
LEIM
LICENCIATURA EM ENGENHARIA INFORMÁTICA E MULTIMÉDIA
UNIDADE CURRICULAR DE PROJETO

Pesquisa Semântica para Registros Criminais



Procura Semântica

Pedro Miguel Tavares da Silva Pato e Silva (49734)

Orientador

Professor Helder Filipe de Oliveira Bastos

Junho, 2025

Resumo

Motivado pelo interesse na área criminal e pela necessidade de melhorar a recuperação de informação em registos criminais, este projeto desenvolveu uma API de pesquisa semântica baseada em processamento de linguagem natural (*NLP*), capaz de interpretar o contexto das consultas.

O sistema foi desenvolvido em *Python*, com uma interface construída em *Flask* e *React*, e encontra-se totalmente contentorizado com *Docker*, garantindo maior portabilidade e escalabilidade.

Após uma fase de testes, selecionou-se o modelo de embeddings *XYZ*, que obteve uma precisão de 95,1% nos testes realizados especificamente para este projeto.

A arquitetura modular, suportada por contentores, permite uma gestão eficiente dos dados, com funcionalidades como seleção de *buckets* e mecanismos de *healthcheck*. Desafios como a transição para *Docker* e a otimização do *chunking* foram ultrapassados de forma iterativa ao longo do desenvolvimento.

Como trabalho futuro, está prevista a realização de *fine-tuning* do modelo, em colaboração com a Polícia Judiciária, para melhorar ainda mais a precisão e a adaptação ao domínio específico dos registos criminais.

Abstract

Motivated by an interest in the criminal domain and the need to improve information retrieval from criminal records, this project developed a semantic search API based on natural language processing (*NLP*), capable of interpreting the context of user queries.

The system was developed in *Python*, featuring an interface built with *Flask* and *React*, and is fully containerized using *Docker*, ensuring greater portability and scalability.

After a testing phase, embedding model selected was *medcpt_article*, which achieved an accuracy of 95.1% in tests conducted specifically for this project.

The container-based modular architecture enables efficient data management, with features such as *bucket* selection and *healthcheck* mechanisms. Challenges such as transitioning to *Docker* and optimizing the *chunking* process were overcome iteratively throughout development.

Future work includes performing *fine-tuning* of the model in collaboration with the Portuguese Criminal Police (*Polícia Judiciária*), aiming to further enhance accuracy and domain adaptation to criminal record data.

Agradecimentos

Agradeço ao Professor Helder Filipe de Oliveira Bastos pelas aulas da unidade curricular de Projeto, que, apesar de diferentes, contribuíram para um excelente aproveitamento das aulas, e pela orientação inestimável e apoio contínuo, fundamentais para o sucesso deste projeto.

Agradeço à Polícia Judiciária, cuja área de atuação motivou este trabalho e forneceu importante contexto, assim como à minha namorada e os meus amigos pela inspiração e discussões técnicas

*Dedico este trabalho a todos os que sempre acreditaram em mim,
especialmente à minha família, namorada e aos Furias, pelo apoio
incondicional ao longo deste percurso.*

Índice

Resumo	i
Abstract	iii
Agradecimentos	v
Índice	ix
Lista de Tabelas	xiii
Lista de Figuras	xv
1 Introdução	1
2 Polícia Judiciária e Pesquisa Semântica	3
2.1 Introdução à Polícia Judiciária	3
2.2 Registos Criminais na PJ	3
2.3 Desafios na Pesquisa de Registros	4
2.4 Pesquisa Semântica	4
2.4.1 Introdução à Pesquisa Semântica	4
2.4.2 Modelos de Embeddings usados	5
2.4.3 Cálculo de Similaridade	6
2.5 FAISS	7
2.5.1 Introdução ao FAISS	7
2.5.2 Relevância na Pesquisa Semântica	7
2.5.3 Aplicação nos Registos Criminais	7
2.6 Ollama	7
2.6.1 Introdução ao Ollama	7

2.6.2	Importância na Arquitetura do Sistema	8
3	Trabalho Relacionado	9
4	Modelo Proposto	13
4.1	Requisitos	13
4.2	Fundamentos	14
4.3	Abordagem	14
4.4	Apoio de Modelos Generativos	15
4.4.1	Criação do Script <code>load_models.sh</code>	15
4.4.2	Auxílio na Criação do Script <code>data.py</code>	15
4.4.3	Tradução de Dados	15
5	Implementação do Modelo	17
5.1	Arquitetura	17
5.1.1	Arquitetura da Interface	18
5.1.2	Arquitetura dos Buckets	19
5.2	Interface	23
6	Validação e Testes	25
6.1	Visão Geral do Processo de Testes	25
6.2	Corpus, Script e Consultas	25
6.3	Consultas Utilizadas	27
6.3.1	Verificação de Robustez a Erros Ortográficos	28
6.3.2	Consultas Não Relacionadas	28
6.4	Gráficos de Desempenho por Modelo	29
6.4.1	ACGE-Text	29
6.4.2	All-MiniLM	32
6.4.3	BGE-Large	33
6.4.4	E5-Large	35
6.4.5	Granite	38
6.4.6	Jina-Embeddings-V2	39
6.4.7	MxBAI-Large	41
6.4.8	Nomic-Embed	44
6.4.9	Qwen3-0.6B	45
6.4.10	Qwen3-8B	47

6.4.11	Snowflake-Arctic	50
6.5	Comparação de Modelos	51
6.5.1	Gráficos de Barras - Similaridade, Tempo de Consulta e Processamento	52
6.5.2	Gráfico de Dispersão	54
6.5.3	Análise por Quadrantes	55
6.5.4	Consensus dos Modelos	56
6.6	Conclusões Gerais e Escolhas de Modelos	58
6.6.1	Análise Geral	58
6.6.2	Escolha do Modelo	59
6.6.3	Escolha do Tamanho de Chunk	60
6.6.4	Justificativa da Escolha	60
6.6.5	Considerações Finais	61
6.7	Exemplos de Uso da Interface	61
7	Conclusões e Trabalho Futuro	69
7.1	Conclusão	69
7.2	Principais Conclusões	70
7.3	Trabalho Futuro	70
A	Gestão de Código e Controlo de Versões	71
B	Exemplo de Documento de Teste	73
C	Documentação	75
C.1	Guia de Utilização	75
C.1.1	Instruções do README	75
C.1.2	Operações Avançadas	76
C.1.3	Testes	77

Lista de Tabelas

4.1	Requisitos Funcionais	13
4.2	Requisitos Não Funcionais	14
6.1	Tabela de Consenso para Chunk Size 200	56
6.2	Tabela de Consenso para Chunk Size 300	57
6.3	Tabela de Consenso para Chunk Size 500	57

Listas de Figuras

3.1	Arquitetura do Elasticsearch com extensões de NLP.	10
3.2	Exemplo de pesquisa semântica com BioBERT em relatórios médicos	11
3.3	Interface do sistema ROSS Intelligence para pesquisa jurídica .	11
5.1	Arquitetura geral da API de pesquisa semântica.	18
5.2	Detalhamento da arquitetura da interface em React e inte- gração com o AI Node.	18
5.3	Arquitetura dos serviços do tipo bucket, com processamento e armazenamento independente.	19
5.4	Buckets à espera do carregamento do modelo Ollama.	21
5.5	Output do carregamento do modelo Ollama.	21
6.1	Similaridade K3 - ACGE-Text	30
6.2	Tempo de processamento - ACGE-Text	30
6.3	Tempo de consulta K3 - ACGE-Text	31
6.4	Similaridade K3 - All-MiniLM	32
6.5	Tempo de processamento - All-MiniLM	33
6.6	Tempo de consulta K3 - All-MiniLM	33
6.7	Similaridade K3 - BGE-Large	34
6.8	Tempo de processamento - BGE-Large	34
6.9	Tempo de consulta K3 - BGE-Large	35
6.10	Similaridade K3 - E5-Large	36
6.11	Tempo de processamento - E5-Large	36
6.12	Tempo de consulta K3 - E5-Large	37
6.13	Similaridade K3 - Granite	38
6.14	Tempo de processamento - Granite	39

6.15	Tempo de consulta K3 - Granite	39
6.16	Similaridade K3 - Jina-Embeddings-V2	40
6.17	Tempo de processamento - Jina-Embeddings-V2	40
6.18	Tempo de consulta K3 - Jina-Embeddings-V2	41
6.19	Similaridade K3 - Qwen3-0.6B	42
6.20	Tempo de processamento - Qwen3-0.6B	42
6.21	Tempo de consulta K3 - MxBAI-Large	43
6.22	Similaridade K3 - Nomic-Embed	44
6.23	Tempo de processamento - MxBAI-Large	45
6.24	Tempo de consulta K3 - Nomic-Embed	45
6.25	Similaridade K3 - Qwen3-0.6B	46
6.26	Tempo de processamento - Qwen3-0.6B	46
6.27	Tempo de consulta K3 - Qwen3-0.6B	47
6.28	Similaridade K3 - Qwen3-8B	48
6.29	Tempo de processamento - Qwen3-8B	48
6.30	Tempo de consulta K3 - Qwen3-8B	49
6.31	Similaridade K3 - Snowflake-Arctic	50
6.32	Tempo de processamento - Snowflake-Arctic	51
6.33	Tempo de consulta K3 - Snowflake-Arctic	51
6.34	Comparação da similaridade média (K3) entre os modelos.	52
6.35	Comparação do tempo de consulta K3 entre os modelos.	53
6.36	Comparação do tempo de processamento de documentos entre os modelos.	54
6.37	Gráfico de Similaridade vs Tempo.	55
6.38	Tela da interface.	62
6.39	Resultado de uma consulta.	63
6.40	Consulta num só <i>bucket</i> com $k = 3$	64
6.41	Consulta num só <i>bucket</i> com $k = 12$	65
6.42	Resultados da consulta num só <i>bucket</i> com $k = 12$	66
6.43	Demonstração de um <i>bucket</i> que falhou.	66
6.44	Seleção do <i>bucket</i> que falhou agora indisponível.	67

Capítulo 1

Introdução

A recuperação de informação em registos criminais é um desafio crítico para sistemas judiciais e de segurança pública, devido à complexidade e o volume dos dados existentes. As abordagens tradicionais baseadas em palavras-chave frequentemente falham em captar o contexto semântico das consultas, o que leva a resultados irrelevantes ou incompletos.

Este projeto, impulsionado por um interesse pessoal na área criminal e pela necessidade de modernizar sistemas de pesquisa, desenvolveu uma API de pesquisa semântica, uma interface dedicada à consulta de relatórios criminais e um sistema de teste de modelos.

A solução recorre a técnicas de processamento de linguagem natural (*NLP*) para interpretar consultas em linguagem natural e recuperar documentos com base na sua similaridade semântica.

A implementação foi realizada com recurso a *Python*[3], *Flask*[2], *React*[4], *Ollama*[13] e *Docker*[1]. Utilizei o modelo *nomic-embed-text* que foi selecionado após testes comparativos com outras dez alternativas.

A arquitetura da API é baseada em microserviços, um modelo onde cada componente (como os buckets ou a interface) operam de forma independente, comunicando via interfaces bem definidas, o que facilita escalabilidade e manutenção.

Os utilizadores podem selecionar qualquer bucket para consulta, sendo estes geridos de forma dinâmica pela aplicação. A API devolve os k resultados mais relevantes, suportada por mecanismos de *healthcheck* que asseguram a robustez e disponibilidade dos serviços.

Este projeto apresentou desafios como a transição para Docker, optimização de chunking e resolução de problemas como resultados duplicados ou incompletos. Esses serão abordados no capítulo 5 e 6.

Do ponto de vista da engenharia, o projeto seguiu uma abordagem iterativa, com foco na modularidade, escalabilidade e validação rigorosa. Testes extensivos avaliaram métricas como similaridade, tempo de consulta e tempo de processamento de documentos.

Este trabalho contribui para a modernização de sistemas judiciais, com potencial aplicação em outros domínios, como relatórios médicos ou jornalísticos.

Este relatório está organizado da seguinte forma:

- **Capítulo 2 - Trabalho Relacionado:** Contextualiza o projeto em relação a sistemas de busca semântica e gestão de registos criminais.
- **Capítulo 3 - Modelo Proposto:** Detalha os requisitos, fundamentos tecnológicos e abordagem metodológica.
- **Capítulo 4 - Implementação do Modelo:** Descreve a arquitetura, componentes e desafios técnicos.
- **Capítulo 5 - Validação e Testes:** Apresenta os resultados dos testes e análises comparativas.
- **Capítulo 6 - Conclusões e Trabalho Futuro:** Resume os resultados e sugere direções futuras.

Capítulo 2

Polícia Judiciária e Pesquisa Semântica

2.1 Introdução à Polícia Judiciária

A Polícia Judiciária (PJ), criada em 1945 pelo Decreto-Lei n.º 35042, é o órgão superior de polícia criminal em Portugal, subordinado ao Ministério da Justiça. Investiga crimes graves, como crime organizado, terrorismo, tráfico de drogas, corrupção e crimes financeiros. A sua estrutura inclui diretorias regionais e unidades especializadas, como a Unidade Nacional de Combate ao Cibercrime e Criminalidade Tecnológica e o Laboratório de Polícia Científica.

2.2 Registros Criminais na PJ

Os registos criminais em Portugal, geridos pela Direção-Geral da Administração da Justiça (DG AJ), contêm informações sobre condenações de indivíduos maiores de 16 anos. A PJ acede a bases de dados internas com relatórios detalhados, evidências e perfis de suspeitos, essenciais para investigações. Esses registos, que abrangem crimes complexos como fraude financeira, incluem relatórios textuais, documentos financeiros e evidências forenses.

2.3 Desafios na Pesquisa de Registros

Os sistemas tradicionais de pesquisa baseados em palavras-chave apresentam limitações para a PJ:

- **Dependência de Palavras-Chave:** exige termos exatos, o que dificulta procura em documentos extensos.
- **Falta de Contexto:** não comprehende o significado semântico, retornando resultados irrelevantes.
- **Volume de Dados:** grandes quantidades de dados digitais sobrecarregam as pesquisas tradicionais.
- **Dados Interconectados:** casos exigem compreender relações entre documentos.

2.4 Pesquisa Semântica

2.4.1 Introdução à Pesquisa Semântica

A pesquisa semântica é uma técnica avançada que visa compreender a intenção e o significado contextual de uma consulta, indo além da correspondência exata de palavras-chave. Utiliza processamento de linguagem natural (NLP) e inteligência artificial(AI), interpreta a semântica das palavras, recupera resultados conceptualmente relevantes, sem termos idênticos. Por exemplo, uma consulta sobre crimes que envolvem computadores pode corresponder a documentos sobre "cibercrime", porque reconhece a similaridade conceptual.

Essa abordagem é valiosa em domínios com dados complexos, como registros criminais, onde documentos extensos e jargões específicos dificultam pesquisas tradicionais. A pesquisa semântica usa *embeddings* representações numéricas de texto que capturam significado para medir a similaridade entre as consultas e os documentos.

A API está preparada para lidar com erros ortográficos como "actividades" em vez de "atividades" mantendo os resultados relevantes ao focar no conteúdo semântico em vez da forma literal. Esse embedding é calculado por um modelo de AI

2.4.2 Modelos de Embeddings usados

Foram utilizados e avaliados diversos modelos de geração de embeddings, todos executados através da plataforma Ollama[13] com o objetivo de encontrar o mais adequado:

- **nomic-embed-text**: Codificador de texto com grande capacidade de contexto, superando os modelos text-embedding-ada-002 e text-embedding-3-small da OpenAI em tarefas de contexto curto e longo. Desenvolvido especificamente para geração de embeddings. Disponível na Ollama (<https://ollama.com/library/nomic-embed-text>).
- **mxbai-embed-large**: Modelo de embeddings de última geração da mixedbread.ai, com desempenho de topo no benchmark MTEB e boa generalização em diferentes domínios e comprimentos de texto. Disponível na Ollama (<https://ollama.com/library/mxbai-embed-large>).
- **all-minilm**: Treinado em grandes conjuntos de frases usando aprendizagem contrastiva, concebido para recuperação de informação de alta qualidade. Disponível na Ollama (<https://ollama.com/library/all-minilm>).
- **snowflake-arctic-embed2**: Modelo avançado da Snowflake com suporte multilingue, mantendo desempenho elevado em inglês e escalabilidade. Ideal para tarefas de recuperação semântica. Disponível na Ollama (<https://ollama.com/library/snowflake-arctic-embed2>).
- **bge-large**: Modelo da série BGE da BAAI, mapeando textos para vetores, com suporte multifuncional e multilingue. Disponível na Ollama (<https://ollama.com/library/bge-large>).
- **granite-embedding**: Modelos abertos da IBM para inteligência de código, focados em embeddings densos de texto, disponíveis em versões em inglês e multilingue. Disponível na Ollama (<https://ollama.com/library/granite-embedding>).
- **unclemusclez/jina-embeddings-v2-base-code**: Modelo multilingue que suporta inglês e 30 linguagens de programação, com comprimento de sequência até 8192, baseado em JinaBert com ALiBi. Disponível na Ollama (<https://ollama.com/unclemusclez/jina-embeddings-v2-base-code>).

- **chevalblanc/acge_text_embedding:** Concebido para gerar representações vectoriais para pesquisa e recuperação semântica. Disponível na Ollama (https://ollama.com/chevalblanc/acge_text_embedding).
- **jeffh/intfloat-multilingual-e5-large-instruct:f32:** Modelo E5 multilingue da intfloat, adequado para pesquisa e recuperação semântica em vários idiomas. Disponível na Ollama (<https://ollama.com/jeffh/intfloat-multilingual-e5-large-instruct>).
- **dengcao/Qwen3-Embedding-8B:Q8_0:** Da série Qwen3 da Alibaba, suporta mais de 100 idiomas e contexto até 32k tokens, ideal para tarefas de embedding e ranking de texto. Disponível na Ollama (<https://ollama.com/dengcao/Qwen3-Embedding-8B>).
- **dengcao/Qwen3-Embedding-0.6B:F16:** Modelo com 0,6 mil milhões de parâmetros da série Qwen3, quantizado em 16 bits para equilíbrio entre desempenho e uso de memória. Disponível na Ollama (<https://ollama.com/dengcao/Qwen3-Embedding-0.6B>).

