

# vLLM e Qdrant

José Ricardo  
Workshop MLOps - 24/02

# —

# Contexto

Onde isso entra em MLOps?

- Docker → empacotar
- MLOps → pipeline e deploy
- Agora: rodar LLM e dar memória pra ele

Treinar modelo é só metade do problema. Servir o modelo e buscar contexto é o resto.

# —

# Problema Real

## Assistente Inteligente para Clínicas de Saúde

Imagine uma clínica que recebe **centenas** de perguntas por dia:

- “Qual o preparo para exame X?”
- “Quais convênios vocês aceitam?”
- “Quanto tempo dura a recuperação?”
- “Preciso de encaminhamento?”

Hoje isso é respondido por:

- Atendentes humanos
- PDFs espalhados
- Sistemas desatualizados

# —

# Problema Real

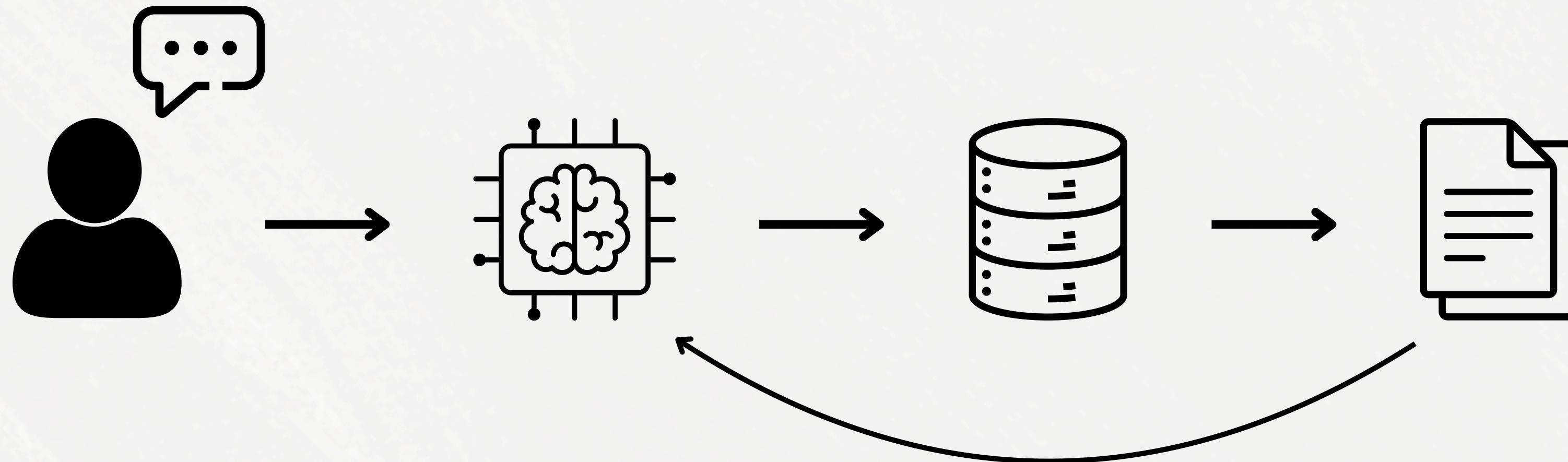
Assistente Inteligente para Clínicas de Saúde

Requisitos:

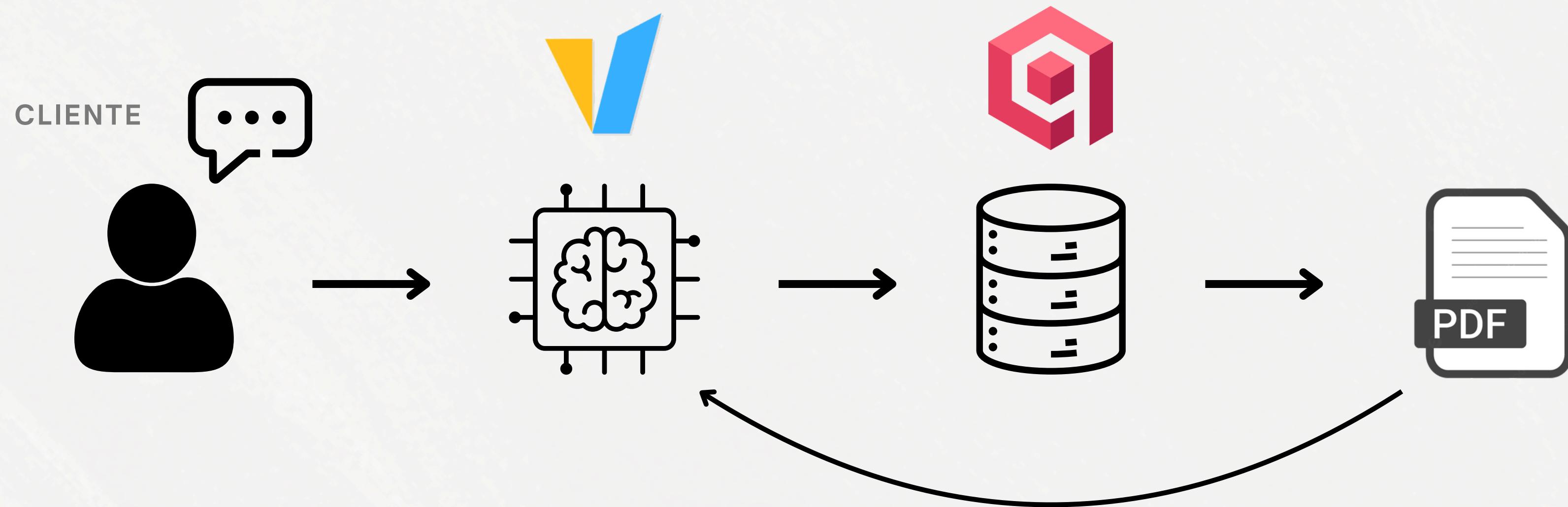
- **Inferência eficiente** de modelos de linguagem
- **Recuperação** de documentos internas
- Arquitetura controlada

É exatamente aqui que entram **vLLM** e **Qdrant**.

