Revista Eletrônica de Iniciação Científica em Computação, 2026, 24:1
ISSN: 3085-8461 • doi: 10.5753/reic.2026.6742

Artigo de Pesquisa/Research Paper

KBchat: Uma abordagem para consulta inteligente de bases de conhecimento

KBchat: An approach to intelligent knowledge base querying

Davi Santos Tavares [Instituto Federal de Educação, Ciência e Tecnologia do Ceará | *davi.tavares07@aluno.ifce.edu.br*]
Robson Gonçalves Fechine Feitosa [Instituto Federal de Educação, Ciência e Tecnologia do Ceará |*robsonfeitosa@ifce.edu.br*]

Instituto Federal de Educação, Ciência e Tecnologia do Estado do Ceará - campus Crato, CE-292, SN - Gisélia Pinheiro, Crato - CE, 63115-500, Brasil.

Resumo. Com o avanço das tecnologias de linguagem natural, este trabalho propõe o desenvolvimento de um sistema de chat com base na abordagem de Geração Aumentada por Recuperação (RAG), voltado à consulta em bases de conhecimento criadas a partir de documentos em PDF. O sistema opera com dois fluxos principais. O primeiro fluxo é responsável pela inserção de documentos na base de conhecimento: os arquivos PDF são processados com a biblioteca Markitdown, convertidos para markdown, segmentados em trechos e transformados em *embeddings* utilizando o modelo all-mpnet-base-v2. Esses vetores são então armazenados em um banco vetorial PostgreSQL com a extensão pgvector. O segundo fluxo representa o chat em que o usuário interage com a base: ao receber uma pergunta, o sistema gera seu *embedding*, passa essa pergunta por um pré-processamento, realiza uma busca vetorial pelos trechos mais relevantes e constrói um *prompt* contextualizado, enviado ao modelo Gemini-2.0-Flash-001, que gera a resposta. A aplicação foi desenvolvida com Python e Flask para o *backend* e React no *frontend*, integrando técnicas contemporâneas de NLP e IA generativa. Os resultados demonstram a viabilidade do sistema para consultas eficientes e contextualizadas em grandes volumes de texto, com potencial aplicação em contextos corporativos, educacionais, jurídicos, entre outros.

Abstract. With the advancement of natural language processing technologies, this work proposes the development of a chat system based on the Retrieval-Augmented Generation (RAG) approach, designed for querying knowledge bases created from PDF documents. The system operates with two main workflows. The first workflow is responsible for ingesting documents into the knowledge base: PDF files are processed using the Markitdown library, converted to markdown, segmented into passages, and transformed into embeddings using the all-mpnet-base-v2 model. These vectors are then stored in a PostgreSQL vector database with the pgvector extension. The second workflow represents the chat interaction, where the user queries the knowledge base: upon receiving a question, the system generates its embedding, applies preprocessing, performs a vector search for the most relevant passages, and constructs a contextualized prompt, which is sent to the Gemini-2.0-Flash-001 model to generate the response. The application was developed using Python and Flask for the backend, and React for the frontend, integrating contemporary techniques in NLP and generative AI. The results demonstrate the feasibility of the system for efficient and contextualized queries on large volumes of text, with potential applications in corporate, educational, legal, and other domains.

Palavras-chave: RAG, LLM, Embeddings, Chatbot, Base de conhecimento

Keywords: RAG, LLM, Embeddings, Chatbot, Knowledge Base

Recebido/Received: 23 October 2025 • **Aceito/Accepted:** 10 July 2026 • **Publicado/Published:** 07 August 2026

1 Introdução

Nas últimas décadas, o avanço dos *Transformers*, uma arquitetura de rede neural proposta por Vaswani *et al*. [2017], revolucionou o cenário da Inteligência Artificial. Esse avanço possibilitou o desenvolvimento de modelos de linguagem de grande porte (*Large Language Models*, da sigla LLM), como o GPT-3 [Brown *et al*., 2020], capazes de compreender e gerar textos com alto nível de coerência e versatilidade. Para superar as limitações dos LLMs em acessar informações atualizadas ou específicas, surgiu a abordagem RAG, que integra a recuperação de dados externos ao processo de geração de respostas, permitindo que os modelos se baseiem em fontes de conhecimento relevantes [Lewis *et al*., 2020].

Embora existam soluções amplamente difundidas para consulta inteligente de documentos baseadas em RAG, como NotebookLM[1] e Claude[2], essas plataformas são disponibilizadas como serviços centralizados e geralmente requerem o envio dos documentos para infraestruturas mantidas por terceiros. Em diversos contextos, especialmente em instituições públicas, organizações privadas, ambientes acadêmicos e setores que lidam com informações sensíveis, essa característica pode representar uma limitação relacionada à privacidade, confidencialidade ou governança dos dados. Além disso, tais ferramentas oferecem pouca flexibilidade para customização dos componentes internos do *pipeline* RAG, restringindo a substituição de modelos de linguagem, modelos de *embeddings*, mecanismos de recuperação ou estratégias de reranqueamento.

[1]Disponível em: https://notebooklm.google.com/. Acesso em: 08 de junho de 2026.
[2]Disponível em: https://claude.com/. Acesso em: 08 de junho de 2026.

Nesse contexto, o presente trabalho tem como principal objetivo apresentar o desenvolvimento de um sistema, denominado KBchat[3] (*Knowledge Base Chat*), que visa facilitar o acesso de usuários não especializados à tecnologia RAG; e, mais especificamente, permitir a criação de bases de conhecimento personalizadas a partir de documentos PDF, e a interação com essas bases por meio de um chat inteligente. Para isso, são utilizadas tecnologias como o modelo de *embeddings* all-mpnet-base-v2 [Song *et al*., 2020] para a conversão dos conteúdos em vetores, o banco de dados PostgreSQL com a extensão [pgvector, 2025] para o armazenamento vetorial, e o modelo Gemini 2.0 Flash [Google, 2025] para a geração de respostas, orquestrados através de um *backend* desenvolvido em Python [Flask Project, 2025] e um processo de pré-processamento de documentos utilizando a biblioteca Markitdown da Microsoft [Microsoft, 2025].