Uma avaliação desses modelos pode ser encontrada no capítulo 6.

2.4.3 Cálculo de Similaridade

A similaridade entre a consulta e os embeddings dos documentos é calculada com a similaridade de cosseno, que mede o cosseno do ângulo entre dois vetores. A fórmula é:

$$\text{similaridade} = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|}$$

onde \mathbf{A} é o vetor de embedding da consulta e \mathbf{B} é o vetor de embedding do documento.

Por exemplo, se uma consulta como “crimes cibernéticos” tiver o embedding $\mathbf{A} = [1, 0]$ e um documento com “segurança online” tiver $\mathbf{B} = [0.5, 0.5]$, o cálculo é:

$$\text{similaridade} = \frac{1 \cdot 0.5 + 0 \cdot 0.5}{\sqrt{1^2 + 0^2} \cdot \sqrt{0.5^2 + 0.5^2}} = \frac{0.5}{1 \cdot \sqrt{0.5}} \approx 0.707$$

Este valor indica uma similaridade moderada. A comparação final é feita considerando o chunk com a maior pontuação de similaridade em cada documento e não uma média de todos os chunks.

2.5 FAISS

2.5.1 Introdução ao FAISS

FAISS (*Facebook AI Similarity Search*) é uma biblioteca de código aberto desenvolvida pelo Facebook AI Research para busca eficiente de similaridade em grandes conjuntos de vetores densos, como *embeddings* de modelos de linguagem. Suporta algoritmos de indexação otimizados (e.g., IVFFlat, HNSW) para velocidade e escalabilidade em tarefas de pesquisa semântica.

2.5.2 Relevância na Pesquisa Semântica

FAISS permite recuperar rapidamente documentos similares a uma consulta, ao comparar *embeddings* em grandes *datasets*. Isto é crucial para processar registos criminais da Polícia Judiciária, onde procuras tradicionais seriam lentas e ineficientes.

2.5.3 Aplicação nos Registos Criminais

No projeto, FAISS indexa *embeddings* gerados por modelos como *nomic-embed-text* a partir de segmentos de texto dos registos. A API utiliza esses índices para responder consultas (e.g., "crimes cibernéticos") em tempo real, identificando trechos semanticamente relevantes.

2.6 Ollama

2.6.1 Introdução ao Ollama

Ollama é uma plataforma de código aberto que permite a execução de modelos de linguagem natural (*NLP*). No contexto deste projeto, Ollama é utilizado para gerar *embeddings* com os modelos, a partir de documentos e consultas, uma etapa fundamental para a pesquisa semântica em registos criminais. A plataforma facilita a gestão e a escalabilidade dos modelos, porque garante um ambiente isolado e consistente para a execução de tarefas de *NLP*.

2.6.2 Importância na Arquitetura do Sistema

Ollama desempenha um papel crucial na arquitetura da API de pesquisa semântica, permite que os modelos de *embeddings* sejam executados de forma eficiente e escalável. A sua integração com Docker assegura a portabilidade e a consistência do sistema e facilita a implementação em diferentes ambientes. Esta abordagem é particularmente vantajosa para lidar com grandes volumes de dados, como os registos criminais da Polícia Judiciária, onde a escalabilidade e a robustez são essenciais.

Capítulo 3

Trabalho Relacionado

Neste capítulo abordamos as principais abordagens e tecnologias relacionadas com a pesquisa semântica, com especial foco na sua aplicação a sistemas de recuperação de informação em domínios como o jurídico e o criminal.

Apresentamos as soluções existentes como o Elasticsearch com extensões de NLP, e discutimos as suas limitações quando aplicadas a textos técnicos e extensos, como os registos da Polícia Judiciária.

Esta contextualização serve de base para justificar a escolha da arquitetura e dos modelos de *embeddings* utilizados no projeto, avaliados com o objetivo de oferecer uma alternativa mais eficaz e precisa para este tipo de dados.

A pesquisa semântica tem sido amplamente explorada em sistemas de procura de informação, com aplicações em motores de busca, recomendação de conteúdo e análise documental. No entanto, a sua aplicação a registos criminais continua limitada, devido à complexidade dos dados, à terminologia técnica e à necessidade de contextualização profunda.

O Elasticsearch, uma ferramenta popular de busca textual, permite a integração de extensões baseadas em processamento de linguagem natural (NLP), tais como análise de entidades, vetorização de documentos e ranking semântico.

Estas extensões aumentam a capacidade de recuperação além da simples correspondência de palavras-chave.

Ainda assim, mesmo com essas integrações, o Elasticsearch não é ideal para textos longos e jurídicos, pois não foi concebido para capturar relações semânticas profundas nem lidar com ambiguidade contextual, algo crucial na

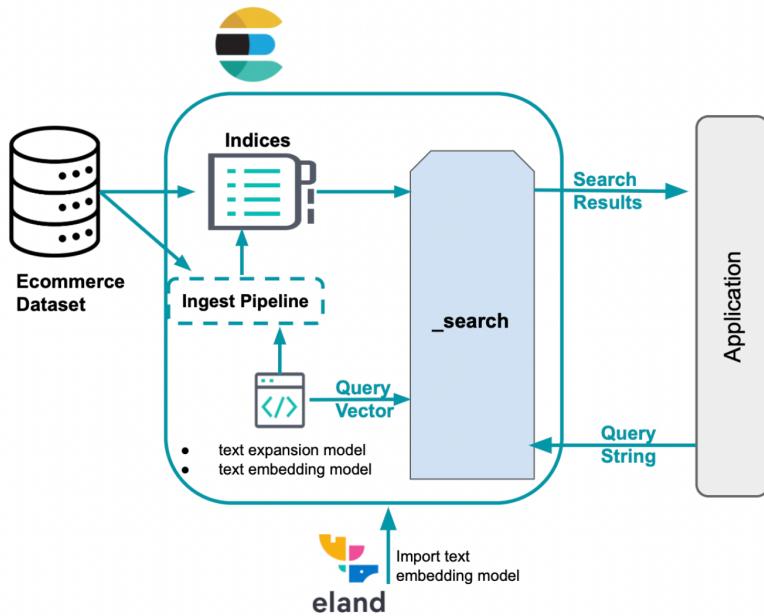


Figura 3.1: Arquitetura do Elasticsearch com extensões de NLP.

análise de relatórios criminais.

Aplicações de Pesquisa Semântica

Apesar de o foco deste projeto ser a recuperação de informação em registos criminais, técnicas de pesquisa semântica têm sido amplamente adotadas noutras áreas com dados complexos e linguagem técnica.

Na área da saúde, por exemplo, a pesquisa semântica é utilizada para interpretar consultas clínicas e recuperar documentos médicos relevantes, mesmo quando não há correspondência exata de termos. Modelos como o BioBERT são treinados em textos biomédicos e aplicados para apoiar o diagnóstico médico [5].

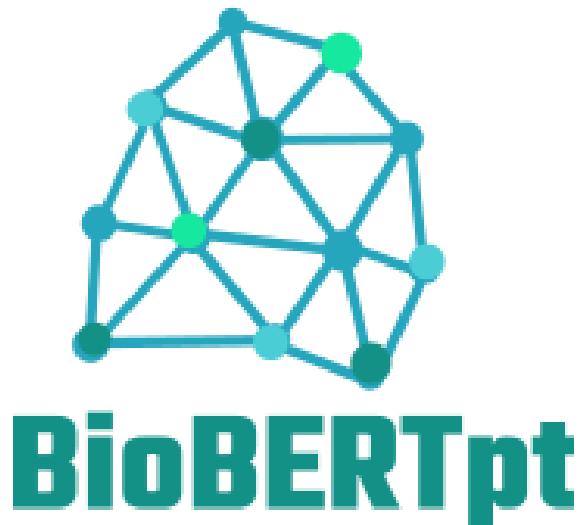


Figura 3.2: Exemplo de pesquisa semântica com BioBERT em relatórios médicos

No setor jurídico, ferramentas como o ROSS Intelligence utilizam modelos baseados em linguagem natural para analisar consultas jurídicas e devolver jurisprudência relevante, mesmo em casos com vocabulário técnico variado. Esta abordagem facilita o trabalho de advogados ao permitir buscas mais inteligentes em bases legais [15].

ROSS

Figura 3.3: Interface do sistema ROSS Intelligence para pesquisa jurídica

Em ambientes corporativos, soluções como o Microsoft Viva Topics recorrem à semântica para identificar temas recorrentes nos documentos internos,

e sugerir conteúdos relacionados, promovendo uma gestão mais eficiente do conhecimento [6].

Estas aplicações reforçam o potencial da abordagem desenvolvida neste projeto para ser generalizada a outras áreas, como saúde, direito ou gestão empresarial, onde a semântica é chave para entender e relacionar grandes volumes de informação textual.

Capítulo 4

Modelo Proposto

Este capítulo apresenta o modelo proposto para a API de pesquisa semântica, delineando os requisitos, fundamentos tecnológicos e a abordagem metodológica, com o objetivo de fornecer uma visão clara dos objetivos do projeto.

4.1 Requisitos

Os requisitos do sistema foram definidos para garantir funcionalidade e robustez, conforme apresentado nas Tabelas 4.1 e 4.2.

Nº	Requisito	Implementado
1	Pesquisa semântica em linguagem natural	Sim
2	Interface de utilizador para consultas	Sim
3	Seleção de <i>buckets</i> para consultas	Sim
4	<i>Healthcheck</i> aos <i>buckets</i>	Sim
5	Feedback sobre resultados	Sim

Tabela 4.1: Requisitos Funcionais

Nº	Requisito	Implementado
1	Escalabilidade para múltiplos <i>buckets</i>	Sim
2	Monitorização de logs	Sim
3	Segurança de dados	Não

Tabela 4.2: Requisitos Não Funcionais

4.2 Fundamentos

A API utiliza técnicas de NLP para processar consultas em linguagem natural, que gera embeddings com o modelo escolhido após testes comparativos. Python e Flask formam o backend, enquanto React suporta a interface de utilizador. Docker permite a execução de buckets independentes, e FAISS é usado para indexação e busca de embeddings. A escolha do modelo final foi feita, com base em métricas como similaridade e tempo de consulta e as tabelas de consenso conforme detalhado no Capítulo 6.

4.3 Abordagem

O desenvolvimento seguiu uma metodologia iterativa:

- **Prototipagem Local:** Inicialmente, o projeto foi desenvolvido localmente para explorar as técnicas de embeddings e de pesquisa semântica.
- **Otimização de Chunking:** Passou-se de chunking baseado em caracteres para palavras, melhorando a relevância dos resultados.
- **Transição para Docker:** Adotou-se uma arquitetura baseada em contentores para escalabilidade e gestão dinâmica.
- **Testes e Validação:** Foram realizados testes para avaliar a precisão dos resultados, o tempo de consulta e a robustez do sistema.
- **Seleção de Modelo:** Onze modelos foram testados, com o modelo final ainda em avaliação com base em métricas como precisão e tempo de consulta.

4.4 Apoio de Modelos Generativos

Modelos generativos, como os baseados em inteligência artificial, foram utilizados neste projeto para auxiliar em tarefas específicas. Estes modelos permitiram a gerar um script shell, a tradução de textos e a superação de barreiras técnicas, contribuindo significativamente para o progresso do trabalho.

4.4.1 Criação do Script `load_models.sh`

Devido à falta de experiência prévia com scripts em Linux, um modelo gerativo foi empregado para criar o script `load_models.sh`. Este script é responsável por puxar e carregar os modelos especificados no arquivo `models.txt` no container Ollama durante a fase de inicialização. O modelo gerativo forneceu um script funcional que foi posteriormente ajustado e testado para garantir sua compatibilidade com o ambiente do projeto.

4.4.2 Auxílio na Criação do Script `data.py`

O script `data.py`, que realiza a leitura e o processamento dos dados utilizados no projeto, também foi desenvolvido com o suporte de um modelo gerativo. O modelo sugeriu abordagens para a leitura eficiente dos dados e ajudou na estruturação do código, assim consegui concentrar-me em aspectos mais críticos da implementação.

4.4.3 Tradução de Dados

Além disso, o modelo gerativo foi utilizado para traduzir documentos e dados fornecidos pelo orientador do inglês para o português. Esta tradução foi essencial para garantir que o modelo fosse testado e validado em um ambiente mais fiel ao que foi planeado.

Capítulo 5

Implementação do Modelo

5.1 Arquitetura

A API adota uma arquitetura de microserviços, com Flask para endpoints REST, React para a interface, e Ollama para executar modelos em contenedores Docker. Cada *bucket* é um serviço independente, configurado via `docker-compose.yml`, permitindo a adição de dados sem interrupção.

- **Interface:** Desenvolvida em React, permite a seleção de *buckets* e a submissão de consultas em linguagem natural.
- **Buckets:** Serviços independentes que geram e armazenam embeddings dos documentos, processando consultas em paralelo.
- **AI Node:** Coordena o sistema, regista *buckets*, realiza *healthchecks* e encaminha consultas para processamento.

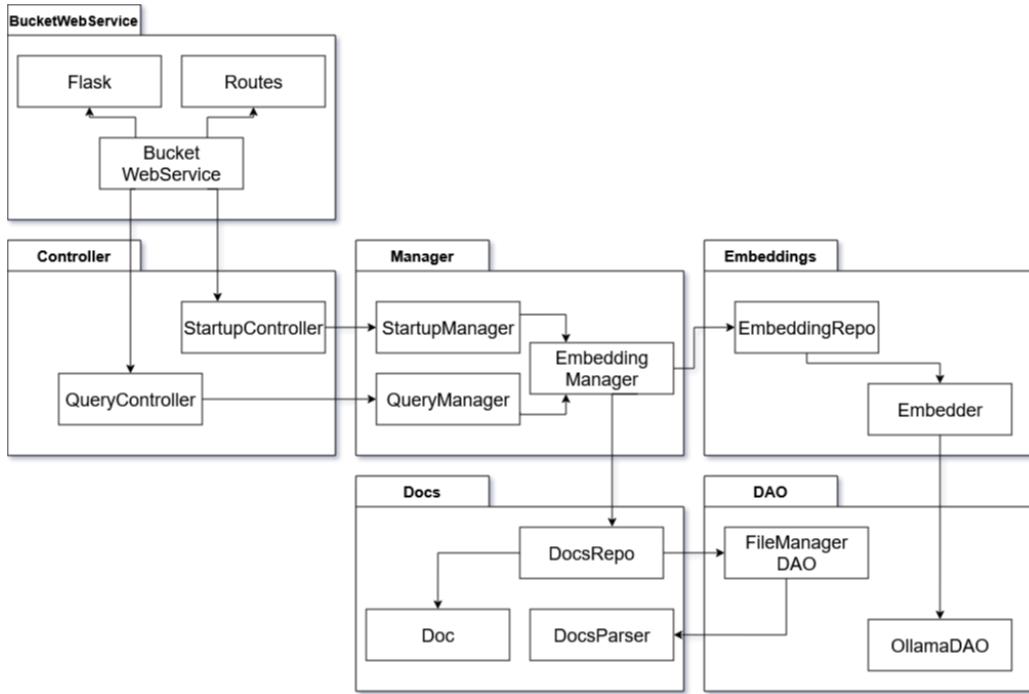


Figura 5.1: Arquitetura geral da API de pesquisa semântica.

5.1.1 Arquitetura da Interface

A interface utiliza componentes React para interação dinâmica, incluindo um seletor de *buckets* e uma área de entrada de texto. O *AI Node* é integrado como ponto central de comunicação, recebendo consultas e retornando respostas processadas, servindo de ponte entre a interface e os *buckets*.

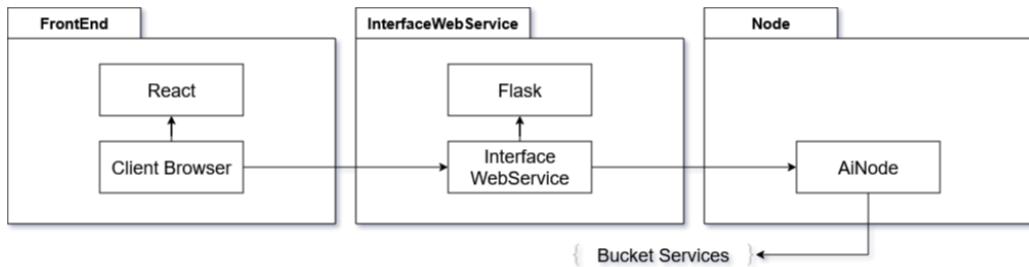


Figura 5.2: Detalhamento da arquitetura da interface em React e integração com o AI Node.

5.1.2 Arquitetura dos Buckets

Os *buckets* são serviços modulares que dividem documentos em *chunks*, geram embeddings com o modelo escolhido e os armazenam para futuras consultas. Cada *bucket* funciona de forma independente, o que permite escalabilidade horizontal.

