UDF – CENTRO DE ENSINO UNIFICADO DO DISTRITO FEDERAL
CURSO DE CIÊNCIA DA COMPUTAÇÃO

JUSTIÇA 4.0 COM IA E BUSCA VETORIAL: ESTEIRA DE RAG PARA AUXILIAR AO PROGRAMA DE MODERNIZAÇÃO DA EXECUÇÃO PENAL DO CNJ/PNUD

GABRIEL OLIVEIRA FARIA – RGM 29615267
PEDRO SAMUEL THIAGO BESERRA VALERIANO – RGM 29697000
ZANDOR DUARTE – RGM 29693829

Brasília – Distrito Federal, Novembro de 2025

GABRIEL OLIVEIRA FARIA – RGM 29615267
PEDRO SAMUEL THIAGO BESERRA VALERIANO – RGM 29697000
ZANDOR DUARTE – RGM 29693829

JUSTIÇA 4.0 COM IA E BUSCA VETORIAL: ESTEIRA DE RAG PARA AUXILIAR AO PROGRAMA DE MODERNIZAÇÃO DA EXECUÇÃO PENAL DO CNJ/PNUD

Trabalho de Conclusão de Curso apresentado ao **Curso de Ciência da Computação** do **UDF – Centro de Ensino Unificado do Distrito Federal**, como requisito parcial para obtenção do título de **Bacharel em Ciência da Computação**.

Orientadora:

Prof.ª Dra. Kerlla de Luz Souza

Coorientador:

Esp. Danrley Willyan da Silva Pereira

Brasília – Distrito Federal, Novembro de 2025

GABRIEL OLIVEIRA FARIA – RGM 29615267
PEDRO SAMUEL THIAGO BESERRA VALERIANO – RGM 29697000
ZANDOR DUARTE – RGM 29693829

JUSTIÇA 4.0 COM IA E BUSCA VETORIAL: ESTEIRA DE RAG PARA AUXILIAR AO PROGRAMA DE MODERNIZAÇÃO DA EXECUÇÃO PENAL DO CNJ/PNUD

Trabalho de Conclusão de Curso apresentado ao **Curso de Ciência da Computação** do **UDF – Centro de Ensino Unificado do Distrito Federal**, como requisito parcial para obtenção do título de **Bacharel em Ciência da Computação**.

Data de aprovação:

A ser definida após a defesa

Banca Examinadora:

A ser definido
Centro Universitário do Distrito Federal

A ser definido
Centro Universitário do Distrito Federal

A ser definido
Centro Universitário do Distrito Federal

Isso vai ser feito pela biblioteca depois da apresentação final

AGRADECIMENTOS

Agradeço, primeiramente a Deus, por mais uma conquista; ao meu orientador, pela dedicação e correções

"Aprender é a única coisa de que a mente nunca se cansa, nunca tem medo e nunca se arrependende."

Leonardo da Vinci, Renascentista e polimata italiano

RESUMO

Este Trabalho de Conclusão de Curso apresenta o desenvolvimento de uma *pipeline* de Geração Aumentada por Recuperação (*Retrieval-Augmented Generation* – RAG) concebida para apoiar consultas sobre informações relacionadas ao Sistema Eletrônico de Execução Unificado (SEEU) do Conselho Nacional de Justiça. O objetivo é propor uma arquitetura que facilite a obtenção de informações de suporte pelos diferentes perfis de usuário, integrando bancos de dados vetoriais a um modelo de linguagem de grande porte (*Large Language Model* – LLM), de modo a fornecer respostas em linguagem natural fundamentadas em documentos originais. A metodologia abrange: (i) coleta automatizada de documentos oficiais, legislação e jurisprudência; (ii) pré-processamento com limpeza e segmentação textual; (iii) vetorização por *embeddings* e indexação em mecanismo de busca semântica; (iv) orquestração, por meio da biblioteca LangChain, entre o motor de recuperação vetorial e o LLM; e (v) disponibilização do serviço por meio de *chatbot* e API REST em ambiente de desenvolvimento. Os resultados esperados incluem ganho de eficiência na recuperação de informações, melhoria da qualidade das respostas e alinhamento às iniciativas Justiça 4.0 e Objetivo de Desenvolvimento Sustentável 16, fortalecendo a transparência e o acesso à Justiça. Conclui-se que a arquitetura proposta oferece base conceitual para a transformação digital da execução penal e constitui fundamento para expansões futuras no âmbito do Poder Judiciário.

Palavras-chaves: execução penal; bancos de dados vetoriais; transformação digital.

ABSTRACT

This Final Undergraduate Thesis presents the development of a Retrieval-Augmented Generation (RAG) pipeline designed to support queries regarding information related to the Unified Electronic Penal Execution System (SEEU) of the National Council of Justice. The objective is to propose an architecture that facilitates the retrieval of support information for different user profiles by integrating vector databases with a Large Language Model (LLM) to provide natural-language responses grounded in original documents. The methodology includes: (i) automated collection of official documents, legislation, and case law; (ii) preprocessing with cleaning and text segmentation; (iii) embedding-based vectorization and indexing in a semantic search engine; (iv) orchestration via the LangChain library between the vector retrieval engine and the LLM; and (v) deployment of the service via chatbot and REST API in a development environment. Expected outcomes include improved information retrieval efficiency, enhanced response quality, and alignment with the Justice 4.0 initiative and UN SDG 16, strengthening transparency and access to justice. It is concluded that the proposed architecture offers a conceptual foundation for the digital transformation of penal execution and provides a basis for future expansions within the Judiciary.

Keywords: penal execution; vector databases; digital transformation.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

ABNT	Associação Brasileira de Normas Técnicas
AACSB	Association to Advance Collegiate Schools of Business
API	Application Programming Interface
BM25	Best Matching 25 (algoritmo de ranking)
CI/CD	Continuous Integration/Continuous Deployment
CKAN	Comprehensive Knowledge Archive Network
CNJ	Conselho Nacional de Justiça
CPU	Central Processing Unit
CSS	Cascading Style Sheets
DOM	Document Object Model
DPR	Dense Passage Retrieval
ETL	Extract, Transform, Load
FAISS	Facebook AI Similarity Search
GPU	Graphics Processing Unit
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IA	Inteligência Artificial
JSON	JavaScript Object Notation
JSONL	JSON Lines
JWT	JSON Web Token
LEP	Lei de Execução Penal
LGPD	Lei Geral de Proteção de Dados
LLM	Large Language Model
ML	Machine Learning
NBR	Norma Brasileira
NLP	Natural Language Processing
OCR	Optical Character Recognition
ODS	Objetivos de Desenvolvimento Sustentável
ONU	Organização das Nações Unidas
PDF	Portable Document Format

PLN	Processamento de Linguagem Natural
PNUD	Programa das Nações Unidas para o Desenvolvimento
RAG	Retrieval-Augmented Generation
REST	Representational State Transfer
SEEU	Sistema Eletrônico de Execução Unificado
SGBD	Sistema Gerenciador de Banco de Dados
STF	Supremo Tribunal Federal
STJ	Superior Tribunal de Justiça
TCC	Trabalho de Conclusão de Curso
TJ-BA	Tribunal de Justiça da Bahia
TJ-CE	Tribunal de Justiça do Ceará
TRF4	Tribunal Regional Federal da 4ª Região
UDF	Centro de Ensino Unificado do Distrito Federal
URL	Uniform Resource Locator
WSGI	Web Server Gateway Interface
XPath	XML Path Language

SUMÁRIO

1 INTRODUÇÃO

Em nível estadual, o Tribunal de Justiça da Bahia (TJ-BA) mantém cerca de 24 000 execuções penais ativas no Sistema Eletrônico de Execução Unificado (SEEU). A Lei de Execução Penal (LEP) exige revisões trimestrais, requisito confirmado pela Súmula 533 do Superior Tribunal de Justiça (STJ); isso representa, na prática, a análise de aproximadamente 8 000 movimentações por mês (cerca de 267 por dia) (Brasil, 1984; Superior Tribunal de Justiça, 2015).

Ainda que a digitalização dos processos reduza o manuseio físico dos autos, o volume de informações a consultar continua elevado. No Mutirão Processual Penal de 2024, por exemplo, foram lançados 18 600 atos em apenas 60 dias—número que ultrapassa a meta estabelecida na Portaria 304/2024 do Conselho Nacional de Justiça (CNJ)—evidenciando a sobrecarga de magistrados e servidores (Tribunal de Justiça da Bahia, 2024; Conselho Nacional de Justiça, 2024). Situação semelhante observou-se no Tribunal de Justiça do Ceará (TJ-CE): a confecção manual de 146 despachos consumiria mais de quatro horas, ao passo que um robô executa cada um em apenas 30 s (Tribunal de Justiça do Ceará, 2023).

Diante desse cenário, este trabalho propõe e desenvolve uma *pipeline* de Geração Aumentada por Recuperação (*Retrieval-Augmented Generation* – RAG) voltada ao apoio a pesquisas jurídicas relacionadas ao SEEU. O objetivo não é substituir a atuação humana na prolação de atos judiciais, mas sim propor uma arquitetura que ofereça: (i) busca semântica sobre a LEP, a Constituição Federal e demais normas correlatas; (ii) resumos e apontamentos normativos que facilitem o entendimento das execuções penais cadastradas; e (iii) sugestões de referências jurisprudenciais e doutrinárias que possam ser utilizadas pelo usuário ao redigir suas próprias decisões.

Essa abordagem devolve tempo às equipes de execução penal, melhora a qualidade das decisões ao tornar a pesquisa jurídica mais precisa e, sobretudo, preserva as garantias processuais. A iniciativa segue a diretriz de modernização judicial sustentada por parcerias entre o Conselho Nacional de Justiça (CNJ) e organismos internacionais—como o Programa das Nações Unidas para o Desenvolvimento (PNUD)—que buscam ampliar a transparência e o acesso à Justiça por meio de soluções digitais e governança inovadora (UNDP, 2024).

A experiência do *e-Government* da Estônia mostra o impacto de infraestrutura digital robusta: 98 % das declarações de imposto passam a ser enviadas on-line em poucos minutos, e o mesmo percentual de empresas é registrado eletronicamente, aumentando a rastreabilidade e reduzindo fraudes (DIVALD, 2021). Guardadas as proporções, o SEEU enfrenta desafio semelhante — consolidar milhares de atos processuais dispersos e assegurar as revisões trimestrais obrigatórias. O caso estoniano indica que automação e padronização digitais podem gerar ganhos análogos de eficiência e transparência, reforçando o valor da *pipeline* RAG proposta.

Apesar dos avanços, a consulta manual a grandes acervos documentais ainda é demorada e sujeita a erros. Pesquisas mostram que bancos de dados vetoriais e *embeddings* reduzem a latência e aumentam a precisão de recuperação (TAIPALUS, 2024; GAO, 2023). Técnicas RAG — que combinam Inteligência Artificial (IA) e *Large Language Models* (LLMs) — despontam como solução para consultas em linguagem natural. Revisões recentes destacam o potencial dessa abordagem (Qwak, 2024; PUJIONO et al., 2024).

Este trabalho apresenta o desenvolvimento de uma *pipeline* RAG concebida para consultas relacionadas ao SEEU, implementada em Python, orquestrada por LangChain, com interface *chatbot* em ambiente de desenvolvimento e empacotada em Docker para facilitar reprodução. A arquitetura proposta permitiria que operadores do Direito realizem buscas intuitivas e integrem a solução a sistemas externos por meio de uma *Application Programming Interface* (API) *Representational State Transfer* (REST) em futuras implementações.

1.1 OBJETIVOS

1.1.1 Objetivo Geral

Desenvolver uma arquitetura de *pipeline* RAG orientada a consultas relacionadas ao SEEU, que permita a recuperação e disponibilização de dados públicos da execução penal por meio de *chatbot* e API REST em ambiente de desenvolvimento, propondo um modelo conceitual que possa contribuir para a modernização dos processos judiciais e ampliar eficiência e transparência em futuras implementações.

1.1.2 Objetivos específicos

1. Coletar e organizar dados públicos relacionados ao SEEU, legislação penal e jurisprudência;
2. Vetorizar esses dados em um banco vetorial (FAISS, OpenSearch ou soluções similares);
3. Conectar o índice vetorial ao LLM selecionado por meio de orquestração via LangChain;
4. Propor arquitetura de *pipeline* RAG com mecanismos de recuperação e geração de respostas;
5. Disponibilizar protótipo de chatbot integrado à *pipeline* em ambiente de desenvolvimento;
6. Implementar uma API REST documentada para acesso à funcionalidade;
7. Preparar o ambiente de empacotamento com Docker para facilitar reprodução;

8. Propor diretrizes para futuras integrações em ambiente de produção.

1.2 TRABALHOS CORRELATOS

Esta seção discute estudos que aplicam RAG a contextos jurídicos ou regulatórios, evidenciando avanços e limitações relevantes para o SEEU.

Edwards (2024) apresenta um *pipeline* RAG que integra grafos de conhecimento — construídos por especialistas e por LLMs — a uma base vetorial, automatizando relatórios de acreditação da *Association to Advance Collegiate Schools of Business* (AACSB). O roteamento de consultas, a decomposição em subconsultas e a síntese automática de respostas reduzem o esforço humano e aumentam a transparência; entretanto, a curadoria desses grafos ainda exige validação manual (EDWARDS, 2024). *Relação com o SEEU*: o sistema proposto neste TCC adota estratégias semelhantes de roteamento e síntese de consultas, porém aplica-as ao domínio jurídico-penal. Enquanto Edwards et al. empregam grafos de conhecimento para estruturar requisitos de acreditação, a presente solução organiza dispositivos legais, súmulas e doutrinas, permitindo recuperação contextualizada de normas da execução penal e facilitando a fundamentação de decisões judiciais.

Pujiono et al. (2024) implementam um chatbot que combina *embeddings* da OpenAI, armazenamento vetorial no Pinecone e geração condicionada à recuperação, respondendo a perguntas sobre normas de agências públicas. O estudo comprova a utilidade do RAG na interpretação de regulamentos, porém não trata acervos processuais volumosos (PUJIONO et al., 2024). *Relação com o SEEU*: o presente trabalho incorpora técnicas de indexação escalável e métricas de cobertura para manipular grande acervo de legislação e doutrinas.

Aquino (2024) descreve um fluxo RAG local para extrair informações estruturadas de documentos de licitação, utilizando *embeddings* BERTimbau, Chroma como *vector store* e LLMs *open source*. O autor relata ganhos de precisão sobre técnicas tradicionais, mas alerta para o elevado custo computacional (AQUINO, 2024). *Relação com o SEEU*: esta pesquisa adota estratégias de compressão e particionamento que reduzem o consumo de recursos, viabilizando a execução em infraestrutura de tribunal estadual sem comprometer a qualidade das respostas.

1.3 SOLUÇÃO PROPOSTA

A solução proposta consiste em uma arquitetura de *pipeline* RAG concebida e desenvolvida em ambiente de prototipagem, organizada nos seguintes módulos conceituais e implementados:

- **Extração e pré-processamento de dados relacionados ao SEEU**: coleta automatizada de documentos oficiais (PDF) e sítios web públicos com informações de suporte;

- **Acesso a acervos legislativos e jurisprudenciais**: recuperação periódica de decisões do STF, STJ, TRF da 4ª Região, legislação penal, Constituição Federal, Código de Processo Penal, súmulas e resoluções em acervos oficiais;
- **Pré-processamento**: limpeza e segmentação textual em *chunks*;
- **Vetorização e indexação**: geração de *embeddings* e armazenamento em base vetorial FAISS;
- **Orquestração**: gerenciamento via LangChain, com estratégia de *fallback* e controle de *prompt*;
- **Interface conversacional (protótipo)**: *chatbot* e API REST documentada em ambiente de desenvolvimento;
- **Camada de validação jurídica**: mecanismos propostos para reduzir alucinações do modelo;
- **Monitoramento e métricas**: estrutura para coleta de tempo de resposta e taxa de erro;
- ***Deployment* conteinerizado**: empacotamento em Docker para facilitar reprodução e futuras implantações;
- **Documentação técnica**: código-fonte comentado, manuais de uso e de desenvolvedor, especificação da API;
- **Mitigação de riscos**: diretrizes para atualização contínua de dependências e tratamento de vulnerabilidades em futuras implementações.

2 FUNDAMENTAÇÃO TEÓRICA E REVISÃO DE LITERATURA

Apresentam-se aqui os conceitos essenciais que embasam este trabalho: bancos de dados vetoriais para armazenamento e consulta de *embeddings* gerados por modelos de *machine learning* (Qwak, 2024); modelos de linguagem de grande porte (*Large Language Models* – LLMs) baseados em arquitetura *Transformer* (LEWIS, 2020; GAO, 2023); o método de Geração Aumentada por Recuperação (RAG), que combina recuperação de documentos e geração de texto para reduzir alucinações (EDWARDS, 2024); e, finalmente, o papel do Programa das Nações Unidas para o Desenvolvimento (PNUD) e sua parceria com o CNJ em iniciativas de transformação digital e acesso à justiça (UNDP, 2025b; UNDP, 2024).

2.1 BANCOS DE DADOS VETORIAIS

2.1.1 Definição

Bancos de dados vetoriais são sistemas especializados em armazenar, indexar e consultar vetores em espaço multidimensional. Esses vetores — denominados *embeddings* — são gerados por modelos de *machine learning* e capturam características semânticas de dados não estruturados, como texto, imagens, áudio e vídeo (Qwak, 2024).

2.1.2 Importância

A busca por similaridade em embeddings é fundamental para diversas aplicações:

- **Sistemas de recomendação** – identificação de itens similares às preferências do usuário;
- **Busca semântica** – consultas que interpretam o significado das palavras, não apenas correspondências exatas;
- **Reconhecimento de padrões** – detecção de faces, objetos ou outros padrões em grandes volumes de dados;
- **Pipelines de IA** – armazenamento eficiente de embeddings utilizados por modelos de *deep learning*.

Tais bases oferecem consultas rápidas, baixa latência e alta escalabilidade — qualidades essenciais em Natural Language Processing (*NLP*), visão computacional e outros domínios que envolvem grandes conjuntos de dados não estruturados (Qwak, 2024).

2.1.3 Análise comparativa de soluções

Elasticsearch: plataforma distribuída e escalável com suporte a busca vetorial pelo `Elastic Vector Search`. Limitação: implementação vetorial ainda pouco madura em alta dimensionalidade.

OpenSearch: *fork* aberto do Elasticsearch com recursos vetoriais nativos e manutenção comunitária. Limitação: requer ajustes finos para consultas complexas.

PostgreSQL + `pgvector`: integra dados relacionais e vetoriais em um mesmo SGBD. Limitação: desempenho inferior em buscas de larga escala.

Milvus: banco vetorial especializado, otimizado para similaridade e escalável a bilhões de vetores. Limitação: maior complexidade de configuração e manutenção.

FAISS: biblioteca de alto desempenho amplamente utilizada em pesquisa. Limitação: não é um SGBD completo, exigindo integração adicional.

