Universidade Federal do Rio de Janeiro
Instituto Alberto Luiz Coimbra de
Pós-Graduação e Pesquisa de Engenharia

Programa de Engenharia de Sistemas e Computação - PESC

CPS-769 - *Introduction to Artificial Intelligence and Generative Learning*

Professores:
Edmundo de Souza e Silva (PESC/COPPE/UFRJ)
Rosa M. Leão (PESC/COPPE/UFRJ)
Participação Especial: Gaspare Bruno (Diretor de Inovação, ANLIX)

Trabalho Final - Lista de Exercícios 06

Luiz Henrique Souza Caldas / Luis Paulo Albuquerque Guedes
lhscaldas@cos.ufrj.br / luis.guedes@coppe.ufrj.br

18 de setembro de 2024

1 Introdução

O presente trabalho tem como objetivo desenvolver uma aplicação que utiliza inteligência artificial para analisar a Qualidade de Experiência (QoE) em transmissões de vídeo. O foco é a criação de uma interface que aceite perguntas dos usuários em linguagem natural e forneça respostas, também em linguagem natural, baseadas nos dados de bitrate e latência. Para tal, a aplicação fará uso das APIs da OpenAI, implementando o raciocínio *Chain of Thought* (CoT) e *function calling* para processar os dados fornecidos, analisar a qualidade da transmissão e responder de maneira dinâmica às questões dos usuários.

O contexto do trabalho está relacionado aos desafios enfrentados por serviços de streaming de vídeo, nos quais a latência entre o servidor e o cliente, bem como a taxa de transmissão, são fatores críticos. A aplicação deverá integrar a análise dos dados de latência e bitrate para fornecer respostas a perguntas como "Qual cliente tem a pior qualidade de recepção de vídeo?"ou "Qual é a melhor estratégia de troca de servidor para otimizar a qualidade de experiência do cliente?". O trabalho exige a implementação de uma lógica de raciocínio em etapas (CoT) para decompor e resolver questões complexas, além da utilização de chamadas de função para realizar cálculos e normalizações necessárias no conjunto de dados.

Os dados fornecidos incluem medições de bitrate e latência, e o principal desafio reside na combinação dessas informações para calcular a QoE, identificar pontos críticos na transmissão e recomendar soluções para otimizar a experiência do usuário.

2 Conceitos Relevantes

2.1 Qualidade de Experiência (QoE)

A *Qualidade de Experiência* (QoE) refere-se à percepção subjetiva do usuário final sobre a qualidade de um serviço, como o streaming de vídeo, com base em fatores técnicos e não técnicos. A QoE é influenciada por parâmetros como a taxa de transmissão de dados (*bitrate*) e a latência (*Round-Trip Time* - RTT), que impactam diretamente a fluidez da transmissão e a qualidade visual. Uma alta QoE é alcançada quando o serviço proporciona uma experiência satisfatória ao usuário, com alta qualidade de vídeo e baixa interrupção, resultante de uma boa gestão de rede e otimização dos recursos de transmissão.

2.2 Chain of Thought (CoT)

O conceito de *Chain of Thought* (CoT) está relacionado ao processo de raciocínio utilizado por modelos de inteligência artificial para resolver problemas complexos de forma sequencial. No contexto deste trabalho, o CoT é implementado para analisar e responder perguntas dos usuários. O modelo divide a pergunta em etapas lógicas e interdependentes, processa cada uma dessas etapas e gera uma resposta clara e explicativa. Esse método permite que a aplicação lide com perguntas mais complexas ao seguir uma sequência estruturada de passos para chegar à resposta correta.

subsectionFunction Calling O *function calling* refere-se à capacidade de uma aplicação de chamar funções específicas durante o processamento dos dados, a fim de realizar cálculos ou operações. No caso deste trabalho, a aplicação faz uso de *function calling* para calcular a QoE e outras métricas de desempenho de vídeo, como a normalização dos dados de *bitrate* e *latência*, ou para determinar os melhores servidores para otimizar a transmissão. Ao chamar essas funções de forma dinâmica, a aplicação pode responder de maneira eficiente a perguntas específicas dos usuários com base nos dados fornecidos.

2.3 Normalização de Dados

A normalização de dados é um processo utilizado para ajustar os valores de diferentes variáveis a uma escala comum, facilitando a comparação e análise. Neste trabalho, a normalização *min-max* é utilizada para trazer os valores de *bitrate* e *latência* para uma faixa comum, garantindo que ambos tenham o mesmo peso no cálculo da QoE. Esse processo é fundamental para evitar que variáveis com escalas diferentes influenciem de maneira desproporcional os resultados finais da análise de qualidade.

2.4 Bitrate e Latência

O *bitrate* representa a taxa de transmissão de dados em uma rede, medido em kilobits por segundo (kbps). Ele define a qualidade e a quantidade de dados transmitidos por unidade de tempo. Já a *latência* (RTT) refere-se ao tempo que um pacote de dados leva para ser enviado do cliente ao servidor e de volta ao cliente, medido em milissegundos (ms). Ambos os parâmetros são essenciais para determinar a qualidade de uma transmissão de vídeo, influenciando diretamente a QoE.

3 Análise Exploratória dos Dados

Essa seção trata-se da Análise Exploratória de Dados (EDA), que visa explorar e entender a estrutura do conjunto de dados, oferecendo uma visão inicial de suas principais características. A EDA utiliza métodos estatísticos e gráficos para investigar a distribuição das variáveis, identificar *outliers*, analisar correlações e padrões. Por meio de ferramentas como histogramas, *boxplots*, gráficos de dispersão e matrizes de correlação, é possível obter uma visão geral dos dados e orientar as próximas etapas de modelagem e análise. Essa abordagem permite garantir que os dados estão adequadamente preparados para análises posteriores.

No trabalho em análise, o processamento adequado de dados é essencial para a qualidade da análise, seja ela em *streaming* de vídeo, transmissão de dados ou em outras áreas onde a latência e a taxa de transmissão desempenham papéis críticos.

Após a filtragem inicial dos dados, chegou-se aos seguintes *boxplots*:

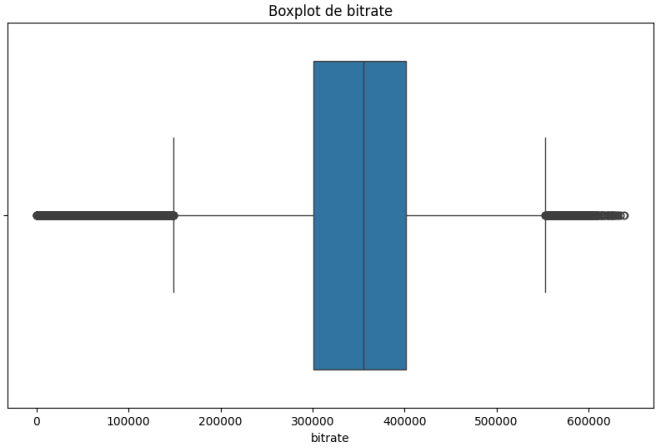

Figura 1: *Boxplot* de bitrate.

