GraphRAG Pipeline Visualizer: Arquitetura de Recuperação Aumentada por Grafos

1. Resumo Executivo

Este repositório hospeda a implementação de referência de uma pipeline **GraphRAG (Graph-based Retrieval-Augmented Generation)**. Diferentemente de sistemas RAG tradicionais, que dependem exclusivamente de busca vetorial (*vector search*) baseada em similaridade de cosseno em um espaço latente plano, esta arquitetura constrói um **Grafo de Conhecimento Estruturado** a partir de documentos não estruturados (PDFs).

O sistema integra **Modelos de Linguagem Grande (LLMs - Google Gemini 2.0)** para extração semântica, **Redes Neurais Convolucionais (CNNs)** com função de perda **Triplet Loss** para refinamento de embeddings, e algoritmos de **Teoria dos Grafos** para detecção de comunidades e centralidade. O objetivo primário é mitigar alucinações estocásticas e permitir inferências do tipo *multi-hop* (conexão lógica de conceitos distantes) através de topologia explícita.

---

2. Arquitetura da Pipeline (Metodologia)

A pipeline é segmentada em 4 estágios macro, subdivididos em processos atômicos auditáveis. Abaixo detalha-se o funcionamento técnico, a justificativa teórica e o diferencial de engenharia de cada etapa.

2.1. Ingestão e Pré-processamento Semântico (Stage: UPLOAD)

Objetivo

Transformação de arquivos PDF binários em unidades de texto processáveis (*chunks*), preservando rigorosamente a hierarquia documental e o contexto semântico.

Procedimento Técnico

1. **Extração via PDF.js:** Leitura bruta dos bytes e conversão para string, com tratamento de *encoding*.
2. **Limpeza Heurística:** Remoção de artefatos de OCR, hifens de quebra de linha e cabeçalhos/rodapés repetitivos que introduzem ruído no espaço vetorial.
3. **Chunking Hierárquico:** Segmentação baseada na estrutura lógica do documento (ex: Artigos Jurídicos, Seções Acadêmicas), em detrimento da contagem arbitrária de tokens.
4. **Enriquecimento via LLM (Gemini 2.0 Flash):** Cada chunk é submetido a uma inferência para:
    - **Classificação Taxonômica:** (ex: "Definição", "Metodologia", "Inciso Legal").

- **Reconhecimento de Entidades Nomeadas (NER):** Extração de palavras-chave fundamentais.
- **Rotulagem Sintética:** Geração de títulos descritivos para facilitar a indexação.

Diferencial & Justificativa

O *Naive Chunking* (corte fixo a cada N tokens) fragmenta contextos semânticos, prejudicando a recuperação. Nossa abordagem hierárquica preserva a unidade de sentido (o "átomo" de informação). O enriquecimento via LLM injeta metadados explícitos que não existem no texto bruto, aumentando a precisão da vetorização subsequente.

2.2. Vetorização e Embeddings (Stage: EMBEDDINGS)

Objetivo

Mapeamento do texto enriquecido para vetores numéricos de alta dimensão (*High-Dimensional Vectors*), convertendo linguagem natural em representações matemáticas processáveis.

Procedimento Técnico

- **Modelo Base:** `text-embedding-004` (Google Gemini) ou fallback para `Sentence-BERT`.
- **Input Rico (Rich Input):** O vetor não é gerado apenas do corpo do texto. A entrada é concatenada da seguinte forma:
  $$Input = [Tipo_{Entidade}] \oplus [Keywords] \oplus [Conteúdo]$$
- **Dimensionalidade:** 768 dimensões.

Diferencial & Justificativa

Ao incorporar metadados (tipo e keywords) no input do embedding, força-se o modelo vetorial a "atentar" para as entidades principais e a estrutura, não apenas para a sintaxe da frase. Isso resulta em vetores que agrupam melhor por tópico e função.

2.3. Refinamento Vetorial via CNN e Triplet Loss (Otimização)

Objetivo

Ajuste fino (*Fine-Tuning*) das posições dos vetores no hiperespaço para maximizar a coesão intraclasse e a separação interclasse, utilizando Aprendizado Supervisionado por Métricas.

Procedimento Técnico

1. **Arquitetura:** Implementação de uma **CNN 1D** otimizada para sequências.
2. **Função de Perda (Loss Function):** Utilização da **Triplet Loss**.
   $$L(A, P, N) = \max(||f(A) - f(P)||^2 - ||f(A) - f(N)||^2 + \alpha, 0)$$
   - Onde A é a âncora, P é positivo (mesma classe/keyword) e N é negativo (classe distinta). α é a margem de separação.

3. **Validação Cruzada (Cross-Validation):**
    - **Estratégia de Split 80/20:** 80% dos vetores compõem o conjunto de treino (onde ocorre a retropropagação do gradiente) e 20% formam o conjunto de validação (para monitoramento de generalização).
    - **Otimizador:** AdamW com decaimento de peso (*weight decay*) para regularização.

Diferencial & Justificativa

Embeddings pré-treinados (como o da OpenAI ou Google) são genéricos. Nosso refinamento adapta a distribuição espacial dos vetores ao **domínio específico** dos documentos carregados. O uso de Triplet Loss é o estado da arte (SOTA) para aprendizado de representações, garantindo que conceitos semanticamente similares fiquem matematicamente próximos.

2.4. Clusterização e Construção do Grafo (Stage: CLUSTERING & GRAPH)

Objetivo

Transformação da nuvem de pontos vetorial em uma estrutura topológica de nós e arestas, permitindo análise de rede.

Procedimento Técnico (Clusterização)

- **Algoritmo:** K-Means++ com determinação dinâmica de K ($\approx \sqrt{N/2}$).
- **Validação:** Cálculo do **Silhouette Score** para medir a consistência dos agrupamentos.
- **Projeção:** Redução de dimensionalidade para visualização 2D (similar a UMAP).

