FACULDADE DE TECNOLOGIA DE PRESIDENTE PRUDENTE

ARQUITETURA PARA BUSCA VETORIAL EM SISTEMAS DE BUSCA EM ARTIGOS CIENTÍFICOS

ARCHITECTURE FOR VECTOR SEARCH IN SCIENTIFIC ARTICLE SEARCH SYSTEMS

João Victor Rizzo Neto[1*]

Ana Carolina Nicolosi da R. Gracioso[2**]

Resumo

Este trabalho propôs o desenvolvimento de uma arquitetura de *software* voltada à busca vetorial, utilizando modelos de linguagem baseados em *embeddings* contextuais. O objetivo foi proporcionar uma estrutura escalável, precisa e de baixo tempo de resposta para sistemas que realizam consultas em grandes volumes de dados, como repositórios de artigos científicos e *blogs* especializados. A pesquisa envolveu a investigação de técnicas modernas de representação vetorial, a definição de uma arquitetura distribuída com uso de *cache* e balanceamento de carga, bem como a implementação e validação da solução em ambiente controlado. Os resultados demonstraram melhorias notáveis em desempenho e precisão, com um tempo médio de resposta entre 15 e 39 milissegundos para consultas completas, contribuindo com soluções práticas para sistemas de busca semântica.

Palavras-chave: busca vetorial, *embeddings*, BERT, arquitetura distribuída, otimização de

[1*] Graduando do curso superior de Análise e Desenvolvimento de Sistemas na Fatec de Presidente Prudente. E-mail: joao.rizzo@fatec.sp.gov.br.

[2**] Docente da FATEC Presidente Prudente. E-mail: ana.gracioso@fatec.sp.gov.br.

desempenho.

Abstract

This work proposes the development of a software architecture designed for vector-based search using language models built on contextual embeddings. The goal was to provide a scalable, accurate, and low-latency structure suitable for systems that perform queries over large datasets, such as scientific article repositories and specialized blogs. The research involved investigating modern techniques for vector representation, defining a distributed architecture with caching and load balancing, and implementing and validating the solution in a controlled environment. The results demonstrated notable improvements in both performance and accuracy, with average response times ranging from 15 to 39 milliseconds for complete queries. These findings contribute practical solutions for semantic search systems.

Keywords: *vector search, embeddings, BERT, distributed architecture, performance optimization.*

1. INTRODUÇÃO

A crescente produção de conteúdo digital, especialmente em ambientes científicos e acadêmicos, tem exigido soluções de busca mais eficientes e precisas. Os métodos tradicionais, baseados em palavras-chave, têm se mostrado limitados diante da complexidade

semântica dos textos. Nesse contexto, técnicas de busca vetorial com *embeddings* contextuais representam uma alternativa promissora para melhorar a relevância e a eficiência na recuperação de informações.

Este trabalho propõe o desenvolvimento de uma arquitetura distribuída para sistemas de busca vetorial, com foco na otimização de tempo de resposta e precisão dos resultados. A proposta foi validada por meio de experimentação com bases reais de artigos científicos e outros conteúdos relacionados.

1.1. Contextualização teórica e prática do tema

Nos últimos anos, técnicas baseadas em aprendizado profundo transformaram a forma como lidamos com textos. Modelos como *BERT (Bidirectional Encoder Representations from Transformers)* e *Sentence-BERT* permitem a representação vetorial de sentenças com alto grau de entendimento contextual. Tais representações, conhecidas como *embeddings*, permitem medir a similaridade entre textos com maior fidelidade semântica.

Na prática, repositórios de artigos científicos enfrentam desafios como o aumento constante de volume de dados e a demanda por respostas rápidas e relevantes. Portanto, há uma necessidade real de arquiteturas escaláveis, com uso eficiente de recursos computacionais e que incorporem técnicas de *cache*, particionamento e balanceamento de carga.

1.2. Justificativa

A relevância do presente estudo reside na crescente demanda por sistemas de busca inteligentes, capazes de entender a linguagem natural em profundidade. O uso de *embeddings* contextuais associado a uma arquitetura eficiente permite superar limitações de sistemas tradicionais. Além disso, a pesquisa contribui com o avanço tecnológico no campo de recuperação de informação, com impacto direto em sistemas acadêmicos, motores de recomendação e assistentes digitais.

1.2.1. Objetivo Geral

Desenvolver e validar uma arquitetura de software que otimize o tempo de resposta em sistemas de busca vetorial, garantindo que consultas com *embeddings* contextuais sejam processadas com precisão, mesmo em grandes volumes de dados.

1.2.2. Objetivos Específicos

- Investigar técnicas de *embeddings* contextuais (ex.: *BERT, Sentence-BERT, w2v*) e algoritmos de busca vetorial
- Propor uma arquitetura distribuída que minimize o consumo de memória e maximize o desempenho, utilizando *cache* e particionamento de dados.
- Implementar e validar a arquitetura proposta em um ambiente realista.
- Avaliar o tempo de resposta, a precisão dos resultados.

