UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE MATEMÁTICA, ESTATÍSTICA E COMPUTAÇÃO CIENTÍFICA
DEPARTAMENTO DE MATEMÁTICA APLICADA

RICARDO RAIMUNDO DA SILVA

Introdução ao processamento de linguagem natural: Desenvolvimento de um chatbot utilizando python

Campinas
28/06/2023

RICARDO RAIMUNDO DA SILVA

Introdução ao processamento de linguagem natural: Desenvolvimento de um chatbot utilizando python

Monografia apresentada ao Instituto de Matemática, Estatística e Computação Científica da Universidade Estadual de Campinas como parte dos requisitos para obtenção de créditos na disciplina Projeto Supervisionado, sob a orientação do(a) Prof. Ricardo Biloti.

Resumo

Esta monografia apresenta uma análise de uma solução de chatbot desenvolvida para atender alunos ingressantes na Universidade Estadual de Campinas (Unicamp). O elemento central, o modelo BERT, apresentou resultados promissores, associando questões com contextos de forma eficaz e gerando respostas precisas. No entanto, ele lutou com questões relacionadas ao jargão jurídico. Foram propostas melhorias de desempenho, incluindo uma lógica para categorizar questões e ampliar o banco de dados de contexto. Embora a solução demonstre bom desempenho e interação com o usuário, existem oportunidades de melhorias para aumentar a eficácia do chatbot para os alunos ingressantes da Unicamp.

Abstract

This monograph presents an analysis of a chatbot solution developed to assist incoming students at the State University of Campinas (Unicamp). The central element, the BERT model, showed promising results, effectively associating questions with contexts and generating accurate responses. However, it struggled with questions related to legal jargon. Performance improvements were proposed, including a logic to categorize questions and expand the context database. Although the solution demonstrates good performance and user interaction, opportunities for improvement exist to increase the chatbot's efficacy for Unicamp's incoming students.

Conteúdo

1 Introdução

A transição para a vida universitária é um momento crítico e muitas vezes desafiador para os estudantes ingressantes. A entrada na universidade envolve uma série de adaptações e aprendizados que vão além do conteúdo acadêmico. Os alunos frequentemente têm dúvidas sobre processos administrativos, serviços disponíveis, atividades acadêmicas, entre outros aspectos da vida universitária. Na Universidade Estadual de Campinas (Unicamp), como em muitas outras instituições de ensino superior, os estudantes ingressantes representam uma população diversificada com necessidades de informação variadas.

Neste contexto, o presente trabalho tem como objetivo desenvolver e implementar uma solução de Chatbot utilizando Python para Processamento de Linguagem Natural (NLP) para auxiliar os alunos ingressantes da Unicamp, respondendo automaticamente às suas perguntas. A solução proposta se baseia em tecnologias de ponta e práticas recomendadas nas áreas de NLP e desenvolvimento web.

1.1 Estrutura da Solução

A solução é composta por várias camadas. Primeiro, um modelo de linguagem BERT pré-treinado é usado, que passa por um ajuste fino com um conjunto de dados de perguntas e respostas relevantes para a Unicamp. Esse modelo é carregado em um servidor backend implementado usando FastAPI, um framework web para construção de APIs com Python. Além disso, um banco de dados não relacional é integrado para armazenar e gerenciar perguntas e respostas, permitindo o acesso rápido a informações relevantes. Finalmente, uma interface de usuário frontend é desenvolvida usando React, permitindo que os alunos enviem suas perguntas e recebam respostas em tempo real.

1.2 Estrutura da Monografia

Esta monografia está estruturada em nove capítulo, após esta introdução, o Capítulo 2 explana um pouco sobre o modelo BERT. O Capítulo 3 apresenta o desenho geral da implementação da solução. O Capítulo 4 aborda o treinamento e o ajuste fino do modelo BERT. O Capítulo 5 foca no desenvolvimento da API usando FastAPI. O Capítulo

6 discute a seleção e implementação do banco de dados não relacional. O Capítulo 7 descreve o desenvolvimento da interface de usuário frontend com React. O Capítulo 8 discute como a solução foi colocada em operação, e o Capítulo 9 apresenta uma análise e discussão dos resultados obtidos com a solução em operação.

