Processo Seletivo
Iniciação Científica em NLP para Serviços de Saúde
Etapa Prática - Relatório

Matheus Gasparotto Lozano

10 de junho de 2024

1 Introdução

RAGs (*Retrieval Augmented Generation*) são sistemas de inteligência artificial que combinam a recuperação de informações com a geração de texto. Esses sistemas integram mecanismos de busca com modelos de linguagem avançados para melhorar a precisão e relevância das respostas geradas. Basicamente, eles primeiro recuperam documentos ou trechos relevantes de uma base de dados ou da web, e em seguida, utilizam esses dados recuperados para gerar respostas mais informadas e contextualmente apropriadas. Essa abordagem é especialmente útil em tarefas que requerem respostas detalhadas e precisas baseadas em grandes volumes de informações.

Este trabalho tem como objetivo implementar um chatbot baseado em RAG que responda às dúvidas sobre o Vestibular da Unicamp 2024 a partir da publicação da Resolução GR-031/2023, de 13/07/2023 que "Dispõe sobre o Vestibular Unicamp 2024 para vagas no ensino de Graduação", disponível em https://www.pg.unicamp.br/norma/31594/0. É dado foco especial nas ferramentas utilizadas para construir este sistema, bem como no modo como o ChatGPT foi utilizado para auxiliar o desenvolvimento.

2 Metodologia

2.1 Uso do ChatGPT

O ChatGPT foi utilizado principalmente durante o desenvolvimento da interface gráfica (que será melhor explicado na seção 2.3), gerando melhoras contínuas até que o resultado desejado fosse obtido. Para a criação do modelo de RAG, por outro lado, o código gerado não foi de grande utilidade, pois os exemplos eram sempre gerados com uso da API da OpenAI (mesmo quando o *prompt* dizia explicitamente para utilizar outras APIs) e a implementação das funções de *retrieval* e *generation* era genérica demais para uso prático. Nesse caso, o ChatGPT foi mais útil para gerar documentação do código e, novamente, correção de erros e exceções gerados.

Para escrita do relatório, o ChatGPT foi utilizado para gerar o primeiro parágrafo da introdução, bem como auxiliar na reescrita de frases ou ideias que julguei mal-formuladas por mim.

2.2 Implementação do Modelo RAG

Para implementação do modelo de RAG e lógica de funcionamento do *chatbot*, foi utilizado o *framework* *LangChain*. Conforme dito na seção 2.1, os códigos gerados pelo ChatGPT não foram de grande ajuda para implementação desta funcionalidade. Assim, para criação do *pipeline* do *RAG* foi seguido, em grande parte, o tutorial disponível na documentação [1].

Primeiro, era necessário obter e pré-processar as informações presentes na URI da Resolução GR-031/2023. o Pacote `langchain_comunnity` foi usado para isso, juntamente com o `BeautifulSoup` para obter apenas os trechos desejados do documento HTML. O texto obtido foi então dividido em documentos com o uso da função `RecursiveCharacterTextSplitter`, disponibilizada pelo *LangChain*. Em seguida, era necessáio transformar a lista de documentos obtidas em um vetor de *embeddings* a ser passado para o modelo. Para isso, foi utilizada a solução *HuggingFaceEmbeddings*, pois ela fornece vários modelos diferentes que podem ser utilizados para gerar diferentes *embeddings* dependendo da necessidade [2]. Esses *embeddings* foram transformados em um *vector store* por meio da base de dados *Chroma*, e esse *vector store* foi escrito no disco.

Por fim, foi criada a "corrente de RAG". Nesse caso, o *vector store* de *embeddings* é carregado do disco e o LLM utilizado para geração das respostas (que será mais detalhado no parágrafo seguinte) é instanciado.

Além disso, é aqui que são passadas as instruções para a geração de *prompt*; principalmente, foi dito para usar as informações do documento para gerar as respostas e sempre responder em português da forma mais concisa possível.

Para geração das respostas com base no contexto obtido, foi utilizada a API do Groq, que utiliza o modelo Llama 3 da Meta. Tal solução foi escolhida por ser gratuita e com um limite de requisições diárias suficiente para esta aplicação. A API foi integrada ao resto do código por meio do pacote *langchain-groq*, seguindo os passos descritos na documentação [3]. A chave da API do Groq foi armazenada em um arquivo `.env` na raiz do diretório, conforme descrito em [4].

