

UNIVERSIDADE FEDERAL DE GOIÁS / INSTITUTO DE INFORMÁTICA

Geração Automática de Datasets

A partir de ligações de Call Center, para o Fine Tuning de LLMs

Alex Echeverria



UNIVERSIDADE FEDERAL DE GOIÁS (UFG)
INSTITUTO DE INFORMÁTICA (INF)

ALEX VICTOR ECHEVERRIA UGARTE SILVA

GERAÇÃO AUTOMÁTICA DE DATASETS

A partir de ligações de Call Center, para o Fine Tuning de LLMs

Goiânia
2024



UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA

TERMO DE CIÊNCIA E DE AUTORIZAÇÃO PARA DISPONIBILIZAR VERSÕES ELETRÔNICAS DE TRABALHO DE CONCLUSÃO DE CURSO DE GRADUAÇÃO NO REPOSITÓRIO INSTITUCIONAL DA UFG

Na qualidade de titular dos direitos de autor, autorizo a Universidade Federal de Goiás (UFG) a disponibilizar, gratuitamente, por meio do Repositório Institucional (RI/UFG), regulamentado pela Resolução CEPEC no 1240/2014, sem ressarcimento dos direitos autorais, de acordo com a Lei no 9.610/98, o documento conforme permissões assinaladas abaixo, para fins de leitura, impressão e/ou download, a título de divulgação da produção científica brasileira, a partir desta data.

O conteúdo dos Trabalhos de Conclusão dos Cursos de Graduação disponibilizado no RI/UFG é de responsabilidade exclusiva dos autores. Ao encaminhar(em) o produto final, o(s) autor(a)(es)(as) e o(a) orientador(a) firmam o compromisso de que o trabalho não contém nenhuma violação de quaisquer direitos autorais ou outro direito de terceiros.

1. Identificação do Trabalho de Conclusão de Curso de Graduação (TCCG)

Nome(s) completo(s) do(a)(s) autor(a)(es)(as): **ALEX VICTOR ECHEVERRIA UGARTE SILVA**

Título do trabalho:

GERAÇÃO AUTOMÁTICA DE DATASETS

A partir de ligações de Call Center, para o Fine Tuning de LLMs

2. Informações de acesso ao documento (este campo deve ser preenchido pelo orientador) Concorda com a liberação total do documento [X] SIM [] NÃO¹

[1] Neste caso o documento será embargado por até um ano a partir da data de defesa. Após esse período, a possível disponibilização ocorrerá apenas mediante: a) consulta ao(à)(s) autor(a)(es)(as) e ao(à) orientador(a); b) novo Termo de Ciência e de Autorização (TECA) assinado e inserido no arquivo do TCCG. O documento não será disponibilizado durante o período de embargo.

Casos de embargo:

- Solicitação de registro de patente;
- Submissão de artigo em revista científica;
- Publicação como capítulo de livro.

Obs.: Este termo deve ser assinado no SEI pelo orientador e pelo autor.



Documento assinado eletronicamente por **Alex Victor Echeverria Ugarte Silva**, **Discente**, em 16/02/2024, às 04:02, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do [Decreto nº 10.543, de 13 de novembro de 2020](#).



Documento assinado eletronicamente por **Fernando Marques Federson, Professor do Magistério Superior**, em 02/09/2024, às 11:22, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do [Decreto nº 10.543, de 13 de novembro de 2020](#).



A autenticidade deste documento pode ser conferida no site https://sei.ufg.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **4383118** e o código CRC **A1367E96**.

Referência: Processo nº 23070.008342/2024-25

SEI nº 4383118

ALEX VICTOR ECHEVERRIA UGARTE SILVA

GERAÇÃO AUTOMÁTICA DE DATASETS

A partir de ligações de Call Center, para o Fine Tuning de LLMs

Relatório final de Trabalho de Conclusão de Curso, apresentado à Universidade Federal de Goiás, como parte das exigências para a obtenção do título de Bacharel em Inteligência Artificial.

Orientador: Prof. Dr. Fernando Marques Federson

Goiânia

2024

Ficha de identificação da obra elaborada pelo autor, através do Programa de Geração Automática do Sistema de Bibliotecas da UFG.

SILVA, ALEX VICTOR ECHEVERRIA UGARTE
GERAÇÃO AUTOMÁTICA DE DATASETS [manuscrito] : A partir de ligações de Call Center, para o Fine Tuning de LLMs / ALEX VICTOR ECHEVERRIA UGARTE SILVA. - 2024.
203 f.

Orientador: Prof. Dr. FERNANDO MARQUES FEDERSON.
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal de Goiás, Instituto de Informática (INF), Inteligência Artificial, Goiânia, 2024.

1. inteligência artificial. 2. modelos grandes de linguagem. 3. geração automática de datasets. I. FEDERSON, FERNANDO MARQUES, orient. II. Título.

CDU 004

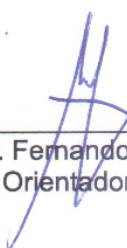
ALEX VICTOR ECHEVERRIA UGARTE SILVA

GERAÇÃO AUTOMÁTICA DE DATASETS


A partir de ligações de Call Center, para o Fine Tuning de LLMs

Relatório final de Trabalho de Conclusão de Curso, apresentado à Universidade Federal de Goiás, como parte das exigências para a obtenção do título de Bacharel em Inteligência Artificial.


Data da Aprovação: 08 de fevereiro de 2024.



Prof. Dr. Fernando Marques Federson
Orientador (INF-UFG)



Prof. Dr. Aldo André Díaz Salazar
Coordenador de TCC do BIA (INF-UFG)



Prof. Dr. Vinícius Sebba Patto
Coordenador do BIA (INF-UFG)



Prof. Dr. Iwens Gervasio Sene Junior
(INF-UFG)

ALEX VICTOR ECHEVERRIA UGARTE SILVA

GERAÇÃO AUTOMÁTICA DE DATASETS

A partir de ligações de Call Center, para o Fine Tuning de LLMs

RESUMO

Este Relatório de Conclusão de Curso tem como objetivo reunir os resultados da minha jornada para me tornar um especialista em **Geração Automática de Datasets (LLMs)**. Uma ilustração e sua narrativa descrevem os períodos de trabalho. Os Apêndices contêm os Termos de Aceite de Entrega e os resultados obtidos durante cada período de trabalho.

Palavras-chave: inteligência artificial, modelos grandes de linguagem, geração automática de datasets.

ABSTRACT

This Course Completion Report aims to bring together the results of my journey to become an expert in **Automatic Dataset Generation (LLMs)**. An illustration and its narrative describe the work periods. The Appendices contain the Delivery Acceptance Terms and the results obtained during each work period.

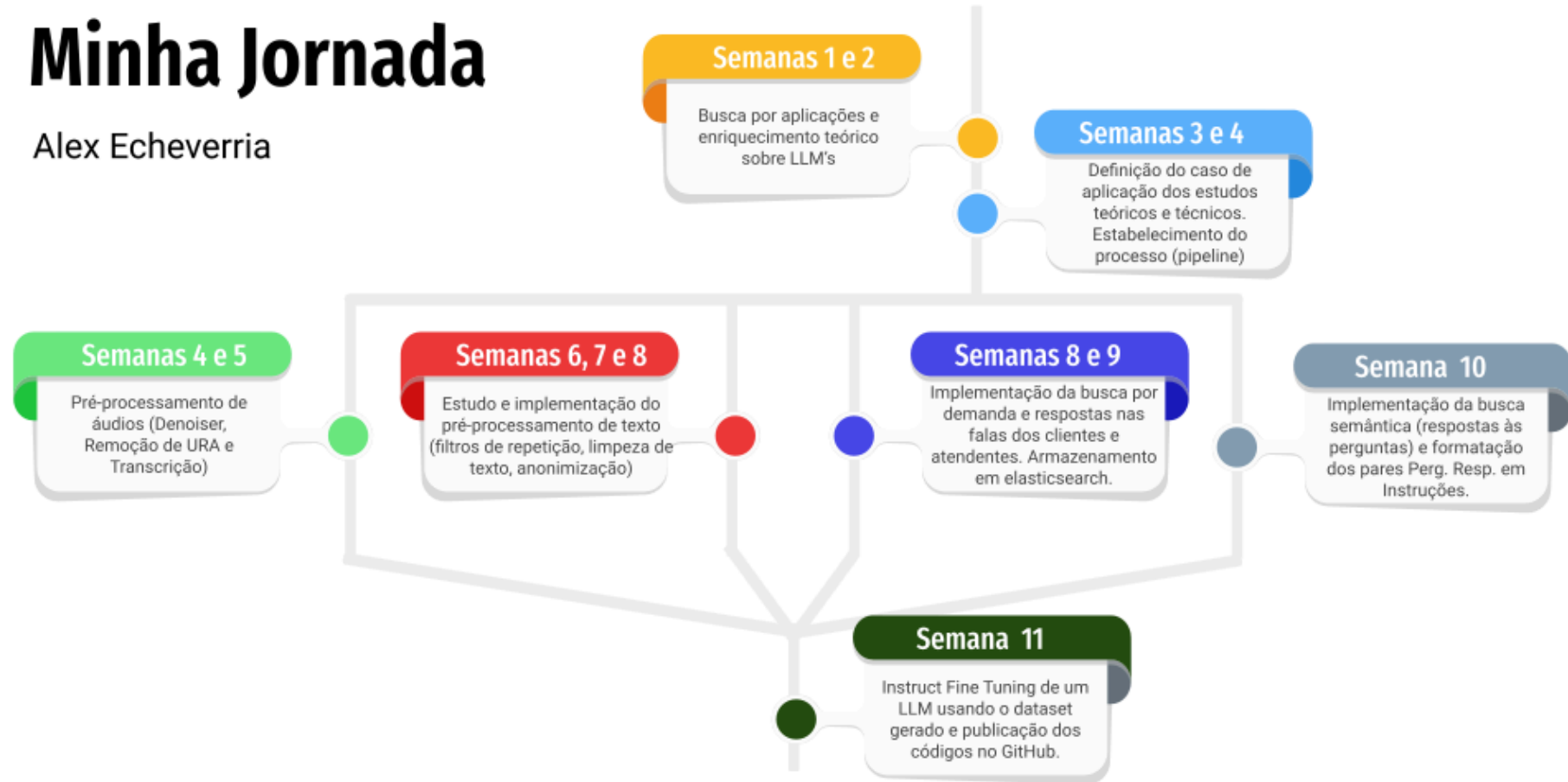
Keywords: artificial intelligence, large language models, automatic dataset generation.

Goiânia

2024

Minha Jornada

Alex Echeverria



Especialista em:
Geração Automática de Datasets (LLMs)

Slides feitos com [Slidesgo](#) e [Framit](#)

MINHA JORNADA

Nome: Alex Victor Echeverria Ugarte Silva

Especialidade: Geração Automática de Datasets (LLMs)

Objetivo deste documento

Durante a jornada empregada na disciplina Residência em IA¹, geraram-se diversos resultados devido ao Processo que fomos submetidos. Esse Processo consistiu na submissão semanal de um documento que visava formalizar tudo que fora desenvolvido ao longo da semana, sendo, então, avaliado por uma banca que considerava tanto o planejamento quanto a execução para o período. Este documento visa descrever os resultados obtidos, fazendo referência aos Termos de Aceite de Entrega e seus documentos associados, focando nos resultados.

Minha Jornada

Definição da aplicação:

Após a popularização da ferramenta ChatGPT a nível mundial, o interesse pelo assunto de Large Language Models (LLMs) avançou de forma significativa entre o meio corporativo e acadêmico. Por esse motivo, lancei mão em busca do aprofundamento nesse crescente tema e, então, tornar-me um especialista, a partir do Processo apresentado na Residência em Inteligência Artificial.

O começo do Processo foi com a **Semana 1**, em que a partir da seção *Scope & Topics* da Conferência Internacional em *Computational Science e Computational Intelligence* (CSCI), consegui definir o recorte necessário para a pesquisa de LLMs, o qual seria aplicações ao Mercado Financeiro. No entanto, após a **Semana 2** do processo ser concluída, senti necessidade de voltar à literatura e aprofundar o conhecimento sobre LLMs em si, para definir com mais propriedade a aplicação. Por tal motivo, buscou-se na literatura por uma revisão bibliográfica acerca do tema, quando foi possível encontrar o artigo A

¹ Dez semanas, entre setembro de 2023 e janeiro de 2024.

Survey of Large Language Models, de Zhao et al., o qual traz uma revisão de literatura aprofundada desde a definição de LLMs, até etapas fundamentais de seu treino e avaliação de resultados, cuja leitura permitiu, pois, a compreensão necessária para prosseguir com a especialização. No **Apêndice 1**, no documento "**Artigo: A Survey of Large Language Models**", encontram-se as principais características desse artigo e um breve detalhamento sobre ele, assim como outras referências utilizadas.

As **Semanas 3 e 4** foram marcantes no Processo, pois foi nesse período que ficou definido o objeto de estudo da pesquisa em si. Portanto, após a leitura do *Survey*, interessei-me particularmente pelo tópico de **criação de datasets instrutivos** para o treinamento de LLMs. Isso se deve à minha atuação, paralelamente à Residência, em um projeto que dispunha um volume massivo de dados não estruturados, os quais eram ligações gravadas de *Call Center*. A ideia central do presente trabalho foi responder à pergunta: **é possível gerar datasets para fine tuning de LLMs de forma automática, no formato mais adequado?** Fato que me fez abandonar a ideia de Mercado Financeiro.

A utilidade pensada para a resposta a essa pergunta é a possibilidade de se criar uma ferramenta automática que busca por demandas (questionamentos, dúvidas, solicitações) de clientes e percorra todas as ligações disponíveis em busca das respostas fornecidas pelos atendentes. Com isso, espera-se capturar respostas às dúvidas reais de clientes e estabelecer uma base com o par Pergunta e Resposta, a qual pode ser utilizada para treinar um LLM para realizar a tarefa de *Question Answering* (Q&A). Não só um produto pode ser formado a partir disso, quanto a minha própria especialização no domínio de Geração Automática de Dataset e, conseqüentemente, meu crescimento profissional.

Planejamento e execução da aplicação:

Em sequência, a **Semana 4** foi quando criou-se o *pipeline* (fluxo ou processo) o qual os dados serão submetidos para serem transformados em um dataset útil para o *fine tuning* de um LLM. Esse *pipeline* consiste em: obtenção de ligações, pré-processamento do áudio da ligação, transcrição da ligação, pré-processamento do texto da ligação, busca por informações relevantes na transcrição, construção de instâncias no formato instrutivo e *fine tuning* de um dataset.

Detalhes sobre o *pipeline* e uma abordagem visual podem ser encontrados no **Apêndice 2**, documento "**Pipeline preliminar**".

Resultados obtidos:

As próximas **Semanas (4 a 10)**, podem ser categorizadas como semanas de implementações de cada um dos blocos apontados no *pipeline*, os quais são ferramentas que promovem transformações nos dados.

O início da implementação foi nas **Semanas 4 e 5**, cujo objetivo era realizar a captura de ligações do *bucket* (*data lake* [repositório] de áudios) e iniciar as etapas de pré-processamento do áudio. Ao fim desse período, foi obtida a primeira ferramenta automática que, a partir de ligações gravadas, separa os canais do atendente e do cliente, realiza o *denoising* dos áudios, remove a Unidade de Resposta Audível (URA) das gravações e, por último, as transcreve. Os detalhes e explicações de cada etapa podem ser encontrados nos **Apêndices 2 e 3**.

Nesse momento, os dados, que antes eram apenas ligações cruas, já estão transcritos e rotulados como "**falas do atendente**" e "**falas do cliente**". Portanto, os dados foram transformados do formato de áudios para o formato de texto, o que marca a primeira grande transformação nos dados.

Naturalmente, surge a necessidade de se tratar o texto gerado, isto é, buscar e consertar erros, remover dados sensíveis e ruídos gerados pela transcrição. Essas etapas são fundamentais para garantir a qualidade do treinamento de um Modelo de Linguagem, o que inclui LLMs. Portanto, nas **Semanas 6, 7 e 8** dessa jornada, realizou-se o estudo de novas referências que explicam a relação da qualidade dos dados e qualidade de treino, além de explorarem técnicas que devem ser realizadas para garantir a qualidade dos dados textuais. Considerações aprofundadas sobre os estudos e as técnicas utilizadas, além de exemplos de problemas encontrados nas transcrições, podem ser encontrados no **Apêndice 4**. Vale ressaltar que não só a limpeza textual foi feita, quanto a anonimização das transcrições, de modo a retirar dados sensíveis presentes nas ligações, que são bem típicos do ambiente de *Call Center* (também explicado no Apêndice 4). Logo, ao final dessas três semanas, foram obtidos dados textuais limpos e iniciou-se o processo de obtenção de informações (demandas de clientes e respostas) a partir dos dados textuais.

Até então, os dados foram transformados de áudio para texto, o texto foi limpo e iniciou-se o processo de busca por informações nesses dados.

Ao longo da **Semana 9**, realizou-se esforços não só para realizar a detecção automática de **demandas de clientes** e **respostas de atendentes** às demandas, mas também a extração de *embeddings* dessas demandas e também dos possíveis candidatos a resposta. Todas essas informações foram armazenadas em um banco de dados baseado em vetores. A explicação detalhada sobre esses conceitos e exemplos obtidos encontram-se no **Apêndice 5**, no documento “**Documentação Semana 7**”. Então, foi armazenado tanto a demanda do cliente, reescrita de forma objetiva e salientando sua requisição em si, quanto a reescrita das falas dos atendentes, ressaltando as informações que eles desejam passar.

Até esse ponto, tem-se o banco de dados vetorial com as demandas dos clientes e as falas dos atendentes, todas **reescritas (Apêndice 5)** de modo a explicitar suas intenções e remover as marcas de oralidade. A partir disso, a base está preparada para a etapa de criação do par Pergunta e Resposta, que foi executada durante a **Semana 10** utilizando o mecanismo conhecido como busca semântica, o qual consiste na busca por documentos mais semelhantes semanticamente a outro documento utilizado como argumento de busca. Com isso, as demandas dos clientes foram recuperadas uma a uma e utilizadas para recuperar um número arbitrário de respostas de atendentes mais semanticamente próximas às demandas dos clientes. Entre outras palavras, estão sendo criados os pares de Pergunta e Resposta. Os detalhes de implementação se encontram no **Apêndice 6**. Então, ao fim dessa semana, obtemos o dataset preparado para *fine tuning*, no formato adequado com instruções em linguagem natural descrevendo a tarefa de Q&A.

Após todas as etapas de transformação e manipulação dos dados, é chegada a hora de realizar *fine tuning* de um LLM para avaliar a estrutura do dataset. Por isso, na **Semana 11**, foi executado o *Instruct Fine Tuning* de um modelo baseado em *Llama 2 7B*, o qual pode ser testado conforme link disponibilizado no **Apêndice 7**, bem como detalhes de implementação.

Todos os códigos criados neste trabalho estão disponibilizados no **Apêndice 8**, “**Códigos**” e no repositório *GitHub* referenciado no **Apêndice 7**, “**Repositório público no GitHub**”.

Conclusões e trabalhos futuros:

Ciente do exposto anteriormente, tento explicitar o esforço empregado durante a Disciplina Residência em IA, a qual teve diversos resultados explicados brevemente pelo texto acima e em detalhes nos Apêndices. Mas o que de fato foi importante, particularmente, é a evolução que passei durante este período. Os documentos dos primeiros Apêndices contém pequenos desvios do padrão acadêmico de documentação, propositalmente deixados neste documento final, de modo que possa ser percebida a diferença no estilo de escrita, relato e ideias deixadas nesses registros ao longo das semanas.

De fato, foram semanas de intensos aprendizados e auto-descobertas, as quais resultaram nas entregas aqui relatadas. Ao final, obteve-se um sistema automático que, a partir de ligações gravadas e armazenadas aos milhares em ambientes de *Call Center*, gera-se um *dataset* pronto para realização de *Instruct Tuning* de LLMs.

Mesmo com resultados relevantes expostos anteriormente, ainda há necessidade de ajustes nos *prompts* utilizados para filtrar as informações, ajustes nos algoritmos de limpeza e validação em outros casos de uso além das Telecomunicações. No entanto, este trabalho mostrou que é possível criar um **sistema automático** de forma viável, que acredito ser **pioneiro em sua área de atuação até o momento**.

Portanto, espero profundamente que este trabalho possa ajudar pessoas ao redor de todo o mundo que se interessem pela **Geração Automática de Datasets**, servindo a esses como ponto de partida em seus estudos, assim como serviu de ponto de partida a **minha jornada**.

APÊNDICE 1

Termo de Aceite de Entrega - Semana 1

Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

Data da Reunião (“gate”) de aprovação: 19 de out. de 2023


Participantes da Entrega [matriculados em Residência em IA]:

Alex Victor Echeverria Ugarte Silva

Entrega: [descrever a ENTREGA: requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

Planejamento da semana:

- Buscar as *keywords* da **CSCI** que melhor descrevem o trabalho final;
- Estabelecer uma área foco para pesquisa e aprofundamento;
- Buscar artigos científicos e revisões da literatura para iniciar o aprofundamento na área;
- Iniciar a busca por materiais que explicam a metodologia científica;
- Separar os artigos encontrados por tópicos que condizem com seus objetivos.

Entrega da semana:  GATE 1 - 191023

Planejamento: [descrever o que pretende fazer para realizar a próxima ENTREGA]

- Ler os artigos selecionados, principalmente as revisões e registrar em uma planilha;
- Buscar mais artigos atuais, de modelos estado da arte;
- Construir o vocabulário necessário para o entendimento aprofundado dos artigos;
- Buscar artigos científicos que exploram a conexão entre mercado financeiro e LLMs;
- Definir uma tarefa do mercado financeiro para a aplicação de LLM.

Observação: [caso precise fazer alguma observação, de qualquer “natureza”]

ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: 

LUANA GUEDES BARROS MARTINS: 

GATE 1 - 19/10/2023

Residência em IA

Aluno: Alex Echeverria

Objetivo: Aprofundar o conhecimento em Large Language Models, bem como nos fundamentos de NLP necessários para o desenvolvimento de LLMs e construção de vocabulário necessário para o estudo e entendimento de artigos científicos da área. Aplicação dos modelos em tarefas específicas do mercado financeiro, dado a complexidade das informações e a precisão e especificidade desses modelos em lidar com informações complexas.

Motivo da escolha por mercado financeiro: O mercado financeiro possui alta quantidade de dados disponíveis, dados passados e presentes disponíveis abundantemente e livremente na internet. Com isso, há inúmeras aplicações que podem ser realizadas e, principalmente, testadas em simulações tanto passadas quanto presentes.

Keywords do "Scope & Topics" CSCI selecionadas:

Natural language processing;

Expert systems;

- são sistemas inteligentes que incorporam bases de conhecimentos e tentam emular o processo de decisão humano de um expert da área. No caso, o sistema deve automaticamente tomar decisões de compras e vendas de ações.

Decision support systems;

- São sistemas que apoiam a tomada de decisão em negócios ou organizações, munindo o usuário do sistema com informações processadas

que o auxilie a levar em consideração dados humanamente difíceis de serem analisados. No caso de LLM + MF, pode-se pensar em análises de quantidades massivas de dados de notícias, tendências e outras informações para auxiliar a tomada de decisão de investidores e outros negócios / usuários.

Automated problem solving;

- O sistema desenvolvido deve ser capaz de solucionar problemas complexos de forma autônoma, isto é, no contexto de mercado financeiro deve ser capaz de extrair informações dos dados disponíveis e decidir entre as opções disponíveis automaticamente.

Knowledge discovery;

- Ao se pensar sobre mercado financeiro, pode-se afirmar que a extração de conhecimento de bases de dados é essencial para fornecer conhecimento aos modelos de linguagem a fim de produzir dicas e conceitos.

Knowledge representation;

- Modelos de linguagem basicamente buscam uma forma de representação da linguagem aprendida, de forma a extrair sentido e dar finalidade a uma tarefa de processamento de linguagem.

Knowledge-intensive problem solving techniques;

- O mercado financeiro é repleto de informações e notícias. Métodos e técnicas capazes de filtrar informações relevantes é necessário e fundamental para bons resultados nesse mercado.

Intelligent data mining and farming;

- É fundamental para o desenvolvimento de soluções que utilizem dados do mercado financeiro com intuito de obter alguma informação relevante que tenha um processo robusto de mineração de dados de diversas fontes.

Neural networks and applications;

Intelligent information fusion;

- Fusão de informações de diversas fontes faz-se necessário para a aplicação de sistemas inteligentes que consomem dados do mercado financeiro.

Unsupervised and Supervised Learning;

- Naturalmente há a necessidade de se trabalhar com dados supervisionados e não supervisionados para treinamento de modelos ou avaliações de desempenho.

Information extraction and web content mining;

- Há a necessidade de se obter informações do mercado financeiro disponíveis na internet. Com isso, a mineração de dados na web faz-se necessária.

High- throughput data analysis;

- A quantidade massiva de dados implica em análises massivas de dados por parte do modelo. Realizar essas análises em tempo hábil faz-se necessário dada a natureza do mercado financeiro.

Simulation and Modeling;

- Dada a natureza do mercado financeiro, simulações e modelagens dos eventos que acontecem fazem-se cruciais para testes de modelos.

Data-Intensive Computing; Parallel Algorithms;

- Pelo fato de ser tão intensivo a quantidade de dados disponíveis para análise e o requisito de tempo hábil, faz-se necessário implementar medidas de eficiência e paralelização na execução dos algoritmos.

Testing Methods;

- É de suma importância o teste intensivo dos modelos gerados para o mercado financeiro, dado que falhas podem significar a perda de quantidades massivas de capital.

Multidimensional Big Data;

- A capacidade de análise de um volume gigantesco de dados multidimensionais (como o texto em si) é um desafio inerente a LLMs e o mercado financeiro.

Sampling Methodologies;

- Amostrar de forma correta dados para executar treinamentos e testes é muito importante para garantir que os modelos não são enviesados e estão

performando de acordo com o esperado, considerando a quantidade massiva de dados disponíveis.

High Performance Computing;

- Computação de alto desempenho faz-se necessária para a execução de algoritmos gigantes com um volume massivo de dados em tempo hábil.

Artigos selecionados:

Rudimentos:

[1] Kauf, Carina, and Anna Ivanova. "A Better Way to Do Masked Language Model Scoring." *arXiv preprint arXiv:2305.10588* (2023).

[2] PETERS, M. E. et al. Deep contextualized word representations. Disponível em: <<https://arxiv.org/abs/1802.05365>>.

[3] PETERS, Matthew E. et al. Dissecting contextual word embeddings: Architecture and representation. *arXiv preprint arXiv:1808.08949*, 2018.

[4] VASWANI, Ashish et al. Attention is all you need. *Advances in neural information processing systems*, v. 30, 2017.

Modelos Clássicos:

[5] DEVLIN, Jacob et al. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[6] LEWIS, Mike et al. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*, 2019.

[7] RADFORD, Alec et al. Improving language understanding by generative pre-training. 2018.

[8] BROWN, Tom et al. Language models are few-shot learners. *Advances in neural information processing systems*, v. 33, p. 1877-1901, 2020.

[9] FLORIDI, Luciano; CHIRIATTI, Massimo. GPT-3: Its nature, scope, limits, and consequences. *Minds and Machines*, v. 30, p. 681-694, 2020.

[10] THOPPILAN, Romal et al. Lamda: Language models for dialog applications. arXiv preprint arXiv:2201.08239, 2022.

Datasets:

[11] BOWMAN, Samuel R. et al. A large annotated corpus for learning natural language inference. arXiv preprint arXiv:1508.05326, 2015.

[12] RAJPURKAR, Pranav et al. Squad: 100,000+ questions for machine comprehension of text. arXiv preprint arXiv:1606.05250, 2016.

[13] JOSHI, Mandar et al. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. arXiv preprint arXiv:1705.03551, 2017.

Avaliação de modelos:

[14] CHANG, Yupeng et al. A survey on evaluation of large language models. arXiv preprint arXiv:2307.03109, 2023.

Eficiência de modelos:

[15] GEIPING, Jonas; GOLDSTEIN, Tom. Cramming: Training a Language Model on a single GPU in one day. In: International Conference on Machine Learning. PMLR, 2023. p. 11117-11143.

[16] HU, Edward J. et al. Lora: Low-rank adaptation of large language models. arXiv preprint arXiv:2106.09685, 2021.

Revisões:

[17] NAVEED, Humza et al. A comprehensive overview of large language models. arXiv preprint arXiv:2307.06435, 2023.

[18] ZHAO, Wayne Xin et al. A survey of large language models. arXiv preprint arXiv:2303.18223, 2023.

Aplicado ao mercado financeiro:

[19] YANG, Hongyang; LIU, Xiao-Yang; WANG, Christina Dan. FinGPT: Open-Source Financial Large Language Models. arXiv preprint arXiv:2306.06031, 2023.

Termo de Aceite de Entrega - Semana 2

Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

Data da Reunião (“gate”) de aprovação: 19 de out. de 2023

Participantes da Entrega [matriculados em Residência em IA]:

Alex Echeverria, Heloisy, Héber Júnior.

Entrega: [descrever a ENTREGA: requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

- Continuação da análise TOP DOWN, início do afunilamento do Objeto de Estudo (LLM).
- Início da construção do vocabulário compartilhado para aprofundamento e entendimento pleno do Objeto de Estudo: [Vocabulário NLP LLM](#)
- Início da leitura em profundidade do artigo: [Artigo: A Survey of Large Language Models](#)
- Construção do repositório compartilhado de artigos: [Artigos NLP LLM](#)
- Inserção de mais artigos de revisão da literatura no repositório
- Abandono da ideia de mercado financeiro.

Planejamento: [descrever o que pretende fazer para realizar a próxima ENTREGA]

- Finalizar a leitura do artigo;
- Definir a Propriedade do Objeto a ser estudada;
- Realizar medição da propriedade;
- Buscar interferências anteriores, *i.e.*, revisar a literatura para a propriedade do objeto definida.

Observação: [caso precise fazer alguma observação, de qualquer “natureza”]

Heloisy e Héber ajudam na elaboração do material, mesmo seguindo por caminhos diferentes.

ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: [Go!](#)

LUANA GUEDES BARROS MARTINS: [Go! ▾](#)

Artigo: A Survey of Large Language Models

[Submitted on 31 Mar 2023 (v1), last revised 11 Sep 2023 (this version, v12)]

Comments: ongoing work; 97 pages, 683 citations.

Cited by: 233

O artigo expõe uma revisão histórica da literatura e de conceitos chave para a ampla compreensão do estado atual de Large Language Models.

De forma bem detalhada, ele foca em diferenciar Language Models de Large Language Models através de suas características principais: quantidade de parâmetros, quantidade de dados utilizados no treinamento e a quantidade de recursos computacionais necessários para a execução do algoritmo.

Uma característica ressaltada pelo artigo é o surgimento de habilidades especiais, além da significativa melhora em performance nas tarefas de *Natural Language Processing (NLP)*, que não são observadas em modelos de linguagens com menos parâmetros.

O artigo busca, além de uma revisão da literatura, fornecer um background teórico dos conceitos básicos de LLM, suas habilidades, técnicas para treinamento, inferência, *alignment* do modelo com os valores humanos (como honestidade e empatia) e recursos disponíveis para a implementação e treinamento de modelos, como API's, bibliotecas, datasets e checkpoints de modelos publicamente disponíveis.