# Pipeline Geral



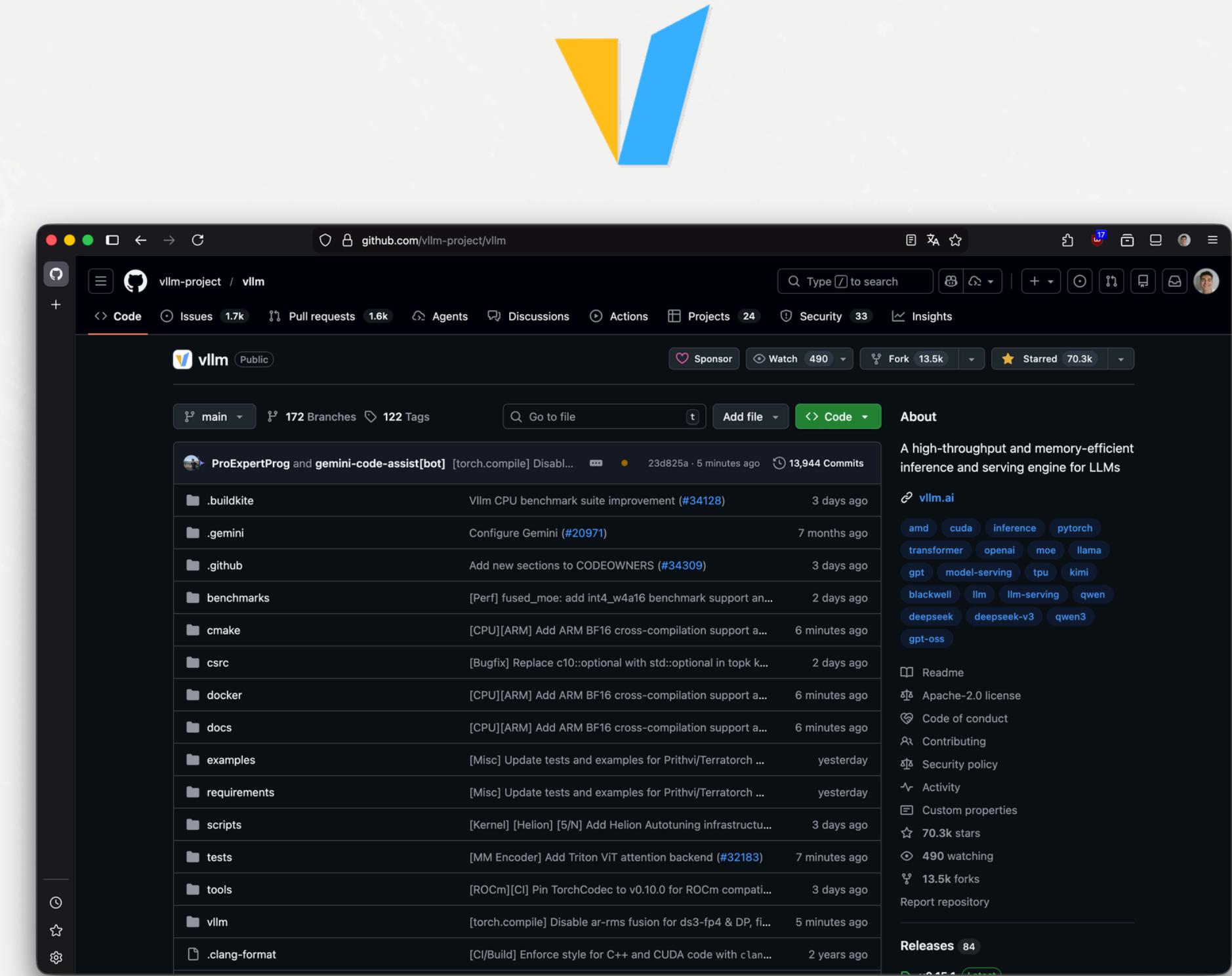
# Nosso Pipeline



# vLLM

O que é?

- Biblioteca **open source**
- Ajuda LLMs a performar de forma mais eficiente e escalável
- O objetivo do vLLM é **maximizar a taxa de throughput** (tokens processados por segundo) para atender a muitos usuários ao mesmo tempo.



