Relatório - **Iniciação Científica em NLP para Serviços de Saúde**
Etapa prática

Nome: Lucas Gomes Bussinger da Silva **RA:** 247314

Objetivo: Desenvolver um chatbot por meio de modelos GPTs para responder perguntas sobre o vestibular Unicamp.

Implementação do Projeto:

- Busca de Informações:
    - Para o chatbot funcionar eficientemente ele precisa ter uma base de dados, com isso, o primeiro passo do projeto foi buscar as informações a respeito da Resolução GR-031/2023, de 13/07/2023, no link https://www.pg.unicamp.br/norma/31594/0. Para isso foi criado o programa "**VestScrapper.py**", que faz um webscrapping de páginas da web, fazendo o download delas e salvando seu conteúdo em um arquivo texto, (neste caso em "**Vestibular_Unicamp_2024.txt**). Este download ocorre apenas de o arquivo texto ainda não existe no diretório.

- Construção do ChatBot:
    - Para simplificar a construção do ChatBot em si, os arquivos "**FuncoesChatBot.py**" e "**parametros.csv**" foram criados, eles servirão de "base" para a construção do bot.
    - Em "**FuncoesChatBot.py**" estão as funções responsáveis por definir a criação de cada parte do bot, com o uso do framework "**langchain**" em python, as principais funções são:
        - "**criar_llm**": Define a criação do modelo de large-language-model que será utilizado pelo bot, de acordo com os parâmtros em "**parametros.csv**".
        - "**documento_repartido**": Divide o ducumento de texto (**Vestibular_Unicamp_2024.txt)** em parte menores, que servirão para criar uma base de dados vetorizada que o bot utilizará.
        - "**vetorizar_documento":** cria uma base de dados vetorizada (de rápido acesso) a partir do documento repartido retornado pela função "**documento_repartido**",

para isso, foi definido um modelo de “**embedding**”, que serve para representar as sequências de caracteres por meios numéricos, o que facilitará a vetorização das partes do texto (base de dados efetiva a ser utilizada pelo bot). Para a vetorização foi utilizado a biblioteca FAISS (Facebook AI Similarity Search (Faiss)), que é efciente para a busca por similaridade (no caso deste ChatBot esta bilbioteca será responsável por pegar partes do texto que tenham a maior similaridade com a pergunta do usuário, e estas partes serão enviadas como contexto, junto à pergunta do usuário para o modelo **llm**)

- “**criar_prompt**”: Cria a estrutura de pergunta que será enviada para o modelo **llm,** inclusive acoplando o histórico do chat, para que o modelo lembre de perguntas feitas anteriormente pelo usuário (até 5 perguntas anteriores).
- “**criar_chain_recuperadora**”: Acopla a **llm, o prompt e o documento vetorizado,** possibilitando o envio à **llm**.

- Em “**parametros.csv**” residem os parâmetros escolhidos para o modelo llm (large language model) e para a busca de informações:

  - **Temperatura:** valor de **0 a 1**, define o quanto o modelo llm fará deduções a partir das informações dadas: com o valor **0** o modelo não fará dedução alguma, e apenas reportará o que está explicitamente escrito na base de dados. Com o valor **1** o modelo deduz exageradamente, podendo até alucinar (retornar informações não conexas com a base de dados)
  - **max_tokens:** O máximo de tokens (certa representação de grupos de caracteres) que são enviados e retornados pela chamada do modelo a partir da API da OpenAi.
  - **overlap_arquivo:** O quanto de contexto será mantido entre diferentes pedaços do arquivo texto (serve para uma parte não ficar completamente desconexa semanticamente da outra)
  - **nome_modelo_llm:** indica o modelo llm da OpenAi que será utilizado, neste caso o “**gpt-4-turbo**”.
  - **web_page:** Página da web em que o ChatBot buscará informações.
  - **output_file:** Nome do arquivo em que a página web será baixada (em formato de texto).