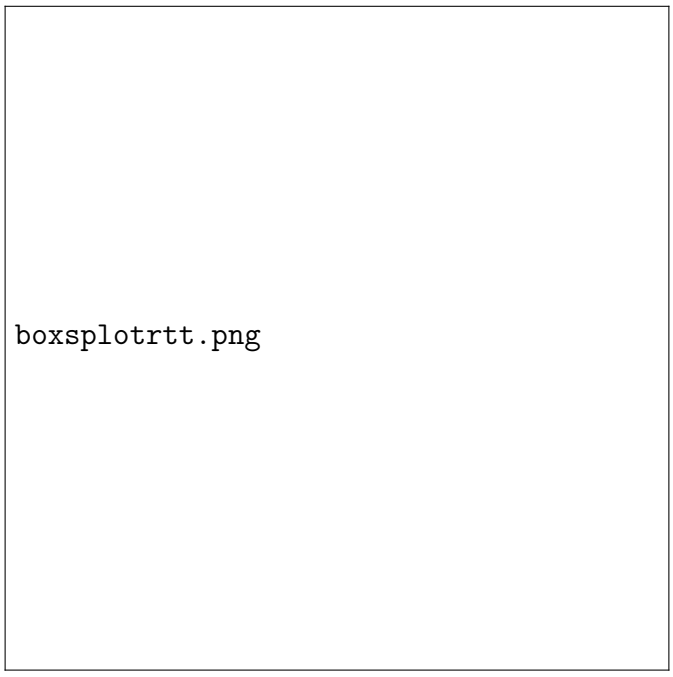

Figura 2: *Boxplot* de RTT.

Também foi possível analisar a distribuição de bitrate e de RTT ao longo do tempo, por meio de gráficos de haste:

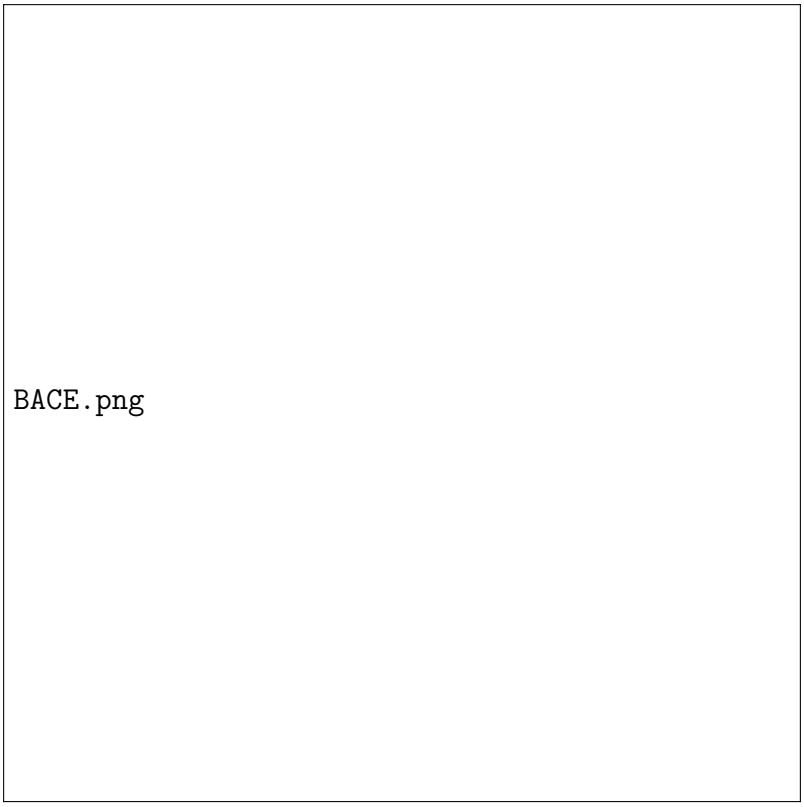

Figura 3: Distribuição de bitrate ao longo do tempo.

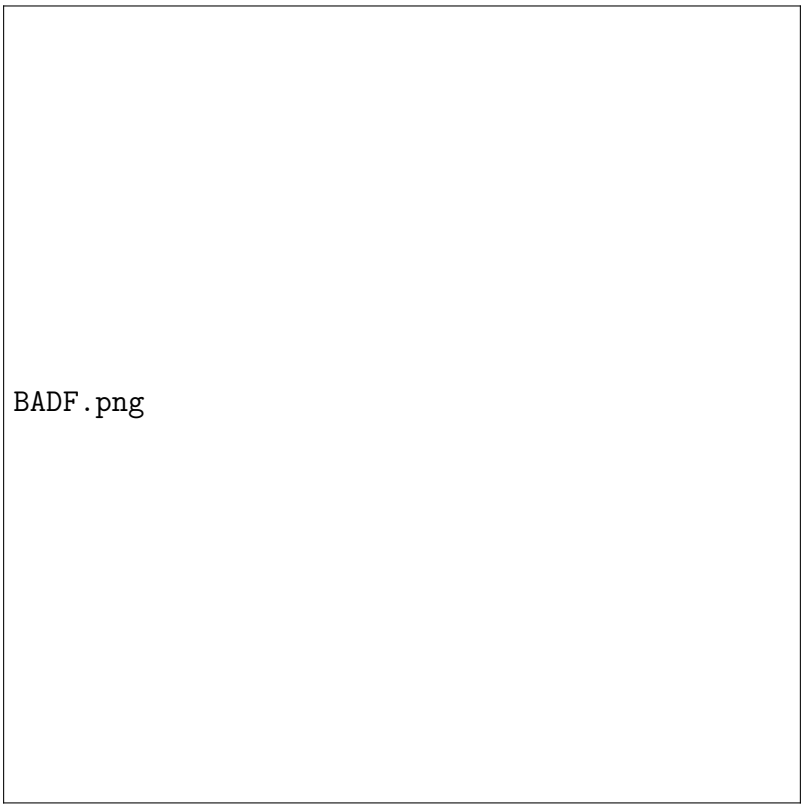

Figura 4: Distribuição de bitrate ao longo do tempo.

Figura 5: Distribuição de bitrate ao longo do tempo.

