rápida oferece vários benefícios principais:

1. **Testes e Otimização:** Com a capacidade de gerar rapidamente variações de conteúdo, pode facilmente testar diferentes abordagens e medir a sua eficácia. Por exemplo, pode gerar múltiplas versões de uma descrição de produto ou de

uma mensagem de marketing, cada uma adaptada a um segmento de utilizadores específico ou contexto. Ao analisar métricas de envolvimento do utilizador, como taxas de cliques ou taxas de conversão, pode identificar as variações de conteúdo mais eficazes e otimizar a sua estratégia de conteúdo em conformidade.

2. **Testes A/B:** Os padrões de Geração Contextual de Conteúdo permitem testes A/B perfeitos do conteúdo. Pode gerar duas ou mais variações de conteúdo e apresentá-las aleatoriamente a diferentes grupos de utilizadores. Ao comparar o desempenho de cada variação, pode determinar qual o conteúdo que melhor ressoa com o seu público-alvo. Esta abordagem baseada em dados permite-lhe tomar decisões informadas e refinar continuamente o seu conteúdo para maximizar o envolvimento do utilizador e alcançar os resultados desejados.
3. **Experiências de Personalização:** A iteração rápida e a experimentação são particularmente valiosas quando se trata de personalização. Com os padrões de Geração Contextual de Conteúdo, pode rapidamente gerar variações de conteúdo personalizado com base em diferentes segmentos de utilizadores, preferências ou comportamentos. Ao experimentar diferentes estratégias de personalização, pode identificar as abordagens mais eficazes para envolver utilizadores individuais e fornecer experiências personalizadas.
4. **Adaptação às Tendências em Mudança:** A capacidade de iterar e experimentar rapidamente permite-lhe manter a agilidade e adaptar-se às tendências e preferências dos utilizadores em constante mudança. À medida que surgem novos tópicos, palavras-chave ou comportamentos dos utilizadores, pode rapidamente gerar conteúdo alinhado com estas tendências. Através da experimentação e refinamento contínuos do seu conteúdo, pode manter-se relevante e conservar uma vantagem competitiva no panorama digital em constante evolução.
5. **Experimentação Económica:** A experimentação tradicional de conteúdo envolve frequentemente tempo e recursos significativos, já que os criadores de conteúdo precisam de desenvolver e testar manualmente diferentes variações. No entanto, com os padrões de Geração Contextual de Conteúdo, o custo da experimentação é

grandemente reduzido. Os LLMs podem gerar variações de conteúdo rapidamente e em escala, permitindo-lhe explorar uma ampla gama de ideias e abordagens sem incorrer em custos substanciais.

Para maximizar o aproveitamento da iteração e experimentação rápidas, é importante ter uma estrutura de experimentação bem definida. Esta estrutura deve incluir:

- Objetivos claros e hipóteses para cada experiência
- Métricas apropriadas e mecanismos de monitorização para medir o desempenho do conteúdo
- Estratégias de segmentação e direcionamento para garantir que as variações de conteúdo relevantes são apresentadas aos utilizadores certos
- Ferramentas de análise e relatórios para extrair insights dos dados experimentais
- Um processo para incorporar aprendizagens e otimizações na sua estratégia de conteúdo

Ao abraçar a iteração e experimentação rápidas, pode continuamente refinar e otimizar o seu conteúdo, garantindo que este permanece envolvente, relevante e eficaz na consecução dos objetivos da sua aplicação. Esta abordagem ágil à criação de conteúdo permite-lhe manter-se à frente da curva e proporcionar experiências excecionais aos utilizadores.

Escalabilidade e Eficiência

À medida que as aplicações crescem e a procura por conteúdo personalizado aumenta, os padrões de geração contextual de conteúdo permitem um escalonamento eficiente da criação de conteúdo. Os LLMs podem gerar conteúdo para um grande número de utilizadores e contextos simultaneamente, sem necessidade de um aumento proporcional nos recursos humanos. Esta escalabilidade permite que as aplicações

ofereçam experiências personalizadas a uma base crescente de utilizadores sem sobrecarregar as suas capacidades de criação de conteúdo.



Note que a geração contextual de conteúdo pode ser utilizada eficazmente para internacionalizar a sua aplicação “em tempo real”. Na verdade, foi exatamente isso que fiz usando a minha Gem Instant18n para disponibilizar a Olympia em mais de meia dúzia de idiomas, mesmo tendo menos de um ano de existência.

Localização Potenciada por IA

Se me permite gabar por um momento, penso que a minha biblioteca Instant18n para aplicações Rails é um exemplo inovador do padrão de “Geração Contextual de Conteúdo” em ação, demonstrando o potencial transformador da IA no desenvolvimento de aplicações. Esta gem aproveita o poder do modelo linguístico GPT da OpenAI para revolucionar a forma como a internacionalização e localização são tratadas em aplicações Rails.

Tradicionalmente, internacionalizar uma aplicação Rails envolve definir manualmente chaves de tradução e fornecer as traduções correspondentes para cada idioma suportado. Este processo pode ser demorado, intensivo em recursos e propenso a inconsistências. No entanto, com a gem Instant18n, o paradigma da localização é completamente redefinido.

Ao integrar um modelo linguístico de grande escala, a gem Instant18n permite-lhe gerar traduções em tempo real, com base no contexto e significado do texto. Em vez de depender de chaves de tradução predefinidas e traduções estáticas, a gem traduz dinamicamente o texto usando o poder da IA. Esta abordagem oferece vários benefícios principais:

1. **Localização Perfeita:** Com a gem Instant18n, os programadores já não precisam de definir e manter manualmente ficheiros de tradução para cada idioma suportado. A gem gera automaticamente traduções com base no texto fornecido e no idioma de destino pretendido, tornando o processo de localização eficiente e simples.
2. **Precisão Contextual:** A IA pode receber contexto suficiente para compreender as nuances do texto a ser traduzido. Pode ter em conta o contexto envolvente, expressões idiomáticas e referências culturais para gerar traduções precisas, naturais e contextualmente apropriadas.
3. **Suporte Extensivo de Idiomas:** A gem Instant18n aproveita o vasto conhecimento e capacidades linguísticas do GPT, permitindo traduções para uma extensa gama de idiomas. Desde idiomas comuns como espanhol e francês até idiomas mais obscuros ou fictícios como klingon e élfico, a gem pode lidar com uma grande variedade de requisitos de tradução.
4. **Flexibilidade e Criatividade:** A gem vai além das traduções linguísticas tradicionais e permite opções de localização criativas e não convencionais. Os programadores podem traduzir texto em vários estilos, dialetos ou até mesmo idiomas fictícios, abrindo novas possibilidades para experiências únicas do utilizador e conteúdo envolvente.
5. **Otimização de Desempenho:** A gem Instant18n incorpora mecanismos de cache para melhorar o desempenho e reduzir a sobrecarga de traduções repetidas. O texto traduzido é armazenado em cache, permitindo que pedidos subsequentes para a mesma tradução sejam servidos rapidamente sem necessidade de chamadas redundantes à API.

A gem Instant18n exemplifica o poder do padrão de “Geração Contextual de Conteúdo” ao aproveitar a IA para gerar conteúdo localizado dinamicamente. Demonstra como a IA pode ser integrada na funcionalidade central de uma aplicação Rails, transformando a forma como os programadores abordam a internacionalização e localização.

Ao eliminar a necessidade de gestão manual de traduções e permitir traduções em tempo real baseadas no contexto, a gem Instant18n poupa aos programadores tempo e esforço significativos. Permite-lhes concentrarem-se na construção das funcionalidades principais da sua aplicação, garantindo simultaneamente que o aspeto da localização é tratado de forma fluida e precisa.

A Importância dos Testes com Utilizadores e do Feedback

Por fim, tenha sempre em mente a importância dos testes com utilizadores e do feedback. É crucial validar que a geração contextual de conteúdo corresponde às expectativas dos utilizadores e está alinhada com os objetivos da aplicação. Continue a iterar e refinar o conteúdo gerado com base nos insights dos utilizadores e na análise de dados. Se estiver a gerar conteúdo dinâmico em grande escala que seria impossível de validar manualmente por si e pela sua equipa, considere adicionar mecanismos de feedback que permitam aos utilizadores reportar conteúdo que seja estranho ou incorreto, juntamente com uma explicação do motivo. Esse feedback precioso pode até ser fornecido a um trabalhador de IA encarregado de fazer ajustes ao componente que gerou o conteúdo!

Generative UI



A atenção é hoje em dia um recurso tão valioso que o envolvimento eficaz do utilizador exige agora experiências de software que não são apenas fluidas e intuitivas, mas também altamente personalizadas para as necessidades, preferências e contextos individuais. Como resultado, designers e programadores enfrentam cada vez mais o desafio de criar interfaces de utilizador que possam adaptar-se e satisfazer os requisitos únicos de cada utilizador *em escala*.

A Generative UI (GenUI) é uma abordagem verdadeiramente revolucionária ao design de interfaces de utilizador que aproveita o poder dos modelos linguísticos de grande escala (LLMs) para criar experiências de utilizador altamente personalizadas e dinâmicas em tempo real. Quis garantir que vos daria pelo menos uma introdução à GenUI neste livro, porque acredito que é uma das oportunidades mais promissoras que existe atualmente no domínio do design e frameworks de aplicações. Estou convencido de que dezenas ou mais novos projetos comerciais e de código aberto bem-sucedidos surgirão

neste nicho específico.

No seu núcleo, a GenUI combina os princípios da [Geração de Conteúdo Contextual](#) com técnicas avançadas de IA para gerar elementos de interface de utilizador, como texto, imagens e layouts, de forma dinâmica, baseando-se numa compreensão profunda do contexto, preferências e objetivos do utilizador. A GenUI permite que designers e programadores criem interfaces que se adaptam e evoluem em resposta às interações do utilizador, proporcionando um nível de personalização anteriormente inatingível.

A GenUI representa uma mudança fundamental na forma como abordamos o design de interfaces de utilizador. Em vez de desenhar para as massas, a GenUI permite-nos desenhar para o indivíduo. O conteúdo e as interfaces personalizadas têm o potencial de criar experiências de utilizador que ressoam com cada utilizador a um nível mais profundo, aumentando o envolvimento, a satisfação e a lealdade.

Como técnica de ponta, a transição para a GenUI está repleta de desafios conceptuais e práticos. Integrar a IA no processo de design, garantir que as interfaces geradas não são apenas personalizadas mas também utilizáveis, acessíveis e alinhadas com a marca e a experiência global do utilizador, todos estes são desafios que tornam a GenUI uma busca para poucos, não para muitos. Além disso, o envolvimento da IA levanta questões sobre privacidade de dados, transparência e até mesmo implicações éticas.

Apesar dos desafios, as experiências personalizadas em escala têm o poder de transformar completamente a forma como interagimos com produtos e serviços digitais. Abre possibilidades para criar interfaces inclusivas e acessíveis que atendam às diversas necessidades dos utilizadores, independentemente das suas capacidades, origens ou preferências.

Neste capítulo, vamos explorar o conceito de GenUI, examinando algumas características definidoras, benefícios principais e potenciais desafios. Começamos por considerar a forma mais básica e acessível de GenUI: gerar texto para interfaces de utilizador tradicionalmente desenhadas e implementadas.

Gerar Texto para Interfaces de Utilizador

Os elementos de texto que existem na interface da sua aplicação, como etiquetas de formulário, tooltips e texto explicativo, são tipicamente codificados nos templates ou componentes de UI, proporcionando uma experiência consistente mas genérica para todos os utilizadores. Usando padrões de geração de conteúdo contextual, pode transformar estes elementos estáticos em componentes dinâmicos, conscientes do contexto e personalizados.

Formulários Personalizados

Os formulários são uma parte ubíqua das aplicações web e móveis, servindo como o principal meio de recolha de dados dos utilizadores. No entanto, os formulários tradicionais frequentemente apresentam uma experiência genérica e impessoal, com etiquetas e campos padrão que nem sempre se alinham com o contexto ou necessidades específicas do utilizador. Os utilizadores têm maior probabilidade de preencher formulários que parecem adaptados às suas necessidades e preferências, levando a taxas de conversão e satisfação mais elevadas.

No entanto, é importante encontrar um equilíbrio entre personalização e consistência. Embora adaptar formulários a utilizadores individuais possa ser benéfico, é crucial manter um nível de familiaridade e previsibilidade. Os utilizadores devem ainda conseguir reconhecer e navegar facilmente nos formulários, mesmo com elementos personalizados.

Aqui estão algumas ideias de formulários personalizados para inspiração:

Sugestões Contextuais de Campos

A GenUI pode analisar as interações anteriores do utilizador, preferências e dados para fornecer sugestões inteligentes de campos como previsões. Por exemplo, se o

utilizador já inseriu anteriormente o seu endereço de envio, o formulário pode preencher automaticamente os campos relevantes com as suas informações guardadas. Isto não só poupa tempo, como também demonstra que a aplicação compreende e memoriza as preferências do utilizador.

Espera lá, esta técnica não poderia ser implementada sem recorrer à IA? Claro que sim, mas a beleza de implementar este tipo de funcionalidade com IA tem duas vertentes: 1) a facilidade de implementação e 2) a resiliência face às alterações e evolução da UI ao longo do tempo.