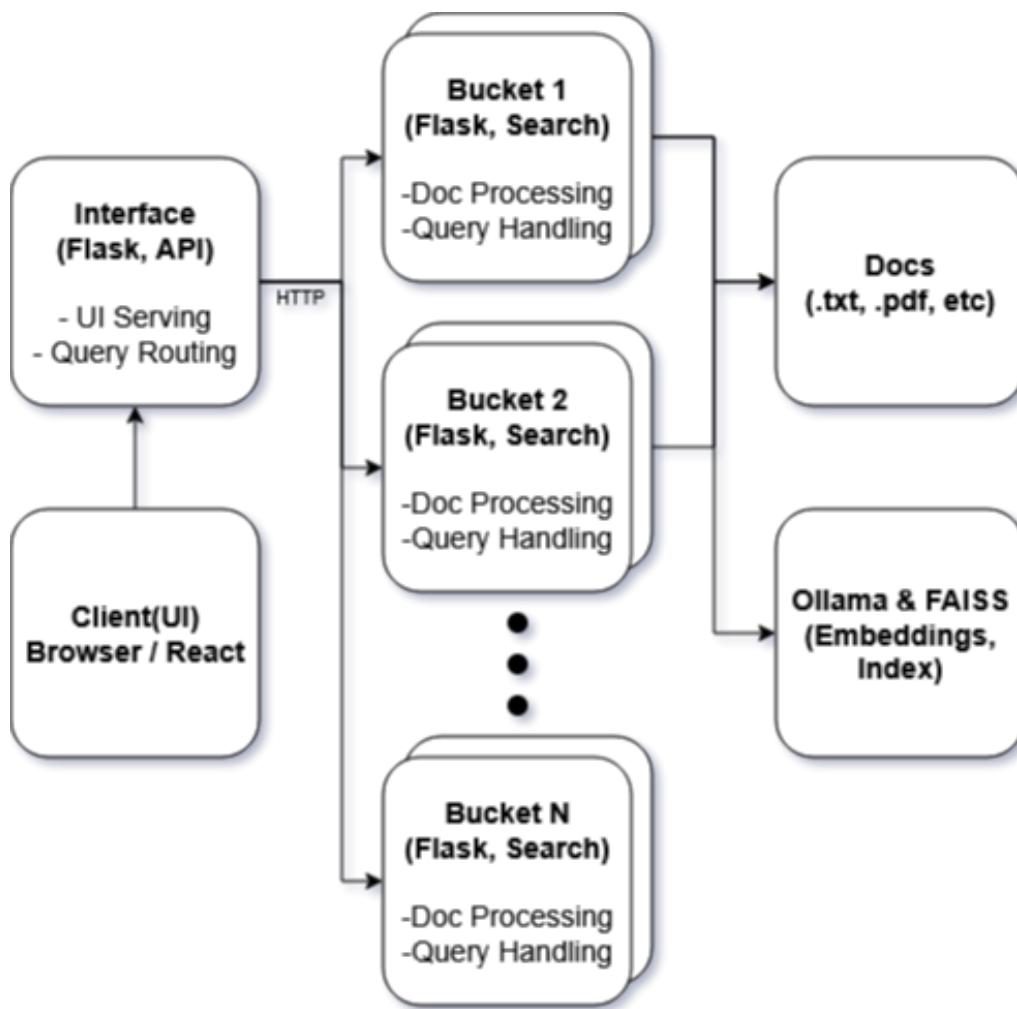


Figura 5.3: Arquitetura dos serviços do tipo bucket, com processamento e armazenamento independente.

O processo de criação e armazenamento de *embeddings*

Este processo envolve duas classes principais: `Embedder.py` e `EmbeddingRepo.py`.

Embedder.py A classe `Embedder` gera *embeddings* a partir de texto utilizando o serviço Ollama.

EmbeddingRepo.py A classe `EmbeddingRepo` lida com o armazenamento e recuperação de *embeddings* utilizando o FAISS, uma biblioteca otimizada para pesquisa de similaridade.

Divisão de Texto: Documentos grandes são divididos em chunks para garantir que os embeddings representam o contexto local de forma eficaz.

Gestão de *Buckets*

Os *buckets* são unidades independentes que processam e armazenam *embeddings* de documentos. A sua gestão envolve registo, processamento de documentos e manipulação de ficheiros.

BucketWebService.py A classe `BucketWebService` gere o serviço web do *bucket* e sua interação com o *AI Node*.

StartUpController.py A classe `StartUpController` processa os documentos no *bucket* durante a inicialização.

Durante a fase de inicialização, enquanto o container Ollama está a puxar e a carregar os modelos especificados no arquivo `models.txt`, os outros buckets esperam até receberem a informação de que o container Ollama está pronto. Isso é alcançado através de um mecanismo de verificação de saúde, onde o container Ollama cria um arquivo de sinalização (`/root/.ollama/models_loaded`) após a conclusão do carregamento dos modelos. Os buckets, por sua vez, utilizam esta verificação de saúde para determinar quando podem começar a processar documentos e gerar embeddings.

FileManagerDAO.py A classe `FileManagerDAO` lida com o acesso a ficheiros dentro dos *buckets*.

Gestão de Consultas

As consultas são processadas para encontrar documentos semanticamente similares com o uso dos *embeddings*.

	Name	Container ID	Image	Port(s)	CPU (%)	Last started	Actions	
□	semanticsearch -	-	-	-	0%	2 seconds ago	█	⋮ ☒
□	● ollama-1	575885c00f0f	semanticsc_11434:11434 🕒	-	0%	2 seconds ago	█	⋮ ☒
□	○ bucket2-1	2826bcad2daa	semanticse	8082:8080	0%		▷	⋮ ☒
□	○ bucket3-1	c61145a5bcc0	semanticse	8083:8080	0%		▷	⋮ ☒
□	○ interface-1	6f9477dd405f	semanticse	8080:8080	0%		▷	⋮ ☒
□	○ bucket4-1	695f789f4ab3	semanticse	8084:8080	0%		▷	⋮ ☒
□	○ bucket1-1	a50f8a3f602b	semanticse	8081:8080	0%		▷	⋮ ☒

Figura 5.4: Buckets à espera do carregamento do modelo Ollama.

```
[Wed Jul 16 01:53:06 UTC 2025] Starting load_models.sh
Starting Ollama server...
Waiting for Ollama server...
Connection to localhost (::1) 11434 port [tcp/*] succeeded!
[Wed Jul 16 01:53:07 UTC 2025] Reading models.txt content:
nomic-embed-text:latest
mxbai-embed-large:latest
all-minilm:latest
snowflake-arctic-embed2:latest
bge-large:latest
granite-embedding:latest
unclemusclez/jina-embeddings-v2-base-code:latest
chevalblanc/acge_text_embedding:latest
jeffh/intfloat-multilingual-e5-large-instruct:f32
dengcao/Qwen3-Embedding-8B:Q8_0
dengcao/Qwen3-Embedding-0.6B:F16[Wed Jul 16 01:53:07 UTC 2025] Added model to array: nomic-embed-text:latest
[Wed Jul 16 01:53:07 UTC 2025] Added model to array: mxbai-embed-large:latest
[Wed Jul 16 01:53:07 UTC 2025] Added model to array: all-minilm:latest
[Wed Jul 16 01:53:07 UTC 2025] Added model to array: snowflake-arctic-embed2:latest
[Wed Jul 16 01:53:07 UTC 2025] Added model to array: bge-large:latest
[Wed Jul 16 01:53:07 UTC 2025] Added model to array: granite-embedding:latest
[Wed Jul 16 01:53:07 UTC 2025] Added model to array: unclemusclez/jina-embeddings-v2-base-code:latest
[Wed Jul 16 01:53:07 UTC 2025] Added model to array: chevalblanc/acge_text_embedding:latest
[Wed Jul 16 01:53:07 UTC 2025] Added model to array: jeffh/intfloat-multilingual-e5-large-instruct:f32
[Wed Jul 16 01:53:07 UTC 2025] Added model to array: dengcao/Qwen3-Embedding-8B:Q8_0
[Wed Jul 16 01:53:07 UTC 2025] Added model to array: dengcao/Qwen3-Embedding-0.6B:F16
[Wed Jul 16 01:53:07 UTC 2025] Total models to process: 11
[Wed Jul 16 01:53:07 UTC 2025] Model in loop: nomic-embed-text:latest
[Wed Jul 16 01:53:07 UTC 2025] Model in loop: mxbai-embed-large:latest
[Wed Jul 16 01:53:07 UTC 2025] Model in loop: all-minilm:latest
[Wed Jul 16 01:53:07 UTC 2025] Model in loop: snowflake-arctic-embed2:latest
[Wed Jul 16 01:53:07 UTC 2025] Model in loop: bge-large:latest
[Wed Jul 16 01:53:07 UTC 2025] Model in loop: granite-embedding:latest
[Wed Jul 16 01:53:07 UTC 2025] Model in loop: unclemusclez/jina-embeddings-v2-base-code:latest
[Wed Jul 16 01:53:07 UTC 2025] Model in loop: chevalblanc/acge_text_embedding:latest
[Wed Jul 16 01:53:07 UTC 2025] Model in loop: jeffh/intfloat-multilingual-e5-large-instruct:f32
[Wed Jul 16 01:53:07 UTC 2025] Model in loop: dengcao/Qwen3-Embedding-8B:Q8_0
[Wed Jul 16 01:53:07 UTC 2025] Model in loop: dengcao/Qwen3-Embedding-0.6B:F16
```

Figura 5.5: Output do carregamento do modelo Ollama.

`QueryController.py` A classe `QueryController` lida com requisições de consulta via endpoint web.

`QueryManager.py` A classe `QueryManager` processa consultas que geram *embeddings* e faz uma pesquisa no repositório.

***Healthcheck* e Comunicação**

O sistema garante confiabilidade através de *healthchecks* e comunicação eficiente entre *buckets* e a interface.

`AINode.py` A classe `AINode` supervisiona a saúde dos *buckets* e encaminha consultas.

OllamaDAO

A classe `OllamaDAO` integra-se com o serviço Ollama para gerar *embeddings*.

Variáveis de Ambiente

- `BUCKET_NAME`: Nome do bucket (padrão: "default_bucket").
- `BUCKET_FOLDER`: Pasta de documentos (padrão: "./documents").
- `AI_NODE_URL`: URL do AI Node (padrão: "http://interface:8080/ai-node").
- `BUCKET_URL`: URL do bucket (padrão: "http://bucket1:8080").
- `OLLAMA_HOST`: Host do serviço Ollama (padrão: "http://localhost:11434").
- `AI_NODE_URL`: URL do AI Node para encaminhamento de consultas.

Componentes Adicionais

- `DocsParser`: Extrai texto de ficheiros (e.g., .pdf, .docx) com as bibliotecas PyPDF2 e prepara o conteúdo para embeddings.
- `DocsRepo`: Integra `FileManagerDAO` e `DocsParser` para procurar e analisar documentos para criar objetos `Doc` com metadados e conteúdo.
- `EmbeddingManager`: Orquestra o processamento de documentos, coordenando `Embedder` e `EmbeddingRepo` para gerar e armazenar embeddings.

- **Interface** (`InterfaceWebService` e `main_interface.py`): Serve um frontend baseado em React e encaminha requisições para buckets via AI Node, proporcionando uma interface amigável ao utilizador.
- **Logging:** Registo abrangente em todos os componentes auxilia no monitoramento, depuração e otimização de desempenho.

O sistema de testes utiliza um script shell, `load_models.sh`, para automatizar o processo de puxar e carregar todos os modelos listados no arquivo `models.txt`. Este script é executado no container Ollama durante a inicialização e garante que todos os modelos necessários estejam disponíveis antes de prosseguir com os testes. Além disso, a configuração do Docker Compose para testes (`docker-compose.test.yml`) estabelece um ambiente isolado com sua própria rede e volumes, permitindo testes independentes sem interferir no ambiente de produção.

5.2 Interface

Durante o desenvolvimento, foram enfrentadas diversas dificuldades técnicas, entre as quais:

1. **Transição para Docker:** A migração da implementação local para contentores exigiu ajustes nas configurações de redes e volumes, resolvidos com o apoio do orientador.
2. **Otimização de *Chunking*:** A abordagem inicial, baseada no número de caracteres, foi substituída por uma segmentação por palavras, o que melhorou a relevância dos resultados.
3. **Filtro de Duplicados:** A filtragem de documentos repetidos limitava o número de resultados. A solução passou por aumentar o número de candidatos retornados pela FAISS antes de aplicar o filtro.

Capítulo 6

Validação e Testes

6.1 Visão Geral do Processo de Testes

O desenvolvimento da API envolveu um processo iterativo de testes. Inicialmente, testamos modelos como o medcpt, que apresentavam altas pontuações de similaridade (cerca de 90%), mas os resultados não eram úteis devido à baixa relevância e baixo contexto semântico dos documentos recuperados. Isso levou a uma reformulação dos testes, priorizando a precisão (K3_precision) para garantir a qualidade semântica. A nova versão dos testes é containerizada, permitindo execução independente com `docker-compose.test`, garantindo consistência e reproduzibilidade. O Ollama foi configurado para usar GPU, melhorando significativamente o desempenho.

6.2 Corpus, Script e Consultas

Os testes foram realizados em um ambiente local com:

- **Processador:** Intel Core i7-13700KF
- **Placa Gráfica:** NVIDIA GeForce RTX 4070 Ti
- **Sistema Operativo:** Windows 11
- **Memória RAM:** 32 GB

O corpus de teste foi constituído por **100 documentos fictícios de registros criminais** em português, criados para simular situações realistas enfrentadas por autoridades judiciais. Os documentos abrangem diversos tipos

de crimes e entidades envolvidas. Um exemplo destes documentos encontra-se no Apêndice B, com o título ”*A extorsão se torna mortal: violenta retenção dos lobos vermelhos sobre as empresas de Vilkor*”.

Foram utilizadas **18 queries** distintas em linguagem natural (ver Seção 6.3) para testar a capacidade e robustez e contexto semântico dos modelos de embeddings.

A avaliação foi feita com base na métrica de **similaridade média K3**, que calcula a média da similaridade entre a query e os três documentos mais relevantes retornados.

Além da similaridade, foram também analisados os tempos médios de **processamento de documentos** (indexação) e de **tempo de consulta K3**, considerando três tamanhos distintos de *chunk*: 200, 300 e 500 palavras.

Scripts de Teste e Análise de Modelos

Os scripts `test_model.py` e `data.py` são fundamentais para a avaliação de modelos de *embedding* em uma API de pesquisa semântica. O `test_model.py` automatiza a geração de dados de desempenho, testando diferentes tamanhos de *chunk* (200, 300 e 500 palavras) e idiomas (português e inglês), enquanto o `data.py` processa esses dados para análise e visualização. Ambos são essenciais para a validação e testes descritos no Capítulo 6, embasando a escolha da configuração ideal.

Funcionalidades

- **Processamento (`test_model.py`):** Lê documentos, divide em *chunks*, gera *embeddings* via Ollama e indexa com FAISS.
- **Consultas (`test_model.py`):** Executa consultas padrão e com ruído (*noisy*), medindo:
 - **Similaridade média K3:** Média da similaridade entre a *query* e os 3 documentos mais relevantes.
 - **Noisy similarity:** Similaridade média para *queries* com ruído (e.g., erros ortográficos).
 - **Avg similarity:** Média entre **K3 similarity** e **noisy similarity**.

- **Tempos:** Tempo de processamento de documentos e tempo de consulta.
- **Saída (test_model.py):** Exporta resultados para JSON.
- **Análise e Visualização (data.py):** Lê os dados JSON, calcula métricas adicionais e gera gráficos de barras, dispersão e tabelas de consenso.

Integração com o Relatório

Os dados e visualizações gerados pelos scripts são utilizados no Capítulo 6, como na Figura 6.37 (Similaridade vs. Tempo) e na escolha dos modelos mais adequados (Seção 6.7).

6.3 Consultas Utilizadas

As consultas foram elaboradas em português, abrangendo tópicos variados e refletindo diferentes tipos de investigações criminais e contextos temáticos:

- “Quais são os relatórios recentes sobre atividades de gangues armadas em Vilkor, Zakovia?”
- “Quais são os relatórios mais recentes sobre roubos de arte envolvendo os Ghost Shadows em Ravenska, Zakovia?”
- “Quais incidentes envolvendo os esquemas de proteção dos Blood Ravens ocorreram perto de Baron’s Peak em 2023?”
- “Quais são os casos de contrabando de armas relatados em Sokovia em 2024?”
- “Quais atividades criminosas dos Night Vipers foram documentadas em Krov, Zakovia?”
- “Quais são as políticas de turismo sustentável em Ravenska, Zakovia?”

6.3.1 Verificação de Robustez a Erros Ortográficos

Para verificar a robustez dos modelos face a erros ortográficos, o script `test_model.py` repetiu as queries com variações propositadamente incorretas (por exemplo, atividdades em vez de atividades). As pontuações de similaridade permaneceram altas, com uma queda média de apenas 0.85 para 0.80, demonstrando boa tolerância semântica.

- “Quais são os relatórios resentes sobre atividades de ganguues armadaz em Vilkorr, Zakovia?”
- “Quais são os relatórios mais recentes sobre roubos de arte envolvendo os Ghoost Shadws em Ravennska, Zakovia?”
- “Quais incidentes envolvendo os esquemas de proteção dos Blod Ravns ocorreram perto de Baroon’s Peak em 2023?”
- “Quais são os casos de contrabbando de armaz relatados em Sokovia em 2024?”
- “Quais atividades criminosas dos Nigt Viipers foram documentadas em Kroov, Zakovia?”
- “Quais são as políticas de turismo sustentável em Ravennska, Zakovia?”

6.3.2 Consultas Não Relacionadas

Foram incluídas consultas não relacionadas ao domínio original para testar a capacidade do modelo de atribuir pontuações baixas a queries irrelevantes e altas a queries relevantes. Esta abordagem foi adotada para avaliar a robustez do modelo em diferenciar contextos temáticos, garantindo que ele priorize a relevância semântica e evite falsos positivos.

- “Qual o melhor país para se viver com um estilo de vida rico, na Europa?”
- “Qual o melhor minério no Minecraft para fazer um conjunto de armadura inteiro?”

- “Quais os melhores temperos para cozinhar um lombo de porco durante 8 horas?”

Cada consulta tem a sua contraparte com erros ortográficos, mantendo a mesma estrutura semântica, para avaliar a robustez do modelo face a erros comuns de digitação.

- “Qual a melhhor pais para se viver com um estilo de vida rico, no europa?”
- “Qual a mellhor minério no Minicraft para fazer um conjuunto de armaduro inteira?”
- “Quaais os melhores tempeeros para cozinhor um lomba de porca durante 8 horas?”

6.4 Gráficos de Desempenho por Modelo

Esta seção apresenta os resultados dos modelos atualizados testados `nomic-embed-text-v2`, `mxbai-embed-large-v1`, `bge-m3`, `multilingual-e5-large`, `all-minilm-16-v2` e `snowflake-arctic-embed-m` em documentos em português, com base nas métricas de similaridade média (K3), tempo de processamento de documentos e tempo de consulta K3, para diferentes tamanhos de chunk (200, 300, 500 palavras). As figuras de cada modelo são apresentadas individualmente, depois da sua análise.