Weaviate: código aberto que combina buscas vetoriais e de grafo, permitindo consultas semânticas e relacionais. Limitação: requer *tuning* avançado para desempenho ótimo.

Oracle Vector DB: integração nativa ao ecossistema Oracle, com alta performance e segurança empresarial. Limitação: licenciamento oneroso e menor flexibilidade frente a soluções abertas (Oracle, 2025).

IBM Vector DB: forte integração com ferramentas de IA da IBM, oferecendo recursos robustos de análise vetorial. Limitação: custo elevado e configuração complexa (IBM, 2025).

2.1.4 Conclusão

Nesta seção, foram apresentados os principais conceitos e aplicações dos bancos de dados vetoriais, bem como uma análise comparativa das soluções mais utilizadas no mercado. Verificou-se que a escolha da tecnologia adequada depende do equilíbrio entre desempenho, escalabilidade e requisitos organizacionais. Plataformas especializadas, como Milvus e FAISS, oferecem maior eficiência em buscas de similaridade, enquanto soluções integradas, como PostgreSQL+pgvector, favorecem a unificação de dados em um único Sistema Gerenciador de Banco de Dados (SGBD). O contínuo aprimoramento nas técnicas de indexação e compressão de *embeddings* consolidará ainda mais o papel dos bancos vetoriais em projetos de IA e processamento de dados semiestruturados.

2.2 LARGE LANGUAGE MODELS (LLMS)

2.2.1 Introdução

Os Modelos de LLMs, baseados na arquitetura Transformer (VASWANI, 2017; NAVEEDA, 2024), elevaram o estado da arte em Processamento de Linguagem Natural (PLN), permitindo síntese, tradução e interpretação semântica de documentos em larga escala. Essas redes neurais podem substituir buscas puramente lexicais por consultas semânticas, aumentando a agilidade e a precisão das respostas. A integração de LLMs a bancos de dados vetoriais (TAIPALUS, 2024; Qwak, 2024) reforça essa capacidade, fornecendo resultados contextualizados a partir de extensos acervos documentais.

2.2.2 Aspectos Técnicos

2.1 Pre-Training

O modelo é submetido a um corpus genérico e volumoso para capturar padrões linguísticos amplos, formando uma base de conhecimento diversificada (NAVEEDA, 2024).

2.2 Fine-Tuning

Realiza-se *fine-tuning* com dados do domínio de conhecimento desejado. Estratégias de regularização (e.g., *dropout*, *early stopping*) evitam *overfitting*. A eficácia é medida por precisão, *recall* e F1-score (YUE, 2023; LAI, 2023).

2.3 Validação e Resultados Esperados

Na literatura especializada, trabalhos que aplicam *fine-tuning* de LLMs a domínios específicos reportam avaliações baseadas em: (i) precisão na recuperação de informações; (ii) qualidade das respostas validadas por especialistas; e (iii) eficiência computacional comparada a métodos tradicionais. De modo geral, esses estudos apontam ganhos expressivos de precisão e *recall* quando modelos de linguagem são adaptados a corpora específicos (YUE, 2023; LAI, 2023). Tais métricas servem de referência para futuras avaliações da arquitetura proposta neste trabalho.

2.2.3 Conclusão

A combinação de pré-treinamento e *fine-tuning*, aliada a bancos vetoriais, constitui abordagem inovadora para consultas jurídicas, reduzindo prazos e aumentando a transparência do Judiciário (BELARMINO, 2025; DIVALD, 2021). A integração desses modelos a mecanismos de recuperação documental, conforme descrito na seção seguinte, potencializa a precisão e a confiabilidade das respostas geradas.

2.3 RETRIEVAL-AUGMENTED GENERATION (RAG)

2.3.1 Introdução

O RAG associa a competência de LLMs em gerar texto à recuperação automática de documentos, reduzindo *alucinações* ao fundamentar as respostas em evidências externas verificáveis (LEWIS, 2020; GAO, 2023; EDWARDS, 2024; PUJIONO et al., 2024). LLMs armazenam conhecimento nos parâmetros (*memória paramétrica*); já o RAG adiciona uma *memória não paramétrica* consultável em tempo real, essencial em cenários como o SEEU, cujo acervo documental é volumoso e dinâmico.

2.3.2 Fundamentos

Data retrieval. Consultas e documentos são convertidos em *embeddings*; métodos densos, como o *Dense Passage Retrieval* (DPR), aproximam vetores por similaridade de cosseno ou distância euclidiana, retornando um subconjunto k-relevante (LEWIS, 2020; TAIPALUS, 2024; MAGEIRAKOS et al., 2025).
Content generation. Um modelo *encoder–decoder* (ex.: BART ou T5) concatena os trechos recuperados ao *prompt* e gera a resposta. O treinamento conjunto (Sec. 2.3.3) ensina o *retriever* a apresentar evidências úteis ao gerador (AQUINO, 2024; BELARMINO, 2025).

2.3.3 Pipeline

1. **Ingestion** – extração de fontes estruturadas (bases SQL,Comprehensive Knowledge Archive Network (CKAN)) e não estruturadas (PDF, HTML); limpeza, segmentação em parágrafos e criação de embeddings com modelos como *all-MiniLM*. Objetos ⟨ID, embedding, metadata⟩ são indexados em repositórios vetoriais (FAISS, Pinecone) (Qwak, 2024; TAIPALUS, 2024).

2. **Retrieval** – a consulta é vetorizada e comparada com o índice; top-k documentos são ranqueados. Estratégias *re-rank* com *cross-encoders* ou fusão heurística (ex.: *Reciprocal Rank Fusion*) aumentam precisão (EDWARDS, 2024).

3. **Treinamento conjunto** – ajuste *end-to-end* de *retriever* e *generator* via *maximum-likelihood* ou *policy-gradient*, fazendo o *retriever* maximizar a probabilidade da resposta correta (ZHANG, 2025).

2.3.4 Variantes

- **RAG-Sequence** – para cada um dos k documentos recuperados, o modelo gera uma resposta completa de forma independente. Em seguida, calcula-se a probabilidade de

cada resposta e faz a marginalização, ou seja, a combinação ponderada dessas probabilidades para produzir a resposta final. Esse método garante que cada documento tenha igual oportunidade de influenciar a resposta global, sendo indicado quando se deseja explorar várias interpretações completas antes da decisão final (LEWIS, 2020; EDWARDS, 2024).

- **RAG-Token** – em vez de gerar respostas inteiras por documento, o modelo reavalia a distribuição de probabilidade a cada novo token, permitindo que diferentes documentos contribuam de forma pontual ao longo da geração. Isso amplia a cobertura informativa e mistura evidências de várias fontes, mas requer mecanismos adicionais de coerência para evitar que o texto final fique fragmentado ou inconsistente (ZHANG, 2025).

2.3.5 Desafios e limitações

- **Latência** – cada consulta envolve busca vetorial + geração, podendo ultrapassar limites de tempo real (SHI, 2025).
- **Atualização em tempo real** – garantir que o índice reflita alterações frequentes do corpus demanda pipelines de reingestão contínua (TAIPALUS, 2024).
- **Qualidade da recuperação** – ruído ou pouca cobertura no índice reduz acurácia; técnicas de *negative-sampling* e *hard negatives* no treinamento mitigam o problema (GAO, 2023; SALEMI; ZAMANI, 2024).
- **Coerência textual** – fusão de múltiplas fontes pode gerar redundância ou mudança de estilo; pós-edição automática e penalidades de repetição auxiliam (ZHANG, 2025).

2.3.6 Conclusão

Esta seção apresentou o método RAG, que combina modelos de linguagem de grande porte com recuperação de documentos para reduzir alucinações e fundamentar respostas em evidências verificáveis. Descreveu-se o pipeline de ingestão, recuperação e treinamento conjunto, bem como variantes como RAG-Sequence e RAG-Token. Foram discutidos desafios relativos à latência, atualização em tempo real, qualidade da recuperação e coerência textual. Conclui-se que, apesar das limitações, o RAG representa avanço significativo para aplicações que exigem precisão e atualização dinâmica, sendo promissor para sistemas que trabalham com grandes acervos documentais, como o SEEU.

2.4 PROGRAMA DAS NAÇÕES UNIDAS PARA O DESENVOLVIMENTO (PNUD)

O PNUD é a agência da Organização das Nações Unidas (ONU) responsável por promover o desenvolvimento humano sustentável e erradicar a pobreza em mais de 170 países e territórios (UNDP, 2025b; UNDP, 2025a). Sediado em Nova York, o PNUD oferece suporte técnico e financeiro a políticas públicas voltadas às populações mais vulneráveis.

2.4.1 Objetivos e mandato

O mandato do PNUD abrange quatro eixos centrais:

- **Erradicação da pobreza** – programas para reduzir a pobreza extrema e melhorar as condições de vida;
- **Desigualdade e inclusão social** – políticas que promovem igualdade de oportunidades;
- **Desenvolvimento sustentável** – iniciativas que conciliam o uso de recursos naturais e a proteção ambiental;
- **Governança democrática** – fortalecimento institucional, transparência e participação cidadã.

Tais ações alinham-se à Agenda 2030 e aos Objetivos de Desenvolvimento Sustentável (ODS), sobretudo o ODS 1 (pobreza) e o ODS 10 (redução das desigualdades) (Wikipedia, 2025).

2.4.2 Estrutura e funcionamento

Financiado por contribuições voluntárias de Estados-membros, setor privado e ONGs, o PNUD é chefiado por um administrador indicado pelo Secretário-Geral da ONU e aprovado pela Assembleia Geral (UNDP, 2025a). No Brasil, opera em parceria com governos, sociedade civil e empresas, direcionando projetos que fomentam o desenvolvimento sustentável e reduzem desigualdades (UNDP, 2025b).

2.4.3 Parceria CNJ–PNUD: direitos humanos e acesso à justiça

Em 2025, o CNJ e o PNUD firmaram acordo de cooperação para fortalecer o Poder Judiciário na promoção de direitos humanos, sustentabilidade socioambiental e acesso à justiça por populações vulnerabilizadas (UNDP, 2024). O projeto complementa iniciativas como:

- **Programa Justiça 4.0** – transformação digital do Judiciário brasileiro, ampliando transparência e celeridade processual;

- **Fazendo Justiça** – melhorias nas políticas de privação de liberdade e reintegração social.

A juíza auxiliar Karen Luise destaca que a ação se alinha à Estratégia 2021-2026 do CNJ, priorizando igualdade e acesso jurisdicional. Para o PNUD, a parceria reforça o ODS 16, que visa instituições eficazes e inclusivas (UNDP, 2024). As atividades previstas contemplam:

1. fortalecimento institucional e capacitação de magistrados;
2. diagnósticos situacionais e desenvolvimento de metodologias inclusivas;
3. projetos-piloto focados em crianças e adolescentes em abrigamento, mulheres, pessoas LGBTQIA+, povos indígenas, pessoas em situação de rua, idosos, pessoas com deficiência e grupos vulneráveis por fatores socioambientais ou raciais.

Impacto esperado

O fortalecimento do sistema judiciário — por meio da digitalização, capacitação e práticas inovadoras — tende a ampliar o acesso efetivo à justiça e a reduzir barreiras estruturais. A cooperação CNJ–PNUD, portanto, contribui para o cumprimento dos compromissos internacionais do Brasil relacionados aos ODS, promovendo uma sociedade mais justa e inclusiva (UNDP, 2024).

2.4.4 Conclusão

O PNUD configura-se como agente estratégico na promoção do desenvolvimento humano sustentável e na erradicação da pobreza, atuando em consonância com a Agenda 2030 e os Objetivos de Desenvolvimento Sustentável. A parceria firmada em 2025 entre o CNJ e o PNUD reforça esse compromisso, ao incorporar iniciativas de transformação digital, capacitação institucional e inclusão social no âmbito do Poder Judiciário brasileiro. Espera-se que tais ações ampliem o acesso efetivo à justiça para grupos vulnerabilizados e fortaleçam a governança democrática, contribuindo para o cumprimento das metas internacionais do Brasil e para a construção de uma sociedade mais equânime e participativa.

3 TECNOLOGIAS

Este capítulo apresenta as tecnologias empregadas: linguagens e bibliotecas de Machine Learning(ML)/NLP, indexação vetorial, modelos de linguagem, orquestração de pipeline e infraestrutura de deployment.

3.1 LINGUAGEM E BIBLIOTECAS PRINCIPAIS

- **Python** – linguagem base, com extensas bibliotecas para NLP e ML (Python Software Foundation, 2024);
- **Pandas** – manipulação de dados tabulares e pré-processamento de textos (The Pandas Development Team, 2024);
- **NumPy** – operações vetoriais e matrizes de alto desempenho (HARRIS et al., 2020);
- **Scikit-learn** - para algoritmos para classificação, regressão, clustering e pré-processamento (PEDREGOSA, 2011);

3.2 INDEXAÇÃO E RECUPERAÇÃO VETORIAL

3.2.1 FAISS

FAISS (Facebook AI Similarity Search) é uma biblioteca de código aberto desenvolvida pelo Facebook AI Research (FAIR) para busca de similaridade vetorial de alta performance em larga escala. Especializada em operações eficientes com bilhões de vetores, utiliza técnicas avançadas como quantização de produto (PQ) e índices invertidos (IVF) para acelerar consultas k-NN em até 5x comparado a soluções convencionais. Sua arquitetura é otimizada para paralelismo em CPU/GPU e permite aplicações em tempo real como sistemas de recomendação, clustering de embeddings e RAG. O FAISS foi escolhido para este projeto por sua eficiência, maturidade e ampla adoção na comunidade acadêmica e industrial. Embora não gerencie metadados ou persistência nativamente, sua integração com sistemas externos como arquivos Parquet e bancos SQL permite implementar cenários completos de produção (Facebook Research, 2024; TAIPALUS, 2024).

3.3 MODELOS DE LINGUAGEM

3.3.1 OpenAI GPT

Para este projeto, foi selecionado o modelo de linguagem da OpenAI (GPT) como LLM principal da *pipeline* RAG. Os modelos GPT (Generative Pre-trained Transformer) são baseados na arquitetura Transformer (VASWANI, 2017) e são treinados em vastos

corpora textuais, permitindo geração de texto coerente e contextualizado em linguagem natural. A escolha da OpenAI justifica-se pela sua API estável, documentação robusta, ampla adoção em ambientes de produção e capacidade comprovada de gerar respostas de alta qualidade quando integrada a sistemas de recuperação de informações. A integração com a OpenAI API permite que a *pipeline* RAG envie o contexto recuperado (documentos relevantes) junto ao *prompt* do usuário, resultando em respostas fundamentadas em fontes verificadas e com menor taxa de alucinação.

3.4 FERRAMENTAS DE EXTRAÇÃO E COLETA DE DADOS

3.4.1 Playwright

A coleta automatizada de dados em portais judiciais apresenta desafios técnicos significativos, sobretudo quando o conteúdo é renderizado dinamicamente por JavaScript. Nesse contexto, o Playwright, biblioteca de automação de navegadores desenvolvida pela Microsoft, foi selecionado para os módulos de *scraping* deste projeto. Sua capacidade de controlar navegadores Chromium, Firefox e WebKit de forma programática permite acessar páginas web complexas, executando-as em modo *headless* para extração de conteúdo textual e metadados. Além disso, a biblioteca oferece mecanismos de interação com elementos da página — tais como cliques e preenchimento de formulários — de forma automatizada e resiliente, o que se revelou fundamental para a coleta sistemática de jurisprudência e documentos normativos dos portais do STF, TRF4 e SEEU. Dessa forma, o Playwright constitui alicerce tecnológico para a aquisição consistente e escalável dos dados que alimentam o *pipeline* RAG.

3.4.2 BeautifulSoup

Complementarmente, a análise estrutural dos documentos HTML e XML obtidos requer ferramentas especializadas em *parsing*. BeautifulSoup, biblioteca Python amplamente utilizada nesse domínio, facilita a navegação, busca e modificação da árvore de elementos por meio de seletores CSS ou expressões XPath. Nos scrapers desenvolvidos, o BeautifulSoup é empregado tanto em conjunto com o Playwright — para extração de conteúdo dinâmico — quanto isoladamente, quando as páginas são estáticas. Sua sintaxe intuitiva e robustez na manipulação de HTML malformado tornaram-na ferramenta essencial para o pré-processamento e estruturação de dados coletados, especialmente na extração de campos específicos como títulos, datas, relatores e conteúdo integral de decisões judiciais. Por conseguinte, a combinação dessas duas ferramentas assegura a cobertura adequada de diferentes tipos de fontes documentais.

3.5 ORQUESTRAÇÃO DE PIPELINE

3.5.1 LangChain

A orquestração eficiente de múltiplos componentes de um sistema RAG — incluindo modelos de linguagem, bases de conhecimento e mecanismos de recuperação — constitui desafio arquitetural significativo. Para enfrentá-lo, este projeto adotou LangChain, biblioteca de código aberto especializada no desenvolvimento de aplicações baseadas em LLMs (LangChain, 2024). Por meio dessa ferramenta, torna-se possível encadear chamadas a modelos de linguagem, gerenciar contextos conversacionais e integrar diferentes componentes — tais como *embeddings* e bases de conhecimento vetoriais — de maneira coesa e reprodutível.

Além disso, a biblioteca oferece suporte a diversos provedores de LLMs, como OpenAI e Hugging Face, facilitando a comparação entre modelos e a substituição de componentes sem alterações estruturais na arquitetura. Essa modularidade revela-se especialmente útil em ambientes de pesquisa, nos quais diferentes configurações precisam ser testadas e avaliadas. Ademais, LangChain inclui utilitários para armazenamento de estado, integração com armazenamento vetorial e orquestração de múltiplos *prompts*, permitindo criar fluxos de trabalho sofisticados para agentes conversacionais, assistentes virtuais e sistemas de busca semântica. Desse modo, a escolha do LangChain alinha-se aos objetivos de prototipagem rápida e escalabilidade futura da solução proposta.

3.5.2 Controle de Prompt e Fallback

Em sistemas baseados em modelos generativos, a confiabilidade das respostas depende fundamentalmente da qualidade e clareza das solicitações. Por conseguinte, foi implementado um mecanismo de controle de *prompt* que valida a completude da requisição antes de enviá-la ao LLM. Caso o modelo retorne uma resposta vaga ou incerta, a lógica de *fallback* direciona a consulta a fontes de dados oficiais ou bases de conhecimento verificadas, combinando a flexibilidade dos modelos generativos com a precisão de dados estruturados.

Ademais, o controle de *prompt* permite reformular automaticamente perguntas problemáticas, melhorando a qualidade das interações e reduzindo a taxa de respostas inadequadas. O mecanismo de *fallback*, por sua vez, pode acessar APIs governamentais, documentos normativos ou bases de conhecimento internas sempre que a recuperação vetorial não fornecer conteúdo suficiente. Essa estratégia híbrida assegura consistência e confiabilidade, minimizando riscos de informações errôneas — aspecto crucial em aplicações jurídicas, nas quais a acurácia das informações é imperativa. Portanto, a arquitetura proposta alia a capacidade generativa dos LLMs ao respaldo factual de fontes oficiais, resultando em experiência de usuário mais robusta e confiável.