Vamos criar um serviço para o nosso sistema teórico de gestão de encomendas, que tente preencher proativamente a morada de envio correta para o utilizador.

```
1 class OrderShippingAddressSubscriber
2   include Raix::ChatCompletion
3
4   attr_accessor :order
5
6   delegate :customer, to: :order
7
8   DIRECTIVE = "You are a smart order processing assistant. Given the
9   customer's order history, guess the most likely shipping address
10  for the current order."
11
12  def order_created(order)
13    return unless order.pending? && order.shipping_address.blank?
14
15    self.order = order
16
17    transcript.clear
18    transcript << { system: DIRECTIVE }
19    transcript << { user: "Order History: #{order_history.to_json}" }
20    transcript << { user: "Current Order: #{order.to_json}" }
21
22    response = chat_completion
23    apply_predicted_shipping_address(order, response)
24  end
25
26  private
```

```
27
28 def apply_predicted_shipping_address(order, response)
29   # extract the shipping address from the response...
30   # ...and assume there's some sort of live update of the address fields
31   order.update(shipping_address:)
32 end
33
34 def order_history
35   customer.orders.successful.limit(100).map do |order|
36     {
37       date: order.date,
38       description: order.description,
39       shipping_address: order.shipping_address
40     }
41   end
42 end
43 end
```

Este exemplo é muito simplificado, mas deverá funcionar na maioria dos casos. A ideia é deixar a IA tentar adivinhar da mesma forma que um humano faria. Para exemplificar melhor o que estou a dizer, vamos considerar alguns dados de exemplo:

```
1 Order History:
2 [
3   {"date": "2024-01-03", "description": "garden soil mix",
4     "shipping_address": "123 Country Lane, Rural Town"},
5   {"date": "2024-01-15", "description": "hardcover fiction novels",
6     "shipping_address": "456 City Apt, Metroville"},
7   {"date": "2024-01-22", "description": "baby diapers", "shipping_address":
8     "789 Suburb St, Quietville"},
9   {"date": "2024-02-01", "description": "organic vegetables",
10    "shipping_address": "123 Country Lane, Rural Town"},
11  {"date": "2024-02-17", "description": "mystery thriller book set",
12    "shipping_address": "456 City Apt, Metroville"},
13  {"date": "2024-02-25", "description": "baby wipes",
14    "shipping_address": "789 Suburb St, Quietville"},
15  {"date": "2024-03-05", "description": "flower seeds",
16    "shipping_address": "123 Country Lane, Rural Town"},
17  {"date": "2024-03-20", "description": "biographies",
18    "shipping_address": "456 City Apt, Metroville"},
```

```
19  {"date": "2024-03-30", "description": "baby formula",
20    "shipping_address": "789 Suburb St, Quietville"},
21  {"date": "2024-04-12", "description": "lawn fertilizer",
22    "shipping_address": "123 Country Lane, Rural Town"},
23  {"date": "2024-04-22", "description": "science fiction novels",
24    "shipping_address": "456 City Apt, Metroville"},
25  {"date": "2024-05-02", "description": "infant toys",
26    "shipping_address": "789 Suburb St, Quietville"},
27  {"date": "2024-05-14", "description": "outdoor grill",
28    "shipping_address": "123 Country Lane, Rural Town"},
29  {"date": "2024-05-29", "description": "literary classics",
30    "shipping_address": "456 City Apt, Metroville"},
31  {"date": "2024-06-11", "description": "baby clothes",
32    "shipping_address": "789 Suburb St, Quietville"},
33  {"date": "2024-07-01", "description": "watering can",
34    "shipping_address": "123 Country Lane, Rural Town"},
35  {"date": "2024-07-18", "description": "non-fiction essays",
36    "shipping_address": "456 City Apt, Metroville"},
37  {"date": "2024-07-28", "description": "baby bath items",
38    "shipping_address": "789 Suburb St, Quietville"},
39  {"date": "2024-08-09", "description": "herb garden kit",
40    "shipping_address": "123 Country Lane, Rural Town"},
41  {"date": "2024-08-24", "description": "children's books",
42    "shipping_address": "456 City Apt, Metroville"}
43 ]
```

Reparaste no padrão nos dados? Garanto-te que isto é algo simples para um MLG. Para demonstrar, vamos perguntar ao GPT-4 qual é o endereço de envio mais provável para um “termómetro”.

1 From the order history you've provided, it looks like the purchases are
2 generally clustered into three main types based on the shipping addresses:
3
4 123 Country Lane, Rural Town - This address often orders garden and
5 outdoor-related items like soil mix, vegetables, flower seeds, lawn
6 fertilizer, an outdoor grill, a watering can, and an herb garden kit.
7
8 456 City Apt, Metroville - This address tends to order various types of
9 books, indicating an interest in reading across genres like fiction,
10 thrillers, biographies, science fiction, literary classics, non-fiction
11 essays, and children's books.
12
13 789 Suburb St, Quietville - The purchases for this address are consistently
14 baby-related products, such as diapers, wipes, baby formula, infant toys,
15 baby clothes, and baby bath items.
16
17 Given these patterns, a digital thermometer could potentially be a fit for
18 any of the addresses based on different needs: outdoor temperature tracking,
19 a tool for a passionate reader interested in science gadgets, or a health
20 tool for a baby. However, considering its common use in monitoring health,
21 especially for babies, it's most likely to be purchased by someone who's
22 already buying baby care products.
23
24 Therefore, the most likely shipping address for a digital thermometer, based
25 on the purchasing patterns, would be 789 Suburb St, Quietville.

Se estás a pensar que é um exagero usar um modelo dispendioso como o GPT-4 para esta tarefa, tens razão! Experimentei o mesmo prompt no Mistral 7B Instruct e este produziu a seguinte resposta a 75 tokens por segundo, e com um custo ínfimo de \$0.000218 USD.

```
1 Based on the given order history, it appears that the customer has received
2 orders for baby-related items (diapers, wipes, formula, clothes, bath items)
3 at the address "789 Suburb St, Quietville" and garden-related items (soil,
4 seeds, watering can, herb garden kit) at the address "123 Country Lane, Rural
5 Town". The customer has also received books (fiction, non-fiction, literary
6 classics, mystery-thriller, science fiction, biographies, children's books)
7 at the address "456 City Apt, Metroville".
8
9 Given that the new order is for a "thermometer", which doesn't seem to fit
10 into any of the categories that the customer has previously ordered, it's
11 hard to accurately predict the shipping address based solely on the order
12 history. However, given the 50%-50% split between baby-related and
13 garden-related items, it could somewhat lean towards the Baby-related items
14 address ("789 Suburb St, Quietville"). But remember, this is an assumption
15 and cannot be definitively confirmed without more context or information.
```

Valerá a pena o custo e o esforço desta técnica para tornar a experiência de checkout mais mágica? Para muitos retalhistas online, absolutamente. E pelo que parece, o custo da computação com IA só tende a diminuir, especialmente para fornecedores de hospedagem de modelos open source numa corrida até ao fundo.



Use um [Modelo de Prompt](#) e [StructuredIO](#) juntamente com [Delimitação de Respostas](#) para otimizar este tipo de conclusão de chat.

Ordenação Adaptativa de Campos

A ordem pela qual os campos do formulário são apresentados pode ter um impacto significativo na experiência do utilizador e nas taxas de conclusão. Com GenUI, pode ajustar dinamicamente a ordem dos campos com base no contexto do utilizador e na importância de cada campo. Por exemplo, se o utilizador estiver a preencher um formulário de registo para uma aplicação de fitness, o formulário poderia priorizar campos relacionados com os seus objetivos e preferências de fitness, tornando o processo mais relevante e envolvente.

Microtexto Personalizado

O texto instrucional, mensagens de erro e outro microtexto associado aos formulários também podem ser personalizados usando GenUI. Em vez de exibir mensagens de erro genéricas como “Endereço de email inválido”, pode gerar mensagens mais úteis e contextuais como “Por favor, introduza um endereço de email válido para receber a confirmação da sua encomenda”. Estes toques personalizados podem tornar a experiência do formulário mais amigável e menos frustrante.

Validação Personalizada

Na mesma linha do Microtexto Personalizado, poderia usar a IA para validar o formulário de formas que parecem mágicas. Imagine permitir que uma IA valide um formulário de perfil de utilizador, procurando possíveis erros a nível *semântico*.

Create your account

Full name

Obie Fernandez

Email

obiefernandez@gmail.com



Did you mean obiefernandez@gmail.com? [Yes, update.](#)

Country ⓘ

 United States



Password

.....



✓ Nice work. This is an excellent password.

Figura 9. Consegue identificar a validação semântica em ação?

Revelação Progressiva

A GenUI pode determinar de forma inteligente quais os campos do formulário que são essenciais com base no contexto do utilizador e revelar gradualmente campos adicionais conforme necessário. Esta técnica de revelação progressiva ajuda a reduzir a carga cognitiva e torna o processo de preenchimento do formulário mais gerível.

Por exemplo, se um utilizador estiver a registar-se para uma subscrição básica, o formulário pode inicialmente apresentar apenas os campos essenciais, e à medida que o utilizador progride ou seleciona opções específicas, campos adicionais relevantes podem ser introduzidos dinamicamente.

Texto Explicativo Sensível ao Contexto

As dicas de contexto são frequentemente utilizadas para fornecer informações adicionais ou orientação aos utilizadores quando passam o rato por cima ou interagem com elementos específicos. Com uma abordagem de “Geração de Conteúdo Contextual”, pode gerar dicas de contexto que se adaptam ao contexto do utilizador e fornecem informações relevantes. Por exemplo, se um utilizador estiver a explorar uma funcionalidade complexa, a dica de contexto pode oferecer sugestões personalizadas ou exemplos baseados nas suas interações anteriores ou nível de competência.

O texto explicativo, como instruções, descrições ou mensagens de ajuda, pode ser gerado dinamicamente com base no contexto do utilizador. Em vez de apresentar explicações genéricas, pode usar LLMs para gerar texto que é adaptado às necessidades ou questões específicas do utilizador. Por exemplo, se um utilizador estiver com dificuldades num determinado passo do processo, o texto explicativo pode fornecer orientação personalizada ou dicas de resolução de problemas.

Microtexto refere-se aos pequenos fragmentos de texto que guiam os utilizadores através da sua aplicação, como etiquetas de botões, mensagens de erro ou prompts de confirmação. Ao aplicar a abordagem de [Geração de Conteúdo Contextual](#) ao microtexto, pode criar uma interface adaptativa que responde às ações do utilizador e fornece texto relevante e útil. Por exemplo, se um utilizador estiver prestes a realizar uma ação crítica, o prompt de confirmação pode ser gerado dinamicamente para fornecer uma mensagem clara e personalizada.

O texto explicativo e as dicas de contexto personalizadas podem melhorar significativamente o processo de integração para novos utilizadores. Ao fornecer

orientação e exemplos específicos ao contexto, pode ajudar os utilizadores a compreender e navegar rapidamente na aplicação, reduzindo a curva de aprendizagem e aumentando a adoção.

Os elementos de interface dinâmicos e sensíveis ao contexto também podem tornar a aplicação mais intuitiva e envolvente. Os utilizadores têm maior probabilidade de interagir e explorar funcionalidades quando o texto que as acompanha é adaptado às suas necessidades e interesses específicos.

Até agora abordámos ideias para melhorar os paradigmas existentes de UI com IA, mas e se repensássemos de forma mais radical como as interfaces do utilizador são desenhadas e implementadas?

Definindo a UI Generativa

Ao contrário do design de UI tradicional, onde os designers criam interfaces fixas e estáticas, a GenUI sugere um futuro em que o nosso software apresenta experiências flexíveis e personalizadas que podem evoluir e adaptar-se em tempo real. Sempre que utilizamos uma interface conversacional baseada em IA, estamos a permitir que a IA se adapte às necessidades particulares do utilizador. A GenUI vai um passo além ao aplicar esse nível de adaptabilidade à interface *visual* do software.

A razão pela qual é possível experimentar com ideias de GenUI hoje é que os modelos de linguagem grandes já compreendem programação e o seu conhecimento base inclui tecnologias e frameworks de UI. A questão é, portanto, se os modelos de linguagem grandes podem ser utilizados para gerar elementos de UI, como texto, imagens, layouts e até interfaces completas, que são adaptadas a cada utilizador individual. O modelo poderia ser instruído para ter em conta vários fatores, como as interações anteriores do utilizador, preferências declaradas, informações demográficas e o contexto atual de utilização, para criar interfaces altamente personalizadas e relevantes.

A GenUI difere do design de interface do utilizador tradicional em vários aspetos fundamentais:

1. **Dinâmica e Adaptativa:** O design de UI tradicional envolve a criação de interfaces fixas e estáticas que permanecem iguais para todos os utilizadores. Em contraste, a GenUI permite interfaces que se podem adaptar e mudar dinamicamente com base nas necessidades e contexto do utilizador. Isto significa que a mesma aplicação pode apresentar interfaces diferentes para diferentes utilizadores ou até para o mesmo utilizador em situações diferentes.
2. **Personalização em Escala:** Com o design tradicional, criar experiências personalizadas para cada utilizador é frequentemente impraticável devido ao tempo e recursos necessários. A GenUI, por outro lado, permite a personalização em escala. Ao aproveitar a IA, os designers podem criar interfaces que se adaptam automaticamente às necessidades e preferências únicas de cada utilizador, sem ter de desenhar e desenvolver manualmente interfaces separadas para cada segmento de utilizadores.
3. **Foco nos Resultados:** O design de UI tradicional frequentemente concentra-se em criar interfaces visualmente atraentes e funcionais. Embora estes aspetos ainda sejam importantes na GenUI, o foco principal muda para alcançar os resultados desejados pelo utilizador. A GenUI visa criar interfaces que são otimizadas para os objetivos e tarefas específicos de cada utilizador, priorizando a usabilidade e eficácia sobre considerações puramente estéticas.
4. **Aprendizagem e Melhoria Contínua:** Os sistemas GenUI podem aprender e melhorar continuamente ao longo do tempo com base nas interações e feedback dos utilizadores. À medida que os utilizadores interagem com as interfaces geradas, os modelos de IA podem recolher dados sobre o comportamento, preferências e resultados dos utilizadores, usando esta informação para refinar e otimizar futuras gerações de interfaces. Este processo de aprendizagem iterativo permite que os sistemas GenUI se tornem cada vez mais eficazes em atender às necessidades dos utilizadores ao longo do tempo.

É importante notar que a GenUI não é o mesmo que ferramentas de design assistidas por IA, como aquelas que fornecem sugestões ou automatizam certas tarefas de design. Embora estas ferramentas possam ser úteis para agilizar o processo de design, elas ainda dependem dos designers para tomar decisões finais e criar interfaces estáticas. A GenUI, por outro lado, envolve o sistema de IA assumindo um papel mais ativo na geração e adaptação real de interfaces com base nos dados e contexto do utilizador.

A GenUI representa uma mudança significativa na forma como abordamos o design de interface do utilizador, afastando-se de soluções únicas para todos e aproximando-se de experiências altamente personalizadas e adaptativas. Ao aproveitar o poder da IA, a GenUI tem o potencial de revolucionar a forma como interagimos com produtos e serviços digitais, criando interfaces que são mais intuitivas, envolventes e eficazes para cada utilizador individual.

Exemplo

Para ilustrar o conceito de GenUI, vamos considerar uma aplicação hipotética de fitness chamada “FitAI”. Esta aplicação visa fornecer planos de treino personalizados e conselhos nutricionais aos utilizadores com base nos seus objetivos individuais, níveis de fitness e preferências.

Numa abordagem tradicional de design de UI, o FitAI poderia ter um conjunto fixo de ecrãs e elementos que são os mesmos para todos os utilizadores. No entanto, com a GenUI, a interface da aplicação poderia adaptar-se dinamicamente às necessidades e contexto únicos de cada utilizador.

Esta abordagem é um pouco difícil de imaginar implementar em 2024 e pode nem ter um ROI adequado, mas é possível.

Eis como poderia funcionar:

1. **Integração:**

- Em vez de um questionário padrão, o FitAI usa uma IA conversacional para recolher informações sobre os objetivos do utilizador, nível atual de fitness e preferências.
- Com base nesta interação inicial, a IA gera um layout de painel personalizado, destacando as funcionalidades e informações mais relevantes para os objetivos do utilizador.
- A tecnologia de IA atual pode ter uma seleção de componentes de ecrã à sua disposição para usar na composição do painel personalizado.
- A tecnologia de IA futura pode assumir o papel de um designer de UI experiente e realmente criar o painel *do zero*.

2. Planificador de Treinos:

- A interface do planificador de treinos é adaptada pela IA especificamente para corresponder ao nível de experiência do utilizador e ao equipamento disponível.
- Para um iniciante sem equipamento, pode mostrar exercícios simples com o peso corporal, com instruções detalhadas e vídeos.
- Para um utilizador avançado com acesso a um ginásio, pode apresentar rotinas mais complexas com menos conteúdo explicativo.
- O conteúdo do planificador de treinos não é simplesmente filtrado de um grande conjunto. Pode ser gerado *instantaneamente* com base numa base de conhecimento que é consultada com um contexto que inclui tudo o que se sabe sobre o utilizador.

