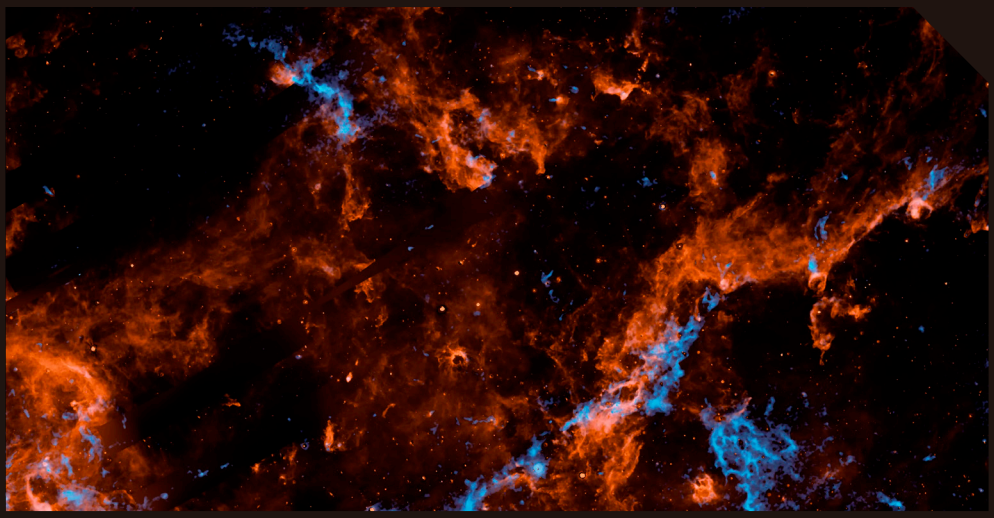
Blucher



Manual do Engenheiro de LLM

Domine a arte da engenharia de grandes modelos de linguagem

PAUL IUSZTIN
MAXIME LABONNE

Manual do Engenheiro de LLM

Domine a arte da engenharia de grandes modelos de linguagem

Paul Iusztin

Maxime Labonne

Manual do Engenheiro de LLM: domine a arte da engenharia de grandes modelos de linguagem
Copyright © 2024 Packt Publishing
© 2026 Paul Iusztin, Maxime Labonne
Editora Edgard Blücher Ltda.

Publisher Edgard Blücher
Editor Eduardo Blücher
Coordenação editorial Rafael Fulanetti
Coordenação de produção Ana Cristina Garcia
Produção editorial Juliana Midori Horie
Diagramação Guilherme Salvador
Revisão de texto Susane Schmieg
Capa Equipe editorial
Imagem da capa NASA/JPL-Caltech/IPAC/Hora et al.

Blucher

Rua Pedroso Alvarenga, 1245, 4º andar
04531-934 – São Paulo – SP – Brasil
Tel.: 55 11 3078-5366
contato@blucher.com.br
www.blucher.com.br

Segundo o Novo Acordo Ortográfico, conforme 6. ed. do *Vocabulário Ortográfico da Língua Portuguesa*, Academia Brasileira de Letras, julho de 2021.

É proibida a reprodução total ou parcial por quaisquer meios sem autorização escrita da editora.

Todos os direitos reservados pela Editora Edgard Blücher Ltda.

Dados Internacionais de Catalogação na Publicação (CIP)
Heytor Diniz Teixeira, CRB-8/10570

Iusztin, Paul

Manual do engenheiro de LLM : domine a arte da engenharia de grandes modelos de linguagem / Paul Iusztin, Maxime Labonne. – São Paulo : Blucher, 2026.

470 p. : il.

Bibliografia
ISBN 978-85-212-3347-3 (Impresso)
ISBN 978-85-212-3345-9 (Eletrônico – Epub)
ISBN 978-85-212-3343-5 (Eletrônico – PDF)

Título original: *LLM Engineer's Handbook: Master the art of engineering large language models from concept to production*

1. Inteligência artificial. 2. Computação. 3. Large language models (LLMs). 4. Grandes modelos de linguagem. I. Título. II. Labonne, Maxime.

CDU 004.8

Índice para catálogo sistemático:
1. Inteligência artificial CDU 004.8

Textos introdutórios

Como meu cofundador na Hugging Face, Clement Delangue, e eu costumamos dizer, a IA está se tornando a forma padrão de desenvolver tecnologia.

Nos últimos 3 anos, LLMs já tiveram um impacto profundo na tecnologia, e estão destinados a ter um impacto ainda maior nos próximos 5 anos. Eles estarão incorporados a cada vez mais produtos e, acredito, no centro de qualquer atividade humana baseada em conhecimento ou criatividade.

Por exemplo, programadores já estão aproveitando LLMs e mudando a forma como trabalham, focando em pensamento e tarefas de ordem superior enquanto colaboram com máquinas. Músicos de estúdio contam com ferramentas impulsionadas por IA para acelerar a exploração do espaço da criatividade musical. Advogados estão ampliando seu impacto por meio da **geração aumentada por recuperação** (**RAG**) e grandes bases de dados de jurisprudência.

Na Hugging Face, sempre defendemos um futuro em que não apenas uma empresa ou um pequeno número de cientistas controlem os modelos de IA usados pelo restante da população, mas sim um futuro em que o maior número possível de pessoas — dos perfis mais diversos possíveis — sejam capazes de aprofundar-se no modo como modelos de machine learning de ponta realmente funcionam.

Maxime Labonne e Paul Iusztin foram fundamentais neste movimento para democratizar LLMs ao escrever este livro e garantir que o maior número possível de pessoas não apenas possa usá-los, mas também adaptá-los, realizar ajuste fino, quantizá-los e torná-los eficientes o suficiente para, de fato, implantá-los no mundo real.

O trabalho deles é essencial, e fico feliz por estarem disponibilizando este recurso para a comunidade. Isso expande a casca convexa do conhecimento humano.

Julien Chaumond

Cofundador e CTO, Hugging Face

Como alguém profundamente imerso no mundo de operações de aprendizado de máquina, estou entusiasmado em endossar *Manual do Engenheiro de LLM*. Este guia abrangente chega em um momento crucial, quando a demanda por especialização em LLMs está disparando em todos os setores.

O que diferencia este livro é sua abordagem prática, de ponta a ponta. Ao conduzir os leitores pela criação de um LLM Twin, ele preenche a lacuna frequentemente assustadora entre teoria e aplicação no mundo real. Da engenharia de dados e do ajuste fino de modelos a tópicos avançados como pipelines de RAG e otimização de inferência, os autores não deixam nada de fora.

Estou particularmente impressionado com a ênfase nos princípios de MLOps e LLMOps. À medida que as organizações dependem cada vez mais de LLMs, entender como construir sistemas escaláveis, reproduzíveis e robustos é primordial. A inclusão de estratégias de orquestração e integração com a nuvem demonstra o compromisso dos autores em capacitar os leitores com habilidades verdadeiramente prontas para produção.

Se você é um profissional experiente de ML buscando se especializar em LLMs ou um engenheiro de software que deseja ingressar nesse campo empolgante, este manual oferece a combinação perfeita de conhecimento fundamental e técnicas de ponta. As explicações claras, os exemplos práticos e o foco nas melhores práticas fazem dele um recurso inestimável para qualquer pessoa séria em dominar a engenharia de LLM.

Em uma era em que a IA está remodelando setores a uma velocidade vertiginosa, *Manual do Engenheiro de LLM* destaca-se como um guia essencial para navegar pelas complexidades dos modelos de linguagem de grande porte. Não é apenas um livro; é um roteiro para se tornar um engenheiro de LLM proficiente no cenário atual movido por IA.

Hamza Tahir

Cofundador e CTO, ZenML

O livro *Manual do Engenheiro de LLM* serve como um recurso inestimável para quem busca uma compreensão prática de LLMs. Por meio de exemplos práticos e de uma exploração abrangente do projeto LLM Twin, o autor desmistifica de forma eficaz as complexidades da construção e implantação de aplicações de LLM em nível de produção.

Uma das características de destaque do livro é o uso do projeto LLM Twin como um exemplo contínuo. Este personagem de IA, projetado para emular o estilo de escrita de um indivíduo específico, fornece uma ilustração tangível de como LLMs podem ser aplicados em cenários do mundo real.

O autor conduz habilmente os leitores pelas ferramentas e tecnologias essenciais necessárias para o desenvolvimento de LLMs, incluindo Hugging Face, ZenML, Comet, Opik, MongoDB e Qdrant. Cada ferramenta é explicada em detalhes, facilitando que os leitores entendam suas funções e como podem ser integradas a um pipeline de LLM.

Manual do Engenheiro de LLM também aborda uma ampla gama de tópicos relacionados ao desenvolvimento de LLMs, como coleta de dados, ajuste fino, avaliação, otimização de inferência e MLOps. Notavelmente, os capítulos sobre ajuste fino supervisionado, alinhamento por preferência e **geração aumentada por recuperação** (**RAG**) fornecem insights aprofundados sobre esses aspectos críticos do desenvolvimento de LLMs.

Um ponto forte particular deste livro está em seu foco na implementação prática. O autor se destaca em fornecer exemplos concretos e orientações sobre como otimizar pipelines de inferência e implantar LLMs de forma eficaz. Isso torna o livro um recurso valioso tanto para pesquisadores quanto para profissionais.

Este livro é altamente recomendado para qualquer pessoa interessada em aprender sobre LLMs e suas aplicações práticas. Ao fornecer uma visão geral abrangente das ferramentas, técnicas e melhores práticas envolvidas no desenvolvimento de LLMs, os autores criaram um recurso valioso que, sem dúvida, será uma referência para muitos engenheiros de LLM.

Antonio Gulli

Diretor Sênior, Google

Sumário

Prefácio xxi

1

Entendendo o conceito e a arquitetura do LLM Twin 1

Compreendendo o conceito de LLM Twin 2

O que é um LLM Twin? 2

Por que construir um LLM Twin é importante 3

Por que não usar o ChatGPT (ou outro chatbot semelhante)? 4

Planejamento do MVP do produto LLM Twin 6

O que é um MVP? 6

Definindo o MVP do LLM Twin 7

Construindo sistemas de ML com pipelines de atributos/treinamento/inferência 8

O problema com o desenvolvimento de sistemas de ML 8

O problema com as soluções anteriores 10

A solução – pipelines de ML para sistemas de aprendizado de máquina 12

O pipeline de características 13

O pipeline de treinamento 14

O pipeline de inferência 14

Benefícios da arquitetura FTI 14

Projetando a arquitetura do sistema do LLM Twin 16
Listando os detalhes técnicos da arquitetura do LLM Twin 16
Como projetar a arquitetura do LLM Twin usando o design do pipeline de FTI 17
Pipeline de coleta de dados 18
Pipeline de características 19
Pipeline de treinamento 20
Pipeline de inferência 21
Considerações finais sobre o design do FTI e a arquitetura do LLM Twin 22
Resumo 23
Referências 23

2
Ferramentas e instalação 25

Ecossistema do Python e instalação do projeto 26
Poetry: gerenciamento de dependências e de ambientes virtuais 27
Poe the Poet: ferramenta de execução de tarefas 29
Ferramentas de MLOps e LLMOps 30
Hugging Face: registro de modelos 31
ZenML: orquestrador, artefatos e metadados 32
Orquestrador 33
Artefatos e metadados 37
Como executar e configurar um pipeline do ZenML 41
Comet ML: rastreador de experimentos 43
Opik: monitoramento de prompts 44
Bancos de dados para armazenamento de dados não estruturados e vetoriais 45
MongoDB: banco de dados NoSQL 45
Qdrant: banco de dados vetorial 45
Preparação para a AWS 46
Configurando uma conta da AWS, uma chave de acesso e a CLI 46
Amazon SageMaker: computação para treinamento e inferência 48
Por que optar pelo AWS SageMaker? 49
Resumo 50
Referências 50

3
Engenharia de dados 53

Projetando o pipeline de coleta de dados do LLM Twin 54

Implementando o pipeline de coleta de dados do LLM Twin 58
Pipeline e etapas do ZenML 58
O despachante: como instanciar o rastreador correto? 63
Os rastreadores 65
Classes base 66
Classe GitHubCrawler 69
Classe CustomArticleCrawler 71
Classe MediumCrawler 73
Os documentos do data warehouse NoSQL 76
Os padrões de software ORM e ODM 76
Implementar a classe ODM 78
Categorias de dados e classes de documentos do usuário 83
Coletando dados brutos no data warehouse 85
Solução de problemas 89
Problemas do Selenium 89
Importe nossos dados de backup 90
Resumo 90
Referências 91

4

Pipeline de atributos de RAG 93

Entendendo a RAG 94
Por que usar RAG? 94
Alucinações 95
Informação antiga 95
O framework RAG básico 95
Pipeline de ingestão de dados 97
Pipeline de recuperação 98
Pipeline de geração 99
O que são embeddings? 100
Por que embeddings são tão poderosos 102
Como os embeddings são criados? 104
Aplicações de embeddings 107
Mais sobre bancos de dados vetoriais 107
Como funciona um banco de dados vetorial? 108
Algoritmos para a criação do índice vetorial 109
Operações de banco de dados 109