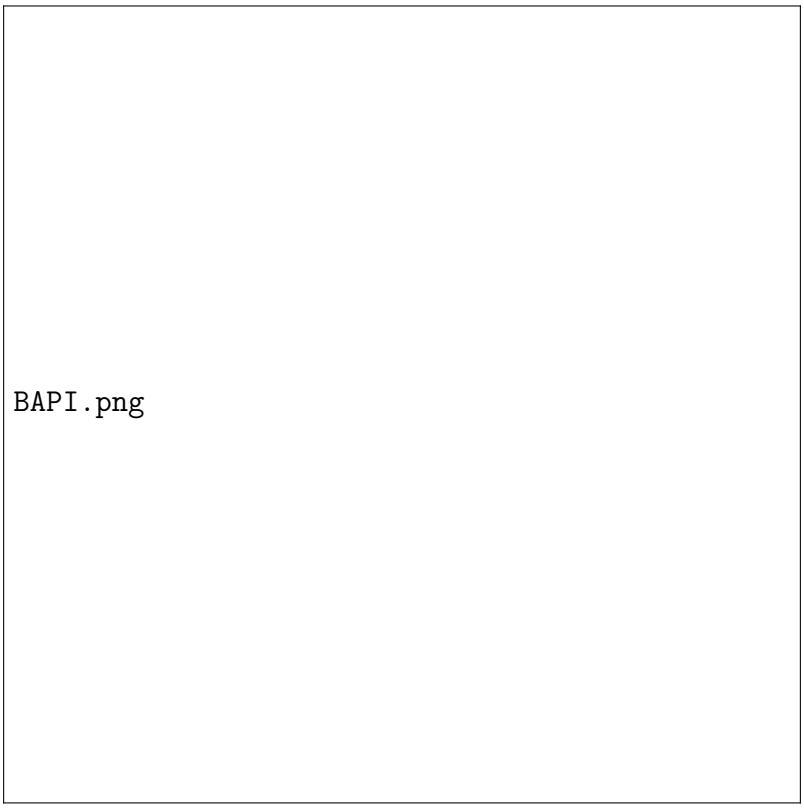

Figura 6: Distribuição de bitrate ao longo do tempo.

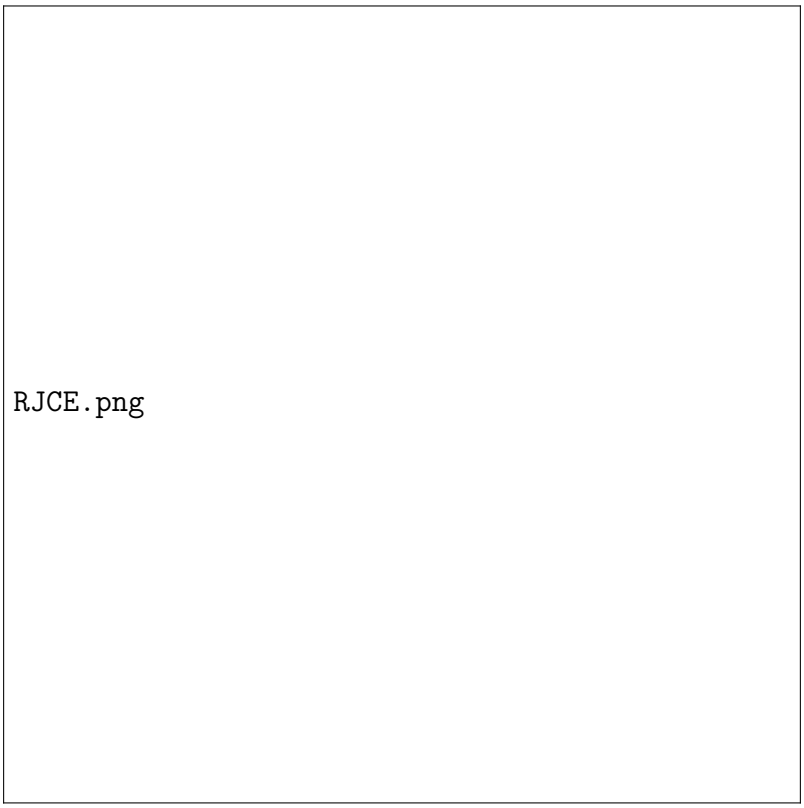

Figura 7: Distribuição de bitrate ao longo do tempo.

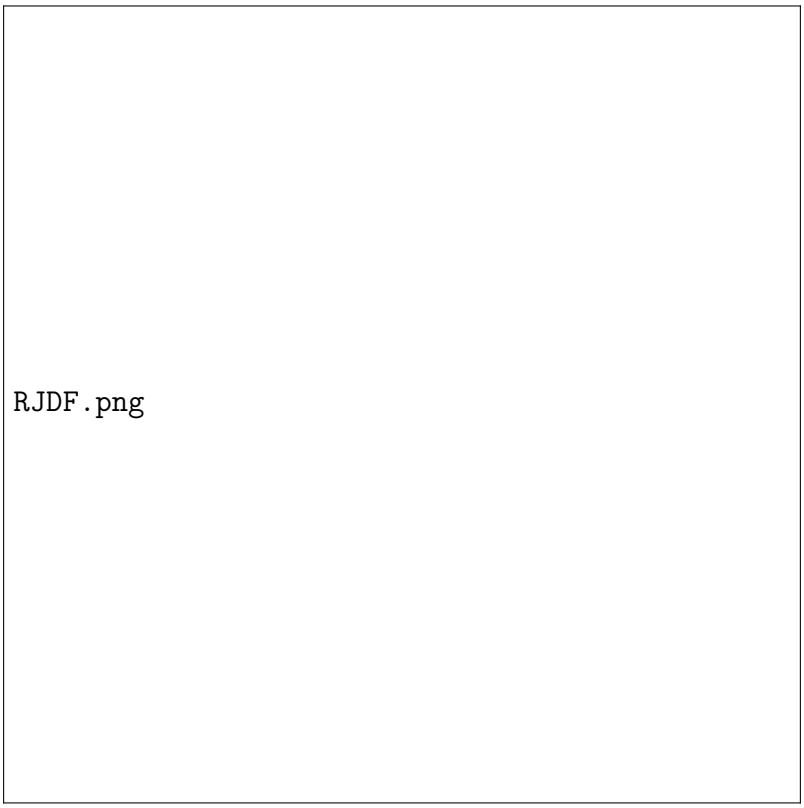

Figura 8: Distribuição de bitrate ao longo do tempo.