Apresenta também a evolução técnica da família de modelos GPT, gráficos contendo marcos históricos do lançamento de LLM na indústria e academia, explicações e dicas de como realizar pré-treino, arquiteturas de modelos, adaptações para LLMs, dentre outras informações relacionadas.

Portanto, esse artigo contém a profundidade necessária para a devida inserção de pesquisadores no assunto de LLM, fornecendo o embasamento teórico necessário para que o pesquisador possa buscar a Propriedade do Objeto a ser estudada de forma fluente, além de ampla referência bibliográfica.

ZHAO, W. X. et al. A Survey of Large Language Models. arXiv (DOI: 2303.18223), 31 mar. 2023.

APÊNDICE 2

Termo de Aceite de Entrega - Semana 3

Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

Data da Reunião (“gate”) de aprovação: 9 de nov. de 2023

Participantes da Entrega [matriculados em Residência em IA]:

Alex Echeverria

Entrega: [descrever a ENTREGA: requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

As tarefas prometidas foram:

- > Finalizar a leitura do artigo;
- > Definir o tema a ser estudado;
- > Buscar na literatura referências base para auxiliar a construção da solução.

- Definição do tema do trabalho: **Automação na Criação de Datasets prontos para Fine-tuning de Grandes Modelos de Linguagem a partir de Chamadas de Call Center.**

- Basicamente, após a leitura da *survey* "A Survey of Large Language Models", de Zhao et al., interessei-me particularmente pela área de preparação de datasets para o adequado fine-tuning de LLM's. Isso se deve ao fato de haverem diversos critérios para a construção adequada das instruções que constituem o dataset para fine-tuning. Diversidade e qualidade das tarefas específicas presentes nas instruções, além de demonstrações e outras estratégias para a construção efetiva do dataset.

Disso, surge a curiosidade: como automatizar o processo de construção de um bom dataset para fine-tuning de LLM's a partir de dados de ligações brutos?

- Estudos preliminares:
Is Prompt All You Need? No. A Comprehensive and Broader View of Instruction Learning; LOU, R. et al., arxiv: 2303.10475, mar. 2023. Disponível em: <https://arxiv.org/abs/2303.10475> . Acesso em 05 nov. 2023.

Efficiently Scale LLM Training Across a Large GPU Cluster with Alpa and Ray. Disponível em: <https://developer.nvidia.com/blog/efficiently-scale-llm-training-across-a-large-gpu-cluster-with-alpa-and-ray/>. Acesso em: 9 nov. 2023.

A Survey of Large Language Models; ZHAO, W. X. et al., arxiv: 2303.18223, sep. 2023. Disponível em: <https://arxiv.org/pdf/2303.18223.pdf> . Acesso em 15 out. 2023.

- Leitura da Survey citada anteriormente [A survey of Large Language Models].

Planejamento: [descrever o que pretende fazer para realizar a próxima ENTREGA]

- Leitura dos papers:
 - [Normalization of non-standard words - ScienceDirect](#)
 - [Is Prompt All You Need? https://arxiv.org/abs/2303.10475](https://arxiv.org/abs/2303.10475)
- Captura dos dados;
- Transcrição dos dados;
- Definição da infraestrutura necessária para o pipeline (recursos);
- Documentação do processo.

Observação: [caso precise fazer alguma observação, de qualquer “natureza”]

Agradecimento aos professores: Cedric, Aldo, Leonardo, Fernando, Wellington pelas conversas, esclarecimentos, opiniões e direcionamentos que me ajudaram a encontrar o tema de minha residência.

Agradeço, especialmente, ao Heinz, Luiz Guilherme e João Gabriel por ajudarem, indiretamente, na definição do tema.

ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: **Go!** ▾

LUANA GUEDES BARROS MARTINS **Go!** ▾

Termo de Aceite de Entrega - Semana 4

Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

Data da Reunião (“gate”) de aprovação: 16 de nov. de 2023

Participantes da Entrega [matriculados em Residência em IA]:

Alex Echeverria

Entrega: [descrever a ENTREGA: requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

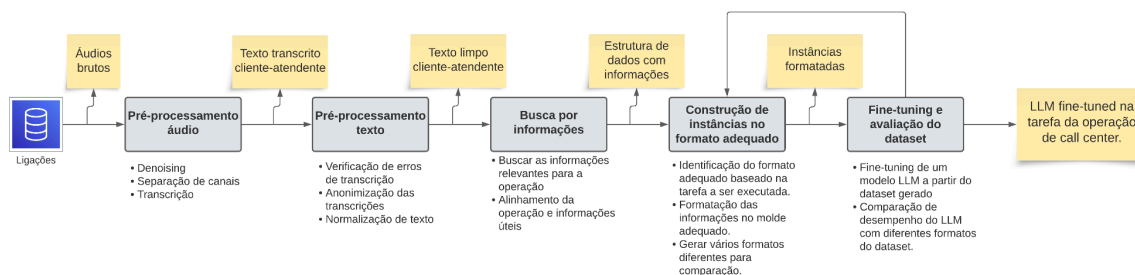
As tarefas prometidas foram:

- > Leitura dos papers:
 - [Normalization of non-standard words - ScienceDirect](#)
 - Is Prompt All You Need? <https://arxiv.org/abs/2303.10475>
- > Captura dos dados;
- > Transcrição dos dados;
- > Definição do pipeline;
- > Documentação do processo.

As tarefas feitas foram:

- > Elaboração prévia do **pipeline** dos dados, a fim de conduzir o fluxo de entregas apropriadamente;

 Pipeline-AutoInstruct4Call



- > Início da **construção dos códigos**:
 - captura de dados;
 - parte do pré-processamento: separação dos canais de áudio e denoiser.
- > Documentação do processo de construção do código, dúvidas e referências utilizadas.

☰ DocumentaçãoSemana1-AutoInstruct4Call

Planejamento: [descrever o que pretende fazer para realizar a próxima ENTREGA]

- Continuar os avanços no desenvolvimento do código
- Continuar a busca por referências para sanar as dúvidas e embasar as evoluções no código.

Observação: [caso precise fazer alguma observação, de qualquer “natureza”]

ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: **Go!** ▾

LUANA GUEDES BARROS MARTINS: **Go!** ▾

Pipeline preliminar

Qual a motivação?

- Como explicado em [1], LLM's que são treinados em instâncias formatadas com descritores de tarefas são mais generalistas a instruções não vistas que outros LLM's treinados em um corpus não formatado. Isso significa que uma simples descrição da tarefa, em linguagem natural, pode melhorar muito a capacidade de generalização do LLM para tarefas não vistas durante o treinamento.
- No universo de Call Centers, é fato a quantidade enorme de ligações / atendimento gravados que são armazenados pela empresa durante o prazo mínimo de 90 dias. Visto isso, é interessante pensar em formas de automatizar o processo de mineração de dados, qual produto que deseja-se obter é o dataset limpo e pronto para o fine tuning de LLM's.

Pipeline:

1 - Transcrição dos dados

- como realizar a transcrição de maneira eficiente?
- quais modelos podem ser utilizados?
- quais requisitos?
- buscar papers.

2 - Pré-processamento do texto

- verificar erros de transcrição
- eliminar dados sensíveis (anonimização)
- normalização de texto.
- papers: [5]

3 - Busca por informações

- quais informações serão úteis para o treino?
- como determinar quais informações serão importantes? (A partir da demanda?)
- info retrieval (Google, search algorithms)
- estruturação das informações buscadas
- buscar papers.

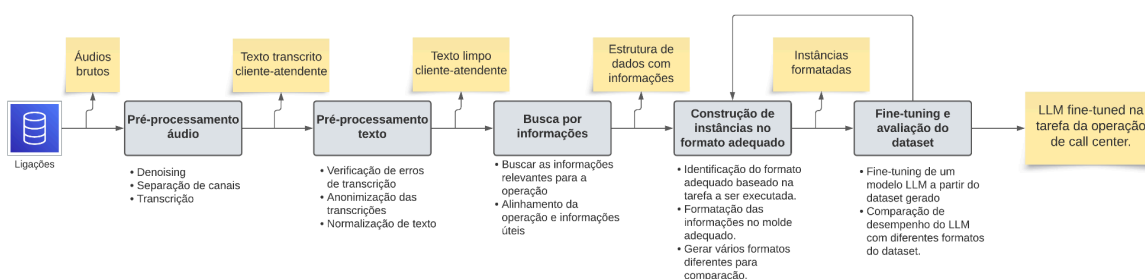
4 - Construção das instâncias formatadas

- quais técnicas serão utilizadas para formatar?
- qual o formato adequado para cada tarefa?
- como identificar a tarefa?
- Papers: [1][2][3][4]

5 - Avaliação do dataset (em performance)

- qual será o benchmark?
- geração de vários modelos e escolha de um?
- qual modelo será feito fine-tuning para avaliação?
- como será o pré-treino desse modelo?
- buscar papers de modelos que comparem seus desempenho em tarefas.

Pipeline visual: [LucidApp - Pipeline](#)



Referências:

[1] **A Survey of Large Language Models**; ZHAO, W. X. et al., arxiv: 2303.18223, sep. 2023. Disponível em: <https://arxiv.org/pdf/2303.18223.pdf> . Acesso em 15 out. 2023.

[2] SANH, V. et al. **Multitask Prompted Training Enables Zero-Shot Task Generalization.** arXiv:2110.08207 [cs], 15 out. 2021. Disponível em: <https://arxiv.org/abs/2110.08207> . Acesso em 04 nov. 2023.

[3] **Is Prompt All You Need? No. A Comprehensive and Broader View of Instruction Learning**; LOU, R. et al., arxiv: 2303.10475, mar. 2023. Disponível em: <https://arxiv.org/abs/2303.10475> . Acesso em 05 nov. 2023.

[4] JASON ZHANSHUN WEI et al. **Finetuned Language Models Are Zero-Shot Learners.** 3 set. 2021. Disponível em: <https://arxiv.org/abs/2109.01652> . Acesso em 05 nov. 2023.

[5] SPROAT, R. et al. **Normalization of non-standard words.** Computer Speech & Language, v. 15, n. 3, p. 287–333, jul. 2001. Disponível em: <https://www.sciencedirect.com/science/article/abs/pii/S088523080190169X> . Acesso em 06 nov. 2023.

Documentação Semana 1: Início das atividades de codificação

1 - Captura dos dados:

As ligações estão armazenadas em um repositório (denomina-se *bucket*) nos servidores da empresa Amazon, especificamente em seus serviços popularmente conhecidos como AWS.

Isso indica a necessidade da criação de um script, por ora em python, para que esses dados sejam baixados para a máquina que irá realizar o processamento.

Foi feito um código primitivo para a captura desses dados. No entanto, ainda é ineficiente pelo fato de não ter sido baseado na documentação própria da AWS.

2 - Pré-processamento dos dados:

Os dados brutos, isto é, as ligações *in natura*, necessitam de pré-processamento para que sejam transcritas de maneira eficiente.

Para isso, foi realizada a separação de canais do áudio, de forma que o canal 0 represente o atendente e o canal 1 represente o cliente. Com essa organização, obtém-se a primeira separação que beneficie a separação das demandas do cliente e as respostas que são fornecidas a essas questões.

Em seguida, realiza-se a remoção de ruídos utilizando o framework *denoiser*, do *facebook research* [1], de tal forma que se evite erros durante a transcrição por causa de ruídos presentes no áudio.

[1] Real Time Speech Enhancement in the Waveform Domain (Interspeech 2020).

Disponível em: <<https://github.com/facebookresearch/denoiser>>. Acesso em: 16 nov. 2023.

APÊNDICE 3

Termo de Aceite de Entrega - Semana 5

Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

Data da Reunião (“gate”) de aprovação: 23 de nov. de 2023

Participantes da Entrega [matriculados em Residência em IA]:

Alex Echeverria

Entrega: [descrever a ENTREGA: requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

As promessas da última semana:

- Continuar desenvolvimento do pipeline;
- **Pensar em checkpoints de validação do pipeline.**

O que foi desenvolvido:

- Documentação da continuação do pipeline:
[DocumentaçãoSemana2-AutoInstruct4Call](#)
Basicamente, avançou-se na questão de denoising e transcrição dos áudios, mas agora com um novo processo de remoção de URA (Unidade de Resposta Audível) das gravações.
- Repositório com os códigos feitos:
[aveusalex/AutoInstruct4LLMsPublic: Public \(github.com\)](https://github.com/aveusalex/AutoInstruct4LLMsPublic)
- Pensar em checkpoints de avaliação parcial de cada etapa do pipeline:

A solução parcial é usar o próprio ChatGPT para avaliar os resultados de cada etapa. Mas ainda será melhorado e pensado com mais profundidade essa questão.

Planejamento: [descrever o que pretende fazer para realizar a próxima ENTREGA]

- **Continuar o desenvolvimento do pipeline -> concluir a etapa pré-processamento de texto, busca por informações e construção de instâncias formatadas.**
- **Melhorar a ideia de avaliação parcial de checkpoints.**

Observação: [caso precise fazer alguma observação, de qualquer “natureza”]

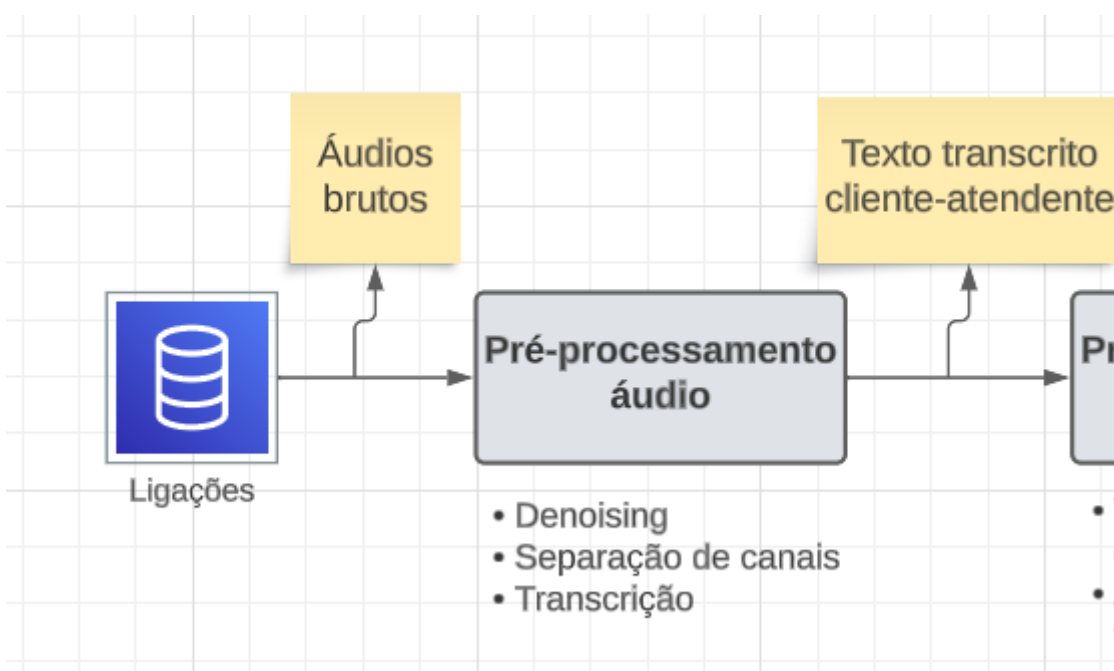
Agradecimentos ao João Gabriel pela ajuda em infraestrutura de recursos computacionais.

ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: [Go!](#)

LUANA GUEDES BARROS MARTINS: [Go!](#)

Documentação Semana 2: Continuação da codificação



Na última semana, já havia iniciado os códigos referentes ao denoising dos áudios e separação de canais. Nesta semana, continuei os trabalhos com a Remoção de URA (que não estava planejada) e a Transcrição a partir dos áudios "denoised" e cortados a URA.

1 - Remoção de URA:

1.1 - Introdução à URA:

As ligações vêm com uma parte inicial chamada URA, acrônimo de Unidade de Resposta Audível, que nada mais é que uma secretária eletrônica para realizar o direcionamento da demanda do cliente a partir da vocalização de opções. O cliente interage diretamente com a URA para escolher o tópico que sua demanda melhor se encaixa, a partir do teclado do aparelho telefônico ou por voz [1]. Por isso, deve-se entender como a URA pode contribuir para o presente estudo e quais as dificuldades que podem ser encontradas.

No contexto deste presente trabalho, a URA não traz informações adicionais, dado que são apenas categorias fixas para direcionamento de demanda ao setor mais adequado, sendo o conhecimento presente na conversa do cliente com o atendente a parte que interessa de fato ao trabalho.

Portanto, deve-se realizar a remoção de todo o trecho que contém a URA, já que não há ganhos de informações para o treino de LLM's (relembrando a necessidade de dados que apresentem perguntas/demandas e as respostas para essas perguntas).

1.2 - Das dificuldades de se remover URA:

Dado que a URA é uma gravação de voz, ela contém aspectos que são invariantes em qualquer ligação, como o timbre, forma de falar, espaçamento da fala, *pitch*, entre outras características. No entanto, como cada cliente terá uma demanda específica, cada interação com a URA é diferente, pois os clientes podem selecionar opções diferentes, com um tempo de escolha diferente (i.e., o tempo que demorou para escolher uma opção), o que torna cada trecho de URA único.

Também há algumas particularidades que dificultam ainda mais a identificação de trechos de URA, que são:

- Presença de músicas e outras gravações para a espera, ou seja, enquanto o cliente aguarda ser atendido há a reprodução de músicas e outras campanhas de marketing.
- A variabilidade do tempo de espera, isto é, o tempo que o cliente aguarda para ser atendido varia de acordo com a demanda dos serviços do call center.

Dado todos esses empecilhos presentes, opto por uma abordagem não supervisionada, já que existem, de fato, padrões entre as URA's e os atendimentos em si.

1.3 - Da abordagem utilizada:

A fim de remover os trechos de URA presentes nos áudios, foi desenvolvida a seguinte abordagem:

- 1) Carregamento do canal do atendente (o que contém a URA);
- 2) Separação do sinal em janelas de tempo fixo;
- 3) Extração de features de cada janela gerada;
- 4) Utilização do K-Means para clusterização das janelas de features;
- 5) Encontrar o ponto de troca entre os clusters, dada uma série temporal formada pelos segmentos e seus respectivos clusters pertencentes;
- 6) Remover os segmentos que estão antes do ponto de troca.

Essa abordagem proposta funcionou em testes de pequena escala, no entanto, apresentou problemas em alguns casos.

Para fins de agilidade, tudo foi feito apenas de forma empírica, sem embasamento formal na literatura.

Ponto chave de melhoria: Quais features de fato devem ser utilizadas?

2 - Transcrição dos áudios:

Para transcrever os áudios já cortados, utiliza-se as ferramentas Faster Whisper e Insanely Fast Whisper, as quais utilizam a ferramenta de transcrição *Whisper*, desenvolvido pela OpenAI. [2, 3, 4]

Essas ferramentas utilizam o modelo Whisper Large como base, melhorando sua eficiência aplicando diversas técnicas como LoRA [5], quantização e Flash Attention [6] para acelerar a transcrição.

[1] **O que é URA, aplicações no atendimento ao cliente e vantagens.** Disponível em: <<https://www.zendesk.com.br/blog/o-que-e-ura/>>. Acesso em: 22 nov. 2023.

[2] **Faster Whisper transcription with CTranslate2.** Disponível em: <<https://github.com/SYSTRAN/faster-whisper>>. Acesso em: 23 nov. 2023.

[3] **SRIVASTAV, V. Insanely Fast Whisper.** Disponível em: <<https://github.com/Vaibhavs10/insanely-fast-whisper>>. Acesso em: 23 nov. 2023.

[4] **RADFORD, A. et al. Robust Speech Recognition via Large-Scale Weak Supervision.** [s.l: s.n.]. Disponível em: <<https://cdn.openai.com/papers/whisper.pdf>>. Acesso em: 23 nov. 2023.

[5] **HU, E. J. et al. LoRA: Low-Rank Adaptation of Large Language Models.** arXiv:2106.09685 [cs], 16 out. 2021. Acesso em: 23 nov. 2023.

[6] **DAO, T. et al. FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness.** Disponível em: <<https://arxiv.org/abs/2205.14135>>. Acesso em: 23 nov. 2023.

APÊNDICE 4

Termo de Aceite de Entrega - Semana 6

Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

Data da Reunião (“gate”) de aprovação: 30 de nov. de 2023

Participantes da Entrega [matriculados em Residência em IA]:

Alex Echeverria

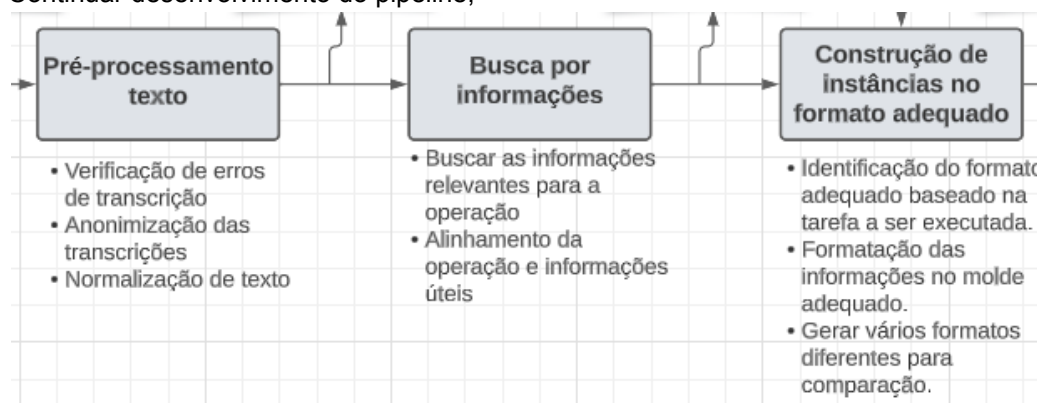
Entrega: [descrever a ENTREGA: requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

Recapitulando:

- Este trabalho propõe a elaboração de um pipeline que consome gravações de áudio de ligações em um ambiente de call center de modo a gerar datasets prontos para Instruct Fine Tuning de LLM's, automaticamente.
- O pipeline pode ser encontrado em:
[Pipeline-AutoInstruct4Call](#)

O que foi planejado para essa semana:

- Continuar desenvolvimento do pipeline;



- Melhorar a ideia de avaliação parcial de checkpoints.

O que foi desenvolvido:

- Leitura do artigo e desenvolvimento de observações relevantes:

☰ DocumentaçãoSemana3-AutoInstruct4Call

- Salienta-se a forma em que são feitas as avaliações dos checkpoints deles:

Utilizam métricas arbitrariamente definidas para avaliar os melhores hiperparâmetros de mistura entre datasets diferentes, analisando métricas de tokens e classificadores binários que identificam a qualidade de um dataset.

Planejamento: [descrever o que pretende fazer para realizar a próxima ENTREGA]

- Utilizar a ferramenta desenvolvida por eles (disponibilizada no Github) para consolidar o estudo;
- Incorporá-la ao pipeline desenvolvido do trabalho;
- Realizar buscas por modelos de busca de informações em texto;
- Melhorar a ideia de avaliação parcial de checkpoints.

Observação: [caso precise fazer alguma observação, de qualquer “natureza”]

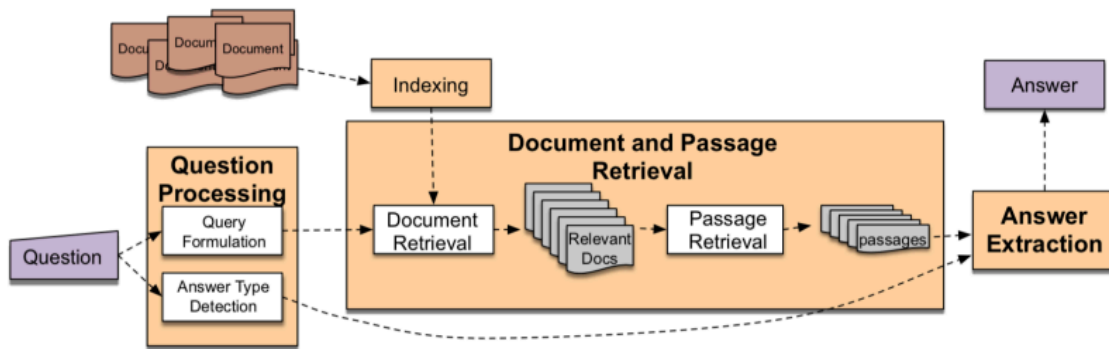
Neste gate, o Professor Aldo André Díaz Salazar esteve na banca avaliadora substituindo a Professora Luana.

ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: Go! ▾

LUANA GUEDES BARROS MARTINS: Em análise! ▾

- Leitura do artigo de ABBASIANAE et al., o qual denomina-se Text-based Question Answering from Information Retrieval and Deep Neural Network Perspectives: A Survey, que visa revisar as diferentes formas de se buscar conhecimento em texto, tanto na visão clássica (estatística e com regras bem definidas) quanto na perspectiva do uso de algoritmos de Deep Learning.



Planejamento: [descrever o que pretende fazer para realizar a próxima ENTREGA]

- Finalizar a implementação dos operadores de limpeza de texto.
- Implementar a ideia apresentada de buscar informações no corpus.
- Formatar as instâncias no formato adequado para Instruct fine-tuning, seguindo como base o que é descrito na literatura.

Observação: [caso precise fazer alguma observação, de qualquer “natureza”]

Neste gate, o Professor Aldo André Díaz Salazar esteve na banca avaliadora substituindo a Professora Luana.

ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: Go!

LUANA GUEDES BARROS MARTINS: Em análise!

Termo de Aceite de Entrega - Semana 8

Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

Data da Reunião (“gate”) de aprovação: 14 de dez. de 2023

Participantes da Entrega [matriculados em Residência em IA]:

Alex Echeverria

Entrega: [descrever a ENTREGA: requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

Recapitulando:

- Este trabalho propõe a elaboração de um pipeline que consome gravações de áudio de ligações em um ambiente de call center de modo a gerar datasets prontos para Instruct Fine Tuning de LLM's, automaticamente.

O que foi planejado para essa semana:

- Finalizar a implementação dos operadores de limpeza de texto.
- Implementar a ideia apresentada de buscar informações no corpus.
- Formatar as instâncias no formato adequado para Instruct fine-tuning, seguindo como base o que é descrito na literatura.

O que foi desenvolvido:

- Os avanços estão descritos no seguinte documento, marcado por **Amarelo** o início da documentação referente a essa semana:
[DocumentaçãoSemana6-AutoInstruct4Call](#)
- Finalização da implementação dos operadores de texto, sendo que apenas 2 foram utilizados sem modificações e 2 serviram de base para implementar o filtro de repetições.
- Implementação de novos operadores para remoção de Nomes e CPF's.
(Operador de nomes foi implementado duas vezes)
- Implementação do buscador de demandas a partir das falas transcritas e pré-processadas dos clientes.

Planejamento: [descrever o que pretende fazer para realizar a próxima ENTREGA]

- Finalizar a implementação da busca por respostas às demandas encontradas / reescritas.
- Implementar o formatador de instâncias para Instruct Fine Tuning.
- Iniciar a implementação do treino de LLM's a partir dos datasets gerados.

Observação: [caso precise fazer alguma observação, de qualquer “natureza”]

Agradecimento especial ao Daniel Fazzioni por ajudar a pensar nos operadores aqui apresentados.

ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: **Go!** ▾

LUANA GUEDES BARROS MARTINS: **Go!** ▾

(Append da semana 6 na documentação da semana 5)

Documentação Semana 4: Desenvolvimento do pipeline com Data-Juicer

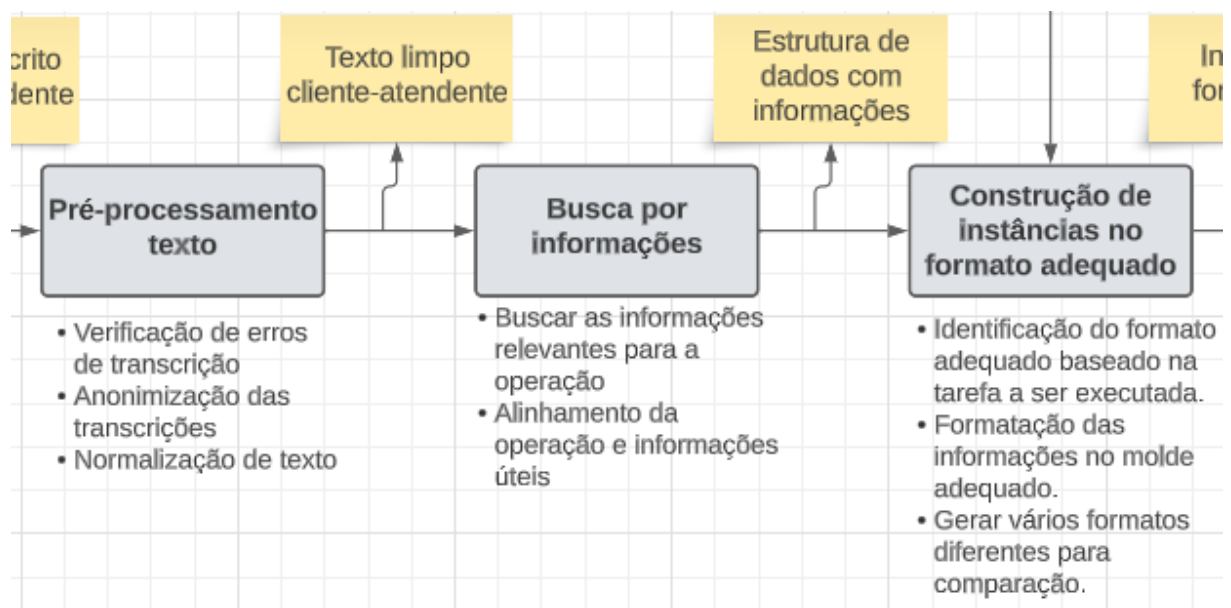


Figura 1: Pipeline que representa o presente trabalho, com ênfase colocada especialmente nas tarefas de pré-processamento de texto, busca por informações e construção de instâncias, dado que foram os tópicos em mente dessa semana.

Introdução:

Na última semana, finalizou-se a etapa referente a transcrição de áudios brutos, obtendo-se, assim, dados textuais. Vale ressaltar que, embora o formato textual extraído da gravação da ligação é o que se deseja, a saída da primeira etapa está sujeita a vários tipos de erros e ruídos que não beneficiam o treino de um LLM, conforme ressaltado em ZHAO et al. [1] e CHEN et al. [2].

Realizando uma breve análise exploratória das transcrições, é notável a presença dos seguintes erros:

- Transcrição indevida das falas, como:

<cliente>: explicar. Ela já negou, disse que foi um serviço que eu tinha na minha linha e de repente </cliente>

<atendente>: E aí E aí E aí E aí E aí E aí E aí E aí </atendente>

- Alucinações inesperadas;
- Deslocamento temporal entre falantes, isto é, falta de precisão temporal na transcrição.

Portanto, faz-se necessário a limpeza desses dados empregando técnicas de pré-processamento de texto. Para isso, foram pesquisadas quais são as formas exploradas nos trabalhos existentes, de tal forma que possam ser aproveitadas neste presente trabalho.

