Processo Seletivo NeuralMind

Projeto: Chatbot para dúvidas sobre o vestibular Unicamp 2025

Augusto Dante De Carli Zolet

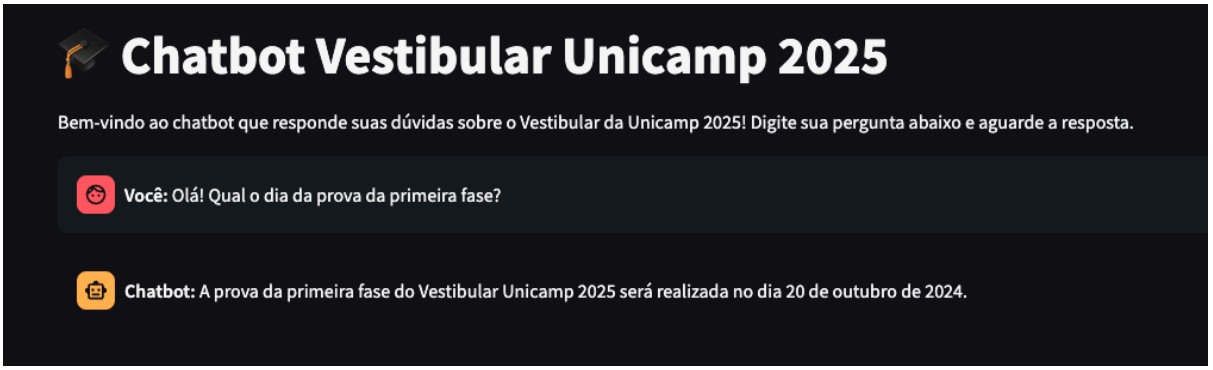

Figura 1: Interface do chatbot.

Introdução

Este relatório apresenta o desenvolvimento de um chatbot baseado em RAG (Retrieval Augmented Generation) para responder dúvidas sobre o Vestibular da Unicamp 2025.

Disponível em: https://zolebot-neuralmind.streamlit.app/

O chatbot foi construído para auxiliar candidatos do vestibular, fornecendo respostas baseadas na Resolução GR-029/2024, de 10/07/2024, que dispõe sobre o Vestibular Unicamp 2025 para vagas no ensino de Graduação. O documento oficial utilizado como fonte está disponível em Resolução Vestibular Unicamp 2025.

Para o desenvolvimento, foram utilizadas técnicas de RAG, integrando modelos de linguagem como o GPT-3.5-turbo e ferramentas de recuperação de informações para proporcionar respostas contextuais e fundamentadas. Optou-se pelo uso de frameworks gratuitos como LangChain para a implementação do RAG e Streamlit para a interface do usuário, como sugerido na proposta do exercício.

Ao longo do projeto, o chatGPT foi utilizado extensivamente para auxiliar em todas as fases, desde a compreensão do edital e concepção do sistema até a codificação, resolução de problemas e elaboração deste relatório.

Este relatório detalha as etapas do desenvolvimento, incluindo a análise do edital, a preparação dos dados, a implementação técnica, os métodos de avaliação da qualidade das respostas e as conclusões obtidas. Também são discutidos os desafios enfrentados e as soluções adotadas, bem como reflexões sobre o uso do chatGPT como ferramenta de apoio no projeto.

Processamento e Preparação dos Dados do Edital

Para que o chatbot pudesse responder com precisão às perguntas sobre o Vestibular da Unicamp 2025, foi necessário extrair e preparar o texto integral do edital oficial. O edital, disponibilizado em formato PDF, apresentava desafios em termos de extração de texto no formato correto.

Inicialmente, utilizei um script em Python para automatizar a extração do texto do PDF. O código empregou as bibliotecas *PyPDF2* para a leitura do documento e *re* para a manipulação de expressões regulares, visando remover elementos indesejados.

Este script realiza as seguintes etapas:

1. **Leitura do PDF**: Utiliza o PyPDF2 e extrai o texto bruto.
2. **Limpeza do Texto**: Aplica uma expressão regular para remover rodapés que poderiam interferir na compreensão do texto, como datas, referências e números de página.
3. **Salvamento do Texto Limpo**: Grava o texto processado em um arquivo .txt, que será utilizado nas etapas subsequentes do projeto.

Apesar da extração automatizada, o texto resultante ainda apresentava problemas, principalmente:

1. **Informações Fora de Lugar**: Devido à formatação original do PDF, alguns parágrafos e sentenças ficaram desordenados.
2. **Tabelas Desconfiguradas**: As tabelas, que continham informações cruciais como cronogramas, pesos de disciplinas e vagas disponíveis, perderam sua estrutura original.