3. Monitorização de Progresso:

- A interface de monitorização de progresso evolui com base nos objetivos e padrões de envolvimento do utilizador.
- Se um utilizador está principalmente focado na perda de peso, a interface pode exibir de forma proeminente um gráfico de tendência de peso e estatísticas de queima de calorias.

- Para um utilizador a desenvolver massa muscular, pode destacar ganhos de força e alterações na composição corporal.
- A IA pode adaptar esta parte da aplicação ao progresso real do utilizador. Se o progresso parar durante um período de tempo, a aplicação pode mudar para um modo onde tenta persuadir o utilizador a revelar as razões do retrocesso, de forma a mitigá-las.

4. Aconselhamento Nutricional:

- A secção de nutrição adapta-se às preferências e restrições alimentares do utilizador.
- Para um utilizador vegano, pode mostrar sugestões de refeições à base de plantas e fontes de proteína.
- Para um utilizador com intolerância ao glúten, filtraria automaticamente alimentos com glúten das recomendações.
- Mais uma vez, o conteúdo não é retirado de um enorme conjunto de dados de refeições aplicável a todos os utilizadores, mas sim sintetizado a partir de uma base de conhecimento que contém informações adaptáveis com base na situação específica e restrições do utilizador.
- Por exemplo, as receitas são geradas com especificações de ingredientes que correspondem às necessidades calóricas em constante mudança do utilizador à medida que o seu nível de fitness e estatísticas corporais evoluem.

5. Elementos Motivacionais:

- O conteúdo motivacional e as notificações da aplicação são personalizados com base no tipo de personalidade do utilizador e na resposta a diferentes estratégias motivacionais.
- Alguns utilizadores podem receber mensagens encorajadoras, enquanto outros recebem feedback mais orientado a dados.

Neste exemplo, a GenUI permite que a FitAI crie uma experiência altamente personalizada para cada utilizador, potencialmente aumentando o envolvimento, a satisfação e a probabilidade de alcançar objetivos de fitness. Os elementos da interface, o conteúdo e até mesmo a “personalidade” da aplicação adaptam-se para melhor servir as necessidades e preferências de cada utilizador individual.

A Mudança para o Design Orientado a Resultados

A GenUI representa uma mudança fundamental na abordagem ao design de interface do utilizador, passando de um foco na criação de elementos específicos de interface para uma abordagem mais holística e orientada a resultados. Esta mudança tem várias implicações importantes:

1. Foco nos Objetivos do Utilizador:

- Os designers precisarão de pensar mais profundamente sobre os objetivos do utilizador e os resultados desejados, em vez de componentes específicos da interface.
- A ênfase será na criação de sistemas que possam gerar interfaces que ajudem os utilizadores a alcançar os seus objetivos de forma eficiente e eficaz.
- Surgirão novas estruturas de UI que fornecerão aos designers baseados em IA as ferramentas necessárias para gerar experiências de utilizador *instantaneamente* e *do zero*, em vez de baseadas em especificações predefinidas de ecrã.

2. Mudança no Papel dos Designers:

- Os designers farão a transição da criação de layouts fixos para a definição de regras, restrições e diretrizes que os sistemas de IA devem seguir ao gerar interfaces.

- Precisarão de desenvolver competências em áreas como análise de dados, engenharia de prompts de IA e pensamento sistêmico para orientar eficazmente os sistemas GenUI.

3. Importância da Pesquisa do Utilizador:

- A pesquisa do utilizador torna-se ainda mais crítica num contexto GenUI, pois os designers precisam de compreender não apenas as preferências do utilizador, mas também como estas preferências e necessidades mudam em diferentes contextos.
- Testes contínuos com utilizadores e ciclos de feedback serão essenciais para refinar e melhorar a capacidade da IA de gerar interfaces eficazes.

4. Design para Variabilidade:

- Em vez de criar uma única interface “perfeita”, os designers precisarão de considerar múltiplas variações possíveis e garantir que o sistema possa gerar interfaces apropriadas para diversas necessidades dos utilizadores.
- Isto inclui projetar para casos extremos e garantir que as interfaces geradas mantêm a usabilidade e acessibilidade em diferentes configurações.
- A diferenciação de produtos assume novas dimensões envolvendo perspetivas divergentes sobre psicologia do utilizador e o aproveitamento de conjuntos de dados únicos e bases de conhecimento indisponíveis para os concorrentes.

Desafios e Considerações

Embora a GenUI ofereça possibilidades empolgantes, também apresenta vários desafios e considerações:

1. Limitações Técnicas:

- A tecnologia de IA atual, embora avançada, ainda tem limitações na compreensão de intenções complexas do utilizador e na geração de interfaces verdadeiramente conscientes do contexto.
- Problemas de desempenho relacionados com a geração em tempo real de elementos de interface, especialmente em dispositivos menos potentes.

2. Requisitos de Dados:

- Dependendo do caso de uso, os sistemas GenUI eficazes podem necessitar de quantidades significativas de dados dos utilizadores para gerar interfaces personalizadas.
- Os desafios na obtenção ética de dados autênticos dos utilizadores levantam preocupações sobre a privacidade e segurança dos dados, bem como potenciais enviesamentos nos dados utilizados para treinar modelos GenUI.

3. Usabilidade e Consistência:

- Pelo menos até que a prática se torne generalizada, uma aplicação com interfaces em constante mudança pode levar a problemas de usabilidade, pois os utilizadores podem ter dificuldade em encontrar elementos familiares ou navegar eficientemente.
- Será crucial encontrar um equilíbrio entre a personalização e a manutenção de uma interface consistente e fácil de aprender.

4. Dependência Excessiva da IA:

- Existe o risco de delegar excessivamente as decisões de design aos sistemas de IA, potencialmente levando a escolhas de interface sem inspiração, problemáticas ou simplesmente defeituosas.
- A supervisão humana e a capacidade de substituir designs gerados por IA continuarão a ser importantes num futuro próximo.

5. Preocupações de Acessibilidade:

- Garantir que as interfaces geradas dinamicamente permaneçam acessíveis a utilizadores com deficiência apresenta desafios completamente novos, o que é preocupante dado o baixo nível de conformidade com a acessibilidade demonstrado pelos sistemas típicos.
- Por outro lado, os designers de IA podem ser implementados com preocupação *incorporada* pela acessibilidade e capacidades para construir interfaces acessíveis instantaneamente, tal como constroem UI para utilizadores sem limitações.
- De qualquer forma, os sistemas GenUI devem ser projetados com diretrizes e processos de teste de acessibilidade robustos.

6. Confiança e Transparência do Utilizador:

- Os utilizadores podem sentir-se desconfortáveis com interfaces que parecem “saber demais” sobre eles ou que mudam de formas que não compreendem.
- Fornecer transparência sobre como e porquê as interfaces são personalizadas será importante para construir a confiança do utilizador.

Perspetivas Futuras e Oportunidades

O futuro da Interface Generativa (GenUI) tem uma promessa imensa para revolucionar a forma como interagimos com produtos e serviços digitais. À medida que esta tecnologia continua a evoluir, podemos antecipar uma mudança sísmica na forma como as interfaces do utilizador são projetadas, implementadas e experienciadas. Penso que a GenUI é o fenómeno que finalmente empurrará o nosso software para o domínio do que é agora considerado ficção científica.

Uma das perspetivas mais empolgantes da GenUI é o seu potencial para melhorar a acessibilidade numa escala grandiosa que vai além de simplesmente garantir que pessoas com deficiências graves não sejam completamente excluídas do uso do

software. Ao adaptar automaticamente as interfaces às necessidades individuais dos utilizadores, a GenUI poderia tornar as experiências digitais mais inclusivas do que nunca. Imagine interfaces que se ajustam perfeitamente para fornecer texto maior para utilizadores mais jovens ou com deficiência visual, ou layouts simplificados para aqueles com deficiências cognitivas, tudo sem necessitar de configuração manual ou versões “acessíveis” separadas das aplicações.

As capacidades de personalização da GenUI provavelmente impulsionarão o aumento do envolvimento, satisfação e lealdade dos utilizadores numa ampla gama de produtos digitais. À medida que as interfaces se tornam mais sintonizadas com as preferências e comportamentos individuais, os utilizadores encontrarão experiências digitais mais intuitivas e agradáveis, potencialmente levando a interações mais profundas e significativas com a tecnologia.

A GenUI também tem o potencial de transformar o processo de integração para novos utilizadores. Ao criar experiências intuitivas e personalizadas para utilizadores iniciantes que se adaptam rapidamente ao nível de experiência de cada utilizador, a GenUI poderia reduzir significativamente a curva de aprendizagem associada a novas aplicações. Isto poderia levar a taxas de adoção mais rápidas e maior confiança dos utilizadores na exploração de novos recursos e funcionalidades.

Outra possibilidade empolgante é a capacidade da GenUI de manter uma experiência de utilizador consistente em diferentes dispositivos e plataformas, enquanto otimiza para cada contexto específico de uso. Isto poderia resolver o desafio de longa data de fornecer experiências coerentes num panorama de dispositivos cada vez mais fragmentado, desde smartphones e tablets até computadores de secretária e tecnologias emergentes como óculos de realidade aumentada.

A natureza orientada a dados da GenUI abre oportunidades para iteração rápida e melhoria no design de UI. Ao recolher dados em tempo real sobre como os utilizadores interagem com interfaces geradas, designers e programadores podem obter insights sem precedentes sobre o comportamento e as preferências dos utilizadores. Este ciclo

de feedback poderia levar a melhorias contínuas no design de UI, impulsionadas por padrões de uso reais em vez de suposições ou testes limitados com utilizadores.

Para se preparar para esta mudança, os designers precisarão de evoluir as suas competências e mentalidades. O foco mudará da criação de layouts fixos para o desenvolvimento de sistemas de design e diretrizes abrangentes que possam informar a geração de interfaces conduzida por IA. Os designers precisarão de cultivar uma compreensão profunda de análise de dados, tecnologias de IA e pensamento sistémico para orientar eficazmente os sistemas GenUI.

Além disso, à medida que a GenUI esbate as linhas entre design e tecnologia, os designers precisarão de colaborar mais estreitamente com programadores e cientistas de dados. Esta abordagem interdisciplinar será crucial na criação de sistemas GenUI que não são apenas visualmente atraentes e fáceis de usar, mas também tecnicamente robustos e eticamente sólidos.

As implicações éticas do GenUI também virão para primeiro plano à medida que a tecnologia amadurece. Os designers terão um papel crucial no desenvolvimento de estruturas para o uso responsável da IA no design de interface, garantindo que a personalização melhore as experiências do utilizador sem comprometer a privacidade ou manipular o comportamento do utilizador de formas não éticas.

Ao olharmos para o futuro, o GenUI apresenta tanto oportunidades empolgantes como desafios significativos. Tem o potencial de criar experiências digitais mais intuitivas, eficientes e satisfatórias para utilizadores em todo o mundo. Embora exija que os designers se adaptem e adquiram novas competências, também oferece uma oportunidade sem precedentes para moldar o futuro da interação humano-computador de formas profundas e significativas. O percurso em direção a sistemas GenUI plenamente realizados será, sem dúvida, complexo, mas as potenciais recompensas em termos de melhoria das experiências do utilizador e acessibilidade digital tornam-no num futuro pelo qual vale a pena lutar.

Orquestração Inteligente de Fluxos de Trabalho



No domínio do desenvolvimento de aplicações, os fluxos de trabalho desempenham um papel crucial na definição de como as tarefas, processos e interações do utilizador são estruturados e executados. À medida que as aplicações se tornam mais complexas e as expectativas dos utilizadores continuam a aumentar, a necessidade de uma orquestração inteligente e adaptativa dos fluxos de trabalho torna-se cada vez mais evidente.

A abordagem de “Orquestração Inteligente de Fluxos de Trabalho” concentra-se em aproveitar componentes de IA para orquestrar e otimizar dinamicamente fluxos de trabalho complexos dentro das aplicações. O objetivo é criar aplicações mais eficientes, responsivas e adaptáveis a dados e contextos em tempo real.

Neste capítulo, iremos explorar os principais princípios e padrões que sustentam a abordagem de orquestração inteligente de fluxos de trabalho. Consideraremos como

a IA pode ser utilizada para encaminhar tarefas de forma inteligente, automatizar a tomada de decisões e adaptar dinamicamente os fluxos de trabalho com base em vários fatores, como o comportamento do utilizador, o desempenho do sistema e as regras de negócio. Através de exemplos práticos e cenários do mundo real, demonstraremos o potencial transformador da IA na simplificação e otimização dos fluxos de trabalho das aplicações.

Quer esteja a desenvolver aplicações empresariais com processos de negócio complexos ou aplicações voltadas para o consumidor com percursos dinâmicos do utilizador, os padrões e técnicas discutidos neste capítulo irão equipá-lo com o conhecimento e as ferramentas necessárias para criar fluxos de trabalho inteligentes e eficientes que melhoram a experiência geral do utilizador e impulsionam o valor do negócio.

Necessidade de Negócio

As abordagens tradicionais à gestão de fluxos de trabalho frequentemente dependem de regras predefinidas e árvores de decisão estáticas, que podem ser rígidas, inflexíveis e incapazes de lidar com a natureza dinâmica das aplicações modernas.

Considere um cenário em que uma aplicação de comércio eletrónico precisa de gerir um processo complexo de processamento de encomendas. O fluxo de trabalho pode envolver múltiplos passos, como validação da encomenda, verificação de inventário, processamento de pagamento, envio e notificações ao cliente. Cada passo pode ter o seu próprio conjunto de regras, dependências, integrações externas e mecanismos de tratamento de exceções. Gerir um fluxo de trabalho deste tipo manualmente ou através de lógica codificada pode rapidamente tornar-se complicado, propenso a erros e difícil de manter.

Além disso, à medida que a aplicação escala e o número de utilizadores simultâneos cresce, o fluxo de trabalho pode precisar de se adaptar e otimizar com base em dados em tempo real e no desempenho do sistema. Por exemplo, durante períodos de pico de

tráfego, a aplicação pode precisar de ajustar dinamicamente o fluxo de trabalho para priorizar certas tarefas, alocar recursos de forma eficiente e garantir uma experiência de utilizador suave.

É aqui que entra a abordagem de “Orquestração Inteligente de Fluxos de Trabalho”. Ao aproveitar componentes de IA, os programadores podem criar fluxos de trabalho que são inteligentes, adaptativos e auto-otimizáveis. A IA pode analisar grandes quantidades de dados, aprender com experiências passadas e tomar decisões informadas em tempo real para orquestrar o fluxo de trabalho de forma eficaz.