3.6 TECNOLOGIAS DE INTERFACE E FRONTEND

3.6.1 Nuxt.js

A construção de interfaces web modernas e interativas requer *frameworks* que equilibrem produtividade de desenvolvimento, desempenho em tempo de execução e manutenibilidade de código. Nesse contexto, Nuxt.js foi selecionado como base para a interface de usuário do protótipo de *chatbot* desenvolvido. Conforme a documentação oficial (Nuxt Team, 2024), o Nuxt é um *framework* de código aberto construído sobre Vue.js, orientado para criação de aplicações web *full-stack* com suporte a renderização do lado do servidor (SSR), geração de sites estáticos (SSG) e modo SPA (*Single Page Application*). Além disso, oferece roteamento automático, gerenciamento de estado integrado e facilidade de integração com APIs REST. Sua arquitetura modular e suporte nativo a TypeScript permitem desenvolvimento ágil e escalável de componentes de apresentação, características que se mostraram adequadas aos requisitos de prototipagem rápida do projeto. A escolha do Nuxt justifica-se, portanto, pela combinação de recursos avançados, maturidade do ecossistema e facilidade de integração com os demais componentes da arquitetura proposta.

3.6.2 Vue.js

No cerne do Nuxt.js encontra-se Vue.js, *framework* JavaScript progressivo para construção de interfaces de usuário baseadas em componentes reativos (Vue.js Team, 2024). Segundo a documentação oficial, o Vue adota um modelo declarativo baseado em componentes que permite construir interfaces complexas a partir de unidades reutilizáveis e auto-contidas. Essa arquitetura componentizada facilita a criação de interfaces dinâmicas e interativas, nas quais cada elemento da interface — tal como o campo de entrada de consultas, o painel de exibição de respostas, a listagem de fontes citadas e os controles auxiliares — é encapsulado em um componente independente e reutilizável. A reatividade nativa do Vue.js, baseada em um sistema de reatividade refinado, garante que qualquer alteração no estado da aplicação seja refletida instantaneamente na interface, proporcionando experiência de usuário fluida e responsiva. Tal característica mostra-se especialmente relevante em sistemas conversacionais, nos quais a percepção de imediatez e interatividade influencia diretamente a aceitação pelos usuários. Por conseguinte, a combinação de Vue.js como base e Nuxt.js como camada de abstração proporciona fundamento sólido para o desenvolvimento da interface do protótipo.

3.7 INFRAESTRUTURA E DEPLOYMENT

3.7.1 Docker

A reprodução consistente de ambientes computacionais constitui requisito fundamental para pesquisas em Ciência da Computação e Engenharia de Software. Nesse sentido, Docker — plataforma de conteinerização amplamente adotada — foi selecionado para empacotar e distribuir os componentes da *pipeline* RAG desenvolvida (Docker Documentation, 2024). Cada contêiner encapsula a aplicação juntamente com suas bibliotecas, dependências e configurações, garantindo consistência entre diferentes ambientes de desenvolvimento, teste e eventual produção.

Além da reprodução, o Docker facilita o versionamento por meio de imagens imutáveis, permitindo rastrear alterações e reverter para versões anteriores quando necessário. A ferramenta também simplifica a escalabilidade de serviços, integrando-se a orquestradores como Docker Compose ou Kubernetes para gerenciar múltiplos contêineres. Ademais, a existência de um repositório público de imagens (Docker Hub) acelera o desenvolvimento ao disponibilizar soluções prontas. Dessa forma, a conteinerização com Docker não apenas atende aos requisitos de reprodução científica, mas também prepara o projeto para futuras implantações em ambientes de produção.

3.7.2 Kubernetes

Embora a conteinerização resolva questões de reprodução, a orquestração de múltiplos contêineres em escala requer plataformas especializadas. Kubernetes, sistema de código aberto amplamente utilizado para orquestração de contêineres, foi considerado na arquitetura proposta como caminho de evolução futura do projeto (Kubernetes Documentation, 2025). Esse sistema permite implantar, escalar e operar aplicações em *clusters* de servidores, agrupando contêineres em unidades denominadas *pods* e facilitando o gerenciamento de cargas de trabalho.

Os componentes principais do Kubernetes — incluindo *kubelet*, *API Server*, *Scheduler* e *Controller Manager* — zelam pelo estado desejado do *cluster*, permitindo escalar serviços automaticamente com base em métricas de uso e garantir alta disponibilidade. Por meio da abstração de serviços e *deployment controllers*, torna-se possível realizar atualizações contínuas e *rolling updates* sem interrupção de serviço. Além disso, a plataforma oferece mecanismos de descoberta de serviço, balanceamento de carga e armazenamento persistente, características essenciais para aplicações em produção. Assim, embora o protótipo atual utilize Docker de forma isolada, a arquitetura foi concebida de modo a facilitar migração futura para ambientes orquestrados por Kubernetes.

3.7.3 FastAPI como Framework da API REST

FastAPI é um *framework* web moderno e de alta performance para construção de APIs em Python, baseado em anotações de tipo (*type hints*) e no padrão ASGI (RAMíREZ, 2024). Conforme sua documentação oficial, o FastAPI foi projetado para ser rápido — tanto em desempenho de execução quanto em velocidade de desenvolvimento — oferecendo validação automática baseada em esquemas Python, geração automática de documentação interativa e suporte nativo a operações assíncronas. Neste projeto, o FastAPI foi escolhido como *framework* principal para implementação da API REST que expõe a *pipeline* RAG. Suas principais vantagens incluem: (i) validação automática de dados de entrada e saída com base em *schemas* Pydantic, reduzindo erros e aumentando a robustez do código; (ii) documentação interativa automática (Swagger/OpenAPI e ReDoc), facilitando testes e integração; (iii) suporte nativo a operações assíncronas, permitindo maior concorrência e eficiência no tratamento de requisições; e (iv) desempenho comparável a *frameworks* como Node.js e Go, aspecto relevante para aplicações que demandam baixa latência. O FastAPI foi utilizado para implementar os *endpoints* de consulta, recuperação de metadados de *clusters*, geração de respostas e integração com o modelo de linguagem da OpenAI, garantindo escalabilidade e manutenção simplificada do código.

4 METODOLOGIA

Este capítulo apresenta a metodologia empregada para o desenvolvimento da *pipeline* RAG proposta, detalhando sua arquitetura conceitual e de implementação, os processos de coleta e tratamento de dados, a indexação vetorial, a orquestração dos componentes e a interface de usuário desenvolvida em ambiente de prototipação. A abordagem metodológica adotada combina revisão bibliográfica, desenvolvimento experimental de protótipo e proposição de arquitetura escalável para futuras implementações em ambiente de produção.

4.1 ARQUITETURA DA *PIPELINE*

A arquitetura da *pipeline* RAG proposta e desenvolvida neste trabalho organiza-se em quatro estágios principais: ingestão de dados, pré-processamento e segmentação textual, indexação vetorial e geração de respostas. Cada estágio foi concebido de forma modular, permitindo flexibilidade na escolha de tecnologias e facilitando manutenção e escalabilidade em futuras implementações. A arquitetura geral, ilustrada na Figura 4.1, demonstra o fluxo de dados desde a coleta até a geração de respostas fundamentadas em fontes oficiais.

4.1.1 Coleta Automatizada de Dados (Fontes e Scrapers)

A coleta de dados é realizada por meio de módulos de extração automatizada (scrapers) desenvolvidos em Python, utilizando Playwright para automação de navegador e BeautifulSoup para parsing de HTML. Os dados são coletados das seguintes fontes oficiais:

- **Supremo Tribunal Federal (STF)**: extração de acórdãos, decisões monocráticas e despachos organizados por artigo do Código Penal, com metadados como relator, partes, legislação citada e texto integral;
- **Superior Tribunal de Justiça (STJ)**: coleta de súmulas, decisões e jurisprudência temática relacionada à execução penal e direito processual penal;
- **Tribunal Regional Federal da 4ª Região (TRF4)**: extração de decisões regionais com metadados estruturados (número do processo, tipo de documento, datas, relator);
- **Sistema Eletrônico de Execução Unificado (SEEU) – Documentação de suporte**: coleta de manuais, orientações técnicas, portarias e instruções normativas publicadas nos portais oficiais do CNJ relacionados ao SEEU.

Os scrapers implementam mecanismos de resiliência como políticas de *retry* com *backoff* exponencial, tratamento de erros de rede, validação de completude dos dados extraídos e registro estruturado de logs. A saída de cada scraper é padronizada em formato JSON Lines (JSONL), com um documento por linha, facilitando processamento incremental

e auditoria. O agendamento das coletas é realizado via *cron jobs* ou Airflow, permitindo atualizações periódicas da base de conhecimento.

4.1.2 Estrutura e Esquema de Saída em JSON Lines

Os documentos extraídos pelos scrapers são armazenados em arquivos JSON Lines (`.jsonl`), nos quais cada linha representa um documento completo em formato JSON. Essa estrutura facilita processamento incremental, append de novos documentos e compatibilidade com ferramentas de processamento distribuído.

O esquema de saída adotado segue a seguinte estrutura:

```
{
  "id": "único identificador (hash ou UUID)",
  "cluster_name": "identificador temático (ex: 'STF_Art_121')",
  "title": "título ou ementa resumida",
  "content": "texto integral da decisão ou documento",
  "url": "URL de origem no portal oficial",
  "metadata": {
    "relator": "nome do relator (quando aplicável)",
    "data_julgamento": "data em formato ISO 8601",
    "legislacao_citada": ["lista de artigos citados"],
    "tipo_documento": "acórdão | decisão | portaria | manual",
    "tribunal": "STF | STJ | TRF4 | CNJ",
    "artigo_cp": "artigo do Código Penal associado (se aplicável)"
  }
}
```

Esse esquema padronizado permite que todos os documentos, independentemente da fonte, sejam processados pelo mesmo pipeline de vetorização e indexação, garantindo uniformidade na recuperação semântica. Os metadados enriquecem as consultas, possibilitando filtragem por tribunal, tipo de documento ou artigo legal durante a fase de recuperação.

4.1.3 Engenharia de *Embeddings* e Indexação Vetorial

A engenharia de *embeddings* constitui etapa crítica da *pipeline* RAG, responsável por transformar textos em representações vetoriais densas que capturam relações semânticas. O processo é composto pelas seguintes etapas:

4.1.3.1 Entrada e Pré-processamento

Cada documento em formato JSONL é submetido ao módulo de chunking (`chunking.py`), que divide o texto em segmentos menores para otimizar a recuperação. A estratégia de divisão segue a hierarquia:

1. Divisão por parágrafos (delimitador `\n\n`);
2. Se os parágrafos forem muito longos, divisão por sentenças;
3. Como *fallback*, aplicação de janela deslizante por caracteres com sobreposição configurável.

Cada *chunk* recebe metadados adicionais: `original_id`, `chunk_index`, `char_start`, `char_end`, `is_chunk` e um identificador único no formato `<doc_id>_chunk_<i>`. Os parâmetros `CHUNK_SIZE` (tamanho máximo em caracteres) e `CHUNK_OVERLAP` (sobreposição entre chunks adjacentes) são configuráveis e controlados em `config.py`.

4.1.3.2 Configuração e Carregamento do Modelo

O modelo de *embeddings* utilizado é baseado na biblioteca `sentence-transformers`. O carregamento é realizado como singleton por meio da função `load_model()`, evitando recargas repetidas durante execução. Os parâmetros principais incluem:

- `EMBEDDING_MODEL`: identificador do modelo (ex.: `all-MiniLM-L6-v2`);
- `EMBEDDING_DIM`: dimensão dos vetores gerados;
- `NORMALIZE_EMBEDDINGS`: flag para normalização L2 dos vetores.

4.1.3.3 Geração de Embeddings

A função central `encode_texts(texts: List[str])` recebe uma lista de textos e retorna uma matriz de *embeddings* com forma `(N, dim)` e tipo `np.float32`, garantindo compatibilidade com FAISS. Quando `NORMALIZE_EMBEDDINGS` está ativado, os vetores são normalizados para norma unitária, permitindo que o produto interno corresponda à similaridade de cosseno.

4.1.3.4 Indexação com FAISS

Durante a indexação, são coletados os textos dos documentos processados e gerados os respectivos vetores. Se o índice FAISS não existir, é criado um `IndexFlatIP(dimension)` (produto interno) envolvido por `IndexIDMap2` para mapear IDs customizados. Os IDs internos são calculados por hash do identificador do documento: `hash(doc_id) % (2**31 - 1)`. Os vetores são adicionados ao índice com `add_with_ids(vectors, ids)`.

4.1.3.5 Busca e Recuperação

Durante uma consulta, o texto da *query* é convertido em vetor de *embedding* (garantido como matriz 2D) e submetido ao índice FAISS com `_index.search(query_vector, k)`, retornando os k vetores mais próximos. Como o índice utiliza produto interno e os vetores estão normalizados, a busca equivale à similaridade de cosseno. Os IDs internos retornados são mapeados de volta aos metadados originais dos documentos, permitindo recuperação do texto completo e informações contextuais.

4.1.3.6 Persistência de Dados

O índice FAISS é salvo em disco como `index.faiss` no caminho configurado. Os metadados (mapeamento de ID interno para campos do documento) são salvos em formato Parquet (`metadata.parquet`) para compatibilidade com PyArrow e eficiência de leitura.

4.1.3.7 GPU e Fallback

A lógica de `maybe_to_gpu` permite mover o índice para GPU se `USE_FAISS_GPU` estiver habilitado e FAISS com suporte GPU estiver disponível. Em caso de falha, o sistema reverte automaticamente para CPU, garantindo funcionamento em ambientes sem aceleração gráfica.

4.1.3.8 Regras Operacionais

- Todos os vetores são armazenados como `float32` com dimensão obtida do modelo;
- O chunking com sobreposição preserva contexto nas bordas dos segmentos, melhorando recuperação;
- IDs por hash são eficientes, mas apresentam pequeno risco de colisões (mitigável com hash criptográfico explícito);
- Metadados do modelo (nome, versão, dimensão) devem ser salvos junto aos dados para facilitar auditoria e reindexação.

Este processo garante que a *pipeline* RAG disponha de um índice vetorial eficiente, escalável e rastreável, fundamental para a qualidade das respostas geradas pelo sistema.

4.1.4 Indexação e Armazenamento

- Indexação dos *embeddings* em OpenSearch ou FAISS, com metadados (origem, data, posição no documento);
- Definição de métricas de similaridade (*cosine similarity*) e *thresholds* de corte.

4.1.5 Orquestração e Consulta

Implementação utilizando LangChain:

- Conversão da consulta em *embedding*;
- Recuperação dos *top-k chunks* mais relevantes;
- Geração da resposta pelo LLM, mesclando múltiplas fontes quando necessário (abordagem RAG-Token).

4.1.6 Desenvolvimento do *Chatbot* e da API

- *Chatbot* baseado em WebSocket para interação síncrona;
- *Endpoints* RESTful em Flask para consultas e coleta de *feedback* de usabilidade.

4.1.7 Proposta de Avaliação e Métricas

A arquitetura desenvolvida contempla mecanismos que permitirão, em futuras implementações, realizar:

- Cálculo de métricas de recuperação (precisão, *recall* e F_1) em conjuntos de questões de *benchmark* previamente definidos;
- Ensaios de usabilidade qualitativos com operadores do Direito, avaliando intuitividade, confiança e adequação ao fluxo de trabalho real.

Tais avaliações, embora não realizadas no escopo deste trabalho, constituem etapa essencial para validação em ambiente de produção e são recomendadas como trabalhos futuros.

4.1.8 Diretrizes para MLOps e Monitoramento

A arquitetura proposta inclui diretrizes para implementação futura de:

- *Pipelines* de CI/CD para *build* e *deploy* automáticos;
- Monitoramento de latência e acurácia, com alertas para degradação de desempenho;
- Mecanismo de *feedback loop* para re-treinamento periódico com dados de uso real.

Tais componentes são fundamentais para operação em ambiente de produção e foram considerados na concepção arquitetural, embora sua implementação completa seja proposta para trabalhos futuros.
A Figura 4.1 apresenta a arquitetura geral da *pipeline* RAG, sintetizando os componentes descritos nas subseções anteriores e evidenciando o fluxo de dados desde a coleta até a geração de respostas.

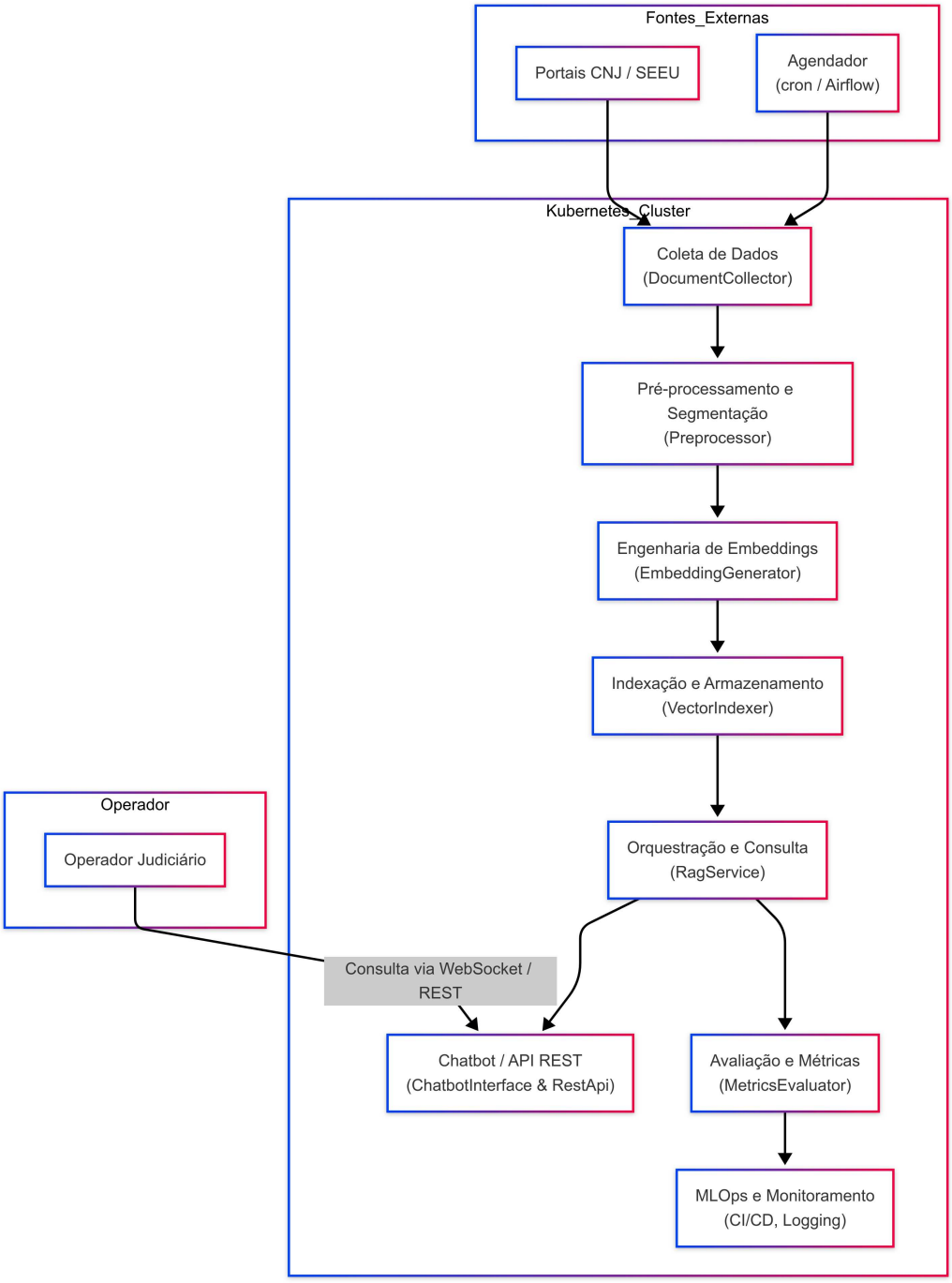

Figura 4.1 – Arquitetura geral da *pipeline* RAG. Fonte: elaborado pelos autores.