2 Modelo BERT

O Bidirectional Encoder Representations from Transformers (BERT) é um modelo de processamento de linguagem natural que tem revolucionado a área. Desenvolvido e introduzido por pesquisadores do Google em 2018, o BERT é parte de uma nova geração de modelos de aprendizado profundo que exploram técnicas avançadas para entender a complexidade e a sutileza da linguagem humana.

Conforme uma análise da literatura científica realizada em 2020, em um período de pouco mais de um ano após sua introdução, o BERT estabeleceu-se como um componente central em experimentos de Processamento de Linguagem Natural (NLP), obtendo citações em mais de 150 publicações acadêmicas. Este número demonstra não apenas o impacto direto do BERT na comunidade de pesquisa em NLP, mas também reflete a rapidez com que este modelo inovador foi adotado.

Em sua implementação original, o BERT foi desenvolvido para a língua inglesa em duas variantes: o BERT BASE e o BERT LARGE. O BERT BASE é composto por 12 encoders com 12 cabeças de auto-atenção bidirecionais, somando um total de 110 milhões de parâmetros. Esta arquitetura de atenção múltipla permite ao modelo capturar complexas interdependências entre palavras e frases.

Por outro lado, o BERT LARGE é ainda mais robusto, apresentando 24 encoders e 16 cabeças de auto-atenção bidirecionais, totalizando 340 milhões de parâmetros. Isso dá ao BERT LARGE uma capacidade ainda maior para discernir contextos e nuances linguísticas.

Esses modelos foram treinados em dois grandes corpora: o Toronto BookCorpus, um conjunto de dados contendo 800 milhões de palavras, e a versão em inglês da Wikipédia, com impressionantes 2.500 milhões de palavras. Essa extensa base de treinamento contribui para a habilidade dos modelos BERT de compreender e gerar linguagem

natural de forma eficiente e coerente.

Nesta monografia, a solução que propomos é fundamentada no BERTimbau Base. Este é um modelo treinado especificamente para o Português Brasileiro, utilizando o brWaC Corpus, um extenso conjunto de dados da web que abrange mais de 2,7 bilhões de tokens. O uso deste modelo pré-treinado permite uma compreensão mais profunda e precisa do Português Brasileiro, e ajuda a lidar com a rica variedade e complexidade do nosso idioma.

2.1 Arquitetura do BERT

A arquitetura do BERT é baseada no Encoder do modelo Transformer. De forma simples, podemos entender a arquitetura do BERT como um empilhamento de blocos de encoder do Transformer. Para compreender a arquitetura do BERT, é importante entender o conceito de Transformers, que são uma arquitetura de rede neural projetada para lidar com sequências de dados, sendo particularmente eficaz para o processamento de linguagem natural.

Existem duas variantes principais do BERT: BERT Base e BERT Large. O BERT Base possui 12 camadas (L) de blocos de encoder do Transformer, tamanho de embedding de 768 (H) e 12 cabeças de atenção (A), totalizando aproximadamente 110 milhões de parâmetros. Por outro lado, o BERT Large é mais robusto, com 24 camadas (L), tamanho de embedding de 1024 (H) e 16 cabeças de atenção (A), somando cerca de 340 milhões de parâmetros.