Uma visão geral de RAG avançada 110
Pré-recuperação 111
Recuperação 114
Pós-recuperação 116
Explorar a arquitetura do pipeline da funcionalidade de RAG do LLM Twin 118
O problema que estamos resolvendo 119
O feature store 119
De onde vêm os dados brutos? 120
Projetando a arquitetura do pipeline do atributo RAG 120
Pipelines em lote 121
Pipelines em lote versus pipelines de streaming 122
Etapas principais 125
Captura de alterações de dados: sincronizando o data warehouse e a feature store 126
Por que os dados são armazenados em dois snapshots? 129
Orquestração 129
Implementando o pipeline do atributo RAG do LLM Twin 130
Configurações 130
Pipeline e etapas do ZenML 131
Consultando o data warehouse 133
Limpeza de dados dos documentos 136
Fragmentar e gerar embeddings dos documentos limpos 137
Carregando os documentos no banco de dados vetorial 139
Entidades de domínio do Pydantic 140
OVM 143
A camada despachante 149
Os handlers 151
Os handlers de limpeza 152
Os handlers de fragmentação 153
Os handlers de embedding 157
Resumo 161
Referências 161

5

Ajuste fino supervisionado 163

Criação de um conjunto de dados de instruções 164
Arcabouço geral 164

Quantidade de dados 166
Curadoria de dados 167
Filtragem baseada em regras 168
Deduplicação de dados 169
Descontaminação de dados 171
Avaliação da qualidade dos dados 171
Exploração de dados 174
Geração de dados 176
Aumento de dados 177
Criação do nosso próprio conjunto de dados de instruções 179
Explorando o SFT e suas técnicas 188
Quando realizar o ajuste fino 188
Formatos de conjuntos de dados de instruções 189
Templates de chat 190
Técnicas de ajuste fino com eficiência de parâmetros 192
Ajuste fino completo 192
LoRA 194
QLoRA 196
Parâmetros de treinamento 197
Taxa de aprendizado e agendador 197
Tamanho do lote 198
Comprimento máximo e empacotamento 199
Número de epochs 199
Otimizadores 200
Decaimento de pesos 200
Checkpointing de gradiente 200
Ajuste fino na prática 201
Resumo 208
Referências 208

6

Ajuste fino com alinhamento às preferências humanas 211

Compreendendo conjuntos de dados de preferências 212
Dados de preferências 212
Quantidade de dados 214
Geração de dados e avaliação 215
Geração de preferências 215
Dicas para geração de dados 216

Avaliação de preferências 217
Criação do nosso próprio conjunto de dados de preferências **219**
Alinhamento às preferências humanas **226**
Aprendizado por reforço a partir de feedback humano 227
Otimização Direta de Preferências (DPO) 228
Implementando DPO **230**
Resumo **237**
Referências **238**

7

Avaliação de LLMs **239**

Avaliação de modelos **240**
Comparando a avaliação de ML e de LLM 240
Avaliações de LLMs de propósito geral 241
Avaliações de LLMs específicas de domínio 243
Avaliações de LLMs específicas à tarefa 245
Avaliação de RAG **248**
Ragas 249
ARES 251
Avaliando o TwinLlama-3.1-8B **252**
Geração de respostas 253
Avaliação de respostas 254
Analisando resultados 259
Resumo **262**
Referências **263**

8

Otimização de inferência **265**

Estratégias de otimização de modelos **266**
Cache KV 267
Processamento contínuo em lotes 269
Decodificação especulativa 270
Mecanismos de atenção otimizados 272
Paralelismo de modelo **273**
Paralelismo de dados 274

Paralelismo de pipeline 275
Paralelismo de tensores 276
Combinando abordagens 277
Quantização de modelos 278
Introdução à quantização 278
Quantização com GGUF e llama.cpp 282
Quantização com GPTQ e EXL2 284
Outras técnicas de quantização 286
Resumo 287
Referências 288

9

Pipeline de inferência de RAG 289

Compreendendo o pipeline de inferência de RAG do LLM Twin 290
Explorando as técnicas avançadas de RAG do LLM Twin 292
Otimizações avançadas de pré-recuperação para RAG: expansão de consulta e autoextração de metadados da consulta 295
Expansão de consulta 295
Autoextração de metadados da consulta 299
Otimização avançada da recuperação em RAG: busca vetorial filtrada 303
Otimização pós-recuperação avançada em RAG: rerranqueamento 304
Implementando o pipeline de inferência RAG do LLM Twin 309
Implementando o módulo de recuperação 309
Reunindo tudo no pipeline de inferência de RAG 316
Resumo 321
Referências 321

10

Implantação do pipeline de inferência 323

Critérios para escolher tipos de implantação 324
Vazão e latência 324
Dados 325
Infraestrutura 325
Entendendo os tipos de implantação de inferência 326

Inferência online em tempo real 328
Inferência assíncrona 328
Transformação em lote offline 329

Arquitetura monolítica versus arquitetura de microsserviços no serviço de modelos 330

Arquitetura monolítica 331
Arquitetura de microsserviços 332
Escolhendo entre arquiteturas monolíticas e de microsserviços 333

Explorando a estratégia de implantação do pipeline de inferência do LLM Twin 334

O treinamento versus o pipeline de inferência 337

Implantação do serviço LLM Twin 338

Implementando o microsserviço de LLM usando AWS SageMaker 339

O que são os DLCs da Hugging Face? 339
Configurando funções do Amazon SageMaker 340
Implantação do modelo LLM Twin no AWS SageMaker 341
Chamando o endpoint de inferência do AWS SageMaker 351

Construindo o microserviço de negócios usando FastAPI 355

Capacidades de escalonamento automático para lidar com picos de uso 358

Registrando um alvo escalável 360
Criação de uma política escalável 361
Limites mínimos e máximos de normalização 362

Período de cooldown 362

Resumo 363
Referências 363

11

MLOps e LLMOps 365

O caminho para o LLMOps: compreendendo suas raízes em DevOps e MLOps 366

DevOps 366

O ciclo de vida do DevOps 367
Os conceitos fundamentais de DevOps 368

MLOps 369

Componentes centrais de MLOps 370
Princípios de MLOps 371

ML vs. engenharia de MLOps 372
LLMOps 373
Feedback humano 374
Salvaguardas 374
Monitoramento de prompts 376
Implantação dos pipelines do LLM Twin na nuvem 378
Compreendendo a infraestrutura 378
Configurar o MongoDB 381
Configurando o Qdrant 382
Configurando a nuvem do ZenML 383
Containerizar o código usando Docker 386
Execute os pipelines na AWS 391
Solução de problemas do erro ResourceLimitExceeded após a execução de um pipeline do ZenML no Amazon SageMaker 393
Adicionando LLMOps ao LLM Twin 394
Fluxo do pipeline de CI/CD do LLM Twin 395
Mais sobre erros de formatação 397
Mais sobre erros de linting 397
Visão geral rápida do GitHub Actions 397
O pipeline de CI 399
Arquivo YAML de CI do GitHub Actions 399
O pipeline de entrega contínua 402
Teste o pipeline de integração contínua e entrega contínua 405
O pipeline de CT 406
Gatilhos iniciais 407
Acionar pipelines a jusante 408
Monitoramento de prompts 411
Alertas 416
Resumo 417
Referências 417

Apêndice

Princípios de MLOps 419

1. Automação ou operacionalização 419
2. Versionamento 421
3. Rastreamento de experimentos 422

4. Testes **422**
Tipos de teste 422
O que testamos? 423
Exemplos de teste 423
5. Monitoramento **426**
Logs 426
Métricas 426
Métricas do sistema 427
Métricas do modelo 427
Derivas 427
Deriva de dados 428
Deriva do alvo 428
Deriva de conceito 429
Como detectar e medir derivas 430
Monitoramento vs. observabilidade **430**
Alertas 431
6. Reprodutibilidade **431**
Índice remissivo **433**

Prefácio

O campo da engenharia de **LLM** emergiu rapidamente como uma área crítica em inteligência artificial e machine learning. À medida que os LLMs continuam a revolucionar o processamento e a geração de linguagem natural, a demanda por profissionais que possam implementar, otimizar e implantar esses modelos de forma eficaz em cenários do mundo real cresceu exponencialmente. A engenharia de LLM abrange uma ampla gama de disciplinas, desde a preparação de dados e o ajuste fino de modelo até a otimização de inferência e a implantação em produção, exigindo uma combinação única de engenharia de software, especialização em machine learning e conhecimento de domínio.

Operações de Machine Learning (**MLOps**) desempenha um papel crucial na implementação bem-sucedida de LLMs em ambientes de produção. MLOps estende os princípios de DevOps a projetos de machine learning, focando em automatizar e racionalizar todo o ciclo de vida de ML. Para LLMs, MLOps é particularmente importante devido à complexidade e à escala desses modelos. Ele aborda desafios como gerenciar grandes conjuntos de dados, lidar com o versionamento de modelos, garantir reprodutibilidade e manter a performance do modelo ao longo do tempo. Ao incorporar práticas de MLOps, projetos de LLM podem alcançar maior eficiência, confiabilidade e escalabilidade, resultando, em última instância, em implantações mais bem-sucedidas e impactantes.

O *Manual do Engenheiro de LLM* é um guia abrangente para aplicar as melhores práticas ao novo campo da engenharia de LLM. Ao longo dos capítulos, os leitores encontrarão conceitos-chave simplificados, técnicas práticas e dicas de especialistas para cada etapa do ciclo de vida de LLMs. O livro aborda tópicos como engenharia de dados, ajuste fino supervisionado, avaliação de modelo, otimização de inferência e **desenvolvimento de pipelines de Retrieval-Augmented Generation** (**RAG**).

Para ilustrar esses conceitos em ação, um projeto de ponta a ponta chamado LLM Twin será desenvolvido ao longo do livro, com o objetivo de imitar o estilo de escrita e a personalidade

de alguém. Esse caso de uso demonstrará como construir um produto mínimo viável para resolver um problema específico, utilizando vários aspectos da engenharia de LLM e de MLOps.

Os leitores podem esperar obter uma compreensão mais profunda de como coletar e preparar dados para LLMs, realizar ajuste fino de modelos para tarefas específicas, otimizar a performance de inferência e implementar pipelines de RAG. Eles aprenderão como avaliar a performance de LLMs, alinhar modelos com preferências humanas e implantar aplicações baseadas em LLMs. O livro também aborda princípios e práticas essenciais de MLOps, permitindo que os leitores construam aplicações baseadas em LLMs escaláveis, reprodutíveis e robustas.

Para quem é este livro

Este livro se destina a uma ampla gama de profissionais e entusiastas de tecnologia interessados nas aplicações práticas de LLMs. É ideal para engenheiros de software que pretendem fazer a transição para projetos de IA. Embora seja útil ter alguma familiaridade com desenvolvimento de software, o livro explica muitos conceitos do zero, tornando-o acessível mesmo para quem é iniciante em IA e aprendizado de máquina.

Para quem já trabalha com aprendizado de máquina, este livro aprimorará suas habilidades de implementar e implantar sistemas baseados em LLMs. Oferecemos um estudo aprofundado dos fundamentos de MLOps, guiando você pelo processo de criação de um produto mínimo viável usando um LLM de código aberto para resolver problemas do mundo real.

O que este livro abrange

Capítulo 1, *Entendendo o conceito e a arquitetura do LLM Twin*, apresenta o projeto LLM Twin, que é usado ao longo do livro como um exemplo completo de uma aplicação de LLM em nível de produção, e define a arquitetura FTI para construir sistemas de ML escaláveis e a aplica ao caso de uso do LLM Twin.

Capítulo 2, *Ferramentas e instalação*, apresenta Python, MLOps e ferramentas de nuvem usadas para construir aplicações de LLM do mundo real, como um orquestrador, rastreador de experimentos, ferramenta de monitoramento de prompts e avaliação de LLM. Mostra como usá-las e instalá-las localmente para teste e desenvolvimento.

Capítulo 3, *Engenharia de dados*, mostra a implementação de um pipeline de coleta de dados que faz raspagem de vários sites, como Medium, GitHub e Substack, e armazena os dados brutos em um data warehouse. Enfatiza a coleta de dados brutos de fontes dinâmicas em vez de conjuntos de dados estáticos para aplicações de ML do mundo real.

Capítulo 4, *Pipeline de atributos de RAG*, apresenta conceitos fundamentais de RAG, como embeddings, o framework RAG básico, bancos de dados vetoriais e como otimizar aplicações RAG. Aplica a teoria de RAG ao arquitetar e implementar o pipeline de funcionalidade de RAG do LLM Twin usando as melhores práticas de software.

