

Relatório Técnico do Desafio de Estágio de Verão 2026

Miguel Pereira Ramos

13 de novembro de 2025

Resumo

Este relatório detalha a arquitetura e as funcionalidades implementadas na solução para o Desafio de Estágio de Verão 2026 da NeuralMind. O projeto consiste em um sistema de chat com capacidade de responder a perguntas com base em conteúdo extraído de websites, utilizando uma arquitetura de Geração Aumentada por Recuperação (RAG).

1 Novas *features*

1.1 Backend

- **Scraping Híbrido (HTML/PDF) e Recursivo:** Foi implementada uma estratégia de scraping que suporta tanto páginas HTML quanto documentos PDF. O sistema pode navegar recursivamente por links dentro de um domínio para coletar dados de múltiplas fontes, ampliando a base de conhecimento. (*Nota: Para a demonstração pública, a funcionalidade de scraping está desativada para seguir as recomendações do enunciado.*)
- **Query Rewriting:** Implementação de uma feature de reescrita de consultas que melhora significativamente a qualidade das recuperações. O sistema reescreve perguntas ambíguas ou mal estruturadas em queries otimizadas para busca vetorial, aumentando a relevância dos chunks recuperados. Isso resulta em respostas mais precisas e contextualmente apropriadas, mitigando problemas de recuperação devido a formulações de perguntas subótimas. Esta feature elevou consideravelmente a taxa de sucesso do RAG ao lidar com variações linguísticas e consultas mal-formadas.
- **Endpoint para Chats Existentes:** Um novo endpoint foi criado para listar as conversas existentes de um usuário. Essa funcionalidade é a base para a implementação do histórico de chats no frontend.
- **Geração Automática de Títulos de Chat:** Para melhorar a usabilidade, o sistema agora gera um título automaticamente para cada nova conversa. O título é criado a partir da primeira pergunta do usuário, utilizando um modelo de linguagem para resumir o tópico.
- **Rota de Exclusão de Chats (DELETE /chat/{chat.id}):** Adicionada rota REST para apagar uma conversa e todas as suas mensagens. Implementada em `backend/app/routers/chat.py` (prefixo `/chat`, método `@router.delete("/chat_id")`), com validação de usuário via cookie JWT e delegação para o repositório (`backend/app/repositories/ai.py::delete_chat`). Em caso de sucesso retorna confirmação; se o chat não existir, responde 404. Um proxy no

Next.js expõe `DELETE /api/chat/[id]` e encaminha o cookie para o backend, simplificando o consumo no frontend.

1.2 Frontend

- **Barra Lateral com Histórico de Chats:** A principal novidade na interface do usuário é a adição de uma barra lateral que exibe todos os chats anteriores do usuário. Isso permite que o usuário navegue facilmente entre conversas passadas e continue de onde parou.
- **Botão para Apagar Conversas:** Na lista de conversas da barra lateral há um botão (ícone de lixeira) para excluir um chat. Ao clicar, o cliente chama a rota `DELETE /api/chat/[id]` (proxy Next.js), remove a conversa da UI e, se a conversa apagada estiver ativa, redireciona para a página inicial.

2 Arquitetura do RAG

O núcleo da inteligência da aplicação reside no backend, onde a arquitetura RAG foi implementada para fornecer respostas contextuais. O fluxo de dados é o seguinte:

2.1 Scraping e Extração de Conteúdo

O processo de ingestão de dados começa com o serviço de scraping, implementado em `backend/app/services/scraping.py`. Este módulo é responsável por:

- Fazer o download de conteúdo de URLs fornecidas.
- Suportar páginas HTML, documentos PDF e scraping recursivo.
- Limpar o HTML, removendo tags irrelevantes como `<script>`, `<style>`, `<header>`, e `<footer>` para focar no conteúdo principal.

2.2 Geração de Embeddings e Vector Store

Uma vez que o texto é extraído, ele passa pelo pipeline de embedding em `backend/app/services/embedding.py`:

- **Chunking:** O texto é dividido em pedaços menores (chunks) utilizando a classe `RecursiveCharacterTextSplitter` da biblioteca LangChain.
- **Embedding:** Cada chunk é transformado em um vetor numérico (embedding) usando o modelo `text-embedding-3-small` da OpenAI.
- **Vector Store:** Os embeddings e os textos correspondentes são armazenados em um banco de dados vetorial **ChromaDB**.