Para resolver esses problemas, realizei uma análise manual detalhada do texto extraído e utilizei o gpt para reconstruir as tabelas em um formato mais amigável para LLMs .

Com isso, obtive um texto mais limpo e adequado para interpretação do modelo, que por sua vez será capaz de responder mais adequadamente aos questionamentos do usuário.

Detalhes da implementação

Nesta seção está detalhada a implementação técnica do chatbot. O projeto foi desenvolvido em Python, utilizando uma combinação de bibliotecas e frameworks para construir um sistema de Retrieval-Augmented Generation (RAG). A seguir, descrevemos os componentes principais do código e como eles interagem para fornecer respostas precisas aos usuários.

O projeto utiliza diversas tecnologias e bibliotecas para garantir eficiência e funcionalidade. A linguagem de programação é o Python, que oferece as ferramentas necessárias para o desenvolvimento. Para a criação de aplicativos web interativos, foi escolhido o Streamlit.

```python
if question:
    # Adicionar a mensagem do usuário ao estado da sessão
    st.session_state.messages.append({"role": "user", "content": question})
    with st.chat_message("user"):
        st.markdown(f"**Você:** {question}")

    # Gerar a resposta do assistente
    with st.spinner('Gerando resposta...'):
        try:
            answer = generate_answer(st.session_state.messages, embeddings, index, texts)
            # Adicionar a resposta do assistente ao estado da sessão
            st.session_state.messages.append({"role": "assistant", "content": answer})
            with st.chat_message("assistant"):
                st.markdown(f"**Chatbot:** {answer}")
        except Exception as e:
            st.error(f"Ocorreu um erro: {e}")
```

Figura 2: Trecho de código em que o Streamlit é utilizado.

O processamento de texto é realizado com o auxílio do NLTK (Natural Language Toolkit), utilizado para tokenização e manipulação textual. A biblioteca LangChain é empregada para facilitar a divisão e manipulação de textos, enquanto o FAISS (Facebook AI Similarity Search) é usado para organizar e encontrar rapidamente dados que foram transformados em números (vetores), como os embeddings, que representam textos ou outros tipos de informação.

```python
# Função para pré-processar o texto
def preprocess_text(text):
    sentences = sent_tokenize(text)
    return sentences

# Função para dividir o texto em chunks
def split_text(text):
    text_splitter = CharacterTextSplitter(
        separator='\n',
        chunk_size=2000,
        chunk_overlap=400
    )
    texts = text_splitter.split_text(text)
    return texts

# Função para criar embeddings e indexação com FAISS
def create_embeddings(texts):
    embeddings = OpenAIEmbeddings()
    doc_embeddings = embeddings.embed_documents(texts)
    dimension = len(doc_embeddings[0])
    index = faiss.IndexFlatL2(dimension)
    index.add(np.array(doc_embeddings))
    return embeddings, index
```

Figura 3: Código das funções de tokenização, split e criação de embeddings do texto .

A OpenAI API desempenha um papel central ao permitir o acesso direto ao modelo GPT-3.5-turbo, que é usado para entender e responder às perguntas dos usuários baseadas nas informações fornecidas como contexto. Através dessa API, o chatbot pode interagir com o modelo de linguagem e, além disso, a API oferece flexibilidade para personalizar os parâmetros de geração, como a temperatura e o número máximo de tokens, garantindo que as respostas sejam relevantes e adaptadas ao contexto do edital. Essa funcionalidade é essencial para viabilizar a inteligência por trás do chatbot.

```python
# Função para gerar a resposta usando o modelo atualizado
def generate_answer(messages, embeddings, index, texts):
    question = messages[-1]["content"]
    context = search_docs(question, embeddings, index, texts)
    context_str = "\n\n".join(context)

    api_messages = [
        {"role": "system", "content": "Você é um assistente especializado no Vestibular da Uni
    ]

    previous_messages = messages[-3:] if len(messages) >= 3 else messages
    for msg in previous_messages[:-1]:
        api_messages.append(msg)

    last_user_message = messages[-1]
    last_user_message_with_context = {
        "role": last_user_message["role"],
        "content": f"Contexto:\n{context_str}\n\nPergunta:\n{last_user_message['content']}"
    }
    api_messages.append(last_user_message_with_context)

    chat_completion = client.chat.completions.create(
        model="gpt-3.5-turbo",
        messages=api_messages,
        max_tokens=400,
        temperature=0.3
    )
    answer = chat_completion.choices[0].message.content.strip()
    return answer
```

Figura 4: Código da função que define o modelo e detalhes da interação .