Capítulo 5, *Ajuste fino supervisionado*, explora o processo de refinar modelos de linguagem pré-treinados para tarefas específicas usando pares de instrução-resposta. Cobre a criação de conjuntos de dados de alta qualidade, a implementação de técnicas de ajuste fino como ajuste fino completo, LoRA e QLoRA, e fornece uma demonstração prática de ajuste fino de um modelo Llama 3.1 8B em um conjunto de dados personalizado.

Capítulo 6, *Ajuste fino com alinhamento às preferências humanas*, apresenta técnicas para alinhar modelos de linguagem às preferências humanas, com foco em **Otimização Direta de Preferências** (**DPO**). Cobre a criação de conjuntos de dados de preferências personalizados, a implementação de DPO, e fornece uma demonstração prática de alinhamento do modelo TwinLlama-3.1-8B usando a biblioteca Unsloth.

Capítulo 7, *Avaliação de LLMs*, detalha vários métodos para avaliar a performance de modelos de linguagem e sistemas de LLM. Introduz avaliações de uso geral e específico de domínio e discute benchmarks populares. O capítulo inclui uma avaliação prática do modelo TwinLlama-3.1-8B usando múltiplos critérios.

Capítulo 8, *Otimização de inferência*, aborda estratégias-chave de otimização como decodificação especulativa, paralelismo de modelo e quantização de pesos. Discute como melhorar a velocidade de inferência, reduzir a latência e minimizar o uso de memória, apresentando motores de inferência populares e comparando suas características.

Capítulo 9, *Pipeline de inferência de RAG*, explora técnicas avançadas de RAG, implementando, do zero, métodos como self-query, rerranqueamento e busca vetorial filtrada. Abrange o projeto e a implementação do pipeline de inferência de RAG do LLM Twin e de um módulo de recuperação personalizado semelhante ao que você vê em frameworks populares como o LangChain.

Capítulo 10, *Implantação do pipeline de inferência*, apresenta estratégias de implantação de ML, como inferência online, assíncrona e em lote, que ajudarão a arquitetar e implantar o modelo com ajuste fino do LLM Twin no AWS SageMaker e a construir um microsserviço FastAPI para expor o pipeline de inferência de RAG como uma API RESTful.

Capítulo 11, *MLOps e LLMOps*, apresenta o que é LLMOps, começando por suas raízes em DevOps e MLOps. Este capítulo explica como implantar o projeto LLM Twin na nuvem, como, por exemplo, os pipelines de ML para a AWS, e mostra como containerizar o código usando Docker e criar um pipeline de CI/CD/CT. Também adiciona uma camada de monitoramento de prompts sobre o pipeline de inferência do LLM Twin.

Apêndice, *Princípios de MLOps*, aborda os seis princípios de MLOps utilizados para construir aplicações de ML escaláveis, reprodutíveis e robustas.

Para tirar o máximo proveito deste livro

Para maximizar sua experiência de aprendizado, espera-se que você tenha, no mínimo, uma compreensão fundamental dos princípios e práticas de desenvolvimento de software. Familiaridade com programação em Python é particularmente benéfica, pois os exemplos e

trechos de código do livro são predominantemente em Python. Embora experiência a priori com conceitos de machine learning seja vantajosa, ela não é estritamente necessária, pois o livro fornece explicações para muitos conceitos fundamentais de IA e ML. No entanto, você deve estar à vontade com estruturas de dados básicas, algoritmos e ter alguma experiência trabalhando com APIs e serviços em nuvem.

Supõe-se familiaridade com sistemas de controle de versão como Git, pois este livro possui um repositório GitHub para exemplos de código. Embora este livro seja projetado para ser acessível a quem é iniciante em IA e LLMs, se você tiver alguma base nessas áreas, achará mais fácil compreender os conceitos e técnicas mais avançados que apresentamos.

Convenções utilizadas

Há diversas convenções de texto utilizadas ao longo deste livro.

`CodeInText`: Indica termos de código no texto, nomes de tabelas de banco de dados, nomes de pastas, nomes de arquivos, extensões de arquivo, caminhos, URLs de exemplo, entrada do usuário e nomes de usuário do Twitter. Por exemplo: "Na função `format_samples` aplicamos o modelo (template) de chat Alpaca a cada mensagem individual."

Um bloco de código é apresentado da seguinte forma:

```
def format_samples(example):
    example["prompt"] = alpaca_template.
format(example["prompt"])
    example["chosen"] = example['chosen'] + EOS_TOKEN
    example["rejected"] = example['rejected'] + EOS_TOKEN
    return {"prompt": example["prompt"], "chosen":
example["chosen"], "rejected": example["rejected"]}
```

Quando desejamos chamar sua atenção para uma parte específica de um bloco de código, as linhas ou itens relevantes são apresentados em negrito:

```
def format_samples(example):
    example["prompt"] = alpaca_template.
format(example["prompt"])
    example["chosen"] = example['chosen'] + EOS_TOKEN
    example["rejected"] = example['rejected'] + EOS_TOKEN
    return {"prompt": example["prompt"], "chosen":
example["chosen"], "rejected": example["rejected"]}
```

Qualquer entrada ou saída na linha de comando é escrita da seguinte forma:

```
poetry install --without aws
```

Negrito: Indica um novo termo, uma palavra importante ou palavras que você vê na tela. Por exemplo, palavras em menus ou caixas de diálogo aparecem no texto assim. Por exemplo:

"Para isso, vá para a guia **Configurações** na parte superior do repositório bifurcado no GitHub. No painel esquerdo, na seção **Segurança** , clique no botão de alternância **Segredos e variáveis** e, por fim, clique em **Actions**."



Avisos ou observações importantes são exibidos assim.



Dicas e truques são exibidos assim.

1

Entendendo o conceito e a arquitetura do LLM Twin

Ao final deste livro, teremos conduzido você na jornada de construção de um produto de **modelo de linguagem de grande porte** (**LLM**) completo. Acreditamos firmemente que a melhor maneira de aprender sobre LLMs e **aprendizado de máquina** (**ML**) em produção é pôr a mão na massa e construir sistemas. Este livro mostrará como construir um LLM Twin, um personagem de IA que aprende a escrever como uma pessoa específica, ao incorporar seu estilo, voz e personalidade em um LLM. Usando este exemplo, vamos conduzi-lo por todo o ciclo de vida de ML, da coleta de dados à implantação e ao monitoramento. A maioria dos conceitos aprendidos ao implementar seu LLM Twin pode ser aplicada em outras aplicações baseadas em LLM ou em ML.

Ao começar a implementar um novo produto, do ponto de vista de engenharia, há três etapas de planejamento pelas quais devemos passar antes de iniciar a construção. Primeiro, é fundamental entender o problema que estamos tentando resolver e o que queremos construir. No nosso caso, o que exatamente é um LLM Twin e por que construí-lo? Nesta etapa, devemos sonhar e focar no "Por quê". Em segundo lugar, para refletir um cenário do mundo real, projetaremos a primeira iteração de um produto com funcionalidade mínima. Aqui, devemos definir claramente as características essenciais necessárias para criar um produto funcional e valioso. As escolhas são feitas com base no cronograma, nos recursos e no conhecimento da equipe. É aqui que fazemos a ponte entre sonhar e focar no que é realista e, por fim, respondemos à seguinte pergunta: "O que vamos construir?"

Por fim, passaremos por uma etapa de design de sistema, detalhando a arquitetura principal e as escolhas de design usadas para construir o sistema de LLM. Observe que os dois primeiros componentes estão principalmente relacionados ao produto, enquanto o último é técnico e tem foco no "Como".

Essas três etapas são naturais na construção de um produto do mundo real. Mesmo que as duas primeiras não exijam muito conhecimento de ML, é fundamental percorrê-las para entender "como" construir o produto com uma visão clara. Em resumo, este Capítulo aborda os seguintes tópicos:

- Compreendendo o conceito de LLM Twin
- Planejando o MVP do produto LLM Twin
- Construindo sistemas de ML com pipelines de atributos/treinamento/inferência
- Projetando a arquitetura do sistema do LLM Twin

Ao final deste Capítulo, você terá uma visão clara do que aprenderá a construir ao longo do livro.

Compreendendo o conceito de LLM Twin

O primeiro passo é ter uma visão clara do que queremos criar e por que é valioso construí-lo. O conceito de um LLM Twin é novo. Assim, antes de entrar nos detalhes técnicos, é essencial entender o que ele é, o que devemos esperar dele e como ele deve funcionar. Ter uma intuição sólida do seu objetivo final torna muito mais fácil assimilar a teoria, o código e a infraestrutura apresentados neste livro.

O que é um LLM Twin?

Em poucas palavras, um LLM Twin é um personagem de IA que incorpora seu estilo de escrita, voz e personalidade em um LLM, que é um modelo de IA complexo. É uma versão digital de você mesmo *projetada* em um LLM. Em vez de um LLM genérico treinado em toda a internet, um LLM Twin é submetido a ajuste fino tendo você como base. Naturalmente, como um modelo de ML reflete os dados em que foi treinado, esse LLM incorporará seu estilo de escrita, voz e personalidade. Usamos intencionalmente a palavra "projetada". Como em qualquer outra projeção, você perde muita informação ao longo do caminho. Assim, esse LLM não *será você*; ele copiará o seu lado refletido nos dados em que foi treinado.

É essencial entender que um LLM reflete os dados em que foi treinado. Se você alimentá-lo com Shakespeare, ele começará a escrever como ele. Se você treiná-lo com Billie Eilish, ele começará a escrever músicas no estilo dela. Isso também é conhecido como transferência de estilo. Esse conceito também é bastante comum na geração de imagens. Por exemplo, digamos que você queira criar uma imagem de um gato usando o estilo de Van Gogh. Vamos alavancar a estratégia de transferência de estilo, mas, em vez de escolher uma personalidade, faremos isso em nossa própria persona.

Para ajustar o LLM a um determinado estilo e voz, juntamente com o ajuste fino, também vamos alavancar várias técnicas avançadas de **geração aumentada por recuperação** (**RAG**) para condicionar o processo autorregressivo com embeddings prévios de nós mesmos.

Exploraremos os detalhes no *Capítulo 5* sobre ajuste fino e *nos Capítulos 4* e *9* sobre RAG, mas, por ora, vamos ver alguns exemplos para entender intuitivamente o que afirmamos anteriormente.

Aqui estão alguns cenários em que você pode realizar o ajuste fino de um LLM para que ele se torne seu gêmeo:

- **Publicações no LinkedIn e threads no X**: Especialize o LLM para redigir conteúdo para redes sociais.
- **Mensagens com seus amigos e familiares**: Adapte o LLM à sua versão sem filtros.
- **Artigos e trabalhos acadêmicos**: Calibre o LLM para redigir conteúdo formal e educacional.
- **Código**: Especialize o LLM para implementar código como você faria.

Todos os cenários anteriores podem ser reduzidos a uma estratégia central: coletar seus dados digitais (ou algumas partes deles) e fornecê-los a um LLM usando diferentes algoritmos. No fim, o LLM reflete a voz e o estilo dos dados coletados. Fácil, certo?

Infelizmente, isso levanta muitas questões técnicas e morais. Primeiro, do ponto de vista técnico, como podemos acessar esses dados? Temos dados digitais suficientes para nos projetarmos em um LLM? Que tipo de dados seria valioso? Em segundo lugar, do ponto de vista moral, é aceitável fazer isso, para começo de conversa? Queremos criar um imitador de nós mesmos? Vai escrever usando nossa voz e personalidade, ou apenas tentará replicá-las?

Lembre-se de que o papel desta seção não é se preocupar com o "O quê" e o "Como", mas com o "Por quê". Vamos entender por que faz sentido ter o seu LLM Twin, por que isso pode ser valioso e por que é moralmente correto se enquadrarmos o problema corretamente.

Por que construir um LLM Twin é importante

Como um engenheiro (ou qualquer outra carreira profissional), construir uma marca pessoal é mais valioso do que um currículo tradicional. O maior problema ao criar uma marca pessoal é que escrever conteúdo em plataformas como LinkedIn, X ou Medium leva muito tempo. Mesmo que você goste de escrever e criar conteúdo, em algum momento ficará sem inspiração ou tempo e sentirá que precisa de ajuda. Não queremos transformar esta seção em um pitch, mas precisamos entender claramente o escopo deste produto/projeto.

Queremos construir um LLM Twin para escrever conteúdo personalizado no LinkedIn, X, Instagram, Substack e Medium (ou outros blogs) usando nosso estilo e voz. Ele não será usado em nenhuma situação imoral, mas atuará como seu copiloto de escrita. Com base no que ensinaremos a você neste livro, você pode ser criativo e adaptá-lo a vários casos de uso, mas vamos nos concentrar no nicho de geração de conteúdo para redes sociais e artigos. Assim, em vez de escrever o conteúdo do zero, podemos fornecer o esqueleto da nossa ideia principal ao LLM Twin e deixá-lo fazer o trabalho pesado.