# vLLM

## Efficient Memory Management for Large Language Model Serving with *PagedAttention*

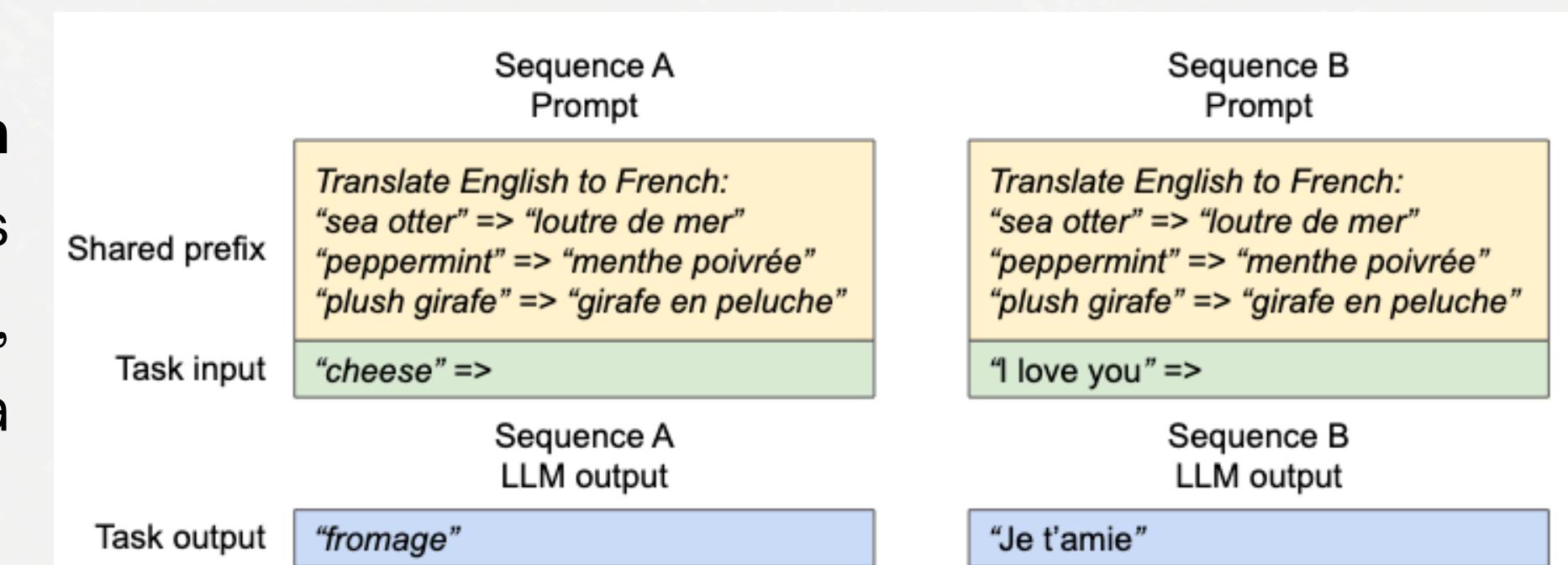
Woosuk Kwon<sup>1,\*</sup> Zhuohan Li<sup>1,\*</sup> Siyuan Zhuang<sup>1</sup> Ying Sheng<sup>1,2</sup> Lianmin Zheng<sup>1</sup> Cody Hao Yu<sup>3</sup>  
Joseph E. Gonzalez<sup>1</sup> Hao Zhang<sup>4</sup> Ion Stoica<sup>1</sup>  
<sup>1</sup>UC Berkeley <sup>2</sup>Stanford University <sup>3</sup>Independent Researcher <sup>4</sup>UC San Diego

- Identificou que sistemas de gerenciamento de memória LLM existentes **não organizam** os cálculos da maneira mais eficiente
- Introduziram a **PagedAttention**, uma técnica de gerenciamento de memória inspirada na memória virtual e nos sistemas de paginação dos sistemas operacionais

# KV Cache

## Resumo

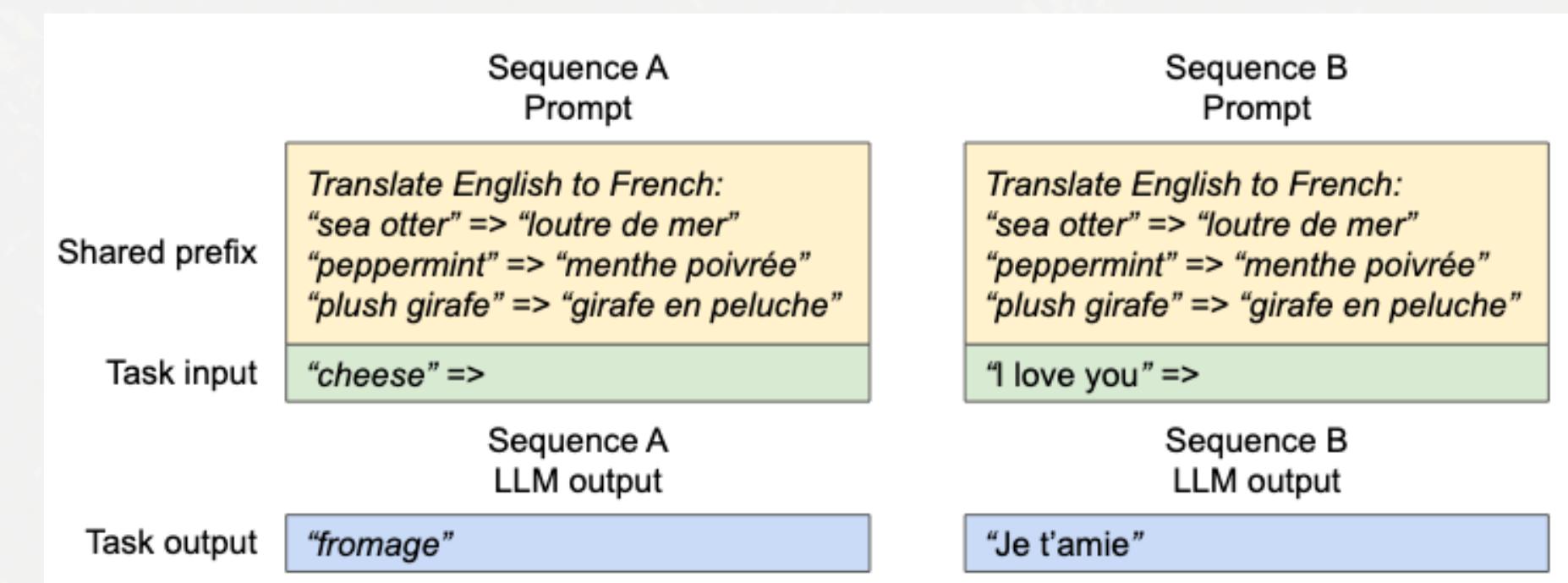
- KV refere-se à **maneira como um LLM formula o significado de uma palavra ou frase**. LLMs processam os KVs de maneira semelhante, pois mantêm o valor correspondente a cada chave (ou token) em seu cache
- Cache refere-se a um **armazenamento de memória de curto prazo**