2.3 Recuperação e Geração de Resposta

Quando um usuário envia uma mensagem, o seguinte processo ocorre no endpoint de chat:

- **Reescrita de Query:** Antes da recuperação, a pergunta do usuário passa por um módulo de reescrita implementado em `backend/app/services/query_rewriter.py`. Este módulo utiliza um modelo de linguagem para reformular a query original, eliminando ambiguidades, preenchendo contexto implícito e otimizando a formulação para maximizar a relevância da busca vetorial. A reescrita transforma queries mal-estruturadas em consultas altamente semanticamente consistentes, melhorando drasticamente a qualidade das recuperações.
- **Recuperação:** A query reescrita é então usada para realizar uma busca por similaridade no ChromaDB, recuperando os chunks mais relevantes do corpus.
- **Aumento de Contexto:** Os chunks recuperados são inseridos no prompt que será enviado ao modelo de linguagem.
- **Geração:** O prompt final é enviado para a API da OpenAI, que gera a resposta.

3 Escolhas Técnicas na Arquitetura RAG

A implementação da pipeline de RAG foi baseada em um conjunto de tecnologias escolhidas por sua eficiência, popularidade na comunidade de IA e facilidade de integração.

3.1 Tecnologias Utilizadas

- **LangChain:** Este framework foi fundamental para orquestrar a pipeline. Sua principal contribuição foi a utilização da classe `RecursiveCharacterTextSplitter`, que permite uma "clusterização"(chunking) de texto inteligente. Em vez de simplesmente dividir o texto por um número fixo de caracteres, ele respeita separadores semânticos como parágrafos, sentenças e espaços, o que ajuda a manter o contexto dentro de cada chunk.
- **ChromaDB:** Como banco de dados vetorial, o ChromaDB foi escolhido por sua simplicidade e capacidade de rodar localmente com persistência em disco. Isso o torna ideal para desenvolvimento e prototipagem rápidos, eliminando a necessidade de configurar um serviço de banco de dados em nuvem para as fases iniciais do projeto. Sua integração nativa com o LangChain simplificou o processo de armazenamento e consulta de embeddings.
- **OpenAI text-embedding-3-small:** Para a geração dos vetores de embedding, este modelo da OpenAI foi selecionado por apresentar um ótimo equilíbrio entre custo, performance e qualidade. É um modelo recente, otimizado para tarefas de recuperação de informação (retrieval), sendo uma escolha sólida para aplicações RAG.

3.2 Motivações Arquiteturais

Nesta subseção detalhamos os porquês das escolhas e como elas se relacionam com requisitos funcionais e não-funcionais do sistema.