Por fim, teremos que verificar se tudo está correto e formatá-lo ao nosso gosto (mais sobre as características concretas na seção *Planejando o MVP do produto LLM Twin*. Assim, projetamos um LLM Twin voltado à escrita de conteúdo que nos ajudará a automatizar nosso processo de escrita. Provavelmente ele fracassará se tentarmos usar este LLM específico em um cenário diferente, pois é aqui que vamos especializar o LLM por meio de ajuste fino, engenharia de prompts e RAG.

Então, por que construir um LLM Twin é importante? Ele ajuda você a fazer o seguinte:

- Crie sua marca
- Automatize o processo de escrita
- Faça um brainstorming de novas ideias criativas



Qual é a diferença entre um copiloto e um LLM Twin?

Um copiloto e gêmeo digital são dois conceitos diferentes que funcionam em conjunto e podem ser combinados em uma solução poderosa:

O copiloto é um assistente ou ferramenta de IA que aprimora as capacidades dos usuários humanos em diversas tarefas de programação, redação ou criação de conteúdo.

O gêmeo serve como uma representação digital 1:1 de uma entidade do mundo real, frequentemente usando IA para preencher a lacuna entre os mundos físico e digital. Por exemplo, um LLM Twin é um LLM que aprende a imitar sua voz, personalidade e estilo de escrita.

Com essas definições em mente, um assistente de IA para redação e criação de conteúdo que escreve como você é o seu copiloto LLM Twin.

Além disso, é fundamental entender que construir um LLM Twin é totalmente ético. O LLM será submetido a ajuste fino apenas com nossos dados digitais pessoais. Não coletaremos nem usaremos dados de outras pessoas para tentar assumir a identidade de qualquer pessoa. Temos um objetivo claro em mente: criar um imitador personalizado do nosso estilo de escrita. Cada pessoa terá seu próprio LLM Twin com acesso restrito.

Claro, há muitas preocupações de segurança envolvidas, mas não entraremos nisso aqui, pois isso, por si só, daria um livro.

Por que não usar o ChatGPT (ou outro chatbot semelhante)?



Esta subseção se referirá ao uso do ChatGPT (ou outro chatbot similar) apenas no contexto de geração de conteúdo personalizado.

Nós já demos a resposta. O ChatGPT não é *personalizado* ao seu estilo de escrita e à sua voz. Pelo contrário, ele é muito genérico, desarticulado e prolixo. Manter uma voz original é fundamental para o sucesso de longo prazo na construção da sua marca. Assim, usar diretamente o ChatGPT ou o Gemini não trará os melhores resultados. Mesmo que você esteja confortável em compartilhar conteúdo não personalizado, usar o ChatGPT de forma irrefletida pode resultar no seguinte:

- **Desinformação devido à alucinação**: Verificar manualmente os resultados em busca de alucinações ou usar ferramentas de terceiros para avaliar seus resultados é uma experiência tediosa e improdutiva.
- **Prompting manual tedioso**: Você deve elaborar manualmente seus prompts e injetar informações externas, o que é uma experiência cansativa. Além disso, as respostas geradas serão difíceis de reproduzir entre várias sessões, pois você não tem controle total sobre seus prompts e os dados injetados. Você pode resolver parte desse problema usando uma API e uma ferramenta como LangChain, mas precisa de experiência em programação para isso.

Com base em nossa experiência, se você quer conteúdo de alta qualidade que agregue valor real, você gastará mais tempo fazendo a depuração do texto gerado do que escrevendo-o você mesmo.

O cerne do LLM Twin consiste no seguinte:

- Quais dados coletamos
- Como pré-processamos os dados
- Como fornecemos os dados ao LLM
- Como encadeamos múltiplos prompts para obter os resultados desejados
- Como avaliamos o conteúdo gerado

O próprio LLM é importante, mas queremos destacar que usar a interface web do ChatGPT é extremamente trabalhoso para gerenciar e injetar diversas fontes de dados ou avaliar as saídas. A solução é construir um sistema de LLM que encapsule e automatize todas as etapas a seguir (replicá-las manualmente a cada vez não é uma solução viável a longo prazo):

- Coleta de dados
- Pré-processamento de dados
- Armazenamento de dados, versionamento e recuperação
- Ajuste fino de LLM
- RAG
- Avaliação da geração de conteúdo

Observe que nunca dissemos para não usar a API GPT da OpenAI, apenas que o framework de LLMs que apresentaremos é agnóstico a LLMs. Assim, se puder ser manipulado programaticamente e expuser uma interface de ajuste fino, poderá ser integrado ao sistema LLM Twin que aprenderemos a construir. A chave para a maioria dos produtos de ML bem-sucedidos é ser centrado em dados e tornar sua arquitetura agnóstica a modelos. Assim, você pode experimentar rapidamente vários modelos com seus dados específicos.

Planejamento do MVP do produto LLM Twin

Agora que entendemos o que é um LLM Twin e por que queremos construí-lo, devemos definir claramente as características do produto. Neste livro, vamos focar na primeira iteração, frequentemente chamada de **produto viável mínimo** (**MVP**), para seguir o ciclo natural da maioria dos produtos. Aqui, o principal objetivo é alinhar nossas ideias com objetivos de negócio realistas e exequíveis, usando os recursos disponíveis para produzir o produto. Mesmo como engenheiro, à medida que assume mais responsabilidades, você deve passar por essas etapas para reduzir a lacuna entre as necessidades do negócio e o que pode ser implementado.

O que é um MVP?

Um MVP é uma versão de um produto que inclui apenas características suficientes para atrair os primeiros usuários e testar a viabilidade do conceito do produto nas fases iniciais de desenvolvimento. Normalmente, o objetivo do MVP é obter insights do mercado com o mínimo de esforço.

Um MVP é uma estratégia poderosa pelos seguintes motivos:

- **Time-to-market acelerado**: Lance um produto rapidamente para ganhar tração inicial
- **Validação da ideia**: Teste-a com usuários reais antes de investir no desenvolvimento completo do produto
- **Pesquisa de mercado**: Obtenha insights sobre o que causa impacto no público-alvo
- **Minimização de riscos**: Reduz o tempo e os recursos necessários para um produto que pode não alcançar sucesso no mercado

Manter-se fiel ao *V* do MVP é essencial, o que significa que o produto deve ser *viável*. O produto deve proporcionar uma jornada do usuário de ponta a ponta, sem características implementadas pela metade, mesmo que o produto seja mínimo. Deve ser um produto funcional, com uma boa experiência do usuário, que as pessoas vão adorar e querer continuar usando para ver como ele evolui até atingir todo o seu potencial.

Definindo o MVP do LLM Twin

Como um experimento mental , vamos supor que, em vez de desenvolver este projeto para este livro, queremos criar um produto real. Nesse caso, quais são nossos recursos? Bem, infelizmente, não muitos:

- Somos uma equipe de três pessoas, com dois engenheiros de ML e um pesquisador de ML
- Nossos laptops
- Financiamento pessoal para computação, como treinamento de LLMs
- Nosso entusiasmo

Como você pode ver, não temos muitos recursos. Mesmo que isso seja apenas um experimento mental, reflete a realidade da maioria das startups no início de sua trajetória. Portanto, precisamos ser muito estratégicos ao definir nosso MVP do LLM Twin e quais características queremos escolher. Nosso objetivo é simples: queremos maximizar o valor do produto em relação ao esforço e aos recursos investidos nele.

Para simplificar, implementaremos as características que possam fazer o seguinte para o LLM Twin:

- Coletar dados dos seus perfis no LinkedIn, Medium, Substack e GitHub
- Realizar ajuste fino de um LLM de código aberto usando os dados coletados
- Preencher um banco de dados vetorial usando nossos dados digitais para RAG
- Criar publicações no LinkedIn aproveitando o seguinte:
  - Prompts do usuário
  - RAG para reutilizar e referenciar conteúdo antigo
  - Novos posts, artigos ou papers como conhecimento adicional para o LLM
- Ter uma interface web simples para interagir com o LLM Twin e permitir o seguinte:
  - Configure seus links de redes sociais e acione a etapa de coleta
  - Envie prompts ou links para recursos externos

Esse será o LLM Twin MVP. Mesmo que não pareça grande coisa, lembre-se de que devemos tornar este sistema econômico, escalável e modular.



Mesmo que nos concentremos apenas nas características centrais do LLM Twin definidas nesta seção, construiremos o produto tendo em mente a pesquisa mais recente em LLMs e as melhores práticas de engenharia de software e MLOps. Nosso objetivo é mostrar como projetar uma aplicação de LLM de forma econômica e escalável.

Até agora, examinamos o LLM Twin pelas perspectivas dos usuários e dos negócios. O último passo é examiná-lo sob a perspectiva da engenharia e definir um plano de desenvolvimento para entender como solucioná-lo tecnicamente. De agora em diante, o foco do livro será a implementação do LLM Twin.

Construindo sistemas de ML com pipelines de atributos/treinamento/inferência

Antes de entrarmos nos detalhes da arquitetura do LLM Twin, precisamos entender um padrão de sistema de ML no núcleo da arquitetura, conhecido como a arquitetura **atributo/treinamento/inferência** (**FTI**). Esta seção apresentará uma visão geral do design do pipeline FTI e de como ele pode estruturar uma aplicação de ML.

Vamos ver como podemos aplicar os pipelines FTI à arquitetura do LLM Twin.

O problema com o desenvolvimento de sistemas de ML

Construir sistemas de ML prontos para produção é muito mais do que apenas treinar um modelo. Do ponto de vista da engenharia, o treinamento do modelo é a etapa mais direta na maioria dos casos de uso. No entanto, o treinamento de um modelo se torna complexo quando se decide sobre a arquitetura e os hiperparâmetros corretos. Isso não é um problema de engenharia, mas sim um problema de pesquisa.

Neste ponto, queremos focar em como projetar uma arquitetura pronta para produção. Treinar um modelo com alta acurácia é extremamente valioso, mas, apenas ao treiná-lo em um conjunto de dados estático, você está longe de implantá-lo de forma robusta. Precisamos considerar como fazer o seguinte:

- Ingerir, limpar e validar dados novos
- Configurações de treinamento versus de inferência
- Computar e disponibilizar características no ambiente adequado
- Disponibilizar o modelo de forma econômica
- Versionar, rastrear e compartilhar os conjuntos de dados e os modelos
- Monitorar sua infraestrutura e seus modelos
- Implantar o modelo em uma infraestrutura escalável
- Automatizar as implantações e o treinamento

Esses são os tipos de problemas que um engenheiro de ML ou de MLOps deve considerar, enquanto a equipe de pesquisa ou de ciência de dados geralmente é responsável pelo treinamento do modelo.

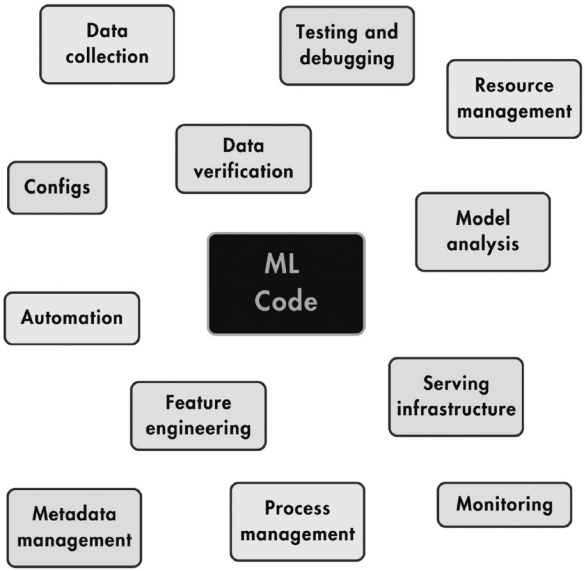


Figura 1.1: Componentes comuns de um sistema de ML

A figura anterior mostra todos os componentes que a equipe do Google Cloud sugere serem necessários para um sistema maduro de ML e MLOps. Além do código de ML, há muitos elementos variáveis. O restante do sistema é composto por configuração, automação, coleta de dados, verificação de dados, teste e depuração, gerenciamento de recursos, análise de modelo, gerenciamento de processos e metadados, infraestrutura de serving e monitoramento. A questão é que há muitos componentes que devemos considerar ao colocar um modelo de ML em produção.

Assim, a questão crítica é: Como conectamos todos esses componentes em um único sistema homogêneo? Precisamos criar um padrão para projetar sistemas de ML de forma clara a fim de responder a essa pergunta.

Soluções semelhantes existem para o software clássico. Por exemplo, em uma visão mais ampla, a maioria das aplicações de software pode ser dividida entre um DB, a lógica de negócios e a camada de UI. Cada camada pode ter a complexidade necessária, mas, em uma visão geral de alto nível, a arquitetura do software padrão pode ser reduzida aos três componentes anteriores.