Revisão histórica:

Segundo ZHAO et al. e CHEN et al. [1, 2], a qualidade dos dados utilizados para pré-treino ou *fine tuning* impactam significativamente no desempenho dos LLM's. A presença de dados tóxicos (aqueles que não vão de encontro aos princípios e valores humanos de respeito, honestidade e usabilidade [1]), ruidosos (que não trazem informações relevantes, como marcadores de uma página do *Wikipedia*) e duplicados irão tanto degradar a performance de modelos em benchmarks quando comparados a outros que foram treinados em *corpora* limpos, quanto causar o chamado "*double descent*" que é a performance de treino piorando consecutivamente, inicialmente, acompanhado pela melhora logo em seguida. Mostra-se necessário, pois, a deduplicação, a filtragem por qualidade e a filtragem por toxicidade a fim de tornar o treino mais estável e evitar a degradação do modelo.

CHEN et al. [2] relata a criação e detalha o funcionamento de uma ferramenta cujo objetivo é auxiliar na etapa de pré-processamento dos *corpora* utilizados tanto no pré-treino e fine tuning de LLM's, incorporando mais de 50 operadores (OP), que são funções que atuam sobre o texto a fim de exercer alguma transformação.

Metodologia:

Para CHEN et al. [2], os grandes desafios incidentes sobre o pré-processamento de corpus para LLM's estão relacionados com a grande heterogeneidade dos dados, devido às diversas fontes em que são obtidos (como páginas web, livros, códigos, artigos científicos, conversas formais e informais, e-mails, etc.), com a dificuldade em se medir o impacto de diferentes "data recipes" (mistura de dados de diversas fontes, utilizado para treinar os modelos) por causa do alto custo ao se treinar LLM's e com a característica fundamental desses conjuntos de treinamento que é seu tamanho massivo, com bilhões e até mesmo trilhões de tokens [2].

Por tal motivo, CHEN et al. desenvolveram a ferramenta Data-Juicer [2], a qual visa abstrair diversos métodos de limpeza de corpus, permitir visualização das transformações e resultados obtidos, garantir eficiência no pré-processamento dos dados e criar um loop de

melhoria do corpus selecionado a partir do desempenho do modelo treinado nesse conjunto de dados.

A ferramenta está organizada em 4 grandes categorias: Formatores, Mapeadores, Filtros e Deduplicadores:

- **Formatadores:** é composto por diversos operadores que transformam os datasets obtidos por diversas fontes (em diversos formatos, tais como .txt, .html, .pdf, .py, .cpp, ...) em um formato intermediário comum, ou seja, uma estrutura de dados comum aos processos seguintes ao pipeline. Essa estrutura se dá por 3 partes: “text” (texto bruto), “meta” (metadados), “stats” (estatísticas extraídas do texto).
- **Mapeadores:** composto por operadores que realizam as modificações “inplace” dos textos, isto é, transformações e adaptações ao texto diretamente para que se encaixe na demanda estrutural do LLM que consumirá esses dados.
- **Filtros:** essa categoria é composta por operadores que são definidos com base em regras bem formadas que analisam as estatísticas do dataset (campo “stats”, como relatado em Formatores), métricas definidas pelo usuário ou informações externas (como listas de stop-words) para remover tais palavras do corpus.
- **Deduplicadores:** como o nome sugere, é composto por uma série de operadores responsáveis por eliminar trechos duplicados no corpus que podem degradar a performance dos modelos e também ocupar espaço desnecessariamente.

Diferenciais da ferramenta:

Como pode ser observado em CHEN et al. [2], há características que destacam a ferramenta proposta por Chen et al. das demais. Essas são:

- **Feedback-driven data processing:**
Esse tipo de orientação do pré-processamento surge com a necessidade de se realizar transformações no dataset que sejam impactantes de fato e, portanto, funcionais. Na ferramenta, foi incorporada diversas ferramentas de visualização e “trackamento” automático de alterações no dataset a cada etapa. Com isso, o usuário fica ciente a cada alteração de quais foram os impactos no corpus em questão.
- **Otimização de hiperparâmetros:**
A ferramenta é capaz de relacionar os hiperparâmetros existentes no

pré-processamento com métricas obtidas ao longo do processo (arbitrárias), de forma que possa haver a seleção automática dos melhores hiperparâmetros a partir desses sinais obtidos.

- Feedback loop integrado com o ambiente de treinamento de LLM's: A ferramenta é capaz de integrar-se com ambientes de desenvolvimento de LLM, como Megatron-LM [3], Deepspeed [4], e HuggingFace-transformers [5] e utilizar as métricas disponibilizadas por essas bibliotecas para, iterativamente, selecionar o corpus com melhor desempenho de treino.

Pertinência a este trabalho:

Como pode ser observado na figura 1, o pipeline construído neste trabalho espera-se o pré-processamento de texto bruto, com ruídos e problemas de construção, além da formatação das informações obtidas dos dados no formato esperado de Instruct Fine Tuning. Com isso, a partir do estudo de CHEN et al. [2], nota-se a grande similaridade entre as obras, o que revela a possibilidade de se utilizar o framework criado para impulsionar a criação do pipeline proposto por este trabalho.

A etapa de busca por informações não é contemplada pelo artigo estudado, portanto é uma demanda exclusiva do presente trabalho, a qual deve ser aprofundada e investigada posteriormente por ser etapa fundamental do pipeline.

ATUALIZAÇÃO SEMANA 5

Após a leitura e entendimento da documentação da ferramenta desenvolvida por CHEN et al. [2], disponibilizada no GitHub [6], verificou-se que, embora haja diferenciais já mencionados anteriormente neste trabalho, o framework não mostra-se tão proveitoso e pertinente ao problema estudado aqui o quanto antes fora estimado.

Para justificar o desencaixe do framework proposto por CHEN et al., exemplifica-se uma transcrição obtida a partir da etapa de transcrição das ligações gravadas. Os dados sensíveis (como nomes, CPF, números de telefone e endereços) foram substituídos por dados fictícios, manualmente, a fim de garantir a privacidade das conversas, mantendo o formato originalmente transcrito:

atendente: Obrigada por aguardar.

cliente: ...
cliente: ...
cliente: ...
cliente: ...
cliente: ...
cliente: ...
cliente: ...
cliente: ...
cliente: ...
cliente: ...
atendente: Informe seu nome completo, seu titular de serviço,
cliente: ...
cliente: ...
cliente: Sra. Maildes Almeida dos Anjos
atendente: e-mail cadastrado, Nome, seu nome completo, seu titular do serviço.
cliente: Maildes Almeida
atendente: E-mail cadastrado.
cliente: 1357
cliente: arroba gmail.com
cliente: 629
atendente: Telefone para contato com o DDD. Eu identifiquei em nosso sistema que o senhor fez uma reclamação no dia 12 do 6
cliente: 98023749 629-8023-7497.
atendente: referente à sua linha, que estava sem sinal.
[...]
cliente: Mas é só internet mesmo
atendente: Certo.
atendente: Um momento.
cliente: Ok e aí e
quando a minha linha não tem problema botar na minha recarga vinha falava o mês todo chegava no final do
mês botava de novo não tinha problema acabou a oi migrou para claro tá horrível
cliente: aí fica pedindo para poder fazer recarga mais recarga
[...]
atendente: O protocolo permanece o mesmo, que é o 4023-123-1234-1234
atendente: Posso lhe auxiliar em algo mais?
cliente: Esse corpo para o Donatel né a senhora poderia repetir para mim
atendente: Isso, é o mesmo protocolo que foi reaberto.
atendente: 4023, 123, 123, 123-12.
[...]
atendente: A Natália Dades, tenha um bom dia. Tchau, tchau.
atendente: Por favor, avalie com uma nota, de 1 a 5, o profissional que lhe atendeu.
atendente: Sendo, 1, muito insatisfeito. 2, insatisfeito.
atendente: Obrigado por participar da nossa pesquisa de satisfação. A Naceu agradece a sua atenção.

No exemplo acima, pode-se notar a presença dos problemas mais frequentes nas transcrições, os quais são fundamentais que a etapa de pré-processamento de texto seja

capaz de limpar:

- **Alucinações:** especialmente em trechos de silêncio, ocorrem frequentes transcrições de termos repetitivos e sem significado para o problema, sujando a transcrição e denegrindo a qualidade dos dados. No exemplo, aparece com o uso repetido dos três pontos como sendo a transcrição da fala do cliente.
- **Despadronização de números:** não há padronização na forma em que números são transcritos, pois variam com a forma que a pessoa fala. Esse fato dificulta a etapa de anonimização dos dados. O exemplo ressalta as diferentes formas em que o atendente e o cliente falam o protocolo.
- **Transcrições erradas:** uma falha do modelo de transcrição, em que nomes e outros termos específicos são transcritos de forma equivocada. No exemplo, tem-se o nome Anatel transcrito como “Donatel”, “A Naceu” e “A Natália Dades”.
- **Temporalidade equivocada:** devido à forma em que organiza-se as transcrições do atendente e do cliente (a partir do modelo de Voice Activity Detection), podem haver erros de temporalidade, isto é, falas passadas do cliente passando à frente de falas do atendente.

Observações sobre o Data-Juicer:

Ao ler a documentação do Data-Juicer [6], observa-se a listagem de todas as suas ferramentas disponíveis, ou seja, uma breve descrição daquilo que chamam de Operadores (OPs). Ao analisar cada um dos **cerca de 50 operadores**, foram feitas as seguintes observações:

Legenda:

operadores não usados -> --

operadores usados -> ++

operadores modificados -> +=

pode ser útil em alguma situação ainda não definida -> ?

-> Mappers OPs (aqueles que transformam o texto in loco):

-- clean_email_mapper -> não tenho trechos de e-mail nas transcrições

-- clean_html_mapper -> não tenho documentos html nas transcrições

+= clean_ip_mapper -> não tenho ip's nas transcrições, mas temos cpf's e outros dados sensíveis.

-- clean_links_mapper -> não tenho links que atrapalhem, ou seja, pode aparecer links nas transcrições, mas fazem sentido aparecer.
-- clean_copyright_mapper -> não tenho comentários de copyright nas transcrições
-- expand_macro_mapper -> não há definições macro, pois não há documentos Latex nas transcrições
++ fix_unicode_mapper -> podem haver caracteres unicode problemáticos como resultado de alucinações na transcrição.
+= nlpaug_en_mapper -> esse operador revela processos para fazer data augmentation. Há alguns interessantes que podem servir, mas não no momento atual. A prioridade é executar o pipeline.
? punctuation_normalization_mapper -> normaliza caracteres especiais para inglês.
-- remove_bibliography_mapper -> não há Latex.
-- remove_comments_mapper -> não há Latex.
-- remove_header_mapper -> Não há Latex.
++ remove_long_words_mapper -> pode ser útil para remover palavras grandes.
++ remove_specific_chars_mapper -> pode ser útil para remover caracteres específicos e problemáticos
-- remove_table_text_mapper -> não há tabelas no corpus em questão.
++ remove_words_with_incorrect_substrings_mapper -> remove substrings das palavras. Pode ser útil para algumas situações.
? sentence_split_mapper -> split text to multiple sentences and join them with '\n'
? whitespace_normalization_mapper -> normalize different kinds of whitespaces to English whitespace.

-> Filter OPs (aqueles que filtram os textos):

? alphanumeric_filter
-- average_line_length_filter -> não faz sentido filtrar os textos por quantidade de linhas
++ character_repetition_filter -> MÉTRICA MAIS IMPORTANTE, dado que há muitos casos em que a transcrição repete muito.
-- clip_similarity_filter -> não há imagens no dataset.
++ flagged_words_filter -> pode ser interessante colocar um filtro de palavras a serem removidas, como xingamentos e etc.
-- image_aspect_ratio_filter -> não há imagens sendo trabalhadas.
-- image_shape_filter -> não há imagens.
-- image_size_filter -> não há imagens.
-- language_id_score_filter -> não há texto em outra língua.
-- maximum_line_length_filter -> não há necessidade de filtra texto pelo tamanho das linhas.
? perplexity_filter

-- special_characters_filter -> não há necessidade de filtrar caracteres especiais.
-- stopwords_filter -> não há necessidade de filtrar stopwords.
-- text_length_filter -> não há necessidade de filtrar textos pelo tamanho.
? token_num_filter -> talvez seja interessante filtrar textos por quantidade de tokens.
-- words_num_filter -> não há necessidade de filtrar textos por quantidade de palavras.
++ word_repetition_filter -> muito importante para lidar com a repetição de palavras presentes na transcrição.
-- suffix_filter -> não há necessidade de filtrar textos por sufixos.
? specified_field_filter
-- specified_numeric_field_filter -> não há necessidade de filtrar textos por campos numéricos.

-> Deduplicators OPs (aqueles que removem duplicatas):

Todos funcionam apenas para remover documentos que são repetidos no corpus. Usa-se apenas para garantir que não haverá documentos repetidos no corpus.
-> No entanto, o ideal é garantir que não haja áudios duplicados.

Portanto, apenas 9 operadores foram notados como úteis. Isso revela que, embora haja a solução já pronta, pelo fato de estar confusa a documentação do uso do framework e não ser facilmente integrável com o código que está sendo desenvolvido, foi decidido que as funções dos OPs serão reaproveitadas e utilizadas no código diretamente, sem usar o framework em si, já que são poucos OPs e também alguns serão adaptados.

Também é importante salientar que poucos operadores são pertinentes aos problemas relatados como mais importantes. Sendo que o único problema que o Data-Juicer pode auxiliar é na questão de repetição de palavras pelas alucinações.

Após o estudo do Operador, ele atua apenas contabilizando a quantidade de palavras existentes, removendo aquelas que repetirem acima de um limiar estabelecido arbitrariamente pelo utilizador da ferramenta. Esse tipo de abordagem é extremamente ingênua, não servindo para detectar alucinações, dado que elas costumam aparecer em quantidades significativas e consecutivamente.

Logo, o Data-Juicer serve como fonte de inspiração para elaborar as próprias ferramentas específicas para resolver o problema em questão.

O que foi implementado:

Por enquanto, foi implementado durante a semana 5 um código para que sejam filtradas repetições de palavras, como demonstrado no exemplo 1, dado que é um problema recorrente em transcrições, devido ao modelo Whisper V3 [7] utilizado.

O código pode ser encontrado no apêndice A, ao final deste documento. Abaixo encontra-se a sua explicação:

- O código foi feito em Python;
 - Baseia-se nas premissas de que as repetições de palavras causadas por alucinações ocorrerão de forma equidistante, ou seja, serão várias repetições consecutivas com um número constante de palavras entre elas;
 - Funciona tanto para palavras repetidas quanto para frases repetidas;
- 1) Primeiramente, a string recebida como argumento da função é dividida em palavras pelos espaços e armazenadas em uma lista. Essa forma diferencia palavras que finalizam com pontuações, como “casa”, “casa.” e “casa,”, fato que não atrapalha, pois as repetições serão sempre com as mesmas palavras.
 - 2) Em seguida, um laço percorre todas as palavras armazenadas na lista e as organiza em um dicionário, em que cada palavra é uma chave e os itens armazenados por essas chaves são os índices em que elas aparecem na lista de palavras. Exemplo: “Eu amo muito sorvete. Ela ama muito sorvete.” -> [Eu, amo, muito, sorvete., Ela, ama, muito, sorvete.] -> {“Eu”:[0], “amo”:[1], “muito”:[2, 6], “sorvete.”:[3, 7], “Ela”:[4], “ama”:[5]}.
Dessa forma, armazena-se as palavras e suas repetições em uma estrutura de dados simples.
 - 3) Após isso, outro laço (não aninhado com o primeiro) percorre as chaves dos dicionário, verificando quais palavras repetem um número arbitrário de vezes (uma pré-seleção para não ter que calcular se as repetições são equidistantes para todas as palavras do dicionário). Caso a palavra apareça acima do limiar determinado, ela seguirá para o próximo passo;
 - 4) Cada palavra selecionada passa por um laço que percorre a lista com os índices de suas repetições na string, calculando a distância da repetição K com a K - 1 e armazenando o resultado em um dicionário auxiliar, em que as chaves dele são as distâncias calculadas. Basicamente, ao final do laço, ter-se-á a relação de repetições equidistantes, ou seja, basta buscar nesse dicionário as chaves (distâncias) que contém mais do que N arbitrárias repetições equidistantes, contando o tamanho da lista de índices armazenadas em cada chave do dicionário;

- 5) As palavras repetidas no dicionário mais que N vezes são então substituídas na lista de palavras por "" (vazio);
- 6) Por fim, a lista é convertida para string novamente, que é retornada pela função.

Exemplo de execução:

Entrada: "Isso é um teste é um teste é um teste é um teste é um teste de equidistante de repetições em um teste."

Saída: "Isso de equidistante de repetições em um teste."

#####

Entrada:

cliente: Mas é só internet mesmo

atendente: Certo.

atendente: Um momento.

cliente: Ok e aí e aí e aí e aí e aí e aí e aí e aí e aí e aí e aí e aí e aí e aí e aí e aí e aí e aí e aí e
quando a minha linha não tem problema botar na minha recarga vinha falava o mês todo chegava no final do
mês botava de novo não tinha problema acabou a oi migrou para claro tá horrível

cliente: aí fica pedindo para poder fazer recarga mais recarga

Saída:

cliente: Mas é só internet mesmo

atendente: Certo.

atendente: Um momento.

cliente: Ok quando a minha linha não tem problema botar na minha recarga vinha falava o mês todo chegava no
final do mês botava de novo não tinha problema acabou a oi migrou para claro tá horrível cliente: aí fica pedindo
para poder fazer recarga mais recarga

Quanto ao processo de busca por informações:

Durante a semana 5, foi feita a leitura do artigo de ABBASIAN et al. [8], o qual visa revisar as diferentes formas de se buscar conhecimento em texto, tanto na visão clássica (estatística e com regras bem definidas) quanto na perspectiva do uso de algoritmos de Deep Learning. O artigo foca em mostrar e explicar diferentes algoritmos e seus usos.

No entanto, para o presente trabalho, busca-se encontrar formas de, a partir do texto caótico, fora de ordem, apresentado no exemplo 1, buscar informações relevantes a partir da demanda do cliente e a resposta ofertada pelo atendente. Essa perspectiva não é

contemplada no trabalho de ABBASIAN TAEB, mas a seguinte figura contemplada em seu trabalho permite ideias interessantes sobre como pode ser feita a arquitetura da busca de informações:

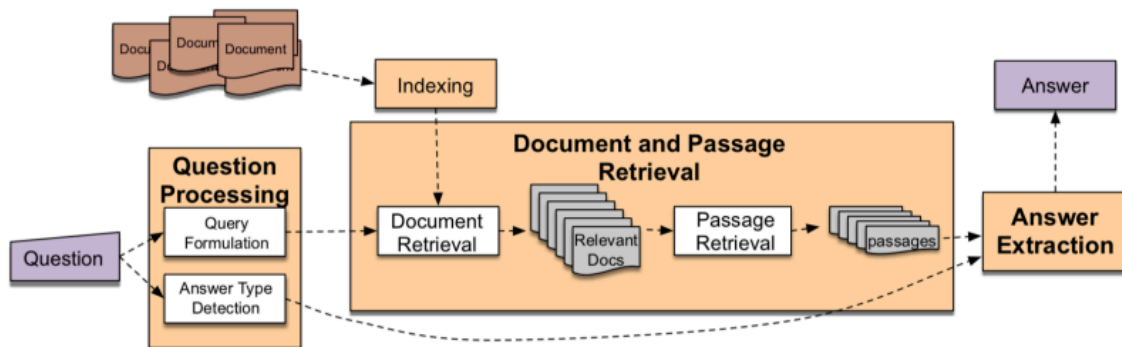


Figura 1: Arquitetura geral de sistemas Question & Answer baseados em texto.

A partir da figura 1, nota-se a ideia de se buscar, dentre os documentos presentes no corpus, aqueles mais semanticamente próximos à pergunta proposta. Além da busca por documentos próximos, cada documento é filetado, extraindo suas passagens ou parágrafos, a fim de se encontrar os segmentos do texto mais próximos à pergunta do cliente.

Portanto, uma forma de se executar a ordenação do pipeline é justamente identificar as demandas dos clientes, usá-las como a “question” no sistema, usar como corpus todas as informações prestadas pelo atendente, em todas as ligações armazenadas, buscando as respostas mais próximas à pergunta separada. Após conseguir identificar as demandas de clientes e sincronizar com N possíveis respostas/soluções, basta formatar essa pergunta no formato adequado para a execução do fine-tuning de um LLM para a criação do sistema de Q&A sem depender da arquitetura da figura 1, em produção.

ATUALIZAÇÃO SEMANA 6

Implementação dos operadores baseados na ferramenta Data-Juicer [6]:

Conforme foi explicado na seção “Observações sobre o Data-Juicer”, poucos operadores foram identificados, de antemão, como aplicáveis para resolver os problemas apresentados neste trabalho.

Após a leitura dos operadores, foram identificados 9 como sendo pertinentes, mas ao todo 15 foram avaliados para o devido uso no pré-processamento das transcrições de ligações, sendo estes:

- **clean_ip_mapper**: remove endereços de IPv4 e IPv6 de textos. Foi descartado pelo fato de apenas usar regex para realizar a limpeza. Não há formas de se aproveitar por não partilhar desse problema, tampouco usar regex para possível remoção de CPF's, números de telefone e outros dados, pois não há um padrão definido em como aparecem.
- **fix_unicode_mapper**: foi implementado na solução, a fim de garantir que não haja caracteres defeituosos resultantes da transcrição.
- **nlpaug_en_mapper**: embora seja interessante realizar data augmentation para o treino de LLM's ser mais robusto, as palavras já são escritas de diferentes formas por causa da transcrição de áudio, ou seja, diferentes nomes e palavras vêm de formas diferentes.
- **punctuation_normalization_mapper**: foi implementado na solução, a fim de garantir que as pontuações sejam as normais utilizadas. (Existem variações para pontuações, por exemplo \hat e $^$, ambas representam acento circunflexo).
- **remove_long_words_mapper**: apenas remove palavras grandes. Não faz sentido para este trabalho, uma vez que não há problemas com o tamanho das palavras, apenas a repetição delas.
- **remove_specific_chars_mapper**: esse operador pode ser implementado. No entanto, ainda não foi notado uso relevante para ela, uma vez que não há problemas detectados com caracteres específicos.
- **remove_words_with_incorrect_substrings_mapper**: não foi notado nenhum problema de substring no contexto deste trabalho.
- **sentence_split_mapper**: esse operador faz sentido, pois uma etapa futura é encontrar segmentos do texto transcrito pelo atendente para utilizar como resposta para o cliente, o que necessita que o texto esteja dividido em sentenças. No entanto, a transcrição já retorna segmentos de texto detectados conforme a pessoa fala.
- **whitespace_normalization_mapper**: apenas transforma diferentes tipos de espaço para o espaço comum utilizado no Inglês. Pode ser implementado a fim de garantir que os espaços sejam os padrões.
- **alphanumeric_filter**: esse operador apenas filtra os textos baseado na taxa de caracteres que são número em relação aos caracteres que são letras. Não há necessidade.

- **character_repetition_filter**: basicamente, esse operador filtra as repetições de caracteres por n-gram. Utilizei-o como base para implementar o filtro criado na semana 5.
- **flagged_words_filter**: não há palavras específicas que devem ser removidas.
- **perplexity_filter**: os modelos não foram treinados ainda para avaliar perplexidade de sentenças. Logo, não há porquê remover sentenças a partir de sua perplexidade.
- **token_num_filter**: apenas filtra a quantidade de tokens, o que não é necessário.
- **word_repetition_filter**: também foi utilizado como base da implementação do filtro próprio.

Novos operadores implementados, além dos presentes no Data-Juicer:

Conforme explicado no início da atualização da semana 5, uma das preocupações deste trabalho é com a privacidade dos dados, uma vez que o texto é extraído da transcrição de ligações em ambiente de call-center, o qual contém diversos dados sensíveis como nomes, CPF's, endereços e números telefônicos.

A maior preocupação está em desidentificar as transcrições, isto é, remover nomes, CPF's e qualquer outro dado que possa identificar as pessoas.

Portanto, foram construídos alguns operadores que realizam a remoção de nomes e CPF's:

remove_nomes:

Para esse operador, trabalhou-se a questão de Named Entity Recognition (NER), isto é, por meio de técnicas de NLP, tentou-se estabelecer a classe que as palavras constituintes do texto pertencem. Para isso, foi utilizada a biblioteca spaCy [9], a qual permite a análise de textos e retorna as entidades reconhecidas presentes.

No entanto, houveram problemas em que o classificador identificava outros termos, como "Claro", "Anatel" como sendo nomes de pessoas, além de também não identificar nomes incomuns como sendo pessoas.

Logo, essa abordagem foi abandonada, mas com ciência de que ela pode ser melhorada realizando análise sintática do texto antes de estabelecer as classificações de fato, como pode ser visto no artigo Medium *Named Entity Recognition with NLTK and SpaCy* [10].

Em seguida, adotou-se uma forma de remoção *naive* para o problema, a qual consiste em utilizar um levantamento de nomes feito pelo IBGE no censo de 2010 [11] e

crawlado, isto é, obtido por Bernardo Loureiro e disponibilizado em sua conta GitHub em [12] e verificar se algum desses nomes estão contidos no texto a ser analisado. Caso esteja, substitui-se o nome pelo marcador NOME.

São listados os 10 mil nomes mais comuns de homens e 10 mil nomes mais comuns de mulheres. A lista contém tantos nomes que algumas palavras acabam sendo removidas, por também serem nomes, como a palavra "não" acabava sendo confundida com o nome "Nao".

De tal forma, apenas os 500 (valor arbitrário) primeiros nomes mais comuns foram utilizados.

remove_CPFs:

Para esse operador, foi preciso lidar com o seguinte problema: os números não aparecem com algum formato padrão, sendo que podem conter palavras no meio, serem transcritos por extenso, caracteres como reticências separando-os ou ainda virem tudo junto normalmente. Por isso, a forma de lidar foi simples e direta: remover todo e qualquer número que aparecer no texto.

Essa saída baseia-se no fato de não haver informações relevantes em números, mas sim na história e problema que o cliente está enfrentando e o que foi feito para resolver os problemas e sanar as dúvidas desse cliente.

Exemplos de textos antes e após limpeza:

Exemplo 1:

Antes do pré-processamento:

E aí Bom dia, Liane. É Maria José. Eu já falei ainda há pouco com a Valéria, e eu já fui pegar o número do CPF. É o seguinte, não sei se você passa pra ela ou se eu continuo com você. Tá. É o seguinte, eu tenho colocado recarga no meu celular e quando eu recebo a ligação é descontado crédito. Então eu liguei lá na central da Claro e eles me falaram que eu fiz um plano onde toda ligação que eu receber fora do estado, ela será descontada. E eu não tinha esse plano. O meu celular era normal. Onde eu estivesse, eu fazia uma ligação e tudo bem. Eu que pagava pela ligação. E segundo a informação que eu tive do atendente, eu vou pagar pela ligação que eu faço e pela ligação que eu recebo. Então, eu não entendi como que funciona isso. É. Desde o dia 21, está acontecendo isso. Eu recebi uma ligação. Eles descontaram. Nos meus créditos. Claro. Eu estou em Minas. Mas o meu DDD desse celular é de São Paulo, é o 17. Minas Gerais. É, eu não fiz isso. Eu vou pagar para receber a chamada e pagar para fazer a ligada também? Nunca vi. Bré paga? Bré paga? 25-9-93-55-66-77 9, 93, 55, 66, 77. Meu ou do titular? Fulano Ciclano Sobrenome Beltrano. 111-222-33344 Não entendi. Maria José. Daqui? Ah, tá. Então, vou dar do meu lá de onde eu moro. 11-222-333. Pode ser esse mesmo, né? Porque eu não tenho outro aqui no momento. Do Leandro, eu não sei. Uhum. Isso. Pré. Não, agora tem uma recarga que eu fiz ontem também, 8 horas, no valor de 20 reais, que já foram todos descontados. Ah, tá. Tá certo. Então, tá bom. Eu tenho 11, 22, 33, 44, 55, 6, 7, 8. 11-222. Santana de Cataguase. Só que com esse protocolo, eu não consigo falar na ouvidoria, não abre atendimento, entendeu? Entendeu? Vou anotar. 11, 22. 33, 4. Não, muito obrigada. Tá bom, muito obrigada. Muito obrigada. Aguardo o senhor. Obrigada.

Após o pré-processamento:

que possa ser tomada para vivo? Depois eles falam que resta... Porque, olha, eles falam que resta assim. Sabe o que eles restam assim? Um centavo, dois centavos, cinco centavos. Isso aí paga meu prejuízo? Não paga, gente. A conta já veio para pagar normalzinha, como se eu estivesse usando o telefone normal. Como é que pode? Por favor, por favor. E aí Ó, essa 6069 eles conseguiram arrumar. É a única que está funcionando. As outras três não estão funcionando. Tá bom? Por favor. São Paulo. São Paulo. São Paulo. Aí tem outro protocolo, é isso? E aí Por favor. Por favor. 7, 7, 4. Tá, como é teu nome e não obrigada faz mesmo e bota aí ó E aí E aí

Após o pré-processamento: E aí Zé. Zé. Já. Deixa eu te falar, eu já abri essa reclamação com vocês. É. O da Embratel, não é isso? É. O da Embratel, não é isso? É, o Isso É sim, foi o que eles me passaram. No dia de , que eu falei com o NOME. É, o ---. NOME, você já foi trabalhada aí? aí, não aguenta mais, não mais nessa, eu vivo não, NOME. E tem mais o que fazer com essa vida não, viu? E aí Ok. NOME NOME de Andrade. Você quer o celular ou fixo? --. Não entendi, moça. Pode ser, porque você é direto em contato comigo, é mais fácil. Do celular? --. Não, não precisa, não. . Ah, é . Moça, nós estamos sem telefone desde o dia de maio nós estamos em de junho, mais de dias eu acho que tem que tomar uma providência com a vida quem vai pagar meu prejuízo? nós dependemos desse telefone para trabalhar a gente liga horas para eles eles mandam alguém vir aqui que não sabe de nada, chega aqui e fala eu tenho que falar com o supervisor. Vai embora e não faz nada. dias que eu estou sem as linhas de telefone para trabalhar. dias, quem vai pagar meu prejuízo? No mínimo, botando por baixo, é uma faixa de R\$. por dia de prejuízo que eu estou tendo por causa da Vivo. Não tem nenhuma providência que precisa ser tomada, que possa ser tomada para vivo? Depois eles falam que resta... Porque, olha, eles falam que resta assim. Sabe o que eles restam assim? Um centavo, dois centavos, cinco centavos. Isso aí paga meu prejuízo? Não paga, gente. A conta já veio para pagar normalzinha, como se eu estivesse usando o telefone normal. Como é que pode? Por favor, por favor. E aí Ó, essa eles conseguiram arrumar. É a única que está funcionando. As outras três não estão funcionando. Tá bom? Por favor. São NOME. São NOME. São NOME. Aí tem outro protocolo, é isso? E aí Por favor. Por favor. , , . Tá, como é teu nome e não obrigada faz mesmo e bota aí ó

Como pode notar, há alguns problemas ainda com essas abordagens, como valores monetários e nomes de cidades sendo apagados, sobrenomes sendo mantidos e alguns nomes podendo manterem-se.

Identificando as demandas a partir da transcrição do cliente:

Para essa tarefa, os textos já pré-processados das falas dos clientes são submetidos ao ChatGPT [13], a fim de reescrever esse texto explicitando a sua demanda, ou seja, o motivo pelo qual o cliente está ligando para o call-center e expor de uma maneira objetiva e direta, sem as típicas marcas de oralidade, como ser prolixo e repetir termos desnecessariamente.