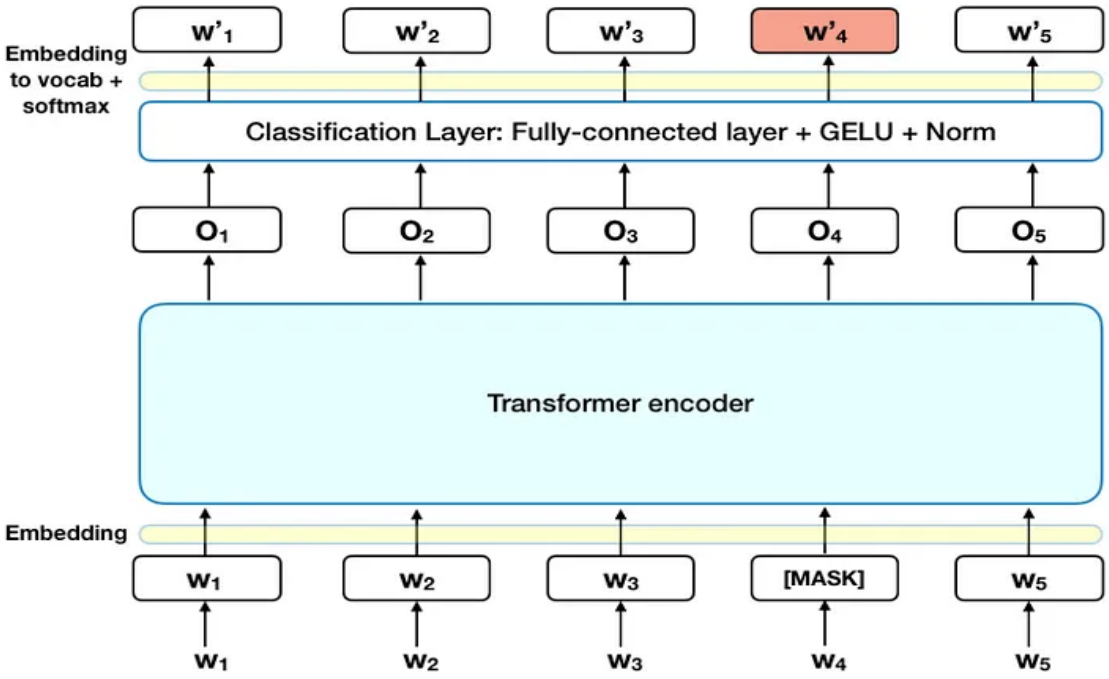

Figura 1: Diagrama representativo da arquitetura BERT.

2.2 Representação Bidirecional

Uma das características fundamentais do BERT é a sua capacidade de treinar representações bidirecionais. Isso significa que, ao contrário dos modelos de linguagem unidirecionais, que lêem os dados de entrada sequencialmente (da esquerda para a direita ou da direita para a esquerda), o BERT leva em consideração o contexto de ambos os lados de cada palavra dentro de uma sentença. Isso é fundamental para compreender o significado e o contexto das palavras.

Por exemplo, a palavra "banco"tem significados diferentes quando usada no contexto "fui ao banco sacar dinheiro"em comparação com "o banco do parque está quebrado". BERT é capaz de compreender esses contextos diferentes, pois analisa as palavras em ambas as direções.

No entanto, treinar um modelo bidirecional não é trivial. O uso de técnicas de treinamento padrão permitiria que cada palavra "visse a si mesma"e isso é problemático porque torna as previsões triviais. O BERT resolve isso usando uma técnica conhecida como Modelo de Linguagem Masked (MLM), que será discutido em detalhes na próxima seção.

2.3 Autoatenção

A autoatenção é um dos mecanismos fundamentais por trás da arquitetura do BERT. Em termos simples, a autoatenção permite que o modelo se concentre em diferentes palavras dentro da sequência de entrada. Por exemplo, em uma sentença como "O gato pulou sobre a cerca", a palavra "pulou"é mais relevante para a palavra "gato"do que "cerca". A autoatenção permite que o modelo associe palavras que estão em posições distantes na sequência, mas que são semanticamente relevantes entre si.

A autoatenção é alcançada através de um mecanismo de atenção de cabeçalho múltiplo, onde o modelo aprende representações de pal

avras em subespaços diferentes e combina essas representações. Isso é crucial para compreender o contexto e as dependências entre diferentes partes de uma sentença.

2.4 Modelo de Linguagem Masked (MLM)

O Modelo de Linguagem Masked (MLM) é uma técnica de treinamento inovadora utilizada pelo BERT para permitir o treinamento bidirecional. Durante o treinamento, algumas das palavras na sequência de entrada são mascaradas (ou ocultadas) e o objetivo é prever essas palavras mascaradas com base no contexto fornecido pelas palavras não mascaradas.

Por exemplo, considere a sentença "O gato pulou sobre a cerca". Durante o treinamento, esta sentença pode ser mascarada como "O gato ___ sobre a cerca"e o modelo precisa prever a palavra mascarada ("pulou") com base no contexto.