# vLLM

## Como gerencia o KV Cache?

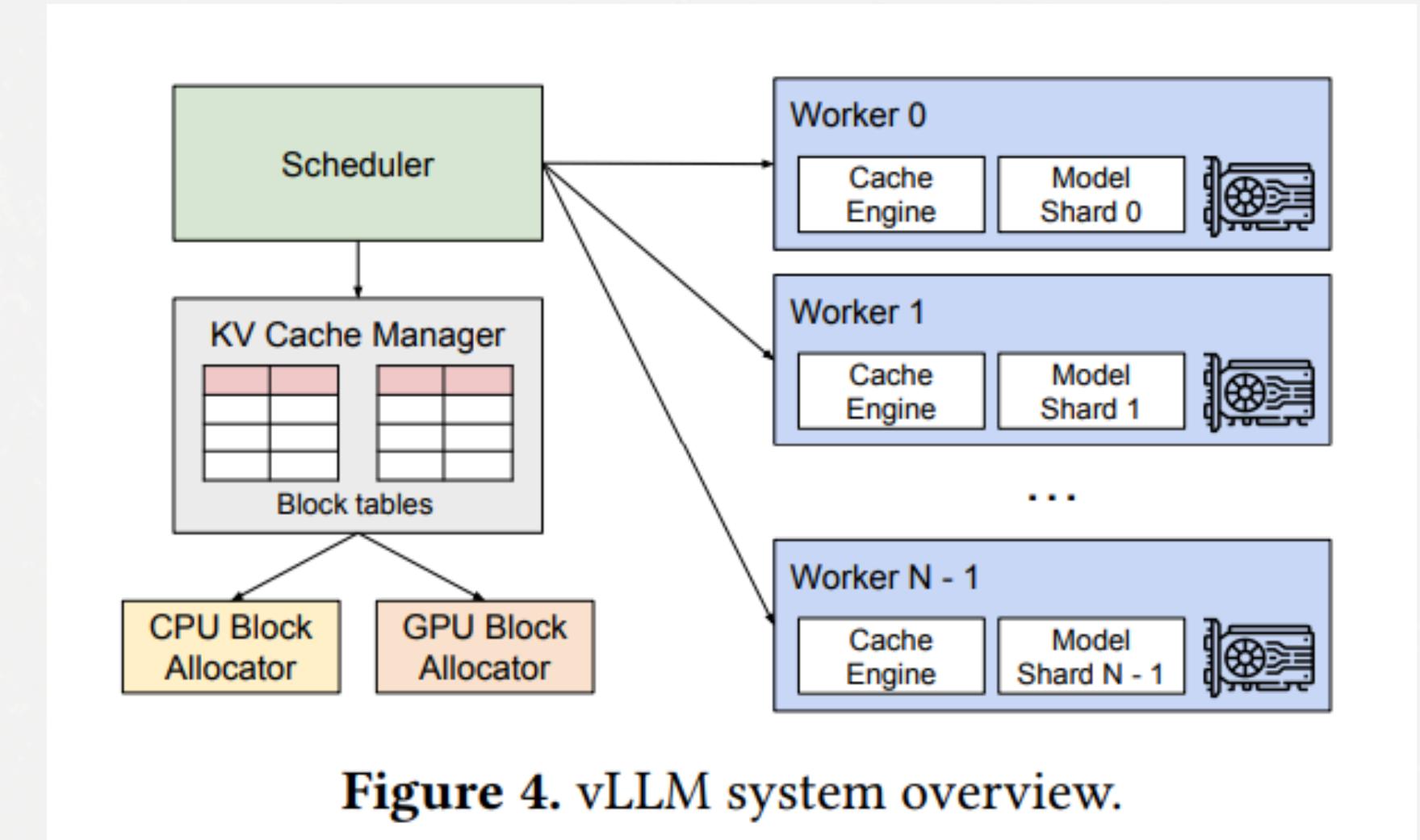
Em vez de gerar uma resposta totalmente nova, o vLLM permite que o cache KV mantenha a memória e **crie atalhos para novas consultas semelhantes** a cálculos computados anteriormente. O processamento dos cálculos de consultas semelhantes em lote (em vez de individualmente) **melhora o rendimento e otimiza a alocação de memória**



# vLLM

Como funciona?

- Divide o KV cache da requisição em **blocos**, cada um podendo conter as K e V de um número fixo de tokens
- Blocos não armazenados em **espaço contíguo**
- Espaço passa a ser alocado dinamicamente, o que evita **espaços ociosos** da GPU
- Permite **compartilhamento de KV cache entre requests**, principalmente em sistemas com system prompt