Temos algo semelhante para aplicações de ML? O primeiro passo é examinar soluções anteriores e por que elas são inadequadas para construir sistemas de ML escaláveis.

O problema com as soluções anteriores

Na *Figura 1.2*, você pode observar a arquitetura típica presente na maioria das aplicações de ML. Ela é baseada em uma arquitetura monolítica em lote que acopla a criação de características, o treinamento de modelo e a inferência no mesmo componente. Ao adotar essa abordagem, você resolve rapidamente um problema crítico no mundo de ML: a assimetria entre treinamento e inferência. A assimetria entre treinamento e inferência ocorre quando as características passadas ao modelo são calculadas de maneira diferente nos momentos de treinamento e de inferência.

Nesta arquitetura, as características são criadas usando o mesmo código. Portanto, o problema de assimetria entre treinamento e serviço é resolvido por padrão. Esse padrão funciona bem ao trabalhar com poucos dados. O pipeline é executado de forma agendada em lote, e as predições são consumidas por uma aplicação de terceiros, como um dashboard.

Infelizmente, construir um sistema de processamento em lote monolítico gera muitas outras questões, como as seguintes:

- As características não são reutilizáveis (pelo seu sistema ou por outros)
- Se os dados aumentarem, você terá que refatorar todo o código para oferecer suporte a PySpark ou Ray
- É difícil reescrever o módulo de predição em uma linguagem mais eficiente, como C++, Java ou Rust
- É difícil compartilhar o trabalho entre várias equipes nos módulos de características, treinamento e predição
- É impossível migrar para uma tecnologia de streaming para treinamento em tempo real

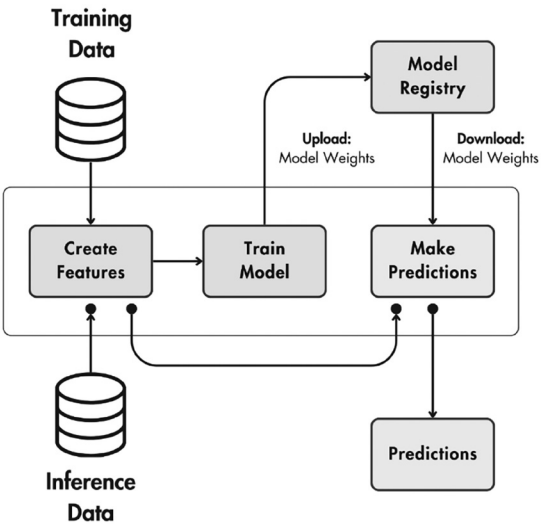


Figura 1.2: Arquitetura monolítica de pipeline em lote

Na *Figura 1.3*, podemos ver um cenário semelhante para um sistema em tempo real. Este caso de uso introduz outro problema além do que listamos anteriormente. Para fazer as predições, precisamos transferir todo o estado por meio da requisição do cliente para que as características possam ser computadas e passadas ao modelo.

Considere o cenário de calcular recomendações de filmes para um usuário. Em vez de simplesmente passar o ID do usuário, devemos transmitir todo o estado do usuário, incluindo seu nome, idade, gênero, histórico de filmes e mais. Essa abordagem é repleta de potenciais erros, pois o cliente precisa entender como acessar esse estado, e ela é fortemente acoplada ao serviço do modelo.

Outro exemplo seria ao implementar um LLM com suporte a RAG. Os documentos que adicionamos como contexto junto com a consulta representam nosso estado externo. Se não armazenássemos os registros em um banco de dados vetorial, nós teríamos que passá-los junto com a consulta do usuário. Para isso, o cliente deve saber como consultar e recuperar os documentos, o que não é viável. É um antipadrão que a aplicação cliente saiba como acessar ou computar as características. Se você não entende como a RAG funciona, explicaremos em detalhes nos *Capítulos 8* e *9*.

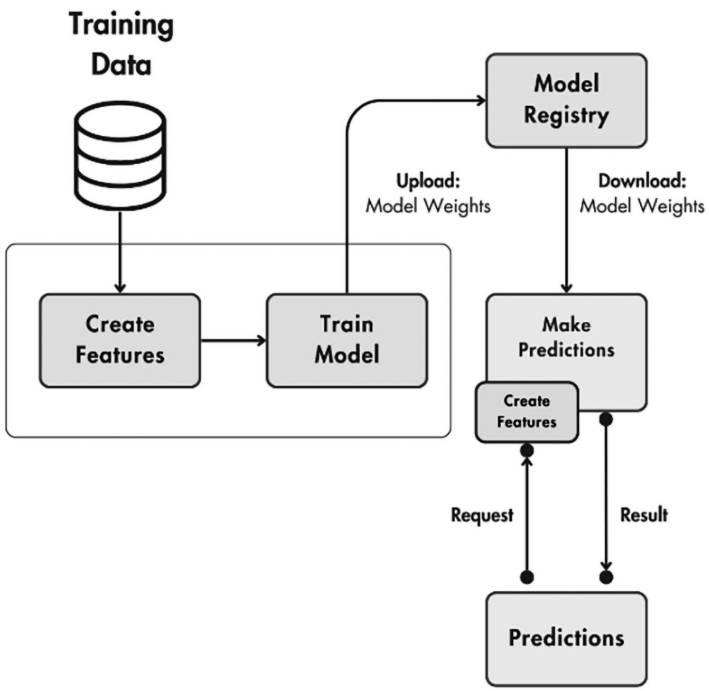


Figura 1.3: Arquitetura sem estado em tempo real

Em conclusão, nosso problema é acessar as características para fazer predições sem passá-las na requisição do cliente. Por exemplo, com base em nosso primeiro exemplo de recomendação

de filmes para um usuário, como podemos prever as recomendações apenas com base no ID do usuário? Lembre-se dessas perguntas, pois iremos respondê-las em breve.

Por fim, no outro extremo, o Google Cloud fornece uma arquitetura pronta para produção, como mostrado na *Figura 1.4*. Infelizmente, mesmo que seja uma solução viável, é muito complexa e pouco intuitiva. Você terá dificuldade em entendê-la se não tiver muita experiência em implantar e manter modelos de aprendizado de máquina em produção. Além disso, não é simples entender como começar pequeno e expandir o sistema ao longo do tempo.

A imagem a seguir é reproduzida a partir de trabalho criado e compartilhado pelo Google e utilizada de acordo com os termos descritos na Licença Creative Commons 4.0 de Atribuição:

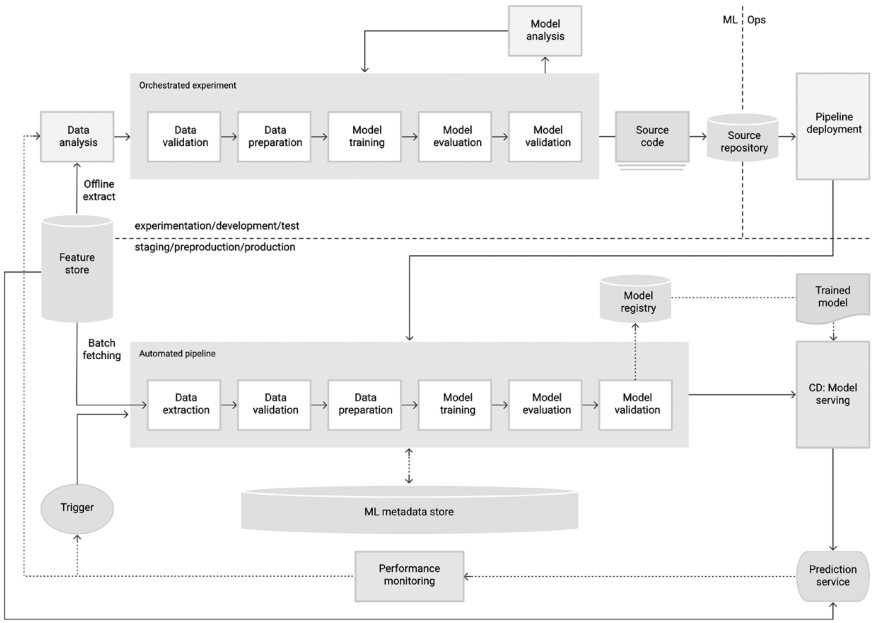


Figura 1.4: Automação de pipeline de ML para treinamento contínuo (fonte: https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning)

Mas é aqui que as arquiteturas de pipeline FTI entram em ação. A seção a seguir mostrará como resolver esses problemas fundamentais usando um design de ML intuitivo.

A solução – pipelines de ML para sistemas de aprendizado de máquina

A solução baseia-se na criação de um mapa mental claro e simples que qualquer equipe ou pessoa pode seguir para calcular as características, treinar o modelo e fazer predições. Com base nesses três passos críticos de que qualquer sistema de ML necessita, o padrão é conhecido como o pipeline FTI. Então, como isso difere do que apresentamos antes?

O padrão sugere que qualquer sistema de ML pode ser reduzido a estes três pipelines: de atributos, de treinamento e de inferência (semelhante às camadas de DB, lógica de negócios e UI do software clássico). Isso é poderoso, pois podemos definir claramente o escopo e a interface de cada pipeline. Além disso, é mais fácil entender como os três componentes interagem. Em última análise, temos apenas três partes móveis em vez de 20, como sugerido na *Figura 1.4*, o que é muito mais fácil de trabalhar e definir.

Como mostrado na *Figura 1.5*, temos os pipelines de atributos, de treinamento e de inferência. Vamos ampliar o foco em cada um deles e entender seu escopo e sua interface.

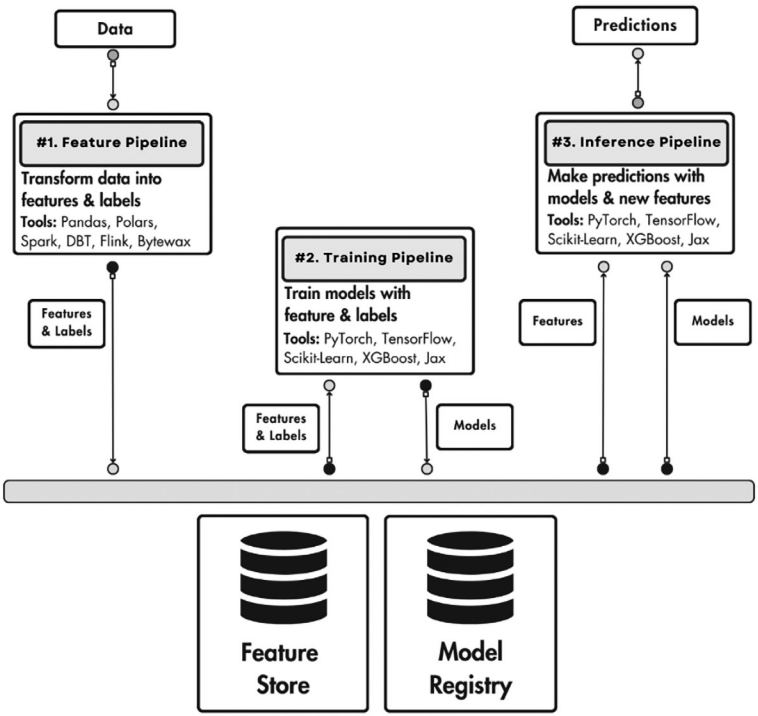


Figura 1.5: Arquitetura dos pipelines de FTI

Antes de entrar nos detalhes, é essencial entender que cada pipeline é um componente diferente que pode ser executado em um processo ou em um hardware diferente. Assim, cada pipeline pode ser escrito usando uma tecnologia diferente, por uma equipe diferente, ou escalado de forma distinta. A ideia-chave é que o design é muito flexível às necessidades da sua equipe. Ele funciona como um mapa mental para estruturar sua arquitetura.

O pipeline de características

O pipeline de características recebe dados brutos como entrada, processa-os e produz as características e os rótulos exigidos pelo modelo para treinamento ou inferência. Em vez

de passá-los diretamente para o modelo, as características e os rótulos são armazenados em um feature store. Sua responsabilidade é armazenar, versionar, rastrear e compartilhar as características. Ao salvar as características em um feature store, sempre temos um estado das nossas características. Assim, podemos enviar facilmente as características para os pipelines de treinamento e de inferência.

Como os dados são versionados, podemos sempre garantir que as características em tempo de treinamento e de inferência correspondam. Assim, evitamos o problema de assimetria entre treinamento e serviço.

O pipeline de treinamento

O pipeline de treinamento recebe as características e os rótulos a partir das características armazenadas como entrada e produz um modelo treinado ou modelos. Os modelos são armazenados em um registro de modelos. Seu papel é semelhante ao de feature stores, mas, desta vez, o modelo é o cidadão de primeira classe. Assim, o registro de modelos armazenará, versionará, rastreará e compartilhará o modelo com o pipeline de inferência.

