Documento Técnico Detalhado: Sistema RAG Jurídico

Sumário

Este documento detalha a arquitetura do Sistema de Recuperação Aumentada por Geração (RAG) especializado em legislação do Estado de Santa Catarina. O projeto é estruturado em duas fases principais – Ingestão e Consulta – e utiliza um Pipeline Inteligente de **3 Rotas** baseado em classificação de intenção, visando alta precisão jurídica (Grounding de Vigência) e otimização de custos e latência de resposta.

1. Visão Geral e Arquitetura do Sistema

O sistema é um **Chatbot de Recuperação Aumentada por Geração (RAG)**, projetado para fornecer respostas precisas, fundamentadas e rastreáveis sobre a legislação de Santa Catarina. A principal função do RAG é mitigar o problema da "Alucinação" (invenção de fatos e informações) inerente aos Grandes Modelos de Linguagem (LLMs), fornecendo contexto verificável (o corpus legal de SC) antes da geração da resposta.

A arquitetura adota uma abordagem de duas fases críticas:

1. **Ingestão (Offline):** Processamento inicial e criação do índice de busca vetorial.
2. **Consulta (Online):** O Pipeline de 3 Rotas que roteia a query do usuário para a estratégia de busca mais eficiente.

Esta separação garante que a parte mais custosa e demorada do processo (criação de embeddings) seja feita apenas uma vez, otimizando o serviço online para **baixa latência e alto throughput**.

2. Fase de Ingestão e Indexação (`src/data_prep.py`)

Esta fase é o motor de preparação dos dados, implementado como um processo **ETL (Extract, Transform, Load)** que converte documentos HTML brutos em artefatos de dados de busca. O arquivo responsável por esta lógica é o `src/data_prep.py`.

2.1. Processo ETL e Normalização (Limpeza de Dados)

O script executa as seguintes operações em sequência:

2.1.1. E - Extração (Extract)

- **Mecanismo:** Uso da biblioteca `BeautifulSoup` para extrair o texto principal dos arquivos HTML das leis.

- **Resultado:** Obtenção do conteúdo textual bruto.

2.1.2. T - Transformação (Transform)

Esta é a etapa mais crítica para a acurácia jurídica:

- **Extração de Metadados:**
  - **Metadados Chave:** O *Ano de Publicação* e o *Nome do Arquivo* são extraídos.
  - **Função:** Estes metadados são essenciais para o **Grounding de Vigência** na Fase de Consulta, permitindo ao LLM citar a origem e a data da lei.
- **Remoção de Conteúdo Taxado (Conteúdo Revogado):**
  - **Lógica:** Implementação de uma lógica robusta que busca e remove trechos de texto que contêm tags HTML (`<s>` ou `<!-- revogado -->`) ou notações explícitas de revogação.
  - **Impacto:** Garante que apenas o conteúdo **potencialmente vigente** seja indexado, eliminando a chance de o chatbot responder com leis obsoletas.
- **Chunking Estratégico:**
  - **Definição:** O texto longo é dividido em unidades lógicas chamadas `chunks` (pedaços).
  - **Tamanho Otimizado:** O limite é de aproximadamente **`1500 caracteres`** (`CHUNK_SIZE_LIMIT`). Este tamanho é ideal, pois é grande o suficiente para capturar contexto semântico completo (artigos inteiros) e pequeno o suficiente para caber na janela de contexto do LLM e ser processado eficientemente pelo modelo de embedding.
  - **Sobreposição (`CHUNK_OVERLAP`):** Aplica-se uma sobreposição de 10% a 20% entre chunks adjacentes. **Propósito:** Evitar a perda de contexto semântico para frases ou sentenças que transbordam a fronteira de um chunk.
  - **Prefixação de Contexto:** Cada `chunk` é **prefixado** com um cabeçalho contendo a `fonte` e o `ano de publicação`. Este cabeçalho é o que o LLM vê e usa para fundamentar a resposta.

2.1.3. L - Carga (Load)

Os chunks processados são vetorizados e carregados nos artefatos de indexação.

2.2. Artefatos de Indexação (Saída)

O resultado da fase de Ingestão são os artefatos de dados carregados na memória na inicialização do Streamlit:

Artefato	Conteúdo	Detalhamento Técnico	Função no Sistema Online
`faiss_index.bin`	Armazena os **vetores** (embeddings) dos `chunks`. Um embedding é a representação numérica de alta dimensão de um	O FAISS (Facebook AI Similarity Search) é uma biblioteca otimizada para busca eficiente de	Usado para a **Busca Vetorial Rápida** (`index.search`) e encontrar os `chunks` mais semanticamente próximos à

	texto.	vizinhos mais próximos em espaços vetoriais, usando o cálculo de similaridade cosseno.	pergunta do usuário, independentemente das palavras-chave.
`documents_map.json`	Mapeia o ID de cada vetor (`doc_1`, `doc_2`, etc.) ao seu **texto original completo** e seus **metadados**.	É um dicionário JSON simples que serve como o **armazenamento de dados** (data store) onde o texto legível é recuperado.	Usado para **Recuperar** o texto real do chunk após o FAISS retornar o ID.
`chunks_processados.json`	**ARTEFATO INTERMEDIÁRIO (Não usado em Prod.)**.	Mantido apenas para **Auditoria e Depuração** do processo de `chunking` e limpeza.	Não é carregado no Streamlit, otimizando memória e tempo de inicialização.