1.3. Fundamentação Teórica

A busca vetorial tem se consolidado como uma das principais abordagens para recuperação de informações em sistemas modernos, especialmente em contextos onde a relevância semântica é mais importante do que a correspondência literal de palavras-chave. Diferentemente das buscas tradicionais baseadas em índices invertidos, a busca vetorial representa textos como vetores em um espaço de alta dimensionalidade, permitindo medir a similaridade entre consultas e documentos de maneira contextual (SALTON; WONG; YANG, 1975).

Essa representação é obtida por meio de modelos de *embeddings*, capazes de converter palavras, frases ou documentos em vetores numéricos que capturam suas relações semânticas. Entre os modelos mais amplamente utilizados, destacam-se o *Word2Vec* (MIKOLOV et al., 2013; LINS, 2020) e o *Sentence-BERT* (REIMERS; GUREVYCH, 2019). O primeiro, utilizado neste trabalho, baseia-se em redes neurais superficiais para aprender representações vetoriais de palavras a partir de grandes corpos textuais, permitindo o cálculo de médias vetoriais para representar sentenças de forma eficiente e leve.

O avanço dos modelos de linguagem e o aumento do poder computacional tornaram possível integrar esses vetores a sistemas de busca com alto desempenho. Bancos de dados vetoriais como *Qdrant, FAISS (Facebook AI Similarity Search) e Pinecone* foram projetados para armazenar e consultar vetores de maneira otimizada, utilizando algoritmos de busca aproximada de vizinhos mais próximos *(ANN – Approximate Nearest Neighbors)*. Dentre essas soluções, o *Qdrant* (QDRANT, 2025) se destaca por oferecer uma *API (Application Programming Interface)* moderna, suporte nativo a filtros híbridos e integração com diversas

linguagens, além de possibilitar execução local ou em nuvem, o que o torna ideal para aplicações modulares e escaláveis (MÜLLER; THOMAS, 2022).

Além disso, arquiteturas modernas fazem uso de microserviços, balanceadores de carga e armazenamentos em memória, possibilitando escalabilidade horizontal e resiliência a falhas (RICHARD; FORD, 2020).

1.4. Metodologia

O presente trabalho adota uma abordagem aplicada e experimental, voltada ao desenvolvimento e validação de uma arquitetura computacional para busca vetorial de artigos científicos. O objetivo foi construir um sistema funcional capaz de realizar indexação, armazenamento e busca semântica de textos, permitindo identificar documentos similares com base em significado, e não apenas em palavras-chave.

A metodologia foi dividida em quatro etapas principais:

1. **Coleta e preparação dos dados:** foram utilizados textos científicos em formato PDF, processados e enviados a um tópico do *Apache Kafka* (plataforma de mensagens distribuídas).
2. **Vetorização e indexação:** os textos foram convertidos em vetores numéricos por meio de *embeddings* gerados com *Word2Vec* e armazenados em um banco vetorial *Qdrant*.
3. **Persistência e metadados:** as informações de cada artigo, como título e identificador único, foram armazenadas no banco *MongoDB* (MONGODB INC., 2025) (Banco de dados *NoSQL* (*Not Only SQL*) orientado a documentos).
4. **Desenvolvimento da *API* de busca:** foi implementado um serviço *REST* (Representational State Transfer) em *FastAPI* (RAMÍREZ, 2025) (*framework* Python para *APIs*), utilizando *Redis (Remote Dictionary Server)* como sistema de *cache*, para realizar consultas vetoriais e retornar os artigos mais relevantes.

Para integração entre os componentes, optou-se por uma arquitetura de microsserviços desacoplada (RICHARD; FORD, 2020), facilitando a escalabilidade e o reaproveitamento futuro dos módulos em outros projetos de busca semântica.

Esses componentes, em conjunto, formam uma arquitetura robusta, escalável e semanticamente inteligente, capaz de realizar buscas contextuais em artigos científicos de

maneira eficiente, com resultados mais relevantes do que os obtidos por métodos tradicionais de pesquisa textual.

1.4.1. Técnicas e Instrumentos

O desenvolvimento foi realizado majoritariamente em Python 3.12, com suporte das seguintes ferramentas:

- ***Apache Kafka*:** para comunicação assíncrona entre produtores e consumidores de mensagens;
- ***MongoDB*:** banco de dados *NoSQL* para armazenar metadados dos artigos;
- ***Qdrant*:** banco de vetores especializado em buscas por similaridade;
- ***FastAPI*:** *framework* para construção da *API* de busca semântica;
- ***Word2Vec* (*Gensim*):** modelo utilizado para geração dos *embeddings* textuais; Utilizei o método de divisão em blocos *"chunks" na geração dos vetores.*
- **Ollama (*Llama2*):** modelo de linguagem usado para resumir textos longos antes da indexação, para obter um vetor com o contexto "completo" do artigo. Pois como citado acima, utilizei o método de divisão em blocos "*chunks*" nos vetores do artigo, o que gera um contexto mais limitado por vetor.
- ***Docker Compose*:** ferramenta de orquestração de *containers Docker*, utilizado para orquestração e *deploy* local dos serviços.

