Relatório Técnico: Implementação de Sistema RAG

Autor: Mateus Augusto de Souza
Data: 07/08/2025
Teste Técnico: Analista de Dados Sênior

---

1. Introdução

Este documento apresenta a arquitetura, as decisões técnicas e os resultados da implementação de um sistema de **Geração Aumentada por Recuperação (RAG)**. O objetivo do projeto foi desenvolver um pipeline capaz de responder perguntas de usuários com base em uma **base de conhecimento institucional**, garantindo respostas **precisas, relevantes e com fontes citadas**.

O sistema foi organizado em quatro módulos principais:

- **Processamento de Documentos:** Carregamento, limpeza e segmentação dos textos.
- **Base Vetorial:** Geração de embeddings e armazenamento para busca semântica.
- **Pipeline RAG:** Orquestração da busca de contexto e geração de respostas via LLM.
- **Avaliação e Monitoramento:** Medição da qualidade com RAGAS e observabilidade via Phoenix.

---

2. Decisões Técnicas e Arquitetura

A arquitetura do sistema foi projetada para ser **modular e escalável**, separando responsabilidades em diferentes serviços.

2.1. Pré-processamento e Chunking

A qualidade da base de conhecimento é crítica para o sucesso de um sistema RAG. Assim, foi implementada uma etapa de pré-processamento.

- **Carregamento e Extração de Metadados:**
  Documentos .txt são carregados de diretórios específicos, e metadados como **Título** e **Categoria** são extraídos do próprio conteúdo para enriquecer a base.
- **Limpeza de Ruído com Perplexidade:**
  Para garantir a indexação apenas de informações de alta qualidade, implementou-se um filtro baseado em perplexidade.
    - **Tecnologia:** pierreguillou/gpt2-small-portuguese (HuggingFace).
    - **Justificativa:** A perplexidade mede o quão "surpreendente" uma sentença é para o modelo. Sentenças com perplexidade acima de PERPLEXITY_THRESHOLD = 600 foram consideradas ruído e removidas, automatizando a limpeza de dados e melhorando a qualidade do contexto fornecido ao LLM.

- **Observação:** Para bases maiores, recomenda-se calcular perplexidades com spaCy + PyTorch e gerar um PERPLEXITY_THRESHOLD dinâmico via percentis.

- **Chunking Semântico:**
  - **Tecnologia:** RecursiveCharacterTextSplitter (LangChain).
  - **Configuração:** chunk_size=500 caracteres, chunk_overlap=50.
  - **Justificativa:** Divide os documentos em chunks coesos, preservando a coesão semântica e garantindo que o contexto não seja perdido abruptamente, melhorando a recuperação de informações relevantes.

2.2. Criação da Base Vetorial

- **Modelo de Embedding:** all-MiniLM-L6-v2 (sentence-transformers)
  - **Justificativa:** Equilíbrio ideal entre performance e qualidade em português, capturando efetivamente a semântica para busca por similaridade.
- **Mecanismo Vetorial:** ChromaDB
  - Simples, integrado nativamente com Python e eficiente para armazenamento de embeddings.
- **Armazenamento de Metadados:**
  Cada chunk vetorizado é salvo com metadados (title, category, source_file, chunk_index) para **filtragem e citação precisa** durante a busca.

2.3. Implementação do Pipeline RAG

O RAG gerencia o fluxo completo, da pergunta do usuário até a resposta final.

1. **Busca (Retrieval):**
   - A pergunta é transformada em embedding (all-MiniLM-L6-v2).
   - Busca por similaridade de cosseno na base vetorial, retornando os **top-5 chunks mais relevantes**.
2. **Geração Aumentada (Augmented Generation):**
   - Chunks recuperados são enviados para o LLMService.
   - **Modelo LLM:** gpt-3.5-turbo (OpenAI via LangChain).
   - **Engenharia de Prompt:**
     - Responder **apenas com base no contexto fornecido**.
     - **Citar fontes** no formato [Documento X] (título - categoria).
     - **Recusar respostas** caso a informação não esteja nos documentos.