- **Arquitetura RAG:** A combinação de recuperação + geração permite mitigar *hallucinations* de modelos de linguagem ao ancorar a resposta em evidências externas [4]. Em vez de treinar ou ajustar um modelo proprietário com todo o corpus, utiliza-se um modelo generalista e complementa-se o contexto dinamicamente, reduzindo custo e tempo de iteração.
- **Scraping Recursivo e Multiformato:** A criação de uma base de conhecimento própria (em vez de depender de fontes públicas pré-indexadas) promove controle de atualização, curadoria e conformidade. O scraping recursivo maximiza cobertura de conteúdo dentro de um domínio, enquanto suporte a HTML e PDF traz heterogeneidade informacional presente em documentação técnica. Optou-se por limpeza estrutural (remoção de scripts, estilos e elementos de navegação) para reduzir ruído sem perder semântica. Seguir boas práticas de ética em coleta (respeito a `robots.txt`, limitação de taxa e uso justo) minimiza riscos legais e de carga excessiva [5].
- **Chunking Semântico:** A divisão em chunks preservando limites lógicos (parágrafos / sentenças) aumenta a precisão de recuperação. Chunks muito grandes diluem relevância; muito pequenos fragmentam contexto e pioram coerência na resposta. A estratégia recursiva da LangChain equilibra esses fatores empiricamente.
- **Armazenamento Vetorial Local (ChromaDB):** Um *vector store* possibilita consultas por similaridade usando métricas em espaço de embeddings (ex.: coseno). Escolheu-se Chroma pela persistência simples em disco, inicialização imediata e integração com LangChain, acelerando prototipagem. Para escala maior, alternativas como FAISS [6] ou estruturas HNSW [7] oferecem desempenho superior em milhões de vetores, mas exigiriam configuração adicional. A escolha atual atende volume moderado esperado (centenas a poucos milhares de documentos) mantendo baixo acoplamento.
- **Modelo de Embedding:** O `text-embedding-3-small` foi adotado por custo/latência reduzidos e boa performance geral em tarefas de similaridade. Modelos abertos como Sentence-BERT [8] poderiam ser usados para reduzir dependência externa, porém implicariam gerenciar infraestrutura de inferência. A opção por serviço gerenciado acelerou desenvolvimento inicial.
- **Query Rewriting:** A incorporação de reescrita de queries no pipeline RAG foi uma decisão fundamental para aumentar a qualidade das recuperações. Sem este módulo, queries ambíguas ou formuladas de forma subótima resultariam em recuperações pobres, degradando significativamente a qualidade das respostas finais. A reescrita permite ao sistema transformar perguntas naturais e frequentemente imprecisas em consultas otimizadas para busca vetorial. Isso melhora drasticamente o recall dos chunks relevantes e, consequentemente, a qualidade contextual das respostas geradas. A implementação utiliza um modelo de linguagem (GPT) com instruções cuidadosamente ajustadas para garantir que a reescrita preserve a intenção da pergunta original enquanto a otimiza para recuperação de informação.
- **Busca por Similaridade Aproximada:** Embora o volume atual permita busca exata (varredura linear), manter a arquitetura compatível com *Approximate Nearest Neighbors* (ANN) abre caminho para escalar sem reformulações profundas. Estruturas como gráficos navegáveis (HNSW) ou índices IVF/Flat em FAISS oferecem trade-offs conhecidos entre recall, latência e memória [9].

- **Separação de Serviços:** O scraper, o pipeline de embedding e os endpoints de chat são módulos distintos para permitir substituição (ex.: trocar modelo de embedding ou estratégia de scraping) sem impactar o restante do sistema. Essa coesão modular facilita testes independentes e futuras extensões (ex.: classificação de qualidade dos chunks antes da indexação).

3.3 Riscos e Mitigações

- **Qualidade do Corpus:** Conteúdo ruidoso ou desatualizado pode degradar respostas. Mitigação: filtros e futura anotação manual de relevância.
- **Escalabilidade:** Crescimento rápido do número de documentos pode exigir migração para índices ANN especializados. Mitigação: abstração da camada de *vector store* permite troca por FAISS ou serviços gerenciados.
- **Dependência de API Externa:** Uso de embeddings OpenAI implica custo e latência de rede. Mitigação: planejamento para suporte alternativo via modelos locais tipo Sentence-BERT.
- **Conformidade Legal no Scraping:** Mudanças em políticas de uso de sites podem exigir ajustes. Mitigação: monitoramento de `robots.txt` e cabeçalhos de resposta.

4 Problemas Encontrados e Correções

Esta seção descreve os principais problemas práticos encontrados durante a implantação e uso do sistema (principalmente em ambiente *cloud*) e as respectivas correções aplicadas.

4.1 Erro no armazenamento do cookie JWT

Sintoma: após login via OAuth com GitHub, o cookie de sessão JWT não era salvo no navegador, impedindo autenticação em chamadas subsequentes.

Causa: não identificado.

Correção: deploy no Railway, substituindo o da Vercel e criação de um intermediário no servidor Next.js. Após migração, o cookie passou a ser salvo corretamente.

4.2 POST /chat retornando 401 (Unauthorized)

Sintoma: chamadas GET autenticadas funcionavam (cookie presente), mas o POST /chat retornava 401.

Causa: o cookie de sessão estava salvo no domínio do **frontend**, enquanto o backend exigia o cookie nas requisições ao seu domínio. Em chamadas *cross-origin*, especialmente em POST com **SameSite**, o navegador pode não enviar o cookie como esperado.

Correção: foi criado um proxy no Next.js em `frontend/src/app/api/chat/route.ts` que:

- Recebe o POST do browser (mesma origem do frontend) e lê o cookie `access_token`.
- Encaminha a requisição ao backend (`/chat`) incluindo explicitamente o cabeçalho `Cookie: access_token=...` e faz o pipe do `stream` de volta ao cliente.
- O cliente de chat (`frontend/src/lib/ai-sdk.ts`) passou a usar `/api/chat` como endpoint, eliminando problemas de CORS/SameSite nesse fluxo.