1.4.2. Ambiente de Estudo

A arquitetura foi validada em ambiente computacional simulado, com base em um *corpus* real de aproximadamente 2.000 documentos (artigos científicos e conteúdos especializados) em português extraídos de blogs e sites de divulgação científica.

1.4.3. Etapas da Pesquisa

1. Levantamento bibliográfico sobre *embeddings* e busca vetorial.
2. Definição e modelagem da arquitetura proposta.
3. Implementação dos principais componentes do sistema: *API*, camada de pré-processamento, indexação, *cache*, motor de busca e monitoramento.
4. Validação da arquitetura utilizando conjunto de dados reais.
5. Coleta e análise dos dados com base em métricas de desempenho e precisão.

2. DESENVOLVIMENTO

Nessa seção foram apresentadas as fundamentações teóricas, a metodologia e os resultados da argumentação proposta neste artigo.

2.1. Arquitetura Proposta

A arquitetura proposta é composta por cinco módulos principais que interagem de forma orquestrada:

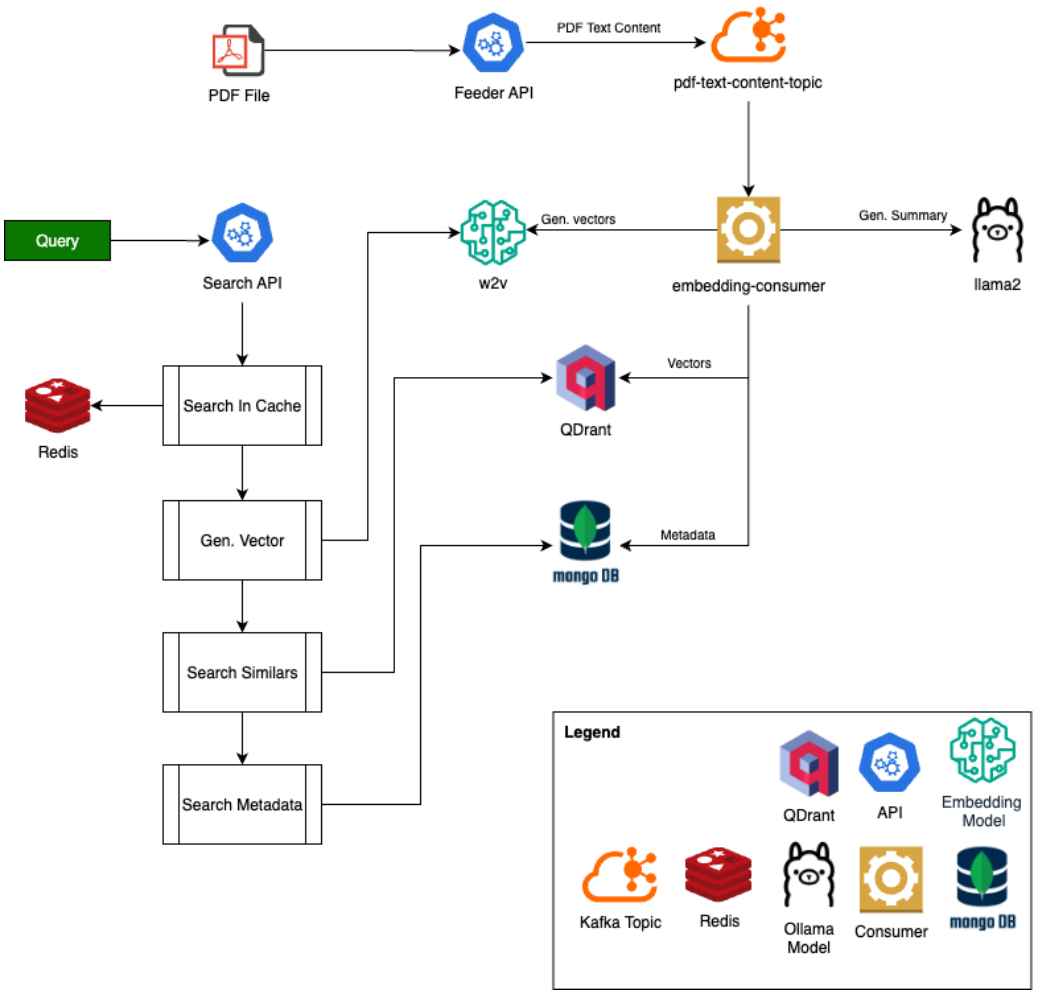
1. *API* de Recebimento de PDF's (Produtor Kafka)
2. Consumer (Indexador)
3. Banco Vetorial (*Qdrant*),
4. Banco de Metadados (*MongoDB*),
5. *API* de Busca (*FastAPI*).
6. *Cache* (*Redis*)