Além disso, a maioria dos registros de modelos modernos oferece suporte a um repositório de metadados que permite especificar aspectos essenciais de como o modelo foi treinado. Os mais importantes são as características, os rótulos e sua versão usada para treinar o modelo. Assim, sempre saberemos em quais dados o modelo foi treinado.

O pipeline de inferência

O pipeline de inferência recebe como entrada as características e os rótulos do repositório de recursos e o modelo treinado do registro de modelos. Com esses dois, as predições podem ser facilmente feitas em lote ou em tempo real.

Como este é um padrão versátil, cabe a você decidir o que fazer com suas predições. Se for um sistema em lote, elas provavelmente serão armazenadas em um DB. Se for um sistema em tempo real, as predições serão servidas ao cliente que as solicitou. Além disso, as características, os rótulos e os modelos são versionados. Podemos atualizar ou reverter facilmente a implantação do modelo. Por exemplo, sempre saberemos que o modelo v1 usa as características F1, F2 e F3, que o modelo v2 usa F2, F3 e F4. Assim, podemos alterar rapidamente as conexões entre o modelo e as características.

Benefícios da arquitetura FTI

Para concluir, o mais importante de que você deve se lembrar sobre os pipelines do FTI é a interface deles:

- O pipeline de características recebe dados e gera as características e os rótulos, que são salvos no repositório de características.

- O pipeline de treinamento consulta o repositório de características em busca de características e rótulos e produz um modelo para o registro de modelos.
- O pipeline de inferência usa as características do repositório de características e o modelo do registro de modelos para fazer predições.

Não importa o quão complexo seu sistema de ML se torne, essas interfaces permanecerão as mesmas.

Agora que entendemos melhor como o padrão funciona, queremos destacar os principais benefícios de usar esse padrão:

- Como há apenas três componentes, é intuitivo de usar e fácil de entender.
- Cada componente pode ser implementado em sua própria stack tecnológica, de modo que possamos adaptá-los rapidamente a necessidades específicas, como dados em grande volume ou em streaming. Além disso, isso nos permite escolher as melhores ferramentas para o trabalho.
- Como há uma interface transparente entre os três componentes, cada um pode ser desenvolvido por uma equipe diferente (se necessário), tornando o desenvolvimento mais gerenciável e escalável.
- Cada componente pode ser implantado, escalado e monitorado de forma independente.

A última coisa que você deve entender sobre o padrão FTI é que o sistema não precisa conter apenas três pipelines. Na maioria dos casos, ele incluirá mais. Por exemplo, o pipeline de características pode ser composto por um serviço que calcula as características e outro que valida os dados. Além disso, o pipeline de treinamento pode ser composto pelos componentes de treinamento e de avaliação.

Os pipelines FTI atuam como camadas lógicas. Assim, é perfeitamente aceitável que cada um seja complexo e contenha múltiplos serviços. No entanto, o essencial é manter a mesma interface quanto à forma como os pipelines FTI interagem entre si por meio do feature store e dos registros de modelos. Ao fazer isso, cada componente FTI pode evoluir de maneira diferente, sem precisar conhecer os detalhes dos demais e sem quebrar o sistema com novas alterações.



Para saber mais sobre o padrão de pipeline FTI, considere ler *De MLOps a Sistemas de ML com Pipelines de Atributos/Treinamento/Inferência* por Jim Dowling, CEO e cofundador da Hopsworks: `https://www.hopsworks.ai/post/mlops-to-ml-systems-with-fti-pipelines`. O artigo dele inspirou esta seção.

Agora que entendemos a arquitetura do pipeline FTI, a etapa final deste Capítulo é ver como ela pode ser aplicada ao caso de uso do LLM Twin.

Projetando a arquitetura do sistema do LLM Twin

Nesta seção, listaremos os detalhes técnicos concretos da aplicação LLM Twin e entender como podemos resolvê-los ao projetar nosso sistema de LLM usando a arquitetura FTI. No entanto, antes de mergulhar nos pipelines, queremos destacar que não focaremos no ferramental ou na stack tecnológica nesta etapa. Queremos apenas definir uma arquitetura de alto nível do sistema, que seja agnóstica de linguagem, framework, plataforma e infraestrutura neste ponto. Focaremos no escopo, na interface e na interconectividade de cada componente. Em capítulos futuros, abordaremos os detalhes de implementação e a stack tecnológica.

Listando os detalhes técnicos da arquitetura do LLM Twin

Até agora, definimos o que o LLM Twin deve suportar do ponto de vista do usuário. Agora, vamos esclarecer os requisitos do sistema de ML sob uma perspectiva puramente técnica:

- Do lado dos dados, precisamos fazer o seguinte:
  - Colete dados do LinkedIn, Medium, Substack e GitHub de forma totalmente autônoma e programada
  - Padronize os dados rastreados e armazene-os em um data warehouse
  - Limpe os dados brutos
  - Crie conjuntos de dados de instruções para ajuste fino de um LLM
  - Segmente e incorpore os dados limpos. Armazene os dados vetorizados em um banco de dados vetorial para RAG.
- Para o treinamento, precisamos fazer o seguinte:
  - Realize ajuste fino de LLMs de vários tamanhos (parâmetros 7B, 14B, 30B ou 70B)
  - Realize ajuste fino com conjuntos de dados de instruções de vários tamanhos
  - Alterne entre tipos de LLMs (por exemplo, entre Mistral, Llama e GPT)
  - Acompanhe e compare experimentos
  - Teste potenciais LLMs candidatos para produção antes de implantá-los
  - Inicie automaticamente o treinamento quando novos conjuntos de dados de instruções estiverem disponíveis.
- O código de inferência terá as seguintes propriedades:
  - Uma interface de API REST para que clientes interajam com o LLM Twin
  - Acesso ao banco de dados vetorial em tempo real para RAG

  - Inferência com LLMs de vários tamanhos
  - Escalonamento automático com base nas requisições dos usuários
  - Implante automaticamente os LLMs que passarem na etapa de avaliação.
- O sistema irá oferecer suporte às seguintes características de LLMOps:
  - Versionamento, linhagem e reutilização de conjuntos de dados de instruções
  - Versionamento, linhagem e reutilização de modelos
  - Rastreamento de experimentos
  - **Treinamento contínuo**, **integração contínua**, e **entrega contínua** (CT/**CI**/**CD**)
  - Monitoramento de prompts e de sistemas



Se algum requisito técnico não fizer sentido agora, tenha paciência conosco. Para evitar repetição, examinaremos os detalhes no respectivo capítulo.

A lista anterior é bastante abrangente. Poderíamos tê-la detalhado ainda mais, mas, neste ponto, queremos focar na funcionalidade principal. Ao implementar cada componente, examinaremos todos os detalhes. Mas, por ora, a pergunta fundamental que devemos nos fazer é: Como podemos aplicar o design do pipeline de FTI para implementar a lista anterior de requisitos?

Como projetar a arquitetura do LLM Twin usando o design do pipeline de FTI

Vamos dividir o sistema em quatro componentes centrais. Você se perguntará o seguinte: "Quatro? Por que não três, como o projeto do pipeline de FTI afirma claramente?" Ótima pergunta. Felizmente, a resposta é simples. Também devemos implementar o pipeline de dados, além dos três pipelines de atributo/treinamento/inferência. De acordo com as melhores práticas:

- A equipe de engenharia de dados é responsável pelo pipeline de dados
- A equipe de engenharia de ML é responsável pelos pipelines de FTI.
- Dado nosso objetivo de construir um MVP com uma equipe pequena, precisamos implementar toda a aplicação. Isso inclui definir a coleta de dados e os pipelines de FTI. Abordar um problema de ponta a ponta é comum em startups que não podem arcar com equipes dedicadas. Assim, os engenheiros precisam assumir múltiplos papéis, dependendo do estado do produto. Ainda assim, em qualquer cenário, saber

como um sistema de ML de ponta a ponta funciona é valioso para compreender melhor o trabalho de outras pessoas.

A *Figura 1.6* mostra a arquitetura do sistema de LLM. A melhor maneira de entendê-la é analisar os quatro componentes individualmente e explicar como funcionam.

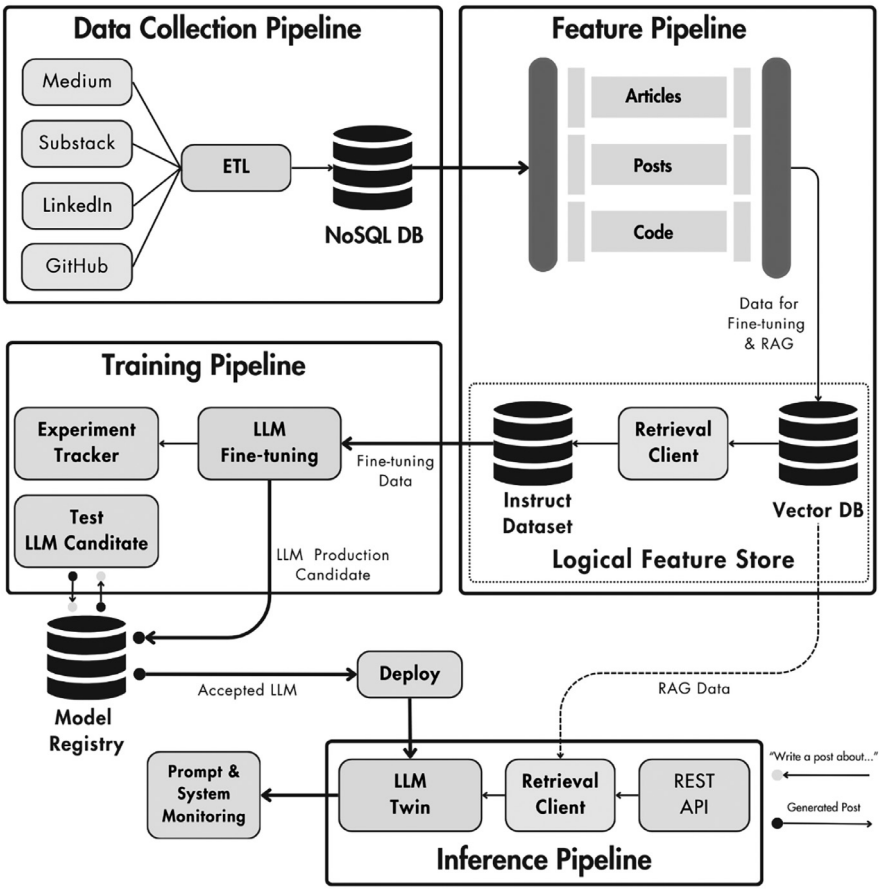


Figura 1.6: Arquitetura de alto nível do LLM Twin

Pipeline de coleta de dados

O pipeline de coleta de dados pipeline envolve rastrear seus dados pessoais a partir do Medium, Substack, LinkedIn e GitHub. Como pipeline de dados, usaremos o **extrair, carregar, transformar** (**padrão ETL**) padrão para extrair dados de plataformas de redes sociais, padronizá-los e carregá-los em um data warehouse.



É fundamental destacar que o pipeline de coleta de dados foi projetado para rastrear dados apenas da sua plataforma de mídia social. Ele não terá acesso aos dados de outras pessoas. Como exemplo para este livro, concordamos em disponibilizar nossos dados coletados para fins de aprendizado. Caso contrário, usar os dados de outras pessoas sem o consentimento delas não é ético.

A saída deste componente será um banco de dados NoSQL, que atuará como nosso data warehouse. Como trabalhamos com dados de texto, que são naturalmente não estruturados, um banco de dados NoSQL se encaixa perfeitamente.

Embora um banco de dados NoSQL, como o MongoDB, não seja classificado como um data warehouse, do nosso ponto de vista, ele atuará como tal. Por quê? Porque ele armazena dados brutos padronizados coletados por vários pipelines de ETL que estão prontos para serem ingeridos em um sistema de ML.

Os dados digitais coletados são agrupados em três categorias:

- Artigos (Medium, Substack)
- Publicações (LinkedIn)
- Código (GitHub)

Queremos abstrair a plataforma onde os dados foram rastreados. Por exemplo, ao alimentar o LLM com um artigo, saber que ele veio do Medium ou do Substack não é essencial. Podemos manter a URL de origem como metadados para fornecer referências. No entanto, dos pontos de vista de processamento, ajuste fino e RAG, é vital saber que tipo de dados ingerimos, pois cada categoria deve ser processada de forma diferente. Por exemplo, a estratégia de fragmentação entre um post, um artigo e um trecho de código será diferente.

Além disso, ao agrupar os dados por categoria, e não pela origem, podemos rapidamente integrar dados de outras plataformas, como X aos posts ou GitLab à coleção de código. Como um sistema modular, devemos acoplar um ETL adicional no pipeline de coleta de dados, e todo o restante funcionará sem novas modificações de código.

Pipeline de características

O papel do pipeline de características é extrair artigos, posts e pontos de dados de código brutos do data warehouse, processá-los e carregá-los no feature store.