O KBchat foi concebido como uma alternativa que permite a construção de bases de conhecimento personalizadas a partir de documentos PDF, oferecendo maior controle sobre o processamento e armazenamento das informações. Seu público-alvo inclui estudantes, pesquisadores, professores e profissionais que necessitam consultar coleções documentais próprias por meio de linguagem natural, sem demandar conhecimentos avançados sobre bancos vetoriais, modelos de linguagem ou *frameworks* de Inteligência Artificial. Diferentemente de soluções proprietárias, a arquitetura do KBchat possibilita a substituição e adaptação de seus principais componentes, permitindo adequação a diferentes requisitos de infraestrutura, custos operacionais e domínios de aplicação. Dessa forma, a proposta busca reduzir barreiras técnicas para adoção de sistemas RAG, ao mesmo tempo em que oferece uma alternativa flexível para cenários nos quais o controle sobre os documentos processados constitui um requisito relevante.

Conforme ilustrado ao longo deste trabalho, observa-se que a integração eficiente entre as técnicas de *embeddings*, recuperação vetorial e geração assistida por contexto permite a construção de um sistema capaz de oferecer respostas precisas e relevantes, viabilizando o uso da tecnologia RAG mesmo em ambientes com recursos limitados. Diante do exposto, este trabalho está estruturado da seguinte forma: na Seção 2, são apresentados os Trabalhos Relacionados. A Seção 3 detalha a metodologia aplicada no desenvolvimento do sistema. A Seção 4 apresenta o desenvolvimento e os resultados obtidos; e, a Seção 5 ilustra a conclusão.

2 Trabalhos Relacionados

A abordagem RAG tem sido amplamente explorada na literatura científica como uma abordagem para mitigar limitações dos modelos de LLMs, especialmente em tarefas que demandam acesso a informações específicas e atualizadas. O trabalho seminal de Lewis *et al*. [2020] estabelece formalmente o paradigma RAG, ao integrar modelos generativos a mecanismos de recuperação de documentos, demonstrando ganhos substanciais em tarefas de *question answering* e na geração de texto baseada em conhecimento externo.

Avanços subsequentes enfatizam a importância da etapa de recuperação no desempenho global do sistema. Abordagens como *Dense Passage Retrieval* (DPR) [Karpukhin *et al*., 2020] sugerem que a qualidade da recuperação influencia diretamente a geração, enquanto modelos como ColBERT [Khattab and Zaharia, 2020] exploram estratégias de recuperação mais refinadas baseadas em interação tardia para melhorar a relevância dos documentos recuperados.

No contexto de avaliação de sistemas RAG, trabalhos recentes têm destacado a necessidade de métricas específicas para capturar não apenas a similaridade textual, mas também a fidelidade e a relevância das respostas. Nesse sentido, Es *et al*. [2024] propõem o framework RAGAS, que introduz métricas como *faithfulness*, *answer relevancy* e *context precision*, permitindo uma avaliação mais alinhada ao comportamento esperado desses sistemas.

Por outro lado, Aquino *et al*. [2024] propuseram um fluxo experimental de RAG voltado para a extração de informações de documentos jurídicos brasileiros, especialmente relacionados a processos licitatórios. O estudo avaliou diferentes parâmetros de *chunking*, *embeddings* e recuperação, alcançando uma acurácia média de 90% na extração de variáveis específicas, demonstrando a eficácia da abordagem em cenários de alta complexidade semântica.

De forma semelhante, Kuratomi *et al*. [2024] desenvolveram um assistente virtual institucional baseado em RAG para consultas a documentos normativos da Universidade de São Paulo. Os resultados indicaram que a qualidade da recuperação impacta diretamente a geração de respostas, com ganhos superiores a 30 pontos percentuais quando os trechos relevantes são corretamente recuperados.

Apesar dos avanços apresentados na literatura, observa-se que muitas soluções ainda demandam infraestrutura computacional significativa ou são direcionadas a domínios específicos, o que limita sua adoção em cenários mais amplos. Nesse contexto, o presente trabalho diferencia-se ao propor uma abordagem orientada à democratização do acesso a sistemas RAG, com foco na construção de bases de conhecimento a partir de documentos PDF e na interação por meio de linguagem natural.

Além disso, diferentemente de abordagens que focam exclusivamente na recuperação ou na geração, a solução proposta integra técnicas de refinamento de consulta (RQ-RAG), recuperação vetorial e reranqueamento com BM25, buscando equilibrar qualidade de resposta e viabilidade computacional. Essa combinação permite a construção de um sistema genérico, de fácil utilização e aplicável a diferentes domínios, mantendo desempenho competitivo mesmo em ambientes com recursos limitados.

A Tabela 1 apresenta uma comparação entre os principais trabalhos relacionados e a abordagem proposta, destacando diferenças quanto ao domínio de aplicação, técnicas empregadas e objetivos do sistema.

[3]Disponível em: https://github.com/davi222-santos/KBchat. Acesso em: 08 de junho de 2026.

Tabela 1. Comparação entre abordagens relevantes para sistemas RAG

Trabalho	**Foco**	**Tipo de Recuperação**	**Reranking**	**Avaliação**	**Aplicação**
Lewis *et al*. [2020]	Fundamentos de RAG	Dense Retrieval	Não	QA benchmarks	Modelo genérico
Karpukhin *et al*. [2020]	Recuperação (DPR)	Dense Retrieval (BERT)	Não	Top-k accuracy	Open-domain QA
Khattab and Zaharia [2020]	Recuperação avançada	Late interaction (ColBERT)	Parcial	Ranking metrics	Busca semântica
Es *et al*. [2024]	Avaliação de RAG	Dependente do sistema	Não	Faithfulness, relevância, precisão	Framework de avaliação
Aquino *et al*. [2024]	Aplicação em domínio jurídico	Vetorial (Chroma)	Não	Acurácia (90%)	Extração de dados legais
Kuratomi *et al*. [2024]	Assistente institucional	Vetorial (FAISS)	Sim	Acurácia variável (até 54%)	Consulta normativa
Este trabalho	Sistema RAG acessível	Vetorial (PGvector + *embeddings*)	Sim (BM25)	RAGAS + avaliação experimental	Chat com base em PDFs

3 Metodologia

Inicialmente, foi realizada uma revisão da literatura, com levantamento de artigos e trabalhos acadêmicos que descreveram a construção de sistemas baseados na abordagem RAG. Em seguida, foi realizada uma análise comparativa para seleção da LLM a ser utilizada. Foram realizados testes com três modelos: DeepSeek, Mistral Small e Gemini 2.0 Flash. O objetivo principal dessa etapa foi identificar o modelo que apresentasse melhor desempenho em termos de velocidade de resposta, capacidade de lidar com *prompts* extensos e viabilidade de utilização em ambientes com recursos computacionais limitados. A Figura 1, a seguir, ilustra o fluxograma das etapas da metodologia utilizada. Já a Tabela 2, a seguir, resume os resultados obtidos na avaliação dos modelos.