5 Trabalhos Futuros e Melhorias

Apesar da robustez da solução atual, há diversas oportunidades para aprimoramento e expansão das funcionalidades. Esta seção descreve os próximos passos pensados para evoluir o projeto.

- **Expansão da Cobertura de Testes:**

- **Testes Unitários e de Integração:** Aumentar a cobertura de testes unitários no backend e frontend para garantir a estabilidade de componentes individuais e suas interações, prevenindo regressões.
- **Avaliação Automatizada com "LLM-as-Judge":** Implementar um pipeline de avaliação automatizada onde um modelo de linguagem avançado (como GPT-5, Gemini 2.5 Pro) atua como "juiz" para avaliar a qualidade das respostas do sistema RAG. Utilizando métricas como *faithfulness* (fidelidade à fonte), *answer relevancy* (relevância da resposta) e *context recall* (recuperação de contexto), é possível criar um framework de testes que valide a assertividade do sistema de forma escalável, similar a ferramentas como RAGAS [10].

- **Aprimoramento da Experiência do Usuário (UX/UI):**

- **Navegação e Gerenciamento de Conversas:** Refinar a interface para tornar a navegação entre chats mais fluida e o gerenciamento de conversas menos truncado. Isso pode incluir animações de transição suaves, atualizações de estado mais reativas e uma interface de gerenciamento que permita fixar, arquivar ou agrupar conversas.
- **Visualização de Fontes:** Implementar uma funcionalidade que mostre quais trechos dos documentos originais foram usados para gerar a resposta, aumentando a transparência e a confiança do usuário no sistema.

- **Suporte a Multimodalidade:**

- **Upload de Arquivos:** Permitir que os usuários façam upload de arquivos (imagens, áudios, PDFs) para que o sistema possa extrair informações e responder perguntas sobre eles. Isso exigiria a integração de modelos de processamento de imagem (OCR, image-to-text) e de áudio (speech-to-text).

- **Outras Features Relevantes:**

- **Otimização de Parâmetros de Embedding:** Realizar testes sistemáticos para otimizar os parâmetros do processo de embedding, como o tamanho dos chunks (número de palavras ou caracteres), a sobreposição entre eles (*overlap*) e o número de documentos recuperados para compor o contexto. Ajustes finos nesses parâmetros podem impactar significativamente a relevância e a coerência das respostas.
- **Mecanismo de Feedback:** Adicionar botões de ”gostei”/”não gostei” (like/dislike) nas respostas geradas. O feedback coletado pode ser usado para refinar o modelo de recuperação ou de geração.
- **Cache Inteligente:** Implementar um sistema de cache para perguntas frequentes, reduzindo a latência e os custos com chamadas à API da OpenAI.
- **Fine-tuning de Modelos:** Realizar o fine-tuning do modelo de embedding com dados específicos do domínio de interesse para melhorar a qualidade da recuperação de documentos.

Referências

- [1] LangChain Documentation. https://python.langchain.com/docs/get_started/introduction
- [2] ChromaDB Documentation. <https://docs.trychroma.com/>
- [3] OpenAI API Documentation. <https://platform.openai.com/docs/introduction>
- [4] Patrick Lewis et al. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. NeurIPS 2020.
- [5] S. Munzert et al. Automated Data Collection with R: A Practical Guide to Web Scraping and Text Mining. Wiley, 2014.
- [6] Jeff Johnson, Matthijs Douze, Hervé Jégou. Billion-scale similarity search with GPUs. IEEE Transactions on Big Data, 2019.
- [7] Yu A. Malkov, D. A. Yashunin. Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. IEEE TPAMI, 2020.
- [8] N. Reimers, I. Gurevych. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. EMNLP 2019.
- [9] J. Aumüller, E. Bernhardsson, A. Faithfull. ANN-Benchmarks: A Benchmarking Tool for Approximate Nearest Neighbor Algorithms. SISAP 2018.
- [10] Shahul Es et al. RAGAS: A RAG Assessment System. arXiv preprint arXiv:2309.15217, 2023.