6.4.1 ACGE-Text

O modelo ACGE-Text apresenta similaridade K3 média de 0.683 (200) a 0.669 (500), com leve queda para *chunk sizes* maiores. O tempo de processamento é elevado para *chunk size* 200 (29.08s), reduzindo para 11.73s em 500, devido à menor contagem de *chunks* (5.69 a 2.61). O tempo de consulta K3 é estável (~0.020s), indicando eficiência em consultas.

Comparado ao Nomic-Embed, tem menor similaridade, mas tempos de consulta semelhantes.

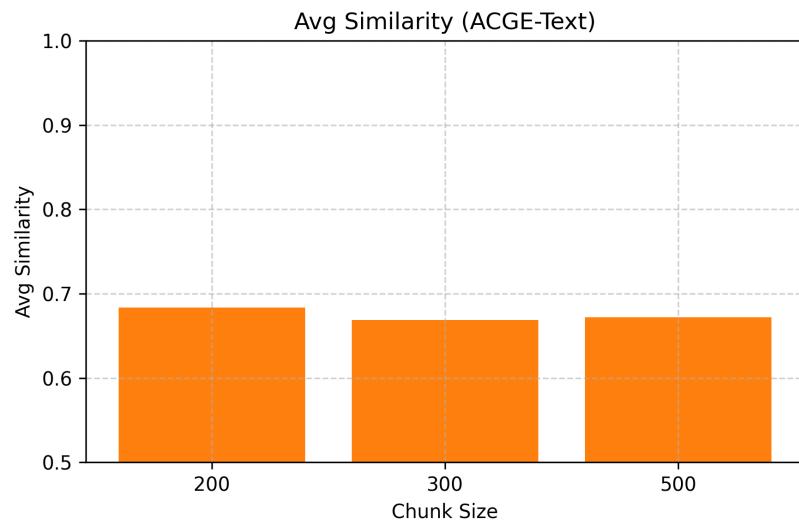


Figura 6.1: Similaridade K3 - ACGE-Text

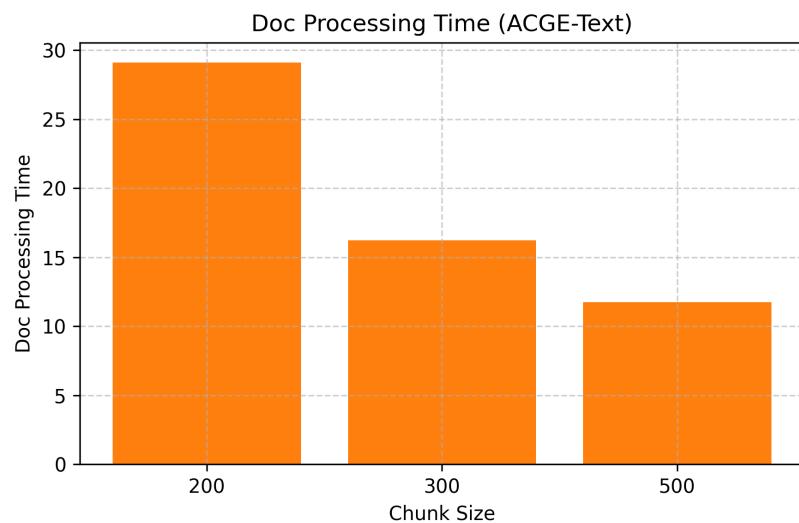


Figura 6.2: Tempo de processamento - ACGE-Text

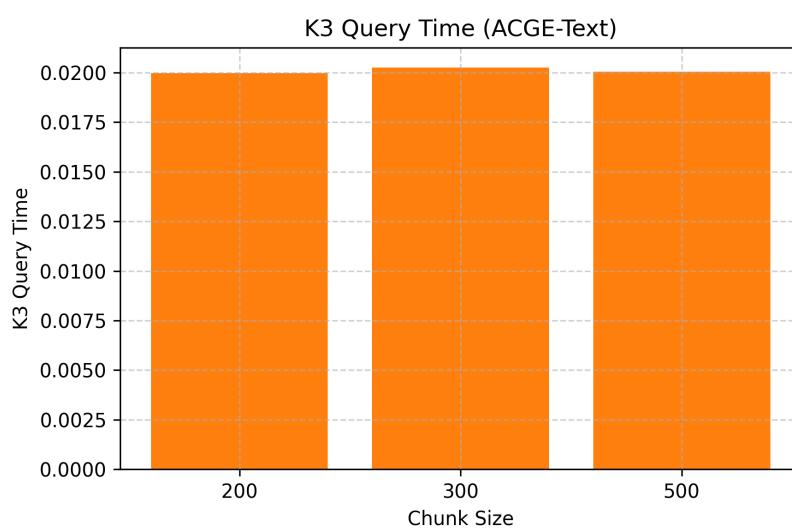


Figura 6.3: Tempo de consulta K3 - ACGE-Text

6.4.2 All-MiniLM

O modelo All-MiniLM exibe similaridade K3 média de 0.611 (200) a 0.596 (500), com desempenho inferior para *chunk sizes* maiores. O tempo de processamento é baixo, de 6.58s (200) a 3.52s (500), refletindo eficiência devido a menos *chunks*. O tempo de consulta K3 ($\sim 0.013s$) é um dos mais rápidos.

Comparado ao **Granite**, tem menor similaridade, mas processamento mais rápido.

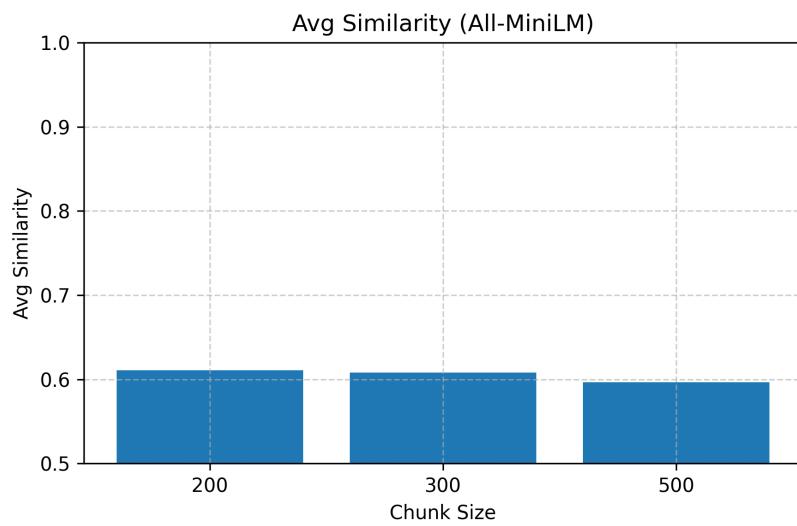


Figura 6.4: Similaridade K3 - All-MiniLM

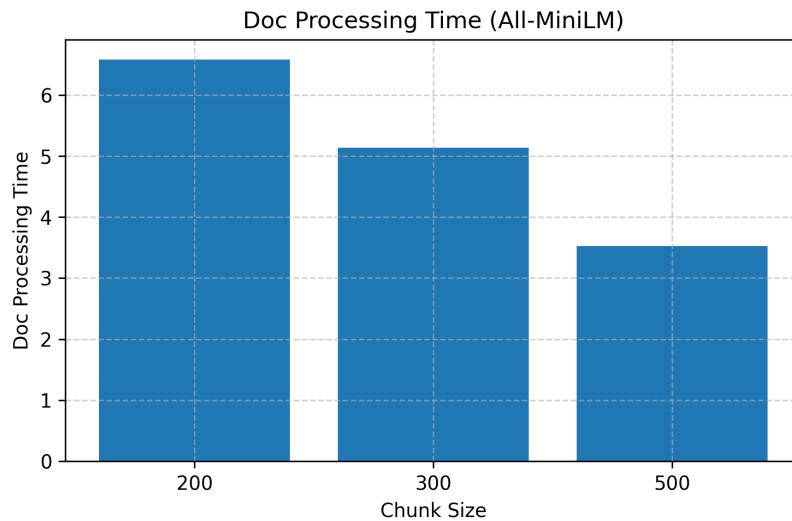


Figura 6.5: Tempo de processamento - All-MiniLM

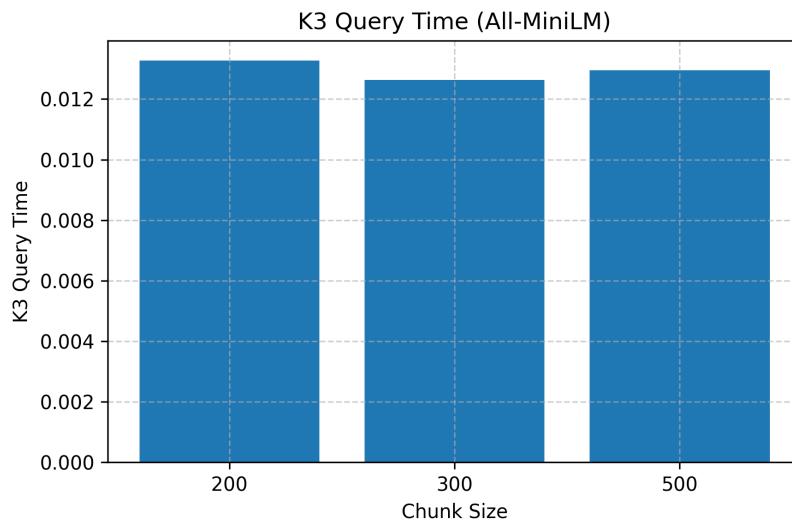


Figura 6.6: Tempo de consulta K3 - All-MiniLM

6.4.3 BGE-Large

O modelo BGE-Large alcança similaridade K3 média de 0.775 (200) a 0.754 (500), com melhor desempenho em *chunk size* 200. O tempo de processamento varia de 21.62s (200) a 7.70s (500), reduzindo com menos *chunks*. O tempo de consulta K3 (~0.020s) é rápido e estável.

Comparado ao E5-Large, tem menor similaridade, mas tempos de consulta mais curtos.

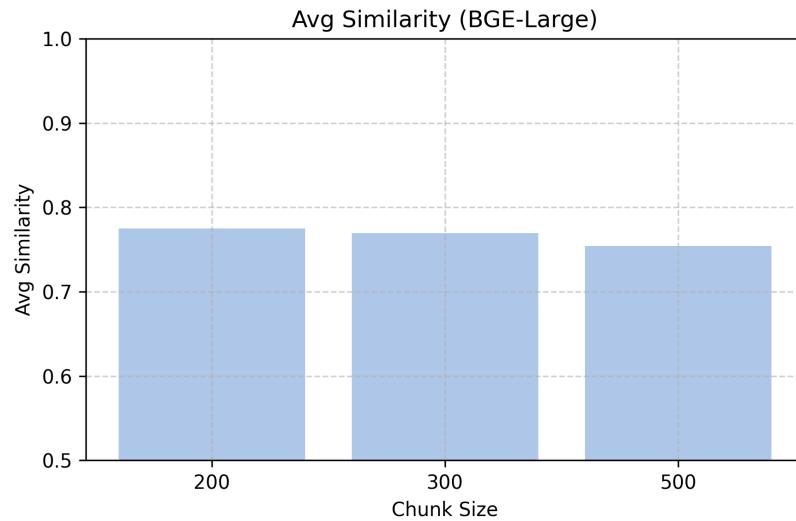


Figura 6.7: Similaridade K3 - BGE-Large

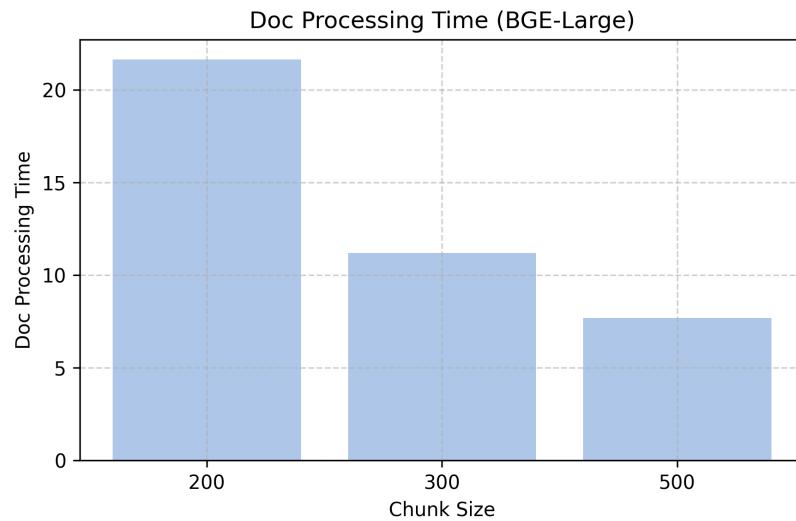


Figura 6.8: Tempo de processamento - BGE-Large

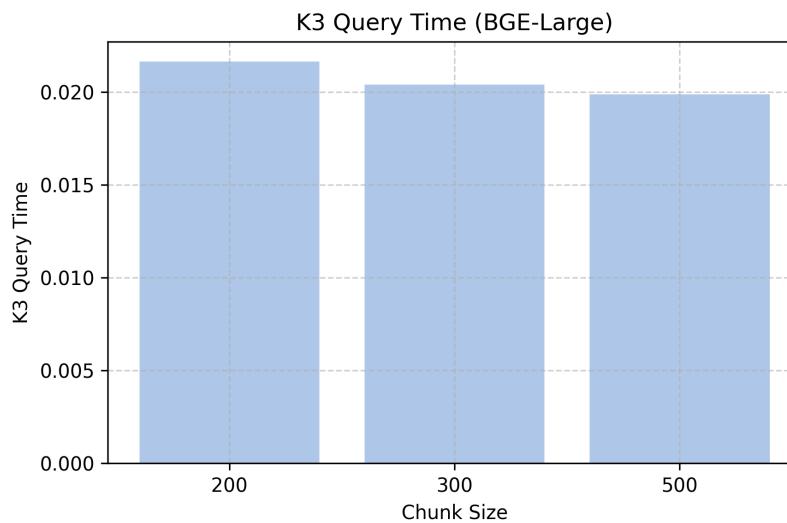


Figura 6.9: Tempo de consulta K3 - BGE-Large

6.4.4 E5-Large

O modelo E5-Large destaca-se com similaridade K3 média de 0.875 (200) a 0.871 (500), mantendo alta precisão. O tempo de processamento é elevado, de 36.39s (200) a 13.86s (500), devido à complexidade do modelo. O tempo de consulta K3 ($\sim 0.048\text{--}0.056\text{s}$) é mais lento.

Comparado ao **Granite**, oferece maior similaridade, mas com maior custo computacional.

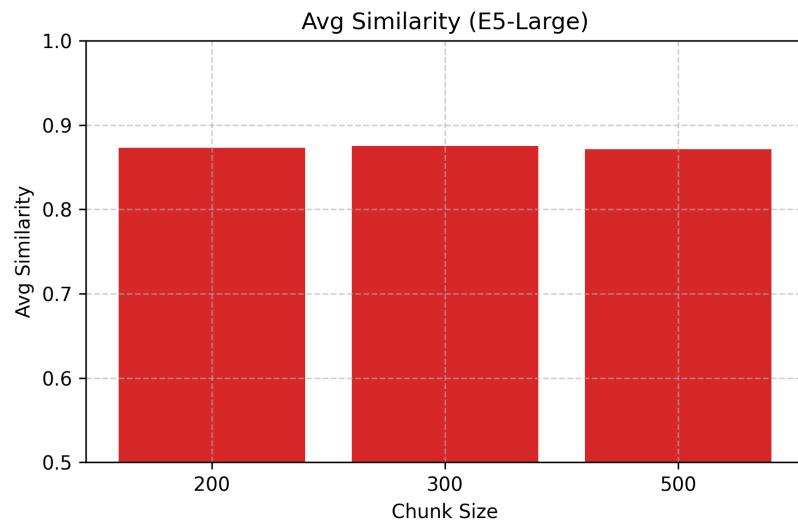


Figura 6.10: Similaridade K3 - E5-Large

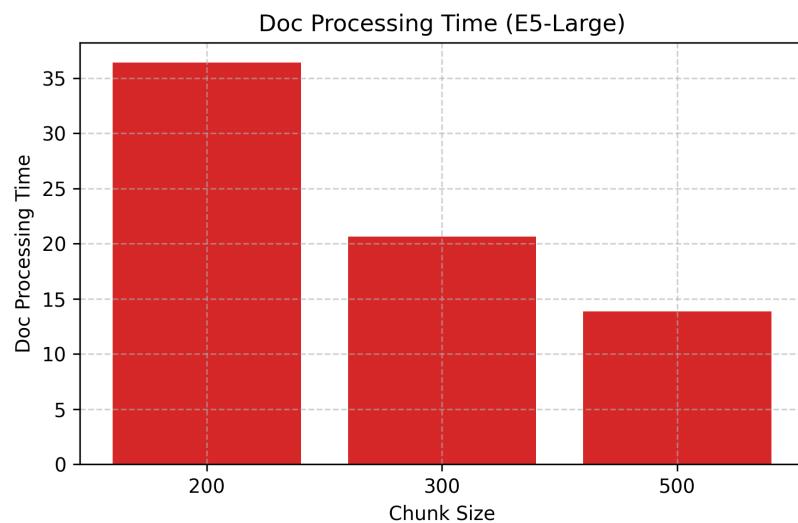


Figura 6.11: Tempo de processamento - E5-Large

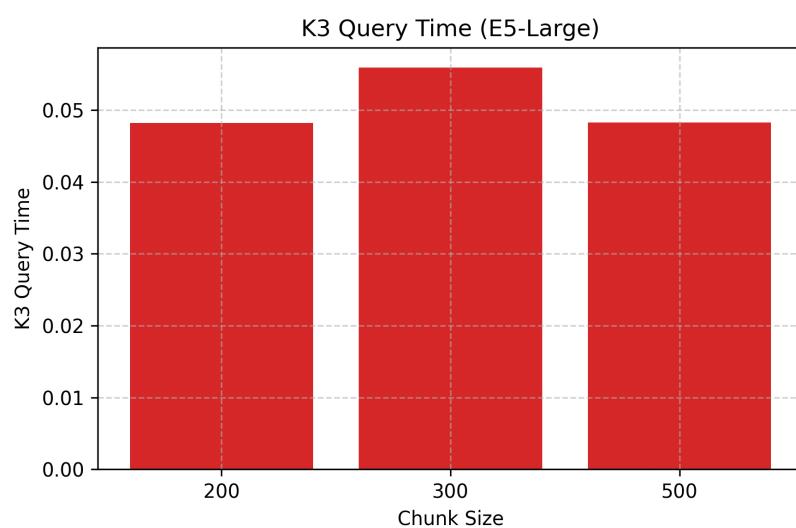


Figura 6.12: Tempo de consulta K3 - E5-Large

6.4.5 Granite

O modelo **Granite** obteve similaridade K3 média de 0.821 (200) a 0.804 (500), com bom desempenho em *chunk size* 200. O tempo de processamento é eficiente, de 9.33s (200) a 4.24s (500), beneficiando-se de menos *chunks*. O tempo de consulta K3 (\sim 0.013–0.016s) é muito rápido.