Para isso, estabeleceu-se o seguinte prompt, de modo a explicitar ao LLM qual sua função e objetivo, além de expor exemplos (few shots) demonstrando como deve ser feito o processo:

Prompt:

Assuma o papel de um cliente ligando a um call center para tirar dúvidas sobre algum serviço contratado que apresenta problemas.

Você receberá a transcrição do que um cliente falou durante seu atendimento (por ser automática, podem haver erros de transcrição! Corrija-os quando puder!)

Seu objetivo é reescrever a demanda do cliente, isto é, entender qual foi a motivação do cliente para ter feito a ligação e reescrevê-la de forma objetiva na primeira pessoa, como se você fosse o cliente dizendo o que deseja. Siga os passos abaixo e verifique os exemplos para realizar isso:

1) Verifique se a transcrição contém uma demanda, isto é, uma dúvida ou reclamação válida, que possa ser esclarecida por um agente. Se não houver, RETORNE apenas a palavra "INVALID" e ignore os próximos passos. Exemplo válido: "Pode alterar. Da internet. Na fixa. Hã? Deixa eu dar uma olhada aqui, só um minutinho. . A internet não está funcionando desde sexta-feira. Eu já tentei contato com eles, marquei suporte técnico e eles não vêm. Eles concluem o serviço e não vêm. Botam no aplicativo como se fosse concluído o serviço e não aparece."

Exemplo inválido: "Vivo. Vivo. Como? E que isso? Ah, você quer o meu telefone? Ah, você quer o meu telefone? Já, já, já, nós já fizemos tudo, já fizemos tudo, já. O seu colo tem"

2) Reescreva a transcrição se passando pelo cliente, ou seja, na primeira pessoa, sendo objetivo e removendo ruídos (marcas de oralidade, texto prolixo...), mantendo a essência da dúvida do cliente para que possa ser respondida por um atendente. Isto é, resuma a dúvida em uma pergunta que possa ser respondida por um agente.

Exemplo original: "Isso mesmo, o valor de R\$ 59,90, o plano que eu fiz foi por os seis primeiros meses. Eu paguei somente uma fatura de R\$ 59,90. 6. 6 primeiros. Já fazem duas semanas que eu estou esperando e a minha fatura venceu. O meu sinal vai cortar e eles não ajustaram minha fatura para poder pagar"

Exemplo reescrito: "Combinei com a empresa de telefonia que deveria pagar R\$ 59,90 nos primeiros 6 meses, no entanto a fatura está vindo com valor fora do combinado em contrato. O que fazer?"

3) Retorne APENAS a demanda reescrita, nada além disso.

Resultados obtidos:

Exemplo 1:

Fala cliente:

E aí Bom dia, NOME. Tudo bem? Meu nome é NOME. Bom dia, NOME. Tudo bem? Meu nome é NOME. Meu cara, é o seguinte. Eu tenho atualmente, na verdade não eu, na minha mãe, ela tem um plano Moza da TIM. Ela já tinha esse plano há um determinado período e não tinha fidelidade. Porém, ela me trouxe em contato com a TIM para poder verificar se tinha condição de fazer alguma negociação para poder alterar o plano, para melhorar os benefícios dela. E a gente sabe que quando tem esses movimentos de mudança de plano, a operadora geralmente renova a fidelidade. Isso foi feito. Porém, hoje a gente recebeu uma proposta da Viva, e nós gostamos, e a gente está desejando fazer essa portabilidade de aviso. Sendo que ontem a gente já fez uma mudança de plano na TIM. E pelo que eu tenho conhecimento, geralmente quando o cliente faz alguma contratação de serviço ou mudança de plano para serviços de internet residencial, tem um período de teste do serviço, onde o cliente, por mais que venha a ter iniciado um novo contrato, ele pode usar, se eu não me engano, por sete dias, e dentro desse período pode cancelar. Aí eu queria saber se isso também vale para serviços de linha móvel, porque aí se for o caso, se valer, aí eu consigo fazer essa portabilidade para mim você

necessariamente tem que pagar uma multa você pode me dar alguma informação a respeito e e o que é uma renovação Eu acho que nós temos dois pontos. Sim. Está no meu modo. Está em amor. E aí Eu agradeço se você me informar, por favor. Isso. Obrigado. Não, só isso mesmo. Obrigado.

Fala reescrita: Minha mãe possui um plano Moza da TIM e queremos fazer uma portabilidade para a Viva. Gostaria de saber se, ao fazer essa mudança de plano, ainda temos o período de teste de 7 dias para cancelar, como acontece com os serviços de internet residencial. Além disso, gostaria de saber se haverá alguma multa ou renovação de fidelidade ao fazer essa portabilidade. Agradeço se puder me informar sobre essas questões. Obrigado.

Exemplo 2:

Fala cliente: E aí É o gênio, né, NOME? É, Eugênia, é porque eu queria saber, estão me ligando constantemente de um número fazendo cobrança, sabe? E não é a primeira vez já tem vários meses sempre me ligando por tal de NOME eu não sou NOME, não sei quem é NOME não conheço nem o NOME não tenho nenhum parente chamado NOME e ficam me ligando direto desse número. O que eu faço para bloquear isso? Me ligam de vários números. Eu já fiz esse cadastro. Tá. Tá. pronto, tá bom só isso vocês através do número vocês não conseguem identificar que empresa é essa não é só ligar pra lá e tá tá bom tá bom, obrigado não né só ligar para lá e tá tá bom obrigado não só isso obrigado e aí E aí E aí Obrigada.

Fala reescrita: Recebo constantemente ligações de um número fazendo cobranças para uma pessoa chamada "NOME", mas eu não conheço essa pessoa e não tenho nenhum parente com esse nome. Como posso bloquear essas ligações? Já fiz o cadastro de bloqueio, mas continuam ligando de vários números diferentes. Vocês conseguem identificar de qual empresa são essas ligações? Obrigado.

Apêndice A - Código Python de filtro de repetições em strings.

```
def filtra_repeticoes_equidistantes(frase: str) -> str:
    """
    Encontra as repetições de palavras que são equidistantes em uma
    frase. Para esse filtro funcionar, ele parte
    do pressuposto que as repetições de palavras que são
    equidistantes são alucinações do Whisper.
    :param frase: frase a ser analisada.
    :return: frase sem as repetições equidistantes.
    """
    palavras = frase.split()
    repeticoes = {}

    # criando um dicionário com as palavras e seus respectivos
    índices
    for i, palavra in enumerate(palavras):
        if palavra not in repeticoes:
            repeticoes[palavra] = [i]
```

```
    else:
        repeticoes[palavra].append(i)

# instanciando a lista que receberá os índices das palavras que
# se repetem equidistantemente
resultado = []

# iterando sobre o dicionário de repetições, selecionando apenas
# as palavras que se repetem pelo menos 4 vezes
for palavra, indices in repeticoes.items():
    distancias_repeticoes = {}
    if len(indices) >= 4: # selecionando apenas as palavras que
# se repetem pelo menos 4 vezes
        # verificando a distância entre as repetições. Se for
# equidistante, será adicionado na lista de resultados
        for i in range(1, len(indices)):
            dist = indices[i] - indices[i - 1] # calculando a
# distância entre as repetições
            if dist not in distancias_repeticoes:
                distancias_repeticoes[dist] = [indices[i - 1],
indices[i]]
            else:
                distancias_repeticoes[dist].append(indices[i -
1])
                distancias_repeticoes[dist].append(indices[i])

        # verificando quantas repetições equidistantes tem para
# cada distância
        for _, indices in distancias_repeticoes.items():
            if len(set(indices)) >= 5: # arbitrário, limiar de
# repetições equidistantes para determinar se é uma alucinação
                resultado += list(set(indices))

# filtrando as palavras que se repetem equidistantemente
for idx in resultado:
    palavras[idx] = ""
palavras = [palavra for palavra in palavras if palavra != ""]

return " ".join(palavras)
```

Referências:

- [1] ZHAO, W. X. et al. **A Survey of Large Language Models**. arXiv (DOI: 2303.18223), 31 mar. 2023.
- [2] CHEN, D. et al. **Data-Juicer: A One-Stop Data Processing System for Large Language Models**. Disponível em: <<https://arxiv.org/abs/2309.02033>>. Acesso em: 30 nov. 2023.
- [3] SHOEYBI, M. et al. 2019. **Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism**. CoRR abs/1909.08053 (2019).
- [4] RASLEY, J. et al. 2020. **Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters**. In KDD. 3505–3506.
- [5] WOLF, T. et al. 2020. **Transformers: State-of-the-Art Natural Language Processing**. In EMNLP (Demos). 38–45
- [6] **Data-Juicer: A One-Stop Data Processing System for Large Language Models**. Disponível em: <<https://github.com/alibaba/data-juicer>>. Acesso em: 30 nov. 2023.
- [7] RADFORD, A. et al. **Robust Speech Recognition via Large-Scale Weak Supervision**. Disponível em <arxiv.org/abs/2212.04356>. 6 dez. 2022.
- [8] ABBASIANAEV, Z.; MOMTAZI, S. **Text-based Question Answering from Information Retrieval and Deep Neural Network Perspectives: A Survey**. Disponível em: <<https://arxiv.org/abs/2002.06612>>. Acesso em: 7 dez. 2023.
- [9] Models · **spaCy Models Documentation**. Disponível em: <<https://spacy.io/models>>. Acesso em 12/12/2023
- [10] LI, S. **Named Entity Recognition with NLTK and SpaCy**. Disponível em: <<https://towardsdatascience.com/named-entity-recognition-with-nltk-and-spacy-8c4a7d88e7da>>. Acesso em 12/12/2023
- [11] IBGE. **Nomes no Brasil**. Disponível em: <<https://censo2010.ibge.gov.br/nomes/#/search>>. Acesso em: 14 dez. 2023.

[12] LOUREIRO, B. **MedidaSP/nomes-brasileiros-ibge**. Disponível em:
<<https://github.com/MedidaSP/nomes-brasileiros-ibge/tree/master>>. Acesso em: 14 dez.
2023.

APÊNDICE 5

Termo de Aceite de Entrega - Semana 9

Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

Data da Reunião (“gate”) de aprovação: 21 de dez. de 2023

Participantes da Entrega [matriculados em Residência em IA]:

Alex Echeverria

Entrega: [descrever a ENTREGA: requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

Recapitulando:

- Este trabalho propõe a elaboração de um pipeline que consome gravações de áudio de ligações em um ambiente de call center de modo a gerar datasets prontos para Instruct Fine Tuning de LLM's, automaticamente.

O que foi planejado para essa semana:

- **Finalizar a implementação da busca por respostas às demandas encontradas / reescritas.**
- **Implementar o formatador de instâncias para Instruct Fine Tuning.**
- **Iniciar a implementação do treino de LLM's a partir dos datasets gerados.**

O que foi desenvolvido:

- Os avanços estão descritos, detalhadamente, no seguinte documento:
[DocumentaçãoSemana7-AutoInstruct4Call](#)

- **Melhorias na Detecção de Demandas:**

Na última entrega, criou-se um prompt para analisar transcrições de ligações e detectar as demandas de forma automática a partir da transcrição do cliente. Inicialmente, muitas transcrições foram classificadas como inválidas, mas após a implementação de um novo prompt, todas as 40 amostras testadas foram reescritas com sucesso, sugerindo a preferência por reescrever transcrições ruidosas ao invés de descartá-las. Exemplifica-se com uma transcrição reescrita de maneira satisfatória, destacando a eficácia do novo método em extrair informações úteis de transcrições inicialmente consideradas inválidas.

- **Armazenamento das Demandas e dos Candidatos a Solução:**

Após a extração das demandas dos clientes de aproximadamente 3 mil transcrições, iniciou-se o armazenamento dessas informações em um banco de dados para facilitar a busca de respostas. O

conceito de Retrieval Augmented Generation (RAG) é empregado para comparar as demandas dos clientes com as respostas dos atendentes através de vetores multidimensionais, buscando otimizar a correspondência entre demandas e soluções. Ainda está em aberto a escolha da melhor opção de banco de dados para armazenar e buscar esses vetores.

- **Leitura do Artigo "Finetuned Language Models are Zero-shot Learners":**

O artigo analisa a eficácia dos Modelos de Linguagem de Grande Escala (LLMs) como aprendizes de zero-shot e few-shot. Revela que os LLMs têm limitações como especialistas em tópicos sem exemplos ou conhecimento prévio, mas propõe um método para aprimorar sua capacidade de processar demandas em linguagem natural. Esse método envolve a criação manual de templates que instruem o LLM sobre como processar dados específicos. A relevância do artigo para o projeto atual reside na estruturação de dados de transcrições e respostas de atendentes para treinar LLMs para responder automaticamente às demandas dos clientes.

Planejamento: [descrever o que pretende fazer para realizar a próxima ENTREGA]

- Implementar o formatador de instâncias para Instruct Fine Tuning.
- Iniciar a implementação do treino de LLM's a partir dos datasets gerados.
- Levantar os resultados obtidos e melhorias.

Observação: [caso precise fazer alguma observação, de qualquer "natureza"]

Essa semana foi menos entregas que o normal, pois havia entrega final de um projeto CEIA que demandou mais atenção que o esperado.

ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: **Go!** ▾

LUANA GUEDES BARROS MARTINS: **Go!** ▾

Documentação Semana 7: Detecção de demandas automaticamente e leitura do artigo base para a formatação do dataset.

Melhorias na detecção de Demandas

Na última semana, finalizou-se o processo de pré-processamento de texto e iniciou-se o processo de busca por demandas do cliente, isto é, transcrever, a partir da ligação, qual a dúvida, reclamação, sugestão ou solicitação da pessoa que está ligando ao call center.

Para isso, lançou-se mão do uso da ferramenta ChatGPT [1], que é um Large Language Model extremamente capaz de analisar textos complexos, para que realize a análise das transcrições (já pré-processadas, com a devida anonimização) e extraia a demanda do usuário.

O prompt utilizado foi construído na semana passada, mas, após verificar os resultados emitidos pelo ChatGPT, percebeu-se que muitas transcrições foram identificadas como inválidas, ou seja, como não fosse possível extrair as informações a partir da transcrição devido a algum problema (o prompt pode ser visto no Apêndice A). Após fornecer um conjunto de validação com 40 transcrições aleatoriamente escolhidas da população de cerca de 3 mil ligações disponíveis, testou-se o prompt antigo (ANTIGO) e um novo prompt construído (NOVO), qual diferença consiste em remover a solicitação de identificar transcrições inválidas, sempre resumindo a demanda, independente qual seja (veja no Apêndice A o novo prompt).

Das 40 amostras fornecidas, ANTIGO detectou que 19 eram transcrições problemáticas, retornando a resposta "INVALID". Já o novo prompt reescreveu todas as 40 transcrições. Isso se torna um problema para avaliar, dado que existem ligações que nada mais são do que ruídos. No entanto, ainda prefere-se a reescrita da transcrição ruidosa, podendo ser removida por alguma ferramenta posteriormente, a perder diversas transcrições por o ChatGPT identificar como inválida.

Abaixo, mostra-se um exemplo de transcrição declarada inválida pelo ANTIGO, mas ainda assim foi reescrita de forma satisfatória pelo NOVO:

<transcription>

Obrigado. Boa tarde, Minata. Você fala com a NOME. Fica lá, mano. Vivo. Vivo. Como? E que isso? Ah, você quer o meu telefone? Ah, você quer o meu telefone? Já, já, já, nós já fizemos tudo, já fizemos tudo, já. O seu colo tem , , , , e passou esse protocolo. Por quê? Isso. Isso. Aqui é uma empresa e nós estamos sem telefone há mais de uma semana. Isso. Então, quer dizer que com esse protocolo eu não consigo registrar reclamação. posicionamento que nós já tivemos já que hoje o moço veio até aqui e falou que não tem prazo para ele regularizar a criação. ...

</transcription>

Reescrita ANTIGO: “INVALID”

Reescrita NOVO: “Boa tarde, meu nome é NOME e sou cliente da Vivo. Estou sem telefone há mais de uma semana e já entrei em contato com a empresa, mas não consigo registrar uma reclamação com o protocolo que me foi fornecido. O que posso fazer para resolver essa situação?”

Como pode-se perceber, ainda havia como resgatar informações dessa transcrição, mas o ANTIGO falhou em cumprir seu objetivo. Uma explicação plausível é a subjetividade em determinar a invalidade de uma transcrição, reforçado pela presença de ruído, como a remoção do protocolo do trecho, sugerindo um defeito e, portanto, a invalidade dela.

Logo, determinou-se que NOVO será utilizado para extrair as demandas de todas as transcrições disponíveis.

Armazenamento das demandas e dos candidatos a solução:

Após todas as cerca de 3 mil transcrições terem as demandas dos clientes extraídas, iniciou-se o processo de armazenamento dessas informações em um banco de dados para realizar buscas.

A ideia é: a partir das demandas extraídas dos usuários, realiza-se uma busca em todas as falas de atendentes disponíveis na base de dados, isto é, não só as falas do atendente da ligação do usuário como todas as de todas as outras ligações existentes serão buscadas.

Com isso, surge a necessidade de alguns pontos: como armazenar essas informações e como realizar busca de textos a partir de outro texto.

Ao pesquisar a literatura, o conceito de Retrieval Augmented Generation (RAG) mostra-se pertinente para solucionar a necessidade de busca de texto para texto. Como pode ser observado em [\[2005.11401v4\] Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks \(arxiv.org\)](#) [2], a busca textual é feita a partir da comparação do *encoding* (ou codificação) das informações presentes no texto de *query*, no caso a demanda do usuário, com o *encoding* dos textos candidatos a respostas (as falas dos atendentes, no presente trabalho). Esses *encodings* são do que vetores representados em um espaço multidimensional \mathbb{R}^n de tal forma que é possível realizar comparações matemáticas de similaridade entre dois ou mais vetores. Portanto, a comparação dita anteriormente é feita a partir do cálculo de similaridade de cosseno, mas outras métricas de distância de vetores é válido também.

Esses vetores denominam-se embeddings a partir da forma que são gerados. Isto é, para texto, utiliza-se um modelo de linguagem para processar as informações e, a partir das ativações das camadas internas dos modelos, gera-se uma representação vetorial, de tamanho fixo, gerada pelo modelo de linguagem treinado. Essas representações conseguem diferenciar, no espaço vetorial a que estão inseridas, o contexto e a semântica do texto

processado. No caso presente, utiliza-se o modelo “text-embedding-ada-002”, modelo da OpenAI [3].

Para armazenar as transcrições e esses vetores que as representam, deve-se utilizar um banco de dados capaz de lidar com esses vetores e realizar buscas a partir deles, de tal forma que, a partir de um vetor *query*, retorna os N vetores mais semelhantes.

Ainda não foi definido a melhor opção.

Leitura do artigo *Finetuned Language Models are Zero-shot Learners*

O artigo *Finetuned Language Models are Zero-shot Learners* [4] foi utilizado como base na etapa de formatação do dataset gerado a partir da identificação de demandas dos usuários e busca pelas opções de resposta mais interessantes. Para isso, a leitura dele é fundamental para a adequada formatação do texto que será fornecido ao LLM em seu Instruct Fine-tuning.

Abaixo, há considerações pontuais e importantes sobre o artigo, além da reflexão de sua pertinência ao presente trabalho:

- O artigo explora a questão de que LLM's funcionam bem quando usados como few-shot learners, ou seja, quando, em seu treinamento ou no prompt, são passados exemplos semelhantes com a forma esperada que ele execute uma tarefa ou retorne as informações processadas;
- Também salienta que os LLM's não funcionam bem como zero-shot learners, ou seja, como especialistas em um assunto. Pedir para executar uma tarefa sem passar exemplos ou fontes de conhecimento, eles não se sairão bem, de acordo com os benchmarks explorados;
- O paper visa explorar um método para melhorar a capacidade de LLM's atuarem como zero-shot Learners, baseando-se na intuição de que tarefas de NLP podem ser descritas via instruções em linguagem natural. Isto é, podem ser descritas à máquina a forma de executar as tarefas ou mesmo uma descrição da tarefa que deve ser feita, para que ela aprenda as nuances da forma que humanos dão ordens.
- A ideia central do artigo é melhorar a forma que LLM's lidam com demandas feitas em linguagem natural pelos humanos, o que revela a proximidade com o presente trabalho, em que o objetivo é estruturar dados obtidos automaticamente a partir das transcrições de demandas de clientes e possíveis respostas de atendentes de tal

forma que viabiliza o treino de LLM's para responder a essas demandas de forma automática.

- Basicamente, a forma com que eles melhoram o dataset é: a partir da tarefa que o dataset possui, manualmente eles criam um número arbitrário de templates que recebem os dados do dataset, explicando em linguagem natural como aqueles dados devem ser trabalhados pelo modelo. Os templates são selecionados aleatoriamente para cada instância do dataset.

Apêndice A - Prompts usados para extração de demanda:

Prompt antigo:

Assuma o papel de um cliente ligando a um call center para tirar dúvidas sobre algum serviço contratado que apresenta problemas.

Você receberá a transcrição do que um cliente falou durante seu atendimento (por ser automática, podem haver erros de transcrição! Corrija-os quando puder!)

Seu objetivo é reescrever a demanda do cliente, isto é, entender qual foi a motivação do cliente para ter feito a ligação e reescrevê-la de forma objetiva na primeira pessoa, como se você fosse o cliente dizendo o que deseja.

Siga os passos abaixo e verifique os exemplos para realizar isso:

1) Verifique se a transcrição contém uma demanda, isto é, uma dúvida ou reclamação válida, que possa ser esclarecida por um agente. Se não houver, RETORNE apenas a palavra "INVALID" e ignore os próximos passos. Exemplo válido: "Pode alterar. Da internet. Na fixa. Hã? Deixa eu dar uma olhada aqui, só um minutinho. . A internet não está funcionando desde sexta-feira. Eu já tentei contato com eles, marquei suporte técnico e eles não vêm. Eles concluem o serviço e não vêm. Botam no aplicativo como se fosse concluído o serviço e não aparece."

Exemplo inválido: "Vivo. Vivo. Como? E que isso? Ah, você quer o meu telefone? Ah, você quer o meu telefone? Já, já, já, nós já fizemos tudo, já fizemos tudo, já. O seu colo tem"

2) Reescreva a transcrição se passando pelo cliente, ou seja, na primeira pessoa, sendo objetivo e removendo ruídos (marcas de oralidade, texto prolixo...), mantendo a essência da dúvida do cliente para que possa ser respondida por um atendente. Isto é, resuma a dúvida em uma pergunta que possa ser respondida por um agente.

Exemplo original: "Isso mesmo, o valor de R\$ 59,90, o plano que eu fiz foi por os seis primeiros meses. Eu paguei somente uma fatura de R\$ 59,90. 6. 6 primeiros. Já fazem duas semanas que eu estou esperando e a minha fatura venceu. O meu sinal vai cortar e eles não ajustaram minha fatura para poder pagar"

Exemplo reescrito: "Combinei com a empresa de telefonia que deveria pagar R\$ 59,90 nos primeiros 6 meses, no entanto a fatura está vindo com valor fora do combinado em contrato. O que fazer?"

3) Retorne APENAS a demanda reescrita, nada além disso.

Prompt novo:

Assuma o papel de um cliente ligando a um call center para tirar dúvidas sobre algum serviço contratado que apresenta problemas.

Você receberá a transcrição do que um cliente falou durante seu atendimento (por ser automática, podem haver erros de transcrição! Corrija-os quando puder!)

Seu objetivo é reescrever a demanda do cliente, isto é, entender qual foi a motivação do cliente para ter feito a ligação e reescrevê-la de forma objetiva na primeira pessoa, como se você fosse o cliente dizendo o que deseja.

Siga os passos abaixo e verifique os exemplos para realizar isso:

1) Reescreva a transcrição se passando pelo cliente, ou seja, na primeira pessoa, sendo objetivo e removendo ruídos (marcas de oralidade, texto prolixo...), mantendo a essência da dúvida do cliente para que possa ser respondida por um atendente. Isto é, resuma a dúvida em uma pergunta que possa ser respondida por um agente.

Exemplo original: "Isso mesmo, o valor de R\$ 59,90, o plano que eu fiz foi por os seis primeiros meses. Eu paguei somente uma fatura de R\$ 59,90. 6. 6 primeiros. Já fazem duas semanas que eu estou esperando e a minha fatura venceu. O meu sinal vai cortar e eles não ajustaram minha fatura para poder pagar"

Exemplo reescrito: "Combinei com a empresa de telefonia que deveria pagar R\$ 59,90 nos primeiros 6 meses, no entanto a fatura está vindo com valor fora do combinado em contrato. O que fazer?"

2) Retorne APENAS a demanda reescrita, nada além disso.

Referências:

[1] **OPENAI. Introducing ChatGPT.** Disponível em: <<https://openai.com/blog/chatgpt>>. Acesso em: 21 dez. 2023.

[2] LEWIS, P. et al. **Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks.** Disponível em: <<https://arxiv.org/abs/2005.11401>>. Acesso em: 20 dez. 2023.

[3] OpenAI API. **Embeddings.** Disponível em: <<https://platform.openai.com/docs/guides/embeddings/what-are-embeddings>>. Acesso em: 21 dez. 2023.

[4] JASON ZHANSHUN WEI et al. **Finetuned Language Models Are Zero-Shot Learners.** Disponível em: <<https://arxiv.org/abs/2109.01652>>. 3 set. 2021.

APÊNDICE 6

Termo de Aceite de Entrega - Semana 10

Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

Data da Reunião (“gate”) de aprovação: 11 de jan. de 2024

Participantes da Entrega [matriculados em Residência em IA]:

Alex Echeverria

Entrega: [descrever a ENTREGA: requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

Recapitulando:

- Este trabalho propõe a elaboração de um pipeline que consome gravações de áudio de ligações em um ambiente de call center de modo a gerar datasets prontos para Instruct Fine Tuning de LLM's, automaticamente.
- Na última entrega, foi feita a detecção de demandas dos clientes a partir de suas falas transcritas. Isso resultou em demandas reescritas com naturalidade e objetivas, como se fosse a própria pessoa falando qual sua dúvida.
- Também idealizou-se como seriam armazenadas as demandas dos clientes e candidatos a respostas (falas transcritas dos atendentes).

O que foi planejado para essa semana:

- **Implementar o formatador de instâncias para Instruct Fine Tuning.**
- **Iniciar a implementação do treino de LLM's a partir dos datasets gerados.**
- **Levantar os resultados obtidos e melhorias.**

O que foi desenvolvido:

- Os avanços estão descritos, detalhadamente, no seguinte documento:
[DocumentaçãoSemana8-AutoInstruct4Call](#)
- **Implementação da busca semântica:**
 - As falas dos clientes foram reescritas pelo ChatGPT para evidenciar o objetivo da ligação, resultando em embeddings numéricos que representam as demandas.
 - Estes embeddings foram gerados pelo modelo "text-ada-002" da OpenAI e armazenados no Elasticsearch.

- As respostas dos atendentes passaram pelo mesmo processo
- A busca por respostas relevantes utiliza a distância de cosseno entre os embeddings das demandas dos clientes e as respostas dos atendentes
- Recuperou-se múltiplas respostas (3) para cada demanda de cliente.

- **Refinamento dos candidatos a resposta:**
 - Utilização do ChatGPT para identificar demandas inválidas nas transcrições.
 - Exemplos de demandas inválidas incluem ruídos e perguntas sem sentido.
 - Tentativa de checagem de validade, mas decisão de remover essa função devido à complexidade e variedade de demandas em produção.
 - Candidatos a resposta com ruídos e informações indesejáveis.
 - Uso do ChatGPT para reescrever candidatos, enfatizando informações relevantes.
 - Objetivo final é construir instâncias pergunta-resposta para treinar modelos de Perguntas e Respostas (Q&A).
 - Uso do ChatGPT para sintetizar uma resposta única e completa a partir dos três candidatos, eliminando ruídos e informações desnecessárias.
 - Resultado é um dataset quase pronto para Instruct Tuning de Large Language Models (LLMs) na tarefa de Q&A.

- **Formatação da busca semântica:**
 - O artigo "Finetuned Language Models are Zero-shot Learners" destaca melhoria significativa na capacidade de abstração e retenção de informações em LLMs ao descrever tarefas de NLP usando linguagem natural.
 - Enfatiza que a descrição clara da tarefa, mesmo de forma breve, contribui para o melhor aprendizado do modelo em tarefas específicas, como Q&A (pergunta e resposta).
 - A implementação baseada no conceito mencionado envolve a criação de instruções em linguagem natural para a tarefa de Q&A.
 - Adotou-se um método simples e efetivo, conforme descrito no artigo, que consiste em incluir instruções em cada par de perguntas e respostas no dataset.
 - Criou-se 10 instruções específicas para Q&A e adição aleatória a cada par de perguntas e respostas disponíveis no conjunto de dados.

Planejamento: [descrever o que pretende fazer para realizar a próxima ENTREGA]

Observação: [caso precise fazer alguma observação, de qualquer "natureza"]

ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: [Go! ▾](#)

LUANA GUEDES BARROS MARTINS: [Go! ▾](#)

Documentação Semana 8: Implementação da busca semântica, busca por respostas às demandas e formatação das instâncias.

Implementação da busca semântica:

- Contextualização:

Na última semana, foi desenvolvido um algoritmo para extrair as demandas dos clientes a partir de toda a transcrição de sua fala. Isto é, a partir da transcrição de toda a interação do cliente com o atendente, recorta-se apenas as falas do cliente e, em seguida, são fornecidas ao ChatGPT para reescrever a fala evidenciando o objetivo da ligação, de forma natural e objetiva.

Com isso, obteve-se as falas dos clientes de forma objetiva e natural, no formato de uma pergunta, salientando a demanda deles. Isso é importante, pois em seguida obtiveram-se os *embeddings*, isto é, as representações numéricas das falas (demandas) dos clientes.

- Extração de embeddings:

Os *embeddings*, como explicado no *crash course* do Google [1], são representações numéricas (vetores) em um espaço dimensional relativamente pequeno, ou seja, vetores algébricos que visam compactar as informações de espaços dimensionais muito mais esparsos. Esse recurso é extremamente importante no campo de Processamento de Linguagem Natural pelo fato de conseguir representar palavras dinamicamente e permitir que operações matemáticas sejam realizadas com esses vetores, ou seja, tanto adição e subtração quanto medir a distância entre vetores são operações que fazem sentido com os *embeddings* de palavras ou frases.

Esses *embeddings* são gerados a partir de modelos de linguagem, ou seja, a partir dos estados internos (ativações de camadas de redes neurais) conforme pode ser observado no livro de Lane et al. [2]. Existem diversos modelos, como pode ser percebido ao longo dos capítulos do livro, desde modelos puramente estatísticos (dos capítulos da Parte 1 do material [2]), até modelos mais complexos, que incorporam mecanismos de atenção e outras soluções mais sofisticadas (como pode ser visto nos capítulos da Parte 2 dele [2]).

Dada a contextualização, para o presente trabalho **foi selecionado o uso do modelo de geração de embeddings "text-ada-002"**, da OpenAI [3, 4], dada a sua facilidade de uso (disponibilizado via API [4]) e robustez apresentada em seu trabalho [2], dado que esse modelo é descendente do GPT-3 [5], modelo de linguagem que foi um marco na história de NLP devido ao seu tamanho, complexidade e capacidades de entender a linguagem humana.

Portanto, **para cada demanda reescrita, fora extraído seu vetor de embeddings** que representa esse texto (demanda), o qual é composto por 1536 *floating point* números. Em seguida, esses vetores foram armazenados em um banco de dados Elasticsearch, conforme explicado adiante.