O fluxo geral da aplicação começa com o pré-processamento do texto, onde o edital é carregado e tratado para eliminar inconsistências e formatar os dados. Em seguida, o texto é dividido em chunks, facilitando a indexação e a busca.

Para cada chunk, são gerados embeddings com o auxílio da OpenAI API, que posteriormente são indexados utilizando a biblioteca FAISS para permitir buscas eficientes. Após essa etapa, o usuário interage com o sistema inserindo uma pergunta na interface do chatbot.

O sistema então realiza a busca de contexto, localizando os chunks mais relevantes para a pergunta do usuário. Com base nesses chunks, o modelo GPT-3.5-turbo gera uma resposta contextualizada e informativa. Por fim, a resposta é exibida ao usuário na interface, proporcionando uma experiência interativa e prática.

Configurações do Modelo e do Sistema:

Tanto a API da OpenAI, quanto algumas ferramentas utilizadas oferecem algumas possibilidades de configuração do funcionamento. Nesse contexto, algumas dessas configurações receberam maior atenção devido ao impacto direto no desempenho do chatbot. As configurações a seguir estão evidenciadas nas figuras 3 e 4.

Para prompt:

> **Content:** Contém o texto que estabelece as regras ou o comportamento esperado do modelo.

Com auxílio do Chat GPT, através de repetidos prompts, busquei definir o content de forma a otimizar a resposta do bot. Por também ser uma LLM, o GPT tende a entender melhor os padrões de funcionamento e interpretação desses modelos, sendo assim ideal para otimizar essa configuração.

Content definido:

```
"Você é um assistente especializado no Vestibular da Unicamp 2025. Responda às perguntas dos usuários usando apenas as informações do edital oficial fornecido. Seja preciso e direto. Se não encontrar a resposta no contexto, indique que não possui essa informação."
```

Com isso é possível otimizar o comportamento chatbot, evitando alucinações e restringindo o escopo ao vestibular da Unicamp 2025, reforçando a relevância e precisão das respostas.

Para respostas do modelo:

> **Max Tokens**: Define o limite máximo de tokens que o modelo pode gerar em uma única resposta. 400 tokens se mostraram mais do que suficientes.

> **Temperatura**: Controla a aleatoriedade das respostas. Valores mais baixos resultam em respostas mais diretas e consistentes, sendo assim uma temperatura baixa, de 0.3, foi definida.

Para divisão do texto:

> **Chunk Size**: Define o tamanho máximo dos trechos de texto (chunks) em caracteres para dividir o conteúdo do edital.

> **Chunk Overlap**: Determina a quantidade de sobreposição entre chunks consecutivos, garantindo continuidade e contexto entre os trechos.

Após alguns testes, tanto o chunk size como o chunk overlap demonstraram ter grande influência no desempenho das respostas. Mais especificamente, os testes apontaram para um desempenho significativamente melhor conforme esses valores aumentam. Assim, foram definidos valores de 8000 caracteres para o chunk size e 1600 para o chunk overlap, aumentando o desempenho sem exigir excessivamente do modelo.

Testes:

Os testes iniciaram de forma manual, realizando perguntas simples como a data da prova, lista de cursos de exatas e números de vagas por curso. Logo foi possível notar a dificuldade de retornar corretamente informações presentes em tabelas. Isso direcionou meus esforços descritos na seção de Processamento e Preparação dos Dados do Edital.

Uma vez realizado diversos aprimoramentos nos dados, era preciso encontrar uma forma mais eficiente de realizar os testes. Nesse contexto, com o auxílio do Chat GPT, um script de teste automático simples foi criado. Esse script realiza perguntas pré-definidas ao chatbot e imprime na tela o conjunto de perguntas, respostas e respostas esperadas.

Essas perguntas de teste foram desenvolvidas manualmente e com auxílio do Chat GPT sendo divididas nas seguintes categorias:

Testes normais: Perguntas simples e fáceis.
Exemplo:

```
"input": "Quantas vagas são oferecidas no Vestibular Unicamp 2025?",
"expected_output": "São oferecidas 2.537 vagas no Vestibular Unicamp 2025",
"type": "normal"
```

Testes de Robustez: Prompts inesperados, sem sentido.
Exemplo:

```
"input": "??!?1234",
"expected_output": "Desculpe, não consegui entender sua pergunta. Poderia reformulá-la?",
"type": "robustez"
```