4.2 DIAGRAMA DE CASO DE USO

Os diagramas de casos de uso são representações visuais que descrevem as interações entre os usuários (atores) e o sistema, identificando as funcionalidades oferecidas e os relacionamentos entre elas. Segundo Edwards (2024), diagramas de casos de uso auxiliam na especificação de requisitos funcionais, facilitando a comunicação entre as partes interessadas e servindo como base para o desenvolvimento e a validação do sistema. No contexto deste trabalho, o diagrama de casos de uso ilustra como operadores judiciários, administradores e componentes externos interagem com a *pipeline* RAG, evidenciando os principais fluxos de consulta, configuração, coleta de dados, processamento e recuperação de informações.

Conforme apresentado na Figura 4.2, o sistema organiza os casos de uso em torno de quatro atores principais: Operador Judiciário, Administrador, Sistema (ator externo que dispara processos automatizados) e os módulos internos que compõem a *pipeline*.

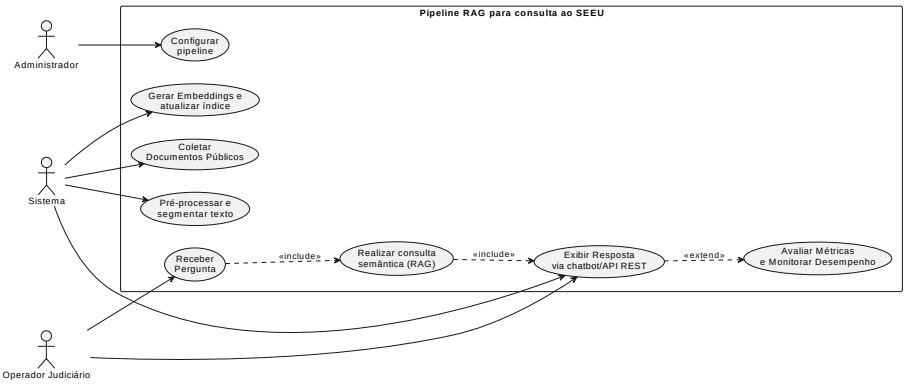

Figura 4.2 – Diagrama de Caso de Uso da *Pipeline* RAG para consulta ao SEEU. Fonte: elaborado pelos autores.

O diagrama da Figura 4.2 apresenta os atores e casos de uso principais da pipeline RAG para o SEEU:

- **Operador Judiciário**: inicia a consulta via “Receber Pergunta”.
- **Administrador**: configura parâmetros da pipeline em “Configurar Pipeline”.
- **Sistema**: atua como ator externo encarregado de disparar e orquestrar os processos de “Coletar Documentos Públicos”, “Pré-processar e segmentar texto”, “Gerar Embeddings e atualizar índice” e “Realizar Consulta Semântica (RAG)”.
- **Include** (setas obrigatórias): indicam os casos de uso que são sempre invocados no fluxo principal.
- **Extend** (setas opcionais): representam funcionalidades adicionais, como a “Avaliar Métricas e Monitorar Desempenho”, que estende o caso de uso “Exibir Resposta via chatbot/API REST”.

A organização dos casos de uso em relacionamentos de inclusão (*include*) e extensão (*extend*) permite identificar dependências obrigatórias e funcionalidades complementares. Os casos de uso de inclusão garantem que operações fundamentais, como a consulta semântica e a atualização da base de conhecimento, sejam sempre executadas quando acionadas por casos de uso superiores. Já as extensões representam comportamentos opcionais que agregam valor ao sistema sem impactar o fluxo principal, como a coleta de métricas de desempenho e a geração de *logs* para auditoria. Essa estrutura modular facilita a manutenção, a evolução do sistema e a rastreabilidade dos requisitos ao longo do ciclo de desenvolvimento.

4.3 ESPECIFICAÇÃO DE CASOS DE USO

No item anterior, foi apresentado o Diagrama de Caso de Uso do sistema. A seguir serão detalhados cada um dos casos de uso, explicando as suas interações com os atores:

4.3.0.1 UC-01 – Enviar Pergunta

O *Operador Judiciário* envia sua dúvida à interface de chatbot para iniciar a consulta semântica.

Tabela 4.1 – Especificação do Caso de Uso UC-01 – Enviar Pergunta

Nome do caso de uso	UC-01 – Enviar Pergunta
Ator Principal	Operador Judiciário
Resumo	Registrar e validar a pergunta do usuário.
Pré-Condições	Interface de chatbot online.
Pós-Condições	Pergunta armazenada e evento emitido para *UC-07*.
Fluxo Principal	
1. Operador acessa a interface. 2. Digita a pergunta e clica em *Enviar*. 3. Sistema valida formato/tamanho. 4. Sistema grava a pergunta e confirma recebimento.	
Fluxo Alternativo	
3a. Pergunta inválida → sistema exibe erro e retorna ao passo 2.	

4.3.0.2 UC-02 – Configurar Pipeline

O *Administrador* define parâmetros e módulos da pipeline RAG.

Tabela 4.2 – Especificação do Caso de Uso UC-02 – Configurar Pipeline

Nome do caso de uso	UC-02 – Configurar Pipeline
Ator Principal	Administrador
Resumo	Ajustar fontes de coleta, periodicidade e parâmetros de indexação.
Pré-Condições	Credenciais válidas de administrador.
Pós-Condições	Parâmetros persistidos e *UC-03 – Atualizar Base de Conhecimento* apto a ser agendado.
Fluxo Principal	
1. Administrador acessa o painel de configuração. 2. Define fontes, cronograma e opções de pré-processamento e indexação. 3. Salva alterações; sistema valida e aplica as configurações.	
Fluxo Alternativo	
2a. Valor inválido → sistema rejeita, exibe mensagem e retorna ao passo 2.	

4.3.0.3 UC-03 – Atualizar Base de Conhecimento

Caso de uso de alto nível que executa todo o *ETL* para manter o índice vetorial sempre atualizado.

Tabela 4.3 – Especificação do Caso de Uso UC-03 – Atualizar Base de Conhecimento

Nome do caso de uso	UC-03 – Atualizar Base de Conhecimento
Ator Principal	Administrador
Relações	«include» UC-04, UC-05 e UC-06
Resumo	Orquestrar a coleta, pré-processamento e (re)indexação dos dados.
Pré-Condições	Parâmetros de pipeline configurados (UC-02).
Pós-Condições	Índice vetorial contém todo o conteúdo recém-coletado.
Fluxo Principal	
1. Administrador dispara atualização ou CronJob executa no agendamento. 2. Sistema aciona *UC-04 – Coletar Documentos Públicos.* 3. Sistema aciona *UC-05 – Pré-processar e Segmentar Texto.* 4. Sistema aciona *UC-06 – Gerar Embeddings e Atualizar Índice.* 5. Resultado agregado é registrado e notificado ao Administrador.	

4.3.0.4 UC-04 – Coletar Documentos Públicos

Extrai automaticamente documentos dos portais SEEU/CNJ.

Tabela 4.4 – Especificação do Caso de Uso UC-04 – Coletar Documentos Públicos

Nome	UC-04 – Coletar Documentos Públicos
Ator Principal	Sistema (Crawler)
Resumo	Baixar PDFs/HTML/JSON dos portais e armazenar cópias brutas.
Pré-Condições	Fontes externas acessíveis.
Pós-Condições	Documentos brutos salvos para pré-processamento.
Fluxo Principal	
1. Crawler inicia sessão nos portais. 2. Pesquisa conteúdos novos. 3. Faz download dos arquivos. 4. Verifica integridade e armazena em repositório bruto.	
Fluxo Alternativo	
1a. Falha de rede → reprograma tentativa e registra log. 2a. Documento corrompido → descarta e alerta administrador.	

4.3.0.5 UC-05 – Pré-processar e Segmentar Texto

Converte PDFs em texto limpo e divide em *chunks*.

Tabela 4.5 – Especificação do Caso de Uso UC-05 – Pré-processar e Segmentar Texto

Nome	UC-05 – Pré-processar e Segmentar Texto
Ator Principal	Sistema (Pre-Processor)
Resumo	Extrair texto, limpar ruídos e segmentar em *chunks*.
Pré-Condições	Documentos brutos disponíveis.
Pós-Condições	*Chunks* prontos para vetorização.
Fluxo Principal	
1. Extrai texto de cada PDF/HTML. 2. Remove cabeçalhos, rodapés e formatação. 3. Separa texto em *chunks* de 500–1000 tokens. 4. Armazena *chunks* para o Embedding Service.	
Fluxo Alternativo	
1a. Falha na extração → registra alerta; prossegue com próximos arquivos.	

4.3.0.6 UC-06 – Gerar Embeddings e Atualizar Índice

Converte *chunks* em vetores e atualiza o índice vetorial.

Tabela 4.6 – Especificação do Caso de Uso UC-06 – Gerar Embeddings e Atualizar Índice

Nome	UC-06 – Gerar Embeddings e Atualizar Índice
Ator Principal	Sistema (Embedding Service)
Resumo	Gerar embeddings e fazer *upsert* no Vector DB.
Pré-Condições	*Chunks* pré-processados disponíveis.
Pós-Condições	Índice vetorial atualizado.
Fluxo Principal	
1. Seleciona *chunks* não indexados. 2. Calcula embedding com modelo pré-treinado. 3. Envia vetor + metadados ao Vector DB. 4. Recebe confirmação (*ACK*).	
Fluxo Alternativo	
1a. Falha no modelo → reprocessa *chunk*; registra erro.	

4.3.0.7 UC-07 – Realizar Consulta Semântica (RAG)

Executa a recuperação de contexto e geração de resposta via LLM.

Tabela 4.7 – Especificação do Caso de Uso UC-07 – Realizar Consulta Semântica (RAG)

Nome	UC-07 – Realizar Consulta Semântica (RAG)
Ator Principal	Sistema (RAG Engine)
Resumo	Vetorizar a pergunta, recuperar top-k documentos e gerar resposta.
Pré-Condições	Pergunta armazenada (UC-01) e índice vetorial online.
Pós-Condições	Resposta gerada e log gravado.
Fluxo Principal	
1. Converte pergunta em vetor. 2. Recupera top-k *chunks* relevantes. 3. Cria *prompt* com contexto + pergunta. 4. Chama LLM e obtém resposta. 5. Formata resposta e grava log.	
Fluxo Alternativo	
1a. Índice indisponível → retorna erro para UC-08.	

4.3.0.8 UC-08 – Exibir Resposta ao Usuário

Entrega a resposta ao Operador e registra entrega.

Tabela 4.8 – Especificação do Caso de Uso UC-08 – Exibir Resposta ao Usuário

Nome	UC-08 – Exibir Resposta ao Usuário
Ator Principal	Operador Judiciário
Relações	«extend» UC-09 – Avaliar Métricas
Resumo	Formatar e entregar a resposta via WebSocket ou REST.
Pré-Condições	Resposta gerada no UC-07.
Pós-Condições	Resposta entregue e log de entrega salvo.
Fluxo Principal	
1. Sistema formata resposta (HTML/JSON). 2. Envia pelo canal apropriado. 3. Recebe *ACK* e registra status.	

4.3.0.9 UC-09 – Avaliar Métricas e Monitorar Desempenho

Calcula métricas (precisão, recall, latência) e dispara alertas.

Tabela 4.9 – Especificação do Caso de Uso UC-09 – Avaliar Métricas e Monitorar Desempenho

Nome	UC-09 – Avaliar Métricas e Monitorar Desempenho
Ator Principal	Sistema (Metrics Service)
Ator Secundário	Administrador
Resumo	Agregar logs, calcular métricas e atualizar dashboards.
Pré-Condições	Logs de UC-07 e UC-08 disponíveis.
Pós-Condições	Painel atualizado; alertas enviados se limites excedidos.
Fluxo Principal	
1. Coleta dados de log/telemetria. 2. Calcula precisão, recall, F1 e latência. 3. Atualiza Painel Grafana/Prometheus. 4. Se métrica fora do SLA, dispara alerta ao Administrador.	

4.3.0.10 Descrição

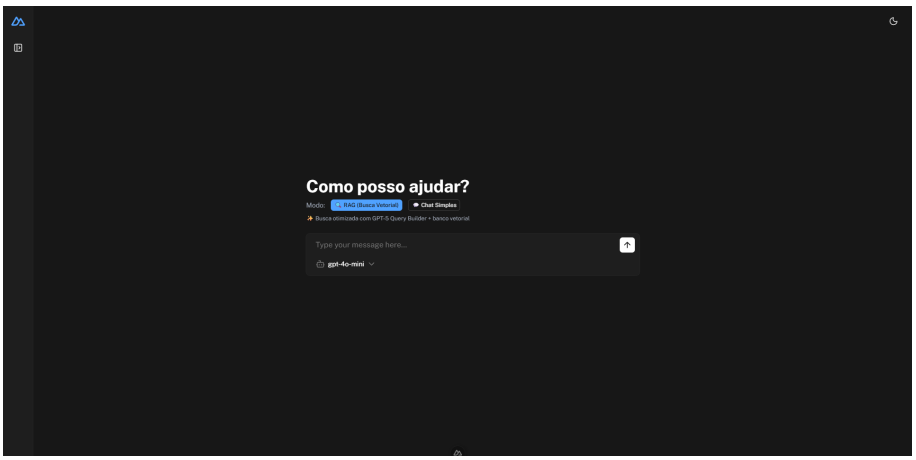

Figura 4.3 – Tela inicial da aplicação (Home). Fonte: elaborado pelos autores.

Conforme ilustrado na Figura 4.3, a tela inicial tem como objetivo permitir que o operador jurídico registre sua pergunta de forma direta e intuitiva. Este é o ponto de entrada para a consulta semântica baseada em documentos do SEEU. Ao submeter a pergunta, o sistema realiza o roteamento interno, executando as etapas de recuperação vetorial e geração de resposta fundamentada por meio do modelo de linguagem integrado.

A funcionalidade central desta interface é a captação da pergunta e o acionamento da pipeline de RAG. O sistema registra a consulta, armazena informações relevantes como o usuário, horário da solicitação e a resposta gerada, contribuindo também para fins de auditoria e análise posterior de métricas.

4.3.0.11 Comandos da tela (botões) – Tela inicial

Tabela 4.10 – Quadro – Comandos da tela (botões) – Tela inicial

Item	Comando	Ação
1	RAG (Busca Vetorial)	Alterna o modo de busca para RAG, permitindo consultas com recuperação de documentos indexados.
2	Chat Simples	Alterna o modo de busca para Chat simples, sem recuperação de documentos.
3	Botão Enviar	Submete a pergunta digitada pelo usuário, validando se o campo não está vazio e executando a busca no modo selecionado (RAG ou Chat).
4	Copiar	Copia o texto da resposta exibida para a área de transferência do sistema.
5	Cards de Resultados	Exibe as citações e documentos recuperados, permitindo visualizar as fontes utilizadas na geração da resposta.

Fonte: Elaborado pelos autores.

4.3.0.12 Campos da tela – Tela inicial

Tabela 4.11 – Quadro – Campos da tela – Tela Inicial

Item	Nome do Campo	Tipo	Tamanho	Máscara	Obrigatório	Valor Padrão	Editável	Visível
1	Campo de entrada de pergunta	Alfanumérico	N/A	Não	Sim	Vazio	Sim	Sim
2	Seleção de Modelo de Linguagem	Seleção	N/A	Não	Não	Conforme configuração	Sim	Sim

Fonte: Elaborado pelos autores.

4.3.0.13 Observações (Tela inicial)

O campo de entrada de texto permite ao usuário inserir sua pergunta em linguagem natural. O tamanho máximo do texto não é definido explicitamente na interface, sendo indicado como "N/A" (não aplicável) na Tabela 4.11. A validação de obrigatoriedade é realizada no momento do envio, impedindo o envio de consultas vazias.

A seleção do modelo de linguagem permite ao usuário escolher entre diferentes modelos disponíveis no sistema. O valor padrão é definido pela configuração do sistema e pode variar conforme a instalação. A lista de modelos é carregada dinamicamente a partir da configuração centralizada da aplicação.

4.4 MÓDULO DE INDEXAÇÃO VETORIAL (DBVECTOR)

4.4.1 Objetivo e Escopo

A recuperação eficiente de informações em grandes acervos documentais constitui desafio fundamental em sistemas de recuperação de informação, especialmente quando aplicados ao domínio jurídico, caracterizado por vocabulário especializado e alta densidade informacional. Nesse contexto, o módulo DBVECTOR foi concebido e desenvolvido para prover uma camada especializada de indexação semântica, responsável pela construção, armazenamento e consulta de representações vetoriais (*embeddings*) de documentos textuais, com ênfase em documentos jurídicos relacionados ao SEEU, legislação penal e jurisprudência.

O propósito central desse módulo é permitir a recuperação eficiente dos documentos mais relevantes para uma consulta formulada em linguagem natural, servindo de alicerce para o *pipeline* de Recuperação Aumentada por Geração (RAG). Para tanto, o DBVECTOR articula os seguintes objetivos específicos: (i) receber documentos previamente estruturados em formato textual, acompanhados de metadados enriquecidos; (ii) realizar pré-processamento textual e geração de *embeddings* por meio de modelos de linguagem especializados; (iii) construir e manter índices vetoriais em *backends* especializados, notadamente FAISS e, de forma opcional, OpenSearch; (iv) expor operações de consulta que

aceitem *queries* em linguagem natural e retornem documentos ordenados por similaridade semântica; e (v) disponibilizar mecanismos de avaliação da qualidade de recuperação e do desempenho do sistema, permitindo aprimoramentos contínuos.