**Figure 4.** vLLM system overview.

# vLLM

Aplicação end-to-end



**Kernels e  
PagedAttention**



**Schedulers e  
Blocos**



**API para  
Inferência**

# vLLM

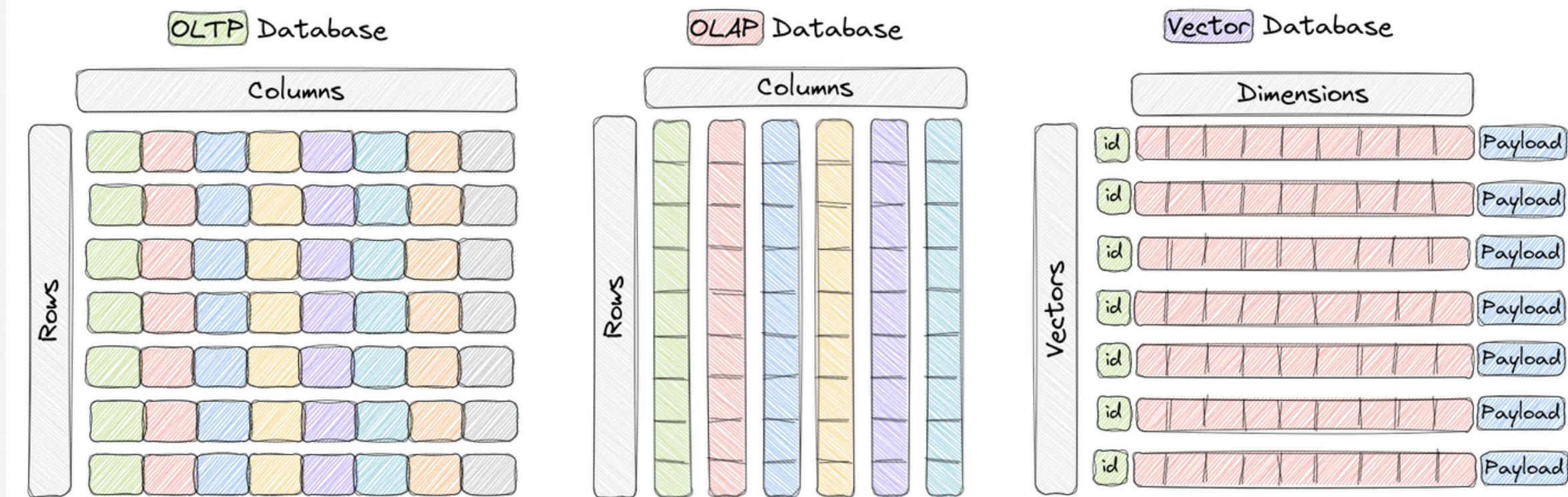
## Benefícios

- **Respostas mais rápidas** (alto desempenho): vLLM consegue processar muito **mais requisições ao mesmo tempo**
- **Redução de custos de hardware**: usa melhor a memória e as GPUs, o que significa que é possível atender a mesma demanda com menos placas de vídeo e menor gasto com infraestrutura
- **Escalabilidade**: consegue lidar com **muitos usuários simultaneamente** sem travar o sistema. Isso é essencial para aplicações com vários pedidos ao mesmo tempo
- **Mais controle e privacidade de dados**: Permite **hospedar o modelo internamente (self-host)**. Assim, a empresa não precisa enviar dados sensíveis para serviços de terceiros

# Banco de dados vetorial

- Biblioteca **open source**
- Ajuda LLMs a performar de forma mais eficiente e escalável
- O objetivo do vLLM é **maximizar a taxa de throughput** (tokens processados por segundo) para atender a muitos usuários ao mesmo tempo.

# Banco de Dados Vetorial

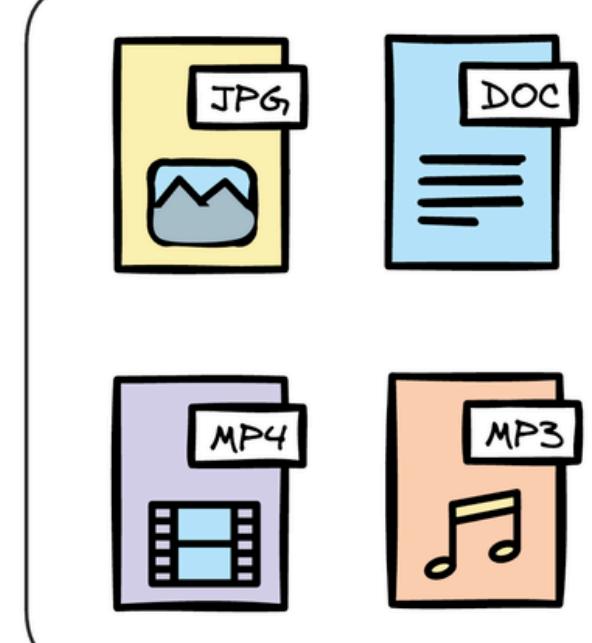


- Os **bancos de dados vetoriais** são um tipo de banco de dados projetado para armazenar e consultar **vetores de alta dimensão** com eficiência.
- Nos bancos de dados OLTP e OLAP tradicionais, os dados são organizados em linhas e colunas (chamadas de tabelas), e as consultas são realizadas com base nos valores dessas colunas
- No entanto, em certas aplicações, incluindo reconhecimento de imagem, NLP e sistemas de recomendação, os dados são frequentemente representados como vetores em um espaço de alta dimensão

# Banco de Dados Vetorial

- São **otimizados** para armazenar e consultar esses vetores de alta dimensão com eficiência
- Permitem uma pesquisa rápida por **similaridade e semântica**, ao mesmo tempo em que permitem aos usuários encontrar os **vetores mais próximos** de um determinado vetor de consulta com base em alguma métrica de distância