- **Para as respostas dos atendentes:**

Também fora executado o mesmo procedimento de extração de embeddings para as falas dos atendentes, ou seja, para cada ligação disponível, pegou-se as transcrições dos atendentes, realizou-se o pré-processamento do texto e, utilizando o modelo "text-ada-002", extraíram-se os embeddings e armazenou-se no servidor Elasticsearch.

- **Do servidor Elasticsearch:**

O serviço do Elasticsearch [6] funciona como um banco de dados e poderoso indexador de documentos. Suas capacidades de escalabilidade e robustez de busca o fazem ser uma ferramenta interessante para uso em produção. No entanto, para o presente trabalho, apenas é necessário um banco de dados que tenha como realizar buscas a partir dos vetores armazenados como metadados dos textos.

Os conteúdos armazenados no servidor Elasticsearch são: o *segmento de texto* (tanto as falas do cliente quanto do atendente), a *persona* (isto é, se é atendente ou cliente), o *arquivo* (o nome do arquivo referente à ligação) e o *embedding* (vetor de valores floats).

Essa busca se dá por meio da distância de cosseno, ou seja, a partir de um vetor query (embedding da demanda de um cliente), busca-se os vetores mais semelhantes a essa query (no caso, são os embeddings das falas dos atendentes).

Portanto, recupera-se um número arbitrário de respostas de atendentes para cada demanda de cliente disponível no banco de dados. O número escolhido de respostas para cada demanda é de 3, apenas por velocidade em etapas de processamento posteriores.

Vale ressaltar que, dada a demanda do cliente presente em uma ligação, seu embedding é comparado com todas as respostas fornecidas pelos atendentes em todas as outras ligações existentes na base de dados. Isso permite, pois, que as respostas à demanda do usuário não se limite a apenas a sua ligação, mas que seja possível identificar informações pertinentes à demanda em todas as outras respostas dadas por diferentes atendentes, em diferentes momentos.

Exemplo da busca semântica:

- **Demanda reescrita do cliente:** "Estou ligando para obter informações sobre a falta de internet e comunicação para que possam ressarcir minha linha. Já fazem duas semanas que estou sem internet. O que devo fazer?"
- **Resposta de atendente com maior nível de similaridade:** "do serviço. E-mail para contato. Um momento. Obrigada por aguardar. Me informe o número para contato

com o DDD? Sabe da localidade? Qual seria o seu reclamado da operadora vivo? Imóvel, fixo, TV por assinatura ou internet? Banda larga? Banda larga. Pode informar pessoal a objetiva ou a descrição do problema? E aí E aí Com a cidade e o estado onde ocorre o problema? Obrigada por aguardar. O senhor reclama que há dias houve um ciclone e ficou sem internet. Entrou em contato com a operadora, foi informado que seria uma massiva, porém o problema não é solucionado. Como providência, solicito a resolução do problema com urgência. Posso finalizar? Após a finalização, não tem como incluir novas informações. Posso finalizar? O atendimento foi finalizado ao vivo. Ela terá um prazo de dias para responder a sua solicitação e contactá-lo. Caso julgue a resposta inadequada, a senhora terá um prazo de dias para reabri-la. Seu protocolo é o ---. São dez dias corridos E caso o problema não seja solucionado A senhora retorna Muito obrigada A NOME agradece Você vai guardar de linha Até um ótimo dia"

Refinamento dos candidatos a resposta:

- **Contextualização:**

Como pode ser visto no exemplo de busca semântica, os candidatos a resposta da demanda (isto é, as falas de atendente mais semelhantes à demanda) se apresentam cheios de ruídos e informações indesejáveis.

Como o objetivo deste trabalho é a geração de datasets para o treinamento de grandes modelos de linguagem, é necessário que uma adequação dos candidatos seja feita, de tal forma que seja salientado apenas a solução para o problema, a resposta da dúvida ou o procedimento realizado para o cliente.

Desse modo, utiliza-se novamente o ChatGPT para que os candidatos a resposta sejam reescritas de tal modo que seja destacado suas informações úteis.

- **Identificação de demandas inválidas:**

No entanto, antes de realizar essa reescrita, usou-se o ChatGPT para tentar identificar demandas que fossem inválidas, isto é, que a transcrição fosse somente ruído (não contivesse uma demanda verdadeira).

Abaixo, estão dois exemplos de demanda inválida:

Demanda 1: 'Eu gostaria de saber como posso fazer um projeto com a oficina. Meu esposo não sabe o que precisa fazer. O que devo fazer nesse caso?'

Demanda 2: 'BIG ERROR'

Como pode ser visto nos exemplos acima, não há uma demanda genuína para a operação de suporte a serviços de telecomunicações e, assim como esse exemplo, podem haver diversos tipos de ruídos causados tanto por erros de transcrição, erros na ligação, quanto o cliente realizar perguntas sem sentido ao contexto.

Foi fornecido ao ChatGPT o prompt 1 (veja Apêndice A), em que solicita a checagem se a demanda é válida ou inválida, para que fosse detectado quais demandas são inválidas em uma população de 3120 demandas reescritas de clientes. Dessas, apenas 2 foram detectadas como inválidas, 3118 detectadas como válidas. No entanto, após uma inspeção visual no conjunto de dados, foram identificados mais de 2 elementos que se enquadram como demanda inválida.

Não despendeu-se tempo na tentativa de melhorar esses resultados, preferiu-se remover essa função de checagem da demanda pois podem existir, em produção, demandas nada a ver e que precisam que o modelo de linguagem lide com elas adequadamente. Portanto, deixou-se as demandas inválidas e suas possíveis respostas.

Parte do pipeline de treino é investigar como a presença desses ruídos afetarão o desempenho dos modelos.

- **Refinamento dos candidatos a resposta das demandas:**

Para cada demanda, foram selecionadas as 3 falas de atendentes mais semelhantes semanticamente de toda a base de dados. Contudo, deve-se realizar um refinamento para remover ruídos e salientar a informação principal que o atendente quis passar em seu atendimento, conforme já explicado anteriormente.

Com isso, fez-se necessário a criação do prompt 2 (veja Apêndice A), em que solicita ao GPT reescrever cada candidato a resposta de forma clara, objetiva e salientar as informações que podem ser úteis para responder à demanda (também passada no prompt), descartando informações irrelevantes.

Logo, chegou-se ao seguinte formato disponível no banco de dados: Demanda reescrita do cliente + 3 candidatos a resposta com ênfase nas informações.

- **Combinação dos candidatos a resposta refinados:**

O objetivo final do presente trabalho é a construção de instâncias do tipo pergunta e resposta para que seja treinado um LLM que realizará a tarefa de Q&A. Para isso, é necessário que os 3 candidatos a resposta sejam transformados em apenas uma resposta, o mais completa possível.

Para tal, é fundamental o uso do ChatGPT, mais uma vez, pelas suas capacidades de entendimento e abstração da linguagem. Por tal motivo, é solicitado no prompt 3 (veja Apêndice A) que leia os três candidatos a resposta e sintetize-os em apenas uma resposta, clara e concisa, que contenha todas as informações disponíveis nos três candidatos e que remova os ruídos e informações desnecessárias que possam estar presentes.

Passou-se esse prompt por todas as instâncias presentes na base de dados e chega-se próximo ao dataset pronto para o Instruct Tuning de LLMs.

Exemplos de instâncias após a combinação:

- **Exemplo 1:**

Demanda do cliente:

"Boa noite, estou tendo problemas com a Vivo. Já fiz várias tentativas de resolver, mas não obtive sucesso. Preciso cancelar o plano do meu marido que faleceu e encerrar a conta, mas a Vivo não está cancelando o plano e continua fazendo débitos automáticos na conta dele. Além disso, eles não estão me reembolsando o valor pago indevidamente. O que devo fazer para resolver essa situação?"

Resposta combinada:

"Para resolver a situação com a Vivo, é necessário entrar em contato com a operadora e fornecer o nome completo do titular do serviço, o número de telefone com DDD e o e-mail para contato. Informe também o protocolo da reclamação aberta contra a Vivo, a data em que foi aberta e a data em que a Vivo entrou em contato para informar o valor da cobrança contestada. A cobrança está correta, pois foi tarifada de acordo com o período de utilização do plano antigo e do novo plano controle. A Vivo ofereceu um desconto como cortesia e a fatura ficou no valor de R\$ [valor]. A operadora enviou o código de barras. Se precisar de mais informações, aguarde para avaliar o atendimento."

- **Exemplo 2:**

Demanda do cliente:

"Boa tarde, meu nome é NOME e sou síndico do Condomínio das Pérolas Bloco A. Já fiz uma reclamação anteriormente e abri um novo protocolo na Anatel. Marcaram um prazo para resolver o problema, mas ninguém apareceu. O que devo fazer agora?"

Resposta combinada:

"Lamento pelo problema que está enfrentando, Sr. NOME. Para que eu possa ajudar, preciso que me informe qual é o serviço reclamado: fixo, banda larga, pós-pago ou pré-pago. Além disso, você poderia me fornecer o número do protocolo feito recentemente na ouvidoria da operadora? Assim, poderei verificar a descrição do problema e tomar as medidas necessárias. O prazo para resposta da operadora é de até X dias. Caso a resposta não seja satisfatória, você terá X dias para reabrir o chamado. Por favor, informe também a cidade onde ocorre o problema. Vou solicitar à operadora que verifique e restabeleça urgentemente a sua linha, além de conceder um desconto proporcional na fatura pelo período em que o serviço não foi utilizado. O número do seu protocolo será encaminhado à operadora, que terá um prazo de até X dias para responder. Caso julgue a resposta insatisfatória, você poderá contestar conosco em até X dias. Agradeço pela sua paciência e compreensão. Boa tarde."

Formatação da busca semântica:

- Contextualização:

Conforme é explicado em *Finetuned Language Models are Zero-shot Learners* [6], é observado uma melhora significativa na capacidade de abstração e retenção de informações de LLM's quando se descreve uma tarefa de NLP usando linguagem natural. Isto é, uma tarefa de Q&A (pergunta e resposta) será melhor aprendida se descrever, mesmo que de forma breve, que o modelo deve responder à pergunta passada.

Esse simples mas efetivo ato é, também, simples de ser implementado. Como ressalta o artigo, basta escrever uma quantidade arbitrária de instruções em linguagem natural para a tarefa a ser feita (no presente trabalho é Q&A) e, para cada par de perguntas e respostas presente no dataset, adicionar uma instrução aleatoriamente escolhida na população de instruções.

Portanto, criou-se 10 instruções que descrevem a tarefa de Q&A (Apêndice B) e que foram adicionadas a cada par de perguntas e respostas disponíveis.

Exemplos de instâncias do dataset final:

1) "Utilizando seus conhecimentos sobre call center, responda à pergunta a seguir:

Q: Por que minha solicitação de envio da fatura detalhada não foi atendida e o número de protocolo está inválido? O que posso fazer para resolver essa situação?

A: Desculpe pelo inconveniente com a solicitação de envio da fatura detalhada e o número de protocolo inválido. Para resolver essa situação, você pode atualizar o número de contato imediatamente. Informe qual é o problema específico com a fatura pendente de fevereiro e abril. Se desejar trocar o número telefônico devido a ameaças recebidas, informe a cidade e o estado em que ocorre o problema. Após a finalização da solicitação, você receberá um aviso dentro de X dias corridos. Caso não esteja satisfeito com a resposta, você terá X dias corridos para reabrir a solicitação. Obrigado por sua compreensão e participação em nossa pesquisa de satisfação. A Naceu agradece sua atenção."

2) "Assuma o papel de um atendente de call center e responda à demanda de um cliente:

Q: Estou com um problema na minha conta da Vivo. Recebi uma fatura de um número de telefone que não é meu, de um DDD de outro estado. Já entrei em contato com a ouvidoria, mas eles só cancelaram o número utilizado, não a conta. Como faço para cancelar essa cobrança indevida?

A: Para resolver o problema com a cobrança indevida na conta da Vivo, é necessário entrar em contato com a ouvidoria da operadora. Informe o número do protocolo da reclamação

anterior e explique que a ouvidoria cancelou apenas o número utilizado, mas não a conta. Além disso, mencione que você está sem o serviço de telefone fixo há dias e que as reclamações anteriores não foram resolvidas. A operadora terá um prazo para responder à sua solicitação e, caso a resposta seja inadequada, você poderá reabrir a reclamação. Anote o número do protocolo para futuras referências."

Apêndice A - Prompts:

Prompt 1:

Você receberá uma fala de um cliente de call center relatando sua demanda, marcado por <dem> e </dem> . Seu objetivo é verificar se de fato existe uma demanda ou é apenas ruído, como uma fala nada a ver, e retornar um valor binário (0 ou 1).

O call center é de uma empresa de telecomunicações e o cliente geralmente liga para reclamar ou tirar dúvidas sobre esse tema.

Caso a demanda seja apenas um ruído (tal como apenas "oi", "Legendas por álamo", "bom dia"), retorne apenas o valor 0.

Se a demanda for válida, retorne apenas o valor 1.

Prompt 2:

Você receberá duas frases: uma demanda de um cliente de call center e uma resposta de um atendente, marcados por <dem> e <res> respectivamente.

Sua tarefa consiste em buscar informações úteis na resposta e reescrever a resposta do atendente de forma clara e objetiva, de modo a dar ênfase à informação útil e descartar o que não for útil.

Observação: como a resposta foi transcrita por um algoritmo, pode conter erros de digitação. Então tente corrigir esses erros e relevar alguns problemas que não afetem a compreensão da resposta.

Te darei \$100 de gorjeta se fizer bem feito, ou seja, reescrever de forma clara, direta e objetiva a resposta do atendente!!

Prompt 3:

Você receberá 3 respostas feitas para uma mesma pergunta de um cliente, demarcadas por um número de 1 a 3. Tudo que você deve fazer é escrever uma única resposta que englobe as informações de todas as 3 outras respostas, de forma objetiva.

Ou seja, entenda o que cada uma das 3 respostas base possuem e escreva uma única resposta que seja coerente.

Pode ser que algumas respostas sejam apenas ruídos, ou seja, não tenham informações relevantes.

Portanto, você deve ser capaz de identificar quais respostas são relevantes e quais não são para construir a resposta final, de maneira concisa e objetiva, sem perder informações.

Evite e remova repetições da fala do cliente, ou seja, ficar repetindo o problema que a pessoa relatou.

Retorne apenas a resposta final e nada além disso.

Apêndice B - Instruções:

- 1) Forneça informações essenciais ao responder à seguinte pergunta:
- 2) Elabore uma resposta embasada nos termos-chave mencionados pelo cliente:
- 3) Apresente uma resposta objetiva com base na seguinte dúvida:
- 4) Responda à pergunta levando em conta variações de terminologia dos clientes:
- 5) Responda de forma objetiva e direta a pergunta de um cliente de call center:
- 6) Levando em consideração o contexto de call center e telecomunicações, responda ao questionamento:
- 7) Leia atentamente à pergunta de um cliente e a responda em seguida:
- 8) Utilizando seus conhecimentos sobre call center, responda à pergunta a seguir:
- 9) Assuma o papel de um atendente de call center e responda à demanda de um cliente:
- 10) Você é um atendente de call center e um cliente te fez uma pergunta. Responda-a:

[1] GOOGLE DEVELOPERS. **Embeddings | Machine Learning**. Disponível em: <<https://developers.google.com/machine-learning/crash-course/embeddings/video-lecture?hl=en>>. Acesso em: 25 jan. 2024.

[2] LANE, H.; HOWARD, C.; HANNES MAX HAPKE. **Natural language processing in action : understanding, analyzing, and generating text with Python**. [s.l.] Usa] Manning Publications, 2019.

[3] OPENAI. **Introducing text and code embeddings**. Disponível em: <<https://openai.com/blog/introducing-text-and-code-embeddings>>.

[4] OPENAI. **Embeddings**. Disponível em: <<https://platform.openai.com/docs/guides/embeddings/use-cases>>. Acesso em: 24 jan. 2024.

[5] BROWN, T. B. et al. **Language Models are Few-Shot Learners**. Disponível em: <<https://arxiv.org/abs/2005.14165>>, 28 maio 2020.

[6] JASON ZHANSHUN WEI et al. **Finetuned Language Models Are Zero-Shot Learners**. Disponível em: <<https://arxiv.org/abs/2109.01652>>. 3 set. 2021.

APÊNDICE 7

Termo de Aceite de Entrega - Semana 11

Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

Data da Reunião (“gate”) de aprovação: 17 de jan. de 2024

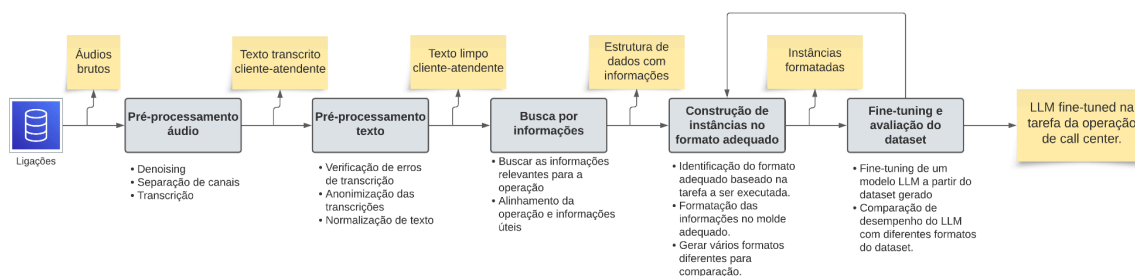
Participantes da Entrega [matriculados em Residência em IA]:

Alex Echeverria

Entrega: [descrever a ENTREGA: requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

Recapitulando:

- Este trabalho propõe a elaboração de um pipeline que consome gravações de áudio de ligações em um ambiente de call center de modo a gerar datasets prontos para Instruct Fine Tuning de LLM's, automaticamente.
- Na última entrega (Gate 10), finalizou-se todo o processo de geração de um dataset pronto para Instruct Fine Tuning. Ou seja, todas as etapas planejadas conforme o *pipeline* foram feitas e entregues, como pode ser verificado abaixo:



O que foi planejado para essa semana:

- **Implementar o treino de LLM's a partir do dataset gerado.**
- **Levantar os resultados obtidos.**
- **Organizar os códigos desenvolvidos em uma versão pública, acessível pelo GitHub.**







O que foi desenvolvido:

- LLM Instruct Tuning:
 - Utilizou-se um dataset gerado sobre ligações relacionadas com o setor de telecomunicações, o qual consiste em 3117 pares de demandas (do cliente) e respostas (do atendente).

- Escolheu-se o modelo Llama2 [1] (Meta), arbitrariamente definido por sua comodidade de já disponibilizar uma versão finetuned na tarefa de *chat* (semelhante a Q&A) e por seu desempenho interessante (como ressaltado no artigo [1]), com 7 bilhões de parâmetros.
 - O treinamento foi feito utilizando a plataforma Lamini [2], a qual tem a interessante capacidade de abstrair o treino de LLM's para apenas 3 linhas de código. O treino é executado em nuvem, auto gerenciado pela própria plataforma.
 - **Como resultado**, obteve-se um modelo Llama 2 7B *finetuned* em exemplos práticos de ligações de um call center de telecomunicações, o qual pode ser usado por clientes para tirar dúvidas. Exemplo interativo do modelo encontra-se no link:
https://app.lamini.ai/share?model_id=2a2dd63966b5f87e977a6db8c81027b1bb2d9971f3be41c1edf66cea911bac87
 - A ferramenta Lamini é um diferencial pela sua capacidade de abstrair detalhes do treinamento. No entanto, isso tem o custo de perder o controle e ajuste fino do modelo, o qual um especialista em treino de LLM's possui. Como o escopo do presente trabalho é a elaboração de datasets, absteve-se o fornecimento de detalhes do treino.
- GitHub:
- Os códigos referentes ao projeto podem ser encontrados no seguinte repositório do GitHub: [aveusalex/AutoInstruct4LLMs: Public \(github.com\)](https://github.com/aveusalex/AutoInstruct4LLMs)
 - Nele, encontram-se todos os códigos referentes ao projeto, bem como instruções de como executar.

Resultados:

- Interagindo com o modelo, pode-se enxergar alguns resultados:

- 1)  Minha fatura de internet está atrasada há 3 meses. Não consigo pagar o boleto e eles me cobram que eu tenho que pagar. que fazer?
 Para resolver o problema com a fatura de internet que está atrasada há 3 meses, sugiro que você entre em contato com a operadora e explique a situação. Eles podem fornecer informações sobre a possibilidade de abater o valor da fatura em atraso ou prorrogar o pagamento. Caso não concorde com a resposta da operadora, você pode abrir uma reclamação e mencionar que não consegue pagar o boleto. A operadora terá um prazo para responder e entrar em contato com você. Se não receber uma resposta adequada, você pode reabrir a reclamação.
- 2)  A operadora Claro fica me mandando mensagens, perturbando minha paciência. Já pedi para não enviarem mensagens mas continuam enviando. O que devo fazer?
 Para resolver essa situação, sugiro que você entre em contato com a ouvidoria da Claro e registre uma reclamação. Caso o prazo fornecido para a solução do seu problema tenha expirado e você ainda não tenha obtido resposta, você pode reclamar neste canal, informando o número do protocolo da sua reclamação na ouvidoria. A operadora tem até 10 dias corridos para responder à sua solicitação. Se você tiver mais alguma dúvida, estou à disposição. Obrigado.
- 3)  Comprei um drone pela internet e não sei o que fazer para homologar ele.
 Já pedi para eles mandarem um protocolo, mas eles pediram para eu ter um drone, o que não consigo fazer. Além disso, gostaria de cancelar o plano com a Sky, mas sei que há uma multa. O problema é apenas com o controle, não com o drone em si. Obrigado. [/INST] Para resolver o problema com o controle do seu drone, já foi feito o pedido de envio de um protocolo. Agora, em relação ao cancelamento do plano com a Sky, é importante saber que haverá uma multa. Para proceder com o cancelamento, você precisa entrar em contato com a prestadora e fazer o pedido. Se tiver mais alguma dúvida, estou à disposição. Obrigado.

- Pode-se perceber que o modelo aprendeu o contexto de call center e consegue abstrair a dúvida do cliente, respondendo, mesmo que sem detalhes (devido ao dataset gerado não conter especificidades), ao problema (sem fugir do contexto), conforme é evidenciado pelos exemplos 1 e 2.
- No entanto, há melhorias que devem ser feitas tanto no treino do modelo, quanto na qualidade do dataset gerado, pois há situações em que o modelo pode gerar resultados confusos, conforme é evidenciado pelo exemplo 3.

Referências:

[1] TOUVRON, H. et al. Llama 2: Open Foundation and Fine-Tuned Chat Models. [s.l: s.n.]. Disponível em: <<https://arxiv.org/pdf/2307.09288.pdf>>. Acesso em: 17 jan. 2024.

[2] Lamini - LLM Platform - Connect your Enterprise Data Warehouse. Disponível em: <<https://www.lamini.ai/>>. Acesso em: 17 jan. 2024.

Planejamento: [descrever o que pretende fazer para realizar a próxima ENTREGA]

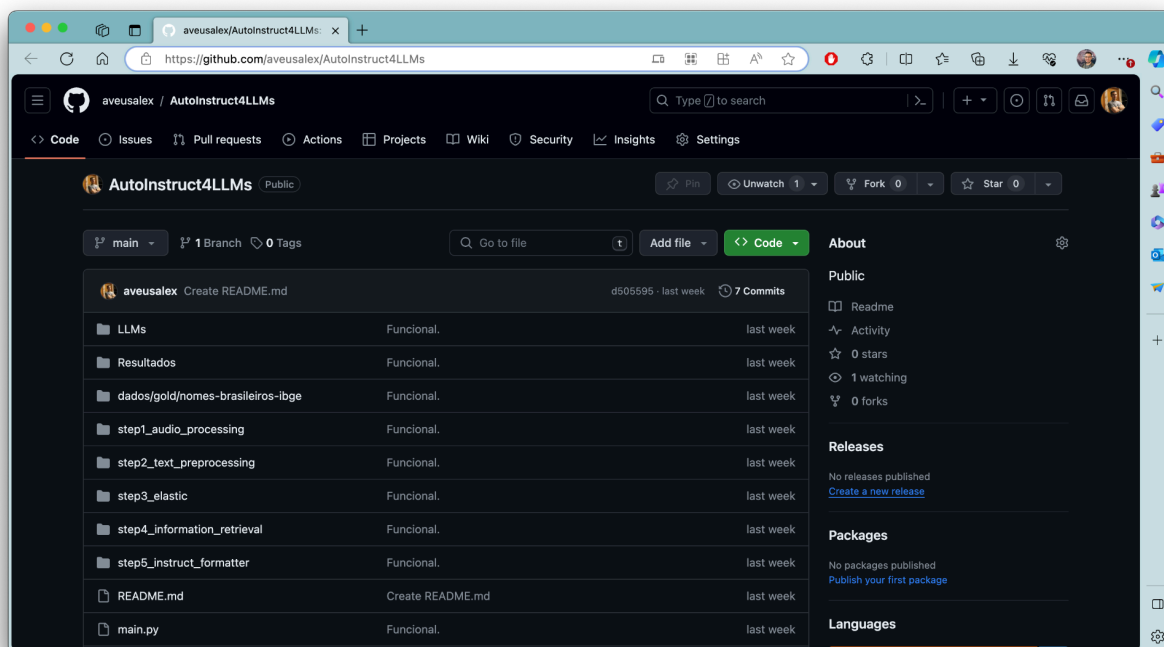
Observação: [caso precise fazer alguma observação, de qualquer “natureza”]

ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: Go! ▾

LUANA GUEDES BARROS MARTINS: Go! ▾

Repositório público no GitHub



Fonte: <https://github.com/aveusalex/AutoInstruct4LLMs>

APÊNDICE 8

Códigos

LLMs/ChatGPT.py

```
import os
import openai
import logging
import datetime
from time import sleep

cwd = os.getcwd().split("AutoInstruct4LLMs")[0] +
"AutoInstruct4LLMs/"

class SimpleCompletion:
    def __init__(self, api_key_path: str, logs_path: str =
f"{cwd}/logs/gpt"):
        self.openai_client =
openai.OpenAI(api_key=open(api_key_path, "r").read())
        self.tokens_in = 0
        self.tokens_out = 0
        self.messages = []

        # criando a pasta de logs
        if not os.path.exists(logs_path):
            os.makedirs(logs_path)

        # criando o logger
        self.logger = logging.getLogger(__name__)
        self.logger.setLevel(logging.INFO)
```

```

        formatter =
logging.Formatter('%(asctime)s:%(levelname)s:%(message)s')
        datet =
datetime.datetime.now().strftime("%Y-%m-%d_%H-%M-%S")
        file_handler =
logging.FileHandler(f"{logs_path}/GPT_{datet}.log",
encoding='utf-8')
        file_handler.setFormatter(formatter)
        self.logger.addHandler(file_handler)

    def set_context(self, context: str) -> None:
        """
        Função para definir o contexto da conversa.
        :param context: contexto da conversa str ou path_audio
apontando para um arquivo de texto
        :return: None
        """
        # verificando se o contexto aponta para um arquivo de
texto
        if os.path.exists(context):
            with open(context, "r") as f:
                context = f.read()

            self.messages.append({'role': 'system', 'content':
context})

    def reset_messages(self, maintain_context=True):
        if maintain_context:
            self.messages = [self.messages[0]]
        else:
            self.messages = []
```

```
def get_completion(self, text: str = None,
model="gpt-3.5-turbo", temperature=0, system=None) -> tuple:
    """
    Função para obter a resposta do modelo de linguagem.
    :param text: o texto que será enviado para o modelo de
    linguagem (opcional)
    :param model: o modelo de linguagem que será utilizado
    (4k prompt padrão)
    :param temperature: nível de aleatoriedade da resposta do
    modelo de linguagem
    :param system: não mexer. Parâmetro interno para trocar o
    modelo caso o modelo atual esteja com problemas
    :return: tuple(str, openai.Response)
    """
    if text is not None:
        self.messages.append({'role': 'user', 'content':
text})

    try:
        response =
self.openai_client.chat.completions.create(
            model=model,
            messages=self.messages,
            temperature=temperature, # this is the degree
of randomness of the model's output
        )
    except openai.BadRequestError as e:
        if system is None:
            self.logger.info(f"Erro ao enviar a mensagem.
Trocando de modelo. Error: {e}")
```

```
        # trocando o modelo por um maior
        return self.get_completion(text,
model="gpt-3.5-turbo-1106", temperature=temperature, system=1)
    else:
        self.logger.info(f"Mensagem extremamente grande.
Error: {e}")
        return "BIG ERROR", None
    except openai.RateLimitError as e:
        self.logger.info(f"Erro ao enviar a mensagem.
Esperando 45 segundos. Error: {e}")
        sleep(45)
        return self.get_completion(text, model=model,
temperature=temperature)

    modelo = response.model
    idd = response.id
    texto_resp = response.choices[0].message.content
    tokens_in = response.usage.prompt_tokens
    tokens_out = response.usage.completion_tokens
    self.logger.info(f"Modelo: {modelo} | ID: {idd} | Texto:
{texto_resp} | Tokens_in: {tokens_in} | Tokens_out: {tokens_out}")
    return response.choices[0].message.content, response

class Chat(SimpleCompletion):
    def __init__(self, api_key_path: str):
        super().__init__(api_key_path)
        self.messages = []

    def add_context(self, context: str):
        # inserindo o contexto no começo da lista de mensagens
```

```
context = {'role': 'system', 'content': context}
self.messages = [context] + self.messages

def add_message(self, role: str, prompt: str):
    self.messages.append({'role': role, 'content': prompt})
```