Na primeira fase, foi testado o modelo DeepSeek-r1:8b, operando localmente em um computador desktop equipado com AMD Ryzen 7 5700U com Radeon Graphics e 20GB de memória RAM DDR4. O modelo utilizado apresentou limitações significativas de hardware, resultando em tempos de resposta de até 2 minutos para perguntas simples. Além disso, a necessidade de manter o modelo armazenado e em execução na máquina local representava um desafio adicional de infraestrutura. Em seguida, foi avaliado o Mistral Small, que possui suporte a entradas de até 128K tokens. Durante os testes, foram observados problemas como timeout em execuções e respostas incompletas em *prompts* mais complexos. Em um teste padrão que solicitava a geração de um texto de 500 palavras, o Mistral não conseguiu atender integralmente à solicitação, gerando um conteúdo consideravelmente abaixo do esperado.

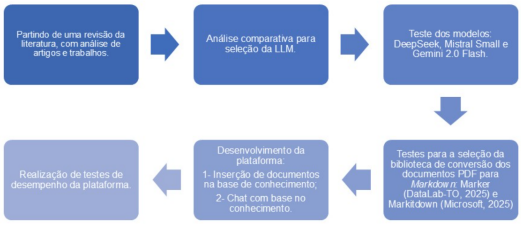

Figura 1. Fluxograma das etapas metodológicas para seleção de LLM, processamento de documentos e desenvolvimento da plataforma de chat.

Por fim, foi avaliado o modelo Gemini 2.0 Flash, que possui uso gratuito de 15 requisições por minuto (RPM), 1500 requisições por dia (RPD), 1.000.000 tokens por minuto (TPM), além de limites de entrada de 1.048.576 tokens e de saída de 8.192 tokens. Nos testes realizados com o mesmo *prompt* utilizado nos demais modelos, o Gemini obteve uma média de resposta em 15 segundos, demonstrando alta eficiência e estabilidade. O Gemini 2.0 Flash foi selecionado para ser utilizado no sistema, devido sua velocidade superior nos testes realizados, estabilidade nas respostas, suporte robusto a tokens e, principalmente, a vantagem de operação via API, eliminando a necessidade de hospedar um modelo de grande porte na máquina local, facilitando a manutenção e reduzindo os requisitos de hardware.

Após a definição da LLM a ser utilizada, foram realizados testes para a seleção da biblioteca responsável pela conversão dos documentos PDF para o formato Markdown, etapa fundamental na preparação dos dados textuais que compõem a base de conhecimento. Foram avaliadas duas bibliotecas: Marker [Datalab-to, 2025] e Markitdown [Microsoft, 2025]. ambas desenvolvidas para extrair a estrutura do texto de documentos e convertê-los em formato Markdown, formato mais adequado para processamento por modelos de linguagem.

A biblioteca Markitdown demonstrou melhor desempenho em termos de tempo de execução e qualidade da estrutura gerada. Entretanto, para garantir maior eficiência no processo, é necessário que o documento de entrada esteja no formato DOCX. Por esse motivo, foi incluída uma etapa adicional no fluxo de preparação dos dados, na qual os arquivos PDF recebidos são convertidos para DOCX utilizando a biblioteca pdf2docx, desenvolvida em Python. Essa conversão ocorre de forma automatizada antes da aplicação do Markitdown.

A Tabela 3 apresenta uma síntese dos resultados obtidos na avaliação das bibliotecas. Nos testes realizados, observou-se que a biblioteca Marker, embora opere diretamente com arquivos PDF, apresenta limitações quanto à preservação da estrutura do documento original. As tabelas são convertidas em blocos de texto corrido, comprometendo a legibilidade e a extração de significado pelo modelo. Além disso, a divisão em seções e subtópicos nem sempre é respeitada, o que dificulta

Tabela 2. Avaliação dos Modelos de LLM Testados

Modelo	Características Principais	Desempenho Observado
DeepSeek-r1:8b	Execução local; requer hardware robusto (GPU com alta VRAM).	Respostas lentas (até 2 minutos); limitações de hardware para uso eficiente.
Mistral Small	Suporte a entradas de até 128K tokens; operação via API.	Timeout em *prompts* complexos; respostas incompletas em textos longos.
Gemini 2.0 Flash	15 RPM, 1500 RPD, 1.000.000 TPM; entrada até 1M tokens; saída até 8K tokens; operação via API.	Respostas rápidas (~15 segundos); alta estabilidade; suporte robusto a grandes *prompts*.

Tabela 3. Comparativo entre bibliotecas de conversão de PDF para Markdown

Biblioteca	Tempo médio (PDF com 10 páginas)	Extração de tabelas	Estrutura de tópicos	Formato ideal de entrada
Marker	30 segundos	Fraca (texto corrido)	Parcial	PDF
Markitdown	3 segundos (incluindo conversão para DOCX)	Alta (mantém estrutura)	Excelente	DOCX

a segmentação adequada do conteúdo.

Em contrapartida, a biblioteca Markitdown apresentou maior fidelidade estrutural, com destaque para a preservação de tabelas e a correta organização hierárquica dos títulos e subtítulos. Essa característica é especialmente relevante para documentos que apresentam informações dispostas lado a lado, como quadros comparativos ou colunas paralelas, que foram interpretadas de forma mais eficaz. O tempo médio de processamento também foi significativamente inferior, totalizando cerca de dois segundos para a conversão completa de um PDF com dez páginas (considerando a conversão prévia para DOCX), frente aos trinta segundos observados com o uso do Marker.