        - **Justificativa:** Minimiza alucinações do LLM e garante fidelidade às fontes.

3. **Armazenamento e Monitoramento:**
    - Interações (pergunta, resposta, contexto, fontes, tempo) são salvas no RAGInteractionDB para análise futura.
    - O **Phoenix Orizom** foi implementado para **monitorar em tempo real as chamadas feitas pelo LangChain/OpenAI**, registrando logs, métricas de desempenho e eventuais erros.
      Isso facilita:

      Debugging: identificar rapidamente onde há falhas ou lentidão.

      Otimização de performance: acompanhar tempos de resposta e consumo de recursos.

      Auditoria: manter histórico detalhado das interações do sistema.

    Benefício Geral:
    Com o Phoenix Orizom, o projeto ganha **visibilidade e controle sobre a operação das APIs e do fluxo de RAG**, garantindo maior confiabilidade, rastreabilidade e possibilidade de ajustes finos em tempo real.

---

3. Estudo de Qualidade e Métricas

O RAGAS calcula métricas **quantitativas e qualitativas** para avaliar a qualidade do sistema.

3.1. Métricas Aplicadas

- **Métricas Automáticas (RAGAS):**
    - faithfulness (Fidelidade): Respostas baseadas no contexto fornecido.
    - answer_relevancy (Relevância): Pertinência da resposta à pergunta.
- **Métricas Avançadas (manuais):**
    - Recall@3: Verifica se documentos relevantes (similarity_score > 0.7) estão entre os 3 primeiros resultados.
    - Precisão Percebida: Normaliza feedback do usuário (1 a 5) para escala de 0 a 1.

Resultados Consolidados:

- Média de faithfulness: 0.708
- Média de answer_relevancy: 0.667
- Média de Recall@3: 0.077
- Média de Precisão Percebida: 0.65

---

4. Recomendações e Insights

4.1. Insights da Análise

- **Baixo Recall na Recuperação de Documentos:**
    - Recall@3 de 0.077indica que os documentos relevantes raramente aparecem entre os top-3.
    - Scores de similaridade baixos (0.0–0.3, até negativos) sugerem falhas na busca vetorial.
- **Respostas Tautológicas ou Incompletas:**
    - Pergunta sobre "due diligence" gerou resposta circular, indicando que o chunk recuperado continha apenas palavras-chave.
    - Pergunta sobre "limite de crédito" exigiu combinação de múltiplos chunks de baixa confiança.
- **Ruído nos Chunks:**
    - Frases como “Informação irrelevante:” indicam que o pré-processamento pode ser aprimorado para eliminar artefatos antes da vetorização.
- **Lacunas na Base de Conhecimento:**
    - Pergunta sobre "sistema de mentoria" identificou corretamente ausência de informação, evitando alucinação, mas aponta necessidade de atualização da base.

4.2. Plano de Ação e Melhorias (Prioritário)

1. Trocar o Modelo de Embedding (Impacto Máximo):

- Substituir all-MiniLM-L6-v2 por modelos mais poderosos: text-embedding-3-large/text-embedding-3-small (pagos) ou all-mpnet-base-v2 (gratuito).

2. Implementar Re-ranker:

- Recuperar **20 documentos candidatos** (bi-encoder).
- Avaliar com **Cross-Encoder** para pontuação semântica mais precisa.
- Reordenar top-20 por relevância semântica.

3. Multi-Agentes com LangGraph:

- Distribuir tarefas entre agentes especializados:
    - Agente de Recuperação por Título
    - Agentes especialistas por tema
- Aumenta eficiência, modularidade e precisão do RAG.

Tabela com perguntas x respostas x avaliações pode ser encontrada no rag_interactions.db

Diferenciais 1. Relatórios Gráficos

A imagem apresentada é uma projeção 3D do espaço vetorial dos embeddings, utilizando uma ferramenta como o TensorFlow Projector. Nesta visualização, cada ponto representa um chunk de texto da base de conhecimento, e cada cor corresponde a uma categoria de documento pré-definida (ex: "Política de Crédito", "Compliance", "Onboarding", etc.).