4.4.2 Arquitetura Lógica e Componentes Principais

A arquitetura lógica do DBVECTOR adota organização em camadas, segregando responsabilidades relacionadas à geração de vetores, ao armazenamento, à consulta e à avaliação. Essa separação de responsabilidades, fundamentada em princípios de engenharia de software, favorece a modularidade, extensibilidade e manutenibilidade do sistema. De forma sintética, destacam-se os seguintes componentes principais:

- **Camada de tratamento de dados**: responsável por limpar, normalizar e, quando necessário, segmentar o texto dos documentos, bem como padronizar campos de metadados antes da vetorização. Essa etapa é crucial para garantir a qualidade dos *embeddings* subsequentes;
- **Camada de *embeddings***: encapsula a integração com modelos de linguagem — sejam eles executados localmente ou acessados remotamente via API — para geração de *embeddings* a partir de trechos de texto, produzindo vetores numéricos em ponto flutuante que capturam relações semânticas;
- **Camada de armazenamento vetorial**: implementa o armazenamento persistente e a busca aproximada por similaridade, com suporte a *backends* especializados como FAISS e, alternativamente, OpenSearch com capacidades vetoriais. A escolha do *backend* influencia diretamente o desempenho e a escalabilidade do sistema;
- **Camada de API e serviços**: expõe interface de consulta (API) que recebe consultas textuais, aciona a geração de *embeddings* da *query*, executa a busca vetorial e retorna os documentos mais similares juntamente com seus metadados enriquecidos, em formato estruturado para consumo pelos demais componentes da *pipeline*;
- **Camada de avaliação**: agrega *scripts* e procedimentos voltados à avaliação de qualidade — por meio de métricas de recuperação como precisão, *recall* e F_1 — e de desempenho das operações de indexação e consulta, permitindo aprimoramentos contínuos.

Essa organização arquitetural permite que alterações em um componente específico — tais como substituição do modelo de *embeddings* ou troca do *backend* de armazenamento — sejam realizadas com impacto reduzido nos demais módulos. Por conseguinte, favorece-se a extensibilidade do sistema e facilita-se sua manutenção evolutiva, características essenciais para projetos de pesquisa e desenvolvimento em ambientes dinâmicos.

4.4.3 Fluxo de Processamento e Consulta

O fluxo lógico de dados no DBVECTOR pode ser descrito em duas fases principais: *indexação* e *consulta*.

Na fase de **indexação**, o sistema recebe como entrada documentos em formato estruturado, normalmente em arquivos JSON Lines (`.jsonl`), contendo texto e metadados. A partir dessa entrada, o fluxo segue as etapas:

1. **Tratamento**: o texto é submetido a procedimentos de limpeza e normalização, que podem incluir remoção de caracteres indesejados, padronização de espaçamento, deduplicação de trechos e eventual segmentação em unidades menores;
2. **Geração de *embeddings***: cada documento, ou cada segmento textual, é convertido em um vetor de características numéricas por meio de um modelo de linguagem pré-treinado;
3. **Construção do índice**: os vetores gerados são inseridos em uma estrutura de índice vetorial (por exemplo, FAISS), a qual é otimizada para operações de busca por vizinhos mais próximos em alta dimensão;
4. **Mapeamento entre vetores e metadados**: paralelamente, é mantido um arquivo ou estrutura de metadados que associa cada vetor (identificado por um *id* interno) ao documento original, preservando campos como título, conteúdo, fonte e outros atributos relevantes.

Na fase de **consulta**, o fluxo é análogo, mas orientado à recuperação:

1. o usuário ou sistema cliente envia uma *query* em texto para a API do DBVECTOR;
2. a *query* é pré-processada e convertida em um vetor de *embedding* pelo mesmo modelo utilizado na indexação;
3. o vetor da *query* é consultado no índice vetorial, que retorna os identificadores dos vetores mais próximos (maior similaridade);
4. os identificadores retornados são utilizados para recuperar os documentos e metadados correspondentes;
5. o conjunto de resultados é ordenado e, se necessário, submetido a um pós-processamento (por exemplo, reordenação ou filtragem) antes de ser devolvido ao cliente.

Esse fluxo garante consistência entre a forma como documentos e consultas são representados, o que é essencial para a qualidade do sistema de recuperação semântica.

4.4.4 Formato dos Dados e Esquema Documental

Os documentos tratados pelo DBVECTOR seguem, em geral, um esquema simples e flexível em formato JSON Lines. Cada linha representa um documento e contém campos típicos como:

- `id`: identificador único do documento;
- `title`: título ou descrição resumida;
- `content`: texto principal do documento (por exemplo, inteiro teor de uma decisão ou conteúdo de um manual);
- `source`: informação sobre a origem (tribunal, sistema, *cluster* temático);
- `metadata`: estrutura adicional com campos específicos (artigo do Código Penal, tipo de documento, data, entre outros).

Os *embeddings* são representados internamente como vetores de números em ponto flutuante (tipicamente `float32`) organizados em matrizes, nas quais cada linha corresponde a um documento ou segmento textual. Embora esses vetores sejam armazenados nos índices vetoriais, o mapeamento entre identificadores internos e documentos originais é mantido em estruturas paralelas, garantindo rastreabilidade e transparência.

4.4.5 Estratégias de Indexação e Armazenamento

O DBVECTOR suporta diferentes estratégias de indexação e armazenamento, com destaque para:

- **FAISS**: utilizado como backend principal para armazenamento local de índices vetoriais de alta performance. Essa solução é adequada para cenários em que o volume de dados é compatível com o ambiente local e se busca baixa latência em consultas;
- **OpenSearch**: empregado como opção alternativa para cenários em que se requerem capacidades adicionais, como busca híbrida (combinação de texto e vetores), replicação, distribuição e integração via APIs REST em ambientes mais próximos de produção.

Em ambos os casos, o sistema mantém a associação entre cada vetor armazenado e seus metadados. Isso permite que, após a obtenção de um conjunto de identificadores pelo índice vetorial, a camada de aplicação recupere o documento completo e informações complementares, que são posteriormente utilizadas pelo *pipeline* RAG para compor respostas fundamentadas.

Questões de compatibilidade de versões (especialmente no caso do FAISS) e de capacidade de armazenamento são consideradas na operação do DBVECTOR. Mudanças significativas de versão ou de configuração do backend podem exigir a reconstrução dos índices a partir dos arquivos de documentos originais.

4.4.6 API de Consulta e Integração com a Aplicação

Para integração com os demais componentes do sistema, em especial com a interface e com a camada de orquestração do RAG, o DBVECTOR disponibiliza uma API de consulta. Essa API, em linhas gerais, oferece:

- um ponto de entrada para recebimento de *queries* textuais;
- rotinas internas para geração de *embeddings* da consulta;
- chamadas ao backend vetorial selecionado (FAISS ou OpenSearch) para recuperação dos vetores mais similares;
- mapeamento dos resultados para documentos e metadados legíveis pela aplicação cliente;
- retorno estruturado, adequado ao consumo por modelos de linguagem ou por outras camadas de apresentação.

Dessa forma, o DBVECTOR atua como um serviço especializado de busca semântica, abstrato em relação ao backend de armazenamento, mas consistente em relação ao formato de entrada e saída, o que favorece a integração com a interface descrita anteriormente e com os módulos de coleta de dados.

4.4.7 Avaliação, Extensões e Considerações Finais

O módulo inclui ainda mecanismos voltados à avaliação da qualidade da recuperação, por meio de conjuntos de dados de teste e *scripts* que calculam métricas pertinentes (como *recall* em diferentes profundidades). Esses recursos permitem comparar configurações de índice, modelos de *embeddings* e estratégias de pós-processamento, contribuindo para a escolha das melhores combinações para o domínio jurídico considerado.

Em termos de extensibilidade, a arquitetura do DBVECTOR permite: (i) a adição de novos backends de armazenamento vetorial, por meio de implementações específicas que seguem um contrato comum de armazenamento e busca; (ii) a substituição ou atualização do modelo de *embeddings*, desde que mantida a interface de geração de vetores; e (iii) a criação de novos *pipelines* de ingestão de dados, capazes de produzir documentos no formato JSONL esperado pelo sistema.

Por fim, cabe destacar que a operação do DBVECTOR está sujeita a limitações de ambiente (por exemplo, disponibilidade de GPU ou CPU, restrições de memória e armazenamento) e a cuidados relacionados à privacidade dos dados indexados. Documentos que contenham informações sensíveis devem ser previamente anonimizados ou tratados conforme as políticas de segurança adotadas. Ainda assim, o módulo fornece uma base robusta e flexível para indexação semântica, desempenhando papel central na qualidade das respostas geradas pelo *pipeline* de Recuperação Aumentada por Geração proposto neste trabalho.

4.5 COLETA DE DADOS – EXTRATOR DO STF

4.5.1 Objetivo e Escopo

A construção de sistemas de recuperação de informação jurídica depende fundamentalmente da disponibilidade de acervos documentais atualizados, estruturados e representativos do domínio de conhecimento. Nesse contexto, o módulo de extração automatizada desenvolvido para o portal de jurisprudência do Supremo Tribunal Federal (STF) foi concebido com o propósito de extrair, organizar e padronizar decisões públicas incluindo acórdãos, despachos e demais documentos transformando-as em registros estruturados adequados para armazenamento e indexação vetorial.

A estratégia de coleta baseia-se em consultas pré-definidas (*queries*) agrupadas por artigo do Código Penal, permitindo a criação de um corpus temático de jurisprudência alinhado aos requisitos da *pipeline* de Recuperação Aumentada por Geração (RAG). Dessa forma, a automação persegue quatro objetivos principais: (i) garantir atualização contínua da base documental, acompanhando a publicação de novas decisões; (ii) padronizar dados heterogêneos oriundos da interface web, reduzindo variabilidade e facilitando processamento posterior; (iii) reduzir trabalho manual e mitigar suscetibilidade a erros humanos; e (iv) alimentar o *pipeline* RAG com dados em formato uniforme e processável, assegurando qualidade e rastreabilidade das informações.

4.5.2 Implementação e Arquitetura

O coletor foi implementado em *Python*, utilizando o framework Scrapy e o Playwright para automação de navegador, dada a dependência do portal em JavaScript para renderização dinâmica de conteúdo. A arquitetura adota organização modular, segregando responsabilidades em componentes especializados: módulo de orquestração de consultas, módulo de extração e navegação, *pipelines* de processamento e validação, e camada de persistência em arquivos JSON Lines.

Para otimizar o aproveitamento de recursos computacionais, a solução emprega processamento paralelo coordenado por mecanismo de fila compartilhada, permitindo execução concorrente de múltiplas consultas. Ademais, o controle de estado persistido

facilita a retomada da execução em caso de interrupções, garantindo resiliência e eficiência operacional.

4.5.3 Fluxo de Extração (Busca, Listagem e Paginação)

A execução inicia a partir de arquivo de configuração contendo consultas pré-definidas, em que cada *query* está associada a um artigo do Código Penal. O gerenciador de fila mantém o controle das consultas processadas, pendentes e em execução, permitindo retomada controlada e evitando processamento duplicado mediante mecanismos de sincronização.

Para cada consulta, o módulo de navegação estabelece sessão de navegador em modo *headless* e acessa a página de pesquisa correspondente. Considerando que os resultados são apresentados de forma paginada tipicamente até 250 decisões por página o coletor aguarda o carregamento completo dos elementos, identifica o total de itens e o número de páginas, e percorre as listagens extraindo metadados básicos por item, tais como título, link, número do caso e artigo associado. Cada item identificado gera requisição subsequente à respectiva página de detalhes para extração do conteúdo integral.

4.5.4 Extração do Inteiro Teor e Campos Estruturados

Após localizar os links das decisões, o scraper acessa a página de detalhes utilizando o mesmo contexto de navegador do *worker*. Como o texto é renderizado dinamicamente em JavaScript, a extração do inteiro teor é, preferencialmente, realizada por meio da API de área de transferência (*clipboard*) exposta pela página: aguarda-se o carregamento do elemento de texto, aciona-se, via JavaScript, o botão de "copiar" e lê-se o conteúdo da área de transferência como texto principal da decisão.

Quando a estratégia de *clipboard* não está disponível ou falha, aplica-se um mecanismo de *fallback* que extrai o texto diretamente do Document Object Model (DOM), por meio de seletores CSS ou XPath. Além do texto integral, são extraídos campos estruturados, como relator, partes, legislação citada e demais metadados relevantes.

4.5.5 Validação, Organização por Artigo e Persistência

Os dados extraídos são encapsulados em objetos estruturados e encaminhados a uma cadeia de *pipelines* de processamento. Essa cadeia compreende três etapas principais:

- **Validação**: verifica a presença de campos obrigatórios (título, URL, conteúdo textual), calcula escore de qualidade baseado em critérios como tamanho do texto, presença de relator e completude estrutural, e descarta registros abaixo de limiar mínimo estabelecido;

- **Organização temática**: identifica o artigo do Código Penal associado a cada decisão por meio de metadados estruturados e direciona o registro para arquivo JSON Lines específico do respectivo artigo, em modo *append*, preservando execuções anteriores e facilitando atualização incremental;
- **Registro de métricas**: coleta estatísticas de execução, incluindo número de itens processados por artigo, distribuição de qualidade e tempo de processamento, permitindo monitoramento e diagnóstico do sistema.

Como resultado, o coletor produz, para cada artigo do Código Penal, um arquivo JSONL contendo uma linha por decisão, com metadados enriquecidos (título, URL de origem, artigo associado, texto completo, relator, legislação citada e escore de qualidade). Esses arquivos constituem a base primária de jurisprudência utilizada no projeto.

4.5.6 Papel no Pipeline RAG (Análise e Impacto)

O scraper garante que o modelo de linguagem disponha de um corpus abrangente e atualizado de decisões organizadas por artigo do Código Penal. Ao agrupar decisões por artigo e prover metadados estruturados (título, relator, URL de origem e texto integral), o coletor:

- melhora a precisão do RAG ao permitir recuperação de trechos altamente relevantes;
- viabiliza consultas contextualizadas por tema ou dispositivo legal;
- aumenta a rastreabilidade das respostas, ao vincular cada saída a documentos oficiais;
- sustenta a atualização contínua do conhecimento jurídico ao incorporar novas decisões aos arquivos JSONL por artigo.

4.5.7 Limitações e Considerações Finais

O coletor depende de interfaces web dinâmicas e, portanto, pode necessitar de ajustes quando o portal do STF alterar significativamente sua estrutura ou APIs. A estratégia de extração por *clipboard*, embora eficiente, requer mecanismos de *fallback* para garantir extração quando essa API não estiver disponível. Além disso, condições de rede e limites impostos pelo servidor podem afetar a completude da coleta, sendo mitigadas por políticas de *retry* e controle de concorrência descritos anteriormente.

Em síntese, o scraper constitui componente central para a integridade e a qualidade dos dados que alimentam o *pipeline* RAG, contribuindo diretamente para a confiabilidade dos resultados do sistema.

4.6 COLETA DE DADOS – SCRAPER DO TRF4

4.6.1 Objetivo e Escopo

O módulo de extração desenvolvido para o Tribunal Regional Federal da 4ª Região (TRF4) complementa o *pipeline* de obtenção e padronização de dados jurídicos, ampliando a cobertura documental com decisões de âmbito regional. Seu propósito é automatizar a busca e a coleta de decisões judiciais publicadas no portal do Tribunal, garantindo organização, consistência e padronização dos registros para posterior indexação vetorial e uso no *pipeline* de Recuperação Aumentada por Geração (RAG).

A estratégia de coleta persegue cinco objetivos específicos: (i) capturar decisões e documentos relevantes de forma contínua e estruturada; (ii) extrair metadados essenciais, incluindo número do processo, tipo de documento, datas de julgamento e publicação, e identificação do relator; (iii) obter o texto integral das decisões, quando disponível na interface pública; (iv) padronizar os registros em formato compatível com a base vetorial; e (v) minimizar inconsistências e duplicidades na base final por meio de mecanismos de validação e deduplicação.

4.6.2 Implementação, Arquitetura e Fluxo de Coleta

A arquitetura do coletor do TRF4 adota organização modular compatível com os demais extratores do projeto, permitindo reúso de componentes e manutenção consistente. A implementação foi realizada em *Python*, utilizando o framework Scrapy e, quando necessário, automação de navegador por meio do Playwright, caso a interface do portal exija execução de JavaScript para renderização de conteúdo. O fluxo geral de coleta compreende as etapas descritas a seguir.

4.6.2.1 Abertura do portal e configuração da busca

O coletor inicia estabelecendo conexão com a página de busca do TRF4 e preparando o ambiente de interação: inicializa o contexto de execução (sessão HTTP ou navegador *headless*), localiza os elementos de interface necessários e carrega o termo de pesquisa configurado.

4.6.2.2 Aplicação de filtros e submissão da consulta

Em seguida, o scraper ajusta filtros pertinentes (como pesquisa avançada, tipo de decisão e intervalos de datas), preenche os campos de busca e submete o formulário para iniciar a consulta, respeitando a lógica de navegação do portal.

4.6.2.3 Extração estruturada dos resultados

Após o retorno da listagem de resultados, cada item é identificado por meio de seletores CSS ou XPath, e são extraídos os campos principais: número do processo, tipo de documento, datas (de julgamento e de publicação), relator e, quando acessível, o texto integral da decisão. Os metadados e o conteúdo textual são encapsulados em um objeto de coleta para posterior padronização.

4.6.2.4 Navegação automática entre páginas

O scraper verifica a existência de paginação e, se presente, avança automaticamente para as páginas subsequentes: aciona o controle de página (botão ou link), aguarda o carregamento completo e retoma a extração até esgotar as páginas de resultados. Esse procedimento garante cobertura integral do conjunto de decisões retornadas pelos filtros utilizados.

4.6.2.5 Padronização e persistência

Cada registro extraído passa por um *pipeline* de normalização, que realiza limpeza textual, normalização de chaves e enriquecimento de metadados. Os registros são gravados em formato compatível com o índice vetorial (por exemplo, JSON Lines), preservando referência à página de origem, ao índice na página e ao termo de busca empregado.

4.6.3 Estratégias de robustez, logs e resiliência

Para garantir execução estável e auditável, o coletor incorpora mecanismos de confiabilidade, entre os quais se destacam:

- geração de logs detalhados, com identificadores de execução e métricas de extração;
- persistência imediata (*write-through*) de cada resultado extraído, minimizando perda de dados em caso de falha;
- mecanismos de *fallback* quando seletores esperados não forem encontrados (uso de seletores alternativos ou marcação do item para revisão manual);
- controle de concorrência e coordenação entre múltiplos *workers*, evitando processamento duplicado de páginas;
- captura de *snapshots* (HTML ou PDF) das páginas consultadas para fins de diagnóstico; e
- tratamento estruturado de exceções, com políticas de *retry* e *backoff*.