3. Fase de Consulta e Geração (`src/rag_service.py`)

O arquivo `src/rag_service.py` orquestra a inteligência do sistema, gerenciando o fluxo de consulta em tempo real e a interação com o modelo Gemini.

3.1. Pipeline de Consulta 3-em-1 (Rotas Inteligentes)

O sistema emprega uma pipeline de decisão que usa o modelo `Gemini-2.5-Flash` para classificar a intenção do usuário **antes** de qualquer busca vetorial. Isso permite um roteamento inteligente, garantindo que recursos caros (busca e geração LLM) sejam usados apenas quando estritamente necessário.

3.1.1. Classificação de Intenção e Reescrita de Query (O Modelo de Roteamento)

- **Entrada:** A consulta atual do usuário **e** o histórico de conversa (para contexto).
- **Mecanismo:** O LLM é instruído via System Prompt a retornar um objeto JSON estruturado com duas chaves: a classificação e, se necessário, a query reescrita.
- **Classificação Detalhada:**

- **'C' (Não Jurídica/Off-Topic):** Perguntas irrelevantes ou fora do escopo (ex: "Qual a capital do Brasil?").
- **'A' (Jurídica Direta):** Pergunta clara e completa sobre o corpus (ex: "Qual o artigo 5º da LC 412?").
- **'B' (Jurídica Conversacional):** Pergunta que depende do diálogo anterior (ex: "E quanto à alíquota do artigo anterior?").

- **Reescrita de Query:**
- Para as intenções 'A' e 'B', o LLM gera uma `rewritten_query`. Este é o **passo mais crucial do RAG conversacional**, transformando entradas dependentes de contexto em consultas claras e **autocontidas** para o mecanismo de busca FAISS.

3.1.2. Fluxo de Execução (O Roteamento)

Rota	Intenção	Ação no Sistema	Otimização
Rota C	Não Jurídica	**Ignora a Busca RAG (FAISS)**. Envia o prompt ao LLM com instrução para recusar educadamente, focando no domínio de leis de SC.	**Economia de Custos e Latência:** Evita o custo de embedding e busca vetorial desnecessários.
Rotas A e B	Jurídica	Prossegue para a Busca e Geração RAG, utilizando a query reescrita.	**Acurácia:** Garante que a busca FAISS utilize a query mais precisa e completa possível.

3.1.3. Busca e Geração RAG (Rotas A e B)

1. **Vetorização:** A `rewritten_query` é vetorizada (gera um novo embedding).
2. **Busca:** O `faiss_index.bin` é consultado para recuperar os `TOP_K_CHUNKS` (geralmente 5) mais relevantes com base na similaridade.
3. **Geração:** O texto dos chunks (o contexto) é enviado ao LLM junto com a `SYSTEM_INSTRUCTION`.
4. **Streaming:** O LLM gera a resposta final **em streaming** (caractere por caractere) para garantir uma experiência de usuário de baixa latência.

3.2. Estratégia de Grounding e Citação de Vigência

A `SYSTEM_INSTRUCTION` é o ponto central de controle do LLM. Ela é altamente restritiva, forçando o modelo a citar a fonte e a lidar com o problema da vigência legal:

Requisito Jurídico	Mecanismo de Implementação	Função da SYSTEM_INSTRUCTION (Regra de Grounding)
Artigos Alterados	O `chunk` contém o texto atualizado **e** o metadado	O LLM é obrigado a citar a **Lei Alteradora** como prova da vigência,

	da **Lei Alteradora** (ex: "Redação dada pela LC 789"), inserido na fase de Ingestão.	demonstrando que a redação mais recente foi utilizada, não a original.
Artigos Não Alterados (Original)	O `src/data_prep.py` injetou o **Ano de Publicação** no cabeçalho do `chunk` original.	O LLM é instruído a usar o **Ano de Publicação** (no cabeçalho) para declarar que "a redação apresentada é a vigente desde [Ano]", tratando artigos que nunca foram alterados, mas que precisam de uma prova de vigência.

4. Estrutura de Arquivos e Deploy

O deploy é otimizado para **Streamlit Community Cloud**, onde a performance de inicialização depende do carregamento eficiente dos artefatos de dados.

Arquivo	Propósito	Detalhamento de Função	Dependências Críticas
`src/data_prep.py`	Ingestão e Indexação (Offline).	Executa todo o pipeline ETL e de Chunking Estratégico para criar os artefatos de busca.	`google-genai`, `faiss-cpu`, `beautifulsoup4`, `tenacity`.
`src/rag_service.py`	**Core Logic (Coração do RAG)**.	Contém a `SYSTEM_INSTRUCTION`, a lógica de **Classificação/Reescrita (3 Rotas)** e orquestra a Busca FAISS e a chamada de Geração **com Streaming**.	`google-genai`, `faiss-cpu`, `numpy`, `tenacity`.
`app_web.py`	Interface de Usuário (Streamlit).	Gerencia o estado da sessão, a interface de chat, carrega o RAG na inicialização e exibe o fluxo de respostas.	`streamlit`, `src/rag_service.py`.
`requirements.txt`	Lista de Bibliotecas.	Lista todas as bibliotecas Python necessárias para o ambiente de	Todas as dependências acima.

		execução no Streamlit Cloud.	
`documents_map.json`	Artefato de Indexação.	O data store que armazena o texto legível dos chunks.	
`faiss_index.bin`	Artefato de Indexação.	O índice vetorial para busca semântica de alta velocidade.	