LLMs/embedding_calculator.py

```
import openai
import logging
import os
import datetime

cwd = os.getcwd().split("AutoInstruct4LLMs")[0] +
"AutoInstruct4LLMs/"

class Ada002:
    def __init__(self, api_key_path: str, logs_path: str =
f"{cwd}/logs/embedding"):
        self.openai_client =
openai.OpenAI(api_key=open(api_key_path, "r").read())
        self.tokens_in = 0

        # verificando se a pasta de logs existe
        if not os.path.exists(logs_path):
            os.makedirs(logs_path)

        # criando o logger
        self.logger = logging.getLogger(__name__)
        self.logger.setLevel(logging.INFO)

        formatter =
logging.Formatter('%(asctime)s:%(levelname)s:%(message)s')
        datet =
datetime.datetime.now().strftime("%Y-%m-%d_%H-%M-%S")
```

```
file_handler = logging.FileHandler(f"{logs_path}/embedding_{date}.log", encoding='utf-8')
file_handler.setFormatter(formatter)
self.logger.addHandler(file_handler)

def get_embedding(self, text: str) -> tuple:
    """
    Função para obter o embedding de um texto.
    Retorna o embedding e o número de tokens usados até o momento.
    :param text: texto que será enviado para o modelo
    :return: tuple(list, int)
    """
    response = self.openai_client.embeddings.create(
        model="text-embedding-ada-002",
        input=text,
        encoding_format="float"
    )

    response_emb = response.data[0].embedding
    response_usage = response.usage.total_tokens
    preco = response_usage * 0.0001/1000
    # registrando o log
    self.logger.info(f" Tokens usados: {response_usage} - Preço: U${preco:.3f} - Texto: {text}")
    self.tokens_in += response_usage

    return response_emb, self.tokens_in
```

```
if __name__ == '__main__':  
    obj = Ada002(api_key_path="../key.txt")  
    print(len(obj.get_embedding("Hello, my name is Alex.")[0]))  
        print(len(obj.get_embedding("Hello, my name is  
Mariana.")[0]))
```

step1_audio_processing/URA/features_extractor.py

```
import concurrent.futures
import os
import librosa
import numpy as np
import time
import pandas as pd
# desativando warnings
import warnings

warnings.filterwarnings("ignore")

MAX_THREADS = 4

class FeaturesClassicas:
    def __init__(self, TIME_SEGMENT: int = 5, path_audios="",
path_save_features: str = "./features", audios_list: list = None):
        """
        Essa classe é responsável por extrair as features
        clássicas de áudio.
        :param TIME_SEGMENT: tamanho do segmento de áudio em
segundos
        :param path_save_features: caminho para salvar as
features
        """
        assert TIME_SEGMENT > 0, "O tamanho do segmento deve ser
maior que 0"
        assert path_save_features != "", "O caminho para salvar
as features não pode ser vazio"
```

```
        assert path_audios != "", "O caminho para os áudios não  
pode ser vazio"  
  
        self.TIME_SEGMENT = TIME_SEGMENT  
        self.path_save = path_save_features  
        if not os.path.exists(path_save_features):  
            os.makedirs(path_save_features)  
        self.path_audios = path_audios  
        self.path_save = path_save_features  
        self.features = []  
  
        # definindo as features de antemão, mas pode ser  
alterado em tempo de execução  
        self.define_features()  
        self.audios_list = audios_list  
  
    def run(self, audio_name):  
        #print(f"Extraindo features do áudio {audio_name}")  
        wave, sr = self.load_file(audio_name)  
        segmentos = self.separa_em_segmentos(wave, sr)  
        features = self.extrai_features(segmentos, sr)  
        df = self.convert_features_to_df(features)  
        self.save_features_to_csv(audio_name, df)  
  
    def load_file(self, audio) -> (np.ndarray, int):  
        """  
        Carrega os arquivos de áudio.  
        Os áudios são armazenados no dicionário self.audios  
        :param audio: nome do áudio  
        :return: None  
        """  
  
        # percorrendo cada áudio e carregando
```

```
        y, sr = librosa.load(os.path.join(self.path_audios,
audio), sr=None, mono=False)

        wave = y[1, :] # carregando somente do ATENDENTE
        return wave, sr

def separa_em_segmentos(self, wave, sr) -> np.ndarray:
    """
    Separa os áudios em segmentos de tamanho TIME_SEGMENT.
    Os áudios são armazenados no dicionário self.audios
    :param wave: áudio
    :param sr: taxa de amostragem
    :return: None
    """

        segmentacao = librosa.util.frame(wave,
frame_length=self.TIME_SEGMENT * sr,
hop_length=self.TIME_SEGMENT * sr)
        return segmentacao

def define_features(self, features: list = None) -> None:
    """
    Define as features que serão extraídas
    :return: None
    """

    if not features:
        self.features = ["segmento",
                        "mean_centroid",
                        "sd_centroid",
                        "median_centroid",
                        "max_centroid",
                        "min_centroid",
```

```
        "mean_sfm",
        "median_sfm",
        "sd_sfm",
        "mean_rolloff_099",
        "median_rolloff_099",
        "sd_rolloff_099",
        "min_rolloff_099",
        "max_rolloff_099",
        "mean_rolloff_001",
        "median_rolloff_001",
        "sd_rolloff_001",
        "min_rolloff_001",
        "max_rolloff_001",]
        # "mean_f0",
        # "median_f0",
        # "sd_f0",
        # "min_f0",
        # "max_f0"]

    else:
        self.features = features

    def extrai_features(self, segmentos: np.ndarray, sr: int) ->
list:
        """
        Extrai as features de cada segmento de áudio usando
        threads.

        :param segmentos: segmentos de áudio
        :param sr: taxa de amostragem
        :return: lista com as features
        """
```

```
        data_parallel = [] # lista para armazenar os dados de
forma paralela (objeto mutável)
        start = time.time()
        # percorrendo cada segmento de áudio
        for idx_ in range(segmentos.shape[1]):
            segmento = segmentos[:, idx_]
            try:
                extrair_features_audio(segmento, sr,
data_parallel, idx_)
            except Exception as e:
                pass

        #print(f"Demorou {time.time() - start:.2f} segundos para
extrair features")
        # armazenando os dados das features no dicionario
        return data_parallel

def convert_features_to_df(self, features) -> pd.DataFrame:
    """
    Converte os dados das features para um dataframe
    :param features: lista com as features
    :return: pd.DataFrame com as features
    """
    # transformando em dataframe
    df_ = pd.DataFrame(features, columns=self.features)
    # ordenando pela coluna segmento
    df_ = df_.sort_values(by=["segmento"])
    # dropando a coluna segmento
    df_ = df_.drop(columns=["segmento"])
    return df_
```

```
def save_features_to_csv(self, audio_name, features, TSV:
bool = False) -> None:
    """
    Salva as features em um arquivo CSV ou TSV
    :return: None
    """
    if TSV:
        sep = "\t"
        final = ".tsv"
        header = False
    else:
        sep = ","
        final = ".csv"
        header = self.features[1:] # removendo a coluna
segmento
        # salvando em csv os dados

features.to_csv(f"{self.path_save}/features_{audio_name.split('/')[
-1]}{final}", sep=sep,
                index=False, header=header)

def extrair_features_audio(audio: np.ndarray, sr: int,
data_parallel: list, segmento: int = None):
    """
    Extrai as features de um segmento de áudio
    :param audio: segmento de áudio
    :param sr: taxa de amostragem
    :param data_parallel: lista para armazenar as features de
forma assíncrona
```

```
        :param segmento: número do segmento para organização dos
dados

        :return: list | None

        """

        if data_parallel is not None:

            assert segmento is not None, "É necessário informar o
segmento para usar de forma assíncrona"

            # Compute spectral centroid

            centroid = librosa.feature.spectral_centroid(y=audio,
sr=srate)

            # FEATURE 1: Spectral Centroid

            start = time.time()

            # 1.1. Compute mean frequency

            meanfreq = np.mean(centroid)

            # 1.2. Compute standard deviation of frequency

            sd = np.std(centroid)

            # 1.3 Compute Median Frequency

            median = np.median(centroid)

            # 1.4 Compute max and min

            maxfreq = np.max(centroid)

            minfreq = np.min(centroid)

            #print(f"F1 demorou {time.time() - start:.2f} segundos")

            # FEATURE 2: Spectral Flatness

            # Compute spectral flatness

            start = time.time()

            sfm = librosa.feature.spectral_flatness(y=audio)

            # Podemos obter a média, mediana, desvio padrão do spectral
flatness

            mean_sfm = np.mean(sfm)
```

```
median_sfm = np.median(sfm)
sd_sfm = np.std(sfm)
#print(f"F2 demorou {time.time() - start:.2f} segundos")

# FEATURE 3: Spectral Rolloff
# Compute spectral rolloff
start = time.time()
spec_rolloff_099 = librosa.feature.spectral_rolloff(y=audio,
sr=rate, roll_percent=0.99)
spec_rolloff_001 = librosa.feature.spectral_rolloff(y=audio,
sr=rate, roll_percent=0.01)
# podemos obter a média, mediana, desvio padrão, mínimo e
máximo do spectral rolloff
mean_rolloff_099 = np.mean(spec_rolloff_099)
median_rolloff_099 = np.median(spec_rolloff_099)
sd_rolloff_099 = np.std(spec_rolloff_099)
min_rolloff_099 = np.min(spec_rolloff_099)
max_rolloff_099 = np.max(spec_rolloff_099)
# O outro extremo: 0.01
mean_rolloff_001 = np.mean(spec_rolloff_001)
median_rolloff_001 = np.median(spec_rolloff_001)
sd_rolloff_001 = np.std(spec_rolloff_001)
min_rolloff_001 = np.min(spec_rolloff_001)
max_rolloff_001 = np.max(spec_rolloff_001)
#print(f"F3 demorou {time.time() - start:.2f} segundos")

# FEATURE 4: Fundamental Frequency - PITCH
# Compute fundamental frequency (pitch)
start = time.time()
# f0, voiced_flag, voiced_probs = librosa.pyin(audio,
fmin=librosa.note_to_hz('C2'),
```

```
#
fmax=librosa.note_to_hz('C7'))
    # # Podemos obter a média, mediana, desvio padrão, mínimo e
máximo do fundamental frequency
    # # ignorar ou remover nan
    # f0 = f0[~np.isnan(f0)]
    # mean_f0 = np.mean(f0)
    # median_f0 = np.median(f0)
    # sd_f0 = np.std(f0)
    # min_f0 = np.min(f0)
    # max_f0 = np.max(f0)
    #print(f"F4 demorou {time.time() - start:.2f} segundos")
    if data_parallel is not None:
        data_parallel.append(
            [segmento, meanfreq, sd, median, maxfreq, minfreq,
mean_sfm, median_sfm, sd_sfm, mean_rolloff_099,
            median_rolloff_099, sd_rolloff_099,
min_rolloff_099, max_rolloff_099, mean_rolloff_001,
            median_rolloff_001,
            sd_rolloff_001, min_rolloff_001,
max_rolloff_001,])# mean_f0, median_f0, sd_f0, min_f0, max_f0]
    else:
        return [meanfreq, sd, median, maxfreq, minfreq,
mean_sfm, median_sfm, sd_sfm, mean_rolloff_099,
            median_rolloff_099, sd_rolloff_099,
min_rolloff_099, max_rolloff_099, mean_rolloff_001,
            median_rolloff_001, sd_rolloff_001,
min_rolloff_001, max_rolloff_001,]
        # mean_f0, median_f0, sd_f0, min_f0, max_f0]
```

```
def FC_parallel(audio: str, path_audios: str,
path_save_features: str, audios_list: list = None):
    """
    Função para executar a extração de features clássicas de
    forma paralela.
    :param audio: nome do áudio
    :param path_audios: caminho para os áudios
    :param path_save_features: caminho para salvar as features
    :param audios_list: opcional: lista de áudios
    """

    fc_obj = FeaturesClassicas(
        path_save_features=path_save_features,
        path_audios=path_audios,
        audios_list=audios_list
    ) # instanciando o objeto de extrair features clássicas
    fc_obj.run(audio)

if __name__ == '__main__':
    path_ = "/dados/bronze/anatel"
    path_save_features_ = "/dados/silver/anatel/features_audios"
    audios_list_ = os.listdir(path_)

    with
concurrent.futures.ProcessPoolExecutor(max_workers=MAX_THREADS) as
executor:
        # executando as threads
        for audio in audios_list_:
            executor.submit(FC_parallel, audio, path_,
path_save_features_)
```

step1_audio_processing/URA/URA_removal.py

```
# usando Kmeans para clusterizar os segmentos
from sklearn.cluster import KMeans
import numpy as np
import pandas as pd
import os
import librosa
import soundfile as sf

class RemoveURA:
    def __init__(self, path_csvs: str, csvs: list = None):
        """
        Essa classe é responsável por remover a URA dos áudios.
        :param path_csvs:
            :param csvs: opcional: uma lista com os nomes dos
arquivos csvs. Se None, lista todos os csvs do diretório.
        """
        self.kmeans = None
        self.path_csvs = path_csvs
        self.filter = np.array([-1, -1, 1, 1])
        self.csvs = csvs

    def run_default(self, TIME_SEGMENT: int) -> pd.DataFrame:
        self.find_csvs()
        self.initiate_kmeans()
        df_final = pd.DataFrame(columns=["audio",
"inicio_atendimento"])
        for csv in self.csvs:
            df = self.load_csv(csv)
```

```
        labels = self.clusteriza_segmentos(df)
        point_of_change = self.filtering(labels)
        # printando o ponto de troca e estimando o tempo que
a URA acaba
        # print(f"O ponto de troca é: {point_of_change}")
        # print(f"{csv} - O tempo que a URA acaba é:
{point_of_change * TIME_SEGMENT} segundos")
        df_aux = pd.DataFrame({"audio": [csv],
"inicio_atendimento": [point_of_change * TIME_SEGMENT]})
        # concatenando
        df_final = pd.concat([df_final, df_aux])

        # salvando o resultado
        df_final.to_csv("trocas_URA_atendimento.csv",
index=False)

        return df_final

    def find_csvs(self) -> None:
        """
        Essa função busca os arquivos csvs no diretório
fornecido.
        :return: None
        """
        if self.csvs is None:
            self.csvs = os.listdir(self.path_csvs)

    def load_csv(self, csv):
        """
        Essa função carrega um csv.
        :return: pd.DataFrame com o csv carregado
        """
```

```
df = pd.read_csv(os.path.join(self.path_csvs, csv), )
return df

def initiate_kmeans(self, n_clusters: int = 2, random_state:
int = 0, n_init: int = 10):
    self.kmeans = KMeans(n_clusters=n_clusters,
random_state=random_state, n_init=n_init)

def clusteriza_segmentos(self, df: pd.DataFrame) ->
np.ndarray:
    """
        Essa função clusteriza os segmentos de áudio com o
algoritmo Kmeans.
        Essa abordagem tenta clusterizar os áudios em 2 classes:
URA e não URA.
        Faz essa clusterização com base somente nas features do
PRÓPRIO áudio, sem levar em consideração os outros.
        :param df: pd.DataFrame com os segmentos de áudio
        :return: np.ndarray com as classes encontradas
    """
    X = df
    kmeans = self.kmeans.fit(X)
    return kmeans.labels_

def filtering(self, binary_sequence) -> int:
    """
        Essa função filtra a sequência binária fornecida com o
filtro definido na classe.
        Basicamente, realiza uma convolução 1D com o filtro e
retorna o resultado.
```

```
        :param binary_sequence: uma sequência binária com as
classes encontradas no áudio

        :param threshold: threshold para encontrar o ponto de
troca

        :return: o index do ponto de troca
        """
        # Aplicar a convolução 1D com o filtro
        convolution_result = np.convolve(binary_sequence,
self.filter, mode='valid')

        # Aplicar uma penalidade conforme aumenta o index dos
elementos
        penalty = np.arange(len(convolution_result)) * 0.01
        convolution_result = abs(convolution_result) - penalty
        # buscando o argmax
        point_of_change = np.argmax(convolution_result) +
(len(self.filter)//2)
        return point_of_change

    def cut_audio(self, audio: str, path_audios: str, time: int,
path_save: str):
        """
        Essa função recorta o áudio no ponto de troca.
        :param audio: o nome do áudio
        :param path_audios: o caminho dos áudios
        :param time: o tempo que a URA acaba
        :param path_save: o caminho para salvar o áudio recortado
        :return: None
        """
        if not os.path.exists(path_save):
            os.makedirs(path_save)
```

```
        wave, sr = librosa.load(os.path.join(path_audios,  
audio), sr=None, mono=False)  
        wave = wave[:, time * sr:]  
        wave = wave.T # https://stackoverflow.com/a/75850841  
        # salvando em mp3  
        path = os.path.join(path_save, audio.replace(".opus",  
".mp3"))  
        sf.write(path, wave, int(sr), format='mp3')  
  
if __name__ == '__main__':  
    path_csvs_ = '../Cientificos/ExtracaoFeatures/features'  
                                                    csvs_  
["features_1e54b932-d382-474d-8e9f-a69e733c7c73.opus.csv"]  
    remover = RemoveURA(path_csvs_, csvs_)  
    df = remover.run_default(5)  
    # percorrendo o df e recortando os áudios  
    path_audios_ = "../audios_analise_qualidade"  
                                                    path_save_cut_audios_  
r"C:\Users\alexv\PycharmProjects\AnaliseAutomatica-RemoveURA\audios  
_analise_qualidade_cut"  
    for idx, row in df.iterrows():  
        audio = row["audio"].replace("features_",  
").replace(".csv", "")  
        inicio_atendimento = row["inicio_atendimento"]  
        # recortando o audio  
        remover.cut_audio(audio, path_audios=path_audios_,  
time=inicio_atendimento, path_save=path_save_cut_audios_)
```

step1_audio_processing/utils/temp_audio.py

```
import os
from fastapi import UploadFile

PATH = os.getcwd().split('AutoInstruct4LLMs')[0] +
'AutoInstruct4LLMs/'
temp_audio_folder = os.path.join(PATH, 'dados/bronze/temp_audios/')

def create_temp_audio(audio, filename) -> str:
    temp_audio_path = temp_audio_folder + filename
    with open(temp_audio_path, 'wb') as temp_audio:
        temp_audio.write(audio.read())
    return temp_audio_path

def delete_temp_audio(filename):
    temp_audio_path = temp_audio_folder + filename
    os.remove(temp_audio_path)
```

step1_audio_processing/channel_separation.py

```
import torchaudio
import os
from tqdm import tqdm

def separa_canais(PATH_SRC: str, PATH_DST: str) -> None:
    """
    Essa função separa os canais de áudio e salva em mp3, organizando
    em pastas com o nome do arquivo e os canais salvos.
    :param PATH_SRC: a pasta onde estão os áudios stereos
    :param PATH_DST: a pasta onde serão salvos os subdiretórios com
    os canais separados
    :return: None
    """

    if not os.path.exists(PATH_DST):
        os.makedirs(PATH_DST)

    audios = os.listdir(PATH_SRC)
    audios = [audio for audio in audios if audio.endswith('.mp3')]

    ja_foram = []
    for audio in tqdm(audios):
        name_audio = audio.split('.')[0]
        if name_audio in ja_foram:
            continue
        ja_foram.append(name_audio)
        # separando os canais de áudio e salvando novamente em mp3
        wav, sr = torchaudio.load(os.path.join(PATH_SRC, audio))
```

```
assert wav.shape[0] == 2, f"O áudio {audio} não é stereo"
audio_cliente = wav[:1, :] # idx 0
audio_atendente = wav[1:, :] # idx 1
del wav

# salvando os audios
if not os.path.exists(os.path.join(PATH_DST, name_audio)):
    os.mkdir(os.path.join(PATH_DST, name_audio)) # criando
uma pasta para o áudio ser salvo
    torchaudio.save(os.path.join(PATH_DST, name_audio,
name_audio + "_cliente_cut.mp3"), audio_cliente, sample_rate=sr)
    torchaudio.save(os.path.join(PATH_DST, name_audio,
name_audio + "_atendente_cut.mp3"), audio_atendente,
sample_rate=sr)
# deletando os audios separados
os.remove(os.path.join(PATH_SRC, audio))
```

step1_audio_processing/denoising.py

```
import os
import torchaudio
import torch
from denoiser import pretrained
from denoiser.dsp import convert_audio
from tools import check_path_exists
from sys import getsizeof
import requests

class Denoising:
    def __init__(self, tmp_files_path: str =
'./dados/bronze/tmp_files'):
        """
        Essa classe realiza a remoção de ruído de um áudio utilizando
diferentes técnicas de denoising.
        :param tmp_files_path:
        """
        self.tmp_files_path = tmp_files_path

        # declarando o modelo de denoising
        self.denoiser_model = None

    @staticmethod
    def denoise_nuvem(audio_path: str = None, save_path: str = None,
audio_name: str = None) -> None:
        """
        Essa função realiza a remoção de ruído de um áudio utilizando
diferentes técnicas de denoising.
```

```
Salva o áudio denoised no mesmo diretório do áudio original.
:param save_path: o caminho onde o áudio denoised será salvo.
:param audio_path: o caminho do arquivo de áudio a ser
removido o ruído.
:param audio_name: o nome do arquivo de áudio a ser removido
o ruído.
:return: None.
"""
    audio_file = open(os.path.join(audio_path, audio_name),
"rb")

url = "http://200.137.197.69:43016/denoiser"

# Realize a requisição POST com o arquivo de áudio
files = {"client_speech": ('audio.opus', audio_file)}
response = requests.post(url, files=files)

# Verifique a resposta
if response.status_code == 200:
    save_path = os.path.join(save_path,
audio_name.replace('.opus', '.ogg'))
    with open(save_path, 'wb') as f:
        f.write(response.content)
else:
    print("Erro na requisição POST. Código de status:",
response.status_code)
    print(response.text)
    exit(-1)

def denoise(self, audio_path: str = None, save_path: str = None,
tensor: torch.tensor = None,
```

```
        sample_rate: int = None, audio_name: str = None) ->
torch.tensor:
    """
        Essa função realiza a remoção de ruído de um áudio utilizando
        diferentes técnicas de denoising.

        Salva o áudio denoised no mesmo diretório do áudio original.

        :param save_path: opcional: o caminho onde o áudio denoised
        será salvo.

        :param audio_path: opcional: o caminho do arquivo de áudio a
        ser removido o ruído.

        :param tensor: opcional: tensor do áudio, se não for passado
        o caminho do áudio.

        :param sample_rate: opcional: sample rate do áudio, se não
        for passado o caminho do áudio.

        :param audio_name: opcional: o nome do arquivo de áudio a ser
        removido o ruído.

        :return: o tensor do áudio denoised.
    """

    if not self.denoiser_model:
        self.denoiser_model = pretrained.dns64().cuda() if
torch.cuda.is_available() else pretrained.dns64() # TODO: pq esse
modelo em específico?

    # Denoiser usando do facebook

    if audio_path:
        wav, sample_rate = torchaudio.load(audio_path)
        wav = wav.cuda() if torch.cuda.is_available() else wav
    else:
        # verificando se o áudio está no cuda
        if tensor.is_cuda:
            wav = tensor
        else:
```

```
wav = tensor.cuda() if torch.cuda.is_available()
else tensor
    if sample_rate != 16000:
        wav = convert_audio(wav, sample_rate,
self.denoiser_model.sample_rate, self.denoiser_model.chin)

        denoised_audio = torch.tensor([]).cuda() if
torch.cuda.is_available() else torch.tensor([])
        with torch.no_grad():
            # dividindo o audio em trechos de 30 segundos e
aplicando o denoiser em cada trecho
            trecho = 120
            for i in range(0, wav.shape[1], sample_rate * trecho):
                denoised_audio = torch.cat((denoised_audio,
self.denoiser_model(wav[:, i:i + sample_rate * trecho])[0]),
                    dim=1)

            del wav

        if save_path:
            # salvando a versão denoised do áudio no mesmo diretório
do áudio original, com frequencia de amostragem
            # de 16000
            check_path_exists(save_path)
            save_path = os.path.join(save_path, audio_name)
            torchaudio.save(save_path, denoised_audio.cpu(), 16000,
format="mp3")
        else:
            return denoised_audio

def separate_channels(self, audio_name: str = None, audio_wave:
torch.tensor = None, sr: int = None,
```

```
        path_voicerecords: str = None, path_save:
str = None):
    """
        Essa função separa os canais de um áudio em dois arquivos
diferentes.
        :param audio_name: opcional: o nome do arquivo de áudio a ser
separado.
        :param audio_wave: opcional: o tensor do áudio a ser
separado.
        :param sr: opcional: o sample rate do áudio a ser separado.
        :param path_voicerecords: opcional: o caminho da pasta onde
os áudios serão salvos.
        :param path_save: opcional: o caminho da pasta onde os áudios
serão salvos.
        :return: None | tuple(torch.tensor(cliente, atendente)) ->
caso seja passado o tensor do áudio, retorna o tensor do áudio
separado.
    """
    if not path_voicerecords and audio_name:
        audio_path = audio_name
    elif audio_name:
        audio_path = os.path.join(path_voicerecords, audio_name)
    # Denoiser do facebook
    if audio_name:
        audio_wave, sr = torchaudio.load(audio_path)

    # extraindo os canais do áudio (cliente e atendente)
    audio_cliente = audio_wave[:1, :] # idx 0
    audio_atendente = audio_wave[1:, :] # idx 1

    if sr != 16000:
```

```
        audio_cliente = torchaudio.transforms.Resample(sr,
16000) (
            audio_cliente)
        audio_atendente = torchaudio.transforms.Resample(sr,
16000) (
            audio_atendente)

        if path_save:
            torchaudio.save(self.tmp_files_path +
f'/cliente_aux_{audio_name}', audio_cliente, 16000)
            torchaudio.save(self.tmp_files_path +
f'/atendente_aux_{audio_name}', audio_atendente, 16000)
        else:
            return audio_cliente, audio_atendente

def denoisar(audio_name: str = None, dns_object: Denoising = None,
path_bronze: str = None, path_silver: str = None, mp=False):
    if mp:
        dns_object = Denoising()
        # split dos canais
        audio_cliente, audio_atendente =
dns_object.separate_channels(audio_name=audio_name,
path_voicerecords=path_bronze)
        # denoise de cada canal separadamente e salva na pasta silver
        audio_cliente = dns_object.denoise(tensor=audio_cliente,
sample_rate=16000)
        audio_atendente = dns_object.denoise(tensor=audio_atendente,
sample_rate=16000)
```

```
# unindo e salvando os canais denoised
    audio_denoised = torch.cat((audio_cliente, audio_atendente),
dim=0)

    audio_denoised = audio_denoised.cpu()
    del audio_cliente, audio_atendente
    audio_name = audio_name.replace('.opus', '.mp3')
        torchaudio.save(os.path.join(path_silver, audio_name),
audio_denoised, 16000, format="mp3")

def denoisar_nuvem(audio_name: str = None, dns_object: Denoising =
None, path_bronze: str = None,
        path_silver: str = None):
    # split dos canais
        audio_cliente, audio_atendente =
dns_object.separate_channels(audio_name=audio_name,
path_voicerecords=path_bronze)

    # salvando os canais separados em ogg
    save_path_bronze = os.path.join(path_bronze, "tmp")
    if not os.path.exists(save_path_bronze):
        os.makedirs(save_path_bronze)

        save_file = os.path.join(save_path_bronze,
audio_name.replace(".opus", ""))
        torchaudio.save(f"{save_file}_cliente.ogg", audio_cliente,
16000, format="ogg")
        torchaudio.save(f"{save_file}_atendente.ogg", audio_atendente,
16000, format="ogg")

    # denoisando de fato
```

```
        dns_object.denoise_nuvem(audio_path=save_path_bronze,
save_path=path_silver,
                                audio_name=audio_name.replace(".opus",
"_cliente.ogg"))
        dns_object.denoise_nuvem(audio_path=save_path_bronze,
save_path=path_silver,
                                audio_name=audio_name.replace(".opus",
"_atendente.ogg"))

if __name__ == '__main__':
    dns = Denoising()
    path_bronze = "../dados/bronze/anatel"
    path_silver = "../dados/silver/anatel"
    audio = "0a0d8d80-0dd2-4cf2-80bd-63de9f2ea37c.opus"
        denoisar_nuvem(audio_name=audio, dns_object=dns,
path_bronze=path_bronze, path_silver=path_silver)
    # limpando o arquivo temporário
    input()
    os.remove(os.path.join(path_bronze, "tmp"))
```

step1_audio_processing/main_audio.py

```
import pandas as pd
from step1_audio_processing.denoising import Denoising, denoisar,
denoisar_nuvem
import os
import time
from tqdm import tqdm
import concurrent.futures
from step1_audio_processing.URA.features_extractor import
FC_parallel
from step1_audio_processing.URA.URA_removal import RemoveURA
from step1_audio_processing.channel_separation import separa_canais
from tools import check_path_exists
from step1_audio_processing.transcricao import STT

"""
path_audio = './dados/bronze/TESTE'
path_audio_denoised = './dados/silver/teste_denoised'
path_features = './dados/silver/teste_features_audios'
path_cut_audios = './dados/silver/teste_cut_audios'
path_transcricoes = 'dados/gold/teste_transcricoes_backup'
"""

def main(path, path_denoised, path_features, path_cut_audios,
path_transcricoes):
    # path_audio das ligações brutas
    check_path_exists(path, path_denoised, path_features,
path_cut_audios, path_transcricoes)
```

```
# perguntando se deseja skipar alguns passos
# skip = input("Deseja skipar alguns passos?\n")
#
#         "1- Denoiser\n"
#
#         "2- Extração de features\n"
#
#         "3- Remoção de URA\n"
#
#         "4- Separar os canais\n"
#
#         "5- Transcrição\n")

skip = ""

# ##### STEP 1: DENOISER #####
#####

# instanciando o objeto do modelo de denoising
if not (skip and '1' in skip):
    print("\033[92m" + "Denoisando..." + "\033[0m")
    dns = Denoising()

# listando os arquivos de áudio no diretório
audios = os.listdir(path)

    audios = [audio for audio in audios if
audio.endswith('.opus')]
    print(f"Total de áudios: {len(audios)}")

# verificando quais já foram denoised
try:
    denoised = os.listdir(path_denoised)
    audios = [audio for audio in audios if
audio.replace('.opus', '_cliente_denoised.wav') not in denoised]
except FileNotFoundError:
    pass

print(f"Total de áudios a serem denoised: {len(audios)}")
```

```
start = time.time()

# denoising de cada áudio de forma sequencial
for audio in tqdm(audios):
    denoisar(audio, dns, path_bronze=path,
path_silver=path_denoised)
    print(f"Tempo total sequencial: {time.time() - start}")

# ##### STEP 2: EXTRAÇÃO DE FEATURES #####
#####
MAX_THREADS = 4
# extraindo features dos áudios
path_ligacoes_denoised = path_denoised # fonte das ligações
denoised
path_save_features_ = path_features # onde irá salvar os csvs
com as features
if not (skip and '2' in skip):
    print("\033[92m" + "Extraindo features..." + "\033[0m")
    audios_list_ = os.listdir(path_ligacoes_denoised)
    audios_list_ = [audio for audio in audios_list_ if
(audio.endswith('.wav') or
audio.endswith('.mp3') or
audio.endswith('.opus'))]
    with
concurrent.futures.ProcessPoolExecutor(max_workers=MAX_THREADS) as
executor:
        # executando as threads
        for audio in audios_list_:
            executor.submit(FC_parallel, audio,
path_ligacoes_denoised, path_save_features_)
```

```
# for audio in tqdm(audios_list_):
    #         FC_parallel(audio, path_ligacoes_denoised,
path_save_features_)

# ##### STEP 3: REMOÇÃO DA URA #####
#####

if not (skip and '3' in skip):
    print("\033[92m" + "Removendo URA..." + "\033[0m")
    # removendo a URA dos áudios
    remover = RemoveURA(path_save_features_)
    df = remover.run_default(5)
    # percorrendo o df e recortando os áudios
    path_save_cut_audios_ = path_cut_audios
    for idx, row in df.iterrows():
        audio = row["audio"].replace("features_",
"" ).replace(".csv", "")
        inicio_atendimento = row["inicio_atendimento"]
        # recortando o audio
        remover.cut_audio(audio,
path_audios=path_ligacoes_denoised,
time=inicio_atendimento,
path_save=path_save_cut_audios_)

# ##### STEP 4: SEPARANDO OS CANAIS #####
#####

# separando os canais de áudio
PATH_SRC = path_cut_audios
PATH_DST = path_cut_audios
if not (skip and '4' in skip):
    print("\033[92m" + "Separando os canais..." + "\033[0m")
    separa_canais(PATH_SRC, PATH_DST)
```

```
# ##### STEP 5: TRANSCRIÇÃO
#####

PATH_GOLD = path_transcricoes
if not (skip and '5' in skip):
    print("\033[92m" + "Transcrevendo..." + "\033[0m")
    # instanciando o objeto do modelo de transcrição
    path_audios = path_cut_audios
    stt = STT(path_voicerecords=PATH_DST)
    # listando os arquivos de áudio no diretório
    audios_dirs = os.listdir(PATH_DST)
    audios_dir = [audio_dir for audio_dir in audios_dirs if
                  os.path.isdir(os.path.join(path_audios,
audio_dir))] # filtrando apenas pastas

    print(f"Total de áudios para fazer transcrição:
{len(audios_dir)}")

    for audio in tqdm(audios_dir):
        # transcrevendo o atendente
        path_atendente = os.path.join(PATH_DST, audio, audio +
"_atendente_cut.mp3")
        transcricao_atendente =
stt.get_local_insanelly_fast_whisper_transcription(path_atendente)
        # transcrevendo o cliente
        path_cliente = os.path.join(PATH_DST, audio, audio +
"_cliente_cut.mp3")
        transcricao_cliente =
stt.get_local_insanelly_fast_whisper_transcription(path_cliente)

    # colocando as transcrições no df
```

```
df_aux_atendente = pd.DataFrame(transcricao_atendente,
columns=["start", "end", "transcription"])
df_aux_atendente["role"] = "atendente"
df_aux_cliente = pd.DataFrame(transcricao_cliente,
columns=["start", "end", "transcription"])
df_aux_cliente["role"] = "cliente"
# concatenando os df
df_aux = pd.concat([df_aux_atendente, df_aux_cliente],
ignore_index=True)
# ordenando o df
df_aux = df_aux.sort_values(by=["start"])
# salvando o df
df_aux.to_csv(os.path.join(PATH_GOLD, audio + ".csv"),
index=False)

if __name__ == '__main__':
    path = '../dados/bronze/anatel_big'
    path_denoised = '../dados/silver/anatel_big_denoised'
    path_features = '../dados/silver/anatel_teste_final_features'
    path_cut_audios = '../dados/silver/anatel_teste_final_cut'
    path_transcricoes =
'../dados/gold/anatel_teste_final_transcricoes'
    main(path, path_denoised, path_features, path_cut_audios,
path_transcricoes)
```