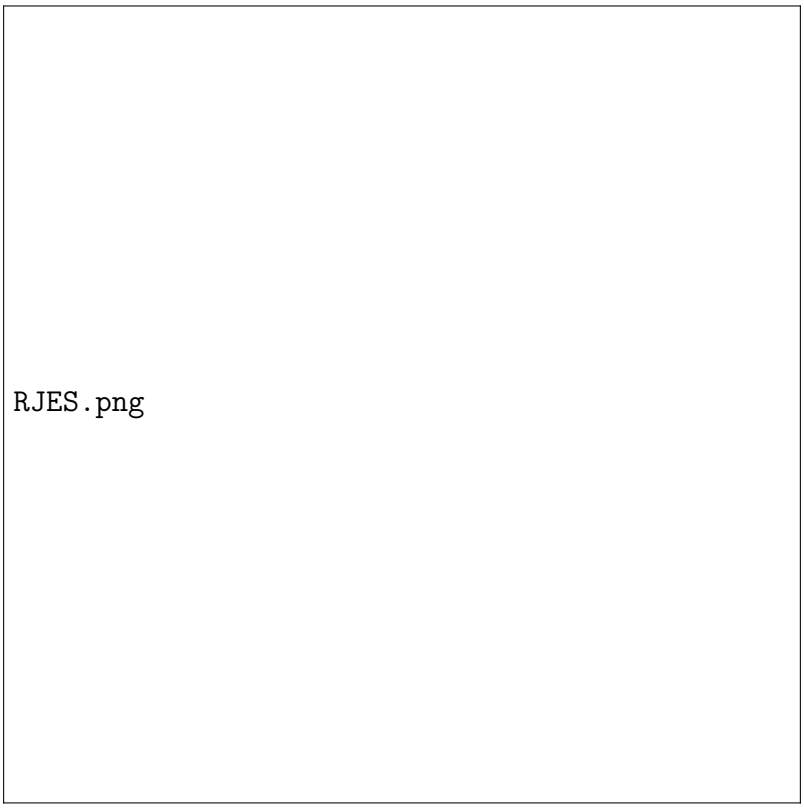

Figura 9: Distribuição de bitrate ao longo do tempo.

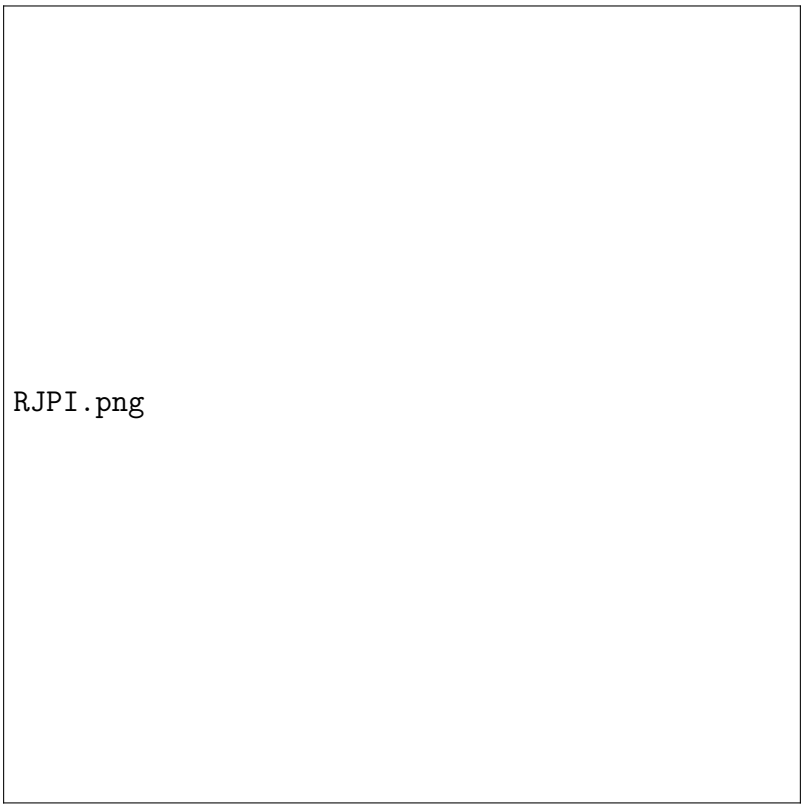

Figura 10: Distribuição de bitrate ao longo do tempo.

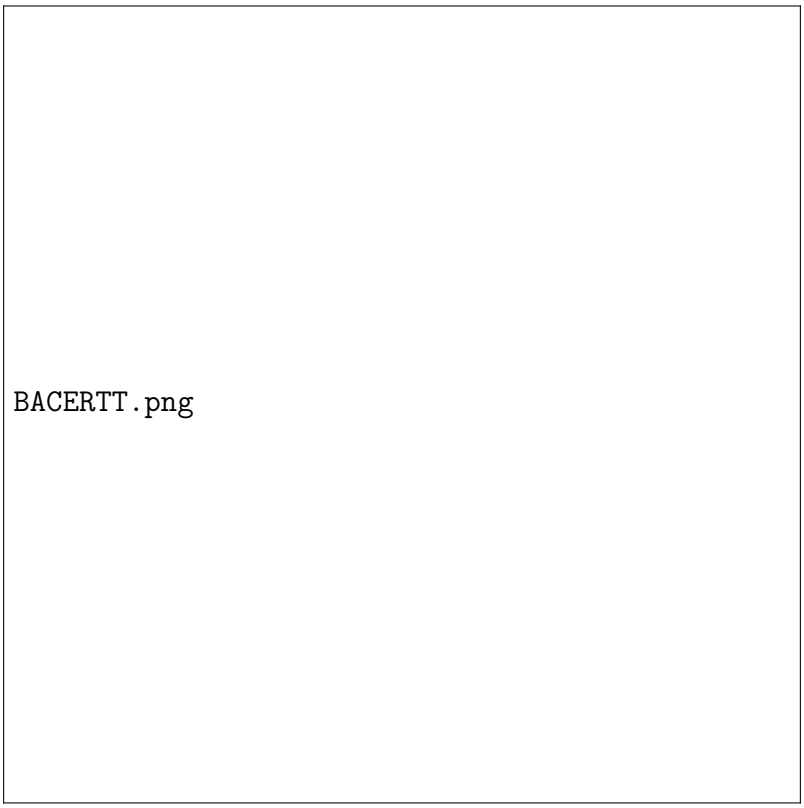

Figura 11: Distribuição de RTT ao longo do tempo.

Figura 12: Distribuição de RTT ao longo do tempo.

Figura 13: Distribuição de RTT ao longo do tempo.