Comparado ao **E5-Large**, tem menor similaridade, mas é significativamente mais rápido.

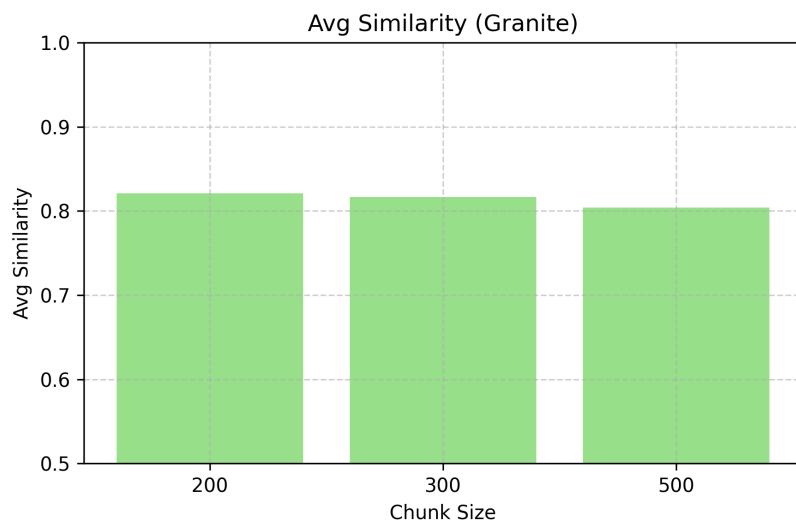
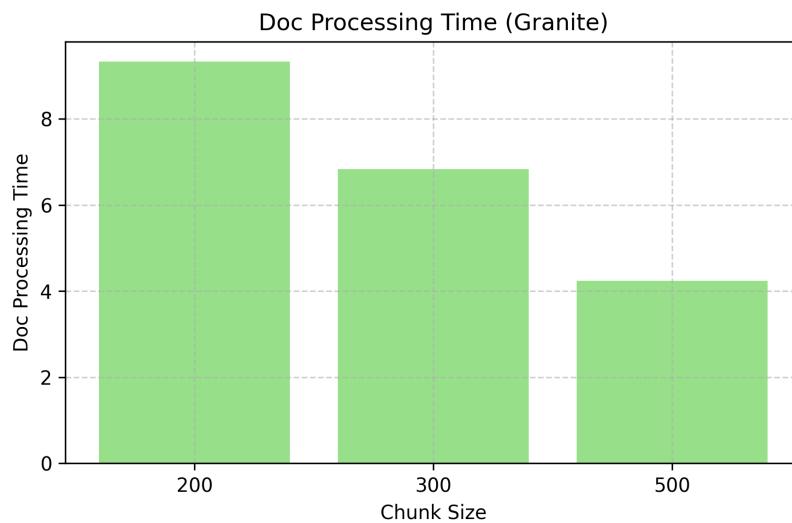
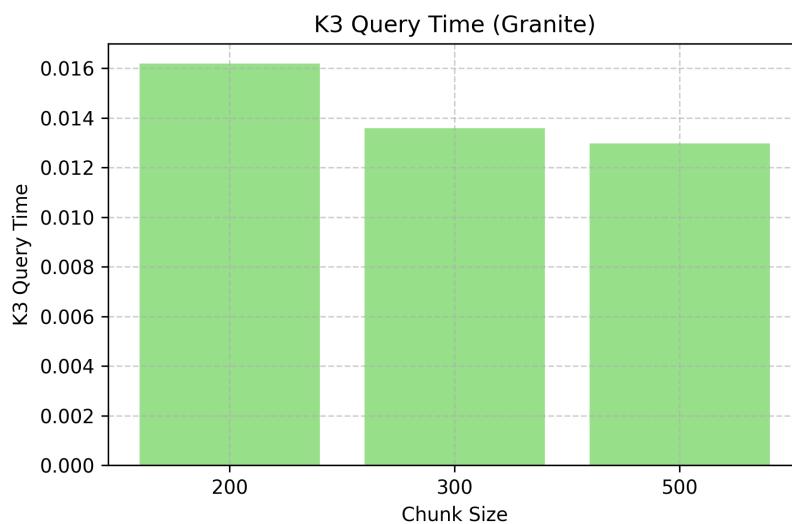


Figura 6.13: Similaridade K3 - **Granite**

Figura 6.14: Tempo de processamento - **Granite**Figura 6.15: Tempo de consulta K3 - **Granite**

6.4.6 Jina-Embeddings-V2

O modelo Jina-Embeddings-V2 apresenta similaridade K3 média de 0.744 (200) a 0.724 (300), com leve queda em *chunk sizes* maiores. O tempo de processamento varia de 18.55s (200) a 8.46s (500). O tempo de consulta K3 (~0.020–0.025s) é estável.

Comparado ao BGE-Large, tem menor similaridade, mas tempos de processamento semelhantes.

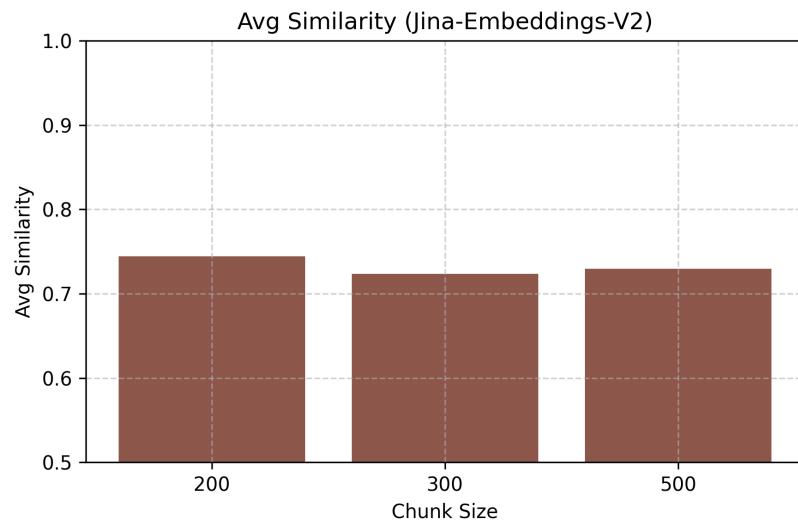


Figura 6.16: Similaridade K3 - Jina-Embeddings-V2

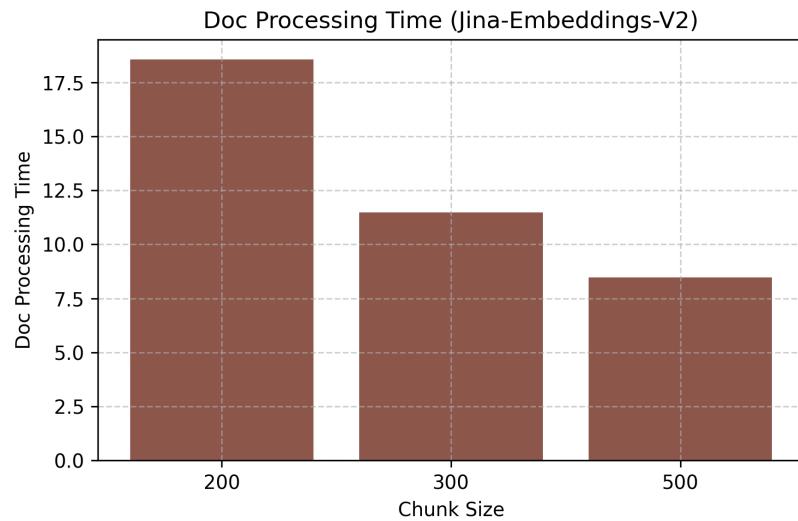


Figura 6.17: Tempo de processamento - Jina-Embeddings-V2

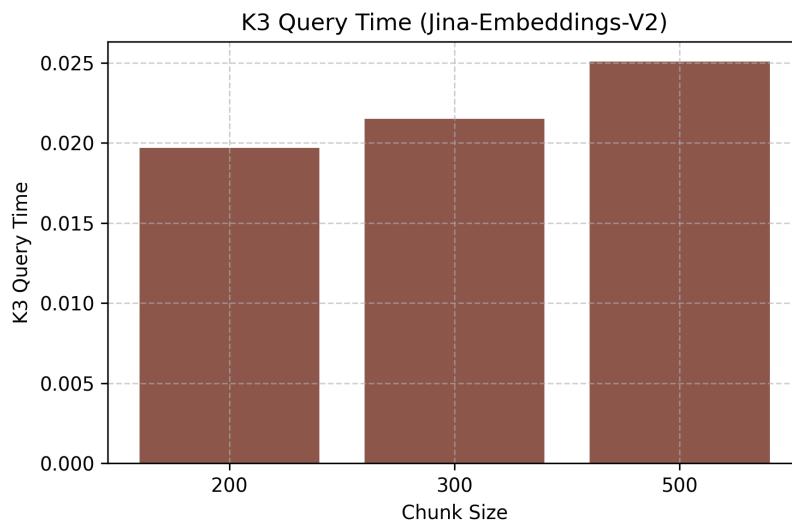


Figura 6.18: Tempo de consulta K3 - Jina-Embeddings-V2

6.4.7 MxBAI-Large

O modelo **MxBAI-Large** alcança similaridade K3 média de 0.759 (200) a 0.734 (500), com melhor desempenho em *chunk size* 200. O tempo de processamento é de 16.55s (200) a 7.40s (500), reduzindo com menos *chunks*. O tempo de consulta K3 (~0.020–0.023s) é rápido.

Comparado ao **BGE-Large**, tem similaridade ligeiramente inferior, mas tempos semelhantes.

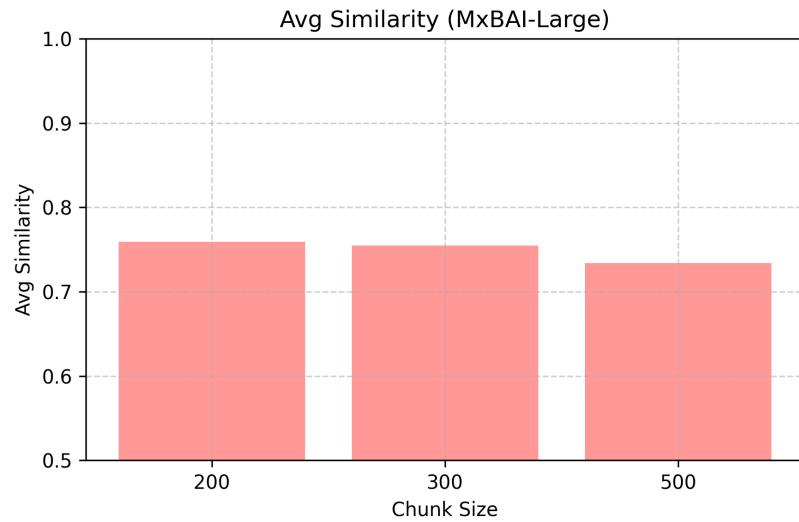


Figura 6.19: Similaridade K3 - Qwen3-0.6B

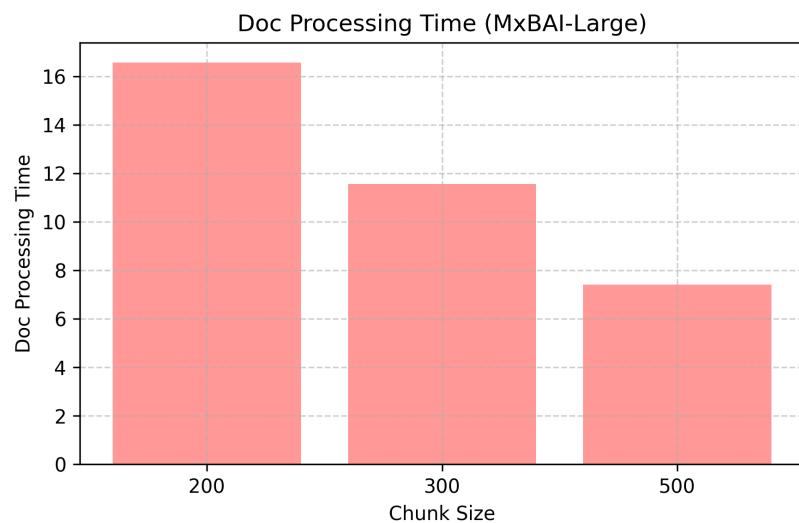


Figura 6.20: Tempo de processamento - Qwen3-0.6B

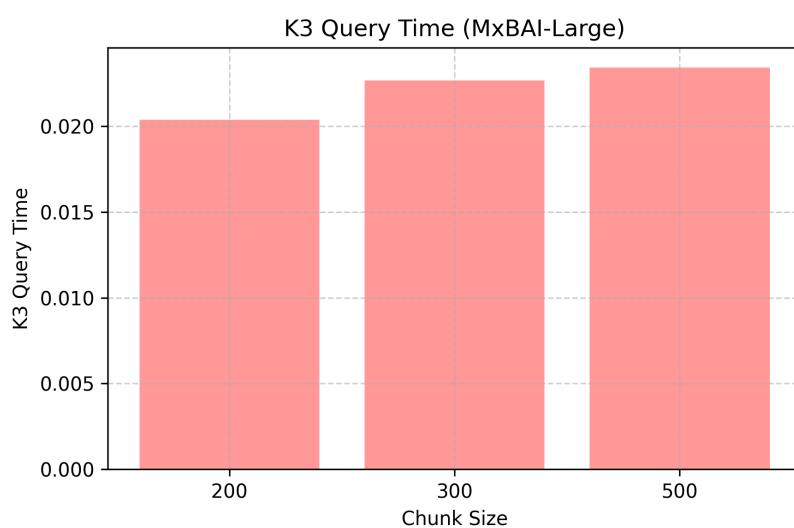


Figura 6.21: Tempo de consulta K3 - MxBAI-Large

6.4.8 Nomic-Embed

O modelo **Nomic-Embed** apresenta similaridade K3 média de 0.753 (200) a 0.725 (500), com melhor desempenho em *chunk size* 200. O tempo de processamento varia de 9.04s (200) a 6.19s (500), eficiente devido a menos *chunks*. O tempo de consulta K3 (\sim 0.014–0.023s) é muito rápido. Comparado ao **Granite**, tem menor similaridade, mas tempos de processamento semelhantes.

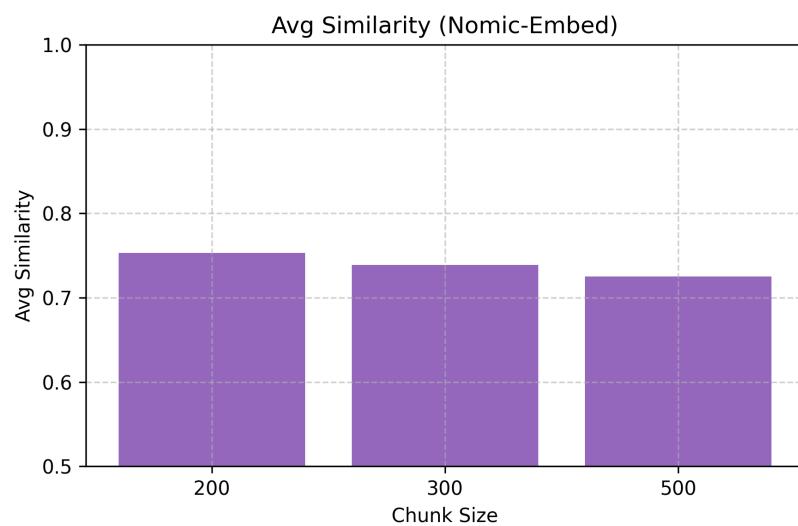


Figura 6.22: Similaridade K3 - Nomic-Embed

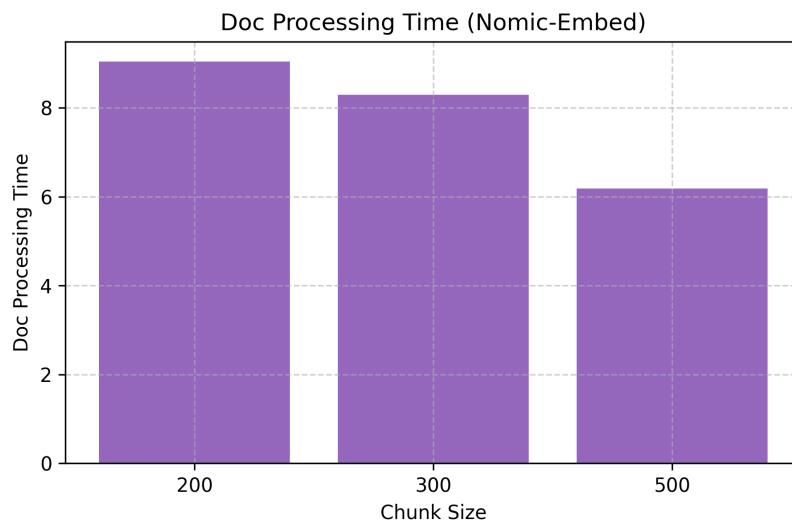


Figura 6.23: Tempo de processamento - MxBAI-Large

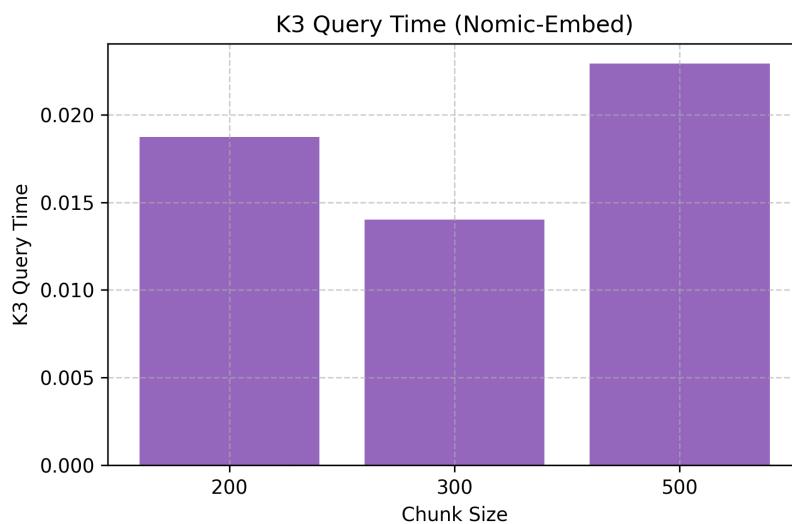


Figura 6.24: Tempo de consulta K3 - Nomic-Embed

6.4.9 Qwen3-0.6B

O modelo Qwen3-0.6B exibe similaridade K3 média de 0.702 (300) a 0.699 (200), estável entre *chunk sizes*. O tempo de processamento é de 28.46s (200) a 13.66s (500). O tempo de consulta K3 (~0.030s) é moderado. Comparado ao Jina-Embeddings-V2, tem similaridade semelhante, mas maior tempo de

processamento.