Esta abordagem é fundamental para permitir que o BERT treine representações bidirecionais. Como foi mencionado anteriormente, a representação bidirecional não pode ser alcançada com os métodos de treinamento padrão, pois permitiria que cada palavra "visse a si mesma", o que é problemático. O MLM resolve esse problema, permitindo que o modelo veja o contexto em ambas as direções, sem ver a palavra que está sendo prevista.

2.5 Ajuste Fino do BERT

O BERT é frequentemente usado como um modelo de linguagem pré-treinado que pode ser ajustado para tarefas específicas de processamento de linguagem natural. Isso é conhecido como ajuste fino do BERT. Durante o ajuste fino, os pesos do modelo pré-treinado são ligeiramente ajustados para se especializar em uma tarefa específica, como classificação de texto, análise de sentimento, ou resposta a perguntas.

O ajuste fino geralmente envolve adicionar uma camada de saída específica da tarefa ao modelo BERT e treinar o modelo em um conjunto de dados específico da tarefa. Durante esse treinamento, os pesos do modelo são atualizados em resposta aos dados da tarefa, enquanto os pesos das camadas pré-treinadas são ajustados em uma extensão menor.

Essa abordagem permite que o modelo aproveite o conhecimento de linguagem geral aprendido durante o pré-treinamento do BERT e se especialize nesse conhecimento para uma tarefa específica. Isso tem se mostrado extremamente eficaz em melhorar o desempenho em uma ampla gama de tarefas de processamento de linguagem natural.

2.6 Conclusão

O BERT representou um avanço significativo no campo do processamento de linguagem natural devido à sua capacidade de entender o contexto das palavras em sequências de texto. Sua arquitetura baseada em Transformer, o uso de representações bidirecionais através do Modelo de Linguagem Masked (MLM), e a capacidade de ajuste fino para tarefas específicas, tornaram-no um dos modelos mais poderosos e versáteis para NLP.

3 Desenho da Solução Implementada

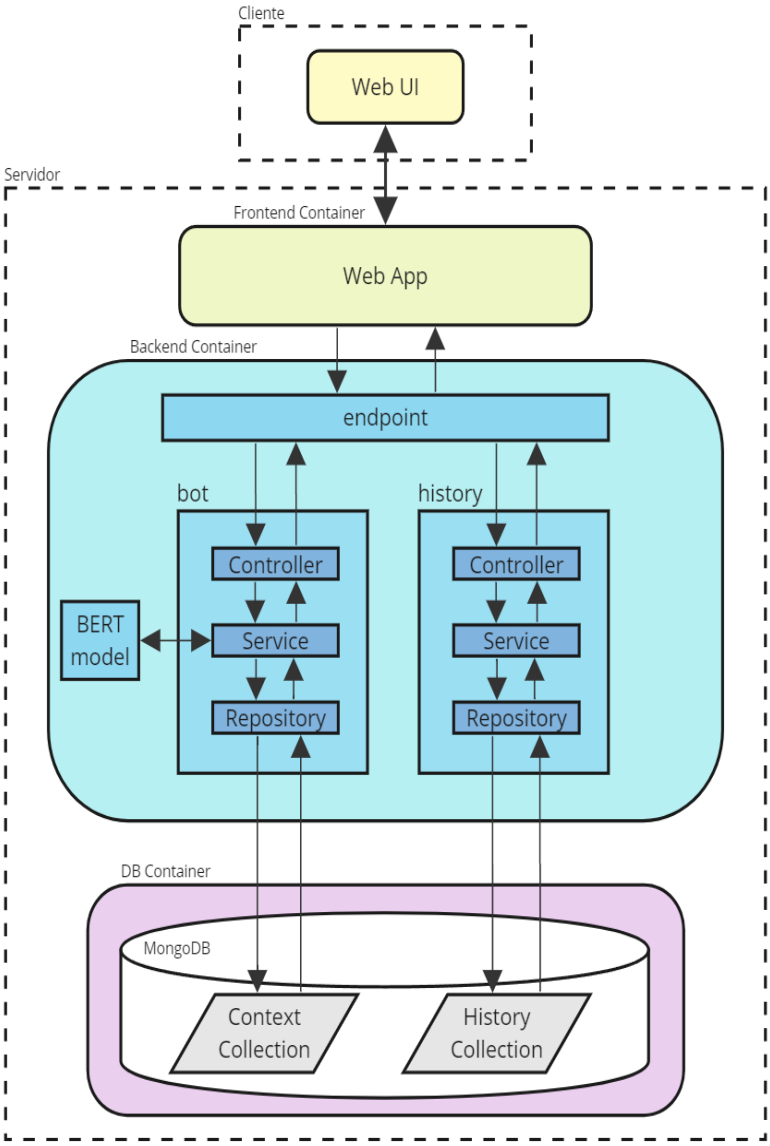

Figura 2: Arquitetura da solução.