4.6.4 Papel do Scraper do TRF4 no Pipeline RAG

O coletor do TRF4 amplia e aprimora a cobertura documental do projeto, contribuindo para a qualidade do *pipeline* RAG ao:

- aumentar o conjunto de decisões disponíveis para vetorização;
- viabilizar atualização contínua da base jurídica sem intervenção manual constante;
- melhorar a precisão das respostas do modelo, ao fornecer documentos regionais e federais complementares;
- assegurar uniformidade no tratamento dos dados coletados, facilitando a indexação e a recuperação conjunta com fontes como STF e SEEU.

De modo geral, o scraper do TRF4 segue os mesmos princípios de organização, padronização e rastreabilidade adotados nos demais coletores, garantindo coerência metodológica no conjunto da solução.

4.7 INTERFACE DE USUÁRIO DA APLICAÇÃO

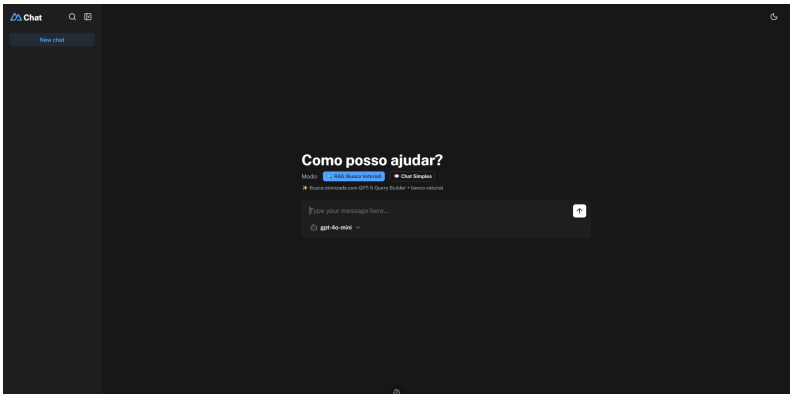

Figura 4.4 – Tela inicial da interface (exemplo). Fonte: elaborado pelos autores.

4.7.1 Objetivo

Esta seção descreve a interface de usuário (IU) da aplicação de Recuperação Aumentada por Geração (RAG). A IU tem por finalidade proporcionar um ambiente unificado para: (i) submissão de consultas em linguagem natural; (ii) apresentação das respostas geradas pelos modelos de linguagem; e (iii) exposição das fontes recuperadas e metadados associados, assegurando rastreabilidade e auditoria dos resultados.

4.7.2 Escopo e requisitos principais

O projeto da IU contempla os seguintes requisitos funcionais e não funcionais, compatíveis com o escopo do trabalho:

- capacidade de receber consultas por meio de um campo de entrada textual;
- seleção de modo de operação (por exemplo, RAG ou *chat* puro) e de modelo;
- exibição ordenada da resposta gerada, com indicação explícita das fontes consultadas e dos trechos citados;
- indicadores de confiança/qualidade da resposta e metadados de cobertura;
- mecanismos de inspeção e diagnóstico (*logs*) disponíveis em ambiente de desenvolvimento;
- desempenho interativo compatível com aplicação de demonstração acadêmica (latências reduzidas e feedback de carregamento).

4.7.3 Arquitetura e implementação

Por coerência com o restante do projeto, a implementação adotou uma arquitetura de front-end baseada em componentes, separando apresentação, orquestração e serviços. A camada de apresentação foi desenvolvida a partir de um template compatível com Nuxt/Vue (adaptado ao propósito deste trabalho) e implementada em TypeScript para prover contratos de dados mais explícitos.

Os principais elementos arquiteturais são:

- componente de página principal, responsável pela orquestração do fluxo de interação;
- componentes de apresentação (*prompt*, painel de resultados, lista de fontes e elementos de utilidade);
- unidades de lógica reutilizáveis (*composables* ou *hooks*) que encapsulam chamadas a serviços, gerenciamento de estado e tratamento de erros;
- adaptadores para comunicação com a camada de serviços (API de backend) que fornecem respostas geradas, documentos recuperados e metadados.

4.7.4 Fluxos de interação

O fluxo básico de interação é composto pelas etapas a seguir:

1. o usuário insere a consulta textual e seleciona, opcionalmente, parâmetros de execução;
2. a IU valida a entrada mínima (ex.: não vazia, tamanho máximo) e transita para estado de carregamento;

3. a chamada é enviada à camada de serviços, que executa o pipeline (recuperação, rankeamento e geração);
4. a IU recebe a resposta e os metadados, renderizando o texto gerado e a listagem de fontes com trechos citados e links de origem;
5. o usuário pode inspecionar documentos, copiar trechos, ou formular consultas de seguimento (*follow-up*).

Para preservar a usabilidade, a interface expõe feedbacks visuais de carregamento e tratamento de erros, bem como controles para cancelamento de requisições quando suportado pelo backend.

4.7.5 Estados, comunicação e contratos de dados

A IU emprega estados reativos (por exemplo, `input`, `loading`, `response`, `sources`) para orientar a renderização. A comunicação com a camada de serviços é assíncrona e baseada em contratos JSON tipados, contendo, no mínimo, os campos: texto da resposta, array de fontes (cada qual com título, URL, trecho e pontuação de relevância) e indicadores de qualidade.

Os contratos adotados asseguram interoperabilidade entre a IU e os pipelines de extração (por exemplo, os arquivos JSON Lines gerados pelos scrapers descritos neste capítulo), permitindo mapear referências e evidências a partir dos metadados armazenados.

4.7.6 Componentes principais

Os componentes de IU priorizam clareza informacional e rastreabilidade; destacam-se:

- **Componente de entrada (Prompt)**: captura a consulta e parâmetros, com validação e sugestões rápidas;
- **Painel de resposta**: apresenta o texto gerado e permite a expansão de citações e visualização das fontes;
- **Lista de fontes**: exibe documentos recuperados, com metadados (cluster, título, URL, trecho) e botão para acessar a origem;
- **Controles auxiliares**: copiar resposta, iniciar *follow-up*, alternar modo (RAG/*chat*) e abrir painéis de diagnóstico (quando aplicável).

4.7.7 Decisões de projeto e boas práticas

Foram adotadas decisões técnicas e de usabilidade alinhadas às exigências do trabalho acadêmico e às boas práticas de engenharia de software:

- separação clara entre apresentação e orquestração de fluxos, favorecendo testabilidade e manutenção;
- contratos tipados entre front-end e serviços para reduzir ambiguidades e facilitar validação;
- habilitação de modos restritos de diagnóstico apenas em ambiente de desenvolvimento, preservando privacidade e segurança em produção;
- registro de eventos e métricas de interação para fins de avaliação de usabilidade e de desempenho.

4.7.8 Limitações e considerações finais

A interface constitui uma camada de apresentação dependente da disponibilidade e do comportamento da camada de serviços (pipelines de busca e modelos de linguagem). Alterações nos contratos, na latência ou na disponibilidade dos serviços podem exigir ajustes na IU. Ademais, a adoção de um template genérico impôs adaptações para o contexto jurídico que podem ser refinadas em trabalhos posteriores.

Em conformidade com a transparência requerida pelo projeto, a IU enfatiza a rastreabilidade das respostas, vinculando cada trecho citado às fontes originais geradas pelos scrapers descritos neste capítulo.

4.8 COLETA DE DADOS – SCRAPER SEEU (SISTEMA ELETRÔNICO DE EXECUÇÃO UNIFICADO)

4.8.1 Objetivo e Escopo

O módulo de extração desenvolvido para coleta de documentos públicos relacionados ao Sistema Eletrônico de Execução Unificado (SEEU) tem por objetivo automatizar a extração, organização e padronização de materiais de suporte à execução penal, tais como orientações técnicas, manuais operacionais, portarias e instruções normativas. Esses materiais são disponibilizados em portais oficiais do CNJ e constituem insumos textuais complementares à jurisprudência dos tribunais, compondo o corpus utilizado no *pipeline* de Recuperação Aumentada por Geração (RAG).

Dessa forma, o coletor persegue quatro objetivos principais: (i) garantir atualização contínua da base documental, acompanhando a publicação de novos materiais normativos; (ii) padronizar formatos heterogêneos de publicação, reduzindo variabilidade estrutural; (iii) reduzir trabalho manual de coleta e mitigar suscetibilidade a erros; e (iv) disponibilizar documentos em formato estruturado, adequado à indexação vetorial e à consulta semântica.

4.8.2 Implementação e Arquitetura

O coletor foi implementado em *Python*, utilizando o framework Scrapy e adotando arquitetura modular alinhada aos demais extratores do projeto. Diferentemente dos coletores de jurisprudência, que lidam com conteúdo dinâmico renderizado por JavaScript, este módulo opera sobre páginas essencialmente estáticas, dispensando automação de navegador e simplificando o processo de extração.

A arquitetura compreende três componentes principais:

- **Módulo de orquestração**: responsável por coordenar as requisições à página índice dos materiais e percorrer as entradas disponíveis sistematicamente;
- ***Pipelines* de processamento**: encarregados de limpar, normalizar e serializar os dados coletados para o formato padronizado adotado pelo banco vetorial;
- **Camada de persistência**: baseada em arquivos JSON Lines, armazenados em diretório específico do projeto, garantindo compatibilidade com os demais módulos.

Essa organização mantém consistência com o padrão de saída dos demais coletores, favorecendo o reúso de componentes e simplificando a etapa de vetorização.

4.8.3 Fluxo de Coleta (Página Índice e Documentos)

O fluxo de coleta do scraper SEEU inicia-se com o envio de uma requisição HTTP à página índice que centraliza os materiais de suporte oficiais. Após o carregamento dessa página, a *spider* identifica cada item disponível na listagem, normalmente composto por título, descrição sucinta e link para o documento correspondente (em HTML, PDF ou outros formatos).

A partir dessa listagem, o fluxo pode ser resumido nas etapas a seguir:

1. envio de requisição à página índice dos materiais de suporte;
2. identificação, por meio de seletores CSS ou XPath, dos elementos que representam cada documento;
3. extração de metadados básicos, tais como título, URL e, quando disponível, breve descrição do conteúdo;
4. construção de um objeto de coleta contendo esses metadados;
5. encaminhamento do objeto ao *pipeline* de processamento para tratamento e persistência.

Quando o documento está disponível em formato HTML acessível, a *spider* tenta realizar a extração imediata do conteúdo textual. Quando se trata de arquivo binário (por

exemplo, PDF), registra-se a URL para que a conversão em texto seja realizada em etapa posterior, com ferramentas específicas de extração textual.

4.8.4 Processamento, Padronização e Persistência

Considerando a diversidade de formatos e estruturas presentes nos materiais públicos relacionados ao SEEU, o *pipeline* de processamento atua em duas frentes principais:

- **Documentos HTML**: o conteúdo é extraído diretamente pelo módulo de *parsing*, com identificação da área central do texto por meio de seletores especializados e remoção de elementos não informativos (menus, rodapés, *scripts* e componentes de navegação);
- **Documentos binários**: os metadados (título, URL e descrição, quando disponível) são registrados, ficando a extração textual completa a cargo de etapa posterior de pré-processamento, que pode envolver bibliotecas especializadas para leitura de PDF ou reconhecimento óptico de caracteres.

Independentemente do formato original, os registros finais são normalizados para estrutura canônica utilizada pelo banco vetorial, compatibilizando-os com os demais coletores do projeto. No *pipeline* de normalização, são aplicadas as seguintes transformações:

- categorização temática automatizada, identificando os documentos como pertencentes ao conjunto de materiais de suporte relacionados ao SEEU;
- normalização de chaves e ordenação dos campos de acordo com o padrão estabelecido no projeto;
- limpeza textual, com remoção de quebras de linha redundantes, espaços em excesso e caracteres indesejados;
- gravação dos registros em formato JSON Lines, em modo *append*, permitindo inserção incremental e preservação de execuções anteriores.

Esse procedimento assegura que todos os documentos públicos relacionados ao SEEU sejam disponibilizados em formato uniforme, compatível com a etapa de vetorização e com as rotinas de auditoria manual.

4.8.5 Integração com o Pipeline RAG

A base documental construída pelo scraper SEEU integra-se ao *pipeline* de Recuperação Aumentada por Geração como fonte complementar à jurisprudência extraída dos tribunais. Enquanto o scraper do STF fornece decisões judiciais organizadas por artigo do

Código Penal, o coletor SEEU disponibiliza documentos normativos e orientativos que contextualizam a execução penal.

Na etapa de indexação vetorial, os registros oriundos do SEEU são processados da mesma forma que os demais documentos: o campo `content` é convertido em vetores de características (*embeddings*), armazenados no índice vetorial juntamente com seus metadados (`cluster_name`, título e URL). Durante as consultas, o modelo RAG pode recuperar, para uma mesma pergunta, tanto decisões judiciais quanto documentos de suporte, permitindo respostas mais completas, contextualizadas e rastreáveis.

4.8.6 Limitações e Considerações Finais

Assim como outros coletores baseados em *web scraping*, o scraper SEEU é sensível a alterações na estrutura das páginas consultadas. Modificações significativas no layout ou na organização dos materiais podem exigir ajustes nos seletores utilizados e novos testes de validação. Além disso, a presença de documentos em formato binário implica dependência de uma etapa adicional de extração textual para torná-los plenamente pesquisáveis.

Recomenda-se, portanto, a adoção de políticas de *retry*, controle de concorrência e registro detalhado de logs, de forma a reduzir o impacto de falhas temporárias de rede ou indisponibilidades momentâneas do servidor. Em conjunto, esses cuidados tornam o coletor SEEU apto a fornecer, de maneira contínua e confiável, uma base de documentos de suporte organizada e padronizada, preparada para alimentar o processo de indexação vetorial do projeto e contribuir para a qualidade das respostas geradas pelo *pipeline* RAG.

4.9 COLETA DE DADOS – EXTRATOR DO STJ

4.9.1 Objetivo e Escopo

O módulo de extração automatizada desenvolvido para o Superior Tribunal de Justiça (STJ) tem por objetivo coletar, processar e padronizar decisões monocráticas e acórdãos publicados no Diário da Justiça e disponibilizados no portal de dados abertos do Tribunal. Diferentemente dos coletores baseados em consultas diretas ao portal de jurisprudência, este módulo opera sobre conjuntos de dados estruturados distribuídos em arquivos compactados, contendo metadados em formato JSON e textos integrais em arquivos de texto puro.

A estratégia de coleta persegue cinco objetivos específicos: (i) automatizar o descobrimento e o download de recursos de dados abertos; (ii) extrair e associar decisões judiciais aos seus respectivos textos integrais por meio de identificadores únicos; (iii) filtrar e selecionar decisões monocráticas relevantes ao escopo do projeto; (iv) normalizar metadados e conteúdo textual para formato uniforme, compatível com os demais coletores; e (v) produzir saída estruturada em JSON Lines, adequada à indexação vetorial e ao uso no *pipeline* RAG.

4.9.2 Implementação e Arquitetura

O coletor foi implementado em *Python*, utilizando o framework Scrapy e bibliotecas especializadas para manipulação de arquivos compactados e interações com APIs de portais de dados abertos. A arquitetura modular é compatível com os demais coletores do projeto, sendo composta pelos seguintes componentes principais:

- **Módulo de descoberta**: responsável por interagir com a API do portal de dados abertos do STJ, listar conjuntos de dados disponíveis e identificar os recursos associados a cada conjunto;
- **Gerenciador de fila**: encarregado de coordenar a fila de recursos a serem processados, manter o estado de execução persistido e permitir retomada de processamento em caso de interrupções;
- **Módulo de download**: encapsula a lógica de requisições HTTP, incluindo tratamento de erros, políticas de *retry* com *backoff* exponencial e armazenamento local dos arquivos em diretório temporário;
- **Módulo de extração e *parsing***: realiza a abertura dos arquivos compactados, identificação dos arquivos JSON de metadados e dos arquivos de texto correspondentes, e o mapeamento entre decisões e seus textos integrais por meio de identificadores únicos;
- **Módulo de normalização**: transforma os metadados brutos em estrutura padronizada, realiza limpeza textual, extração de informações estruturadas (relator, partes, legislação citada) e calcula indicador de qualidade de conteúdo;
- ***Pipelines* de processamento**: aplicam validações, filtram decisões por critérios definidos (tipo de decisão, esfera jurisdicional), eliminam duplicatas e gravam os registros no formato JSON Lines.

Essa organização mantém coerência arquitetural com os demais coletores do projeto, favorecendo o reúso de componentes e simplificando a manutenção da base de código.

4.9.3 Fluxo de Coleta (Descoberta, Download e Extração)

A execução do coletor do STJ compreende cinco etapas principais, descritas a seguir.

4.9.3.1 Descoberta de recursos

O módulo de descoberta consulta a API do portal de dados abertos do STJ e obtém a listagem de conjuntos de dados disponíveis. Para cada conjunto, identifica os

recursos associados tipicamente, arquivos compactados contendo lotes de decisões e extrai os metadados relevantes, tais como identificador do recurso, URL de download, data de publicação e descrição.

4.9.3.2 Enfileiramento e controle de estado

Cada recurso descoberto é adicionado à fila de processamento gerenciada pelo módulo de controle, que mantém registro dos recursos já processados, pendentes e em execução. O estado da fila é persistido em arquivo estruturado, permitindo retomada do processamento em caso de interrupções e evitando reprocessamento duplicado de recursos.

4.9.3.3 Download e armazenamento temporário

Múltiplos processos coordenados pelo gerenciador de fila realizam o download dos arquivos compactados de forma paralela. Cada arquivo é armazenado temporariamente em diretório específico, possibilitando reprocessamento posterior sem necessidade de novo download e reduzindo a carga sobre os servidores do STJ.

4.9.3.4 Extração e mapeamento de decisões

Após o download, o módulo de extração descompacta o arquivo e identifica os componentes contidos: tipicamente, um ou mais arquivos JSON com metadados das decisões e uma coleção de arquivos de texto nomeados de acordo com identificadores únicos. O módulo realiza o mapeamento entre cada decisão e o arquivo de texto correspondente, tolerando variações de nomenclatura e aplicando heurísticas de *fallback* em caso de inconsistências.

4.9.3.5 Filtragem e seleção

Cada decisão extraída é submetida a critérios de filtragem para identificar aquelas relevantes ao escopo do projeto. São considerados, entre outros, os seguintes critérios:

- tipo de documento: decisões monocráticas;
- esfera jurisdicional: compatível com o âmbito definido no projeto;
- presença de texto integral: decisões sem texto associado são marcadas para auditoria posterior, mas não descartadas imediatamente.

4.9.4 Normalização, Rastreabilidade e Persistência

Após a extração e a filtragem, os registros passam pelo módulo de normalização, que realiza quatro transformações principais:

- **Padronização de campos**: conversão de datas para formato ISO (AAAA-MM-DD), extração e estruturação de informações sobre relator, identificação de partes processuais e legislação citada;
- **Limpeza textual**: remoção de caracteres de controle, normalização de espaços em branco, correção de quebras de linha e eliminação de artefatos de extração;
- **Cálculo de qualidade**: atribuição de escore de qualidade de conteúdo baseado em critérios como tamanho do texto, presença de metadados essenciais e completude da estrutura documental;
- **Geração de metadados de rastreabilidade**: inclusão de metadados de proveniência contendo informações sobre origem do arquivo, identificador do recurso, URL de download, caminho interno no arquivo compactado e *timestamp* de processamento.