Dessa forma, optou-se pelo uso da biblioteca Markitdown no sistema desenvolvido, assegurando maior eficiência na etapa de preparação dos documentos e melhor qualidade dos dados utilizados no processo de RAG.

Com os testes das ferramentas finalizados, iniciou-se o desenvolvimento da plataforma, que foi estruturado em dois fluxos principais: 1) Inserção de documentos na base de conhecimento; e 2) Chat com a base de conhecimento. Para o fluxo de inserção de documentos, foi inicialmente implementado o módulo de upload de arquivos PDF na interface web utilizando React, com integração via API REST desenvolvida em Python utilizando o framework Flask. Em seguida, foi implementado o processamento dos arquivos PDF no *backend*, utilizando a biblioteca Markitdown para converter os documentos para o formato markdown.

Posteriormente, desenvolveu-se a etapa de divisão do conteúdo em trechos (*chunks*) com a utilização da biblioteca MarkdownHeaderTextSplitter, integrada ao framework Langchain, permitindo a segmentação dos documentos de acordo com a hierarquia de títulos (tags #, ## e ###). Cada trecho segmentado com 2000 caracteres foi então processado para geração dos *embeddings*, utilizando o modelo all-mpnet-base-v2 disponibilizado pelo HuggingFace. Finalmente, foi realizada a integração com o banco de dados PostgreSQL, utilizando a extensão pgvector para armazenamento dos vetores gerados, juntamente com metadados como nome do documento, conteúdo de cada trecho e identificação da base de conhecimento.

Para o fluxo de chat com a base de conhecimento, foi implementada a funcionalidade de envio de queries pela interface web, conectando-se ao back-end via API. Inicialmente, a *query* (consulta) do usuário passa por técnicas de engenharia de *prompts*, como o uso de *guardrails*, que estabelecem limites e instruções para orientar a geração de respostas de maneira segura e controlada, e o RQ-RAG (*Refined Query for Retrieval-Augmented Generation*), que refina a consulta original com base no contexto, tornando-a mais clara, específica e eficaz para o processo de recuperação de trechos relevantes via RAG [Chan *et al*., 2024]. Após isso, a *query* do usuário passa para geração de *embeddings* com o mesmo modelo all-mpnet-base-v2, garantindo compatibilidade com a base vetorial armazenada. Foi utilizada a consulta de similaridade no banco de dados PostgreSQL, por meio da busca vetorial para recuperação dos trechos semanticamente mais próximos da pergunta do usuário. Em seguida, foi implementada a montagem automática do *prompt*, combinando a *query* com os trechos recuperados. O *prompt* final foi enviado ao modelo Gemini 2.0 Flash via integração com o framework Langchain, com temperatura 0 (zero) e a resposta gerada foi disponibilizada na interface do usuário em tempo real.

4 Desenvolvimento

O desenvolvimento do sistema seguiu um fluxo estruturado com as seguintes etapas: a definição do escopo, que incluiu a delimitação das funcionalidades e a construção da arquitetura; o desenvolvimento do *backend*, que compreendeu a configuração do ambiente, o fluxo de inserção de documentos e o fluxo de interação no chat; e, por fim, a construção do *frontend*, que abrangeu a parte visual e a experiência do usuário ao interagir com o sistema, conforme detalhado nas próximas subseções.

4.1 Escopo

O escopo do sistema foi definido de forma a atender aos requisitos funcionais do projeto, estabelecendo claramente as funcionalidades e os limites de atuação da solução. Entre os pontos contemplados, destacam-se a implementação de um *backend* responsável pela gestão dos fluxos de dados e pela interação entre documentos e o chat; e, o desenvolvimento de um *frontend* intuitivo, que prioriza a usabilidade e a clareza

na apresentação das informações ao usuário. Além disso, foram considerados aspectos de escalabilidade, segurança e integração, garantindo que o sistema pudesse ser expandido e adaptado a diferentes contextos de utilização, sem perder eficiência nem comprometer a experiência do usuário.

4.1.1 Delimitação das funcionalidades

O sistema desenvolvido tem como escopo principal permitir a criação de bases de conhecimento a partir de documentos em formato PDF e a posterior consulta a essas bases por meio de um chat inteligente com IA. Os documentos aceitos devem estar em formato exclusivamente textual, sem a presença de imagens digitalizadas, pois o sistema não conta com recursos de OCR (*Optical Character Recognition*). As funcionalidades foram delimitadas de modo que o sistema armazenasse e processasse apenas arquivos PDF contendo texto estruturado, convertendo-os em representações vetoriais para posterior recuperação.

Durante a interação via chat, as respostas são geradas unicamente com base no conteúdo previamente inserido na base de conhecimento pelo próprio usuário. Não há qualquer integração com mecanismos de busca externos ou acesso à internet, uma vez que todo o contexto utilizado para a geração das respostas é restrito ao material armazenado localmente no banco vetorial.

A interface do sistema foi projetada com foco na simplicidade e usabilidade. O *frontend* possui apenas duas abas principais, uma destinada à inserção de documentos, onde o usuário pode enviar novos arquivos e vinculá-los a uma base específica, e outra destinada ao chat, na qual o usuário seleciona a base de conhecimento desejada e interage com ela em linguagem natural. A proposta da interface é oferecer uma experiência direta e objetiva, adequada tanto para usuários técnicos quanto não técnicos.

4.1.2 Arquitetura

A Figura 2 ilustra o diagrama de arquitetura do sistema geral do sistema, que contempla os dois principais fluxos envolvidos: o processo de inserção de documentos na base de conhecimento; e, o processo de interação via chat com recuperação e geração de respostas baseadas nesses documentos.