Este capítulo mostra a arquitetura e o funcionamento de nossa solução de chatbot, que estarão hospedadas em containers Docker, a solução é composta por uma aplicação frontend desenvolvida em React JavaScript, uma API backend desenvolvida em Python usando o FastAPI e um banco de dados MongoDB. A solução também faz uso do modelo BERT pré-treinado para fornecer respostas adequadas aos usuários.

3.1 Frontend

O frontend da solução é uma aplicação web desenvolvida em JavaScript utilizando o framework React, que é hospedada em um container Docker. Essa aplicação tem a função de interagir com o usuário, coletando perguntas digitadas e exibindo respostas obtidas por meio de chamadas à API backend.

Quando o usuário insere uma pergunta, a aplicação React faz uma requisição HTTP POST à API backend, fornecendo a pergunta como dado da requisição. Após a chamada da API, a aplicação aguarda a resposta e, quando a obtém, a exibe na página web.

3.2 Backend

A API backend é desenvolvida em Python, utilizando o framework FastAPI. Ela é responsável por processar as solicitações vindas do front-end, consultar o banco de dados para buscar os contextos relevantes, utilizar o modelo BERT para gerar uma resposta apropriada e, finalmente, salvar as perguntas e respostas no histórico para futuras consultas.

A arquitetura FastAPI permite lidar com altas cargas de requisições HTTP de maneira eficiente, tornando-a uma opção adequada para a solução de chatbot. Quando a API recebe uma pergunta do frontend, ela primeiramente consulta o banco de dados MongoDB, em busca de contextos que podem ser relevantes para a pergunta.

Com os contextos em mãos, a API consulta o modelo BERT pré-treinado. O BERT, ou Bidirectional Encoder Representations from Transformers, é uma arquitetura de modelo de linguagem natural baseada em transformers que considera o contexto das palavras para produzir respostas. Assim, ele é capaz de gerar respostas relevantes à pergunta do usuário.

Após a geração da resposta, antes de retornar ao frontend, a API grava a pergunta e a resposta no banco de dados MongoDB para manter um histórico das interações do chatbot.

3.3 Database

O MongoDB é um banco de dados NoSQL orientado a documentos, que armazena os dados em estruturas tipo BSON, similares a JSON. Nesta solução, ele é hospedado em um container Docker separado e contém duas collections principais: 'contextos' e 'historico'.

A collection 'contextos' armazena os contextos que são usados para alimentar o modelo BERT, permitindo a geração de respostas mais relevantes às perguntas dos usuários. Quando a API recebe uma pergunta, ela consulta essa collection para obter os contextos relevantes.

A collection 'historico' registra as perguntas e respostas que foram processadas pelo chatbot. Após a geração de cada resposta, a API insere um novo documento nesta collection, registrando a pergunta do usuário, a resposta do chatbot e o timestamp da interação.

Em suma, a solução de chatbot proposta é uma arquitetura robusta e eficiente, capaz de gerar respostas relevantes e manter um registro de todas as interações para futuras análises.

4 Ajuste do Modelo (Fine-tuning)

O código do notebook utilizado para o ajuste fino do modelo, treinamento do modelo para responder a perguntas dado um contexto, se encontra no repositório:

github.com/p2o/chatbot-unicamp-training

Este notebook do Google Colab para a tarefa de fine-tuning do modelo BERT, utilizando o conjunto de dados SQuAD em português tem como objetivo desenvolver um sistema de perguntas e respostas eficaz com a capacidade de processar perguntas e contextos em português e fornecer respostas precisas.