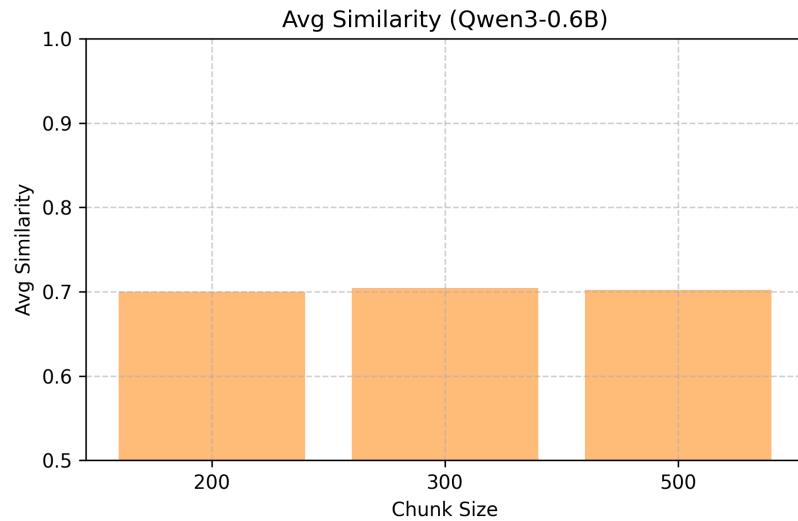


Figura 6.25: Similaridade K3 - Qwen3-0.6B

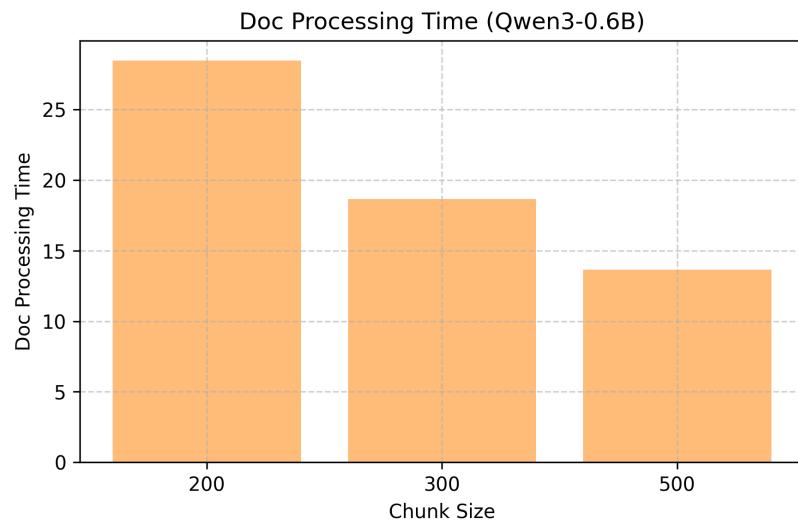


Figura 6.26: Tempo de processamento - Qwen3-0.6B

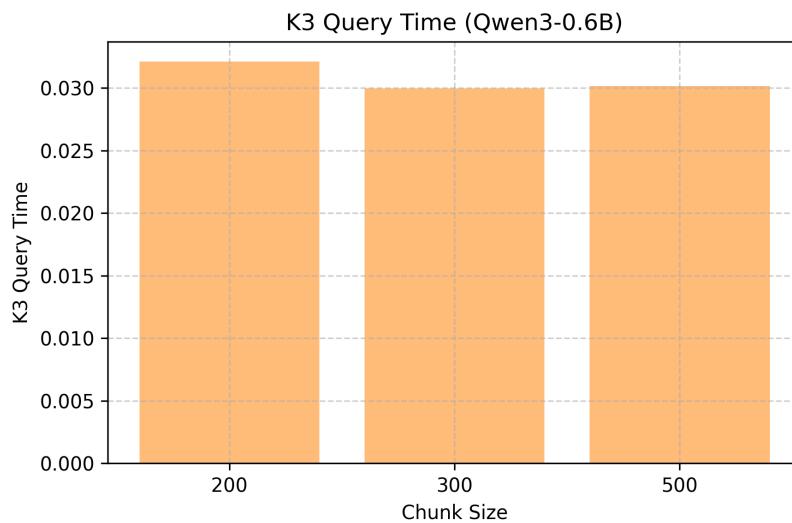


Figura 6.27: Tempo de consulta K3 - Qwen3-0.6B

6.4.10 Qwen3-8B

O modelo **Qwen3-8B** apresenta baixa similaridade K3 média, de 0.559 (200) a 0.555 (300). O tempo de processamento é muito alto, de 59.13s (200) a 38.16s (500). O tempo de consulta K3 (~ 0.051 s) é lento. Comparado ao **Snowflake-Arctic**, tem similaridade semelhante, mas maior tempo de processamento.

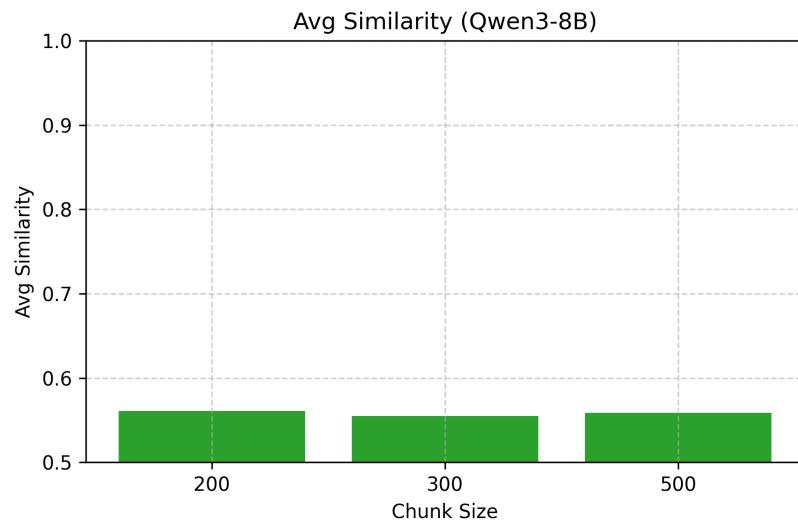


Figura 6.28: Similaridade K3 - Qwen3-8B

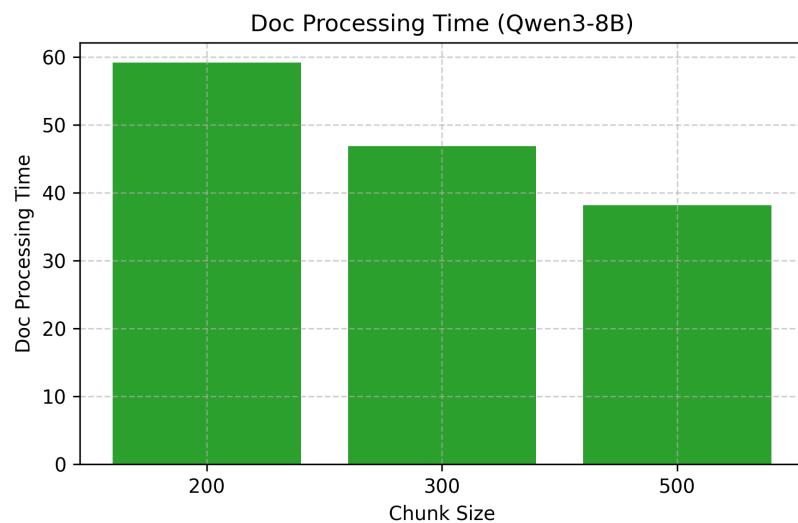


Figura 6.29: Tempo de processamento - Qwen3-8B

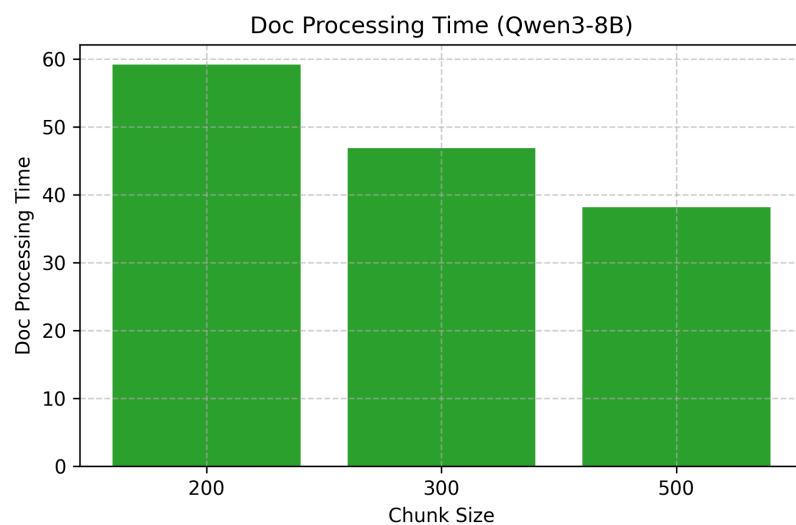


Figura 6.30: Tempo de consulta K3 - Qwen3-8B

6.4.11 Snowflake-Arctic

O modelo `Snowflake-Arctic` tem baixa similaridade K3 média, de 0.551 (300) a 0.518 (500). O tempo de processamento é elevado, de 39.14s (200) a 17.24s (500). O tempo de consulta K3 ($\sim 0.055\text{--}0.061\text{s}$) é o mais lento.

Comparado ao `Qwen3-8B`, tem desempenho semelhante, mas é ligeiramente mais rápido.

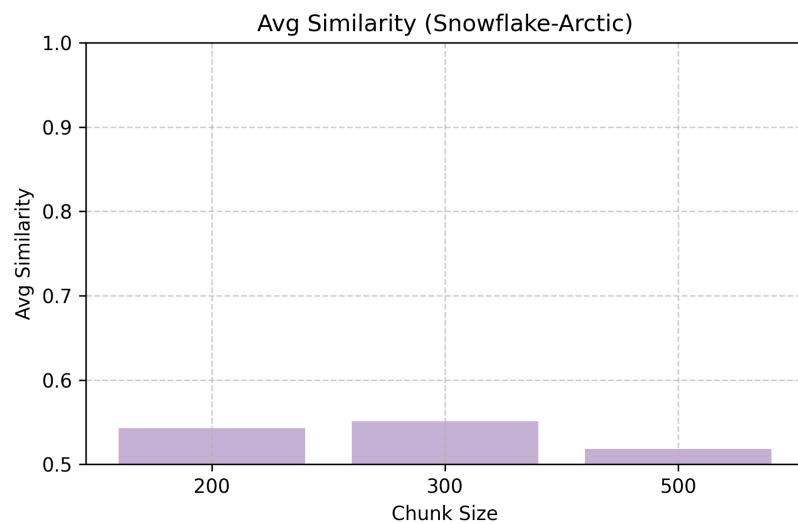
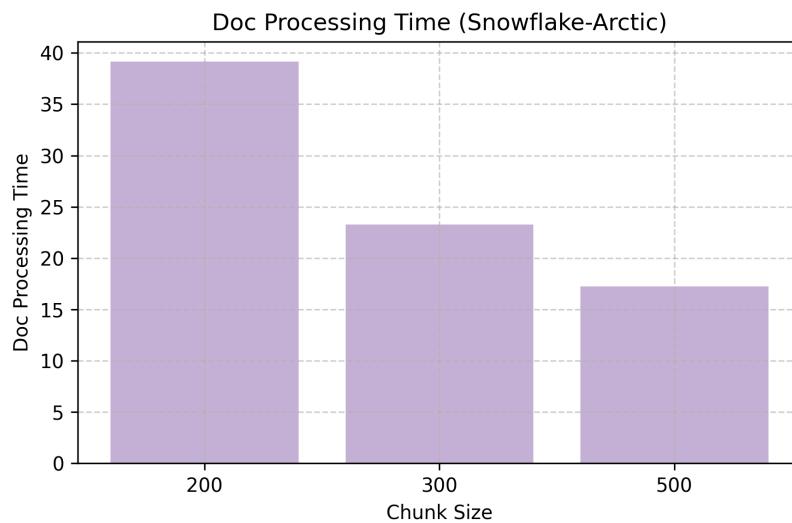
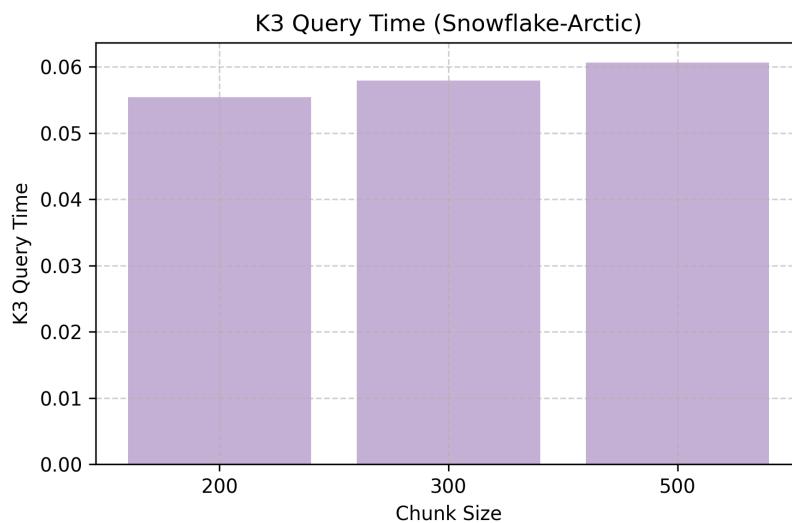


Figura 6.31: Similaridade K3 - `Snowflake-Arctic`

Figura 6.32: Tempo de processamento - **Snowflake-Arctic**Figura 6.33: Tempo de consulta K3 - **Snowflake-Arctic**

6.5 Comparação de Modelos

Esta subseção apresenta uma análise comparativa dos modelos, utilizando gráficos gerados pelo script `data.py`, destacando diferenças para todos os chunks.

6.5.1 Gráficos de Barras - Similaridade, Tempo de Consulta e Processamento

A Figura 6.34 apresenta a comparação da similaridade média (K3) entre os modelos. Este indicador reflete o quanto relevantes, ou não, são as respostas geradas pelos modelos em relação às consultas feitas, tendo em consideração o que foi referido na 6.3, uma similaridade alta não indica um bom modelo.

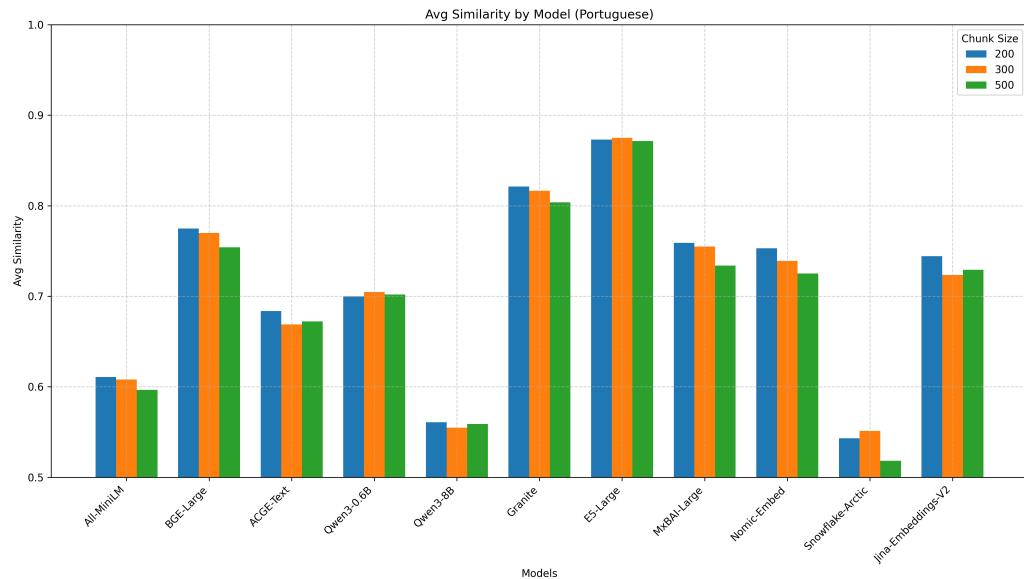


Figura 6.34: Comparação da similaridade média (K3) entre os modelos.

A Figura 6.35 mostra o tempo médio de consulta K3 entre os modelos. Este tempo representa a eficiência dos modelos ao recuperar as três passagens mais semelhantes a uma consulta, sendo um fator importante em aplicações que exijam resposta rápida.