## Vector Database



Unstructured data

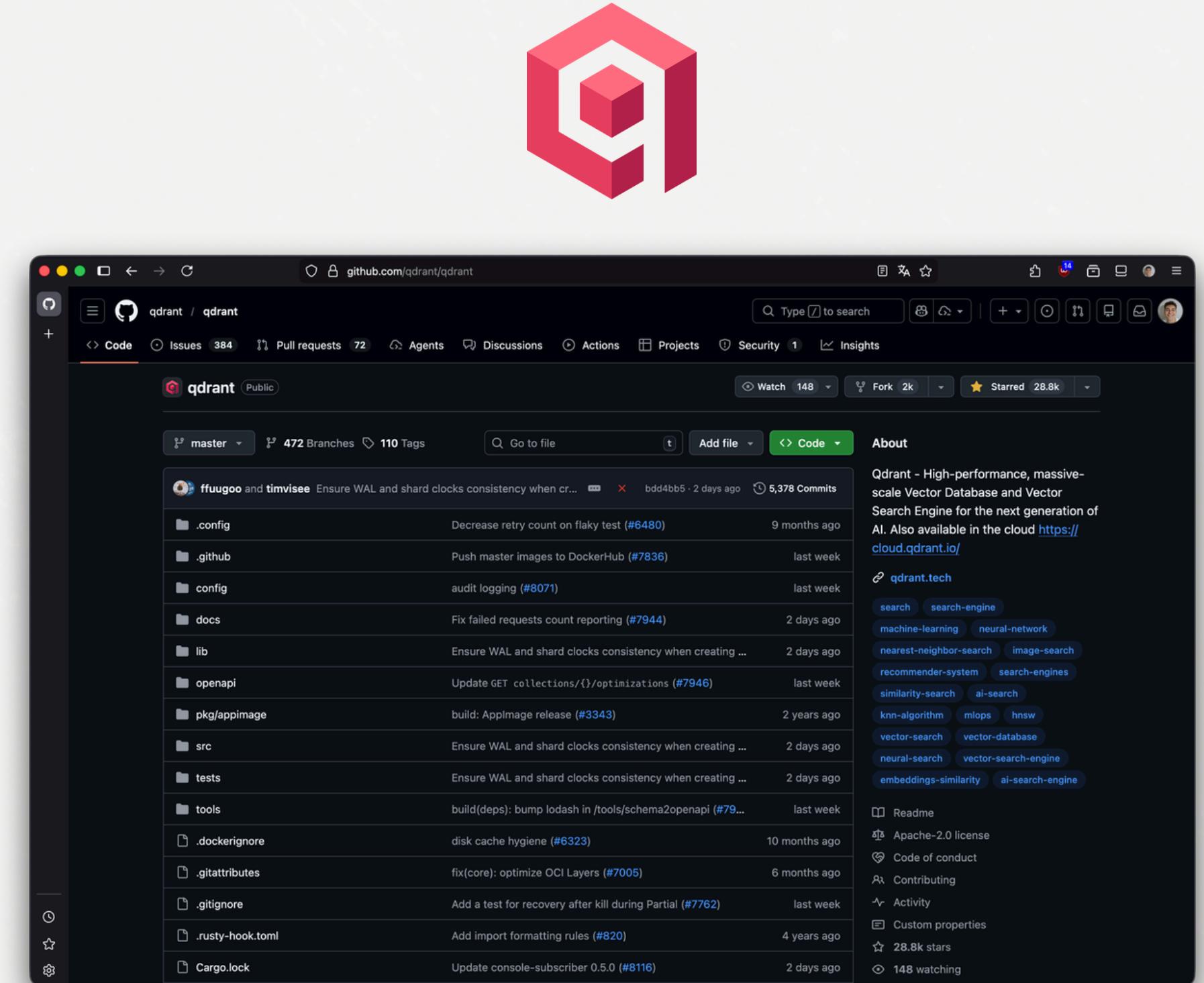
0.2	-1.7	...	2.3
0.4	0.5	...	-1.7
4.1	-1.9	...	-1.5
-1.1	0.7	...	5.3
-3.5	2.3	...	0.5
-1.7	0.4	...	0.2
2.3	0.2	...	0.7
-1.9	4.1	...	-2.4
0.5	-1.5	...	2.3