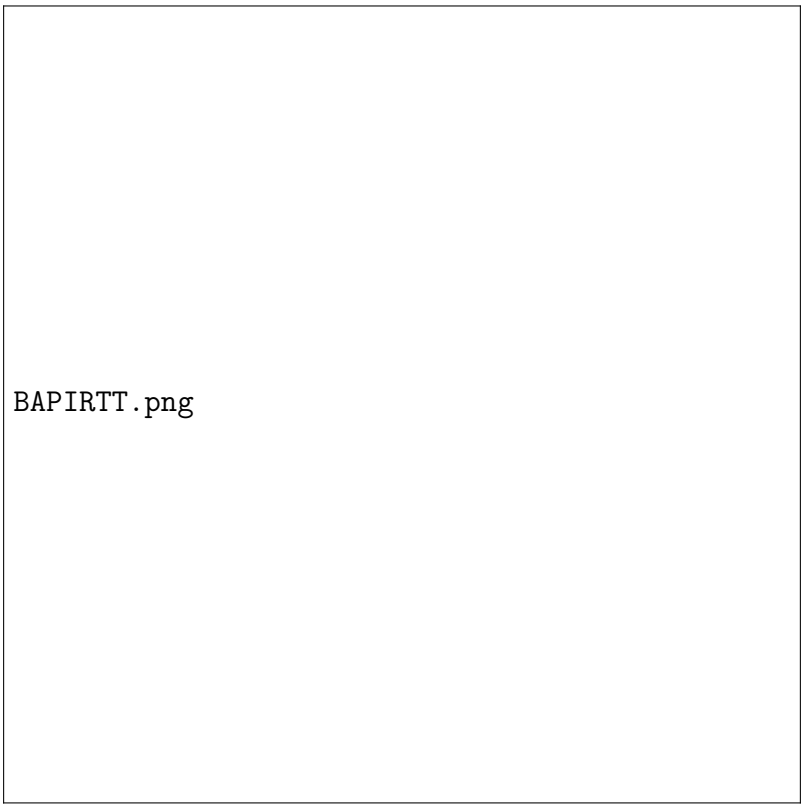

Figura 14: Distribuição de RTT ao longo do tempo.

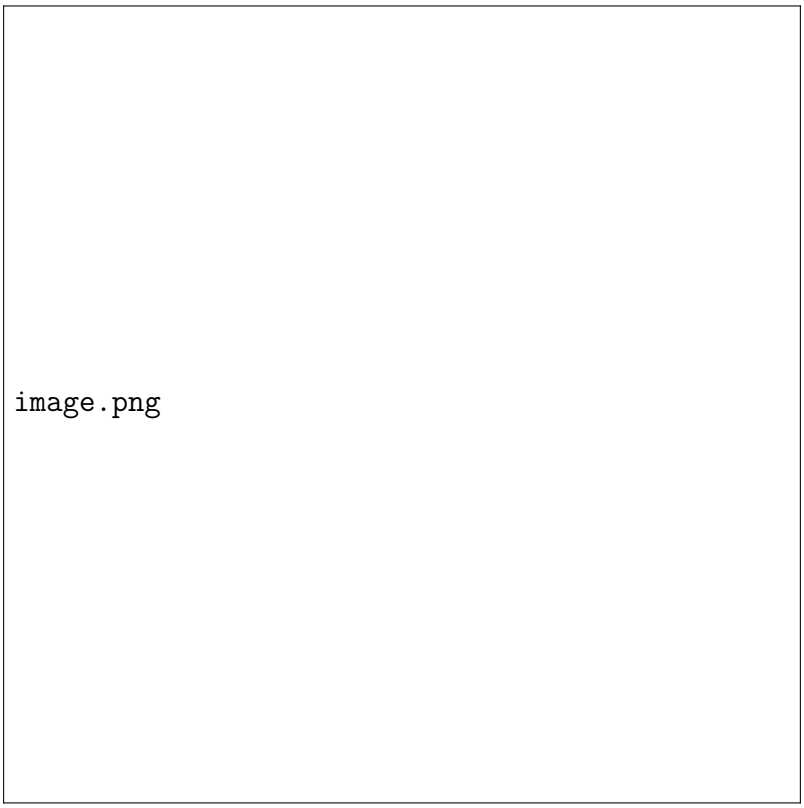

Figura 15: Distribuição de RTT ao longo do tempo.

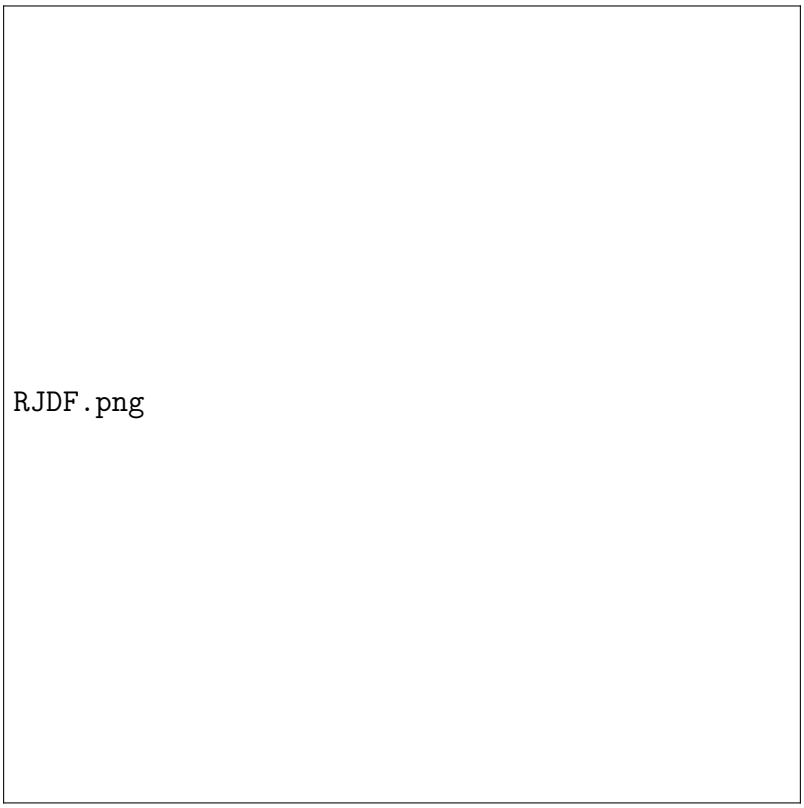

Figura 16: Distribuição de RTT ao longo do tempo.