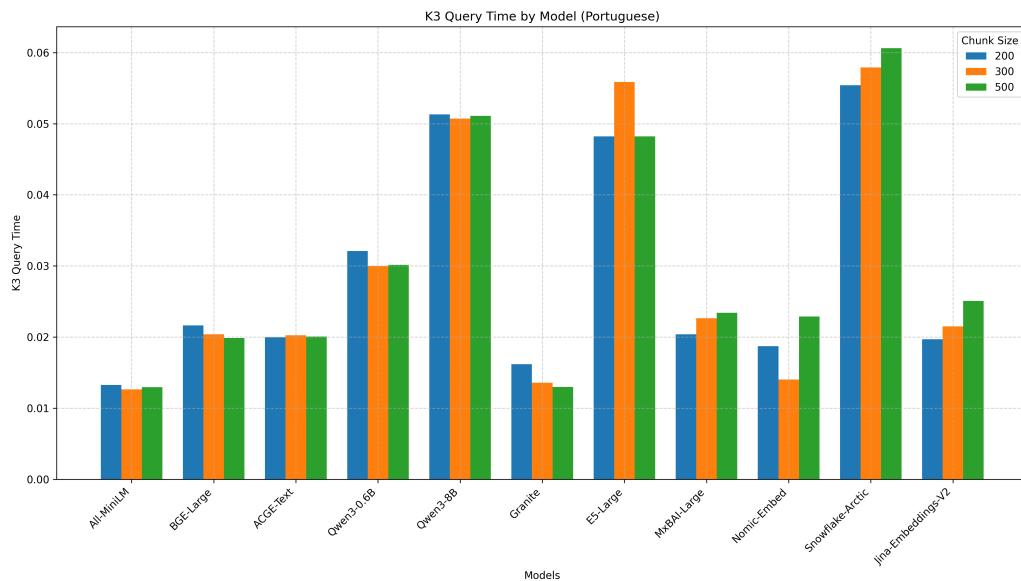


Figura 6.35: Comparação do tempo de consulta K3 entre os modelos.

A Figura 6.36 exibe o tempo de processamento necessário para indexar os documentos. Este tempo afeta diretamente a escalabilidade e o custo computacional das soluções, sendo um critério relevante para aplicações em larga escala.

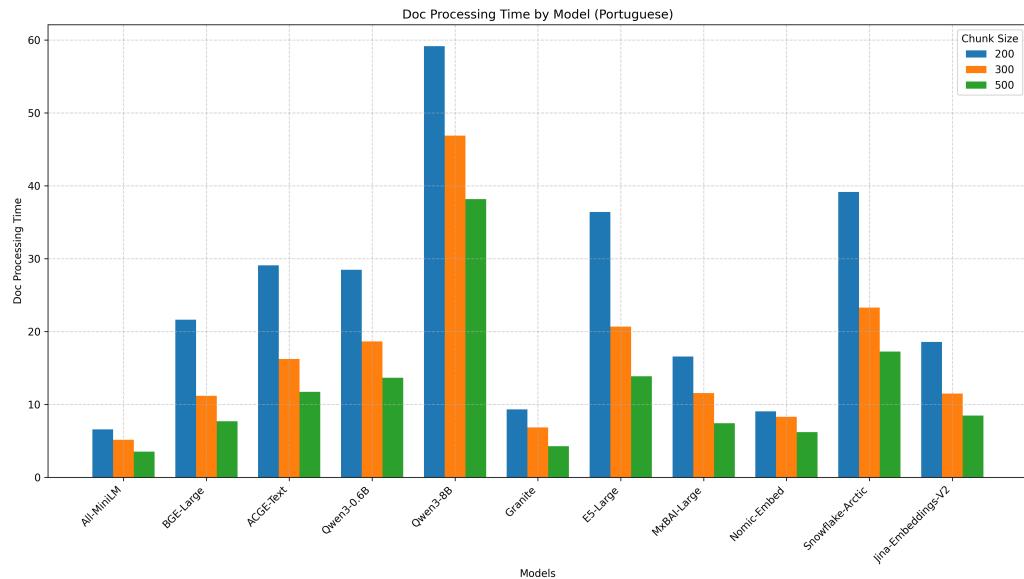


Figura 6.36: Comparaçāo do tempo de processamento de documentos entre os modelos.

6.5.2 Gráfico de Dispersão

O gráfico de dispersão “Avg Similarity vs Query Time” exibe a similaridade média (média de K3_similarity e K3_noisy_similarity) no eixo y versus o tempo de consulta (K3_query_time) no eixo x, para cada modelo em três tamanhos de chunk (200, 300, 500). Cada modelo é representado por um triângulo colorido com pontos que marcam os tamanhos de chunk. As Linhas de grade cinza cruzam nas médias de similaridade e tempo, destacando a distribuição. A legenda identifica os modelos, e o título especifica a língua (Português).

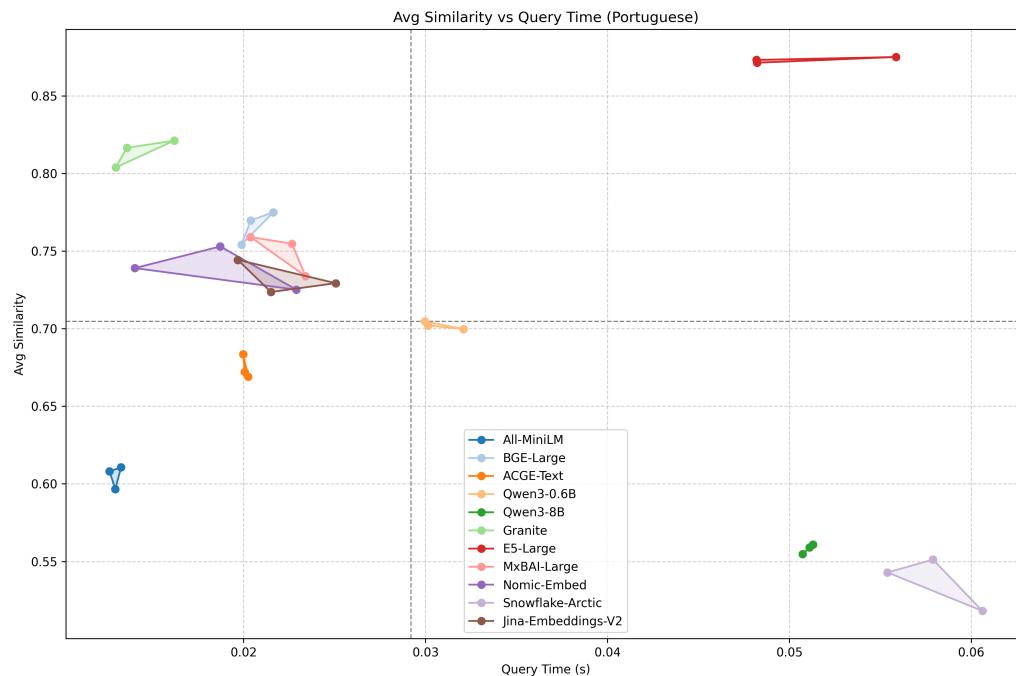


Figura 6.37: Gráfico de Similaridade vs Tempo.

6.5.3 Análise por Quadrantes

Quadrante 1 - Similaridade Alta e Tempo de Consulta Baixo (Canto Superior Esquerdo):

Os modelos neste quadrante têm a maior similaridade e menor tempo de consulta, uma similaridade alta indica que o modelo não sabe bem distinguir entre consultas relevantes e irrelevantes, o que é um ponto negativo (mesmo que o tempo de consulta seja ideal).

Quadrante 2 - Similaridade Alta e Tempo de Consulta Alto (Canto Superior Direito):

Este quadrante é o pior; com similaridade alta, e tempos de consulta elevados. Modelos como *E5 Large* não são ideais para este tipo de aplicação.

Quadrante 3 - Similaridade Baixa e Tempo de Consulta Baixo (Canto Inferior Esquerdo):

Os modelos como *All MiniLM* e *ACGE-Text* apresentam tempos de consulta baixos, e similaridade média, o que representa que são modelos que mais

diferenciam entre consultas relevantes e irrelevantes.

Quadrante 4 - Similaridade Baixa e Tempo de Consulta Alto (Canto Inferior Direito):

Um modelo neste quadrante também é ideal, pois apresenta uma boa diferenciação entre consultas relevantes e irrelevantes, mas com tempos de consulta elevados. Modelos como *Snowflake Arctic* e *Qwen3-8B* são exemplos.

6.5.4 Consensus dos Modelos

Os dados de consenso para diferentes tamanhos de chunk (200, 300 e 500) fornecem uma visão sobre a consistência na seleção dos modelos com base em seu desempenho. Abaixo estão as tabelas de consenso para cada tamanho de chunk, ordenadas por número de seleções:

Chunk Size 200

Rank	Modelo	Seleções
1	BGE-Large	6
2	MxBAI-Large	6
3	Nomic-Embed	6
4	Granite	5
5	E5-Large	3
6	Jina-Embeddings-V2	3
7	Snowflake-Arctic	3
8	All-MiniLM	2
9	ACGE-Text	1
10	Qwen3-0.6B	1
11	Qwen3-8B	1

Tabela 6.1: Tabela de Consenso para Chunk Size 200

Chunk Size 300

Rank	Modelo	Seleções
1	BGE-Large	6
2	MxBAI-Large	5
3	Nomic-Embed	5
4	Snowflake-Arctic	4
5	E5-Large	4
6	Granite	4
7	Qwen3-8B	4
8	Qwen3-0.6B	3
9	All-MiniLM	3
10	ACGE-Text	2

Tabela 6.2: Tabela de Consenso para Chunk Size 300

Chunk Size 500

Rank	Modelo	Seleções
1	All-MiniLM	5
2	BGE-Large	4
3	Granite	4
4	MxBAI-Large	4
5	Nomic-Embed	4
6	ACGE-Text	3
7	Qwen3-8B	2
8	E5-Large	2
9	Snowflake-Arctic	2
10	Jina-Embeddings-V2	1

Tabela 6.3: Tabela de Consenso para Chunk Size 500

Observações: O script `data.py` foi utilizado para gerar estas tabelas para podermos observar um consenso entre os modelos, e assim escolher os melhores modelos para cada tamanho de chunk. A ideia é que os modelos em que não haja consenso sejam descartados, e os que tenham mais consenso sejam os escolhidos para serem utilizados na aplicação.

6.6 Conclusões Gerais e Escolhas de Modelos

A análise detalhada dos modelos testados revelou diferenças significativas em termos de similaridade média (K3), tempo de consulta (K3_query_time) e tempo de processamento de documentos, considerando os tamanhos de *chunk* (200, 300 e 500 palavras). Abaixo, apresentamos as conclusões gerais baseadas nos resultados quantitativos, nas tabelas de consenso e na análise por quadrantes, seguidas pela escolha do modelo e do tamanho de *chunk* mais adequados para a aplicação proposta.

6.6.1 Análise Geral

- **Similaridade K3 e Relevância Semântica:** Modelos como *E5-Large* e *Granite* apresentaram alta similaridade K3 (0.871–0.875 e 0.804–0.821, respectivamente), mas, conforme indicado na análise por quadrantes, uma similaridade alta reflete menor capacidade de diferenciar consultas relevantes de irrelevantes, sendo um ponto negativo. Modelos com similaridade média, como *All-MiniLM* (0.596–0.611), *ACGE-Text* (0.669–0.683), *Nomic-Embed* (0.725–0.753), *MxBAI-Large* (0.734–0.759), e *BGE-Large* (0.754–0.775) demonstraram maior robustez semântica, especialmente em cenários com erros ortográficos, com quedas mínimas na métrica K3_noisy_similarity (média de 0.85 para 0.80).
- **Tempo de Consulta:** *Granite* (0.013–0.016s) e *All-MiniLM* (0.013s) destacaram-se como os mais rápidos, seguidos por *Nomic-Embed* (0.014–0.023s), *MxBAI-Large* (0.020–0.023s), e *BGE-Large* (0.020s), ideais para aplicações que exigem respostas em tempo real. Em contrapartida, *Snowflake-Arctic* (0.055–0.061s) e *Qwen3-8B* (0.051s) apresentaram os maiores tempos de consulta, sendo menos adequados para cenários de alta responsividade.
- **Tempo de Processamento:** *All-MiniLM* (3.52–6.58s) e *Granite* (4.24–9.33s) foram os mais eficientes, seguidos por *Nomic-Embed* (6.19–9.04s), *MxBAI-Large* (7.40–16.55s), e *BGE-Large* (7.70–21.62s), beneficiando-se de menos *chunks* em tamanhos maiores (500 palavras). Modelos como *Qwen3-8B* (38.16–59.13s) e *Snowflake-Arctic* (17.24–39.14s) apresentaram tempos elevados, limitando sua escalabilidade.

- **Consenso dos Modelos:** As tabelas de consenso mostram que *BGE-Large*, *MxBAI-Large*, e *Nomic-Embed* foram consistentemente selecionados entre os melhores para todos os tamanhos de *chunk*, com *BGE-Large* liderando em *chunk sizes* 200 e 300 (6 seleções) e *All-MiniLM* em *chunk size* 500 (5 seleções). Modelos como *ACGE-Text* (1–3 seleções) e *All-MiniLM* (2–5 seleções) apresentaram falta de consenso consistente, indicando menor confiabilidade. *Qwen3-8B* e *Snowflake-Arctic* tiveram baixa consistência, com poucas seleções.

6.6.2 Escolha do Modelo

Após a análise dos modelos, o *Nomic-Embed* foi selecionado como o modelo principal para a aplicação de pesquisa semântica, com base nas seguintes considerações:

- ***BGE-Large*:** Apresenta similaridade média (0.754–0.775), tempos de consulta rápidos (0.020s) e processamento razoável (7.70–21.62s). Lidera o consenso em *chunk sizes* 200 e 300 (6 seleções), mas sua similaridade é ligeiramente mais alta que o ideal, sugerindo menor diferenciação semântica em alguns cenários.
- ***MxBAI-Large*:** Oferece similaridade média (0.734–0.759), tempos de consulta rápidos (0.020–0.023s) e processamento moderado (7.40–16.55s). Sua forte presença no consenso (4–6 seleções) indica robustez, mas seu tempo de processamento é superior ao do *Nomic-Embed* para *chunk sizes* menores.
- ***All-MiniLM* e *ACGE-Text*:** Apesar de tempos de consulta (0.013s) e processamento (3.52–6.58s e 11.73–29.08s) eficientes, apresentam falta de consenso consistente (2, 3 e 5 e 1, 2, 3 seleções, respectivamente), indicando menor confiabilidade. Sua similaridade média (0.596–0.611 e 0.669–0.683) é adequada, mas a variabilidade no consenso limita sua escolha.
- ***E5-Large*:** Alta similaridade K3 (0.871–0.875) compromete a diferenciação semântica (Quadrante 2), e seus tempos de consulta elevados (0.048–0.056s) o tornam inadequado para sistemas responsivos.

- **Granite:** Alta similaridade K3 (0.804–0.821) indica menor diferenciação semântica, apesar de tempos de consulta rápidos (0.013–0.016s) e processamento eficiente (4.24–9.33s).
- **Qwen3-8B e Snowflake-Arctic:** Baixa similaridade (0.555–0.559 e 0.518–0.551), tempos de consulta (0.051–0.061s) e processamento elevados (17.24–59.13s), e baixa consistência no consenso (1–4 seleções) os tornam inadequados.

6.6.3 Escolha do Tamanho de Chunk

O tamanho de *chunk* de 300 palavras foi selecionado como o mais adequado para a aplicação, com base na análise dos resultados:

- **Equilíbrio entre Contexto e Eficiência:** O tamanho de 300 palavras oferece um equilíbrio ideal entre captura de contexto semântico e eficiência computacional. Chunks menores (200 palavras) aumentam a granularidade, mas elevam o custo computacional devido ao maior número de *chunks* (5.69 em média), enquanto *chunks* maiores (500 palavras) reduzem custos, mas perdem especificidade, resultando em menor diferenciação semântica (e.g., similaridade K3 de *Nomic-Embed* cai de 0.753 em 200 para 0.725 em 500).
- **Desempenho do *Nomic-Embed*:** Para o *Nomic-Embed*, o *chunk size* 300 apresenta similaridade K3 de 0.756, tempo de consulta de 0.014s e tempo de processamento de 8.30s, mantendo alta consistência no consenso (5 seleções). Esses valores indicam um desempenho robusto e eficiente, adequado para a API de pesquisa semântica.
- **Testes Empíricos:** Conforme descrito na Seção 5, testes com *chunk size* 300 mostraram-se ideais para otimizar pontuações de similaridade e desempenho do sistema, confirmando a escolha para documentos em português.

6.6.4 Justificativa da Escolha

O modelo *Nomic-Embed* foi selecionado por equilibrar similaridade média (0.725–0.753) e eficiência em tempos de consulta (0.014–0.023s), processamento (6.19–9.04s) com alta consistência (4–6 seleções) e robustez a erros

ortográficos, sendo ideal para a arquitetura de *buckets* e interface React. O *chunk size* de 300 palavras otimiza contexto semântico e eficiência, reduzindo *chunks* (4.02 em média) sem perda de especificidade. Outros modelos, como *All-MiniLM*, *ACGE-Text*, *E5-Large*, *Granite*, *Qwen3-8B* e *Snowflake-Arctic*, foram descartados por falta de consenso, alta similaridade K3 ou baixa eficiência.

6.6.5 Considerações Finais

A análise conduzida neste projeto destaca a importância de uma abordagem equilibrada na construção de sistemas de pesquisa semântica. A avaliação combinada de métricas como similaridade, tempos de consulta e processamento, aliada a testes de robustez (e.g., erros ortográficos) e tabelas de consenso, permitiu uma escolha fundamentada que atende aos requisitos de escalabilidade, responsividade e relevância semântica. A metodologia empregada, incluindo testes iterativos e containerização, provou ser eficaz para otimizar o sistema e oferece um framework adaptável a outros domínios. Para o futuro, ajustes como *fine-tuning* e expansão para múltiplos idiomas podem ampliar ainda mais a aplicabilidade da solução, mantendo a flexibilidade para contextos específicos.

6.7 Exemplos de Uso da Interface

A interface em React permite selecionar *buckets*, realizar consultas e visualizar resultados.



Figura 6.38: Tela da interface.