Embeddings

# Qdrant

## O que é?

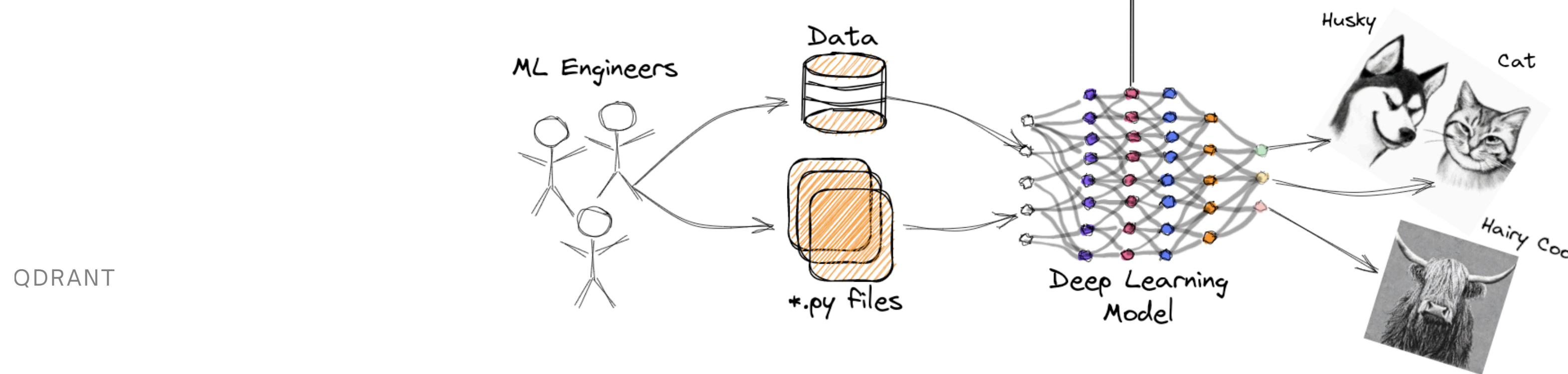
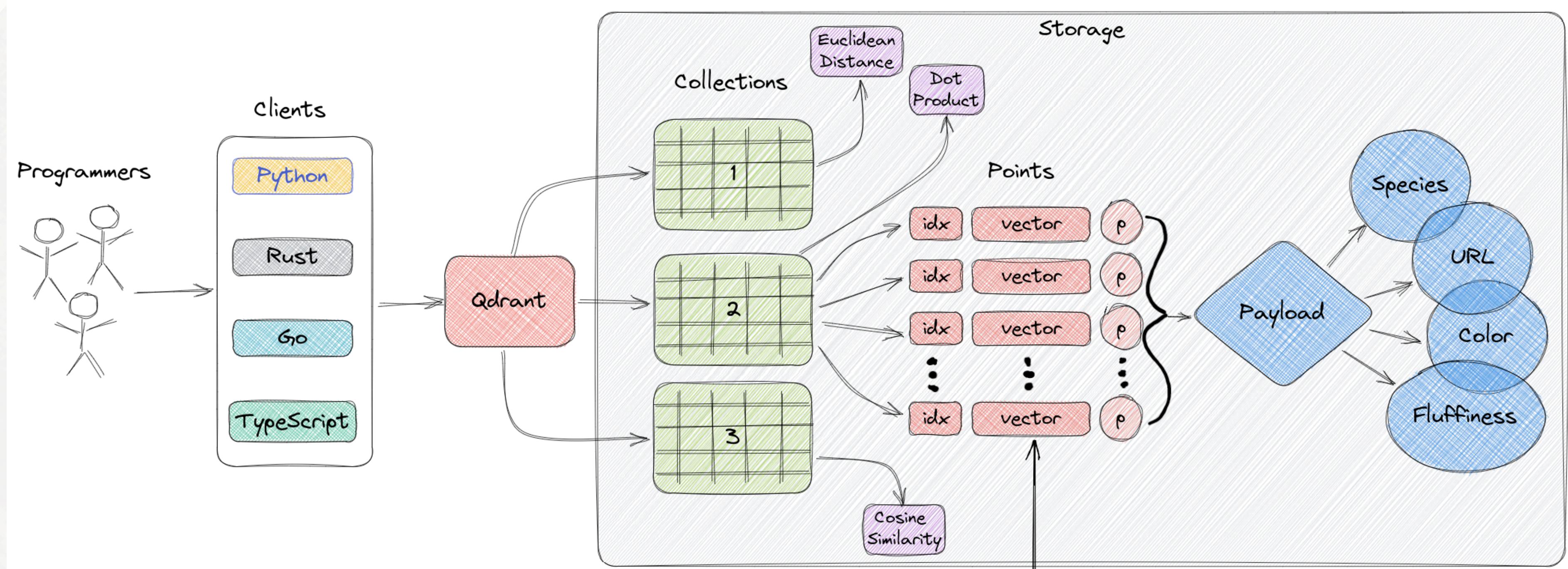
- Vector Database open source e uma search engine por similaridade de vetores escrito em Rust
- Rápido, escalável e pronto para produção, com uma API para armazenar, pesquisar e gerenciar vetores



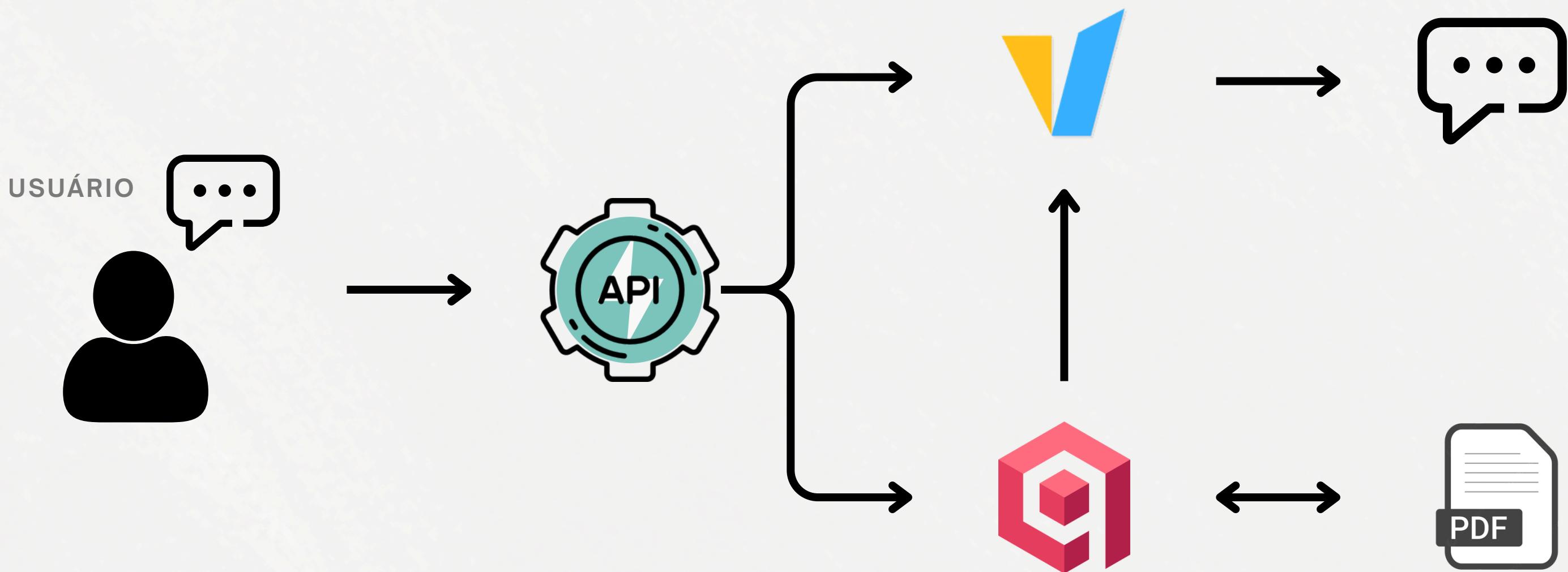
# Qdrant

O que é?