Esses metadados de rastreabilidade são essenciais para garantir auditabilidade e reprodutibilidade, permitindo que cada decisão seja rastreada até sua origem no portal de dados abertos do STJ.

Os registros normalizados são encapsulados em objetos padronizados e encaminhados ao *pipeline* de persistência, que realiza três etapas principais:

- validação de campos obrigatórios (identificador único, título, conteúdo textual);
- eliminação de duplicatas com base em identificadores únicos de documento e recurso;
- gravação em arquivo JSON Lines único, com uma linha por decisão, em modo *append*.

Opcionalmente, o coletor pode preservar os arquivos de texto extraídos em diretório separado para inspeção manual ou reprocessamento posterior.

4.9.5 Papel no Pipeline RAG

O scraper do STJ amplia significativamente a cobertura documental do projeto, fornecendo decisões monocráticas de um dos principais tribunais superiores do país. Ao integrar esses dados ao *pipeline* RAG, o sistema passa a dispor de:

- maior diversidade de fontes jurídicas, contemplando decisões do STJ além daquelas do STF e de tribunais regionais;
- decisões organizadas e rastreáveis até a origem no portal de dados abertos, aumentando a confiabilidade e a auditabilidade das respostas geradas;
- metadados estruturados e normalizados, facilitando consultas temáticas, filtragem por relator, legislação citada ou período de publicação;

- conteúdo textual limpo e pronto para vetorização, otimizando a qualidade dos *embeddings* e a precisão da recuperação semântica.

A integração do coletor do STJ reforça, portanto, a capacidade do *pipeline* RAG de fornecer respostas contextualizadas, embasadas em um corpus abrangente e diversificado de decisões judiciais.

4.9.6 Limitações e Considerações Finais

O coletor do STJ, assim como os demais scrapers baseados em dados abertos, é sensível a alterações na estrutura dos arquivos ZIP, nos esquemas de metadados JSON e na nomenclatura dos arquivos de texto. Inconsistências na associação entre `seqDocumento` e arquivos `.txt` podem ocorrer, sendo mitigadas por heurísticas de *fallback* e registro detalhado de casos não resolvidos no bloco de rastreabilidade.

A dependência de APIs externas (CKAN) implica necessidade de monitoramento contínuo e adaptação a eventuais mudanças nos endpoints, parâmetros de consulta ou estrutura de resposta. Além disso, condições de rede e políticas de acesso podem afetar a taxa de sucesso dos downloads, sendo recomendada a adoção de mecanismos robustos de *retry*, controle de concorrência e registro de logs detalhados.

Em síntese, o extrator do STJ constitui componente fundamental para a abrangência e a qualidade da base documental que alimenta o *pipeline* RAG, contribuindo diretamente para a confiabilidade, a rastreabilidade e a completude das respostas fornecidas pelo sistema.

4.10 INTEGRAÇÃO DA INTERFACE COM A *PIPELINE* RAG

Esta seção descreve como a interface de usuário, a API REST, os módulos de indexação vetorial (DBVECTOR) e os modelos de linguagem se integram para formar um sistema coeso e funcional. A integração garante que as consultas dos operadores do Direito sejam processadas de forma eficiente, com recuperação semântica precisa e geração de respostas fundamentadas em fontes verificáveis.

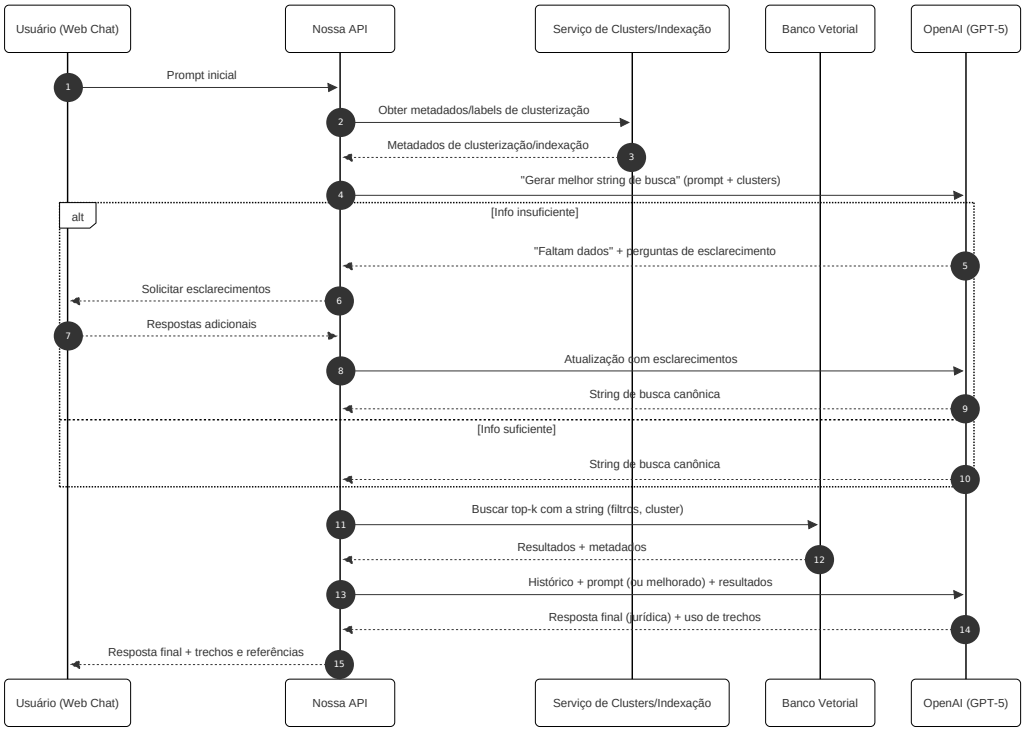

Figura 4.5 – Fluxo de integração entre interface, API, serviços de indexação, banco vetorial e modelo de linguagem. Fonte: elaborado pelos autores.

A Figura 4.5 apresenta o fluxo de integração entre o usuário da interface *web*, a API da aplicação, o serviço de indexação vetorial, o banco de dados vetorial e o modelo de linguagem responsável pela geração das respostas em linguagem natural. Nesse diagrama evidenciam-se as trocas de mensagens desde o envio do *prompt* inicial até a devolução da resposta final, acompanhada dos trechos e referências utilizados, incluindo o tratamento de situações em que o sistema identifica falta de informações e solicita esclarecimentos adicionais ao usuário.

Do ponto de vista arquitetural, essa integração complementa a visão geral da *pipeline* RAG apresentada anteriormente (Figura 4.1), na qual os módulos de coleta, pré-processamento, engenharia de *embeddings*, indexação e orquestração foram descritos de forma isolada. A interface de usuário e a API atuam como porta de entrada para o operador judiciário, encapsulando a complexidade da busca vetorial e da geração de respostas, e expondo a *pipeline* de forma unificada e orientada a casos de uso.

4.10.1 Fluxo de Execução

O fluxo inicia-se quando o operador judiciário insere uma pergunta na interface *web*. A aplicação *frontend* normaliza o texto, organiza o histórico da conversa e envia o *prompt* inicial para a API RAG, que desempenha o papel de orquestrador da requisição. A partir desse *prompt*, a API consulta o serviço de *clusters* e indexação, responsável por disponibilizar metadados de clusterização e rótulos associados aos índices vetoriais

previamente construídos a partir de documentos do SEEU e de fontes normativas correlatas. Esses metadados retornam à API e são combinados ao *prompt* original para formar o contexto que será submetido ao modelo de linguagem na etapa de refinamento da consulta.

Em seguida, a API solicita ao modelo de linguagem a geração de uma string de busca otimizada a partir do prompt do usuário e dos metadados de clusterização. Caso o modelo detecte informações insuficientes para formular uma consulta precisa, ele devolve uma mensagem indicando a falta de dados, acompanhada de perguntas de esclarecimento. A API repassa essas perguntas à interface, que as exibe ao operador; as respostas fornecidas pelo usuário retornam à API, são incorporadas ao contexto e um novo pedido de geração é feito, até que se obtenha uma string de busca canônica adequada para consulta no índice vetorial.

Uma vez definida a *string* de busca canônica, a API a envia ao serviço de *clusters* e indexação, que executa a busca vetorial no banco de *embeddings*, aplicando filtros e parâmetros de similaridade como *top-k* e *thresholds* configurados. O serviço retorna o conjunto de documentos mais relevantes com metadados (identificadores, títulos, trechos e origem). Esses resultados são consolidados em um objeto de resposta intermediária, que alimenta a etapa final de geração.

Na composição da resposta, a API envia ao modelo de linguagem o histórico da conversa, o prompt refinado e os trechos recuperados. O modelo gera uma resposta em linguagem natural ancorada nas fontes, retornando tanto o texto quanto a lista de documentos utilizados com seus trechos e referências. A interface apresenta esse pacote ao usuário como mensagem principal seguida de citações clicáveis, preservando o vínculo com os documentos do SEEU e demais bases normativas.

4.10.2 Contrato de Dados e Telemetria

Do ponto de vista de contrato de dados, a integração é realizada por chamadas HTTP ao *endpoint* da API RAG, que recebe um *payload* contendo o *prompt*, o histórico da sessão, filtros e parâmetros de recuperação, e responde com um objeto estruturado contendo: o texto gerado, pontuação de confiança, metadados das fontes e eventuais sinais de alerta (por exemplo, baixa cobertura documental). A interface interpreta esse objeto, atualiza estados internos (`loading`, `error`, `results`) e renderiza tanto o texto do modelo quanto os metadados das evidências.

Além do fluxo principal, a integração contempla telemetria e *logs* que reportam latência, falhas de comunicação e cobertura documental. A interface pode exibir mensagens ao usuário sobre a necessidade de refinamento da consulta e, em ambiente de desenvolvimento, painéis auxiliares com informações de diagnóstico sobre a recuperação vetorial e o comportamento da *pipeline*.

4.10.3 Proposta de Integração com o SEEU

A arquitetura proposta não contempla integração direta com os sistemas internos do SEEU, mas sim uma abordagem baseada em dados públicos relacionados ao sistema. Os módulos de extração automatizada (*scrapers*) e processos de coleta descritos anteriormente geram arquivos JSON Lines com documentos públicos — tais como manuais, orientações técnicas e normativos — disponibilizados em portais oficiais do CNJ relacionados ao SEEU. Esses documentos são pré-processados, segmentados e transformados em *embeddings*, que ficam armazenados no banco vetorial (DBVECTOR) e referenciados pelos serviços de indexação consultados pela API.

Dessa forma, cada resposta produzida pela *pipeline* mantém um elo rastreável com os documentos públicos relacionados ao SEEU e com a base normativa que os sustenta, reforçando transparência e auditabilidade. Em futuras implementações, essa arquitetura poderia evoluir para integração direta com APIs oficiais do SEEU, caso disponibilizadas, permitindo acesso a dados processuais específicos respeitando-se os requisitos de segurança e privacidade estabelecidos pelo CNJ.

4.10.4 Considerações Finais

Ao concentrar a complexidade técnica em um único ponto de entrada (API RAG) e manter contratos claros entre frontend, serviços de indexação e modelos de linguagem, a solução melhora a manutenibilidade, facilita a substituição de componentes (por exemplo, outro Vector DB ou modelo) e atende às diretrizes de modernização da execução penal por meio de uma camada de apresentação simples e auditável.

5 RESULTADOS ESPERADOS

Este capítulo descreve, de forma detalhada, os resultados que se almeja alcançar com a execução do presente Trabalho de Conclusão de Curso (TCC). Tais resultados derivam dos objetivos estabelecidos no Capítulo 1.1 e da metodologia apresentada no Capítulo 4, respeitando as normas da Associação Brasileira de Normas Técnicas – ABNT (NBR 14724:2011) para trabalhos acadêmicos.

5.1 PROPOSTA DE ARQUITETURA DE *PIPELINE* RAG FUNCIONAL

Em alinhamento ao objetivo geral e aos objetivos específicos 1 a 4 (coleta, organização, vetorização e conexão ao LLM), o trabalho propõe e desenvolve, em ambiente de prototipação, uma arquitetura de *pipeline* RAG capaz de integrar um modelo de LLM a um índice vetorial contendo documentos públicos relacionados ao SEEU, legislação penal, súmulas e doutrinas correlatas. O protótipo desenvolvido oferece interface conversacional (*chatbot*) em língua portuguesa e API REST documentada em ambiente de desenvolvimento, permitindo consultas em linguagem natural com respostas fundamentadas nas fontes originais. Espera-se que essa arquitetura proposta sirva de base conceitual e técnica para futuras implementações em ambiente de produção, correspondendo aos objetivos específicos 5, 6 e 7 (disponibilização do protótipo de *chatbot*, implementação da API REST documentada e preparação do ambiente de empacotamento).

5.2 POTENCIAL GANHO DE EFICIÊNCIA NA RECUPERAÇÃO DE INFORMAÇÕES

A arquitetura proposta visa proporcionar redução no tempo médio despendido pelos operadores do Direito para localizar e consolidar informações relacionadas ao SEEU, quando comparado ao processo manual atualmente utilizado. Embora testes formais de eficiência e usabilidade com usuários finais não tenham sido realizados no escopo deste trabalho, a fundamentação teórica e a arquitetura desenvolvida sugerem que futuras implementações, acompanhadas de ensaios controlados, poderão mensurar ganhos quantitativos de tempo de resposta, empregando métricas de precisão, *recall* e F_1. Tais avaliações constituem etapa essencial para validação em ambiente de produção e representam oportunidade para trabalhos futuros.

5.3 ESCALABILIDADE E PORTABILIDADE COMPROVADAS

Em conformidade com o objetivo específico 8 (preparar o ambiente de *deployment* com segurança e escalabilidade), a solução será entregue em contêineres Docker orquestrados

via Kubernetes, garantindo portabilidade entre ambientes e escalabilidade horizontal necessária para suportar picos de demanda. A arquitetura deverá manter latência média de recuperação inferior a 1 segundo para consultas padrão.

5.4 ALINHAMENTO INSTITUCIONAL E POTENCIAL IMPACTO SOCIAL

Almeja-se que a arquitetura proposta e o protótipo desenvolvido se alinhem às iniciativas de transformação digital do Conselho Nacional de Justiça (Programa Justiça 4.0) e aos Objetivos de Desenvolvimento Sustentável nº 16 da Organização das Nações Unidas, oferecendo base conceitual e técnica para futuras soluções que contribuam para a transparência e o acesso à justiça de populações vulnerabilizadas. A efetivação desse potencial dependerá de implementações subsequentes em ambiente de produção, acompanhadas de avaliações de impacto e adequação às políticas institucionais do Poder Judiciário.

5.5 BASE PARA MELHORIA CONTÍNUA

Será implementado um mecanismo de *feedback loop* que registre as interações dos usuários e permita o re-treinamento periódico do modelo de linguagem, assegurando a evolução constante do sistema e a adaptação às mudanças normativas ou procedimentais.

5.6 DOCUMENTAÇÃO TÉCNICA COMPLETA

Serão entregues: código-fonte comentado, arquivos `Dockerfile`, manual do desenvolvedor, manual do usuário final e documentação da API. Essa documentação facilitará a reprodutibilidade acadêmica e a eventual adoção da solução por outros órgãos do Judiciário.

5.7 MITIGAÇÃO DE RISCOS OPERACIONAIS

O projeto contemplará um plano de mitigação de riscos que inclua atualização contínua de dependências *open source*, testes automatizados de regressão e políticas de segurança da informação, garantindo a confiabilidade e a sustentabilidade da aplicação em produção.

A consecução dos resultados elencados neste capítulo demonstrará a viabilidade técnica e o impacto prático da aplicação de técnicas de RAG na execução penal brasileira, servindo de base para futuras pesquisas e para possíveis expansões em âmbito nacional.

6 MELHORIAS FUTURAS E JUSTIFICATIVA

6.1 INTRODUÇÃO

Este capítulo apresenta propostas de melhorias e extensões para a *pipeline* RAG desenvolvida neste trabalho. As sugestões são organizadas por área temática (infraestrutura, modelos e *embeddings*, indexação, módulos de extração, API/interface, testes, observabilidade e segurança) e têm por objetivo orientar trabalhos futuros, experimentos acadêmicos e evoluções para ambientes de produção.

6.2 VISÃO GERAL DAS MELHORIAS PROPOSTAS

As principais direções de melhoria identificadas incluem:

- **Escalabilidade e disponibilidade**: migração da arquitetura para suporte a índices distribuídos e serviços de alta disponibilidade em produção;
- **Qualidade dos *embeddings***: avaliação de modelos mais recentes e estratégias de *fine-tuning* específicas para o domínio jurídico;
- **Robustez dos módulos de extração**: implementação de mecanismos de recuperação automática, tolerância a bloqueios e paralelização;
- **Observabilidade e testes**: ampliação da cobertura de testes automatizados e implementação de métricas de monitoramento em tempo real;
- **Segurança e governança de dados**: fortalecimento de privacidade, controle de acesso e conformidade com a Lei Geral de Proteção de Dados (LGPD).

6.3 ARQUITETURA E INFRAESTRUTURA

Proposta de melhoria: Evoluir a arquitetura para suportar execução em produção com orquestração completa via Kubernetes, armazenamento gerenciado de índices (OpenSearch/Milvus em *cluster*) e *pipelines* de processamento baseadas em filas (RabbitMQ, Redis ou Cloud Tasks).

Justificativa: A solução atual, baseada em FAISS local e *scripts* ad hoc, é adequada para prototipagem, mas não oferece alta disponibilidade, replicação automática ou escalabilidade horizontal – requisitos essenciais para uso em produção e experimentos em larga escala.

Benefícios esperados: Tolerância a falhas, balanceamento de carga, reinício automático de serviços, capacidade de lidar com crescimento de dados e possibilidade de realizar testes A/B entre diferentes *backends* de armazenamento vetorial.

Sugestão de implementação: Definir infraestrutura mínima de produção utilizando Helm *charts*, *manifests* Kubernetes com *StatefulSets* para armazenamento vetorial persistente e volumes compartilhados. Prioridade: alta.

6.4 MODELOS DE *EMBEDDINGS* E GERENCIAMENTO DE MODELOS

Proposta de melhoria: Avaliar modelos de *embeddings* mais recentes, especialmente aqueles multilingues ou ajustados ao domínio jurídico, implementar versionamento de *embeddings* e mecanismos de re-indexação incremental. Considerar técnicas de quantização e aproximação para reduzir custos de armazenamento.

Justificativa: A qualidade da recuperação semântica depende fundamentalmente da qualidade dos *embeddings*. Modelos aprimorados ou submetidos a *fine-tuning* específico para linguagem jurídica tendem a aumentar significativamente as métricas de precisão e *recall*.