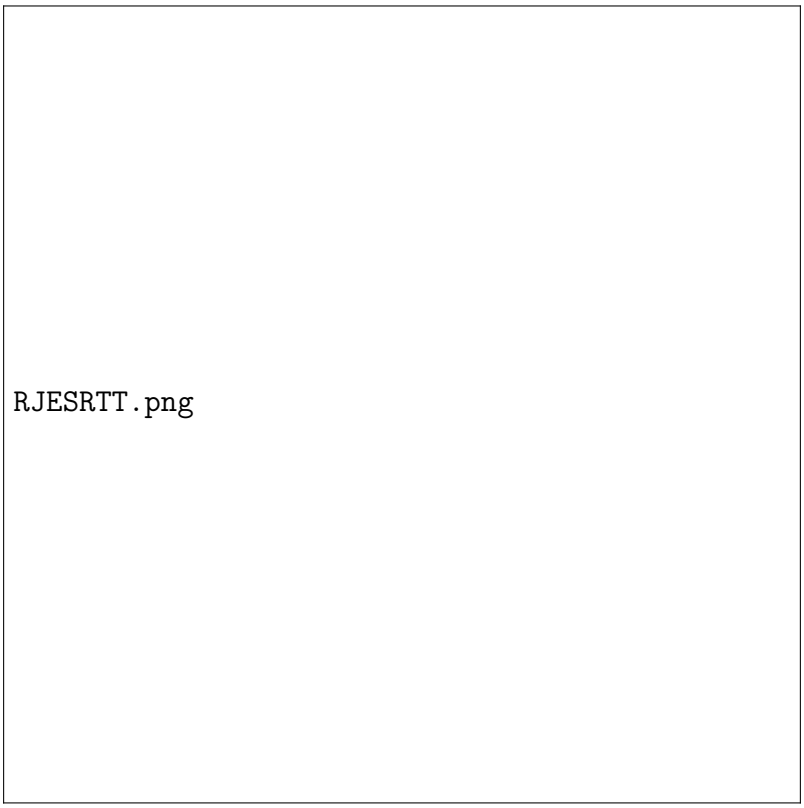

Figura 17: Distribuição de RTT ao longo do tempo.

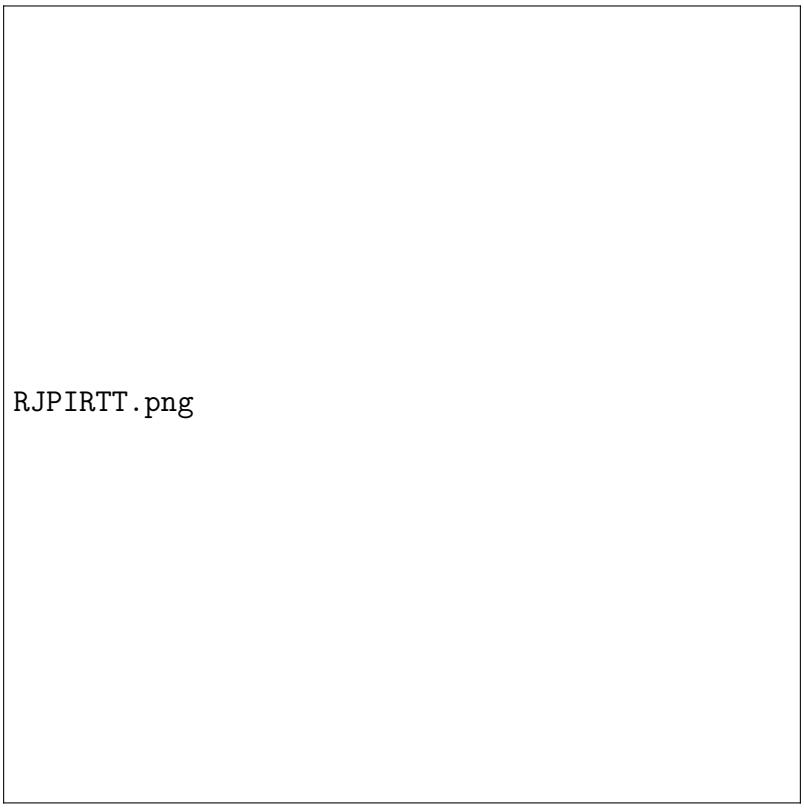

Figura 18: Distribuição de RTT ao longo do tempo.

4 Implementação

4.1 Estrutura do Sistema

O código principal utiliza o conceito de *Chain of Thoughts* (Cadeia de Pensamentos) em conjunto com a API da *OpenAI* e o framework *LangChain* para resolver perguntas complexas sobre Qualidade de Experiência (QoE) em uma rede de transmissão de vídeo. O *Chain of Thoughts* permite que o sistema divida o raciocínio em várias etapas lógicas, processando as informações de forma estruturada. A API da *OpenAI* é utilizada para interpretar a pergunta e gerar respostas em linguagem natural, enquanto o *LangChain* organiza o fluxo de processamento, conectando cada etapa de forma coerente para que a resposta seja precisa e relevante. O fluxograma pode ser visualizado na figura abaixo:

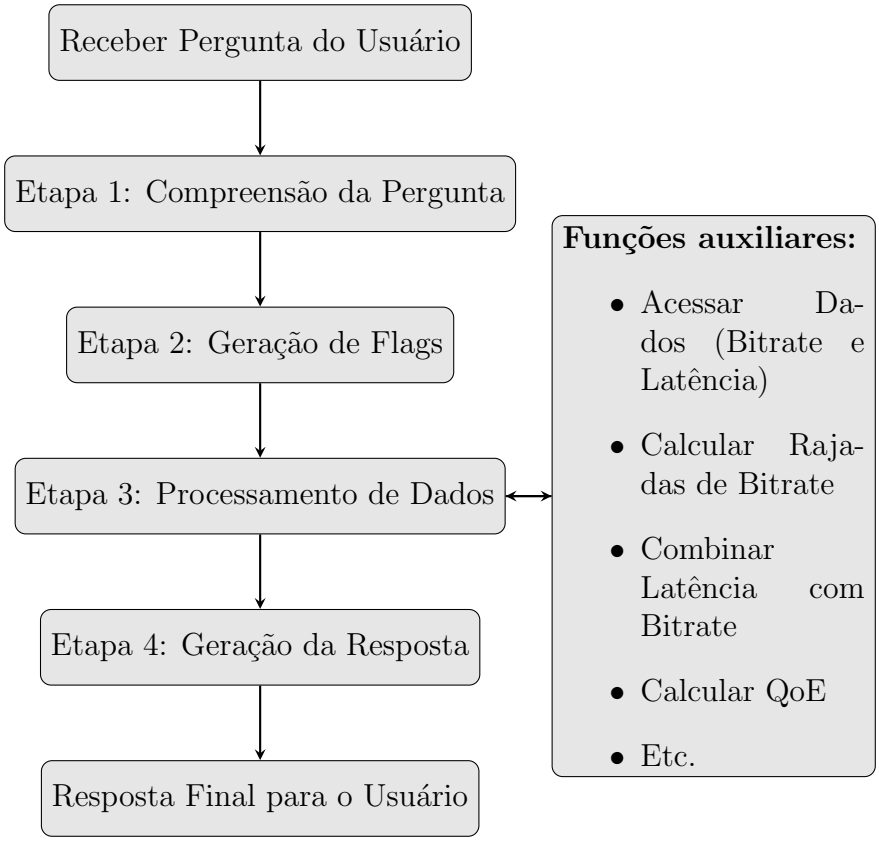

Figura 19: Diagrama de Blocos do Funcionamento do Sistema

A seguir são descritas de forma detalhada cada etapa do processo:

- **Primeira Etapa - Compreensão da Pergunta:** Nesta etapa, o sistema utiliza uma instância do modelo GPT-4o-mini para interpretar a pergunta do usuário em linguagem natural. O modelo é chamado com um prompt específico para decompor a pergunta em passos lógicos que guiam o processamento dos dados no contexto do banco de dados disponível. Essa instância do modelo foca em analisar a estrutura da pergunta e definir quais informações precisam ser extraídas ou calculadas. O uso do modelo aqui pode ser considerado como uma instância de **compreensão de linguagem natural**.