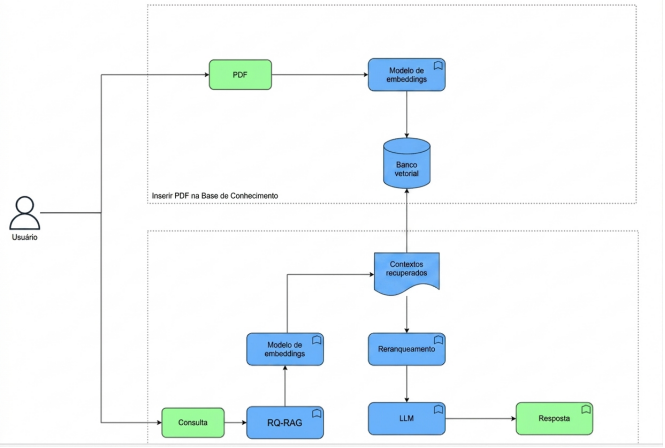

Figura 2. Diagrama da arquitetura do sistema detalhando os processos de ingestão de documentos (processamento de PDF e armazenamento vetorial) e consulta com geração baseada em recuperação (RAG e re-ranqueamento).

4.2 Backend

O *backend* do sistema foi desenvolvido em Python, utilizando o framework Flask para a criação da API responsável por orquestrar os fluxos de inserção e consulta na base de conhecimento. A estrutura do código segue uma organização modular, dividida entre rotas, serviços e utilitários, de modo a facilitar a manutenção e a escalabilidade do sistema.

As rotas estão organizadas em três arquivos principais: chat.py, document.py e kb.py. O arquivo chat.py gerencia a rota responsável por processar as perguntas enviadas pelos usuários, acionando os módulos de recuperação de contexto e geração de respostas. Já o document.py concentra as rotas relacionadas ao envio e exclusão de documentos PDF, atuando como intermediário entre a entrada do usuário e o *pipeline* de inserção na base vetorial. Por fim, o kb.py lida com a criação, listagem e remoção de bases de conhecimento, permitindo ao usuário gerenciar suas próprias coleções de documentos.

A inicialização da aplicação ocorre por meio do arquivo app.py, que instancia o objeto Flask, registra os blueprints das rotas e executa o servidor. Essa estrutura permite que cada módulo atue de forma independente, mas coordenada, garantindo uma separação clara entre o tratamento das requisições e a lógica de negócio.

As próximas subseções apresentam em detalhes os dois principais fluxos que compõem o funcionamento do *backend*: o processo de inserção de documentos na base de conhecimento e a interação por meio do chat, nos quais os arquivos de serviço (*services/*) assumem papel central na execução das tarefas.

4.2.1 Fluxo de inserção de documentos

O processo de inserção de documentos no *backend* é responsável por receber arquivos PDF enviados pelo usuário, convertê-los para um formato textual adequado e armazenar seus conteúdos na base de dados vetorial para posterior recuperação semântica.

Inicialmente, o arquivo PDF é convertido em um arquivo DOCX temporário por meio da biblioteca pdf2docx. Essa conversão é necessária porque a ferramenta MarkItDown, utilizada para transformar o documento em texto Markdown, apresenta melhor eficiência e fidelidade na extração quando o arquivo está no formato DOCX, além de não suportar a extração de imagens diretamente. Para contornar a presença de imagens no PDF, a função de extração de imagens do pdf2docx é desativada, garantindo que apenas o conteúdo textual seja processado. Segue o trecho da conversão de DOCX para Markdown utilizando a biblioteca Markitdown.

```
1 md_converter = MarkItDown(enable_plugins=
      False)
2 result = md_converter.convert_stream(
      docx_io, filename_hint="file.docx")
```

Código 1: Converterndo texto em Markdown

Após a conversão para DOCX, o arquivo é enviado ao módulo MarkItDown, que realiza a conversão para Markdown, preservando a estrutura e a formatação básica do texto. O resultado é um conteúdo textual limpo e estruturado, pronto para o processamento seguinte.

Com o texto Markdown em mãos, o módulo pgvector.py gerencia a segmentação do conteúdo em fragmentos

(*chunks*) e a geração dos *embeddings*, que são vetores numéricos representando o significado semântico de cada trecho. Esses *embeddings* são obtidos a partir do modelo sentence-transformers/all-mpnet-base-v2 e armazenados em uma tabela do banco de dados PostgreSQL configurada com a extensão pgvector, que permite a indexação e busca eficiente por similaridade vetorial.

Antes de inserir os dados, o sistema verifica a existência prévia do documento para evitar duplicidade. Cada fragmento é então salvo no banco, associado ao usuário, à base de conhecimento e ao nome do documento. O índice vetorial aplicado à tabela possibilita consultas rápidas e precisas baseadas no conteúdo semântico dos textos.

Além da inserção, o *backend* oferece funcionalidades para remoção de documentos individuais ou da base completa, garantindo o controle e gerenciamento eficaz dos dados.

4.2.2 Fluxo de interação no chat

A funcionalidade de chat inteligente permite que o usuário interaja, em linguagem natural, com as bases de conhecimento previamente criadas a partir de documentos em PDF. Esse recurso possibilita a recuperação de informações específicas com base no conteúdo inserido pelo próprio usuário, sem depender de mecanismos de busca externos.

Ao receber uma pergunta, o sistema realiza a vetorização da consulta utilizando o mesmo modelo de *embeddings* semânticos empregado na etapa de inserção dos documentos. Esse vetor é comparado com os vetores armazenados no banco de dados PostgreSQL por meio de busca vetorial baseada em distância cosseno, retornando os trechos mais semanticamente similares.

Após essa recuperação inicial, os resultados passam por um processo de reranqueamento com o algoritmo BM25 [Robertson and Zaragoza, 2009]. Essa etapa tem como objetivo reorganizar os trechos com base na correspondência textual entre a pergunta do usuário e o conteúdo recuperado, considerando a frequência e a relevância dos termos presentes. Isso permite uma combinação entre semântica e relevância lexical, priorizando os trechos que, além de semanticamente próximos, também contêm termos diretamente relacionados à pergunta. O *reranking*, portanto, contribui significativamente para o aumento da precisão e da utilidade das respostas geradas.

Com os trechos reranqueados, o sistema realiza a montagem do *prompt* que será enviado ao modelo de linguagem. Essa montagem ocorre em duas etapas: na primeira, a pergunta original é refinada com base no conteúdo encontrado (técnica conhecida como RQ-RAG), tornando-a mais clara e orientada ao domínio dos documentos. Na segunda, é construído o *prompt* final, que inclui a pergunta refinada, os trechos recuperados e instruções específicas para o modelo.