O fluxo de dados ocorre da seguinte maneira: os artigos são recebidos pela *api* de recebimento de pdfs, extraído os textos do pdf, e então enviados ao Kafka, consumidos pelo indexador que realiza a vetorização e resumo, armazenados no *Qdrant* e MongoDB, e posteriormente consultados pela *API*, que primeiramente verifica o *cache* para a *query* buscada, e em caso de falha, vetoriza e busca no *Qdrant*.

2.2. Visão Geral da Arquitetura

A arquitetura é orientada a eventos, permitindo o processamento assíncrono e escalável dos componentes. Essa abordagem possibilita lidar com grandes volumes de dados sem bloquear o fluxo de ingestão. Podemos observar o diagrama da arquitetura completa na Figura 1.

Figura 1 - Diagrama geral da arquitetura.

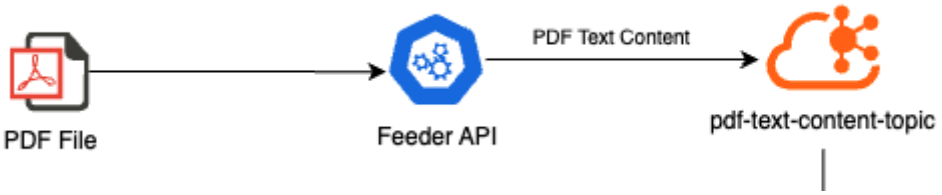


Fonte: Autor

2.3 Módulo de Recebimento de PDF (*FastAPI* + Produtor Kafka)

A *API* foi desenvolvida em *FastAPI*, fornecendo o *endpoint* "/*upload*". Esse *endpoint* é responsável por receber o arquivo PDF e o título referente a ele, extrair o texto utilizando a biblioteca *pdfplumber* e enviar para um tópico kafka. Para melhor entendimento do fluxo, vide Figura 2.

Figura 2 - Módulo de recebimento de PDF.



Fonte: Autor

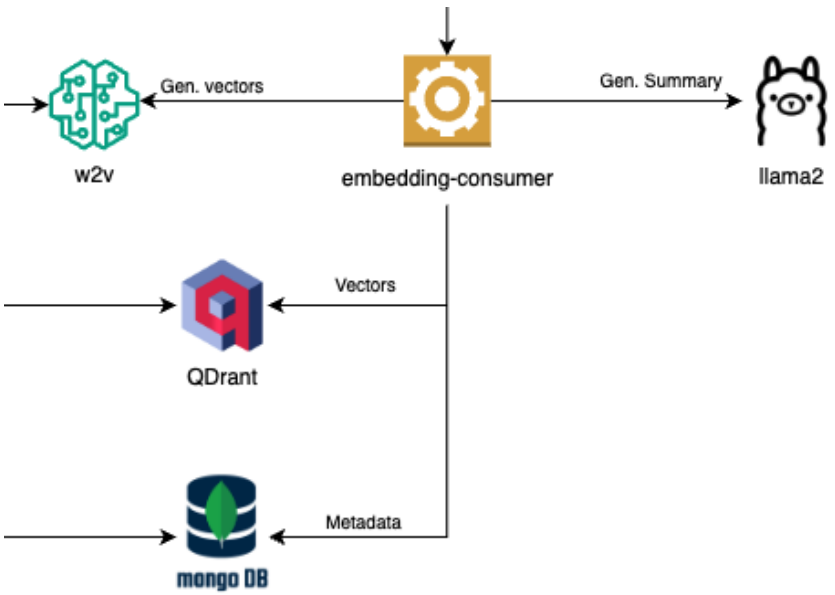
2.4 Módulo de Indexação (Consumer)

O Consumer é responsável por consumir mensagens do tópico Kafka contendo o texto completo dos artigos. As principais funções executadas são:

1. **Divisão em blocos (“*chunks*”)** de tamanho fixo (2000 *tokens*, com sobreposição de 50) para evitar perda de contexto;
2. **Geração de *embeddings*** com o modelo *Word2Vec*;
3. **Resumo automático** via modelo *Llama2* do Ollama;
4. **Inserção dos vetores no *Qdrant***, incluindo o *embedding* do resumo como vetor representativo do artigo;
5. **Registro de metadados no MongoDB**, evitando duplicidades através do *hash* do conteúdo textual.