Benefícios esperados: Maior relevância nas respostas, redução de falsos positivos e melhoria nas métricas de avaliação (precisão@k, *recall*@k). O versionamento permite experimentação controlada e comparação sistemática entre modelos.

Sugestão de implementação: Adicionar módulo de experimentação que registre modelo, hiperparâmetros e *seeds* aleatórias; utilizar *pipelines* automatizadas para re-indexação incremental com registro de metadados por versão. Prioridade: alta.

6.5 INDEXAÇÃO E BACKENDS DE BUSCA VETORIAL

Proposta de melhoria: Comparar FAISS com alternativas (OpenSearch vector search, Milvus, Pinecone) considerando custo, latência e facilidade operacional; implementar suporte a *sharding* e replicação; oferecer mecanismo de *fallback* híbrido combinando busca vetorial e léxica (BM25).

Justificativa: Diferentes *backends* apresentam compromissos distintos. FAISS é rápido localmente, mas não oferece clusterização nativa; OpenSearch permite consultas híbridas e persistência integrada.

Benefícios esperados: Menor latência sob carga, maior resiliência e melhores resultados ao combinar sinais léxicos e semânticos.

Sugestão de implementação: Criar *benchmarks* automatizados e uma camada de abstração para alternar *backends* via configuração. Prioridade: média-alta.

6.6 PIPELINES DE INGESTÃO E TRATAMENTO DE DADOS

Proposta de melhoria: Tornar as *pipelines* idempotentes, tolerantes a falhas e escaláveis; adicionar etapas de normalização jurídica, incluindo extração robusta de metadados, enriquecimento semântico e deduplicação avançada.

Justificativa: A qualidade dos dados de entrada impacta diretamente a qualidade da recuperação e da avaliação do sistema. *Pipelines* manuais dificultam a reprodutibilidade e a auditoria do processo.

Benefícios esperados: Redução de ruído no índice, indexação mais rápida e reprocessamento controlado quando *embeddings* ou modelos são atualizados.

Sugestão de implementação: Adotar *frameworks* de orquestração (Airflow, Prefect) ou *jobs* em Kubernetes, com armazenamento intermediário (Parquet/NDJSON) e *checksums* para identificação de mudanças. Prioridade: média.

6.7 MÓDULOS DE EXTRAÇÃO AUTOMATIZADA (TRF4, STF, STJ)

Proposta de melhoria: Tornar os módulos de extração mais resilientes, implementando mecanismos de *retry/backoff*, tratamento de CAPTCHAs e rotação de proxies; instrumentar métricas de sucesso e erro; padronizar saída (esquemas e contratos JSONL). Modularizar os componentes para reuso e permitir execução distribuída.

Justificativa: A extração confiável garante cobertura adequada de dados e respeito às políticas dos portais. A arquitetura atual pode ser frágil frente a mudanças de estrutura (DOM) e bloqueios.

Benefícios esperados: Redução de falhas, menor necessidade de intervenção manual e melhor reprodutibilidade do conjunto de dados.

Sugestão de implementação: Encapsular interações Playwright em adaptadores testáveis, usar filas para distribuir trabalho e registrar estados em banco de dados leve (SQLite/Redis). Prioridade: alta.

6.8 API, INTERFACE E EXPERIÊNCIA DO USUÁRIO

Proposta de melhoria: Adicionar mecanismos de autenticação e autorização (API *keys*/JWT), limitação de taxa de requisições (*rate limiting*), paginação e filtros avançados; implementar na interface funcionalidades de *feedback* do usuário sobre relevância das respostas para coleta de sinais e treinamento posterior.

Justificativa: Para ambientes de produção, o controle de acesso é obrigatório. Sinais coletados dos usuários permitem medir e melhorar continuamente a relevância do sistema.

Benefícios esperados: Segurança do serviço, métricas de uso e ciclo de *feedback* humano no processo (*human-in-the-loop*) para otimização contínua.

Sugestão de implementação: Integrar FastAPI com *middleware* de autenticação; adicionar componente de avaliação de resultado na interface, conectado a *endpoint* que registra *feedback*. Prioridade: média.

6.9 TESTES, VALIDAÇÃO E REPRODUTIBILIDADE

Proposta de melhoria: Ampliar cobertura de testes (unitários, integração, *end-to-end*), adicionar testes de carga e cenários de regressão; padronizar *seeds* aleatórias, salvar pontos de referência e criar *scripts* de reprodutibilidade para experimentos.

Justificativa: A confiança nas mudanças e a capacidade de comparar configurações experimentais dependem de testes abrangentes e reprodutibilidade dos resultados.

Benefícios esperados: Ciclos de desenvolvimento mais rápidos, redução de regressões e resultados experimentais confiáveis para validação científica.

Sugestão de implementação: Ampliar conjunto de testes com *fixtures* que simulem armazenamentos e modelos; integrar integração contínua (CI) via GitHub Actions para execução automática de testes e *benchmarks*. Prioridade: alta.

6.10 OBSERVABILIDADE E OPERAÇÕES

Proposta de melhoria: Instrumentar serviços com métricas (Prometheus), rastreamento distribuído (OpenTelemetry) e *logs* estruturados; criar painéis de monitoramento (Grafana) com latência, vazão (*throughput*), erros e qualidade de resposta.

Justificativa: O monitoramento é fundamental para identificar gargalos e regressões, além de apoiar decisões de otimização e escalonamento.

Benefícios esperados: Visibilidade operacional, diagnósticos mais rápidos e fundamentação para decisões de escalonamento e ajuste fino.

Sugestão de implementação: Adicionar *middleware* para métricas no FastAPI, exportadores para Prometheus e coletores de rastreamento. Prioridade: média.

6.11 SEGURANÇA, PRIVACIDADE E GOVERNANÇA DE DADOS

Proposta de melhoria: Implementar anonimização de dados sensíveis, políticas de retenção, *logs* auditáveis e controle de acesso granular; avaliar conformidade com normas aplicáveis, especialmente a Lei Geral de Proteção de Dados (LGPD).

Justificativa: O projeto lida com documentos jurídicos que podem conter dados pessoais. A conformidade legal é imprescindível para uso além do contexto de pesquisa acadêmica.

Benefícios esperados: Mitigação de riscos legais, maior aceitação institucional e proteção adequada de dados pessoais.

Sugestão de implementação: Criar módulo de sanitização de textos antes da indexação, implementar consentimento informado e processar dados pessoais com marcadores e técnicas de ocultação (*redaction*). Prioridade: alta.

6.12 DOCUMENTAÇÃO, REPRODUTIBILIDADE E PROCESSO DE PESQUISA

Proposta de melhoria: Centralizar documentação operacional, apresentar instruções reproduzíveis para experimentos (datasets, *seeds*, versões de modelos) e exemplos "how-to" para ingressar novos colaboradores.

Justificativa: A documentação abrangente facilita a avaliação no contexto acadêmico e acelera a adoção por outros pesquisadores.

Benefícios esperados: Redução da barreira de entrada, facilidade para replicação e validação científica dos resultados.

Sugestão de implementação: Atualizar documentação principal do projeto, criar manuais operacionais e adicionar seção de experimentos com *scripts* automatizados. Prioridade: média.

6.13 CRONOGRAMA DE IMPLEMENTAÇÃO SUGERIDO

Com base nas prioridades identificadas, propõe-se o seguinte cronograma de implementação das melhorias:

1. **Curto prazo (1 a 3 meses):** Fortalecimento dos módulos de extração, versionamento de *embeddings*, implementação de autenticação na API e ampliação de testes unitários.

2. **Médio prazo (3 a 9 meses):** Orquestração de *pipelines*, avaliação comparativa de *backends* vetoriais, implementação de monitoramento básico com Prometheus e Grafana.

3. **Longo prazo (9 a 18 meses):** Migração para infraestrutura clusterizada com Kubernetes, suporte a reindexação em larga escala, *fine-tuning* de modelos jurídicos e publicação de conjunto de dados documentado para reprodutibilidade.

6.14 CONSIDERAÇÕES FINAIS DO CAPÍTULO

As melhorias propostas neste capítulo visam evoluir a *pipeline* RAG implementada em uma plataforma robusta para experimentação acadêmica e, potencialmente, uso em ambientes de produção. As prioridades destacadas equilibram ganhos de qualidade (aprimoramento de *embeddings*, *pipelines* e módulos de extração) com necessidades operacionais (segurança, observabilidade e infraestrutura escalável). Implementações incrementais, acompanhadas de testes sistemáticos e documentação adequada, permitirão avaliar o impacto de cada melhoria e ajustar a trajetória conforme os resultados experimentais obtidos.

6.15 RECOMENDAÇÕES PARA TRABALHOS FUTUROS

Com base na análise apresentada, recomendam-se as seguintes ações para trabalhos futuros:

- Selecionar 2 a 3 itens prioritários (por exemplo: robustez dos módulos de extração, versionamento de *embeddings* e ampliação de testes com integração contínua) e estabelecer tarefas com critérios de aceitação bem definidos.
- Implementar avaliação comparativa (*benchmark*) entre FAISS e OpenSearch considerando latência e precisão no conjunto de dados atual.
- Planejar revisão de segurança e privacidade com orientador e, se necessário, consultor especializado em proteção de dados.

- Selecionar 2 a 3 itens prioritários (por exemplo: robustez dos módulos de extração, versionamento de *embeddings* e ampliação de testes com integração contínua) e estabelecer tarefas com critérios de aceitação bem definidos.
- Implementar avaliação comparativa (*benchmark*) entre FAISS e OpenSearch considerando latência e precisão no conjunto de dados atual.
- Planejar revisão de segurança e privacidade com orientador e, se necessário, consultor especializado em proteção de dados.

REFERÊNCIAS

AQUINO, I. V. d. **Extracting Information from Brazilian Legal Documents with Retrieval-Augmented Generation**. Dissertação (Trabalho de Conclusão de Curso (Bacharel em Ciências da Computação)) — Universidade Federal de Santa Catarina, Florianópolis, Brazil, 2024. Acesso em: nov. 2025. Disponível em: <https://repositorio.ufsc.br/handle/123456789/262519>.

BELARMINO, M. e. a. Aplicação de large language models na análise e síntese de documentos jurídicos: Uma revisão de literatura. **arXiv preprint**, abs/2504.00725v1, 2025. Acesso em: nov. 2025. Disponível em: <http://arxiv.org/abs/2504.00725v1>.

Brasil. **Lei n.º 7.210, de 11 de julho de 1984 (Lei de Execução Penal)**. 1984. Diário Oficial da União, 13 jul. 1984. Acesso em: nov. 2025. Disponível em: <https://www.planalto.gov.br/ccivil_03/leis/l7210.htm>.

Conselho Nacional de Justiça. **Portaria n.º 304, de 14 de março de 2024**. 2024. Acesso em: nov. 2025. Disponível em: <https://atos.cnj.jus.br/atos/detalhar/4659>.

DIVALD, S. **E-formalization case study: e-Estonia: a digital society for the transition to formality**. Genebra: International Labour Office, 2021. Acesso em: nov. 2025. ISBN 9789220342244. Disponível em: <https://www.ilo.org/publications/e-estonia-digital-society-transition-formality>.

Docker Documentation. **Docker Overview**. 2024. <https://docs.docker.com/get-started/docker-overview/>. Accessed: 2025-06-02.

EDWARDS, C. Hybrid context retrieval augmented generation pipeline: Llm-augmented knowledge graphs and vector database for accreditation reporting assistance. **ArXiv preprint**, abs/2405.15436v1, 2024. Disponível em: <https://arxiv.org/abs/2405.15436v1>.

Facebook Research. **FAISS**. 2024. Acesso em: nov. 2025. Disponível em: <https://github.com/facebookresearch/faiss>.

GAO, Y. e. a. Retrieval-augmented generation for large language models: A survey. **arXiv preprint**, arXiv:2312.10997, 2023. Acesso em: nov. 2025. Disponível em: <https://arxiv.org/abs/2312.10997>.

HARRIS, C. R.; MILLMAN, K. J.; WALT, S. J. van der; GOMMERS, R.; VIRTANEN, P.; COURNAPEAU, D.; WIESER, E.; TAYLOR, J.; BERG, S.; SMITH, N. J.; KERN, R.; PICUS, M.; HOYER, S.; KERKWIJK, M. H. van; BRETT, M.; HALDANE, A.; RÍO, J. F. del; WIEBE, M.; PETERSON, P.; GÉRARD-MARCHANT, P.; SHEPPARD, K.; REDDY, T.; WECKESSER, W.; ABBASI, H.; GOHLKE, C.; OLIPHANT, T. E. Array programming with NumPy. **Nature**, Springer Science and Business Media LLC, v. 585, n. 7825, p. 357–362, set. 2020. Disponível em: <https://doi.org/10.1038/s41586-020-2649-2>.

IBM. **IBM Vector Database**. 2025. Acesso em: nov. 2025. Disponível em: <https://www.ibm.com/think/topics/vector-database>.

Kubernetes Documentation. **Concepts Overview**. 2025. <https://kubernetes.io/docs/concepts/overview/>. Acesso em: 02 jun. 2025.

LAI, J. e. a. Large language models in law: A survey. **arXiv preprint 2312.03718v1**, 2023. Acesso em: nov. 2025. Disponível em: <https://arxiv.org/abs/2312.03718>.

LangChain. **LangChain**. 2024. Acesso em: nov. 2025. Disponível em: <https://github.com/hwchase17/langchain>.

LEWIS, P. e. a. Retrieval-augmented generation for knowledge-intensive nlp tasks. In: **Advances in Neural Information Processing Systems**. Vancouver, Canada: [s.n.], 2020. v. 33, p. 9459–9474. Acesso em: nov. 2025. Disponível em: <https://proceedings.neurips.cc/paper/2020/file/6b493230205f780e1bc26945df7481e5-Paper.pdf>.

MAGEIRAKOS, V.; WU, B.; ALONSO, G. Cracking vector search indexes. **arXiv preprint arXiv:2503.01823**, 2025. Acesso em: nov. 2025. Disponível em: <https://arxiv.org/pdf/2503.01823>.

NAVEEDA, H. e. a. A comprehensive overview of large language models. **arXiv preprint arXiv:2307.06435v10**, 2024. Acesso em: nov. 2025. Disponível em: <http://arxiv.org/abs/2307.06435v10>.

Nuxt Team. **Introduction – Get Started with Nuxt**. 2024. Acesso em: nov. 2025. Disponível em: <https://nuxt.com/docs>.

Oracle. **Oracle Vector Database**. 2025. Acesso em: nov. 2025. Disponível em: <https://www.oracle.com/br/database/vector-database/>.

PEDREGOSA, F. e. a. Scikit-learn: Machine learning in Python. **Journal of Machine Learning Research**, v. 12, p. 2825–2830, 2011.

PUJIONO, I.; AGTYAPUTRA, I. M.; RULDEVIYANI, Y. Implementing retrieval-augmented generation and vector databases for chatbots in public services agencies context. **5572-Article Text**, 2024. Acesso em: nov. 2025. Disponível em: <https://ejournal.nusamandiri.ac.id/index.php/jitk/article/view/5572>.

Python Software Foundation. **Python Language Reference**. 2024. Acesso em: nov. 2025. Disponível em: <https://www.python.org>.

Qwak. **Integrating Vector Databases with LLMs: A Hands-On Guide**. 2024. Acesso em: nov. 2025. Disponível em: <https://www.qwak.com/post/utilizing-llms-with-embedding-stores>.

RAMíREZ, S. **FastAPI**. 2024. Acesso em: nov. 2025. Disponível em: <https://fastapi.tiangolo.com/>.

SALEMI, A.; ZAMANI, H. Evaluating retrieval quality in retrieval-augmented generation. In: **Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '24)**. Washington, DC, USA: [s.n.], 2024. p. 1–6. Disponível em: <https://doi.org/10.1145/3626772.3657957>.

SHI, Y. e. a. Scalable overload-aware graph-based index construction for 10-billion-scale vector similarity search. In: **Companion Proceedings of the ACM on Web Conference 2025**. [s.n.], 2025. p. 1303–1307. Acesso em: nov. 2025. Disponível em: <https://arxiv.org/html/2502.20695v1>.

Superior Tribunal de Justiça. **Súmula 533**. 2015. Acesso em: nov. 2025. Disponível em: <https://www.stj.jus.br/sites/portalp/Paginas/stj/sumula-533>.

TAIPALUS, T. Vector database management systems: Fundamental concepts, use-cases, and current challenges. **Cognitive Systems Research**, 2024. ISSN 1389-0417. Acesso em: nov. 2025. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1389041724000093>.

The Pandas Development Team. **Pandas**. 2024. Acesso em: nov. 2025. Disponível em: <https://pandas.pydata.org>.

Tribunal de Justiça da Bahia. **Mutirão Processual Penal do TJ-BA contabiliza mais de 18,6 mil atos processuais no SEEU**. 2024. Acesso em: nov. 2025. Disponível em: <https://www.tjba.jus.br/portal/mutirao-processual-penal-do-tjba-contabiliza-mais-de-186-mil-atos-processuais-no-seeu>.

Tribunal de Justiça do Ceará. **Novos robôs realizam 146 despachos em 1 h 15 min**. 2023. Acesso em: nov. 2025. Disponível em: <https://www.tjce.jus.br/noticias/novos-robos-realizam-146-despachos-em-1h15min/>.

UNDP. **PNUD e CNJ assinam projeto focado em direitos humanos e acesso à Justiça**. 2024. Acesso em: nov. 2025. Disponível em: <https://www.undp.org/pt/brazil/news/pnud-e-cnj-assinam-projeto-focado-em-direitos-humanos-e-acesso-justica>.

UNDP. **O PNUD e a ONU**. 2025. Acesso em: nov. 2025. Disponível em: <https://www.undp.org/pt/brazil/o-pnud-e-onu>.

UNDP. **Sobre o PNUD**. 2025. Acesso em: nov. 2025. Disponível em: <https://www.undp.org/pt/brazil/sobre-o-pnud>.

VASWANI, A. e. a. Attention is all you need. In: **Advances in Neural Information Processing Systems**. [s.n.], 2017. Acesso em: nov. 2025. Disponível em: <https://arxiv.org/pdf/1706.03762.pdf>.

Vue.js Team. **Introduction – Vue.js**. 2024. Acesso em: nov. 2025. Disponível em: <https://vuejs.org/guide/introduction>.

Wikipedia. **Programa das Nações Unidas para o Desenvolvimento**. 2025. Acesso em: nov. 2025. Disponível em: <https://pt.wikipedia.org/wiki/Programa_das_Na%C3%A7%C3%B5es_Unidas_para_o_Desenvolvimento>.

YUE, S. e. a. Disc-lawllm: Fine-tuning large language models for intelligent legal services. **arXiv preprint**, 2023. Acesso em: nov. 2025. Disponível em: <https://github.com/FudanDISC/DISC-LawLLM>.

ZHANG, Z. e. a. Fine-grained retrieval-augmented generation for visual question answering. **arXiv preprint arXiv:2502.20964**, 2025. Acesso em: nov. 2025. Disponível em: <https://arxiv.org/pdf/2502.20964>.