- A Construção em si do ChatBot ocorre em “**ChatBot.py**”, em que acontece o webscrapping da pagina da web, execução das funções em “**FuncoesChatBot.py**” com base nos parâmtros de “**parametros.csv**” e comunicação com o site (feito com streamlit do python, em “**site.py**”) a partir dos arquivos no diretório “**files_apoio**”

- Construção do site:
  - Para construir o site foi utilizado o framework “**streamlit**” co python, que permite um deployment gratuito do projeto.

- Resultados da avaliação do ChatBot:

  - Para a avaliação foi criado o programa “**avaliador.py**” e o a pasta “**avaliacao**”, dentro desta pasta estão localizados vários arquivos .txt da resposta da AI para as perguntas presentes em “**perguntas_respostas.json**”, para diferentes valores da **Temperatura** do modelo de large-language-model.

  - Aqui verifica-se que o modelo dá respostas mais completas e longas para valores mais altos, e respostas mais curtas e concisas para valores mais baixos, durante este processo percebeu-se também que o bot demorou bem mais para responder perguntas que exigissem uma resposta muito longa como "Quais cursos são oferecidos no vestibular Unicamp 2024?", o que é esperado, já que mais tokens precisam ser processados e mandados pela API. Após isso, um bom meio-termo para a interpretação da Resolução GR-031/2023 foi uma **Temperatura** de 0.55.

  - Outra Observação é que quando o histórico do chat não possui limites (ou seja, todas as perguntas e respostas são armazenadas, desde o início do chat) o ChatBot ia ficando progressivamente mais lento, já que a cada vez mais informação (tokens) ia sendo passada como contexto para a próxima pergunta, sobrecarregando cada vez mais o modelo e a API. Com isso, fez-se a escolha do ChatBot “lembrar” até 5 perguntas anteriores da pergunta atual.

- Dificuldades encontrada durante o desenvolvimento:
    - Achar uma boa temperatura para as respostas das perguntas.

    - Descobrir que o modelo llm em si não possui “memória”, e que para simular isso, a cada nova pergunta o histórico do chat deve ser passado como contexto toda vez, o que limita uma “memória” em conversações longas, ja que o ChatBot fica progressivamente mais lento em suas respostas, e gasta cadavez mais tokens (que possuem valor monetário atrelado a eles)

    - Arrumar problemas no servidor do Streamlit

- Como testar o modelo:
    - Remotamente:
        - Acessar o site https://lucas-bussinger-assistente-vestibular-unicamp-site-rdihl6.streamlit.app/ e enviar a mensagem “ativar: sua_api_key_da_OpenAi” para ativar o bot, a partir disso ele estará ativo e responderá às perguntas.

    - Localmente:
        - Caso não esteja sendo possivel rodar remotamente, fazer um clone do repositório Lucas-Bussinger/Assistente_Vestibular_Unicamp: Assiatente_Vestibular_Unicamp_2024 (github.com), ter os requisitos de “**requirements.txt**” instalados no computador (assim como python), abrir um terminal na pasta do repositório clonado e rodar o comando : **streamlit run site.py,** e então enviar a mensagem “ativar: sua_api_key_da_OpenAi” para ativar o bot, a partir disso ele estará ativo e responderá às perguntas.

        - Localmente é possivel alterar os parâmetros de “**parametros.csv**” livremente

- Ideia de otimização:
    - Outra abordagem para este projeto seria treinar um modelo com dados de várias perguntas e respostas esperadas, ou seja, fazer um **“fine tuning**” do modelo, deixando ele mais rápido e

preciso, mas isso demoraria muito tempo para fazer, muito mais que uma semana.
    - Outros testes interessantes a serem feitos seriam testes em outras páginas web, para ver como o modelo se saíria tratando de informação em diferentes formatos (tabelas por exemplo).

- Referências:
    - O aprendizado necessário para montar este ChatBot veio das seguintes fontes:
        - Documentação da OpenAI: API Reference - OpenAI API
        - Chat GPT: ChatGPT
        - Documentação do framework LangChain: Introduction | 🦜🔗 LangChain
        - Documentação do framework Streamlit: Streamlit documentation
        - Diversos vídeos no youtube.