As características do padrão FTI já estão presentes.

Aqui estão algumas propriedades personalizadas do pipeline de características do LLM Twin:

- Ele processa três tipos de dados de forma diferente: artigos, posts e código
- Ele contém três etapas principais de processamento necessárias para ajuste fino e RAG: limpeza, fragmentação e embedding

- Ele cria dois snapshots dos dados digitais, um após a limpeza (usado para ajuste fino) e um após o incorporação (usado para RAG)
- Ele usa um repositório lógico de atributos em vez de um repositório de atributos especializado

Vamos detalhar um pouco mais a parte do feature store lógico. Como em qualquer sistema baseado em RAG, uma das peças centrais da infraestrutura é um banco de dados vetorial. Em vez de integrar outro banco de dados, mais especificamente, um feature store especializado, usamos o banco de dados vetorial, além de alguma lógica adicional para verificar todas as propriedades de um feature store de que nosso sistema precisa.

O banco de dados vetorial não oferece o conceito de conjunto de dados de treinamento, mas pode ser usado como um banco de dados NoSQL. Isso significa que podemos acessar pontos de dados usando seu ID e o nome da coleção. Assim, podemos consultar facilmente o banco de dados vetorial em busca de novos pontos de dados, sem qualquer lógica de busca vetorial. Por fim, encapsularemos os dados recuperados em um artefato versionado, rastreado e compartilhável—mais sobre artefatos no *Capítulo 2*. Por ora, você deve saber que é um conceito de MLOps usado para encapsular dados e enriquecê-los com as propriedades listadas antes.

Como o restante do sistema acessará o feature store lógico? O pipeline de treinamento usará os conjuntos de dados de instrução como artefatos, e o pipeline de inferência consultará o banco de dados vetorial para obter contexto adicional usando técnicas de busca vetorial.

Para o nosso caso de uso, isso é mais do que suficiente pelos seguintes motivos:

- Os artefatos funcionam muito bem para casos de uso offline como treinamento
- O banco de dados vetorial é projetado para acesso online, do qual precisamos para inferência.

Nos próximos capítulos, porém, explicaremos como as três categorias de dados (artigos, posts e código) são limpas, fragmentadas e vetorizadas.

Para concluir, recebemos pontos de dados brutos de artigos, posts ou código, processamos esses pontos e os armazenamos em um repositório de atributos para torná-los acessíveis aos pipelines de treinamento e inferência. Observe que eliminar toda a complexidade e focar apenas na interface se encaixa perfeitamente no padrão FTI. Bonito, não é?

Pipeline de treinamento

O pipeline de treinamento consome conjuntos de dados de instrução do feature store, realiza ajuste fino em um LLM com ele, e armazena os pesos do LLM ajustado em um registro de modelos. Mais concretamente, quando um novo conjunto de dados de instrução estiver disponível no feature store lógico, acionaremos o pipeline de treinamento, consumiremos o artefato e realizaremos o ajuste fino do LLM.

Nas fases iniciais, a equipe de ciência de dados é responsável por esta etapa. Eles executam vários experimentos para encontrar o melhor modelo e os melhores hiperparâmetros para a tarefa, seja por meio de ajuste de hiperparâmetros automático ou manualmente. Para comparar e escolher o melhor conjunto de hiperparâmetros, usaremos um rastreador de experimentos para registrar em log tudo que tenha valor e compará-lo entre os experimentos. Por fim, eles escolherão os melhores hiperparâmetros e o LLM ajustado e irão propô-lo como o candidato a LLM de produção. O LLM proposto é então armazenado no registro de modelos. Após o término da fase de experimentação, armazenamos e reutilizamos os melhores hiperparâmetros encontrados para eliminar as restrições manuais do processo. Agora, podemos automatizar completamente o processo de treinamento, conhecido como treinamento contínuo.

O pipeline de teste é acionado para uma análise mais detalhada do que durante o ajuste fino. Antes de implantar o novo modelo em produção, avaliá-lo com um conjunto mais rigoroso de testes é fundamental para verificar que o candidato mais recente é melhor do que o que está atualmente em produção. Se essa etapa for aprovada, o modelo é finalmente marcado como aceito e implantado no pipeline de inferência de produção. Mesmo em um sistema de ML totalmente automatizado, recomenda-se ter uma etapa manual antes de aceitar um novo modelo de produção. É como apertar o botão vermelho antes de uma ação significativa com altas consequências. Assim, nesta etapa, um especialista analisa um relatório gerado pelo componente de teste. Se tudo parecer bom, ele aprova o modelo e a automação pode continuar.

As particularidades deste componente estarão nos aspectos de LLM, tais como os seguintes:

- Como você implementa um pipeline agnóstico a LLMs?
- Quais técnicas de ajuste fino deve usar?
- Como você escala o algoritmo de ajuste fino para LLMs e conjuntos de dados de vários tamanhos?
- Como você escolhe um LLM candidato para produção a partir de vários experimentos?
- Como você testa o LLM para decidir se o coloca em produção ou não?

Ao final deste livro, você saberá como responder a todas essas perguntas.

Um último aspecto que queremos esclarecer é **CT**. Nosso design modular permite a alavancagem rápida de um orquestrador de ML para agendar e acionar diferentes partes do sistema. Por exemplo, podemos agendar o pipeline de coleta de dados para rastrear dados toda semana.

Então, podemos acionar o pipeline de características quando novos dados estiverem disponíveis no data warehouse e o pipeline de treinamento quando novos conjuntos de dados de instruções estiverem disponíveis.

Pipeline de inferência

O pipeline de inferência é a última peça do quebra-cabeça. Ele está conectado ao registro de modelos e ao feature store lógico. Ele carrega um LLM ajustado por ajuste fino do registro

de modelos e, a partir do feature store lógico, acessa o banco de dados vetorial para RAG. Ele recebe requisições de clientes por meio de uma REST API na forma de consultas. Ele utiliza o LLM ajustado por ajuste fino e o acesso ao banco de dados vetorial para realizar RAG e responder às consultas.

Todas as consultas dos clientes, prompts enriquecidos usando RAG e respostas geradas são enviadas a um sistema de monitoramento de prompts para analisar, depurar e entender melhor o sistema. Com base em requisitos específicos, o sistema de monitoramento pode acionar alarmes para tomar ações manual ou automaticamente.

No nível de interface, este componente segue exatamente a arquitetura FTI, mas, ao dar zoom, podemos observar características exclusivas de um sistema de LLM e RAG, tais como as seguintes:

- Um cliente de recuperação usado para fazer buscas vetoriais para RAG
- Modelos de prompt usados para mapear consultas do usuário e informação externa para entradas de LLM
- Ferramentas especiais para monitoramento de prompts

Considerações finais sobre o design do FTI e a arquitetura do LLM Twin

Não precisamos ser extremamente rígidos em relação ao padrão FTI. É uma ferramenta usada para esclarecer como projetar sistemas de ML. Por exemplo, em vez de usar um feature store dedicado só porque é assim que se faz, no nosso sistema é mais fácil e barato usar um feature store lógico baseado em um banco de dados vetorial e artefatos. O importante era focar nas propriedades necessárias que um feature store oferece, como um conjunto de dados de treinamento versionado e reutilizável.

Por fim, explicaremos brevemente os requisitos de computação de cada componente. A coleta de dados e o pipeline de características são majoritariamente baseados em CPU e não exigem máquinas potentes. O pipeline de treinamento requer máquinas potentes com GPUs que possam carregar um LLM e realizar ajuste fino. O pipeline de inferência fica em um meio-termo. Ele ainda precisa de uma máquina potente, mas exige menos recursos de processamento do que a etapa de treinamento. No entanto, ele deve ser testado com cuidado, pois o pipeline de inferência faz interface diretamente com o usuário. Assim, queremos que a latência esteja dentro dos parâmetros exigidos para uma boa experiência do usuário. Ainda assim, usar o padrão FTI não é um problema. Podemos escolher os requisitos de computação adequados para cada componente.

Além disso, cada pipeline será dimensionado de forma diferente. Os pipelines de dados e de características serão dimensionados horizontalmente com base na carga de CPU e RAM. O pipeline de treinamento será dimensionado verticalmente, adicionando mais GPUs. O pipeline de inferência será dimensionado horizontalmente com base no número de solicitações dos clientes.

Para concluir, a a arquitetura de LLM apresentada atende a todos os requisitos técnicos listados no início da seção. Ela processa os dados conforme solicitado, e o treinamento é modular e pode ser rapidamente adaptado a diferentes LLMs, conjuntos de dados ou técnicas de ajuste fino. O pipeline de inferência oferece suporte a RAG e é exposto como uma API REST. No lado de LLMOps, o sistema suporta versionamento de conjuntos de dados e de modelos, linhagem e reutilização. O sistema possui um serviço de monitoramento, e toda a arquitetura de ML é projetada com CT/CI/CD em mente.

Isto conclui a visão geral de alto nível da arquitetura do LLM Twin.

Resumo

Este primeiro capítulo foi fundamental para entender o objetivo do livro. Como um livro orientado a produto que o conduzirá pela construção de um sistema de ML de ponta a ponta, era essencial compreender inicialmente o conceito de um LLM Twin. Em seguida, apresentamos o que é um MVP e como planejar nosso MVP do LLM Twin com base em nossos recursos disponíveis. Depois disso, traduzimos nosso conceito em uma solução técnica prática com requisitos específicos. Nesse contexto, introduzimos o padrão de projeto FTI e demonstramos sua aplicação no mundo real ao projetar sistemas que são ao mesmo tempo modulares e escaláveis. Por fim, aplicamos com sucesso o padrão FTI para projetar a arquitetura do LLM Twin de modo a atender a todos os nossos requisitos técnicos.

Ter uma visão clara do panorama geral é essencial ao construir sistemas. Compreender como um único componente será integrado ao restante da aplicação pode ser muito valioso ao trabalhar nele. Começamos com uma apresentação mais abstrata da arquitetura do LLM Twin, focando no escopo, na interface e na interconectividade de cada componente.

Os capítulos seguintes explorarão como implementar e implantar cada componente. No que diz respeito a MLOps, vamos mostrar como usar uma plataforma de computação, um orquestrador, um registro de modelos, artefatos e outras ferramentas e conceitos para dar suporte a todas as boas práticas de MLOps.

Referências

- Dowling, J. (2024a, 11 de julho). *From MLOps to ML Systems with Feature/Training/Inference Pipelines*. *Hopsworks*. `https://www.hopsworks.ai/post/mlops-to-ml-systems-with-fti-pipelines`
- Dowling, J. (2024b, 5 de agosto). *Modularity and Composability for AI Systems with AI Pipelines and Shared Storage*. *Hopsworks*. `https://www.hopsworks.ai/post/modularity-and-composability-for-ai-systems-with-ai-pipelines-and-shared-storage`
- Joseph, M. (2024, 23 de agosto). *The Taxonomy for Data Transformations in AI Systems*. *Hopsworks*. `https://www.hopsworks.ai/post/a-taxonomy-for-data-transformations-in-ai-systems`

- *MLOps: Continuous delivery and automation pipelines in machine learning*. (2024, 28 de agosto). Google Cloud. https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning
- Qwak. (2024a, 2 de junho). *CI/CD for Machine Learning in 2024: Best Practices to build, test, and Deploy* | Infer. *Medium*. https://medium.com/infer-qwak/ci-cd-for-machine-learning-in-2024-best-practices-to-build-test-and-deploy-c4ad869824d2
- Qwak. (2024b, 23 de julho). *5 Best Open Source Tools to build End-to-End MLOPs Pipeline* in 2024. *Medium*. https://medium.com/infer-qwak/building-an-end-to-end-mlops-pipeline-with-open-source-tools-d8bacbf4184f
- Salama, K., Kazmierczak, J., & Schut, D. (2021). *Practitioners guide to MLOPs: A framework for continuous delivery and automation of machine learning* (1ª ed.) [pdf]. Google Cloud. https://services.google.com/fh/files/misc/practitioners_guide_to_mlops_whitepaper.pdf

2

Ferramentas e instalação

Este capítulo apresenta todas as ferramentas essenciais que serão usadas ao longo do livro, especialmente para implementar e implantar o projeto de LLM Twin. Neste ponto do livro, não planejamos apresentar conceitos aprofundados de LLM, RAG, MLOps ou LLMOps. Vamos guiá-lo rapidamente pela nossa stack de tecnologia e pelos pré-requisitos, para evitar nos repetirmos ao longo do livro sobre como configurar uma ferramenta específica e por que a escolhemos. A partir do *Capítulo 3*, começaremos a explorar nosso caso de uso de LLM Twin implementando um ETL de coleta de dados que rastreia dados na internet.