Um ponto essencial dessa instrução é a orientação explícita para que a resposta seja gerada exclusivamente com base nas informações contidas nos documentos da base de conhecimento. O *prompt* deixa claro que não deve haver uso de dados externos, da internet ou de conhecimento genérico, mesmo que a pergunta envolva assuntos amplos. Essa diretriz garante que o conteúdo gerado esteja sempre ancorado em evidências fornecidas pelo próprio usuário, respeitando os limites do escopo definido.

A geração da resposta ocorre de forma local, por meio do modelo Gemini 2.0 Flash, do Google [2025], acessado pela biblioteca *langchain_google_genai*. O modelo processa o *prompt* estruturado e retorna uma resposta contextualizada, coerente e rastreável, com base exclusivamente nos documentos armazenados.

4.3 Frontend

O *frontend* do sistema foi desenvolvido com React, uma biblioteca JavaScript de código aberto mantida pelo Facebook, que permite que você crie interfaces de usuário a partir de peças individuais chamadas de componentes [React, 2025]. O React possibilita a criação de aplicações web dinâmicas e escaláveis por meio de uma abordagem declarativa, facilitando o desenvolvimento e a manutenção da interface.

A estrutura do projeto foi organizada de forma modular, com separação entre páginas, componentes, *hooks* e serviços. Ferramentas como Vite foram utilizadas para otimizar o ambiente de desenvolvimento, proporcionando um carregamento rápido e uma experiência ágil durante a codificação [Evans, 2025]. Também foi utilizado o TailwindCSS, um framework CSS utilitário, que facilitou a construção de uma interface limpa, responsiva e customizável [Martins, 2025].

O sistema conta com três telas principais, cada uma voltada para funcionalidades específicas. Na tela das bases de conhecimento, o usuário visualiza todas as bases que possui e pode criar uma nova base, inserindo um nome e uma descrição, além de gerenciar ou excluir bases já existentes, o que facilita a organização e o controle dos conteúdos armazenados.

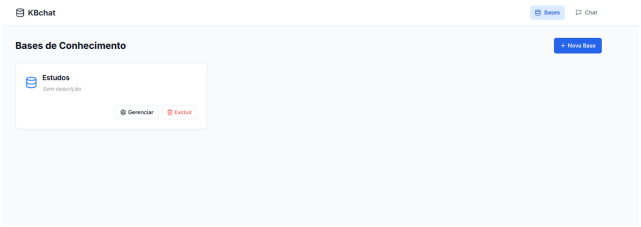

Figura 3. Interface de gerenciamento das bases de conhecimento da plataforma KBchat, exibindo a opção de criação de novos repositórios e a base "Estudos"criada.

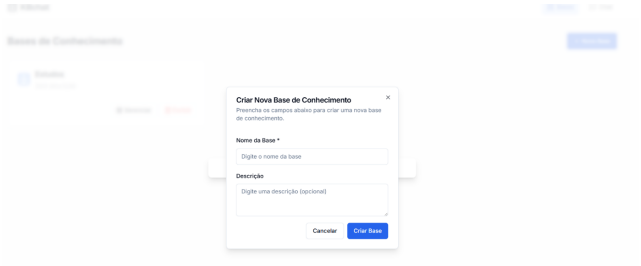

Figura 4. Caixa de diálogo para criação de uma nova base de conhecimento, apresentando o campo obrigatório "Nome da Base"e o campo de preenchimento opcional "Descrição".

Ao clicar em gerenciar uma base, opção este presente na tela de bases de conhecimento, o usuário é direcionado para a tela de gerenciamento da base de conhecimento, onde pode enviar novos documentos para a base selecionada ou

apagar documentos existentes, permitindo a manutenção e atualização do conteúdo de forma prática e direta.

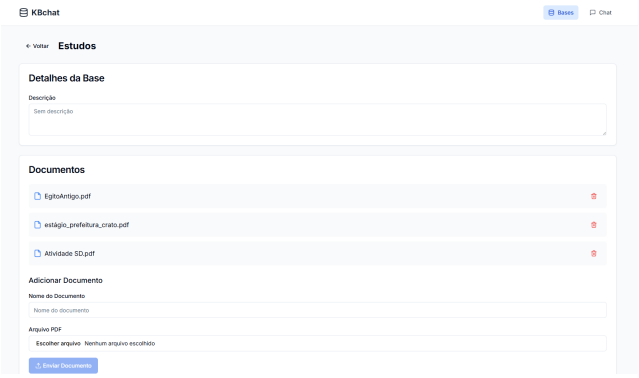

Figura 5. Interface de gerenciamento da base de conhecimento "Estudos", apresentando a lista de documentos indexados e a seção para upload e inserção de novos arquivos em formato PDF.

Já na tela de chat, o usuário tem a opção de escolher qual base de conhecimento vai usar como contexto para conversa, e a caixa de texto para enviar sua pergunta para a inteligência artificial.

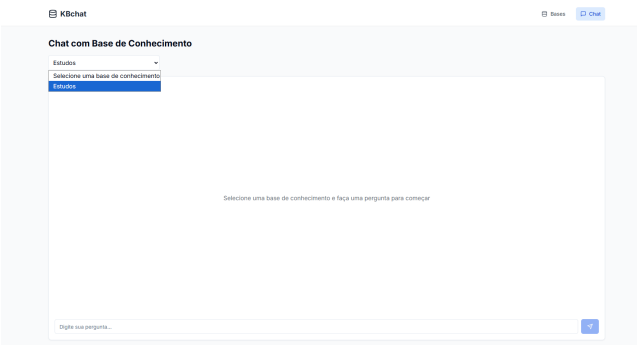

Figura 6. Interface de conversação do sistema KBchat, destacando o menu de seleção da base de conhecimento ("Estudos") e o campo de inserção de perguntas do usuário.