2.3 Implementação da Interface Gráfica e *Deploy*

Para implementação da interface gráfica, foi utilizado o *framework Streamlit*, em especial o componente *streamlit-chat*. Tais ferramentas foram utilizadas por possibilitarem a criação de uma UI de um *chatbot* de forma prática e rápida, necessitando apenas de algumas linhas de código para gerar um protótipo funcional. Além disso, o *Streamlit*, por meio do *Streamlit Community Cloud*, fornece uma plataforma gratuita e fácil de configurar para realização de *deploy* da aplicação.

Conforme mencionado na seção 2.1, grande parte desta funcionalidade foi implementada com uso do ChatGPT, através de um processo de melhorias contínuas no código inicialmente gerado. Não houve maiores dificuldades para implementação da interface gráfica, porém não foi possível realizar *deploy* do sistema em sua forma completamente funcional. Apesar da integração com GitHub e instalação das dependências ter ocorrido sem grandes problemas (nada que não pudesse ser selecionado com ajuda do ChatGPT), não foi possível realizar *upload* do arquivo onde o *vector store* foi escrito em disco, como descrito na seção 2.2. Assim, apesar do *deploy* do *chatbot* estar funcionando sem gerar nenhum erro de execução, ele não consegue acessar o contexto para responder às perguntas enviadas.

3 Execução e Resultados

Para executar o sistema localmente, é necessário configurar a variável de ambiente `GROQ_API_KEY`. Isso pode ser feito por meio do comando `export GROQ_API_KEY=<sua-chave-api>`.

Após clonar o repositório (disponível em https://github.com/Lozavival/chatbot-vestibular-unicamp/tree/main) e instalar as dependências, o primeiro passo para testar o sistema é buscar os documentos na *web* (isto é, buscar a redação da Resolução GR-031/2023 no URL fornecido) e gerar os *embeddings* que serão usados pelo modelo para gerar as respostas. Para isso, basta executar o comando `fab CreateEmbeddings`. Após a conclusão desse processo, é possível executar o *chatbot* tanto via interface de linha de comando (CLI) ou via interface gráfica no navegador (UI), o que é feito através dos comandos `fab RunChatbotCLI` ou `fab RunChatbotUI`, respectivamente. Alternativamente, a versão UI também pode ser executada pelo comando `streamlit run src/chatbot_app.py`.

As Figuras 1 e 2 mostram exemplos do funcionamento de ambas as versões. Podemos ver que, em ambos os casos, foram geradas as respostas corretas para cada pergunta, a saber, o número de vagas oferecidas, a fórmula do cálculo da nota padronizada, o valor da taxa de inscrições, a relação de livros e as datas de cada prova. Por outro lado, a Figura 3 mostra exemplos de *prompts* que não obtiveram resposta do modelo: o número de vagas reservadas ao ENEM e o tempo de prova da 2ª fase. Buscando manualmente na Resolução GR-031/2023, o número de vagas oferecidas pelo Edital ENEM-Unicamp 2024 é estabelecido pelo Artigo 1º, Inciso I, enquanto o tempo de prova de cada dia da 2ª fase é estabelecido pelo Artigo 23º, Parágrafo 4º.