O desenvolvimento é realizado em Python, com um amplo uso de bibliotecas de aprendizado profundo e processamento de linguagem natural, incluindo transformers, datasets e pytorch-xla. A primeira etapa envolve a configuração do ambiente de trabalho do Google Colab para acessar os recursos armazenados no Google Drive, e a instalação das bibliotecas necessárias.

Em seguida, um conjunto de variáveis é definido, especificando detalhes como a versão do conjunto de dados SQuAD a ser utilizada, o checkpoint do modelo BERT pré-treinado em português e o tamanho do lote de dados para treinamento.

O conjunto de dados SQuAD em português é então baixado e extraído. Os dados são processados e convertidos para o formato que o modelo de perguntas e respostas requer. Isso envolve a leitura dos arquivos JSON, a extração das informações relevantes e a gravação dos dados em um novo arquivo.

O módulo AutoTokenizer é usado para carregar o tokenizador pré-treinado que corresponde ao modelo BERT especificado. Este tokenizador é usado para tokenizar os dados de entrada antes de alimentá-los no modelo.

Uma vez preparados os dados, um processo de treinamento é iniciado utilizando um ambiente de processamento distribuído com a Unidade de Processamento Tensorial (TPU). Uma função de treinamento é definida e usada com a classe Trainer da biblioteca transformers para treinar o modelo. Uma série de parâmetros de treinamento são definidos, incluindo max_length e doc_stride, que controlam o comprimento máximo da entrada e a sobreposição entre partes do contexto, respectivamente.

O processo de treinamento inclui a iteração sobre o conjunto de dados de treinamento para preparar os exemplos de treinamento e mapear as respostas para as posições tokenizadas correspondentes. Uma vez treinado o modelo, ele é salvo juntamente com o tokenizador para uso futuro.

Para avaliar a eficácia do modelo treinado, um exemplo de código é fornecido que faz perguntas a partir de um contexto definido e imprime a resposta obtida pelo modelo, juntamente com a pontuação atribuída à resposta e as posições de início e fim da resposta no contexto original.

Finalmente, o modelo treinado é movido do ambiente de treinamento local para o Google Drive para armazenamento permanente. Isso é feito usando o módulo pathlib para manipulação de caminhos de arquivos.

5 Desenvolvimento da API (Backend Container)

O código da API desenvolvida se encontra no repositório:

github.com/p2o/chatbot-unicamp-backend

Este capítulo destina-se a detalhar o funcionamento e a arquitetura da nossa API backend, desenvolvida em Python com FastAPI, que é executada em um container Docker. Esta API é responsável por receber solicitações da página web, consultar o banco de dados para obter os contextos, consultar o modelo BERT pré-treinado para obter uma resposta à pergunta e salvar a pergunta e a resposta no banco de dados para fins históricos. A arquitetura da API é composta por dois módulos principais: o módulo 'bot' e o módulo 'history', que são estruturados em camadas: 'Controller', 'Service' e 'Repository'.

5.1 Execução em Docker

A API backend é encapsulada em um container Docker, o que garante isolamento, portabilidade e consistência em diferentes ambientes de execução. O Docker empacota a aplicação e suas dependências em um objeto de container binário que pode ser distribuído e executado em qualquer sistema que suporte Docker.

O arquivo 'Dockerfile' usado para construir a imagem Docker inclui todas as dependências necessárias para executar a aplicação, incluindo o interpretador Python, a biblioteca FastAPI, a biblioteca SQLAlchemy para interação com o banco de dados e a biblioteca Transformers da Hugging Face para interação com o modelo BERT.

5.2 Módulo Bot

O módulo 'bot' é responsável por receber a pergunta do usuário através do 'BotController', procurar o contexto relevante utilizando o 'BotService' e consultar o modelo BERT pré-treinado para gerar uma resposta utilizando o 'BotRepository'.

5.2.1 BotController

O 'BotController' é responsável por receber as solicitações HTTP da página web e encaminhá-las para o 'BotService'. Ele implementa rotas FastAPI que definem os pontos finais da API que a página web usa para fazer solicitações.

5.2.2 BotService

O ‘BotService‘ recebe a pergunta do usuário do ‘BotController‘ e faz a requisição ao ‘BotRepository‘ para obter o contexto relevante. Com a pergunta do usuário e o contexto obtido, ele consulta diretamente o modelo BERT pré-treinado para gerar a resposta.