step1_audio_processing/transcricao.py

```
import torch
from transformers import pipeline
#         from         step1_audio_processing.utils.temp_audio         import
create_temp_audio, delete_temp_audio
from faster_whisper import WhisperModel
import os
import requests
from transformers.utils import is_flash_attn_2_available

class STT:
    def __init__(self, path_voicerecords: str, sample_rate: int =
16000):
        self.path_voicerecords = path_voicerecords
        if not os.path.exists(self.path_voicerecords):
            os.makedirs(self.path_voicerecords)

        self.sample_rate = sample_rate

        self.DURACOES_AUDIOS = []
        print('----- IS GPU ENABLED: -----')
        print(torch.cuda.is_available())
        self.device = "cuda" if torch.cuda.is_available() else "cpu"
        self.faster_whisper = None
        self.IFwhisper = None

    def get_local_faster_whisper_transcription(self, audio) -> str:
        transcript = ""
        # caso o modelo não tenha sido carregado ainda
```

```
        if self.faster_whisper is None:
            self.IFwhisper = None
            self.faster_whisper = WhisperModel("large-v3",
device=self.device, compute_type="int8")

            assert ".wav" in audio.filename or ".mp3" in audio.filename,
"The file must have an .wav or .mp3 extension"
            segments, _ = self.faster_whisper.transcribe(audio,

vad_filter=True,

vad_parameters=dict(min_silence_duration_ms=2200,

threshold=0.7,

min_speech_duration_ms=250,

window_size_samples=1024,

speech_pad_ms=400),

language='pt')

            result = []
            for segment in segments:
                result.append((segment.start, segment.end,
segment.text))

            return result

def get_local_insanelly_fast_whisper_transcription(self, audio)
-> list:
```

```
        assert ".wav" in audio or ".mp3" in audio, "The file must
have an .wav or .mp3 extension"

transcript = ""

# verificando se o modelo não foi carregado ainda
if self.IFwhisper is None:
    self.faster_whisper = None
    self.IFwhisper = pipeline(
        "automatic-speech-recognition",
        model="openai/whisper-large-v3",
        # select checkpoint from
https://huggingface.co/openai/whisper-large-v3#model-details
        torch_dtype=torch.float16,
        device="cuda:0", # or mps for Mac devices
        model_kwargs={"attn_implementation":
"flash_attention_2"} if is_flash_attn_2_available() else {
        "attn_implementation": "sdpa"},
    )

# gerando os outputs
outputs = self.IFwhisper(audio,
                           chunk_length_s=30,
                           batch_size=24,
                           generate_kwargs={"language":
"portuguese"},
                           return_timestamps=True)

chunks = outputs['chunks']
del outputs

# formatando no formato start, end, transcription
outputs = [(chunk['timestamp'][0], chunk['timestamp'][1],
chunk['text']) for chunk in chunks]
```

```
return outputs

def get_transcription_nuvem(self, audio_path):
    """
    Essa função recebe o tensor referente a um áudio e retorna a
    transcrição do mesmo.
    :param audio_path: caminho do áudio
    :return: tupla contendo a transcrição e a duração do áudio.
    """

    audio_file = open(audio_path, "rb")

    url =
"http://200.137.197.69:43016/local_faster_whisper_transcription"

    # Realize a requisição POST com o arquivo de áudio
    files = {"client_speech": ('audio.wav', audio_file)}
    response = requests.post(url, files=files)

    # Verifique a resposta
    if response.status_code == 200:
        return response.json()
    else:
        raise ("Erro na requisição POST. Código de status:",
response.status_code)
```

step2_text_preprocessing/fix_unicode_mapper.py

```
# Código extraído do repositório Data-Juicer, disponível em:  
#  
https://github.com/alibaba/data-juicer/blob/main/data\_juicer/ops/mapper/fix\_unicode\_mapper.py  
  
import ftfy  
  
def fix_unicode(text):  
    text = ftfy.fix_text(text)  
    return text
```

step2_text_preprocessing/main_preprocessing.py

```
# esse código é responsável por realizar o pré-processamento de
# textos passados para ele, baseando-se nos problemas
# pertinentes ao projeto.
from step2_text_preprocessing import trans_repetition_error
from step2_text_preprocessing import fix_unicode_mapper
from step2_text_preprocessing import punctuation_normalization_mapper
from step2_text_preprocessing import remove_nomes
import os
import logging
import datetime
import re

cwd = os.getcwd().split("AutoInstruct4LLMs")[0] +
"AutoInstruct4LLMs/"

class Preprocessamento:
    """
    Classe responsável por realizar o pré-processamento de textos.
    Suas etapas são:
    1) Remoção de repetições equidistantes de palavras e caracteres.
    2) Correção de caracteres unicode.
    3) Normalização de pontuação.
    4) Remoção de números.
    5) Remoção de nomes.
    """
```

```
def __init__(self, logs_path: str) =
f"{cwd}/logs/preprocessamento_texto"):
    # criando a pasta de logs
    if not os.path.exists(logs_path):
        os.makedirs(logs_path)

    # instanciando objeto de remoção de nomes
    self.remove_nomes = remove_nomes.NomeIdentifier()

    # criando o logger
    self.logger = logging.getLogger(__name__)
    self.logger.setLevel(logging.INFO)

    formatter =
logging.Formatter('%(asctime)s: %(levelname)s: %(message)s')

    datet =
datetime.datetime.now().strftime("%Y-%m-%d_%H-%M-%S")

    file_handler =
logging.FileHandler(f"{logs_path}/preprocessamento_{datet}.log",
encoding='utf-8')

    file_handler.setFormatter(formatter)
    self.logger.addHandler(file_handler)

def preprocesar_texto(self, texto: str, cliente_id: str) ->
str:
    """
    Função para realizar o pré-processamento de um texto.
    :param texto: texto que será pré-processado
    :param cliente_id: id do cliente para logging
    :return: texto pré-processado
    """
    original_text = texto
```

```

text =
trans_repetition_error.filtra_repeticoes_equidistantes(texto)
    text = fix_unicode_mapper.fix_unicode(text)
text =
punctuation_normalization_mapper.normalize_punctuation(text)
    # anonimizando CPFs
    text = self.remove_numeros(text)
    # anonimizando nomes
    text = self.remove_nomes.remover(text)
        self.logger.info(f"\nID: {cliente_id}\nTexto original:
<<{original_text}>>\nTexto pré-processado: <<{text}>>\n\n")
    return text

def remove_numeros(self, texto: str) -> str:
    """
    Função para remover os numeros presente no texto.
    :param texto: texto que será pré-processado
    :return: texto pré-processado
    """
    # removendo usando re
    texto = re.sub(r'\d+', '', texto)
    return texto
```

step2_text_preprocessing/punctuation_normalization_mapper.py

```
# Código extraído do repositório Data-Juicer, disponível em:  
#  
https://github.com/alibaba/data-juicer/blob/main/data\_juicer/ops/mapper/punctuation\_normalization\_mapper.py  
  
punctuation_unicode = {  
    ',': ',',  
    '。': '.',  
    '\\': '\\',  
    '"': '"',  
    "'": "'",  
    '`': '`',  
    '«': '«',  
    '»': '»',  
    '1': '1',  
    ']' : ']',  
    '[' : '[',  
    '《': '《',  
    '》': '》',  
    '<': '<',  
    ':': ':',  
    ':': ':',  
    '?': '?',  
    '!': '!',  
    '(': '(',  
    ')': ')',  
    ';': ';',  
    '-': '-',  
    '_': '_'
```

```
'.' : '.',  
'~' : '~',  
'/' : '/',  
'...' : '...',  
'-' : '-',  
'<' : '<',  
'>' : '>',  
'[' : '[',  
 ]' : ']',  
'%' : '%',  
'▶' : '-',  
}  
  

```

step2_text_preprocessing/remove_nomes.py

```
# esse código é responsável por analisar os textos e identificar os
nomes presentes neles, substituindo por NOME

import re
import os
import pandas as pd

cwd = os.getcwd().split("AutoInstruct4LLMs")[0] +
"AutoInstruct4LLMs/"

"""
Disclaimer:

Tentei utilizar a biblioteca do SPACY como forma de identificar os
nomes, mas não estava funcionando bem, identificando
algumas palavras equivocadamente como nomes e não identificando
alguns nomes como tal. Por isso, optei por utilizar
uma lista de nomes brasileiros extraída do IBGE, que contém nomes
reais.

No entanto, essa lista estava com alguns nomes muito diferentes e
específicos, como "claro", "nao", que faziam com que
outras palavras fossem identificadas como nomes, equivocadamente.

Por isso, optei por utilizar apenas os 500 nomes mais comuns, que
são mais genéricos e não causam esse problema.

MELHORIA:

Uma forma de melhorar a abordagem com spacy é utilizando o NLTK
para avaliar a estrutura sintática da frase antes de
usar o spacy para determinar a função das palavras.
```

```
Um tutorial a ser seguido é
https://towardsdatascience.com/named-entity-recognition-with-nltk-and-spacy-8c4a7d88e7da
"""

class NomeIdentifier:
    def __init__(self):
        self.lista_nomes = self.__carrega_nomes()

    def remover(self, texto):
        """
        Função para remover os nomes presentes no texto.
        :param texto: texto que será pré-processado
        :return: texto pré-processado
        """
        for word in texto.split():
            # removendo pontuação
            word = re.sub(r'^\w\s$', '', word)
            original_word = word
            nome_acentuado = word

            # substituindo acentos e caracteres especiais por suas
formas sem acento
            word = re.sub(r'[áâãäå]', 'a', word)
            word = re.sub(r'[éèêë]', 'e', word)
            word = re.sub(r'[íîï]', 'i', word)
            word = re.sub(r'[óòôõö]', 'o', word)
            word = re.sub(r'[úûü]', 'u', word)
            word = re.sub(r'[ç]', 'c', word)
            nome = word.lower()
```

```
        nome_acentuado = nome_acentuado.lower()
        if nome in self.lista_nomes or nome_acentuado in
self.lista_nomes:
            texto = texto.replace(original_word, "NOME")
        return texto

    @staticmethod
    def __carrega_nomes():
        """
        Função para carregar a lista de nomes.
        :return: lista de nomes
        """
        # usando apenas os 500 nomes mais comuns
        df_mas =
pd.read_csv(f"{cwd}/dados/gold/nomes-brasileiros-ibge/ibge-mas-1000
0.csv", nrows=500)
        df_fem =
pd.read_csv(f"{cwd}/dados/gold/nomes-brasileiros-ibge/ibge-fem-1000
0.csv", nrows=500)
        lista_nomes = list(df_mas['nome']) + list(df_fem['nome'])
        lista_nomes = [nome.lower() for nome in lista_nomes]

        return lista_nomes
```

step2_text_preprocessing/trans_repetition_error.py

```
import time

def filtra_repeticoes_equidistantes(frase: str) -> str:
    """
        Encontra as repetições de palavras que são equidistantes em uma
        frase. Para esse filtro funcionar, ele parte
        do pressuposto que as repetições de palavras que são
        equidistantes são alucinações do Whisper.
        :param frase: frase a ser analisada.
        :return: frase sem as repetições equidistantes.
    """
    palavras = frase.split()
    repeticoes = {}

    # criando um dicionário com as palavras e seus respectivos
    índices
    for i, palavra in enumerate(palavras):
        if palavra not in repeticoes:
            repeticoes[palavra] = [i]
        else:
            repeticoes[palavra].append(i)

    # instanciando a lista que receberá os índices das palavras que
    se repetem equidistantemente
    resultado = []

    # iterando sobre o dicionário de repetições, selecionando apenas
    as palavras que se repetem pelo menos 4 vezes
```

```
for palavra, indices in repeticoes.items():
    distancias_repeticoes = {}
    if len(indices) >= 4: # selecionando apenas as palavras que
se repetem pelo menos 4 vezes
        # verificando a distância entre as repetições. Se for
equidistante, será adicionado na lista de resultados
        for i in range(1, len(indices)):
            dist = indices[i] - indices[i - 1] # calculando a
distância entre as repetições
            if dist not in distancias_repeticoes:
                distancias_repeticoes[dist] = [indices[i - 1],
indices[i]]
            else:
                distancias_repeticoes[dist].append(indices[i -
1])
                distancias_repeticoes[dist].append(indices[i])

        # verificando quantas repetições equidistantes tem para
cada distância
        for _, indices in distancias_repeticoes.items():
            if len(set(indices)) >= 5: # arbitrário, limiar de
repetições equidistantes para determinar se é uma alucinação
                resultado += list(set(indices))

# filtrando as palavras que se repetem equidistantemente
for idx in resultado:
    palavras[idx] = ""
palavras = [palavra for palavra in palavras if palavra != ""]

return " ".join(palavras)
```

```
if __name__ == '__main__':
    frase = "Isso é um teste é um teste é um teste é um teste é um
teste de equidistante de repetições em um teste."
    with open("../aux.txt", encoding="utf-8") as f:
        frase2 = f.read()
    start = time.time()
    res = filtra_repeticoes_equidistantes(frase)
    end = time.time()
    print(f"Tamanho frase: {len(frase)}\nTempo: {end -
start:.3f}\n")
    print(res)

    start = time.time()
    res2 = filtra_repeticoes_equidistantes(frase2)
    end = time.time()
    print(f"Tamanho frase 2: {len(frase2.split())}\nTempo: {end -
start:.3f}\n")
    print(res2)
```

step3_elastic/step0_mapping.py

```
embedding_mapping = {
  "settings": {
    "analysis": {
      "analyzer": {
        "text_analyzer": {
          "tokenizer": "standard",
          "filter": [
            "lowercase",
            "asciifolding",
            "stemmer_text"
          ]
        }
      },
      "filter": {
        "stemmer_text": {
          "type": "stemmer",
          "language": "brazilian"
        }
      }
    }
  },
  "mappings": {
    "dynamic": "strict",
    "properties": {
      "segmento_de_texto": { # o texto transcrito do segmento
de áudio
        "type": "text",
        "fields": {
          "keyword": {
```

```
        "type": "keyword"
    }
},
"persona": { # atendente ou cliente
    "type": "keyword"
},
"categoria": { # demanda ou solução / resposta
    "type": "keyword"
},
"arquivo": {
    "type": "keyword"
},
"fala_original": {
    "type": "text"
},
"embed": {
    "type": "dense_vector",
    "dims": 1536
}
}
}
```

step3_elastic/step1_create_index.py

```
from step0_mapping import embedding_mapping
from elasticsearch import Elasticsearch
from pprint import pprint

if __name__ == '__main__':
    anatel_index = 'anatel_teste_final_index'
    password_elastic = "Vlihd6gtJorsSwrEjSKL"

    client = Elasticsearch("https://localhost:9200",
                           ca_certs="./http_ca.crt",
                           basic_auth=("elastic", password_elastic),
                           verify_certs=False)

    if not client.indices.exists(index=anatel_index):
        client.indices.create(index=anatel_index,
                              settings=embedding_mapping['settings'],
                              mappings=embedding_mapping['mappings'])
        pprint(client.indices.get_mapping(index=anatel_index))
        pprint(client.indices.get_settings(index=anatel_index))
    else:
        print("Index already created.")
        # perguntando se quer apagar o index
        quest = "y" #input("Do you want to delete the index? (y/n)
        ")

        if quest == 'y':
            print("Deleting index...")
            client.indices.delete(index=anatel_index)
```

step3_elastic/step2_search.py

```
from elasticsearch import Elasticsearch
import os
from LLMs import embedding_calculator
import time

cwd = os.getcwd().split("AutoInstruct4LLMs")[0] +
"AutoInstruct4LLMs/"

class DocumentsSearch:
    def __init__(self, index_name: str, password_elastic):
        """
        :param index_name: nome do índice
        """
        assert password_elastic is not None, "Defina a senha do
elasticsearch"
        self.es = Elasticsearch("https://localhost:9200",
ca_certs=f"{cwd}/step3_elastic/http_ca.crt",
basic_auth=("elastic",
password_elastic),
verify_certs=False)
        self.index_name = index_name
        self.emb = embedding_calculator.Ada002(api_key_path=f"{cwd}/key.txt")

    def semantic_search(self, phrase: str, persona: str,
top_n=10):
        """
```

Perform a semantic search in the given index using the provided embedding.

```
:param phrase: A frase para buscar
:param persona: Valor do campo 'persona' para filtrar
(e.g., "atendente", "cliente")
:param top_n: Número de resultados top para retornar
"""
# Calculando o embedding da frase
embedding, _ = self.emb.get_embedding(phrase)
query = {
    "script_score": {
        "query": {
            "bool": {
                "must": {
                    "match_all": {}
                },
            },
            "filter": {
                "term": {
                    "persona": persona
                }
            }
        },
    },
    "script": {
source":
        "cosineSimilarity(params.query_vector, 'embed') + 1.0",
        "params": {"query_vector": embedding}
    }
}
```

```
response = self.es.search(  
    index=self.index_name,  
    body={"query": query, "size": top_n},  
    _source=False # Change to True if you want to see  
the source documents  
)  
  
respostas = []  
for resposta in response['hits']['hits']:  
    hit_id = resposta["_id"]  
    semelhanca = resposta['_score']  
    source = self.es.get(index=self.index_name,  
id=hit_id)["_source"]  
    respostas.append([source, semelhanca])  
  
return respostas  
  
# Execução principal  
if __name__ == "__main__":  
    anatel_search = DocumentsSearch("anatel_index")  
    frase = "Quero reclamar da vivo, eles tão me cobrando caro  
demais e não foi combinado esse preço"  
    persona = "atendente" # Exemplo: filtrar por documentos  
relacionados a 'cliente'  
    start = time.time()  
    results = anatel_search.semantic_search(frase, persona)  
    print(f"{time.time() - start:.2f} segundos")  
    for resultado in results:  
        print(resultado)
```

step4_information_retrieval/prompts/combina_prompt.txt

Você receberá 5 respostas feitas para uma mesma pergunta de um cliente, demarcadas por um número de 1 a 5.

Tudo que você deve fazer é escrever uma única resposta que englobe as informações de todas as 5 outras respostas.

Ou seja, entenda o que cada uma das 5 respostas base possuem e escreva uma única resposta que seja coerente.

Pode ser que algumas respostas sejam apenas ruídos, ou seja, não tenham informações relevantes.

Portanto, você deve ser capaz de identificar quais respostas são relevantes e quais não são para construir a resposta final.

Retorne apenas a resposta final e nada além disso.

step4_information_retrieval/prompts/combina_prompt_v2.txt

Você receberá 3 respostas feitas para uma mesma pergunta de um cliente, demarcadas por um número de 1 a 3.

Tudo que você deve fazer é escrever uma única resposta que englobe as informações de todas as 3 outras respostas, de forma objetiva.

Ou seja, entenda o que cada uma das 3 respostas base possuem e escreva uma única resposta que seja coerente.

Pode ser que algumas respostas sejam apenas ruídos, ou seja, não tenham informações relevantes.

Portanto, você deve ser capaz de identificar quais respostas são relevantes e quais não são para construir a resposta final, de maneira concisa e objetiva, sem perder informações.

Evite e remova repetições da fala do cliente, ou seja, ficar repetindo o problema que a pessoa relatou.

Retorne apenas a resposta final e nada além disso.

step4_information_retrieval/prompts/prompt.txt

Assuma o papel de um cliente ligando a um call center para tirar dúvidas sobre algum serviço contratado que apresenta problemas.

Você receberá a transcrição do que um cliente falou durante seu atendimento (por ser automática, podem haver erros de transcrição! Corrija-os quando puder!)

Seu objetivo é reescrever a demanda do cliente, isto é, entender qual foi a motivação do cliente para ter feito a ligação e reescrevê-la de forma objetiva na primeira pessoa, como se você fosse o cliente dizendo o que deseja.

Siga os passos abaixo e verifique os exemplos para realizar isso:

1) Verifique se a transcrição contém uma demanda, isto é, uma dúvida ou reclamação válida, que possa ser esclarecida por um agente. Se não houver, RETORNE apenas a palavra "INVALID" e ignore os próximos passos.

Exemplo válido: "Pode alterar. Da internet. Na fixa. Hã? Deixa eu dar uma olhada aqui, só um minutinho. . A internet não está funcionando desde sexta-feira. Eu já tentei contato com eles, marquei suporte técnico e eles não vêm. Eles concluem o serviço e não vêm. Botam no aplicativo como se fosse concluído o serviço e não aparece."

Exemplo inválido: "Vivo. Vivo. Como? E que isso? Ah, você quer o meu telefone? Ah, você quer o meu telefone? Já, já, já, nós já fizemos tudo, já fizemos tudo, já. O seu colo tem"

2) Reescreva a transcrição se passando pelo cliente, ou seja, na primeira pessoa, sendo objetivo e removendo ruídos (marcas de oralidade, texto prolixo...), mantendo a essência da dúvida do cliente para que possa ser respondida por um atendente. Isto é, resuma a dúvida em uma pergunta que possa ser respondida por um agente.

Exemplo original: "Isso mesmo, o valor de R\$ 59,90, o plano que eu fiz foi por os seis primeiros meses. Eu paguei somente uma fatura de R\$ 59,90. 6. 6 primeiros. Já fazem duas semanas que eu estou esperando e a minha fatura venceu. O meu sinal vai cortar e eles não ajustaram minha fatura para poder pagar"

Exemplo reescrito: "Combinei com a empresa de telefonia que deveria pagar R\$ 59,90 nos primeiros 6 meses, no entanto a fatura está vindo com valor fora do combinado em contrato. O que fazer?"

3) Retorne APENAS a demanda reescrita, nada além disso.

step4_information_retrieval/prompts/promptV2.txt

Assuma o papel de um cliente ligando a um call center para tirar dúvidas sobre algum serviço contratado que apresenta problemas.

Você receberá a transcrição do que um cliente falou durante seu atendimento (por ser automática, podem haver erros de transcrição! Corrija-os quando puder!)

Seu objetivo é reescrever a demanda do cliente, isto é, entender qual foi a motivação do cliente para ter feito a ligação e reescrevê-la de forma objetiva na primeira pessoa, como se você fosse o cliente dizendo o que deseja.

Siga os passos abaixo e verifique os exemplos para realizar isso:

1) Reescreva a transcrição se passando pelo cliente, ou seja, na primeira pessoa, sendo objetivo e removendo ruídos (marcas de oralidade, texto prolixo...), mantendo a essência da dúvida do cliente para que possa ser respondida por um atendente. Isto é, resuma a dúvida em uma pergunta que possa ser respondida por um agente.

Exemplo original: "Isso mesmo, o valor de R\$ 59,90, o plano que eu fiz foi por os seis primeiros meses. Eu paguei somente uma fatura de R\$ 59,90. 6. 6 primeiros. Já fazem duas semanas que eu estou esperando e a minha fatura venceu. O meu sinal vai cortar e eles não ajustaram minha fatura para poder pagar"

Exemplo reescrito: "Combinei com a empresa de telefonia que deveria pagar R\$ 59,90 nos primeiros 6 meses, no entanto a fatura está vindo com valor fora do combinado em contrato. O que fazer?"

2) Retorne APENAS a demanda reescrita, nada além disso.

step4_information_retrieval/prompts/refina_prompt1.txt

Você receberá uma fala de um cliente de call center relatando sua demanda, marcado por <dem> e </dem> .

Seu objetivo é verificar se de fato existe uma demanda ou é apenas ruído, como uma fala nada a ver, e retornar um valor binário (0 ou 1) .

O call center é de uma empresa de telecomunicações e o cliente geralmente liga para reclamar ou tirar dúvidas sobre esse tema.

Caso a demanda seja apenas um ruído (tal como apenas "oi", "Legendas por álamo", "bom dia"), retorne apenas o valor 0.

Se a demanda for válida, retorne apenas o valor 1.

step4_information_retrieval/prompts/refina_prompt2.txt

Você receberá duas frases: uma demanda de um cliente de call center e uma resposta de um atendente, marcados por <dem> e <res> respectivamente.

Sua tarefa consiste em duas etapas:

1) verificar se a resposta do atendente contém alguma informação útil de acordo com o que o cliente pediu;

2) se não tiver informação útil, retornar apenas o valor "Não é útil" e ignorar os passos posteriores, nada além disso (Exemplo de saída: "Não é útil");

3) se tiver informação útil, reescreva a resposta do atendente de forma clara e objetiva, de modo a dar ênfase à informação útil e descartar o que não for útil.

Observação: como a resposta foi transcrita por um algoritmo, pode conter erros de digitação. Então tente corrigir esses erros e relevar alguns problemas que não afetem a compreensão da resposta.

É muito importante para meu projeto que você consiga detectar de forma adequada as informações úteis, então por favor se esforce para fazer um bom trabalho.

NÃO FUNCIONOU DIREITO. EXEMPLOS:

```
/Users/alexecheverria/miniconda3/envs/AutoInstruct4LLMs/bin/python  
/Users/alexecheverria/PycharmProjects/AutoInstruct4LLMs/information_retrieval/step5_refina_candidatos_para_cada_demanda.py
```

Demanda: <dem>Eu gostaria de reclamar sobre o controle da Sky, pois ele está com defeito e não está ligando. Já pedi para eles mandarem um novo controle, mas eles pediram para eu tirar o cartão do receptor, o que não consigo fazer por ter deficiência visual. Além disso, gostaria de cancelar o plano com a Sky, mas sei que há uma multa. O problema é apenas com o controle, não com o aparelho em si. Obrigado.</dem>

Candidato: <res>informação que o senhor deseja? Recomendo o quê? Ah, sim. Você já tentou entrar em contato com --? Você entrou em contato e não consigo anotar protocolo é isso mas é importante que o seu informe protocolo entendeu é importante reforçar aqui no próximo atendimento anotar o seu informe protocolo se eu posso pedir para alguém anotar para o senhor, tá bom? Confirme, por favor, o seu nome completo. Telefone para contato com o DTT. e aí e aí e aí e aí eu estava dizendo que a operadora não não enviou um novo controle o que que está acontecendo e aí controle o que está acontecendo um dia agora o, por favor, senhor. Estou apenas apresentando algumas informações. Estou apenas acrescentando algumas informações no distrito. Ah, no caso, se você não quer fazer, é para o KBQ comentar. Se você quer, eu somente com a isenção da multa. Certo, me informe por favor... Me informe por favor a cidade e o estado onde ocorre o problema? Certo, a descrição está da seguinte maneira. O consumidor reclama que está sem suporte da operadora, o controle da TV está com defeito e está inutilizado. Não foi enviado novo controle. Controle do receptor. Certo. Está com defeito e está inutilizado. Não foi enviado novo controle. Solicitar como providência a troca do controle para a utilização do serviço. Após a finalização, não é possível acrescentar mais nenhuma informação. Se eu quiser acrescentar algo mais, eu posso finalizar. Receptor que grava? Certo. O controle do receptor de gravação está com defeito, tá bom? posso finalizar

então? certo senhor, a operadora é a Sky, correto? o serviço é apenas TV ou tem internet também? certo, a Sky tem um prazo de dias corridos para responder sua solicitação caso seu jogo de resposta não é adequado, o prazo de dias corridos para responder sua solicitação. Caso seu dia de resposta não seja adequado, passam dias corridos para reabertura. O protocolo desta reclamação é o ---. Posso dar de ler algo mais? Está certo, então, senhor. Solicito que a guarda-linha para avaliar a maturidade do MECA não até agradece o tanto atraso. Por favor, avalie com uma nota de a, o profissional que lhe atendeu. Sendo, , muito insatisfeito. , obrigado por participar da nossa pesquisa de satisfação. Anastel agradece a sua atenção.</res>

Resposta GPT: A resposta do atendente não contém informações úteis. Não é útil.

.....
continue

Demanda: <dem>Eu gostaria de reclamar sobre o controle da Sky, pois ele está com defeito e não está ligando. Já pedi para eles mandarem um novo controle, mas eles pediram para eu tirar o cartão do receptor, o que não consigo fazer por ter deficiência visual. Além disso, gostaria de cancelar o plano com a Sky, mas sei que há uma multa. O problema é apenas com o controle, não com o aparelho em si. Obrigado.</dem>

Candidato: <res>reclama da internet ou da TV? Ah, sim, sim. É, está certo. Obrigada. Eu estou acrescentando, até o momento acrescentei que a consumidora reclama que não solicitou serviço algum com a Sky e a senhora desconhece cobrança indevida que senhora recebeu, correto? Pronto, acrescentei também que a

atendente disse que tinha sido cancelado, mas até o momento a consumidora está sendo cobrada novamente por um serviço que não solicitou, correto? Como providência, será desejo cancelar cobrança indevida com máxima urgente, não é isso? Então, agora... Certo. Então, agora está mais um aumento de gentileza. Obrigada, senhora, por aguardar. Peço que a senhora me informe o estado e a cidade onde ocorre o problema. Certo, senhora. Como providência, acrescentei que senhora deseja cancelar cobrança indevida com máxima urgência, correto? Certo. Mas acrescentei sim. Nós informamos que após finalizar, não é possível acrescentar nenhuma informação. A senhora deseja acrescentar algo mais ou posso enviar para a Sky? Tudo bem. Foi enviado e nós informamos que ela vai ter um prazo de dias corridos. Tudo bem. Reabrir o protocolo até dias também. O seu protocolo de hoje da NOMEtel, ele é Ajudem algo mais? Com nada, peço que aguarde em linha para avaliar o atendimento. NOME, até agradeço a ligação. Tenha uma boa tarde. Por favor, avalie com uma nota, de a , o profissional que lhe atendeu. Tendo, , muito insatisfeito. , insatisfeito. , regular. , satisfeito. E , muito satisfeito. Obrigado por participar da nossa pesquisa de satisfação. A Naceu agradece a sua atenção.</res>

Resposta GPT: Não é útil.