Figura 1: Exemplo de funcionamento da versão CLI do chatbot

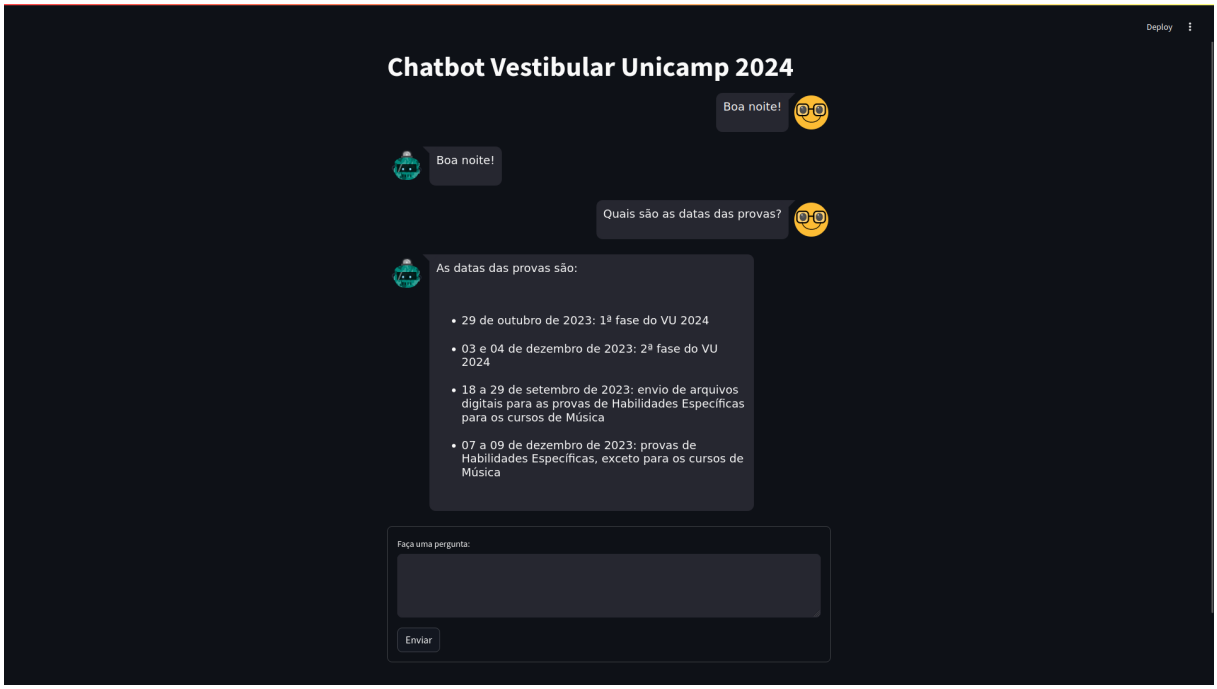

Figura 2: Exemplo de funcionamento da versão UI do chatbot

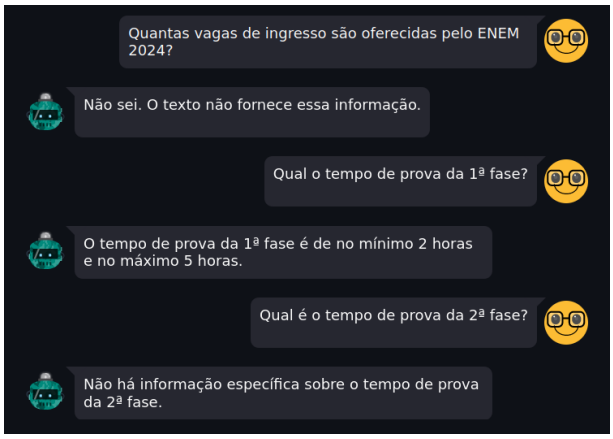

Figura 3: Exemplo de funcionamento da versão UI do chatbot

Sendo assim, é possível ver que, embora o *chatbot* saiba responder a algumas perguntas básicas corretamente, ele não consegue obter a informação necessária para outras, mesmo quando tal informação está presente no texto da resolução. Apesar disso, uma vantagem que o sistema apresentou é que não foram geradas "alucinações"em nenhum dos testes realizados, ou seja, o modelo não inventou informações quando não foi capaz de encontrá-las no texto, em vez disso sempre retornando variações das respostas mostradas na Figura 3.

4 Conclusão e Trabalhos Futuros

Apesar do *chatbot* não ter sido capaz de encontrar algumas informações básicas no texto, o sistema se mostrou capaz de responder corretamente à maioria das perguntas feitas durante os testes. O ChatGPT revelou-se fundamental para impulsionar o desenvolvimento do projeto, pois acelerou a criação do sistema, frequentemente oferecendo a estrutura de código pronta para implementação, sendo necessárias apenas correções feitas com base na documentação das ferramentas empregadas.

Os próximos passos incluem a integração do sistema com um banco de dados para armazenar o *vector store* de *embeddings* gerado, possibilitando que seja feito *deploy* do *chatbot*. Além disso, é possível testar diferentes modelos de *embedding* do *HuggingFace*, bem como variar os parâmetros `chunk_size` e `chunk_overlap` do separador de texto, em busca de uma melhor acurácia nas respostas geradas.

Referências

[1] LangChain, Inc. Build a retrieval augmented generation (rag) app, 2024. URL https://python.langchain.com/v0.2/docs/tutorials/rag/.

[2] Hugging Face. Models. URL https://huggingface.co/models?library=sentence-transformers.

[3] LangChain, Inc. Groq, 2024. URL https://python.langchain.com/v0.2/docs/integrations/chat/groq/.

[4] Max Kahan. Python environment variables (env vars): A primer, 2023. URL https://developer.vonage.com/en/blog/python-environment-variables-a-primer#store-environment-variables-in-files.