Na primeira parte do capítulo, apresentaremos as ferramentas no ecossistema Python para gerenciar várias versões do Python, criar um ambiente virtual e instalar as dependências com versões fixadas necessárias para que nosso projeto seja executado. Junto com a apresentação dessas ferramentas, também mostraremos como instalar o repositório `LLM-Engineers-Handbook` na sua máquina local (caso você queira testar o código por conta própria): `https://github.com/PacktPublishing/LLM-Engineers-Handbook`.

Em seguida, exploraremos todas as ferramentas de MLOps e LLMOps que usaremos, começando com ferramentas mais genéricas, como um registro de modelos, e passando para ferramentas mais voltadas para LLM, como ferramentas de avaliação de LLM e monitoramento de prompts. Também entenderemos como gerenciar um projeto com vários pipelines de ML usando o ZenML, um orquestrador que preenche a lacuna entre ML e MLOps. Além disso, exploraremos rapidamente quais bancos de dados usaremos para armazenamento NoSQL e vetorial. Mostraremos como executar todos esses componentes em sua máquina local usando o Docker. Por fim, revisaremos rapidamente a AWS e mostraremos como criar um usuário e chaves de acesso da AWS, além de instalar e configurar a AWS CLI para manipular seus recursos na nuvem programaticamente. Também exploraremos o SageMaker e por que o usamos para treinar e implantar nossos LLMs de código aberto.

Se você já está familiarizado com essas ferramentas, pode pular este capítulo sem problemas. Também explicamos como instalar o projeto e configurar todos os componentes necessários no `README` do repositório. Assim, você também tem a opção de usá-lo como documentação mais concisa se você planeja executar o código enquanto lê o livro.

Para resumir tudo isso, neste capítulo, vamos explorar os seguintes tópicos:

- Ecossistema do Python e instalação do projeto
- Ferramentas de MLOps e LLMOps
- Bancos de dados para armazenamento de dados não estruturados e vetoriais
- Preparação para a AWS

Ao final deste capítulo, você conhecerá todas as ferramentas que usaremos ao longo do livro. Além disso, você terá aprendido a instalar o repositório `LLM-Engineers-Handbook`, configurar o restante das ferramentas e usá-las se executar o código enquanto lê o livro.

Ecossistema do Python e instalação do projeto

Qualquer projeto em Python precisa de três ferramentas fundamentais: o interpretador do Python, o gerenciamento de dependências e uma ferramenta de execução de tarefas. O interpretador do Python executa seu projeto em Python como esperado. Todo o código neste livro foi testado com Python 3.11.8. Você pode baixar o interpretador do Python aqui: `https://www.python.org/downloads/`. Recomendamos instalar a versão exata do Python (Python 3.11.8) para executar o projeto LLM Twin usando `pyenv`, tornando o processo de instalação simples.

Em vez de instalar várias versões globais do Python, recomendamos gerenciá-las usando `pyenv`, uma ferramenta de gerenciamento de versões do Python que permite gerenciar várias versões do Python entre projetos. Você pode instalá-la usando este link: `https://github.com/pyenv/pyenv?tab=readme-ov-file#installation`.

Depois de instalar `pyenv`, você pode instalar a versão mais recente do Python 3.11, usando `pyenv`, da seguinte forma:

```
pyenv install 3.11.8
```

Agora liste todas as versões do Python instaladas para confirmar que foi instalado corretamente:

```
pyenv versions
```

Você deve ver algo assim:

```
# * sistema
#   3.11.8
```

Para definir o Python 3.11.8 a versão padrão em todo o seu sistema (sempre que você abrir um novo terminal), use o seguinte comando:

```
pyenv global 3.11.8
```

No entanto, pretendemos usar o Python 3.11.8 apenas localmente em nosso repositório. Para isso, primeiro precisamos clonar o repositório e navegar até ele:

```
git clone https://github.com/PacktPublishing/LLM-Engineers-
Handbook.git
cd LLM-Engineers-Handbook
```

Como definimos um `arquivo` .python-version dentro do repositório, `pyenv` saberá obter a versão a partir daquele arquivo e usá-la localmente sempre que você estiver trabalhando dentro daquele diretório. Para confirmar isso, execute o seguinte comando enquanto estiver no repositório:

```
python --version
```

A saída deve ser:

```
# Python 3.11.8
```

Para criar o arquivo `.python-version`, você deve executar `pyenv local 3.11.8` uma vez. Em seguida, `pyenv` sempre saberá usar essa versão do Python enquanto estiver trabalhando em um diretório específico.

Agora que instalamos a versão correta do Python usando `pyenv`, vamos prosseguir para o Poetry, que usaremos como nosso gerenciador de dependências e de ambiente virtual.

Poetry: gerenciamento de dependências e de ambientes virtuais

Poetry é um dos gerenciadores de dependências e de ambientes virtuais mais populares dentro do ecossistema do Python. Mas vamos começar esclarecendo o que é um gerenciador de dependências. Em Python, um gerenciador de dependências permite especificar, instalar, atualizar e gerenciar bibliotecas ou pacotes externos (dependências) dos quais um projeto depende. Por exemplo, este é um arquivo de requisitos simples do Poetry que usa Python 3.11 e os pacotes Python `requests` e `NumPy` .

```
[tool.poetry.dependencies]
python = "^3.11"
requests = "^2.25.1"
numpy = "^1.19.5"

[build-system]
```

```
requires = ["poetry-core"]
build-backend = "poetry.core.masonry.api"
```

Ao usar o Poetry para fixar as versões das dependências, você garante que sempre instala a versão correta das dependências que seus projetos utilizam. O Poetry, por padrão, salva todas as suas dependências nos arquivos `pyproject.toml`, que são armazenados na raiz do seu repositório, como você pode ver no repositório LLM-Engineers-Handbook clonado.

Outra grande vantagem de usar o Poetry é que ele cria um novo ambiente virtual do Python no qual instala a versão do Python e as dependências especificadas. Um ambiente virtual permite isolar as dependências do seu projeto das dependências globais do Python e de outros projetos. Com isso, você garante que não haja conflitos de versão entre projetos. Por exemplo, vamos supor que o Projeto A precise de `numpy == 1.19.5`, e o Projeto B precise de `numpy == 1.26.0`. Se você mantiver ambos os projetos no ambiente global do Python, isso não funcionará, pois o Projeto B sobrescreverá a instalação do `NumPy`, o que corromperá o Projeto A e o impedirá de continuar trabalhando. Com o uso do Poetry, você pode isolar cada projeto em seu próprio ambiente do Python, com suas próprias dependências, evitando quaisquer conflitos de dependência.

Você pode instalar o Poetry aqui: `https://python-poetry.org/docs/`. Usamos o Poetry 1.8.3 ao longo do livro. Depois que o Poetry estiver instalado, navegue até o seu repositório LLM-Engineers-Handbook clonado e execute o seguinte comando para instalar todas as dependências necessárias do Python:

```
poetry install --without aws
```

Este comando identifica todas as dependências do seu repositório que estão listadas nos arquivos `pyproject.toml` e `poetry.lock`. Após a instalação, você pode ativar o ambiente do Poetry executando `poetry shell` no seu terminal ou prefixando todos os seus comandos de CLI da seguinte forma: poetry run.

Uma última observação sobre o Poetry é que ele fixa as versões exatas da árvore de dependências no arquivo `poetry.lock` com base nas definições adicionadas ao arquivo `project.toml`. Embora o arquivo `pyproject.toml` possa especificar intervalos de versão (por exemplo, `requests = "^2.25.1"`), o arquivo `poetry.lock` registra a versão exata (por exemplo, `requests = "2.25.1"`) que foi instalada. Ele também fixa as versões das subdependências (dependências das suas dependências), que podem não estar listadas explicitamente no seu arquivo `pyproject.toml`. Ao fixar todas as dependências e subdependências em versões específicas, o arquivo `poetry.lock` garante que todas as instalações do projeto usem as mesmas versões de cada pacote. Essa consistência leva a um comportamento previsível, reduzindo a verossimilhança de encontrar problemas do tipo "funciona na minha máquina".

Outras ferramentas semelhantes ao Poetry são Venv e conda para criação de ambientes virtuais. Ainda assim, elas não têm a opção de gerenciamento de dependências. Assim, você deve fazer isso por meio dos arquivos `requirements.txt` padrão do Python, que são menos poderosos do que os `arquivos de bloqueio` do Poetry. Outra opção é

Pipenv, que em termos de recursos é mais parecido com o Poetry, mas é mais lento, e `uv`, que é um substituto do Poetry escrito em Rust, tornando-o extremamente rápido. `uv` tem muito potencial para substituir o Poetry, vale a pena testar: `https://github.com/astral-sh/uv`.

A peça final do quebra-cabeça é analisar a ferramenta de execução de tarefas que usamos para gerenciar todos os nossos comandos de CLI.

Poe the Poet: ferramenta de execução de tarefas

Poe the Poet é um plugin baseado no Poetry que é usado para gerenciar e executar todos os comandos de CLI necessários para interagir com o projeto. Ele ajuda a definir e executar tarefas no seu projeto Python, simplificando a automação e a execução de scripts. Outras opções populares são Makefile, Invoke ou shell scripts, mas o Poe the Poet elimina a necessidade de escrever shell scripts ou Makefiles separados para gerenciar as tarefas do projeto, tornando-o uma forma elegante de gerenciar tarefas usando o mesmo arquivo de configuração que o Poetry já usa para dependências.

Ao trabalhar com o Poe the Poet, em vez de ter todos os seus comandos documentados em um arquivo README ou outro documento, você pode adicioná-los diretamente ao seu arquivo `pyproject.toml` e executá-los na linha de comando com um alias. Por exemplo, usando o Poe the Poet, podemos definir as seguintes tarefas em um arquivo `pyproject.toml`:

```
[tool.poe.tasks]
teste = "pytest"
format = "black ."
start = "python main.py"
```

Você pode então executar essas tarefas usando o `comando` poe:

```
poetry poe test
poetry poe format
poetry poe start
```

Você pode instalar o Poe the Poet como um plugin do Poetry, da seguinte forma:

```
poetry self add 'poethepoet[poetry_plugin]'
```

Para concluir, usar uma ferramenta como uma fachada sobre todos os seus comandos de CLI é necessário para executar sua aplicação. Isso reduz significativamente a complexidade da aplicação e melhora a colaboração, pois atua como documentação pronta para uso.

Supondo que você tenha o `pyenv` e o Poetry instalados, aqui estão todos os comandos que você precisa executar para clonar o repositório e instalar as dependências e o Poe the Poet como um plugin do Poetry:

```
git clone https://github.com/PacktPublishing/LLM-Engineers-
```

MANUAL DO ENGENHEIRO DE LLM

GPT/LLMS ■ INTELIGÊNCIA ARTIFICIAL ■ MACHINE LEARNING ■ DEEP LEARNING ■ DATA SCIENCE

A inteligência artificial evolui rapidamente, e os Large Language Models (LLMs) lideram essa transformação. Este livro apresenta, de forma prática, como projetar, treinar e implantar LLMs em cenários reais utilizando boas práticas de MLOps. O leitor aprende a construir sistemas escaláveis, modulares e prontos para produção, indo além de notebooks isolados. O conteúdo aborda engenharia de dados, fine-tuning supervisionado, otimização de inferência, alinhamento de preferências e processamento em tempo real. Ideal para profissionais que desejam aplicar LLMs de forma eficiente e confiável em projetos reais.

O que você vai encontrar:

- **Implementação de pipelines de dados robustos e gestão do ciclo de treinamento de LLMs**
- **Criação e refinamento de LLMs com exemplos práticos**
- **Introdução a LLMOps e princípios de MLOps, como orquestração e monitoramento de prompts**
- **Fine-tuning supervisionado e avaliação de LLMs**
- **Implantação de soluções completas com AWS e outras ferramentas**
- **Desenvolvimento de sistemas escaláveis e modulares**
- **Aplicações de RAG com construção de pipelines de features e inferência**

"*Manual do Engenheiro de LLM* é um guia prático e essencial para quem deseja dominar aplicações com LLMs. Com exemplos reais, o livro aborda fine-tuning, RAG, MLOps e otimização de inferência, além de ferramentas como Hugging Face, MongoDB e Qdrant. Uma referência valiosa para engenheiros e profissionais de IA generativa."

— Antonio Gulli, Senior Director, Google

Paul Iusztin é engenheiro sênior de ML e MLOps, com experiência em IA generativa, visão computacional e sistemas de ML em produção. Atuou na Metaphysic, CoreAI, Everseen e Continental, e é fundador da Decoding ML, plataforma educacional focada em aplicações reais de machine learning.

Maxime Labonne é Head of Post-Training na Liquid AI, doutor em Machine Learning pelo Instituto Politécnico de Paris e Google Developer Expert em IA/ML. Contribui ativamente para a comunidade open source com cursos, ferramentas e modelos de IA, além de ser autor do best-seller *Hands-On Graph Neural Networks Using Python*.

Conheça mais livros em:

tech.Blucher.com.br

ISBN 978-85-212-3347-3