Procedimento Técnico (Arestas Híbridas)

A conexão entre dois nós (A e B) não é binária. O peso da aresta W_{AB} é calculado por uma função de custo composta:

$$W_{AB} = (\text{Overlap}(A,B) \times 0.6) + (\text{Jaccard}(A,B) \times 0.4)$$

- **Interseção Semântica (Jaccard):** Baseada nas palavras-chave extraídas pela IA.
- **Coeficiente de Sobreposição (Overlap):** Útil para detectar relações de subconjunto (hierarquia).
- **Filtro de Confiança:** Arestas com $W_{AB} < 0.35$ são descartadas para reduzir ruído (sparsification).

Diferencial & Justificativa

A maioria dos RAGs utiliza apenas *K-Nearest Neighbors (KNN)*. Nós criamos arestas explícitas baseadas em **vocabulário compartilhado** e **topologia**. Isso permite detectar comunidades temáticas (ex: cluster de "Direito Penal") e calcular métricas de centralidade (identificando os conceitos "Hub" do documento).

📊 3. Métricas de Auditoria e Qualidade

O sistema gera automaticamente um **Relatório Técnico (Qualis A1)** contendo indicadores fundamentais para validação científica:

1. **Modularidade (Q):** Mede a força da divisão do grafo em módulos. $Q > 0.4$ indica estrutura comunitária robusta.
2. **Densidade do Grafo:** Razão entre arestas existentes e possíveis. Controla a dispersão da informação.
3. **Silhouette Score:** Validação da consistência dos clusters (intervalo -1 a 1). Valores > 0.5 indicam alta coesão.
4. **Centralidade (Degree/Betweenness):** Identificação matemática dos nós mais influentes na rede.

🚀 4. Guia de Instalação e Uso

Para executar o projeto, siga os passos abaixo:

Pré-requisitos

- **Node.js v18 ou superior** - Download
- **Chave de API do Google Gemini** - Obtenha aqui

Instalação e Configuração

```
# 1. Clonar o repositório (ou fazer download)
git clone https://github.com/seu-user/graphrag-visualizer.git
cd GraphRAG-Pipeline---SANDECO-main

# 2. Instalar dependências
npm install

# 3. Configurar Variáveis de Ambiente
# Copie o arquivo de exemplo
cp .env.example .env

# Edite o arquivo .env e adicione sua chave da API do Gemini:
# API_KEY=sua_chave_api_aqui
```

Nota Importante sobre a API Key:

- Abra o arquivo `.env` no editor de texto
- Substitua `sua_chave_api_aqui` pela sua chave real do Google Gemini
- A chave deve ter o formato: `AIzaSyABCDEFGHIJKLMNOPQRSTUVWXYZ...`

```
# 4. Iniciar o servidor de desenvolvimento
npm run dev
```

A aplicação estará disponível em `http://localhost:5173`

Protocolo de Execução do Pipeline

1. **Etapa 1 - Upload:** Acesse a interface e faça upload de PDFs (documentos acadêmicos, jurídicos ou técnicos recomendados).
2. **Etapa 1 - Enriquecimento IA:** Clique em **"🤖 Enriquecer com Gemini AI"** para:
    - Limpar e normalizar o texto
    - Classificar hierarquicamente cada chunk
    - Extrair palavras-chave e entidades
3. **Etapa 2 - Embeddings:**
    - Selecione o modelo de embedding (recomendado: **Gemini text-embedding-004** para máxima qualidade)
    - Clique em **"Gerar Embeddings"**
    - Para embeddings locais, escolha Sentence-BERT ou USE
4. **Etapa 3 - Refinamento CNN (Opcional):**
    - Configure hiperparâmetros:
        - Learning Rate: `0.005`
        - Margin: `0.2`
        - Mining Strategy: `hard` (recomendado)
        - Epochs: `15-20`
    - Clique em **"🧠 Treinar CNN"**
    - Acompanhe as métricas de treino e validação em tempo real
5. **Etapa 4 - Clusterização:** Execute a clusterização com K-Means++ e visualize a distribuição espacial dos vetores
6. **Etapa 5 - Construção do Grafo:** Gere o grafo de conhecimento com arestas ponderadas baseadas em:
    - Similaridade semântica (Jaccard Index)
    - Co-ocorrência de palavras-chave
    - Relações hierárquicas
7. **Análise e Exportação:**
    - Explore o painel de "Análise de Clusters" para identificar tópicos emergentes
    - Visualize métricas do grafo (densidade, modularidade, centralidade)
    - Exporte dados em CSV
    - Gere relatório técnico completo

⚠ 5. Limitações Conhecidas

- **Custo Computacional Client-Side:** O refinamento da CNN é executado no navegador. Para datasets massivos (>10k chunks), recomenda-se a migração para um backend Python (PyTorch/TensorFlow).
- **Dependência de LLM:** A qualidade final do grafo é diretamente proporcional à qualidade da extração de entidades realizada pelo Gemini na Etapa 1.
- **Janela de Contexto:** Referências que cruzam chunks muito distantes podem perder a conexão direta se não houver vocabulário compartilhado explícito.

👨‍💻 6. Autoria e Créditos

Desenvolvido como prova de conceito para arquiteturas avançadas de Sistemas de Recuperação de Informação.

- **Engenharia de Prompt:** Otimizada para Gemini 2.0 Flash.
- **Visualização de Dados:** D3.js Force Simulation e Recharts.
- **AUTOR :** Prof. Marcelo Claro Laranjeira
- **Padrão de Projeto:** Programação Reativa Funcional (React Hooks).