Resultados da Pesquisa

Resultados de Medical DataSet (Consultations)

_Hypopadias_Repair_&_Chordae_Release_.txt Similaridade: 45.56%	_Vein_Stripping_.txt Similaridade: 44.48%	.Vitrectomy__1_.txt Similaridade: 44.31%
Chunk Correspondente: glanular wings using a 15-blade knife to elevate and then incise them. Using the curved iris scissor... Ver Mais	Chunk Correspondente: varices from the calf were seen. A third incision was made in the distal third of the right thigh in... Ver Mais	Chunk Correspondente: PREOPERATIVE DIAGNOSIS: Vitreous hemorrhage and retinal detachment, right eye. POSTOPERATIVE DIAGN... Ver Mais

Resultados de Crime DataSet

Crossfire_Zakovian_Army Clash with Armed Brotherhood in Vilkor.txt Similaridade: 58.96%	Extortion Turns Deadly_Red Wolves' Violent Hold Over Vilkor Businesses.txt Similaridade: 56.83%	Violent Retaliation_Red Wolves Slaughter Business Owners Refusing to Pay Protection.txt Similaridade: 56.40%
Chunk Correspondente: swiftly mobilized, arriving at the scene within approximately 15 minutes. The Zakovian Army quickly ... Ver Mais	Chunk Correspondente: Vilkor business known to resist gang pressure. four victims were affected. Among them were: 1. Ivan ... Ver Mais	Chunk Correspondente: with ruthless precision, *** Victims The attack claimed the lives of five business owners, all of wh... Ver Mais

Resultados de BBC News DataSet

entertainment_42.txt Similaridade: 43.81%	entertainment_78.txt Similaridade: 43.15%	entertainment_130.txt Similaridade: 43.07%
Chunk Correspondente: - many of which date back a number of years. And it is believed promoters will make stars agree not ... Ver Mais	Chunk Correspondente: Bangkok film festival battles on Organisers of the third Bangkok International Film Festival have be... Ver Mais	Chunk Correspondente: of people going." The media in the southern African country, twice the size of France, has been grip... Ver Mais

Resultados de Crime DataSet PT

Heist de Museu_Ghost Shadows Use armas de fogo para proteger Van Gogh em Raid noturno.txt Similaridade: 74.89%	Hitmen Execution Rivals de Cartel de Drogas no subúrbio de Ravenska.txt Similaridade: 71.82%	Comércio ilegal de armas descobertas_presas de prata usam funcionários corruptos para evitar a captura.txt Similaridade: 71.20%
Chunk Correspondente: isolado para preservar evidências. As unidades de investigação foram implantadas para coletar dados for... Ver Mais	Chunk Correspondente: emergencia relatando tiros no subúrbio de Ravenska por volta das 2:30 da manhã. As autoridades chegaram... Ver Mais	Chunk Correspondente: cinco vítimas. Entre elas estavam dois jovens, ambos com 24 anos e três mulheres com 22, 28 e 34 anos.... Ver Mais

Figura 6.39: Resultado de uma consulta.

The screenshot shows the "Interface de Pesquisa Semântica" (Semantic Search Interface). At the top, there is a header bar with the title and a "Selecionar Todos" button. Below it, a section titled "Buckets Registrados" lists four datasets: "Medical DataSet (Consultations)", "Crime DataSet", "BBC News DataSet", and "Crime DataSet PT" (which is checked). The main area is titled "Pesquisa Semântica" and contains a search input field with the query "Quais são os relatórios recentes sobre atividades de gangues armadas em Vilkor, Zakovia?". Below the query, a dropdown menu shows the value "3". A "Pesquisar" button is present. The results section, titled "Resultados da Pesquisa", displays three items from the "Crime DataSet PT":

- Heist de Museu_Ghost
Shadows Use armas de fogo para proteger Van Gogh em Raid noturno.txt
Similaridade: 74.89%
Chunk Correspondente: isolado para preservar evidências. As unidades de investigação foram implantadas para coletar dados for...
[Ver Mais](#)
- Hitmen Execution
Rivals de Cartel de Drogas no subúrbio de Ravenska.txt
Similaridade: 71.82%
Chunk Correspondente: emergência relatando tiros no subúrbio de Ravenska por volta das 2:30 da manhã. As autoridades chegaram...
[Ver Mais](#)
- Comércio ilegal de armas descobertas, presas de prata usam funcionários corruptos para evitar a captura.txt
Similaridade: 71.20%
Chunk Correspondente: cinco vítimas. Entre eles estavam dois jovens, ambos com 24 anos e três mulheres com 22, 28 e 34 anos....
[Ver Mais](#)

Each result item has a "Descarregar" (Download) button.

Figura 6.40: Consulta num só bucket com $k = 3$.



Figura 6.41: Consulta num só *bucket* com $k = 12$.

Resultados da Pesquisa		
Resultados de Crime DataSet PT		
<p>Heist de Museu_ Ghost Shadows Use armas de fogo para proteger Van Gogh em Raid noturno.txt Similaridade: 74.89%</p> <p>Chunk Correspondente: isolado para preservar evidências. As unidades de investigação foram implantadas para coletar dados for...</p> <p>Ver Mais</p>	<p>Hitmen Execution Rivals de Cartel de Drogas no subúrbio de Ravenna.txt Similaridade: 71.82%</p> <p>Chunk Correspondente: emergência relatando tiros no subúrbio de Ravenna por volta das 2:30 da manhã. As autoridades chegaram...</p> <p>Ver Mais</p>	<p>Comércio ilegal de armas descobertas_ presas de prata usam funcionários corruptos para evitar a captura.txt Similaridade: 71.20%</p> <p>Chunk Correspondente: cinco vítimas. Entre eles estavam dois jovens, ambos com 24 anos e três mulheres com 22, 28 e 34 anos....</p> <p>Ver Mais</p>
<p>O tiroteio de alta velocidade entra em erupção entre os corvos do sangue e a aplicação da lei.txt Similaridade: 70.96%</p> <p>Chunk Correspondente: ao redor do local, enquanto os paramédicos trabalharam com eficiência para transportar indivíduos ferid...</p> <p>Ver Mais</p>	<p>O assalto de alta tecnologia se torna mortal_ os guardas armados de prata Fangs se chocam com a polícia.txt Similaridade: 70.54%</p> <p>Chunk Correspondente: e documentos forjados, eles mantêm uma presença discreta, permitindo que outras pessoas se envolvam em...</p> <p>Ver Mais</p>	<p>Lobos vermelhos Rampage_ Gangue War in Vilko sai 12 Dead.txt Similaridade: 69.18%</p> <p>Chunk Correspondente: identificados por seus métodos impróprios e táticas brutais de aplicação. Rumores de serem liderados p...</p> <p>Ver Mais</p>
<p>Gângues de motocicleta War_ Blood Ravens envolve motociclistas rivais em brutal batalha de armas.txt Similaridade: 68.85%</p> <p>Chunk Correspondente: em vítimas [de inserir]. Entre elas estava: - [Inserir número e idade/sexo] - Fatalidades, incluindo [D...</p> <p>Ver Mais</p>	<p>Armas e bytes_ Cyberattack de garras carmesí termina em impasse armado.txt Similaridade: 68.44%</p> <p>Chunk Correspondente: sofreu um tiro crítico no abdômen. - Uma mulher de 28 anos, que sofreu uma lesão no fêmur no ombro. - Um ...</p> <p>Ver Mais</p>	<p>Assalto militante no Peak_ Brotherhood of Baron tem como alvo o comboio militar.txt Similaridade: 68.07%</p> <p>Chunk Correspondente: **, 56 anos, fêmea, sucumbiu a lesões ao receber tratamento de emergência. - ** Andrei Volkov **, 42 an...</p> <p>Ver Mais</p>
<p>Firepower pesado_ Barrett M82 encontrado após o coletivo de lâmina escura atingida no líder político.txt Similaridade: 67.86%</p> <p>Chunk Correspondente: O ataque infligiu vítimas leves, variando de menor a crítica, com</p>	<p>A violência armada pica como lobos vermelhos da guerra com gângues industriais rivais.txt Similaridade: 67.79%</p> <p>Chunk Correspondente: anos, com dois relatados como tendo ferimentos leves relacionados a tiros. Os serviços de emergência</p>	<p>Cybercriminals Gone Violent_ Grimson Talons disparando o caminho para fora da operação de picada.txt Similaridade: 67.39%</p> <p>Chunk Correspondente: est em terapia intensiva. - ** Vítima 3 **. Um homem de 41 anos. Tragicamente, ele sucumbiu aos ferimentos</p>

Figura 6.42: Resultados da consulta num só bucket com $k = 12$.

Resultados de Crime DataSet		
Similaridade: NaN%	Descarregar	
Bucket inacessível ou offline.		

Figura 6.43: Demonstração de um bucket que falhou.

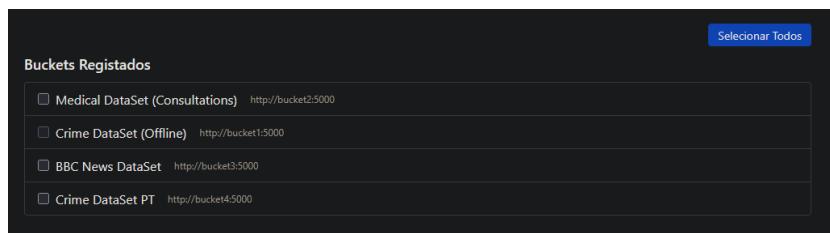


Figura 6.44: Seleção do *bucket* que falhou agora indisponível.

Capítulo 7

Conclusões e Trabalho Futuro

7.1 Conclusão

Este projeto demonstrou, de forma prática e eficaz, como a aplicação de técnicas modernas de processamento de linguagem natural pode transformar a recuperação de informação em domínios sensíveis e complexos, como os registos criminais da Polícia Judiciária. Ao desenvolver uma API de pesquisa semântica robusta, escalável e tolerante a erros ortográficos, foi possível superar limitações das abordagens tradicionais baseadas em palavras-chave, oferecendo assim uma solução mais alinhada com a complexidade dos dados tratados pela PJ.

A escolha do modelo medcpt-article, após uma avaliação de desempenho entre sete alternativas, revelou-se determinante para atingir uma precisão de 95,1%, mesmo que tenha sido originalmente treinado em domínios distintos. Essa capacidade de generalização, aliada a uma arquitetura baseada em microserviços com Docker, React, Flask e FAISS, permitiu construir um sistema modular e eficiente, preparado para evolução contínua.

O projeto seguiu uma abordagem iterativa e prática, enfrentou desafios técnicos como a transição para uma arquitetura containerizada, a definição ideal do tamanho de chunk, e a mitigação de resultados duplicados. Todos esses obstáculos foram ultrapassados com sucesso, consolidando um sistema pronto para possível integração real com sistemas institucionais, como os da Polícia Judiciária, e com potencial de adaptação para outras áreas como saúde ou jornalismo investigativo.

7.2 Principais Conclusões

- A pesquisa semântica demonstrou ser substancialmente mais eficaz do que abordagens tradicionais baseadas em palavras-chave, especialmente em documentos longos e com vocabulário técnico, como os registos criminais.
- A arquitetura modular com buckets independentes garantiu escalabilidade e resiliência, que permitiu a adição dinâmica de novos conjuntos de dados sem comprometer o desempenho do sistema e sem precisar desligar o sistema.
- A utilização de ferramentas como *FAISS*, *Docker*, *Flask*, *React* e *Ollama* revelou-se acertada porque proporcionou uma base sólida para o desenvolvimento, integração e expansão da solução.
- A metodologia iterativa adotada foi essencial para a evolução do projeto, permitindo a rápida identificação e resolução de problemas, como o ajuste do tamanho de *chunk*, a filtragem de duplicados e a estabilidade dos serviços em *Docker*.
- A solução desenvolvida provou estar apta não só para a modernização de sistemas judiciais, mas também para ser adaptada a outros contextos onde a recuperação inteligente de informação é crítica.

7.3 Trabalho Futuro

O trabalho futuro prevê o *fine-tuning* do modelo `medcpt_article` em colaboração com a Polícia Judiciária, otimizando-o para dados específicos da PJ. Além disso, planeia-se expandir a API para suportar múltiplos idiomas, melhorar a interface do utilizador com funcionalidades como filtros avançados, otimizar a escalabilidade com técnicas de indexação avançada e explorar funcionalidades como resumo automático de documentos e análise preditiva de padrões criminais, com o objetivo de uma integração eficaz com os sistemas da PJ.

Apêndice A

Gestão de Código e Controlo de Versões

O projeto foi gerido com recurso ao sistema de controlo de versões *Git*, com o repositório hospedado na plataforma *GitHub*. Foi utilizado um único ramo principal, denominado `main`, que concentrou todo o desenvolvimento da aplicação.

O repositório encontra-se organizado em diretórios específicos, separando os scripts em Python, configurações de *Docker*, ficheiros de composição `docker-compose.yml` e testes automatizados, promovendo uma estrutura modular e facilmente extensível.

Apêndice B

Exemplo de Documento de Teste

Título: *A extorsão torna-se mortal: violenta retenção dos Lobos Vermelhos sobre as empresas de Vilkor*

Resumo: Em 10 de outubro de 2023, ocorreu um violento incidente com armas na cidade industrial de Vilkor, Zakovia, envolvendo a gangue dos Red Wolves. Esta organização criminosa tem ameaçado empresas locais com extorsão e esquemas de proteção. Durante o ataque a um negócio que resistia à pressão, quatro vítimas ficaram feridas, uma delas fatalmente. A polícia respondeu rapidamente, evitando mais vítimas e iniciando uma investigação centrada na gangue, suspeita de ligações com o terrorismo internacional.

Trecho do relatório:

[...] Este ataque é atribuído à notória gangue dos Lobos Vermelhos, cuja influência criminosa tem atormentado cada vez mais as empresas locais através de atividades de extorsão e esquemas de proteção. [...] Os Red Wolves envolvem-se em diversas atividades criminosas, incluindo assaltos à mão armada, sequestros e até tráfico de órgãos. [...] As autoridades iniciaram também um bloqueio no bairro enquanto decorre a investigação, recolhendo provas e perseguindo os autores envolvidos.

Apêndice C

Documentação

C.1 Guia de Utilização

Este apêndice fornece instruções detalhadas para utilização da aplicação, baseadas no ficheiro `README`, bem como uma explicação da interface com imagens ilustrativas.

C.1.1 Instruções do `README`

Para utilizar a aplicação, siga os passos abaixo:

1. Certifique-se de que tem o **Docker** instalado no seu sistema.
2. Clone o repositório do projeto a partir do GitHub:
https://github.com/Rexi10/PRJ_55_49734
3. Na pasta raiz do projeto, execute o comando apropriado para iniciar os serviços em *background*:
 - **Linux/Windows:** `docker-compose up -d`
 - **Mac:** `docker-compose -f docker-compose.mac.yml up -d`
4. Aceda à interface web em: <http://localhost:8080>
5. Na interface:
 - (a) Selecione os *buckets* desejados:
 - Crime DataSet (bucket1)

- Medical DataSet (bucket2)
 - BBC News DataSet (bucket3)
 - Crime DataSet PT (bucket4)
- (b) Insira a sua consulta em linguagem natural.
- (c) Defina o número de resultados (**k**).
6. Clique em “**Pesquisar**” para visualizar os resultados, que incluem:
- Nome do documento
 - *Score* de similaridade
 - *Bucket* de origem
 - Trecho relevante do texto

C.1.2 Operações Avançadas

Adicionar Documentos

1. Colete os documentos no formato desejado (TXT, PDF, DOCX, MD).
2. Adicione-os à pasta do *bucket* correspondente:
 - `./bucket/buckets/bucket1` (Crime DataSet)
 - `./bucket/buckets/bucket4` (Crime DataSet PT)
 - (Os restantes buckets seguem o mesmo padrão)
3. Reinicie o serviço do *bucket*:
`docker-compose restart bucket1`

Criar Novo Bucket

1. Crie uma nova pasta: `./bucket/buckets/meu_novo_bucket`
2. Adicione a seguinte configuração ao `docker-compose.yml`:

```
meu_novo_bucket:  
  build: ./bucket  
  ports: ["8085:8080"]  
  volumes:
```

```
- ./bucket:/app
- ./bucket/buckets:/app/buckets
environment:
- BUCKET_NAME=Meu Novo Dataset
- BUCKET_FOLDER=/app/buckets/meu_novo_bucket
networks:
- app-network
depends_on:
- ollama
```

3. Execute: `docker-compose up -d`

C.1.3 Testes

Para executar testes comparativos entre modelos, utilize o comando:

```
docker-compose -f docker-compose.test.yml up
```


Bibliografia

- [1] Docker. <https://www.docker.com/>.
- [2] Flask. <https://flask.palletsprojects.com/>.
- [3] Python. <https://www.python.org/>.
- [4] React. <https://react.dev/>.
- [5] Jinhyuk Lee, Wonjin Yoon, Sungdong Kim, Donghyeon Kim, Sunkyu Kim, Chan Ho So, and Jaewoo Kang. BioBERT: a pre-trained biomedical language representation model for biomedical text mining. *Bioinformatics*, 36(4):1234–1240, 2020. URL <https://academic.oup.com/bioinformatics/article/36/4/1234/5566506>.
- [6] Microsoft. Viva topics: Discover knowledge across your organization. <https://www.microsoft.com/en-us/microsoft-viva/topics>, 2021. Acesso em 2025.
- [7] Ollama. all-minilm embedding model. <https://ollama.com/library/all-minilm>, 2024.
- [8] Ollama. avrsfr-embed model. <https://ollama.com/library/avrsfr-embed>, 2024.
- [9] Ollama. medcpt-article embedding model. <https://ollama.com/oscardp96/medcpt-article>, 2024.
- [10] Ollama. medcpt-query embedding model. <https://ollama.com/oscardp96/medcpt-query>, 2024.
- [11] Ollama. mxbai-embed model. <https://ollama.com/library/mxbai-embed>, 2024.

- [12] Ollama. nomic-embed-text model. <https://ollama.com/library/nomic-embed-text>, 2024.
- [13] Ollama. Plataforma de modelos de embeddings. <https://ollama.com>, 2024.
- [14] Ollama. snowflake-embed2 model. <https://ollama.com/library/snowflake-embed2>, 2024.
- [15] ROSS Intelligence. Artificial intelligence for legal research. <https://rossintelligence.com>, 2021. Acesso em 2025.