5.2.3 BotRepository

O ‘BotRepository‘ é responsável pela interação com o banco de dados para buscar os contextos. Ele recebe a solicitação do ‘BotService‘, faz a consulta ao banco de dados para obter o contexto relevante e retorna esse contexto para o ‘BotService‘.

5.3 Módulo History

O módulo ‘history‘ é responsável por salvar a pergunta e a resposta no banco de dados para fins históricos. Ele consiste no ‘HistoryController‘, ‘HistoryService‘ e ‘HistoryRepository‘.

5.3.1 HistoryController

O ‘HistoryController‘ recebe a pergunta e a resposta do ‘BotController‘ e encaminha para o ‘HistoryService‘.

5.3.2 HistoryService

O ‘HistoryService‘ recebe a pergunta e a resposta do ‘HistoryController‘ e encaminha para o ‘HistoryRepository‘ para serem salvos no banco de dados.

5.3.3 HistoryRepository

O ‘HistoryRepository‘ é responsável pela interação com o banco de dados. Ele recebe a pergunta e a resposta do ‘HistoryService‘, salva no banco de dados e retorna a confirmação de que os dados foram salvos corretamente.

Em resumo, essa arquitetura permite uma separação clara de responsabilidades e facilita a manutenção e o escalonamento da aplicação.

6 Banco de Dados (DB Container)

O código para o banco de dados MongoDB se encontra no repositório: github.com/p2o/chatbot-unicamp-mongodb

7 Desenvolvimento do Frontend (Frontend Container)

O código para o frontend desenvolvido se encontra no repositório: github.com/p2o/chatbot-unicamp-frontend

Neste capítulo, descreveremos como o aplicativo front-end em JavaScript utilizando o framework React foi implementado. O aplicativo está contido em um container Docker e a sua principal função é permitir que o usuário digite uma pergunta que é enviada a uma API, recebendo a resposta para ser exibida no navegador.

7.1 Organização do Projeto

O código fonte do projeto está localizado na pasta 'src' no repositório 'chatbot-unicamp-frontend'. Esta pasta contém uma série de arquivos e diretórios que estruturam a aplicação. Segue uma breve descrição de alguns dos arquivos e diretórios principais:

- 'components': Este diretório contém componentes React reutilizáveis que compõem a interface do usuário. Cada componente é definido em seu próprio arquivo JavaScript (.jsx).

- 'App.css': Este arquivo contém os estilos CSS globais que são aplicados em toda a aplicação. CSS (Cascading Style Sheets) é uma linguagem usada para descrever a aparência e a formatação de um documento escrito em HTML.

- 'App.jsx': Este arquivo é o componente principal da aplicação React. Ele serve como um container que engloba todos os outros componentes da aplicação.

- 'main.jsx': Este arquivo é o ponto de entrada da aplicação, onde o componente principal 'App.jsx' é renderizado no DOM (Document Object Model).

7.2 Componente ‘PostQuestion‘

O componente ‘PostQuestion‘ é responsável por coletar a pergunta do usuário, enviar essa pergunta para a API e exibir a resposta recebida.

Inicialmente, o componente renderiza um formulário com um campo de entrada onde o usuário pode digitar sua pergunta. Ao submeter o formulário, a pergunta do usuário é enviada para a API através de uma requisição HTTP. Essa requisição é realizada usando a função ‘fetch‘ do JavaScript, que irá retornar com a resposta da API.

A resposta da API é então convertida em um objeto JavaScript usando o método ‘.json()‘ da resposta. O objeto resultante é armazenado no estado do componente React usando o método ‘setState‘, o que causa a re-renderização do componente.

Finalmente, o componente ‘PostQuestion‘ renderiza a resposta da API na página web. Isso é feito através de um elemento ‘$< p >$‘ que é preenchido com o campo de resposta do objeto de resposta da API.

7.3 Containerização com Docker

A aplicação foi encapsulada em um container Docker para facilitar a implantação e garantir a consistência do ambiente de execução em diferentes plataformas. Para fazer isso, criamos um ‘Dockerfile‘ que define o ambiente do container, incluindo o sistema operacional, as dependências do software e os comandos para iniciar o servidor.