Esse módulo foi desenvolvido de forma flexível, conforme apresentado na Figura 3, permitindo que, caso surja a necessidade de extrair informações adicionais dos artigos — como nome dos autores, data de publicação, palavras mais frequentes, possíveis categorias ou outros metadados —, o próprio modelo Llama 2 possa ser utilizado para realizar essa extração, bastando apenas ajustar o conteúdo do *prompt*. Essa funcionalidade não foi aplicada nesta arquitetura, uma vez que não havia demanda para tais dados no escopo atual.

Figura 3 - Módulo de Indexação



Fonte: Autor

2.5 Módulo de Armazenamento Vetorial (*Qdrant*)

O *Qdrant* foi escolhido por oferecer suporte nativo a consultas por similaridade vetorial com métricas de distância do tipo *COSINE*. Cada vetor armazenado representa semanticamente um trecho de texto (chunk) ou o resumo do artigo.

Ao realizar uma busca, o sistema calcula a similaridade entre o vetor da consulta e os vetores armazenados, retornando os resultados mais próximos.

2.6 Módulo de Persistência (MongoDB)

O MongoDB armazena informações estruturais e metadados dos artigos, como:

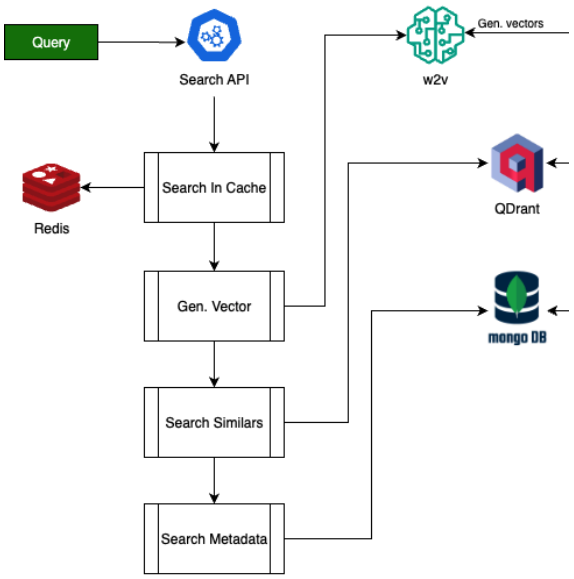
- ID
- Título
- *Hash* do conteúdo (para evitar duplicidade)
- Data de indexação

A separação entre o armazenamento vetorial e os metadados garante maior desempenho nas buscas e flexibilidade no gerenciamento de dados.

2.7 Módulo de Busca Semântica (*FastAPI*)

A *API* desenvolvida em *FastAPI* fornece o *endpoint* " */search*", responsável por receber uma consulta textual, verificar o *cache*, gerar seu *embedding* e buscar os vetores mais similares no *Qdrant*. O sistema retorna os títulos e identificadores dos artigos mais relevantes. Podemos observar na Figura 4, a entrada da consulta, geração do *embedding* e retorno dos resultados.

Figura 4. Fluxo da *API*



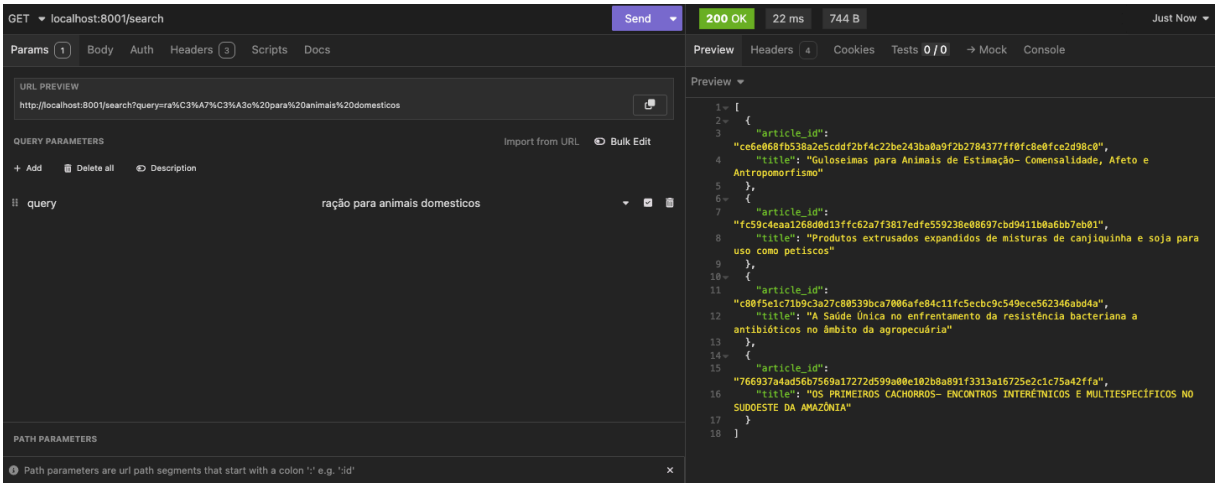
Fonte: Autor

3. EXPERIMENTOS E VALIDAÇÃO

Para validar a arquitetura, foram realizados testes com um conjunto de artigos científicos em português. Cada documento foi processado, resumido e indexado. Em seguida, foram executadas consultas semânticas para verificar a precisão dos resultados, conforme ilustrado nas Figuras 5, 6 e 7.