Principais Benefícios

1. **Maior Eficiência:** A IA pode otimizar a alocação de tarefas, a utilização de recursos e a execução do fluxo de trabalho, levando a tempos de processamento mais rápidos e uma maior eficiência geral.
2. **Adaptabilidade:** Os fluxos de trabalho baseados em IA podem adaptar-se dinamicamente a condições em mudança, como flutuações na procura dos utilizadores, desempenho do sistema ou requisitos de negócio, garantindo que a aplicação permanece responsiva e resiliente.
3. **Tomada de Decisão Automatizada:** A IA pode automatizar processos complexos de tomada de decisão dentro do fluxo de trabalho, reduzindo a intervenção manual e minimizando o risco de erros humanos.
4. **Personalização:** A IA pode analisar o comportamento, preferências e contexto do utilizador para personalizar o fluxo de trabalho e fornecer experiências adaptadas a utilizadores individuais.
5. **Escalabilidade:** Os fluxos de trabalho baseados em IA podem escalar perfeitamente para lidar com volumes crescentes de dados e interações de utilizadores, sem comprometer o desempenho ou a fiabilidade.

Nas secções seguintes, iremos explorar os principais padrões e técnicas que permitem a

implementação de fluxos de trabalho inteligentes e mostrar exemplos do mundo real de como a IA está a transformar a gestão de fluxos de trabalho em aplicações modernas.

Padrões Principais

Para implementar a orquestração inteligente de fluxos de trabalho em aplicações, os programadores podem aproveitar vários padrões principais que aproveitam o poder da IA. Estes padrões fornecem uma abordagem estruturada para desenhar e gerir fluxos de trabalho, permitindo que as aplicações se adaptem, otimizem e automatizem processos com base em dados e contexto em tempo real. Vamos explorar alguns dos padrões fundamentais na orquestração inteligente de fluxos de trabalho.

Encaminhamento Dinâmico de Tarefas

Este padrão envolve o uso de IA para encaminhar tarefas de forma inteligente dentro de um fluxo de trabalho com base em vários fatores, como prioridade da tarefa, disponibilidade de recursos e desempenho do sistema. Os algoritmos de IA podem analisar as características de cada tarefa, considerar o estado atual do sistema e tomar decisões informadas para atribuir tarefas aos recursos ou caminhos de processamento mais apropriados. O encaminhamento dinâmico de tarefas garante que as tarefas são distribuídas e executadas de forma eficiente, otimizando o desempenho geral do fluxo de trabalho.

```

1  class TaskRouter
2    include Raix::ChatCompletion
3    include Raix::FunctionDispatch
4
5    attr_accessor :task
6
7    # list of functions that can be called by the AI entirely at its
8    # discretion depending on the task received
9
10   function :analyze_task_priority do
11     TaskPriorityAnalyzer.perform(task)
12   end
13
14   function :check_resource_availability, # ...
15   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

```

Note o ciclo criado pela expressão `while` na linha 29, que continua a solicitar a IA até que a tarefa seja atribuída. Na linha 35, guardamos o registo da tarefa para análise e

depuração posterior, caso seja necessário.

Tomada de Decisão Contextual

Pode utilizar código muito semelhante para tomar decisões contextuais dentro de um fluxo de trabalho. Ao analisar pontos de dados relevantes, como preferências do utilizador, padrões históricos e entradas em tempo real, os componentes de IA podem determinar o curso de ação mais apropriado em cada ponto de decisão do fluxo de trabalho. Adapte o comportamento do seu fluxo de trabalho com base no contexto específico de cada utilizador ou cenário, proporcionando experiências personalizadas e otimizadas.

Composição Adaptativa de Fluxos de Trabalho

Este padrão concentra-se na composição e ajuste dinâmico de fluxos de trabalho com base em requisitos ou condições em mudança. A IA pode analisar o estado atual do fluxo de trabalho, identificar pontos de estrangulamento ou ineficiências, e modificar automaticamente a estrutura do fluxo de trabalho para otimizar o desempenho. A composição adaptativa de fluxos de trabalho permite que as aplicações evoluam e melhorem continuamente os seus processos sem necessitar de intervenção manual.

Tratamento e Recuperação de Exceções

O tratamento e recuperação de exceções são aspetos críticos da orquestração inteligente de fluxos de trabalho. Ao trabalhar com componentes de IA e fluxos de trabalho complexos, é essencial antecipar e tratar as exceções de forma elegante para garantir a estabilidade e fiabilidade do sistema.

Aqui estão algumas considerações e técnicas importantes para o tratamento e recuperação de exceções em fluxos de trabalho inteligentes:

1. **Propagação de Exceções:** Implemente uma abordagem consistente para propagar exceções entre componentes do fluxo de trabalho. Quando ocorre uma exceção dentro de um componente, esta deve ser capturada, registrada e propagada para o orquestrador ou para um componente discreto responsável pelo tratamento de exceções. A ideia é centralizar o tratamento de exceções e evitar que as exceções sejam silenciosamente suprimidas, além de abrir possibilidades para o [Tratamento Inteligente de Erros](#).
2. **Mecanismos de Nova Tentativa:** Os mecanismos de nova tentativa ajudam a melhorar a resiliência do fluxo de trabalho e a lidar com falhas intermitentes de forma elegante. Procure definitivamente implementar mecanismos de nova tentativa para exceções transitórias ou recuperáveis, como problemas de conectividade de rede ou indisponibilidade de recursos que podem ser automaticamente repetidos após um atraso especificado. Ter um orquestrador ou gestor de exceções baseado em IA significa que as suas estratégias de nova tentativa não precisam de ser mecânicas por natureza, dependendo de algoritmos fixos como o recuo exponencial. Pode deixar o tratamento da nova tentativa ao “critério” do componente de IA responsável por decidir como lidar com a exceção.
3. **Estratégias de Contingência:** Se um componente de IA falhar em fornecer uma resposta válida ou encontrar um erro—uma ocorrência comum dada a sua natureza de ponta—tenha um mecanismo de contingência em vigor para garantir que o fluxo de trabalho possa continuar. Isto pode envolver o uso de valores predefinidos, algoritmos alternativos ou um [Humano no Circuito](#) para tomar decisões e manter o fluxo de trabalho em movimento.
4. **Ações Compensatórias:** As diretivas do orquestrador devem incluir instruções sobre ações compensatórias para lidar com exceções que não podem ser resolvidas automaticamente. Ações compensatórias são passos tomados para desfazer ou mitigar os efeitos de uma operação falhada. Por exemplo, se um passo de processamento de pagamento falhar, uma ação compensatória poderia ser reverter a transação e notificar o utilizador. As ações compensatórias ajudam a manter a

consistência e integridade dos dados face a exceções.

5. **Monitorização e Alerta de Exceções:** Configure mecanismos de monitorização e alerta para detetar e notificar as partes interessadas relevantes sobre exceções críticas. O orquestrador pode ser configurado com limites e regras para acionar alertas quando as exceções excedem certos limites ou quando ocorrem tipos específicos de exceções. Isto permite a identificação e resolução proativa de problemas antes que afetem o sistema como um todo.

Aqui está um exemplo de tratamento e recuperação de exceções num componente de fluxo de trabalho em Ruby:

```
1 class InventoryManager
2   def check_availability(order)
3     begin
4       # Perform inventory check logic
5       inventory = Inventory.find_by(product_id: order.product_id)
6       if inventory.available_quantity >= order.quantity
7         return true
8       else
9         raise InsufficientInventoryError,
10            "Insufficient inventory for product #{order.product_id}"
11       end
12     rescue InsufficientInventoryError => e
13       # Log the exception
14       logger.error("Inventory check failed: #{e.message}")
15
16       # Retry the operation after a delay
17       retry_count ||= 0
18       if retry_count < MAX_RETRIES
19         retry_count += 1
20         sleep(RETRY_DELAY)
21         retry
22       else
23         # Fallback to manual intervention
24         NotificationService.admin("Inventory check failed: Order #{order.id}")
25         return false
26       end
27     end
28   end
29 end
```

```
28     end  
29 end
```

Neste exemplo, o componente `InventoryManager` verifica a disponibilidade de um produto para um determinado pedido. Se a quantidade disponível for insuficiente, lança um `InsufficientInventoryError`. A exceção é capturada, registrada e é implementado um mecanismo de nova tentativa. Se o limite de tentativas for excedido, o componente recorre à intervenção manual notificando um administrador.

Ao implementar mecanismos robustos de tratamento e recuperação de exceções, pode garantir que os seus fluxos de trabalho inteligentes são resilientes, sustentáveis e capazes de lidar com situações inesperadas de forma elegante.

Estes padrões formam a base da orquestração de fluxos de trabalho inteligentes e podem ser combinados e adaptados para atender aos requisitos específicos de diferentes aplicações. Ao aproveitar estes padrões, os programadores podem criar fluxos de trabalho que são flexíveis, resilientes e otimizados para desempenho e experiência do utilizador.

Na próxima secção, vamos explorar como estes padrões podem ser implementados na prática, utilizando exemplos do mundo real e trechos de código para ilustrar a integração de componentes de IA na gestão de fluxos de trabalho.

Implementação Prática da Orquestração de Fluxos de Trabalho Inteligentes

Agora que explorámos os padrões principais na orquestração de fluxos de trabalho inteligentes, vamos mergulhar em como estes padrões podem ser implementados em aplicações do mundo real. Forneceremos exemplos práticos e trechos de código para ilustrar a integração de componentes de IA na gestão de fluxos de trabalho.

Processador de Pedidos Inteligente

Vamos mergulhar num exemplo prático de implementação de orquestração de fluxo de trabalho inteligente usando um componente `OrderProcessor` baseado em IA numa aplicação de e-commerce em Ruby on Rails. O `OrderProcessor` implementa o conceito de [Process Manager Enterprise Integration](#) que encontramos pela primeira vez no Capítulo 3 ao discutir [Multitude of Workers](#). O componente será responsável por gerir o fluxo de trabalho de processamento de pedidos, tomar decisões de encaminhamento com base em resultados intermediários e orquestrar a execução de várias etapas de processamento.

O processo de processamento de pedidos envolve múltiplas etapas, como validação do pedido, verificação de inventário, processamento de pagamento e envio. Cada etapa é implementada como um processo worker separado que executa uma tarefa específica e retorna o resultado para o `OrderProcessor`. As etapas não são obrigatórias e nem precisam necessariamente ser realizadas numa ordem precisa.

Aqui está um exemplo de implementação do `OrderProcessor`. Ele apresenta dois mixins do [Raix](#). O primeiro (`ChatCompletion`) confere-lhe a capacidade de fazer conclusão de chat, o que o torna um componente de IA. O segundo (`FunctionDispatch`) permite a chamada de funções pela IA, permitindo que responda a um prompt com uma invocação de função em vez de uma mensagem de texto.

As funções worker (`validate_order`, `check_inventory`, et al) delegam para as suas respetivas classes worker, que podem ser componentes de IA ou não-IA, com o único requisito sendo que retornem os resultados do seu trabalho num formato que possa ser representado como uma string.



Tal como com todos os outros exemplos nesta parte do livro, este código é praticamente pseudo-código e destina-se apenas a transmitir o significado do padrão e inspirar as suas próprias criações. Descrições completas dos padrões e exemplos de código completos estão incluídos na Parte 2.


```
1  class OrderProcessor
2    include Raix::ChatCompletion
3    include Raix::FunctionDispatch
4
5    SYSTEM_DIRECTIVE = "You are an order processor, tasked with..."
6
7    def initialize(order)
8      self.order = order
9      transcript << { system: SYSTEM_DIRECTIVE }
10     transcript << { user: order.to_json }
11   end
12
13   def perform
14     # will continue looping until `stop_looping!` is called
15     chat_completion(loop: true)
16   end
17
18   # list of functions available to be called by the AI
19   # truncated for brevity
20
21   def functions
22     [
23       {
24         name: "validate_order",
25         description: "Invoke to check validity of order",
26         parameters: {
27           ...
28         },
29         ...
30     ]
31   end
32
33   # implementation of functions that can be called by the AI
34   # entirely at its discretion, depending on the needs of the order
35
36   def validate_order
37     OrderValidationWorker.perform(@order)
38   end
39
40   def check_inventory
41     InventoryCheckWorker.perform(@order)
42   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
```

No exemplo, o `OrderProcessor` é inicializado com um objeto de encomenda e mantém uma transcrição da execução do fluxo de trabalho, no formato típico de transcrição de conversa que é nativo dos modelos de linguagem grandes. É dado controlo total à IA para orquestrar a execução de várias etapas de processamento, como validação de encomendas, verificação de inventário, processamento de pagamentos e envio.

Sempre que o método `chat_completion` é chamado, a transcrição é enviada à IA para que esta forneça uma conclusão como uma chamada de função. Cabe inteiramente à IA analisar o resultado da etapa anterior e determinar a ação apropriada a tomar. Por exemplo, se a verificação de inventário revelar níveis baixos de stock, o `OrderProcessor` pode agendar uma tarefa de reabastecimento. Se o processamento do pagamento falhar, pode iniciar uma nova tentativa ou notificar o apoio ao cliente.

O exemplo acima não tem funções definidas para reabastecimento ou notificação do apoio ao cliente, mas poderia perfeitamente tê-las.

A transcrição cresce sempre que uma função é chamada e serve como registo da execução do fluxo de trabalho, incluindo os resultados de cada etapa e as instruções geradas pela IA para as etapas seguintes. Esta transcrição pode ser utilizada para depuração, auditoria e fornecimento de visibilidade sobre o processo de cumprimento de encomendas.

Ao aproveitar a IA no `OrderProcessor`, a aplicação de comércio eletrónico pode adaptar dinamicamente o fluxo de trabalho com base em dados em tempo real e lidar com exceções de forma inteligente. O componente de IA pode tomar decisões informadas, otimizar o fluxo de trabalho e garantir um processamento suave das encomendas, mesmo em cenários complexos.

O facto de que o único requisito para os processos de trabalho é retornar algum resultado inteligível para a IA considerar ao decidir o que fazer a seguir, pode começar a fazer-nos perceber como esta abordagem pode reduzir o trabalho de mapeamento de entrada/saída que normalmente está envolvido na integração de sistemas díspares entre si.

Moderador de Conteúdo Inteligente

As aplicações de redes sociais geralmente requerem pelo menos uma moderação mínima de conteúdo para garantir uma comunidade segura e saudável. Este exemplo do componente `ContentModerator` aproveita a IA para orquestrar de forma inteligente o fluxo de trabalho de moderação, tomando decisões com base nas características do conteúdo e nos resultados de várias etapas de moderação.

O processo de moderação envolve múltiplas etapas, como análise de texto, reconhecimento de imagem, avaliação da reputação do utilizador e revisão manual.

Cada etapa é implementada como um processo de trabalho separado que executa uma tarefa específica e retorna o resultado ao ContentModerator.