- **Segunda Etapa - Geração de Flags:** Nesta etapa, o *GPT-4o-mini* é utilizado novamente, mas agora com um prompt orientado para analisar os passos lógicos gerados no step anterior. O modelo converte esses passos em parâmetros e flags que serão usados para guiar o processamento dos dados. Cada flag indica uma ação específica a ser tomada, como filtrar por um período de tempo ou calcular uma métrica. Essa é uma nova instância do mesmo modelo, atuando como um **modelo de parametrização e extração de informações**. Entre as flags disponíveis estão:

  - `unrelated_to_db`: Indica se a pergunta não está relacionada aos dados do banco de dados.
  - `bitrate_burts`: Ativada quando a pergunta envolve o cálculo da média do bitrate dentro de rajadas de medições.
  - `latency_match`: Usada quando a pergunta requer a latência que coincide com as rajadas de bitrate.
  - `worst_qoe_client`: Sinaliza que o sistema deve identificar qual cliente teve a pior QoE.
  - `server_qoe_consistency`: Indica a necessidade de verificar qual servidor tem a QoE mais consistente.
  - `server_change_strategy`: Ativada para determinar a melhor estratégia de troca de servidor para otimizar a QoE.
  - `qoe_change`: Ativada para simular como mudanças de bitrate ou latência afetariam a QoE.

  Além dessas, há parâmetros como `client`, `server`, `datahora_inicio` e `datahora_final`, que filtram os dados de acordo com a pergunta.

- **Terceira Etapa - Processamento dos Dados:** A partir das flags geradas, o sistema consulta o banco de dados e realiza os cálculos necessários. Um dos principais desafios nesse processo é que o **bitrate** é medido em rajadas, enquanto as medições de **latência** são feitas de forma contínua, mas não sincronizadas com essas rajadas. Para calcular a Qualidade de Experiência (QoE), o sistema primeiro agrupa as medições de bitrate em rajadas, calculando a média do bitrate para cada uma delas. Isso é feito criando um **dataframe** com a média do bitrate dentro de um intervalo de até 5 segundos entre as medições, agrupando os dados que estão temporalmente próximos.

  Em seguida, o sistema precisa associar esses dados de bitrate com as medições de latência, que não ocorrem nos mesmos instantes que as rajadas de bitrate. Para resolver isso, o sistema cria uma segunda coluna no dataframe que contém a **média da latência** calculada em uma janela de 5 minutos centrada no **timestamp** de cada rajada de bitrate. Essa abordagem permite aproximar as medições de latência de forma a coincidir com os períodos das rajadas de bitrate, possibilitando o cálculo preciso da QoE.

- **Quarta Etapa - Geração da Resposta em Linguagem Natural:** Na última etapa, o sistema usa uma terceira instância do *GPT-4o-mini*, desta vez para gerar uma resposta em linguagem natural baseada nos resultados do processamento de dados. O modelo é orientado a construir uma resposta clara e compreensível para o usuário, utilizando os dados

processados como entrada. O prompt utilizado nesta instância orienta o modelo a focar na geração de texto coerente e relevante. Este uso do modelo é classificado como uma instância de **geração de texto**. Por exemplo, a resposta pode informar qual cliente teve a pior qualidade de recepção de vídeo ou como uma mudança na latência pode afetar a QoE.

4.2 Funções auxiliares

As funções auxiliares abaixo são utilizadas na terceira etapa do código principal para realizar tarefas específicas de processamento e manipulação dos dados extraídos do banco de dados. Elas facilitam o cálculo das rajadas de bitrate, a correspondência com as medições de latência e a normalização dos dados para o cálculo da Qualidade de Experiência (QoE), além de outras operações importantes para garantir que os dados sejam corretamente preparados e organizados antes de gerar a resposta final.

- **Função** `salvar_dataframes_em_txt` **(utilizada apenas para debug):** Esta função salva uma lista de `DataFrames` em sequência em um arquivo de texto. Ela é útil para armazenar e visualizar os resultados de diferentes etapas do processamento de dados, como as medições filtradas ou calculadas na etapa 3.

- **Função** `salvar_variaveis_em_txt` **(utilizada apenas para debug):** Similar à função anterior, esta função salva uma lista de variáveis (que podem ser números ou strings) em um arquivo de texto, permitindo registrar e consultar os valores calculados ao longo do processamento.

- **Função** `aux_get_dataframes_from_db`**:** Esta função é responsável por extrair os dados das tabelas `bitrate_train` e `rtt_train` do banco de dados `trabalho_raw.db` e convertê-los em `DataFrames` para serem manipulados durante o processamento.

- **Função** `aux_convert_datahora_to_timestamp`**:** Converte strings de data e hora no formato `YYYY-MM-DD HH:MM:SS` para o formato `timestamp` Unix, que é usado para filtrar e organizar os dados com base no tempo.

- **Função** `aux_convert_timestamp_to_datahora`**:** Realiza o processo inverso da função anterior, convertendo `timestamps` Unix de volta para strings de data e hora no formato legível, utilizado para exibir os resultados.

- **Função** `aux_filter_by_time`**:** Filtra as medições dentro de um intervalo de tempo específico, permitindo que o processamento dos dados se limite ao período relevante, com base em datas e horas fornecidas pelo usuário.

- **Função** `aux_calculate_bitrate_bursts`**:** Agrupa medições de bitrate que ocorrem em rajadas, separadas por no máximo 5 segundos. Em seguida, calcula o bitrate médio para cada rajada, permitindo identificar padrões de transmissão ao longo do tempo para cada cliente e servidor.

- **Função** `aux_find_latency_for_bursts`**:** Associa medições de latência (rtt) com rajadas de bitrate. Para cada rajada de bitrate, ela busca os dados de latência que estão dentro de uma janela de tempo em torno da rajada, e calcula a latência média correspondente.

- **Função `aux_adicionar_normalizacao`:** Normaliza os valores de bitrate e latência para garantir que os dados sejam comparáveis, permitindo que a QoE seja calculada de forma mais precisa, independentemente de variações extremas nos valores.
- **Função `aux_calcular_qoe`:** Calcula a Qualidade de Experiência (QoE) com base nos valores normalizados de bitrate e latência. A QoE é um indicador crucial para medir a qualidade de recepção de vídeo para um cliente.
- **Função `aux_simular_qoe_com_aumento_latencia`:** Simula como a QoE de um cliente seria afetada se a latência (rtt) aumentasse em um determinado percentual, permitindo prever o impacto de variações nas condições de rede.