4.3.1 Testes de validação do sistema

Para a condução dos experimentos, foi utilizado um documento com aproximadamente 10 páginas, contendo conteúdo relacionado à Promoção de Saúde na Escola, do Ministério da Saúde e do Sistema Único de Saúde (SUS)[4]. A escolha desse material buscou representar um cenário real de consulta a documentos técnicos e informativos. Para avaliar o sistema, foi construído manualmente um conjunto de teste composto por 30 perguntas e suas respectivas respostas de referência, elaboradas por um especialista da área da saúde, a partir do conteúdo do documento. O conjunto foi dividido em duas categorias: (i) perguntas básicas, formuladas de maneira direta e com elevada similaridade lexical em relação ao texto original; e (ii) perguntas avançadas, construídas com o uso de sinônimos, paráfrases e variações linguísticas, com o objetivo de avaliar a capacidade do sistema em recuperar e interpretar informações semanticamente equivalentes. No total, foram consideradas 20 perguntas básicas e 10 perguntas avançadas.

[4]Disponível em: `https://www.gov.br/saude/pt-br/composicao/saps/pse/arquivos/2021/faq_pse-1.pdf/view`. Acesso em 21 de julho de 2026.

A avaliação foi conduzida manualmente por um especialista da área da saúde, tomando como referência alguns dos atributos conceituais empregados pelo *framework* RAGAS [Es *et al.*, 2024] (como, por exemplo: *Faithfulness*, *Answer Relevancy*, *Context Precision* e *Context Recall*), especialmente aqueles relacionados à relevância da resposta em relação à pergunta (*Answer Relevancy*) e à aderência da resposta ao conteúdo presente na base de conhecimento (*Faithfulness*). Diferentemente da aplicação completa do *framework*, que utiliza modelos de linguagem para o cálculo automatizado de métricas específicas, neste trabalho para cada pergunta do conjunto de teste, a resposta produzida pelo sistema foi comparada manualmente com uma resposta de referência e classificada como correta ou incorreta.

Logo, uma resposta foi considerada correta quando atendia simultaneamente aos seguintes critérios: (i) respondia adequadamente à pergunta formulada; (ii) apresentava informações compatíveis com o conteúdo do documento utilizado como base de conhecimento; (iii) não introduzia informações contraditórias ou não suportadas pelo contexto recuperado. Caso algum desses critérios não fosse atendido, a resposta era classificada como incorreta.

Os resultados obtidos são apresentados na Tabela 4. Observa-se que o sistema respondeu corretamente a 18 das 20 perguntas básicas, alcançando uma taxa de acerto de 90%. Para as perguntas avançadas, foram obtidos 8 acertos em 10 questões, correspondendo a uma taxa de acerto de 80%. Considerando o conjunto completo de avaliação, o sistema respondeu corretamente a 26 das 30 perguntas propostas, resultando em uma taxa geral de acerto de 86,66%. Os resultados indicam que o sistema apresenta desempenho consistente tanto em perguntas diretamente relacionadas ao texto-fonte quanto em questões formuladas com maior variação linguística. Contudo, observa-se uma redução moderada no desempenho para perguntas avançadas, sugerindo que a recuperação e interpretação de informações expressas por meio de paráfrases e sinônimos ainda representam um desafio para o *pipeline* adotado.

Cabe destacar que o documento utilizado nos experimentos apresentava estrutura heterogênea, contendo tanto seções bem organizadas, com títulos, subtítulos e tópicos claramente definidos, quanto trechos menos estruturados. Essa característica influencia diretamente o desempenho de sistemas baseados em RAG, uma vez que documentos mais organizados tendem a favorecer a segmentação em trechos semanticamente coerentes e, consequentemente, melhorar a recuperação de contexto relevante. Em contrapartida, conteúdos menos estruturados podem introduzir ruído durante o processo de recuperação, impactando a qualidade das respostas geradas.

Tabela 4. Resultados da avaliação manual do sistema

Categoria	Perguntas	Acertos	Taxa de Acerto	Descrição
Perguntas básicas	20	18	90%	Formulações diretas com elevada similaridade lexical em relação ao texto-fonte.
Perguntas avançadas	10	8	80%	Formulações com sinônimos, paráfrases e variações linguísticas.
Total	30	26	86%	Conjunto completo de avaliação.

5 Conclusão

O sistema de chat utilizando a metodologia RAG proposto no presente trabalho se mostrou uma opção viável para consultas em bases de conhecimento específicas. Ao unir técnicas atuais de Processamento de Linguagem Natural (PLN), como incorporações semânticas, busca vetorial, reclassificação e criação contextualizada por LLMs, foi possível desenvolver um sistema funcional e acessível, capaz de fornecer respostas relevantes com base em documentos enviados pelo próprio usuário.

Uma característica positiva do sistema é a garantia que as respostas sejam geradas apenas com os conteúdos armazenados na base de conhecimento, sem utilização de fontes externas. Essa abordagem reforça a confiabilidade do sistema em situações onde a rastreabilidade e a fidelidade às informações são importantes.

Durante o desenvolvimento, foram realizadas escolhas técnicas baseadas em testes comparativos que permitiram selecionar ferramentas e modelos mais adequados ao objetivo proposto. A escolha do modelo Gemini 2.0 Flash, da biblioteca Markitdown e do all-mpnet-base-v2 auxiliou no equilíbrio entre o desempenho, a qualidade e a disponibilidade do sistema. A divisão da arquitetura em módulos, tanto no *backend* quanto no *frontend*, também facilitou a organização, a manutenção e futuras expansões do sistema. Além disso, o presente trabalho buscou enfrentar um dos desafios atuais, ao tornar o acesso às tecnologias avançadas de inteligência artificial mais democrático; ao oferecer um sistema gratuito, oferecendo uma alternativa concreta para pessoas e instituições que não têm acesso a uma infraestrutura de alto custo de aquisição, mas que precisam explorar o potencial da IA para organizar e pesquisar grandes volumes de informações em suas bases.

Como sugestões para trabalhos futuros, destacam-se a inclusão de ferramentas de OCR para permitir a extração de conteúdo de imagens em documentos digitalizados e a exploração de fluxos completos baseados em novas tendências do RAG, como o Agentic RAG, que integra mecanismos de planejamento e execução de tarefas de forma mais autônoma, permitindo que o sistema realize múltiplas etapas de busca e geração de respostas de maneira encadeada. Outra sugestão de aprimoramento como trabalhos futuros, é a adoção de estratégias avançadas de recuperação, tais como o RAG-Fusion, que demonstram capacidade de combinar múltiplas fontes de evidência, elevando a qualidade dos trechos recuperados; e, a utilização de modelos de *embeddings* com maior dimensionalidade, que pode se apresentar como uma alternativa importante para a captura de características semânticas mais sutis, favorecendo interpretações contextuais mais ricas e precisas.