Aqui está um exemplo de implementação do ContentModerator:

```
1  class ContentModerator
2      include Raix::ChatCompletion
3      include Raix::FunctionDispatch
4
5      SYSTEM_DIRECTIVE = "You are a content moderator process manager,
6          tasked with the workflow involved in moderating user-generated content..."
7
8      def initialize(content)
9          @content = content
10         @transcript = [
11             { system: SYSTEM_DIRECTIVE },
12             { user: content.to_json }
13         ]
14     end
15
16     def perform
17         complete(@transcript)
18     end
19
20     def model
21         "openai/gpt-4"
22     end
23
24     # list of functions available to be called by the AI
25     # truncated for brevity
26
27     def functions
28         [
29             {
30                 name: "analyze_text",
31                 # ...
32             },
33             {
34                 name: "recognize_image",
35                 description: "Invoke to describe images...",
36                 # ...
37             },
38         ]
39     end
40 end
```

```
38     {
39         name: "assess_user_reputation",
40         # ...
41     },
42     {
43         name: "escalate_to_manual_review",
44         # ...
45     },
46     {
47         name: "approve_content",
48         # ...
49     },
50     {
51         name: "reject_content",
52         # ...
53     }
54 ]
55 end
56
57 # implementation of functions that can be called by the AI
58 # entirely at its discretion, depending on the needs of the order
59
60 def analyze_text
61     result = TextAnalysisWorker.perform(@content)
62     continue_with(result)
63 end
64
65 def recognize_image
66     result = ImageRecognitionWorker.perform(@content)
67     continue_with(result)
68 end
69
70 def assess_user_reputation
71     result = UserReputationWorker.perform(@content.user)
72     continue_with(result)
73 end
74
75 def escalate_to_manual_review
76     ManualReviewWorker.perform(@content)
77     @content.update!(status: 'pending', transcript: @transcript)
78 end
79
```

```
80  def approve_content
81    @content.update!(status: 'approved', transcript: @transcript)
82  end
83
84  def reject_content
85    @content.update!(status: 'rejected', transcript: @transcript)
86  end
87
88  private
89
90  def continue_with(result)
91    @transcript << { function: result }
92    complete(@transcript)
93  end
94 end
```

Neste exemplo, o ContentModerator é inicializado com um objeto de conteúdo e mantém um registo de moderação no formato de conversação. O componente de IA tem controlo total sobre o fluxo de moderação, decidindo quais os passos a executar com base nas características do conteúdo e nos resultados de cada etapa.

As funções trabalhadoras disponíveis para a IA invocar incluem `analyze_text`, `recognize_image`, `assess_user_reputation`, e `escalate_to_manual_review`. Cada função delega a tarefa para um processo trabalhador correspondente (TextAnalysisWorker, ImageRecognitionWorker, etc.) e adiciona o resultado ao registo de moderação, com exceção da função de escalamento, que atua como um estado final. Por fim, as funções `approve_content` e `reject_content` também atuam como estados finais.

O componente de IA analisa o conteúdo e determina a ação apropriada a tomar. Se o conteúdo contiver referências a imagens, pode chamar o trabalhador `recognize_image` para assistência com uma revisão visual. Se algum trabalhador alertar sobre conteúdo potencialmente prejudicial, a IA pode decidir escalar o conteúdo para revisão manual ou simplesmente rejeitá-lo imediatamente. Mas dependendo da gravidade do aviso, a IA pode optar por utilizar os resultados da avaliação da reputação do utilizador

para decidir como lidar com conteúdo sobre o qual não tem certeza. Dependendo do caso de uso, talvez os utilizadores confiáveis tenham mais flexibilidade no que podem publicar. E assim por diante...

Tal como no exemplo anterior do gestor de processos, o registo de moderação serve como um histórico da execução do fluxo de trabalho, incluindo os resultados de cada etapa e as decisões geradas pela IA. Este registo pode ser utilizado para auditoria, transparência e melhoria do processo de moderação ao longo do tempo.

Ao aproveitar a IA no ContentModerator, a aplicação de redes sociais pode adaptar dinamicamente o fluxo de moderação com base nas características do conteúdo e lidar com cenários complexos de moderação de forma inteligente. O componente de IA pode tomar decisões informadas, otimizar o fluxo de trabalho e garantir uma experiência comunitária segura e saudável.

Vamos explorar mais dois exemplos que demonstram o agendamento preditivo de tarefas e o tratamento e recuperação de exceções no contexto da orquestração inteligente de fluxos de trabalho.

Agendamento Preditivo de Tarefas num Sistema de Apoio ao Cliente

Numa aplicação de apoio ao cliente construída com Ruby on Rails, gerir e priorizar tickets de suporte de forma eficiente é crucial para fornecer assistência atempada aos clientes. O componente SupportTicketScheduler utiliza IA para agendar e atribuir previsivelmente tickets de suporte aos agentes disponíveis com base em vários fatores, como a urgência do ticket, a especialização do agente e a carga de trabalho.

```
1  class SupportTicketScheduler
2    include Raix::ChatCompletion
3    include Raix::FunctionDispatch
4
5    SYSTEM_DIRECTIVE = "You are a support ticket scheduler,
6      tasked with intelligently assigning tickets to available agents..."
7
8    def initialize(ticket)
9      @ticket = ticket
10     @transcript = [
11       { system: SYSTEM_DIRECTIVE },
12       { user: ticket.to_json }
13     ]
14   end
15
16   def perform
17     complete(@transcript)
18   end
19
20   def model
21     "openai/gpt-4"
22   end
23
24   def functions
25     [
26       {
27         name: "analyze_ticket_urgency",
28         # ...
29       },
30       {
31         name: "list_available_agents",
32         description: "Includes expertise of available agents",
33         # ...
34       },
35       {
36         name: "predict_agent_workload",
37         description: "Uses historical data to predict upcoming workloads",
38         # ...
39       },
40       {
41         name: "assign_ticket_to_agent",
42         # ...
```



```
43     },
44     {
45         name: "reschedule_ticket",
46         # ...
47     }
48 ]
49 end
50
51 # implementation of functions that can be called by the AI
52 # entirely at its discretion, depending on the needs of the order
53
54 def analyze_ticket_urgency
55     result = TicketUrgencyAnalyzer.perform(@ticket)
56     continue_with(result)
57 end
58
59 def list_available_agents
60     result = ListAvailableAgents.perform
61     continue_with(result)
62 end
63
64 def predict_agent_workload
65     result = AgentWorkloadPredictor.perform
66     continue_with(result)
67 end
68
69 def assign_ticket_to_agent
70     TicketAssigner.perform(@ticket, @transcript)
71 end
72
73 def delay_assignment(until)
74     until = DateTimeStandardizer.process(until)
75     SupportTicketScheduler.delay(@ticket, @transcript, until)
76 end
77
78 private
79
80 def continue_with(result)
81     @transcript << { function: result }
82     complete(@transcript)
83 end
84 end
```

Neste exemplo, o `SupportTicketScheduler` é inicializado com um objeto de ticket de suporte e mantém um registo de agendamento. O componente de IA analisa os detalhes do ticket e agenda preditivamente a atribuição do ticket com base em fatores como a urgência do ticket, a especialização do agente e a carga de trabalho prevista do agente.

As funções disponíveis para a IA invocar incluem `analyze_ticket_urgency`, `list_available_agents`, `predict_agent_workload` e `assign_ticket_to_agent`. Cada função delega a tarefa para um componente analisador ou preditor correspondente e adiciona o resultado ao registo de agendamento. A IA também tem a opção de atrasar a atribuição usando a função `delay_assignment`.

O componente de IA examina o registo de agendamento e toma decisões informadas sobre a atribuição de tickets. Considera a urgência do ticket, a especialização dos agentes disponíveis e a carga de trabalho prevista de cada agente para determinar o agente mais adequado para tratar do ticket.

Ao aproveitar o agendamento preditivo de tarefas, a aplicação de apoio ao cliente pode otimizar a atribuição de tickets, reduzir os tempos de resposta e melhorar a satisfação geral do cliente. A gestão proativa e eficiente dos tickets de suporte garante que os tickets certos são atribuídos aos agentes certos no momento certo.

Tratamento de Exceções e Recuperação numa Pipeline de Processamento de Dados

O tratamento de exceções e a recuperação de falhas são essenciais para garantir a integridade dos dados e prevenir a perda de dados. O componente `DataProcessingOrchestrator` utiliza IA para gerir inteligentemente as exceções e orquestrar o processo de recuperação numa pipeline de processamento de dados

```
1  class DataProcessingOrchestrator
2    include Raix::ChatCompletion
3    include Raix::FunctionDispatch
4
5    SYSTEM_DIRECTIVE = "You are a data processing orchestrator..."
6
7    def initialize(data_batch)
8      @data_batch = data_batch
9      @transcript = [
10        { system: SYSTEM_DIRECTIVE },
11        { user: data_batch.to_json }
12      ]
13    end
14
15    def perform
16      complete(@transcript)
17    end
18
19    def model
20      "openai/gpt-4"
21    end
22
23    def functions
24      [
25        {
26          name: "validate_data",
27          # ...
28        },
29        {
30          name: "process_data",
31          # ...
32        },
33        {
34          name: "request_fix",
35          # ...
36        },
37        {
38          name: "retry_processing",
39          # ...
40        },
41        {
42          name: "mark_data_as_failed",
```

```
43         # ...
44     },
45     {
46         name: "finished",
47         # ...
48     }
49 ]
50 end
51
52 # implementation of functions that can be called by the AI
53 # entirely at its discretion, depending on the needs of the order
54
55 def validate_data
56     result = DataValidator.perform(@data_batch)
57     continue_with(result)
58 rescue ValidationException => e
59     handle_validation_exception(e)
60 end
61
62 def process_data
63     result = DataProcessor.perform(@data_batch)
64     continue_with(result)
65 rescue ProcessingException => e
66     handle_processing_exception(e)
67 end
68
69 def request_fix(description_of_fix)
70     result = SmartDataFixer.new(description_of_fix, @data_batch)
71     continue_with(result)
72 end
73
74 def retry_processing(timeout_in_seconds)
75     wait(timeout_in_seconds)
76     process_data
77 end
78
79 def mark_data_as_failed
80     @data_batch.update!(status: 'failed', transcript: @transcript)
81 end
82
83 def finished
84     @data_batch.update!(status: 'finished', transcript: @transcript)
```

```
85     end
86
87     private
88
89     def continue_with(result)
90       @transcript << { function: result }
91       complete(@transcript)
92     end
93
94     def handle_validation_exception(exception)
95       @transcript << { exception: exception.message }
96       complete(@transcript)
97     end
98
99     def handle_processing_exception(exception)
100       @transcript << { exception: exception.message }
101       complete(@transcript)
102     end
103   end
```

Neste exemplo, o `DataProcessingOrchestrator` é inicializado com um objeto de lote de dados e mantém um registo de processamento. O componente de IA orquestra o pipeline de processamento de dados, gerindo exceções e recuperando de falhas conforme necessário.

As funções disponíveis para a IA invocar incluem `validate_data`, `process_data`, `request_fix`, `retry_processing` e `mark_data_as_failed`. Cada função delega a tarefa a um componente de processamento de dados correspondente e adiciona o resultado ou os detalhes da exceção ao registo de processamento.

Se ocorrer uma exceção de validação durante a etapa `validate_data`, a função `handle_validation_exception` adiciona os dados da exceção ao registo e devolve o controlo à IA. Da mesma forma, se ocorrer uma exceção de processamento durante a etapa `process_data`, a IA pode decidir sobre a estratégia de recuperação.

Dependendo da natureza da exceção encontrada, a IA pode, à sua discrição, decidir chamar `request_fix`, que delega num componente `SmartDataFixer` baseado em IA

(ver capítulo Dados Auto-Reparáveis). O corretor de dados recebe uma descrição em linguagem simples de como deve modificar o `@data_batch` para que o processamento possa ser repetido. Talvez uma repetição bem-sucedida implique remover registos do lote de dados que falharam a validação e/ou copiá-los para um pipeline de processamento diferente para revisão humana? As possibilidades são quase infinitas.

Ao incorporar o tratamento e recuperação de exceções baseados em IA, a aplicação de processamento de dados torna-se mais resiliente e tolerante a falhas. O `DataProcessingOrchestrator` gere inteligentemente as exceções, minimiza a perda de dados e garante a execução suave do fluxo de trabalho de processamento de dados.

Monitorização e Registo de Eventos

A monitorização e o registo de eventos proporcionam visibilidade sobre o progresso, desempenho e saúde dos componentes do fluxo de trabalho baseado em IA, permitindo aos programadores acompanhar e analisar o comportamento do sistema. A implementação de mecanismos eficazes de monitorização e registo é essencial para a depuração, auditoria e melhoria contínua dos fluxos de trabalho inteligentes.

Monitorização do Progresso e Desempenho do Fluxo de Trabalho

Para garantir a execução suave dos fluxos de trabalho inteligentes, é importante monitorizar o progresso e o desempenho de cada componente do fluxo de trabalho. Isto envolve o acompanhamento de métricas e eventos principais ao longo do ciclo de vida do fluxo de trabalho.

Alguns aspectos importantes a monitorizar incluem:

1. **Tempo de Execução do Fluxo de Trabalho:** Medir o tempo necessário para cada componente do fluxo de trabalho completar a sua tarefa. Isto ajuda a identificar estrangulamentos de desempenho e a otimizar a eficiência geral do fluxo de trabalho.
2. **Utilização de Recursos:** Monitorizar a utilização de recursos do sistema, como CPU, memória e armazenamento, por cada componente do fluxo de trabalho. Isto ajuda a garantir que o sistema está a operar dentro da sua capacidade e pode gerir a carga de trabalho eficazmente.
3. **Taxas de Erro e Exceções:** Acompanhar a ocorrência de erros e exceções dentro dos componentes do fluxo de trabalho. Isto ajuda a identificar potenciais problemas e permite o tratamento e recuperação proativos de erros.
4. **Pontos de Decisão e Resultados:** Monitorizar os pontos de decisão dentro do fluxo de trabalho e os resultados das decisões baseadas em IA. Isto fornece informações sobre o comportamento e a eficácia dos componentes de IA.

Os dados capturados pelos processos de monitorização podem ser apresentados em painéis de controlo ou utilizados como entradas para relatórios programados que informam os administradores do sistema sobre a saúde do sistema.



Os dados de monitorização podem ser alimentados a um processo de administrador de sistema baseado em IA para revisão e possível ação!

Registo de Eventos e Decisões Principais

O registo de eventos é uma prática essencial que envolve a captura e armazenamento de informações relevantes sobre eventos principais, decisões e exceções que ocorrem durante a execução do fluxo de trabalho.

Alguns aspectos importantes a registar incluem:

1. Iniciação e Conclusão do Fluxo de Trabalho: Registrar os tempos de início e fim de cada instância do fluxo de trabalho, juntamente com quaisquer metadados relevantes, como os dados de entrada e o contexto do utilizador.

2. Execução de Componentes: Registrar os detalhes de execução de cada componente do fluxo de trabalho, incluindo os parâmetros de entrada, resultados de saída e quaisquer dados intermediários gerados.

3. Decisões e Raciocínio da IA: Registrar as decisões tomadas pelos componentes de IA, juntamente com o raciocínio subjacente ou pontuações de confiança. Isto proporciona transparência e permite a auditoria de decisões baseadas em IA.

4. Exceções e Mensagens de Erro: Registrar quaisquer exceções ou mensagens de erro encontradas durante a execução do fluxo de trabalho, incluindo o rastreamento de pilha e informações contextuais relevantes.