Primeiro, o arquivo Dockerfile especifica a imagem base a ser usada (por exemplo, node:14), depois copia os arquivos do diretório de trabalho atual para o container. Após isso, o arquivo define um comando para instalar todas as dependências listadas no ‘package.json‘ e, em seguida, um comando para iniciar a aplicação.

O container Docker resultante pode ser facilmente implantado em qualquer sistema que tenha o Docker instalado, fornecendo uma maneira conveniente e eficiente de distribuir e executar a aplicação.

8 Análise e Discussão dos Resultados

Neste capítulo, faremos uma análise e discussão dos resultados obtidos com a solução de Chatbot desenvolvida para auxiliar os alunos ingressantes da Universidade Estadual de Campinas (Unicamp). Serão abordados diversos aspectos da solução, desde a arquitetura e funcionamento do modelo BERT até a interação do usuário com a interface web.

8.1 Avaliação do Modelo BERT

O modelo BERT é a peça central da solução de Chatbot, sendo responsável por gerar as respostas às perguntas dos usuários. Com base nos resultados obtidos durante o treinamento e ajuste fino do modelo, podemos avaliar sua eficácia.

Durante o treinamento, utilizamos o conjunto de dados SQuAD em português para a tarefa de perguntas e respostas. O modelo BERT foi capaz de aprender a associar perguntas aos contextos e gerar respostas precisas. No entanto, verificamos que o modelo não foi capaz de responder adequadamente a perguntas relacionadas a contextos com textos escritos em jargão jurídico.

Além disso, o modelo BERT treinado foi capaz de lidar com perguntas de diferentes temas e contextos, fornecendo respostas relevantes e coerentes. No entanto, isso ainda não foi suficiente para garantir a compreensão e geração de linguagem natural em contextos específicos de forma eficiente.

8.2 Performance da API

A API desenvolvida utilizando o framework FastAPI provou ser eficiente no processamento de requisições HTTP e no fornecimento de respostas em tempo real aos usuários. A arquitetura da API, dividida em módulos "bot"e "history", permitiu uma separação clara de responsabilidades e facilitaria a manutenção e o escalonamento da aplicação.

O módulo "bot"é responsável por receber as solicitações da página web, consultar o contexto relevante no banco de dados e consultar o modelo BERT pré-treinado para gerar a resposta. Uma abordagem possível para melhorar essa estrutura seria adi-

cionar uma lógica para categorizar as perguntas e assim buscar de forma mais precisa o contexto correto.

8.3 Interface do Usuário

A interface do usuário desenvolvida utilizando o framework React permitiu uma interação intuitiva e amigável com o Chatbot. Os usuários podem digitar suas perguntas em um campo de entrada e receber as respostas em tempo real.

O componente "PostQuestion" foi responsável por coletar a pergunta do usuário, enviar a requisição para a API e exibir a resposta recebida na página web. A integração entre a interface do usuário e a API foi suave e sem problemas, garantindo uma experiência de uso agradável para os usuários.

8.4 Futuras Melhorias

Embora a solução de Chatbot desenvolvida tenha alcançado resultados promissores, ainda existem algumas áreas que poderiam ser aprimoradas.

Uma melhoria possível seria expandir a base de dados de contextos para tornar o Chatbot ainda mais preciso e abrangente. Além disso, a incorporação de técnicas de aprendizado de reforço poderia ser explorada para melhorar ainda mais o desempenho do modelo BERT.

Outra melhoria seria a implementação de uma função de aprendizado contínuo, permitindo que o Chatbot aprenda com as interações dos usuários e melhore suas respostas ao longo do tempo.

Finalmente, a aplicação de técnicas de processamento de linguagem natural, como lematização e detecção de entidades, poderia melhorar a compreensão do contexto e a geração de respostas mais precisas.

Em suma, a solução desenvolvida demonstra um bom desempenho na resposta às perguntas dos usuários e na interação com a interface do usuário. No entanto, existem várias oportunidades de melhoria que podem ser exploradas no futuro para tornar o Chatbot cada vez mais eficaz e útil para os alunos ingressantes da Unicamp.