4.3 Interface

Para que o usuário pudesse interagir com a LLM (Large Language Model), foi implementada uma interface web simples através de uma aplicação Flask. A interface facilita o uso do modelo GPT-4o-mini, permitindo que os usuários façam perguntas relacionadas à Qualidade de Experiência (QoE) e obtenham respostas em linguagem natural. Abaixo, explico como essa interface funciona, com base no código fornecido.

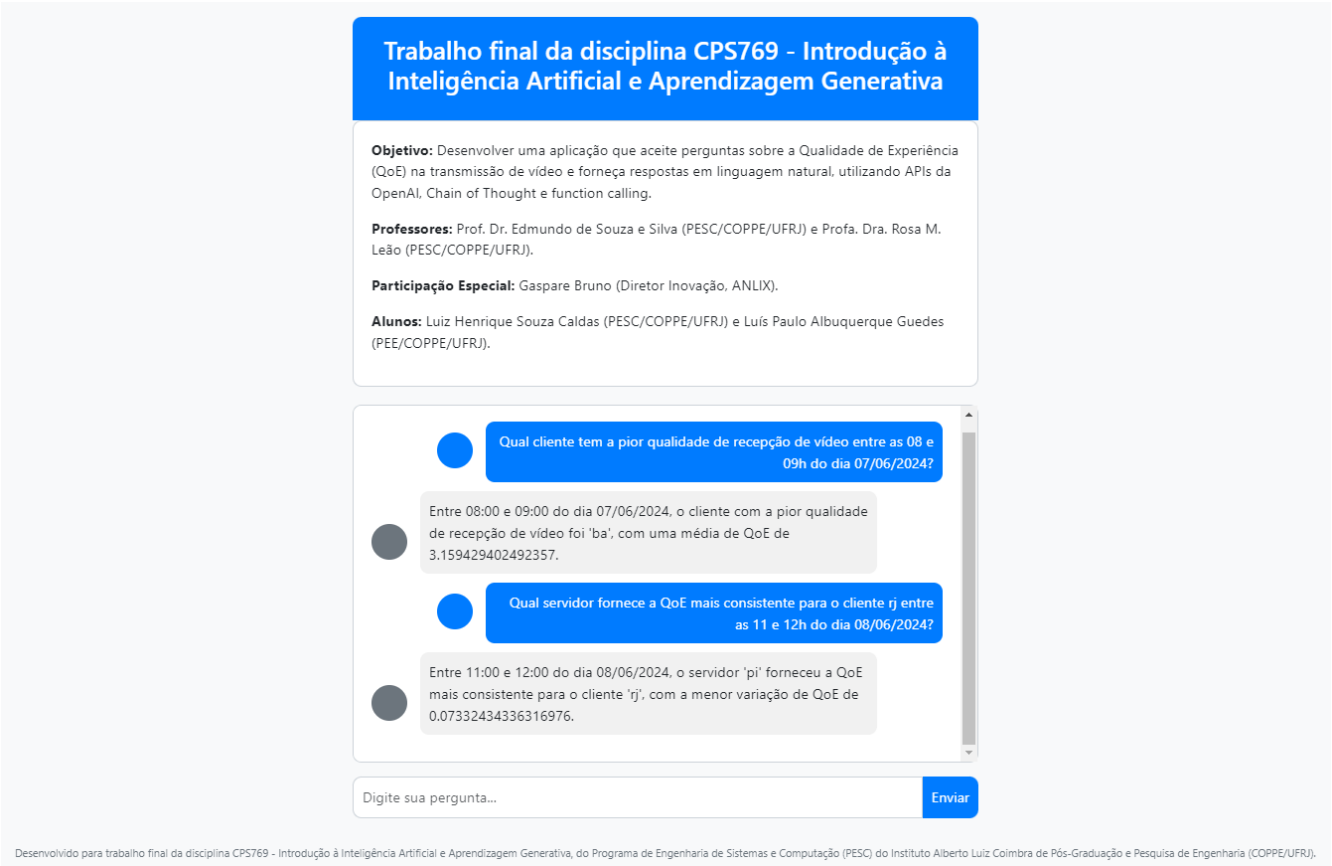

Figura 20: Interface para o usuário interagir com a LLM

4.3.1 Estrutura da Interface

- **app.py (Back-end)**: O arquivo `app.py` implementa o servidor Flask, que serve a página web e processa as interações do usuário. O servidor possui duas rotas principais:
  - /: Carrega a página HTML que exibe a interface do chatbot.
  - **/chat**: Recebe as perguntas enviadas pelo usuário e utiliza a função `responder_pergunta` (definida no arquivo `llm_model`) para gerar a resposta. A resposta é enviada de volta para a interface no formato JSON.

  Esse back-end faz a ponte entre a interface do usuário e o modelo de linguagem, processando as perguntas e retornando as respostas.

- **index.html (Front-end)**: O arquivo `index.html` define a interface gráfica do chatbot. Ele usa HTML, Bootstrap para o design responsivo, e JavaScript para controlar o envio de mensagens e a exibição das respostas. A página exibe uma caixa de chat onde o usuário digita sua pergunta e visualiza as respostas. Há também uma área com um resumo explicando o objetivo da aplicação, o curso, e os envolvidos no projeto.

- **chatbot.js (JavaScript)**: O JavaScript controla a lógica do envio de perguntas e recebimento de respostas. A função `sendMessage()` captura o texto que o usuário digitou, exibe a pergunta na interface e envia a mensagem para o servidor Flask usando uma requisição `POST` para a rota `/chat`. Assim que a resposta é recebida do servidor, a função `appendMessage()` exibe a resposta do chatbot na interface.

4.3.2 Funcionamento da Interface

Quando o usuário acessa a página, ele vê uma interface simples de chatbot, onde pode digitar perguntas. O sistema Flask no back-end processa as perguntas utilizando a LLM GPT-4o-mini. O JavaScript no front-end é responsável por capturar as interações do usuário e exibir as respostas dinamicamente na página.

Fluxo de Operação:

1. O usuário digita uma pergunta na caixa de entrada e clica no botão "Enviar".
2. A pergunta é enviada para o back-end Flask através de uma requisição `POST`.
3. No back-end, a pergunta é processada pela função `responder_pergunta`, que utiliza a LLM para gerar uma resposta.
4. A resposta é enviada de volta para o front-end, onde é exibida na caixa de chat.

Essa interface facilita o uso da LLM de maneira intuitiva, permitindo que qualquer usuário faça perguntas e receba respostas de forma interativa, sem precisar entender como o modelo funciona nos bastidores.

5 Resultados

6 Conslusão