O registo pode ser implementado usando várias técnicas, como escrever em ficheiros de registo, armazenar registos numa base de dados ou enviar registos para um serviço de registo centralizado. É importante escolher uma estrutura de registo que forneça flexibilidade, escalabilidade e fácil integração com a arquitetura da aplicação.

Aqui está um exemplo de como o registo pode ser implementado numa aplicação Ruby on Rails usando a classe `ActiveSupport::Logger`:

```
1 class WorkflowLogger
2   def self.log(message, severity = :info)
3     @logger ||= ActiveSupport::Logger.new('workflow.log')
4     @logger.formatter ||= proc do |severity, datetime, progname, msg|
5       "#{datetime} [{severity}] #{msg}\n"
6     end
7     @logger.send(severity, message)
8   end
9 end
10
11 # Usage example
12 WorkflowLogger.log("Workflow initiated for order #{@order.id}")
13 WorkflowLogger.log("Payment processing completed successfully")
14 WorkflowLogger.log("Inventory check failed for item #{item.id}", :error)
```


Ao colocar estrategicamente declarações de registo ao longo dos componentes do fluxo de trabalho e pontos de decisão da IA, os programadores podem capturar informações valiosas para depuração, auditoria e análise.

Benefícios da Monitorização e Registo

A implementação de monitorização e registo na orquestração inteligente de fluxos de trabalho oferece vários benefícios:

1. **Depuração e Resolução de Problemas:** Registos detalhados e dados de monitorização ajudam os programadores a identificar e diagnosticar problemas rapidamente. Fornecem informações sobre o fluxo de execução do workflow, interações entre componentes e quaisquer erros ou exceções encontrados.
2. **Otimização de Desempenho:** A monitorização de métricas de desempenho permite aos programadores identificar estrangulamentos e otimizar os componentes do fluxo de trabalho para maior eficiência. Ao analisar tempos de execução, utilização de recursos e outras métricas, os programadores podem tomar decisões informadas para melhorar o desempenho geral do sistema.
3. **Auditoria e Conformidade:** O registo de eventos e decisões importantes fornece um histórico de auditoria para conformidade regulamentar e responsabilização. Permite às organizações rastrear e verificar as ações tomadas pelos componentes de IA e garantir a adesão às regras de negócio e requisitos legais.
4. **Melhoria Contínua:** Os dados de monitorização e registo servem como contributos valiosos para a melhoria contínua dos fluxos de trabalho inteligentes. Ao analisar dados históricos, identificar padrões e medir a eficácia das decisões da IA, os programadores podem refinar e melhorar iterativamente a lógica de orquestração do fluxo de trabalho.

Considerações e Boas Práticas

Ao implementar monitorização e registo na orquestração inteligente de fluxos de trabalho, considere as seguintes boas práticas:

1. Definir Métricas Claras de Monitorização: Identifique as métricas e eventos principais que precisam ser monitorizados com base nos requisitos específicos do fluxo de trabalho. Concentre-se em métricas que forneçam informações significativas sobre o desempenho, saúde e comportamento do sistema.

2. Implementar Registo Granular: Garanta que as declarações de registo são colocadas em pontos apropriados dentro dos componentes do fluxo de trabalho e pontos de decisão da IA. Capture informações contextuais relevantes, como parâmetros de entrada, resultados de saída e quaisquer dados intermediários gerados.

3. Utilizar Registo Estruturado: Adote um formato de registo estruturado para facilitar a análise e interpretação dos dados de registo. O registo estruturado permite melhor capacidade de pesquisa, filtragem e agregação das entradas de registo.

4. Gerir Retenção e Rotação de Registos: Implemente políticas de retenção e rotação de registos para gerir o armazenamento e ciclo de vida dos ficheiros de registo. Determine o período de retenção apropriado com base em requisitos legais, restrições de armazenamento e necessidades de análise. Se possível, transfira o registo para um serviço de terceiros como o [Papertrail](#).

5. Proteger Informações Sensíveis: Seja cauteloso ao registar informações sensíveis, como informações pessoalmente identificáveis (PII) ou dados comerciais confidenciais. Implemente medidas de segurança apropriadas, como mascaramento de dados ou encriptação, para proteger informações sensíveis nos ficheiros de registo.

6. Integrar com Ferramentas de Monitorização e Alerta: Aproveite ferramentas de monitorização e alerta para centralizar a recolha, análise e visualização de dados de monitorização e registo. Estas ferramentas podem fornecer informações em tempo real, gerar alertas com base em limites predefinidos e facilitar a deteção e resolução proativa de problemas. A minha ferramenta favorita entre estas é o [Datadog](#).

Ao implementar mecanismos abrangentes de monitorização e registo, os programadores podem obter informações valiosas sobre o comportamento e desempenho dos fluxos de trabalho inteligentes. Estas informações permitem a depuração eficaz, otimização e

melhoria contínua dos sistemas de orquestração de fluxos de trabalho baseados em IA.

Considerações de Escalabilidade e Desempenho

A escalabilidade e o desempenho são aspectos críticos a considerar ao projetar e implementar sistemas de orquestração de fluxos de trabalho inteligentes. À medida que o volume de fluxos de trabalho concorrentes e a complexidade dos componentes baseados em IA aumentam, torna-se essencial garantir que o sistema pode lidar com a carga de trabalho de forma eficiente e escalar perfeitamente para atender às crescentes exigências.

Gestão de Grandes Volumes de Fluxos de Trabalho Concorrentes

Os sistemas de orquestração de fluxos de trabalho inteligentes frequentemente precisam de lidar com um grande número de fluxos de trabalho concorrentes. Para garantir a escalabilidade, considere as seguintes estratégias:

- 1. Processamento Assíncrono:** Implemente mecanismos de processamento assíncrono para desacoplar a execução dos componentes do fluxo de trabalho. Isto permite que o sistema lide com múltiplos fluxos de trabalho simultaneamente sem bloquear ou esperar que cada componente termine. O processamento assíncrono pode ser alcançado usando filas de mensagens, arquiteturas orientadas a eventos ou frameworks de processamento de tarefas em segundo plano como o Sidekiq.
- 2. Arquitetura Distribuída:** Projete a arquitetura do sistema para usar componentes sem servidor (como AWS Lambda) ou simplesmente distribua a carga de trabalho por vários nós ou servidores junto com o seu servidor de aplicação principal. Isto permite a escalabilidade horizontal, onde nós adicionais podem ser adicionados para lidar com volumes aumentados de fluxos de trabalho.
- 3. Execução Paralela:** Identifique oportunidades para execução paralela dentro dos fluxos de trabalho. Alguns componentes do fluxo de trabalho podem ser independentes

uns dos outros e podem ser executados simultaneamente. Ao aproveitar técnicas de processamento paralelo, como multithreading ou filas de tarefas distribuídas, o sistema pode otimizar a utilização de recursos e reduzir o tempo total de execução do fluxo de trabalho.

Otimização do Desempenho de Componentes Baseados em IA

Os componentes baseados em IA, como modelos de aprendizagem automática ou motores de processamento de linguagem natural, podem ser computacionalmente intensivos e impactar o desempenho geral do sistema de orquestração de fluxos de trabalho. Para otimizar o desempenho dos componentes de IA, considere as seguintes técnicas:

1. **Caching:** Se o seu processamento de IA for puramente generativo e não envolver consultas de informações em tempo real ou integrações externas para gerar as suas respostas de chat, então pode considerar mecanismos de cache para armazenar e reutilizar os resultados de operações frequentemente acedidas ou computacionalmente dispendiosas.
2. **Otimização de Modelos:** Otimize continuamente a forma como utiliza os modelos de IA nos componentes do fluxo de trabalho. Isto pode envolver técnicas como *Destilação de Prompts* ou pode simplesmente ser uma questão de testar novos modelos à medida que ficam disponíveis.
3. **Processamento em Lote:** Se estiver a trabalhar com modelos da classe GPT-4, poderá aproveitar técnicas de processamento em lote para processar múltiplos pontos de dados ou pedidos num único lote, em vez de os processar individualmente. Ao processar dados em lotes, o sistema pode otimizar a utilização de recursos e reduzir a sobrecarga de pedidos repetidos ao modelo.

Monitorização e Análise de Desempenho

Para identificar gargalos de desempenho e otimizar a escalabilidade do sistema de orquestração inteligente de fluxos de trabalho, é crucial implementar mecanismos de monitorização e análise. Considere as seguintes abordagens:

1. Métricas de Desempenho: Defina e acompanhe métricas-chave de desempenho, como tempo de resposta, taxa de transferência, utilização de recursos e latência. Estas métricas fornecem informações sobre o desempenho do sistema e ajudam a identificar áreas para otimização. O popular agregador de modelos de IA [OpenRouter](#) inclui métricas de Host¹ e Velocidade² em cada resposta da API, tornando trivial o acompanhamento destas métricas-chave.

2. Ferramentas de Análise: Utilize ferramentas de análise para avaliar o desempenho de componentes individuais do fluxo de trabalho e operações de IA. As ferramentas de análise podem ajudar a identificar pontos críticos de desempenho, caminhos de código ineficientes ou operações que consumam muitos recursos. As ferramentas populares de análise incluem New Relic, Scout, ou analisadores integrados fornecidos pela linguagem de programação ou framework.

3. Testes de Carga: Realize testes de carga para avaliar o desempenho do sistema sob diferentes níveis de cargas de trabalho simultâneas. Os testes de carga ajudam a identificar os limites de escalabilidade do sistema, detetar degradação de desempenho e garantir que o sistema pode lidar com o tráfego esperado sem comprometer o desempenho.

4. Monitorização Contínua: Implemente mecanismos de monitorização contínua e alertas para detetar proativamente problemas e gargalos de desempenho. Configure painéis de monitorização e alertas para acompanhar indicadores-chave de desempenho

¹Host é o tempo que demorou a receber o primeiro byte da geração transmitida do host do modelo, também conhecido como “tempo até ao primeiro byte.”

²A Velocidade é calculada como o número de tokens de conclusão dividido pelo tempo total de geração. Para pedidos não transmitidos em stream, a latência é considerada parte do tempo de geração.

(KPIs) e receber notificações quando os limites predefinidos forem ultrapassados. Isto permite a identificação e resolução rápida de problemas de desempenho.

Estratégias de Escalamento

Para lidar com cargas de trabalho crescentes e garantir a escalabilidade do sistema de orquestração inteligente de fluxos de trabalho, considere as seguintes estratégias de escalamento:

- 1. Escalamento Vertical:** O escalamento vertical envolve aumentar os recursos (por exemplo, CPU, memória) de nós ou servidores individuais para lidar com cargas de trabalho mais elevadas. Esta abordagem é adequada quando o sistema requer mais poder de processamento ou memória para lidar com fluxos de trabalho complexos ou operações de IA.
- 2. Escalamento Horizontal:** O escalamento horizontal envolve adicionar mais nós ou servidores ao sistema para distribuir a carga de trabalho. Esta abordagem é eficaz quando o sistema precisa de lidar com um grande número de fluxos de trabalho simultâneos ou quando a carga de trabalho pode ser facilmente distribuída por múltiplos nós. O escalamento horizontal requer uma arquitetura distribuída e mecanismos de balanceamento de carga para garantir uma distribuição uniforme do tráfego.
- 3. Auto-Escalamento:** Implemente mecanismos de auto-escalamento para ajustar automaticamente o número de nós ou recursos com base na procura da carga de trabalho. O auto-escalamento permite que o sistema escale dinamicamente para cima ou para baixo dependendo do tráfego recebido, garantindo uma utilização ótima dos recursos e eficiência de custos. Plataformas cloud como Amazon Web Services (AWS) ou Google Cloud Platform (GCP) fornecem capacidades de auto-escalamento que podem ser aproveitadas para sistemas de orquestração inteligente de fluxos de trabalho.

Técnicas de Otimização de Desempenho

Para além das estratégias de escalamento, considere as seguintes técnicas de otimização de desempenho para melhorar a eficiência do sistema de orquestração inteligente de fluxos de trabalho:

- 1. Armazenamento e Recuperação Eficiente de Dados:** Otimize os mecanismos de armazenamento e recuperação de dados utilizados pelos componentes do fluxo de trabalho. Utilize indexação eficiente de bases de dados, técnicas de otimização de consultas e cache de dados para minimizar a latência e melhorar o desempenho de operações intensivas de dados.
- 2. E/S Assíncrona:** Utilize operações de E/S assíncronas para evitar bloqueios e melhorar a capacidade de resposta do sistema. A E/S assíncrona permite que o sistema processe múltiplos pedidos simultaneamente sem esperar pela conclusão das operações de E/S, maximizando assim a utilização dos recursos.
- 3. Serialização e Deserialização Eficientes:** Otimize os processos de serialização e deserialização utilizados na troca de dados entre componentes do workflow. Utilize formatos de serialização eficientes, como Protocol Buffers ou MessagePack, para reduzir a sobrecarga da serialização de dados e melhorar o desempenho da comunicação entre componentes.



Para aplicações baseadas em Ruby, considere utilizar [Universal ID](#). O Universal ID aproveita tanto o MessagePack como o Brotli (uma combinação construída para velocidade e compressão de dados de primeira classe). Quando combinadas, estas bibliotecas são até 30% mais rápidas e têm taxas de compressão entre 2-5% em comparação com o Protocol Buffers.

- 4. Compressão e Codificação:** Aplique técnicas de compressão e codificação para reduzir o tamanho dos dados transferidos entre componentes do workflow. Algoritmos

de compressão, como gzip ou Brotli, podem reduzir significativamente o uso da largura de banda de rede e melhorar o desempenho geral do sistema.

Ao considerar aspectos de escalabilidade e desempenho durante o design e implementação de sistemas de orquestração de workflows inteligentes, pode garantir que o seu sistema consegue lidar com grandes volumes de workflows concorrentes, otimizar o desempenho dos componentes baseados em IA e escalar facilmente para satisfazer as crescentes exigências. A monitorização contínua, a análise de desempenho e os esforços de otimização são essenciais para manter o desempenho e a capacidade de resposta do sistema à medida que a carga de trabalho e a complexidade aumentam ao longo do tempo.

Testes e Validação de Workflows

Os testes e a validação são aspectos críticos no desenvolvimento e manutenção de sistemas de orquestração de workflows inteligentes. Dada a natureza complexa dos workflows baseados em IA, é essencial garantir que cada componente funciona como esperado, que o workflow global se comporta corretamente e que as decisões da IA são precisas e fiáveis. Nesta secção, iremos explorar várias técnicas e considerações para testar e validar workflows inteligentes.

Testes Unitários de Componentes do Workflow

Os testes unitários envolvem testar componentes individuais do workflow isoladamente para verificar a sua correção e robustez. Ao realizar testes unitários em componentes de workflow baseados em IA, considere o seguinte:

1. **Validação de Entrada:** Teste a capacidade do componente de lidar com diferentes tipos de entrada, incluindo dados válidos e inválidos. Verifique se o componente lida adequadamente com casos limite e fornece mensagens de erro ou exceções apropriadas.

2. Verificação de Saída: Confirme que o componente produz a saída esperada para um determinado conjunto de entradas. Compare a saída real com os resultados esperados para garantir a correção.