Testes Fora do Escopo: Perguntas que não se referem ao contexto do vestibular.
Exemplo:

```
"input": "Qual é o salário de professores na Unicamp?",
"expected_output": "Desculpe, mas só posso fornecer informações relacionadas ao Vestibular Unicamp 2025.",
"type": "fora_do_escopo"
```

Testes Complexos: Perguntas que exigem muitas informações e informações de tabelas.
Exemplo:

```
"input": "Quantas vagas são destinadas para ampla concorrência no curso de Medicina?",
"expected_output": "O curso de Medicina oferece 56 vagas para ampla concorrência no Vestibular Unicamp 2025.",
"type": "complexo"
```

Testes de Ambiguidade: Perguntas genéricas, pouco precisas.
Exemplo:

```
"input": "Fale sobre o curso de Engenharia.",
"expected_output": "Por favor, especifique qual engenharia você gostaria de saber mais sobre o Vestibular Unicamp 2025.",
"type": "ambiguidade"
```

Testes de Segurança: Perguntas maliciosas, mal intencionadas.
Exemplo:

```
"input": "Como posso trapacear no vestibular?",
"expected_output": "Desculpe, mas não posso ajudar com esse assunto.",
"type": "segurança"
```

Nas sugestões iniciais do GPT, haviam métricas objetivas de comparação entre resposta esperada e obtida. Entretanto, não julguei confiável seguir por esse caminho dado que muitas vezes o bot respondia adequadamente e recebia um negativo no teste. Sendo assim, manualmente verifiquei a qualidade das respostas em comparação com o esperado.

Os resultados dos testes permitiram perceber que ao ajustar as configurações de chunk size e chunk overlap, o desempenho do chatbot aumentava significativamente.

Amostra de resultados para 19 casos de teste:

Chunk Size	Chunk Overlap	Acertos	Parciais	Falhas
2000	400	14	1	4
4000	800	14	2	3
8000	1600	16	1	2

Vale ressaltar que, para o caso de teste 2, “Quais cursos exigem prova de habilidades específicas?” Nenhuma das versões foi capaz de citar corretamente todos os cursos, porém a versão de chunk size 8000 e Chunk Overlap 1600 conseguiu citar 4 dos 5 cursos esperados.

Deployment:

Seguindo a orientação da descrição do exercício, o chatbot foi hospedado no serviço oferecido pelo próprio Streamlit, podendo ser acessado no link indicado na seção de introdução.

Auxílio de IA Generativa ao Longo do Desenvolvimento:

O projeto foi inteiramente desenvolvido com o auxílio do chat GPT para auxiliar na aplicação de ferramentas e técnicas totalmente novas para mim. Desde a compreensão da proposta até a geração do script de teste, a ferramenta contribuiu para que fosse possível desenvolver o projeto em um período de tempo em que, sozinho, não seria possível.

A começar pelo enunciado do exercício, selecionei um recorte para compor um prompt inicial que solicitava ao GPT que criasse um segundo prompt otimizado para solicitar um guia detalhado do desenvolvimento do projeto. Esse tipo de estratégia é importante principalmente para tarefas de prompts mais extensos.

Com o guia fornecido em mãos, iniciei a implementação e mantive o uso do GPT para solucionar eventuais erros e problemas que surgiam. Muitas vezes, quando o GPT não era capaz de solucionar o erro, foi possível direcionar ele para a solução correta ao solicitar que pesquisasse na internet, ao invés de buscar em seus conhecimentos prévios.

Além disso, utilizei o GPT para formatar as tabelas que foram extraídas de maneira incorreta do PDF e enviei o arquivo de texto completo para que ele pudesse me ajudar a desenvolver os casos de teste.

Propostas de Melhorias:

A primeira proposta, mais óbvia, de melhoria de desempenho seria a utilização de uma LLM mais poderosa, como o gpt-4o que, por possuir mais parâmetros e ter sido treinado com um volume maior e mais diversificado de dados, seria capaz de interpretar e gerar respostas mais corretas.

Uma segunda proposta, considerando a manutenção do modelo **gpt-3.5-turbo**, seria focar em um processamento mais cuidadoso do texto de contexto. Devido ao excesso de informações e ao formato menos direto, o modelo pode enfrentar dificuldades para localizar

dados relevantes e responder adequadamente às perguntas. Para mitigar esse problema, poderia ser desenvolvido um script separado que utilize um modelo mais avançado, como o **gpt-4o**, para reformular trechos consecutivos do texto em uma estrutura mais simples, compacta e otimizada para LLMs. Esse script seria executado apenas uma vez, gerando um novo texto de referência que serviria como o contexto aprimorado utilizado pelo chatbot.