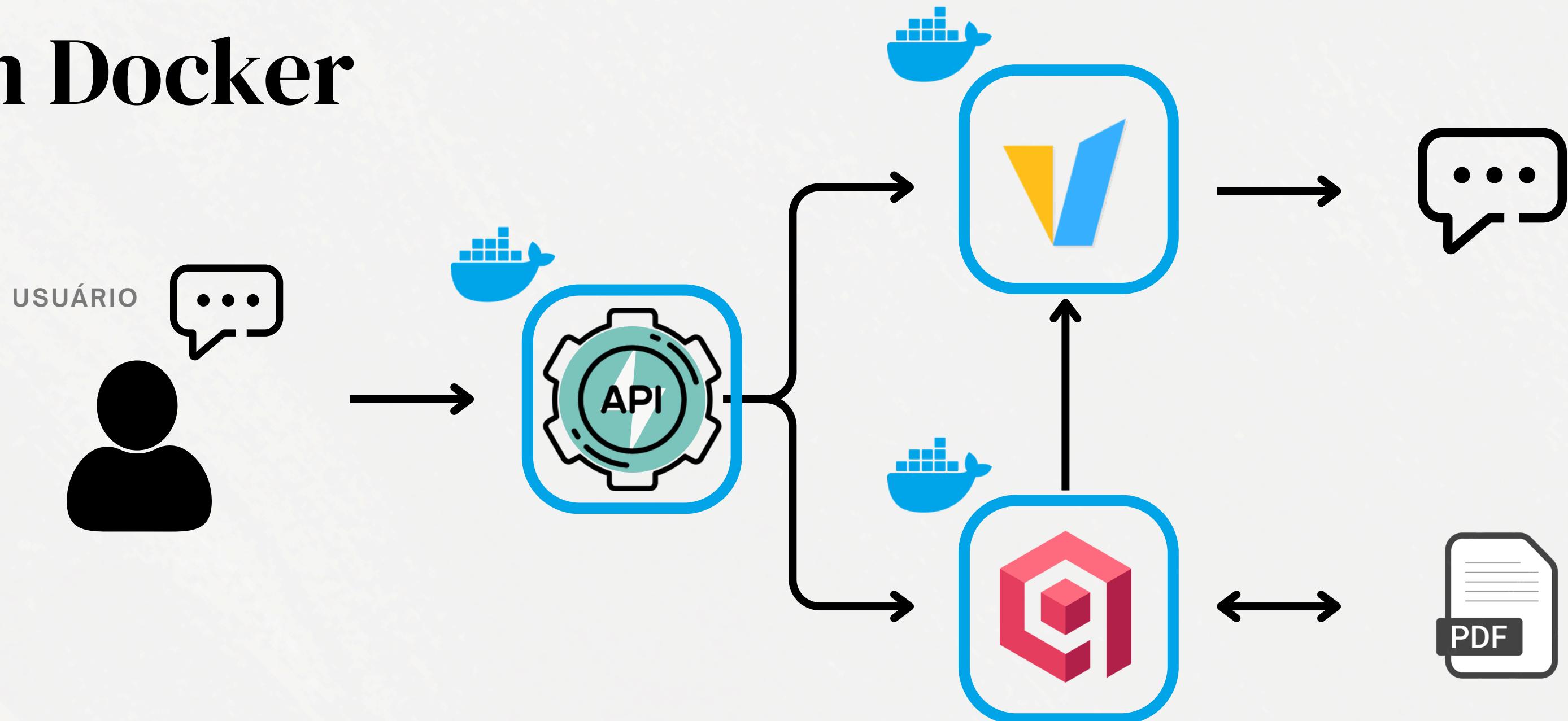
- **Coleções** (Collections): **Conjunto de vetores** (pontos) onde é possível fazer buscas. Todos os vetores da mesma coleção têm a mesma dimensão e usam uma única métrica de distância.
- **Métricas de Distância** (Distance Metrics): **Medem a similaridade entre vetores**. Devem ser escolhidas ao criar a coleção e dependem do modelo de IA usado para gerar os vetores
- **Pontos** (Points): Unidade principal do Qdrant. Cada ponto contém:
  - ID: identificador único
  - **Vetor**: representação numérica de dados (texto, imagem, áudio etc.)
  - Payload: dados extras em formato JSON (metadados)
- **Clientes** (Clients): Linguagens de programação usadas para se **conectar ao Qdrant**.

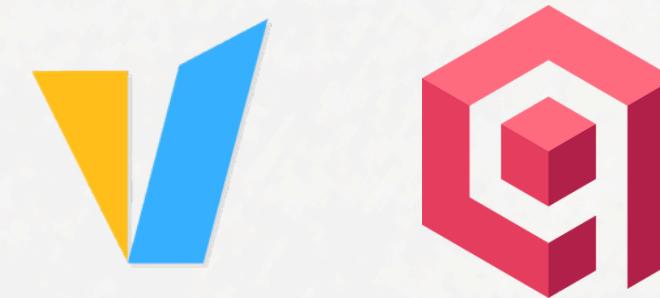


# Pipeline Demo



# Pipeline Demo com Docker





# Obrigado!

José Ricardo  
Workshop MLOps - 24/02