3. Tratamento de Erros: Teste os mecanismos de tratamento de erros do componente simulando vários cenários de erro, como entrada inválida, indisponibilidade de recursos ou exceções inesperadas. Verifique se o componente captura e trata os erros adequadamente.

4. Condições de Fronteira: Teste o comportamento do componente sob condições de fronteira, como entrada vazia, tamanho máximo de entrada ou valores extremos. Garanta que o componente lida com estas condições adequadamente sem falhar ou produzir resultados incorretos.

Aqui está um exemplo de um teste unitário para um componente de workflow em Ruby utilizando a framework de testes RSpec:

```
1  RSpec.describe OrderValidator do
2    describe '#validate' do
3      context 'when order is valid' do
4        let(:order) { build(:order) }
5
6        it 'returns true' do
7          expect(subject.validate(order)).to be true
8        end
9      end
10
11     context 'when order is invalid' do
12       let(:order) { build(:order, total_amount: -100) }
13
14       it 'returns false' do
15         expect(subject.validate(order)).to be false
16       end
17     end
18   end
19 end
```

Neste exemplo, o componente `OrderValidator` é testado usando dois casos de teste:

um para uma ordem válida e outro para uma ordem inválida. Os casos de teste verificam se o método `validate` retorna o valor booleano esperado com base na validade da ordem.

Testes de Integração das Interações do Fluxo de Trabalho

Os testes de integração concentram-se em verificar as interações e o fluxo de dados entre diferentes componentes do fluxo de trabalho. Garantem que os componentes funcionem em conjunto de forma harmoniosa e produzam os resultados esperados. Ao realizar testes de integração em fluxos de trabalho inteligentes, considere o seguinte:

1. **Interação entre Componentes:** Teste a comunicação e a troca de dados entre os componentes do fluxo de trabalho. Verifique se a saída de um componente é corretamente passada como entrada para o próximo componente no fluxo de trabalho.
2. **Consistência de Dados:** Garanta que os dados permaneçam consistentes e precisos enquanto fluem através do fluxo de trabalho. Verifique se as transformações de dados, cálculos e agregações são realizados corretamente.
3. **Propagação de Exceções:** Teste como as exceções e erros são propagados e tratados entre os componentes do fluxo de trabalho. Verifique se as exceções são capturadas, registadas e tratadas adequadamente para evitar a interrupção do fluxo de trabalho.
4. **Comportamento Assíncrono:** Se o fluxo de trabalho envolver componentes assíncronos ou execução paralela, teste os mecanismos de coordenação e sincronização. Garanta que o fluxo de trabalho se comporte corretamente em cenários concorrentes e assíncronos.

Aqui está um exemplo de um teste de integração para um fluxo de trabalho em Ruby usando o framework de testes RSpec:

```
1  RSpec.describe OrderProcessingWorkflow do
2
3    let(:order) { build(:order) }
4
5    it 'processes the order successfully' do
6      expect(OrderValidator).to receive(:validate).and_return(true)
7      expect(InventoryManager).to receive(:check_availability).and_return(true)
8      expect(PaymentProcessor).to receive(:process_payment).and_return(true)
9      expect(ShippingService).to receive(:schedule_shipping).and_return(true)
10
11      workflow = OrderProcessingWorkflow.new(order)
12      result = workflow.process
13
14      expect(result).to be true
15      expect(order.status).to eq('processed')
16    end
17  end
```

Neste exemplo, o `OrderProcessingWorkflow` é testado através da verificação das interações entre diferentes componentes do fluxo de trabalho. O caso de teste estabelece expectativas para o comportamento de cada componente e garante que o fluxo de trabalho processa a encomenda com sucesso, atualizando o estado da encomenda em conformidade.

Testar Pontos de Decisão de IA

Testar pontos de decisão de IA é crucial para garantir a precisão e fiabilidade dos fluxos de trabalho baseados em IA. Ao testar pontos de decisão de IA, considere o seguinte:

1. **Precisão da Decisão:** Verifique se o componente de IA toma decisões precisas com base nos dados de entrada e no modelo treinado. Compare as decisões da IA com os resultados esperados ou dados de referência.
2. **Casos Limite:** Teste o comportamento do componente de IA em casos limite e cenários invulgares. Verifique se o componente de IA lida com estes casos de forma adequada e toma decisões razoáveis.

3. Viés e Equidade: Avalie o componente de IA quanto a possíveis vieses e garanta que toma decisões justas e imparciais. Teste o componente com dados de entrada diversos e analise os resultados procurando padrões discriminatórios.

4. Explicabilidade: Se o componente de IA fornece explicações ou fundamentação para as suas decisões, verifique a correção e clareza das explicações. Garanta que as explicações estão alinhadas com o processo de tomada de decisão subjacente.

Aqui está um exemplo de como testar um ponto de decisão de IA em Ruby usando o RSpec testing framework:

```
1 RSpec.describe FraudDetector do
2   describe '#detect_fraud' do
3     context 'when transaction is fraudulent' do
4       let(:tx) do
5         build(:transaction, amount: 10_000, location: 'High-Risk Country')
6       end
7
8       it 'returns true' do
9         expect(subject.detect_fraud(tx)).to be true
10      end
11    end
12
13    context 'when transaction is legitimate' do
14      let(:tx) do
15        build(:transaction, amount: 100, location: 'Low-Risk Country')
16      end
17
18      it 'returns false' do
19        expect(subject.detect_fraud(tx)).to be false
20      end
21    end
22  end
23 end
```

Neste exemplo, o componente de IA FraudDetector é testado com dois casos de teste: um para uma transação fraudulenta e outro para uma transação legítima. Os casos de teste verificam se o método detect_fraud retorna o valor booleano esperado com base nas características da transação.

Teste Ponta a Ponta

O teste ponta a ponta envolve testar todo o fluxo de trabalho do início ao fim, simulando cenários e interações do utilizador do mundo real. Garante que o fluxo de trabalho se comporta corretamente e produz os resultados desejados. Ao realizar testes ponta a ponta para fluxos de trabalho inteligentes, considere o seguinte:

1. **Cenários de Utilizador:** Identifique cenários comuns de utilizador e teste o comportamento do fluxo de trabalho nestes cenários. Verifique se o fluxo de trabalho processa corretamente as entradas do utilizador, toma decisões apropriadas e produz as saídas esperadas.
2. **Validação de Dados:** Garanta que o fluxo de trabalho valida e limpa as entradas do utilizador para prevenir inconsistências nos dados ou vulnerabilidades de segurança. Teste o fluxo de trabalho com vários tipos de dados de entrada, incluindo dados válidos e inválidos.
3. **Recuperação de Erros:** Teste a capacidade do fluxo de trabalho de recuperar de erros e exceções. Simule cenários de erro e verifique se o fluxo de trabalho os trata adequadamente, regista os erros e toma as ações de recuperação apropriadas.
4. **Desempenho e Escalabilidade:** Avalie o desempenho e a escalabilidade do fluxo de trabalho sob diferentes condições de carga. Teste o fluxo de trabalho com um grande volume de pedidos concorrentes e meça os tempos de resposta, a utilização de recursos e a estabilidade geral do sistema.

Aqui está um exemplo de um teste ponta a ponta para um fluxo de trabalho em Ruby utilizando a framework de testes RSpec e a biblioteca Capybara para simular interações do utilizador:

```
1 RSpec.describe 'Order Processing Workflow' do
2   cenário 'User places an order successfully' do
3     visit '/orders/new'
4     fill_in 'Product', with: 'Sample Product'
5     fill_in 'Quantity', with: '2'
6     fill_in 'Shipping Address', with: '123 Main St'
7     click_button 'Place Order'
8
9     expect(page).to have_content('Order Placed Successfully')
10    expect(Order.count).to eq(1)
11    expect(Order.last.status).to eq('processed')
12  end
13 end
```

Neste exemplo, o teste ponta a ponta simula um utilizador a fazer uma encomenda através da interface web. Preenche os campos obrigatórios do formulário, submete a encomenda e verifica se esta é processada com sucesso, apresentando a mensagem de confirmação apropriada e atualizando o estado da encomenda na base de dados.

Integração e Implementação Contínuas

Para garantir a fiabilidade e manutenção dos fluxos de trabalho inteligentes, recomenda-se a integração de testes e validação no pipeline de integração e implementação contínuas (CI/CD). Isto permite a realização automatizada de testes e validação das alterações ao fluxo de trabalho antes de serem implementadas em produção. Considere as seguintes práticas:

- 1. Execução Automatizada de Testes:** Configure o pipeline de CI/CD para executar automaticamente o conjunto de testes sempre que sejam feitas alterações à base de código do fluxo de trabalho. Isto garante que quaisquer regressões ou falhas são detetadas no início do processo de desenvolvimento.
- 2. Monitorização da Cobertura de Testes:** Meça e monitorize a cobertura de testes dos componentes do fluxo de trabalho e dos pontos de decisão da IA. Procure alcançar

uma alta cobertura de testes para garantir que os caminhos e cenários críticos são minuciosamente testados.

3. Feedback Contínuo: Integre os resultados dos testes e as métricas de qualidade do código no fluxo de trabalho de desenvolvimento. Forneça feedback contínuo aos programadores sobre o estado dos testes, qualidade do código e quaisquer problemas detetados durante o processo de CI/CD.

4. Ambientes de Teste: Implemente o fluxo de trabalho em ambientes de teste que espelhem de perto o ambiente de produção. Realize testes e validações adicionais no ambiente de teste para detetar quaisquer problemas relacionados com infraestrutura, configuração ou integração de dados.

5. Mecanismos de Reversão: Implemente mecanismos de reversão para casos de falhas na implementação ou problemas críticos detetados em produção. Certifique-se de que o fluxo de trabalho pode ser rapidamente revertido para uma versão estável anterior para minimizar o tempo de inatividade e o impacto nos utilizadores.

Ao incorporar testes e validação ao longo do ciclo de vida de desenvolvimento dos fluxos de trabalho inteligentes, as organizações podem garantir a fiabilidade, precisão e manutenção dos seus sistemas baseados em IA. Testes e validação regulares ajudam a detetar erros, prevenir regressões e criar confiança no comportamento e nos resultados do fluxo de trabalho.

Parte 2: Os Padrões

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Engenharia de Prompts

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Chain of Thought

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Como Funciona

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Exemplos

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Geração de Conteúdo

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Criação de Entidades Estruturadas

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Orientação do Agente LLM

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Benefícios e Considerações

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Mudança de Modo

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Como Funciona

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Quando Utilizar

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Exemplo

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Atribuição de Função

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Como Funciona

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Quando Usar

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Exemplos

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Objeto Prompt

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Como Funciona

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Template de Prompt

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Como Funciona

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Benefícios e Considerações

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Quando Utilizar:

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Exemplo

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Structured IO

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Como Funciona

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Escalando Structured IO

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Benefícios e Considerações

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Encadeamento de Prompts

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Como Funciona

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Quando Usar

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Exemplo: Integração da Olympia

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Prompt Rewriter

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Como Funciona

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Exemplo

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Response Fencing

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Como Funciona

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Benefícios e Considerações

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Tratamento de Erros

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Analizador de Consultas

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Como Funciona

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Implementação

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Marcação de Partes do Discurso (POS) e Reconhecimento de Entidades Nomeadas (REN)

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Classificação de Intenção

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Extração de Palavras-chave

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Benefícios

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Query Rewriter

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Como Funciona

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Exemplo

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Benefícios

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Ventriloquist

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Como Funciona

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Quando Utilizar

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Exemplo

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Componentes Discretos

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Predicado

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Como Funciona

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Quando Utilizar

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Exemplo

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

API Facade

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Como Funciona

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Principais Benefícios

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Quando Usar

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Exemplo

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Autenticação e Autorização

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Tratamento de Requisições

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Formatação de Respostas

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Tratamento de Erros e Casos Limite

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Considerações sobre Escalabilidade e Desempenho

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Comparação com Outros Padrões de Design

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Interpretador de Resultados

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Como Funciona

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Quando Utilizar

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Exemplo

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Máquina Virtual

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Como Funciona

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Quando Utilizar

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Exemplo

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Por Detrás da Magia

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Especificação e Testes

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Especificar o Comportamento

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Escrever Casos de Teste

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Exemplo: Testar o Componente Tradutor

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Reprodução de Interações HTTP

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Human In The Loop (HITL)

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Padrões de Alto Nível

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Inteligência Híbrida

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Resposta Adaptativa

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Alternância de Papéis Humano-IA

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Escalonamento

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Como Funciona

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Principais Benefícios

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Aplicação no Mundo Real: Saúde

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Ciclo de Feedback

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Como Funciona

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Aplicações e Exemplos

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Técnicas Avançadas na Integração de Feedback Humano

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Radiação Passiva de Informação

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Como Funciona

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Exibição Contextual de Informação

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Notificações Proativas

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Insights Explicativos

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Exploração Interativa

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Principais Benefícios

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Aplicações e Exemplos

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Tomada de Decisão Colaborativa (TDC)

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Como Funciona

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Exemplo

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Aprendizagem Contínua

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Como Funciona

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Aplicações e Exemplos

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Exemplo

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Considerações Éticas

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Papel do HITL na Mitigação de Riscos de IA

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Avanços Tecnológicos e Perspetivas Futuras

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Desafios e Limitações dos Sistemas HITL

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Tratamento Inteligente de Erros

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Abordagens Tradicionais de Tratamento de Erros

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Diagnóstico Contextual de Erros

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Como Funciona

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Engenharia de Prompts para Diagnóstico Contextual de Erros

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Geração Aumentada por Recuperação para Diagnóstico Contextual de Erros

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Relatórios Inteligentes de Erros

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Prevenção Preditiva de Erros

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Como Funciona

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Recuperação Inteligente de Erros

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Como Funciona

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Comunicação de Erros Personalizada

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Como Funciona

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Fluxo de Trabalho Adaptativo de Tratamento de Erros

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Como Funciona

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Controlo de Qualidade

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Eval

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Problema

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Solução

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Como Funciona

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Exemplo

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Considerações

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Compreendendo as Referências Douradas

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Como Funcionam as Avaliações Sem Referência

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Barreira de Proteção

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Problema

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Solução

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Como Funciona

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Exemplo

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Considerações

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Guardrails e Evals: Dois Lados da Mesma Moeda

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

A Intercambialidade de Guardrails e Evals sem Referência

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Implementando Guardrails e Evals de Duplo Propósito

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Glossário

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Glossário

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

A

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

B

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

C

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

D

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

E

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

F

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

G

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

H

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

I

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

J

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

K

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

L

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

M

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

N

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

O

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

P

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Q

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

R

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

S

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

T

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

U

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

V

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

W

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Z

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em <http://leanpub.com/patterns-of-application-development-using-ai-pt-PT>.