1. **"ração para animais domésticos"**

Figura 5 - Resultados *query* "ração para animais domésticos"



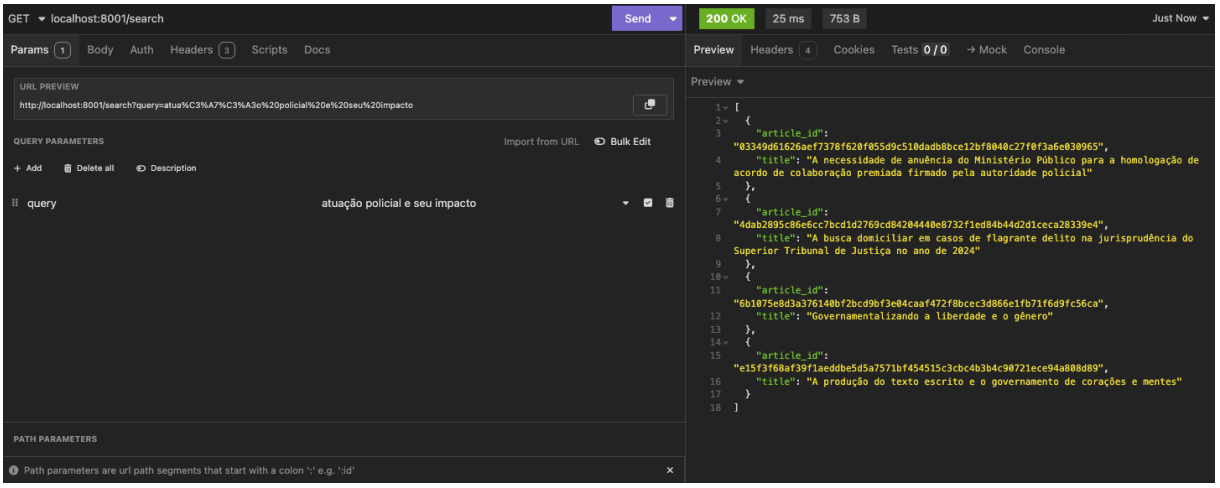
Fonte: Autor

Resultados:

- Guloseimas para Animais de Estimação: Comensalidade, Afeto e Antropomorfismo (Barbosa Osório, Andréa, 2019)
- Produtos extrusados expandidos de misturas de canjiquinha e soja para uso como petiscos (Fernandes; Wang; Ascheri; de Oliveira; Costa, 2002)
- A Saúde Única no enfrentamento da resistência bacteriana a antibióticos no âmbito da agropecuária (Silva; Luiza; Bermudez; Schneider, 2025)
- Os primeiros cachorros - Encontro interétnicos e multiespecíficos no sudoeste da Amazônia (Ferreira Vander Velden, Felipe, 2018)

2. “atuação policial e seu impacto”

Figura 6 - Resultados *query* “atuação policial e seu impacto”



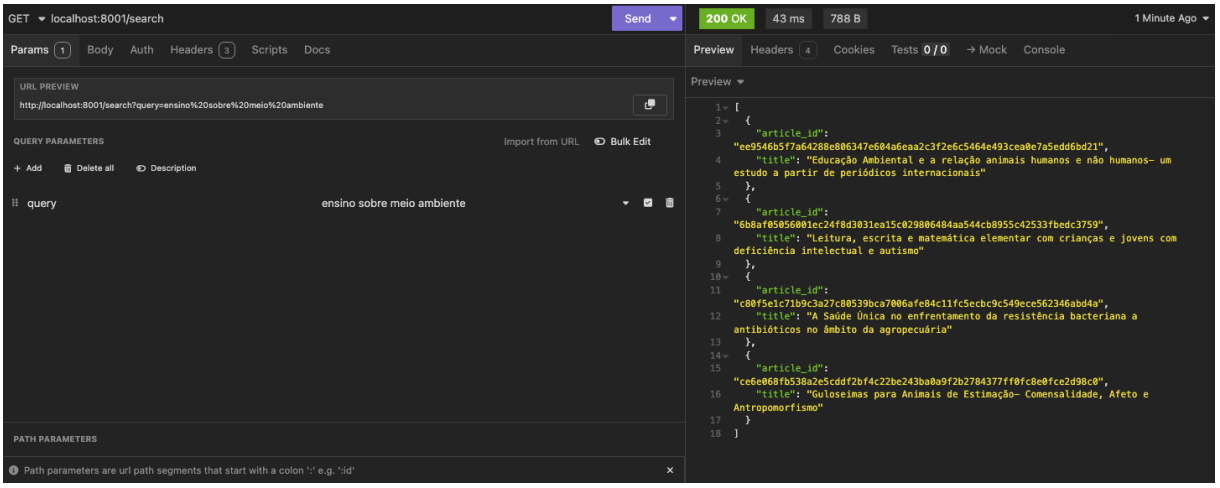
Fonte: Autor

Resultados:

- A necessidade de anuência do Ministério Público para a homologação de acordo de colaboração premiada firmado pela autoridade policial (Walter da Rosa, Luísa, 2023)
- A busca domiciliar em casos de flagrante delito na jurisprudência do Superior Tribunal de Justiça no ano de 2024 (Johner, Marcos Afonso, 2025)
- Governamentalizando a liberdade e o gênero (Ferreira; Goldbach; Grammatikopoulos; Gama Santos; Clemente, 2025)
- A produção do texto escrito e o governamento de corações e mentes (Amado; de Aguiar, 2025)

3. “ensino sobre meio ambiente”

Figura 7 - Resultados *query* “ensino sobre meio ambiente”



Fonte: Autor

Resultados:

- Educação Ambiental e a relação animais humanos e não humanos- um estudo a partir de periódicos internacionais (Simão; Cavalari, 2025)
- Leitura, escrita e matemática elementar com crianças e jovens com deficiência intelectual e autismo (Almeida; Domeniconi;Benitez;Buzatto; do Espírito Santo;da Silva;De Souza, 2025)
- A Saúde Única no enfrentamento da resistência bacteriana a antibióticos no âmbito da agropecuária (Silva; Luiza; Bermudez; Schneider, 2025)
- Guloseimas para Animais de Estimação: Comensalidade, Afeto e Antropomorfismo (Barbosa Osório, Andréa, 2019)

Os resultados mostraram que o sistema retornou artigos com forte relevância temática, mesmo quando as palavras exatas não coincidiam, demonstrando a efetividade da abordagem vetorial.

O tempo médio de resposta para uma busca completa ficou em torno de **15 a 39** millisegundos, dependendo do tamanho do índice vetorial.

4. RESULTADOS E DISCUSSÕES

Os experimentos confirmaram que a busca vetorial oferece resultados semanticamente mais precisos do que a busca textual tradicional. Em termos de desempenho, o sistema alcançou um tempo médio de resposta entre 15 e 39 milissegundos para buscas completas,

validando a otimização de cache e arquitetura proposta. A inclusão do resumo gerado automaticamente melhorou a precisão, pois o vetor representativo do artigo capturou de forma mais fiel o seu conteúdo geral.

Entre as principais vantagens observadas:

- Capacidade de identificar artigos conceitualmente semelhantes;
- Redução de falsos negativos (artigos relevantes que não seriam encontrados por busca exata);
- Estrutura modular e escalável, facilitando a manutenção.

Entretanto, algumas limitações foram identificadas:

- Dependência do modelo de embedding para o idioma português;
- Custo computacional durante a indexação;
- Possível necessidade de re-indexação com modelos mais robustos (como BERT ou Sentence Transformers).

5. CONSIDERAÇÕES SOBRE ESCALABILIDADE E MELHORIAS FUTURAS

O sistema foi projetado com foco em escalabilidade horizontal, podendo distribuir o processamento entre múltiplos consumidores Kafka e nós *Qdrant*.

Como trabalhos futuros, destacam-se:

- A substituição do *Word2Vec* por modelos de *embeddings* mais recentes e pagos, como *SBERT* ou *OpenAI Embeddings*;
- A implantação em ambientes de nuvem com balanceamento de carga;
- O suporte multilíngue para artigos em inglês e espanhol;
- A implementação de sistemas de monitoramento para identificar possíveis gargalos ou falhas.
- A Implementação de um sistema de *logs* para monitorar o comportamento interno.
- Implementar leitura em lotes de mensagens do kafka e envio em lote para o modelo de geração de resumos, para otimizar as escritas no *Qdrant* e as chamadas ao Llama.
- Implementação de uma camada de correção textual no módulo de busca semântica, para melhorar os resultados. ex: “alimntacao de animal domesticos” ser transformado

em “alimentação de animais domésticos”

- Implementar a extração de metadados com o modelo ollama2. Utilizar um *prompt* mais robusto para extrair e categorizar informações dos textos, para melhorar o contexto e a qualidade das buscas.

6. CONCLUSÃO

O desenvolvimento da arquitetura proposta demonstrou a viabilidade e a eficiência do uso de busca vetorial em sistemas destinados à recuperação de artigos científicos. A adoção de modelos de embedding semântico, aliados a uma infraestrutura modular, possibilita resultados mais precisos e contextualmente relevantes quando comparados a abordagens tradicionais baseadas em palavras-chave.