Por fim, o sistema aqui apresentado atendeu aos escopo proposto, e espera-se contribuir com a aplicação prática de RAG em sistemas interativos e acessíveis, ao mesmo tempo em que, tal sistema, possa também servir de base para futuras pesquisas e aprimoramentos na área de sistemas inteligentes baseados em documentos.

Declarações complementares

Contribuições dos autores

Davi Santos Tavares: Concepção e desenho do estudo (Conceptualization); Desenvolvimento do software e experimentos (Software, Investigation); Análise e interpretação dos resultados (Formal analysis); Redação do manuscrito (Writing – Original Draft, Visualization).

Robson Gonçalves Fechine Feitosa: Supervisão técnica e metodológica (Supervision, Methodology); Revisão e edição do manuscrito (Writing – Review & Editing); Validação e verificação dos resultados (Validation).

Todos os autores leram e aprovaram o manuscrito final.

Conflitos de interesse

Os autores declaram que não têm nenhum conflito de interesses.

Disponibilidade de dados e materiais

Os conjuntos de dados e os códigos-fonte gerados e/ou analisados durante o presente estudo serão disponibilizados mediante solicitação ao autor correspondente.

Referências

Aquino, I. V. d., dos Santos, M. M., Dorneles, C. F., and Carvalho, J. T. (2024). Extracting information from brazilian legal documents with retrieval augmented generation. In *Proceedings of the 39th Workshop on Data Science Against Corruption in the Public Sector (DS-COPS) - Simpósio Brasileiro de Banco de Dados (SBBD)*, pages 280–287, Florianópolis, SC, Brazil. Sociedade Brasileira de Computação. DOI: 10.5753/sbbd_estendido.2024.244241.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., *et al*. (2020). Language models are few-shot learners. In *Advances in neural information processing systems*, volume 33, pages 1877–1901. Disponível em: `https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf`. Acesso em: 22 jul. 2026.

Chan, C., Xu, C., Yuan, R., Luo, H., Xue, W., Guo, Y., and Fu, J. (2024). RQ-RAG: Learning to refine queries for retrieval-augmented generation. In *Conference on Language Modeling (COLM 2024)*. Disponível em: `https://openreview.net/forum?id=tzE7VqsaJ4`. Acesso em: 22 jul. 2026.

Datalab-to (2025). Marker: Convert pdf to markdown json quickly with high accuracy. Disponível em: `https://github.com/datalab-to/marker`. Acesso em: 22 jul. 2026.

Es, S., James, J., Anke, L. E., and Schockaert, S. (2024). Ragas: Automated evaluation of retrieval augmented generation. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations (EACL)*, pages 150–158. DOI: 10.18653/v1/2024.eacl-demo.16.

Evans, M. (2025). Vite: A next generation frontend tooling. Disponível em: `https://vitejs.dev/`. Acesso em: 22 jul. 2026.

Flask Project (2025). Flask. Disponível em: `https://`

flask.palletsprojects.com/. Acesso em: 22 jul. 2026.

Google (2025). Gemini 2.0 flash. Disponível em: https://cloud.google.com/vertex-ai/generative-ai/docs/models/gemini/2-0-flash. Acesso em: 22 jul. 2026.

Karpukhin, V., Oguz, B., Min, S., Lewis, P., Wu, L., Edunov, S., Chen, D., and Yih, W.-t. (2020). Dense passage retrieval for open-domain question answering. In Webber, B., Cohn, T., He, Y., and Liu, Y., editors, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6769–6781, Online. Association for Computational Linguistics. DOI: 10.18653/v1/2020.emnlp-main.550.

Khattab, O. and Zaharia, M. (2020). Colbert: Efficient and effective passage search via contextualized late interaction over bert. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '20, page 39–48, New York, NY, USA. Association for Computing Machinery. DOI: 10.1145/3397271.3401075.

Kuratomi, G., Pirozelli, P., Cozman, F. G., and Peres, S. M. (2024). A rag-based institutional assistant. In *Proceedings of the 21st Encontro Nacional de Inteligência Artificial e Computacional (ENIAC)*, pages 755–766, Belém, PA, Brazil. Sociedade Brasileira de Computação. DOI: 10.5753/eniac.2024.245243.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., Riedel, S., and Kiela, D. (2020). Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA. Curran Associates Inc. Disponível em: https://proceedings.neurips.cc/paper_files/paper/2020/file/6b493230205f780e1bc26945df7481e5-Paper.pdf. Acesso em: 22 jul. 2026.

Martins, A. (2025). Tailwindcss: Estilos utilitários para um design eficiente. Disponível em: https://tailwindcss.com/. Acesso em: 22 jul. 2026.

Microsoft (2025). Markitdown. Disponível em: https://github.com/microsoft/markitdown. Acesso em: 22 jul. 2026.

pgvector (2025). pgvector. Disponível em: https://github.com/pgvector/pgvector. Acesso em: 22 jul. 2026.

React (2025). Aprenda react. Disponível em: https://pt-br.react.dev/learn/. Acesso em: 22 jul. 2026.

Robertson, S. E. and Zaragoza, H. (2009). *The Probabilistic Relevance Framework: BM25 and Beyond*, volume 3, pages 333–389. Now Publishers. DOI: 10.1561/1500000019.

Song, K., Tan, X., Qin, T., Lu, J., and Liu, T.-Y. (2020). Mpnet: Masked and permuted pre-training for language understanding. *Advances in neural information processing systems*, 33:16857–16867. Disponível em: https://proceedings.neurips.cc/paper_files/paper/2020/file/c3a690be93aa602ee2dc0ccab5b7b67e-Paper.pdf. Acesso em: 22 jul. 2026.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 6000–6010, Red Hook, NY, USA. Curran Associates Inc. Disponível em: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf. Acesso em: 22 jul. 2026.