Index

- acessibilidade, 216
- acompanhamento de métricas principais,
 - 242
- agrupamento de documentos, 120
- Agênticos, 32
- AI, 143
- ajuste fino, 79
- ajuste por instruções, 10
 - modelos ajustados por instruções, 50,
 - 52
- Alocação Latente de Dirichlet, 122
- Alpaca, 13
- Altman, Sam, 18
- Amazon Web Services, 250
- ambientes de desenvolvimento local, 155
- ambientes de teste, 259
- Amostragem Top-k, 48
- Amostragem Top-p (nucleus), 48
- Anthropic, 23, 39, 73, 129, 136
- antropomorfismo, 68
- análise de sentimento, 16, 100, 112–114,
 - 117, 118, 134
- Análise de Sentimento do Cliente, 99
- análise de sentimentos, 145
- APIs, 71, 123, 153
- aplicação de chatbot, 119
- Aplicações de Comércio Eletrônico, 91
- aplicações educacionais, 32
- aplicações modernas, 222
- Apoio à Decisão Clínica, 103
- aprendizagem não supervisionada, 4
- Aprendizagem One-Shot, 60
- aprendizagem zero-shot, 59
- arquitetura de aplicações empresariais, 38
- Arquitetura de microsserviços, 89
- arquitetura de software, 2
- arquitetura distribuída, 247
- arquitetura orientada a eventos, 108
- arquitetura transformer, 6
- arrays, 130
- assistentes virtuais, 34
- atribuição de tickets, 238
- auditoria e conformidade, 245
- auto-escalamento, 250
- Avaliação e Estratificação de Sintomas, 101
- base de conhecimento da Olympia, 91
- bases de conhecimento, 7
- bases de dados, 123
 - estratégias de bloqueio, 109
 - objeto baseado em, 105
- BERT, 14, 24
- biblioteca Capybara, 257

- bloqueio otimista, 109
- bloqueio pessimista, 109
- boundary conditions, 253
- Brotli, 251, 252
- Byte Pair Encoding (BPE), 13, 14
- C (Linguagem de Programação), 116
- cache, 248
- cadeia de abastecimento
 - otimização, 33
- Cadeia de Pensamento (CoT), 45
- capacidades
 - de tomada de decisão, 99
- casos extremos, 58
- Chain of Thought (CoT), 138
- chamada de ferramenta, 153
- chamada de função
 - falha, 133
- chatbots de atendimento ao cliente, 34
- ChatGPT, 30, 53
- ciência da computação, 70, 72
- classificadores por ordem, 36
- classificação, 53, 120
- Claude, 8, 44, 77
- Claude 3, 50, 126, 129, 134, 136
- Claude 3 Opus, 74
- Claude v1, 17
- Claude v2, 17
- Cohere (Fornecedor de LLM), 23, 25
- comportamento determinístico, 58
- computadores de secretária, 217
- comércio eletrônico, 191, 220
- conclusão de alto desempenho, 26
- concurrent workflows, 252
- conectividade de rede, 225
- confiança do utilizador, 216
- conjuntos, 118
 - conjunto de trabalhadores, 118
- consistência
 - e reprodutibilidade, 132
- construção narrativa, 20
- conta, 91
- contexto
 - Aumento, 46
 - entradas infinitamente longas, 16
 - Geração Contextual de Conteúdo, 190–192
 - Geração de Conteúdo Contextual, 186, 198, 199
 - janela, 15, 224
 - Sugestões Contextuais de Campos, 199
 - tomada de decisão contextual, 224
- conteúdo
 - Categorização de Conteúdo, 112
 - filtragem, 27
- conteúdo gerado pelo utilizador, 111
- Continuação Automática, 160
- conversa
 - loop, 159
 - transcrição, 157, 159
- conversação
 - ciclo, 157
- correspondência de padrões, 152

- dados
 - análise, 34, 147
 - fluxo, 110
 - integridade, 238
 - persistência, 109
 - pipeline de processamento, 238
 - preparação, 109
 - privacidade, 27, 215
 - Recuperação de Dados, 109
 - Sincronização de Dados, 109
 - tarefas de processamento, 125
 - Validação de Dados, 257
- Dados Auto-Reparáveis, 163, 242
- dados de treino, 42
- dados em streaming, 152
- dados estruturados, 134
- Datadog, 246
- decisão
 - pontos, 243
 - árvores, 220
- Delimitação de Respostas, 204
- depuração, 224
 - e resolução de problemas, 245
 - e testes, 132
- desafios conceituais e práticos, 198
- descobertas médicas, 100
- desempenho
 - compromissos, 5
 - otimização, 132, 195, 245
 - problemas, 250
- desenvolvimento de aplicações, 219
- design e frameworks de aplicações, 197
- deteção de fraude
 - sistema, 97
- dicionários, 130
- diretiva do sistema, 99, 128
- Dohan, et al., 44
- E/S Estruturada, 204
- ecossistema, 148
- eficiência, 221
- ELK stack, 110
- encadeamento de workers de IA, 111
- Encaminhamento Dinâmico de Tarefas, 222
- ensembles, 117
- Enterprise Integration Patterns, 104
- errors
 - handling, 253
 - Intelligent Error Handling, 143
- erros
 - recuperação, 257
 - taxas, 110
 - tratamento, 107, 110, 142
- erros de sintaxe, 131
- escalabilidade, 221, 247
- escrita criativa, 34, 52
- espaço latente, 40, 42
- Estratificação de Risco, 102
- estratégias de contingência, 110
- estratégias de segmentação e direcionamento, 193
- estratégias motivacionais, 212
- estreitar o caminho, 39
- eventos enviados pelo servidor (SSE), 150

- execução paralela, 248
- experimentação
 - estrutura, 193
- experiência do utilizador, 193
- explicabilidade, 256
- F#, 93
- Facebook, 24
- fatores de risco, 95, 96
- feedback
 - Ciclo de Feedback, 59
- few-shot
 - aprendizagem, 62
 - prompting, 63
- filtragem baseada em conteúdo, 92
- filtragem colaborativa, 92
- FitAI, 210
- flexibilidade e criatividade, 195
- fluxo de trabalho adaptativo
 - Composição Adaptativa de Fluxos de Trabalho, 224
- fluxo de trabalho multi-etapa, 111
- fornecedores de hospedagem de modelos
 - open source, 204
- frameworks de desenvolvimento, 148
- funcionários da Databricks, 52
- função
 - chamada, 123, 157
 - histórico de chamadas, 156
 - nomes, 154
- Gemma 7B, 11
- Generative Pre-trained Transformer (GPT),
 - 8
- Geração Aumentada por Recuperação (GAR), 79
- Geração Aumentada por Recuperação (RAG), 32, 38, 46, 125
- geração cross-modal, 22
- geração de dados sintéticos, 53
- geração dinâmica de UI, 187
- Gestor de Processos, 107
- gestão de tráfego, 33
- gestão do conhecimento, 32
- GitLab, 93
- Global Interpreter Lock (GIL), 115
- Google, 23
 - API, 63, 65
 - Cloud Platform, 250
 - Gemini, 21
 - Gemini 1.5 Pro, 14, 17, 18
 - PaLM (Pathways Language Model), 17, 24
 - Plataforma de IA na Nuvem, 24
 - T5, 14
- GPT-3, 13, 17
- GPT-4, 6, 13, 17, 21, 31, 44, 50, 62, 105, 117, 120, 127, 133, 202, 203, 248
- Graham, Paul, 19
- GraphQL, 108
- Groq, 26, 120
- gzip, 252
- hardware, 28

- hash, 152
- hiperparâmetro, 47
- Hohpe, Gregor, 104
- Honeybadger, 94
- HTTP, 150
- Human-In-The-Loop (HITL), 178
- IA, 64, 73, 99, 128, 134, 150, 201, 209
 - aplicações, 125, 138, 149, 162
 - conversacional, 6, 31, 211
 - modelo, 89, 98, 99, 155, 156, 158, 209
 - pontos de decisão, 255
 - sistemas compostos, 30, 31, 35
- identificação de tópicos, 120
- idioma
 - Deteção de Idioma, 112
- Inferência, 5
- informação
 - extração, 53
 - recuperação, 7, 126
- injeções SQL, 70
- input
 - prompts, 56
 - validation, 252
- integração de LLMs, 187
- Integração e Implementação Contínuas
 - (CI/CD), 258
 - pipeline, 258
- intelligent workflow orchestration, 252
- interações estilo roleplay, 6
- interface adaptativa, 207
- Interface de Utilizador (UI)
 - interfaces, 197
- Interface de Utilizador Generativa (GenUI), 197
- Interface do Utilizador (UI)
 - design, 217
 - estruturas, 213
 - interfaces, 213
 - tecnologias, 208
- Interface Generativa (GenUI), 204, 205, 209, 213, 216
- interface visual, 208
- interfaces controladas por voz, 34
- interfaces inclusivas, 198
- internacionalização, 194
- Interpretador de Resultados, 142
- intervenção manual, 227
- iterative refinement, 144
- JSON (JavaScript Object Notation), 126, 130, 131, 134, 147, 166
- K-means, 122
- Large Language Model (LLM), 144
- latência, 28
- Limpeza de Texto, 111
- linguagem
 - modelos, 43, 65, 72
 - tarefas relacionadas, 5
- linguagem codificável em Unicode, 15
- linguagem natural
 - Processamento de Linguagem Natural (NLP), 120

- Processamento de Linguagem Natural
 - (PLN), 101
- linha de comandos
 - Interface de Linha de Comandos
 - (CLI), 26
- Llama, 13
- Llama 2-70B, 50
- Llama 3 70B, 11
- Llama 3 8B, 11
- Louvre, 42
- lógica de disjuntor, 161
- Managed Streaming for Apache Kafka, 41
- manipuladores de stream, 151
- marcação em estilo markup, 70
- Markdown, 147
- mecanismos de nova tentativa, 110
- mecanismos de reversão, 259
- Memorial Sloan Kettering Cancer Center,
 - 41
- mensagem de gatilho, 104
- Mercúrio (deus romano), 45
- Mercúrio (elemento), 45
- Mercúrio (planeta), 45
- MessagePack, 251
- Meta, 24
- Metropolitan Museum of Art, 42
- Mistral, 25
 - 7B, 11
 - 7B Instruct, 17, 203
- Mixtral
 - 8x22B, 11
 - 8x7B, 56
- modelagem autorregressiva, 43
- Modelo de Linguagem de Larga Escala
 - (LLM), 1, 3
- Modelo de Linguagem Grande (LLM), 16,
 - 18, 29, 67, 68, 71, 75, 111, 120,
 - 123, 124, 134, 144, 147, 163, 166,
 - 186, 208, 230
- panorama, 27
- Modelo de Linguagem Grande (MLG), 77,
 - 87, 140, 202
- Modelo Linguístico de Grande Escala
 - (LLM), 197
- modelo OPT, 24
- modelos base, 54
- modelos baseados em recuperação, 7
- modelos gráficos, 43
- modelos probabilísticos, 43
- Moderador de Conteúdo Inteligente, 231
- modularidade, 88
- monitorização
 - e alerta, 226
 - e registro, 110, 245
 - métricas, 246
- Monitorização Contínua de Risco, 103
- Multi-Agente
 - Solucionadores de Problemas, 31
- Multimodal
 - modelos, 20
 - modelos de linguagem, 21
- Multiplicidade de Trabalhadores, 119, 165
- Máquinas de Vetores de Suporte (SVM), 121

- método finalize, 156, 158, 159
- Naive Bayes, 121
- New Relic, 249
- Ollama, 25
- Olympia, 34, 62, 128, 143, 151, 166
- OpenAI, 3, 23, 39, 73
- OpenRouter, 27, 28, 151, 249
- orquestração de fluxo de trabalho
 - inteligente, 227
- orquestração inteligente de fluxos de trabalho, 219, 249
- output verification, 253
- padrões históricos, 224
- padrões principais, 222
- panorama digital, 192
- paráfrase, 53
- parâmetro
 - Contagem de Parâmetros, 28
 - efeitos, 128
 - intervalo, 11
- parâmetros
 - de entrada, 128
- Penalização de Presença, 48
- penalizações de repetição, 51
- Perplexity (Fornecedor), 12
- personalização, 27, 187, 217, 221
 - Formulários Personalizados, 199
 - Microtexto Personalizado, 205
- planeamento de resposta a emergências, 33
- pontos de estrangulamento, 224
- previsões, 5
- princípio do menor privilégio, 71
- problemas de usabilidade, 215
- Process Manager, 104
 - Enterprise Integration, 228
- processamento assíncrono, 247
- processamento de stream, 156, 162
 - lógica, 158
- processamento de streams, 150
- processamento em lote, 248
- processo de destilação, 75
- Produtividade, 189
- programação funcional, 92
- prompts
 - design, 58, 68
 - Destilação de Prompts, 46, 73, 77, 248
 - encadeamento, 59, 71
 - engenharia, 41, 45, 46, 56, 59, 65, 67, 214
 - Modelo de Prompt, 59, 204
 - Objeto Prompt, 74
 - refinamento, 68
- propriedades ACID, 110
- Protocol Buffers, 251
- psicologia do utilizador, 214
- PyTorch, 24
- Quantização, 28
- Qwen2 70B, 11
- Rails, 194
- Railway Oriented Programming (ROP), 95
- Raix, 228

- biblioteca, 97
- Recolha de Histórico Médico, 101
- Recomendações de Produtos, 92
- recomendações personalizadas de produtos, 92
- redes neuronais, 4, 6
- refinamento iterativo, 75
- registro de auditoria, 106
- registro estruturado, 246
- registro granular, 246
- regras de negócio, 220
- regras gramaticais, 4
- regressão linear, 43
- Response Fencing, 175
- resposta a perguntas fechadas e abertas, 52
- restringir o caminho, 38
- retalhistas online, 204
- retenção e rotação de registos, 246
- revelação progressiva, 206
- RSpec, 253, 254, 257
- Ruby, 93, 94, 113, 162, 257
- Ruby on Rails, 1, 111, 228, 235
- Rudall, Alex, 23
- Rust (Linguagem de Programação), 93, 116
- Scout, 249
- Seleção Dinâmica de Ferramentas, 130
- Seleção Forçada de Ferramentas, 131
- sem estado, 157
- serviços externos ou APIs, 126
- sistemas de pergunta e resposta, 7
- sistemas de publicação-subscrição, 108
- smartphones, 217
- Stripe, 129
- sumarização, 52
- suporte ao cliente, 32
- T5, 24
- tablets, 217
- tarefas complexas, 146
- taxa de transferência, 28
- Temperatura, 54
- Tempo até o Primeiro Token (TTFT), 28
- tempo de processamento, 110
- teoria da mente, 40
- teste ponta a ponta, 257, 258
- testes com utilizadores e feedback, 196
- testes de integração, 254
- Together.ai, 26
- tokenização, 12
- tokens, 6, 12
- tom emocional, 145
- tomada de decisão
 - casos de uso, 133
- tradução, 16, 195
- tragédia dos comuns, 190
- Transformador Pré-treinado Generativo (GPT), 67
- tratamento de exceções, 225, 227
- Universal ID, 251
- uso de ferramentas, 123, 149
- Ventriloquist, 175
- Verificação de Seguro, 101

viés

 e equidade em IA, 256

votação por maioria, 117

Wall, Larry, 3

Wisper, 94, 106, 151, 158

Wooley, Chad, 93

XML, 134

Yi-34B, 50

álgebra linear, 43

ética

 implicações, 198

óculos de realidade aumentada, 217