A utilização do *Qdrant* como banco vetorial se mostrou estratégica, devido à sua performance otimizada para consultas de alta dimensionalidade, suporte nativo a filtragens híbridas e integração simplificada com linguagens modernas. Essas características permitiram que o sistema alcançasse uma escalabilidade consistente, sem comprometer a acurácia dos resultados.

O módulo de processamento e indexação foi construído de maneira flexível, permitindo não apenas a inclusão de novos artigos, mas também a extração de diferentes tipos de informações sem a necessidade de modificações estruturais significativas. Esse design modular garante a adaptabilidade da solução para cenários futuros, como classificação automática de artigos, análise de autoria ou agrupamento temático.

Durante os experimentos de validação, observou-se que a arquitetura é capaz de identificar relações semânticas entre textos, mesmo quando estes não compartilham termos literais. Isso reforça a eficiência da abordagem vetorial para aplicações de busca semântica em repositórios acadêmicos, contribuindo para uma experiência de pesquisa mais inteligente e contextualizada.

Em síntese, o trabalho confirma que a aplicação de arquiteturas baseadas em *embeddings* vetoriais constitui uma solução promissora e extensível para melhorar a qualidade das buscas em repositórios científicos, contribuindo significativamente para o avanço da pesquisa e disseminação do conhecimento.

REFERÊNCIAS

LINS, Marcus Vinícius. Word Embeddings: representação vetorial de textos para Machine Learning. Medium, 2020. Disponível em: https://tail-ufpb.medium.com/word-embeddings-representa%C3%A7%C3%A3o-vetorial-de-textos-para-machine-learning-74a227e18478. Acesso em: 20 jun. 2025.

O'MARA, Pete. Embeddings in Production, or How Nothing Scales Like You'd Expect It To (Part 1: Costs to Embed). Barnacle Labs, 2022. Disponível em: https://medium.com/barnacle-labs/embeddings-in-production-or-how-nothing-scales-like-youd-expect-it-to-part-1-costs-to-embed-a82482765215. Acesso em: 20 jun. 2025.

MIKOLOV, Tomas; CHEN, Kai; CORRADO, Greg; DEAN, Jeffrey. Efficient Estimation of Word Representations in Vector Space. arXiv preprint arXiv:1301.3781, 2013. Disponível em: https://arxiv.org/abs/1301.3781. Acesso em: 12 out. 2025.

REIMERS, Nils; GUREVYCH, Iryna. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In: Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing. 2019. Disponível em: https://arxiv.org/abs/1908.10084. Acesso em: 12 out. 2025.

QDRANT. Vector Search Engine for the Next Generation of AI Applications. Qdrant Documentation, 2025. Disponível em: https://qdrant.tech/documentation/. Acesso em: 12 out. 2025.

MONGODB INC. MongoDB Documentation. MongoDB, 2025. Disponível em: https://www.mongodb.com/docs/. Acesso em: 12 out. 2025.

RAMÍREZ, Sebastián. FastAPI Documentation. FastAPI, 2025. Disponível em: https://fastapi.tiangolo.com/. Acesso em: 12 out. 2025.

SALTON, Gerard; WONG, A.; YANG, C. S. A vector space model for automatic indexing. Communications of the ACM, v. 18, n. 11, p. 613–620, 1975. DOI: 10.1145/361219.361220.

DENADAI, Ricardo. WordEmbeddingPortugues: repositório contendo implementações e modelos prontos para utilização em projetos de língua portuguesa (pt-BR). GitHub, 2025. Disponível em: https://github.com/rdenadai/WordEmbeddingPortugues. Acesso em: 12 out. 2025.

RICHARD, Neal; FORD, Mark. Arquitetura de software: as partes difíceis. Rio de Janeiro: Alta Books, 2020. 384 p.

RICHARD, Neal; FORD, Mark. Fundamentos da arquitetura de software. Rio de Janeiro: Alta Books, 2019. 304 p.

MÜLLER, Hans; THOMAS, Eva. Vector Databases for Semantic Search: Performance and Scalability. Journal of Information Retrieval, v. 23, n. 4, p. 345–367, 2022.

KHRONOS GROUP. ONNX: Open Neural Network Exchange. Khronos Group, 2025. Disponível em: https://onnx.ai/. Acesso em: 12 out. 2025.

Agradecimentos

Agradeço primeiramente a Deus, por me conceder sabedoria, força e saúde durante toda a jornada de desenvolvimento deste trabalho.

Agradeço à minha esposa e à minha família, pelo apoio incondicional, paciência e incentivo em todos os momentos.

À minha orientadora, Prof.ª Ana Carolina Nicolosi da R. Gracioso, pela dedicação, disponibilidade e orientações fundamentais para a realização deste artigo.

Por fim, agradeço à Faculdade de Tecnologia de Presidente Prudente (Fatec) e aos colegas de curso, pelo suporte, compartilhamento de conhecimentos e constante estímulo à pesquisa e à inovação.