.....

continue

Demanda: <dem>Eu gostaria de reclamar sobre o controle da Sky, pois ele está com defeito e não está ligando. Já pedi para eles mandarem um novo controle, mas eles pediram para eu tirar o cartão do receptor, o que não consigo fazer por ter deficiência visual. Além disso, gostaria de cancelar o plano com a Sky, mas sei que há uma multa. O problema é apenas com o controle, não com o aparelho em si. Obrigado.</dem>

Candidato: <res>Só um momento. Obrigado por aguardar. Sua reclamação ficou a seguinte. O consumidor reclama que cancelou o plano. Após , dias, recebeu ligação da operadora com a nova proposta. Aceitou, onde teria um acordo de meses, ativado em novembro. Porém, sempre recebe valores acima do acordado. Reclama, mas o problema é persistente. Pagou e atualmente houve o cancelamento do serviço por pendência de um valor já pago e não tem solução do problema em reclamação. O senhor quer como providência do valor de acordo com o contratado e reativação do serviço de imediato. Se ele deseja acrescentar algo mais, eu posso finalizar. Qual é o estado e cidade onde ocorre o seu problema? Após eu finalizar a reclamação, não posso acrescentar nenhuma informação. Posso finalizar. Só para confirmar, o operador está correto? A Sky tem dias corridos para entrar em contato. Se julgar a resposta inadequada, tem dias para reabrir. Seu protocolo é . Algo mais? Certo, eu vou pedir para a guardainha Para avaliar meu atendimento Mas eu agradeço e tenho uma boa noite Ah, já foi enviado para o operador Certo, eu vou pedir para a guardainha para avaliar o meu atendimento. NOME, eu te agradeço e tenha uma boa noite. Ah, já foi enviado para o operador. Agora é só aguardar o operador entrar em contato. Tá bom? Algo mais, senhor NOME? Vou transferir o seu para avaliação. NOME, eu te agradeço e tenha uma boa noite. Obrigado. Por favor, avalie com uma nota de a, o profissional que lhe atendeu. Sendo, , muito insatisfeito. , insatisfeito. , regular. , satisfeito. E , muito satisfeito. Obrigado por participar da nossa pesquisa de satisfação. A Naceu agradece a sua atenção.</res>

Resposta GPT: Não é útil.

.....

continue

step4_information_retrieval/prompts/refina_prompt2_v2.txt

Você receberá duas frases: uma demanda de um cliente de call center e uma resposta de um atendente, marcados por <dem> e <res> respectivamente.

Sua tarefa consiste em buscar informações úteis na resposta e reescrever a resposta do atendente de forma clara e objetiva, de modo a dar ênfase à informação útil e descartar o que não for útil.

Observação: como a resposta foi transcrita por um algoritmo, pode conter erros de digitação. Então tente corrigir esses erros e relevar alguns problemas que não afetem a compreensão da resposta.

Te darei \$100 de gorjeta se fizer bem feito, ou seja, reescrever de forma clara, direta e objetiva a resposta do atendente!!

step4_information_retrieval/step1_identificador_demanda.py

```
# esse código visa identificar as demandas de forma automática,
utilizando o ChatGPT, a partir das transcrições do
# cliente e reescrevê-las de forma mais clara e objetiva.

from LLMs import ChatGPT
import os
import pandas as pd
import logging
import datetime

from step2_text_preprocessing import main_preprocessing as
preprocessing
from tqdm import tqdm

cwd = os.getcwd().split("AutoInstruct4LLMs")[0] +
"AutoInstruct4LLMs/"

class IdentificadorDemanda:
    """
    Primeiro: carregar as transcrições em CSV
    Segundo: Selecionar apenas as falas do cliente
    Terceiro: Pedir ao chatgpt identificar possíveis demandas nas
falas do cliente
    Quarto: Salvar as demandas identificadas em um CSV
    """
    def __init__(self, transcricoes_path: str = None,
api_key_path: str = "./key.txt",
logs_path: str = f"{cwd}/logs/id_demanda"):
```

```
        self.transcricoes = [trans for trans in
os.listdir(transcricoes_path) if trans.endswith(".csv")]
    self.path_transcricoes = transcricoes_path
    self.api_key_path = api_key_path
    self.logs_path = logs_path
    # instanciando objetos
    self.chatgpt = ChatGPT.SimpleCompletion(api_key_path)
    self.prepro_engine = preprocessing.Preprocessamento()
    self.logger = self.__criar_logger()
    self.demandas = self.__load_demandas()
    self.ja_foram = [dem[0] for dem in self.demandas]
    self.tokens_in = 0
    self.tokens_out = 0

    def identificar_demandas(self, path_to_save=None):
        """
        Essa função carrega as transcrições, executa o
        pré-processamento do texto e pede ao ChatGPT para identificar
        as demandas.
        :return:
        """
        assert path_to_save is not None, "Você deve especificar
qual o path para salvar as demandas"
        for transc in tqdm(self.transcricoes):
            # verificando se a transcrição já foi processada
            if transc in self.ja_foram:
                self.logger.info(f" Arquivo {transc} já foi
processado!")
                continue
            # carregando as transcrições
```

```
df =
pd.read_csv(f"{self.path_transcricoes}/{transc}") # seria
interessante usar parquet
    falas_cliente = self.__falas_cliente(df)
    # realizando o pré-processamento do texto
    falas_cliente =
self.prepro_engine.preprocessar_texto(falas_cliente,
cliente_id=transc)
    falas_cliente = "\n\n<transcription>\n" +
falas_cliente + "\n</transcription>"
    # enviando as falas do cliente para o ChatGPT
    context =
open(f"{cwd}/step4_information_retrieval/prompts/promptV2.txt",
"r").read() # definindo o contexto / Prompt
    context = context + falas_cliente
    fala_reescrita, etc =
self.chatgpt.get_completion(context)
    # resetando as mensagens
    self.chatgpt.reset_messages()
    if etc is not None:
        self.tokens_in += etc.usage.prompt_tokens
        self.tokens_out += etc.usage.completion_tokens
    # salvando as demandas reescritas
    self.demandas.append((transc, falas_cliente,
fala_reescrita))
    self.ja_foram.append(transc)
    # logs
    self.logger.info(f" Arquivo: {transc} - Fala
reescrita: {fala_reescrita}")
    if len(self.demandas) % 10 == 0:
        self.logger.info(f" Salvando demandas...")
```

```
        self.__salvar_demandas(path_to_save)
# salvando as demandas
self.__salvar_demandas(path_to_save)

    def __load_demandas(self, path: str =
f"{cwd}/dados/gold/demandas_identificadas"):
    """
    Função para carregar as demandas identificadas
    :return: None
    """
    try:
        df =
pd.read_csv(f"{path}/demandas_identificadas.csv")
        self.logger.info(f" Demandas carregadas!")
        return [(row[0], row[1], row[2]) for row in
df.values]
    except FileNotFoundError:
        self.logger.info(f" Nenhuma demanda encontrada!")
        return []

    def __salvar_demandas(self, path: str =
f"{cwd}/dados/gold/demandas_identificadas"):
    """
    Função para salvar as demandas identificadas
    :param path: path_audio para salvar as demandas
    :return: None
    """
    # criando a pasta de logs
    if not os.path.exists(f"{cwd}/{path}"):
        os.makedirs(f"{cwd}/{path}")
    # gerando csv
```

```
        df = pd.DataFrame(self.demandas, columns=['arquivo',
'fala_cliente', 'fala_reescrita'])
        df.to_csv(f"{cwd}/{path}/demandas_identificadas.csv",
index=False)
        # salvando os logs
        self.logger.info(f" Finalizado!")
        self.logger.info(f" TOTAL Tokens in: {self.tokens_in} |
Preço: {(self.tokens_in/1000) * 0.0015}")
        self.logger.info(f" TOTAL Tokens out: {self.tokens_out}
| Preço: {(self.tokens_in/1000) * 0.002}")

    def __falas_cliente(self, transcricao: pd.DataFrame) -> str:
        """
        Função para selecionar apenas as falas do cliente
        :param transcricao: dataframe com as transcrições
        :return: str com as falas do cliente
        """
        return transcricao[transcricao['role'] ==
'cliente']['transcription'].str.cat(sep=" ")

    def __criar_logger(self):
        """
        Função para criar o logger
        """
        # criando a pasta de logs
        if not os.path.exists(self.logs_path):
            os.makedirs(self.logs_path)

        # criando o logger
        logger = logging.getLogger(__name__)
        logger.setLevel(logging.INFO)
```

```
        formatter =
logging.Formatter('%(asctime)s:%(levelname)s:%(message)s')
        datet =
datetime.datetime.now().strftime("%Y-%m-%d_%H-%M-%S")
        file_handler =
logging.FileHandler(f"{self.logs_path}/IdentificadorDemanda_{datet}
.log", encoding='utf-8')
        file_handler.setFormatter(formatter)
        logger.addHandler(file_handler)
        return logger

    if __name__ == '__main__':
        idd =
IdentificadorDemanda("../dados/gold/deprecated/anatel_transcricoes"
, api_key_path="../key.txt")
        idd.identificar_demandas()
```

step4_information_retrieval/step2_demandas_to_embedding.py

```
import pandas as pd
from LLMs import embedding_calculator
import os
from tqdm import tqdm
from elasticsearch import Elasticsearch
import urllib3
urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)

cwd = os.getcwd().split("AutoInstruct4LLMs")[0] +
"AutoInstruct4LLMs/"
if not os.path.exists(f"{cwd}/dados/gold/embeddings"):
    os.makedirs(f"{cwd}/dados/gold/embeddings")

def main(path_demandas: str = None, password_elastic=None,
es_index=None):
    """
    Essa função tem por objetivo transformar as demandas
    identificadas em embeddings e indexá-las no elasticsearch.
    """
    assert password_elastic is not None, "Deve disponibilizar a
senha da instância elasticsearch"
    assert path_demandas is not None, "Deve disponibilizar o
diretório das demandas identificadas."
    assert es_index is not None, "Defina o indice elasticsearch"
    es = Elasticsearch("https://localhost:9200",
```

```
ca_certs=f"{cwd}/step3_elastic/http_ca.crt",
        basic_auth=("elastic", password_elastic),
        verify_certs=False)

# carregando as demandas extraídas das transcrições
df = pd.read_csv(f"{cwd}/{path_demandas}/demandas_identificadas.csv")
# criando coluna embeddings no dataframe
df["embedding"] = None
df['embedding'] = df['embedding'].astype('object')
# criando os embeddings para cada demanda
obj = embedding_calculator.Ada002(api_key_path=f"{cwd}/key.txt")
for i, row in tqdm(df.iterrows(), total=df.shape[0]):
    embedding, tokens_in = obj.get_embedding(row["fala_reescrita"])
    doc = {
        "segmento_de_texto": row.fala_reescrita,
        "persona": "cliente",
        "categoria": "demanda",
        "arquivo": row.arquivo,
        "fala_original": row.fala_cliente,
        "embed": embedding
    }

    es.index(index=es_index, document=doc)

# salvando o dataframe
print(f"Tokens usados: {obj.tokens_in} - Preço: U${obj.tokens_in * 0.0001/1000:.3f}")
```

```
if __name__ == '__main__':  
    main()
```

step4_information_retrieval/step3_falas_atendentes_to_embeddings.py

```
import pandas as pd
from LLMs import embedding_calculator
import os
from tqdm import tqdm
from elasticsearch import Elasticsearch
from step2_text_preprocessing.main_preprocessing import
Preprocessamento
import urllib3
urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)

cwd = os.getcwd().split("AutoInstruct4LLMs")[0] +
"AutoInstruct4LLMs/"
if not os.path.exists(f"{cwd}/dados/gold/embeddings"):
    os.makedirs(f"{cwd}/dados/gold/embeddings")

def main(path_transcricoes: str, password_elastic=None,
es_index=None):
    """
    Essa função tem por objetivo transformar as falas dos
    atendentes em embeddings e indexá-las no elasticsearch.
    """
    assert password_elastic is not None, "Deve fornecer a senha
do elasticsearch."
```

```
        assert es_index is not None, "Deve fornecer o indice
elasticsearch."

# instanciando o objeto de preprocessamento
obj_preprocessamento = Preprocessamento()

# instancia o objeto do elasticsearch

es = Elasticsearch("https://localhost:9200",

ca_certs=f"{cwd}/step3_elastic/http_ca.crt",
                basic_auth=("elastic", password_elastic),
                verify_certs=False)

# carregando as transcrições
transcricoes = os.listdir(f"{cwd}/{path_transcricoes}")

obj = embedding_calculator.Ada002(api_key_path=f"{cwd}/key.txt")

for transc in tqdm(transcricoes):
    # carregando as transcricoes do atendente como uma frase
    df = pd.read_csv(f"{cwd}/{path_transcricoes}/{transc}")
    # selecionando as falas do atendente
    df = df[df.role == "atendente"]
    # agrupando as falas do atendente em uma string
    fala_atendente = " ".join(df.transcription)

    # preprocessando o texto

    fala_atendente = obj_preprocessamento.preprocessar_texto(fala_atendente, transc)

    # remover numeros
```

```
fala_atendente =
obj_preprocessamento.remove_numeros(fala_atendente)

# calculando o embedding da fala do atendente
embedding, _ = obj.get_embedding(fala_atendente)
# salvando embedding no dataframe geral
doc = {
    "segmento_de_texto": fala_atendente,
    "persona": "atendente",
    "arquivo": transc,
    "fala_original": "N/A",
    "embed": embedding
}

es.index(index=es_index, document=doc)

# salvando o dataframe
print(f"Tokens usados: {obj.tokens_in} - Preço:
U${obj.tokens_in * 0.0001/1000:.3f}")

if __name__ == '__main__':
    main()
```

step4_information_retrieval/step4_busca_respostas_pelas_demandas.py

```
"""
    Esse código tem por objetivo realizar, para cada demanda, a
    busca semântica no elasticsearch por respostas.
"""

from step3_elastic.step2_search import DocumentsSearch
from time import time
import pandas as pd
import os
from tqdm import tqdm
import urllib3

urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)

cwd = os.getcwd().split("AutoInstruct4LLMs")[0] +
"AutoInstruct4LLMs/"

def main(path_demandas=None, password_elastic=None):
    """
        Essa função tem por objetivo buscar as respostas mais
        semelhantes às demandas feitas pelos clientes.
    """
    assert path_demandas is not None, "Especifique o caminho
    para as demandas"
    assert password_elastic is not None, "Defina a senha do
    elasticsearch"
    # instanciando o objeto de busca semântica:
    anatel_search = DocumentsSearch("anatel_teste_final_index",
    password_elastic)
```

```
# buscando as demandas dos clientes

df = pd.read_csv(f"{cwd}/{path_demandas}/demandas_identificadas.csv")

# instanciando o dataframe final
df_final = pd.DataFrame(columns=["arquivo",
"fala_reescrita_cliente", "respostas_candidatas", "tempo_busca",
"similaridade_max"])

for idx, row in tqdm(df.iterrows(), total=df.shape[0]):
    frase = row.fala_reescrita
    persona = "atendente"

    start = time() # medindo o tempo que leva para a busca
    results = anatel_search.semantic_search(frase, persona)
    tempo = time() - start

    # verificando
    resultado_pesquisa = ""
    similaridade_maxima = -1
    for resultado in results:
        texto_candidato = resultado[0]['segmento_de_texto']
        similaridade = resultado[1]
        if similaridade > similaridade_maxima:
            similaridade_maxima = similaridade

        resultado_pesquisa += texto_candidato + "<sep>"

    # Coletando o nome do arquivo e a fala reescrita do
cliente
```

```
        arquivo = row['arquivo'] # Substitua 'arquivo' pelo
nome da coluna correspondente, se for diferente
        fala_reescrita_cliente = row['fala_reescrita'] #
Presumo que esta seja a coluna correta

# Incluindo os dados no dataframe
df_aux = pd.DataFrame({
    "arquivo": [arquivo],
    "fala_reescrita_cliente": [fala_reescrita_cliente],
    "respostas_candidatas": [resultado_pesquisa],
    # ou a melhor resposta, se você a estiver
armazenando em outra variável
    "tempo_busca": [tempo],
    "similaridade_max": [similaridade_maxima]
})

df_final = pd.concat([df_final, df_aux], axis=0,
ignore_index=True)

# salvando o dataframe

df_final.to_csv(f"{cwd}/{path_demandas}/demandas_e_respostas_candid
atas.csv", index=False)
```

step4_information_retrieval/step5_refina_candidatos_para_cada_demanda.py

```
import pandas as pd
import os

from LLMs.ChatGPT import SimpleCompletion
from tqdm import tqdm

cwd = os.getcwd().split("AutoInstruct4LLMs")[0] +
"AutoInstruct4LLMs/"

# instanciando o objeto do chatgpt

gpt_obj = SimpleCompletion(api_key_path=f"{cwd}/key.txt")

def main(path_demandas=None):
    assert path_demandas is not None, "Passe um caminho da pasta
das demandas"

    path =
f"{cwd}/{path_demandas}/demandas_e_respostas_candidatas.csv"

    # Carrega o arquivo csv com as demandas e as respostas
candidatas

    df = pd.read_csv(path)

    if "respostas_refinadas" in df.columns:
        respostas_refinadas = df[df['respostas_refinadas'] !=
"NOT YET"]['respostas_refinadas'].tolist()
    else:
        respostas_refinadas = []
```

```
# iterando por cada linha do dataframe e extraindo a demanda
e os candidatos

for index, row in tqdm(df.iterrows(), total=df.shape[0]):
    # verificando se o dataframe contém a coluna
    respostas_refinadas

    if "respostas_refinadas" in df.columns:
        # verificando qual o primeiro not yet dessa coluna
        if df["respostas_refinadas"][index] != "NOT YET":
            continue

    demanda = row["fala_reescrita_cliente"]
    if int(index) % 10 == 0 and int(index) != 0:
        print("salvando")
        df["respostas_refinadas"] = respostas_refinadas +
(["NOT YET"] * (df.shape[0] - int(index)))
        # salvando o dataframe
        df.to_csv(path, index=False)

##### NÃO DEU CERTO -> POUCOS DETECTADOS
COMO INVÁLIDOS.

# # step 1: verificar se a demanda é válida ou não.
# # carregando o prompt / contexto
# contexto =
open(f"{cwd}/step4_information_retrieval/prompts/refina_prompt1.txt", "r").read()

# # setando o contexto
# gpt_obj.set_context(contexto)
# # obtendo a resposta do modelo de linguagem
# resposta1 = gpt_obj.get_completion(text=demanda)[0]
# # realizando o append da resposta na lista
validade_demandas
# validade_demandas.append(resposta1)
```

```
# gpt_obj.reset_messages(maintain_context=False) #
resetando as mensagens para o próximo loop
#####

demanda = "<dem>" + demanda + "</dem>"
# step 2: verificar se os candidatos são válidos ou não,
caso a demanda seja válida.
# realizando o split dos candidatos
candidatos =
row['respostas_candidatas'].split("<sep>")[:-1]

# iterando por cada candidato e verificando se contém
informações pertinentes à demanda.
respostas_gpt = []
# iterando apenas pelos 3 primeiros candidatos por causa
de tempo
for candidato in candidatos[:3]:
    candidato = "<res>" + candidato + "</res>"
    # carregando o prompt / contexto
    contexto =
open(f"{cwd}/step4_information_retrieval/prompts/refina_prompt2_v2.
txt", "r").read()
    # setando o contexto
    gpt_obj.set_context(contexto)
    # obtendo a resposta do modelo de linguagem
    resposta2 = gpt_obj.get_completion(text=demanda +
"\n" + candidato)[0]
    # armazenando as respostas
    respostas_gpt.append(resposta2)
    gpt_obj.reset_messages(maintain_context=False) #
resetando as mensagens para o próximo loop
```

```
respostas_refinadas.append("<sep>".join(respostas_gpt))

# criando as colunas no dataframe para armazenar as
validades das demandas e as respostas refinadas
df["respostas_refinadas"] = respostas_refinadas

# salvando o dataframe
df.to_csv(path, index=False)
```

step4_information_retrieval/step6_combina_respostas_refinadas.py

```
import pandas as pd
import os

from LLMs.ChatGPT import SimpleCompletion
from tqdm import tqdm

cwd = os.getcwd().split("AutoInstruct4LLMs")[0] +
"AutoInstruct4LLMs/"

# instanciando o objeto do chatgpt

gpt_obj = SimpleCompletion(api_key_path=f"{cwd}/key.txt")

def main(path_demandas=None):
    assert path_demandas is not None, "Passe um caminho da pasta
das demandas"

    path =
f"{cwd}/{path_demandas}/demandas_e_respostas_candidatas.csv"

    # Carrega o arquivo csv com as demandas e as respostas
candidatas

    df = pd.read_csv(path)

    if "respostas_refinadas" in df.columns:
        respostas_refinadas = df[df['respostas_refinadas'] !=
"NOT YET"]['respostas_refinadas'].tolist()
    else:
        respostas_refinadas = []
```

```
# iterando por cada linha do dataframe e extraindo a demanda
e os candidatos

for index, row in tqdm(df.iterrows(), total=df.shape[0]):
    # verificando se o dataframe contém a coluna
    respostas_refinadas

    if "respostas_refinadas" in df.columns:
        # verificando qual o primeiro not yet dessa coluna
        if df["respostas_refinadas"][index] != "NOT YET":
            continue

    demanda = row["fala_reescrita_cliente"]
    if int(index) % 10 == 0 and int(index) != 0:
        print("salvando")
        df["respostas_refinadas"] = respostas_refinadas +
(["NOT YET"] * (df.shape[0] - int(index)))
        # salvando o dataframe
        df.to_csv(path, index=False)

##### NÃO DEU CERTO -> POCOS DETECTADOS
COMO INVÁLIDOS.

# # step 1: verificar se a demanda é válida ou não.
# # carregando o prompt / contexto
# contexto =
open(f"{cwd}/step4_information_retrieval/prompts/refina_prompt1.txt", "r").read()

# # setando o contexto
# gpt_obj.set_context(contexto)
# # obtendo a resposta do modelo de linguagem
# resposta1 = gpt_obj.get_completion(text=demanda)[0]
# # realizando o append da resposta na lista
validade_demandas
# validade_demandas.append(resposta1)
```

```
# gpt_obj.reset_messages(maintain_context=False) #
resetando as mensagens para o próximo loop
#####

demanda = "<dem>" + demanda + "</dem>"
# step 2: verificar se os candidatos são válidos ou não,
caso a demanda seja válida.
# realizando o split dos candidatos
candidatos =
row['respostas_candidatas'].split("<sep>")[:-1]

# iterando por cada candidato e verificando se contém
informações pertinentes à demanda.
respostas_gpt = []
# iterando apenas pelos 3 primeiros candidatos por causa
de tempo
for candidato in candidatos[:3]:
    candidato = "<res>" + candidato + "</res>"
    # carregando o prompt / contexto
    contexto =
open(f"{cwd}/step4_information_retrieval/prompts/refina_prompt2_v2.
txt", "r").read()
    # setando o contexto
    gpt_obj.set_context(contexto)
    # obtendo a resposta do modelo de linguagem
    resposta2 = gpt_obj.get_completion(text=demanda +
"\n" + candidato)[0]
    # armazenando as respostas
    respostas_gpt.append(resposta2)
    gpt_obj.reset_messages(maintain_context=False) #
resetando as mensagens para o próximo loop
```

```
respostas_refinadas.append("<sep>".join(respostas_gpt))

# criando as colunas no dataframe para armazenar as
validades das demandas e as respostas refinadas
df["respostas_refinadas"] = respostas_refinadas

# salvando o dataframe
df.to_csv(path, index=False)
```

step5_instruct_formatter/step1_instruct_func.py

```
from random import sample
import os

cwd = os.getcwd().split("AutoInstruct4LLMs")[0] +
"AutoInstruct4LLMs/"

def sample_instruction() -> str:
    """
    Essa função seleciona aleatoriamente uma instrução e a
    retorna.
    :return: instrução selecionada aleatoriamente
    """
    instructions = ["Com seus conhecimentos, responda à seguinte
    solicitação de um cliente:",
                    "Elabore uma resposta para a demanda do
    usuário:",
                    "Apresente uma resposta objetiva para a
    seguinte demanda:",
                    "Responda à solicitação do cliente levando
    em conta seus conhecimentos:",
                    "Responda de forma objetiva e direta a
    demanda de um cliente de call center:",
                    "Levando em consideração o contexto de call
    center e telecomunicações, responda a demanda:",
                    "Leia atentamente à pergunta de um cliente e
    a resposta em seguida:",
                    "Utilizando seus conhecimentos sobre call
    center, responda à pergunta a seguir:",
```

```
        "Assuma o papel de um atendente de call
center e responda à demanda de um cliente:",
        "Você é um atendente de call center e um
cliente te fez uma pergunta. Responda-a:"]
    instruction = sample(instructions, 1)[0]
    return instruction

def add_instruction_to_dataframe(df_path: str) -> None:
    import pandas
    """
        Essa função adiciona uma coluna de instruções a um
dataframe.
    :param df_path: caminho do dataframe
    """
    df_path = cwd + df_path
    df = pandas.read_csv(df_path)
    df['instruction'] = ""
    df["instruction"] = df["instruction"].apply(lambda x:
sample_instruction())
    df = df[["arquivo", "instruction", "fala_reescrita_cliente",
"respostas_unificadas"]]
    #adicionando um marcador no nome do arquivo para indicar que
ele já foi processado
    df_path = df_path.split(".csv")[0] + "_instruct.csv"
    df.to_csv(df_path, index=False)
```

main.py

```
from step1_audio_processing.main_audio import main as
main_audio
from step3_elastic.step0_mapping import embedding_mapping
from step4_information_retrieval.step1_identificador_demanda
import IdentificadorDemanda
from step4_information_retrieval.step2_demandas_to_embedding
import main as main_demandas_emb
from
step4_information_retrieval.step3_falas_atendentes_to_embeddings
import main as main_respostas_emb
from
step4_information_retrieval.step4_busca_respostas_pelas_demandas
import main as main_busca_respostas
from
step4_information_retrieval.step5_refina_candidatos_para_cada_deman
da import main as main_refina_candidatos
from
step4_information_retrieval.step6_combina_respostas_refinadas
import main as main_combina_candidatos
from step5_instruct_formatter.step1_instruct_func import
add_instruction_to_dataframe
from elasticsearch import Elasticsearch
from pprint import pprint
import os
import pandas as pd
cwd = os.getcwd().split("AutoInstruct4LLMs")[0] +
"AutoInstruct4LLMs/"
```

```
OPERACAO = "teste" # selecione aqui o nome da operação. Deve
ser o mesmo nome da pasta onde os áudios estão (bronze)

check_path = f"./checkpoints_{OPERACAO}.txt"
if os.path.exists(check_path):
    checkpoint = open(check_path).read()
else:
    checkpoint = "0"
    with open(check_path, "w") as file:
        file.write(checkpoint)

# definindo as variáveis de paths
path_audio = f'./dados/bronze/{OPERACAO}'
path_audio_denoised = f'./dados/silver/{OPERACAO}_denoised'
path_audio_features = f'./dados/silver/{OPERACAO}_features'
path_audio_cut = f'./dados/silver/{OPERACAO}_cut'
path_audio_transcricoes =
f'./dados/gold/{OPERACAO}_transcricoes'
path_demandas = f"./dados/gold/demandas_{OPERACAO}"

anatel_index = f'{OPERACAO}_index'
password_elastic = "Vlihd6gtJorsSwrEjSKL"

# ##### AUDIO STEPS
#####

if checkpoint == "0":

    main_audio(path_audio, path_audio_denoised,
path_audio_features, path_audio_cut, path_audio_transcricoes)

    checkpoint = "1"
```

```
with open(check_path, "w") as file:
    file.write(checkpoint)

# ##### INDEXING STEPS
#####

if checkpoint == "1":
    client = Elasticsearch("https://localhost:9200",
                           ca_certs="./http_ca.crt",
                           basic_auth=("elastic",
password_elastic),
                           verify_certs=False)

    if not client.indices.exists(index=anatel_index):
        client.indices.create(index=anatel_index,
settings=embedding_mapping['settings'],
mappings=embedding_mapping['mappings'])
        pprint(client.indices.get_mapping(index=anatel_index))
        pprint(client.indices.get_settings(index=anatel_index))
    else:
        print("Index already created.")

    checkpoint = "2"
    with open(check_path, "w") as file:
        file.write(checkpoint)

# ##### BUSCA POR DEMANDAS E RESPOSTAS
#####

if checkpoint == "2":
    print("\033[92mIniciando a identificação de
demandas...\033[0m")
```

```
        idd = IdentificadorDemanda(path_audio_transcricoes,  
api_key_path="./key.txt")  
        idd.identificar_demandas(path_to_save=path_demandas)  
        checkpoint = "3"  
        with open(check_path, "w") as file:  
            file.write(checkpoint)  
  
        if checkpoint == "3":  
            # indexando as demandas achadas no elastic search e  
calculando seus embeddings  
            print("\033[92mIniciando a indexação de demandas no  
elasticsearch...\033[0m")  
            main_demandas_emb(path_demandas, password_elastic,  
es_index=anatel_index)  
            checkpoint = "4"  
            with open(check_path, "w") as file:  
                file.write(checkpoint)  
  
        if checkpoint == "4":  
            # indexando as respostas (falas atendentes) no elastic  
search e calculando seus embeddings  
            print("\033[92mIniciando a indexação de respostas dos  
atendentes no elasticsearch...\033[0m")  
            main_respostas_emb(path_transcricoes=path_audio_transcricoes,  
password_elastic=password_elastic,  
                                es_index=anatel_index)  
            checkpoint = "5"  
            with open(check_path, "w") as file:  
                file.write(checkpoint)
```

```
    if checkpoint == "5":
        # buscando respostas para as demandas através da busca
semântica
        print("\033[92mIniciando a busca semântica por respostas
para as demandas...\033[0m")
        main_busca_respostas(path_demandas=path_demandas,
password_elastic=password_elastic)
        checkpoint = "6"
        with open(check_path, "w") as file:
            file.write(checkpoint)

    if checkpoint == "6":
        # refinando as respostas encontradas através da busca
semântica
        print("\033[92mIniciando o refinamento das
respostas...\033[0m")
        main_refina_candidatos(path_demandas=path_demandas)
        checkpoint = "7"
        with open(check_path, "w") as file:
            file.write(checkpoint)

    if checkpoint == "7":
        # combinando as respostas refinadas em uma única resposta,
para ficar o par demnada - resposta
        print("\033[92mIniciando a combinação das respostas
refinadas...\033[0m")
        main_combina_candidatos(n_threads=30,
path_demandas=path_demandas)
        checkpoint = "8"
        with open(check_path, "w") as file:
            file.write(checkpoint)
```

```
if checkpoint == "8":
    # adicionando as instruções ao dataset
    add_instruction_to_dataframe(df_path=path_demandas +
"/demandas_respostas_final.csv")
    checkpoint = "9"
    with open(check_path, "w") as file:
        file.write(checkpoint)

if checkpoint == "9":
    print("\033[92mProcesso finalizado com sucesso!\033[0m")
    df = pd.read_csv(cwd + path_demandas +
"/demandas_respostas_final_instruct.csv")
    amostras = df.sample(2)
    for i, amostra in amostras.iterrows():
        print(f'Exemplos:\nInstrução:
{amostra["instruction"]}\n'
              f'Demanda: {amostra["fala_reescrita_cliente"]}\n'
              f'Resposta: {amostra["respostas_unificadas"]}\n')
```

tools.py

```
import os

def check_path_exists(*args):
    for path in args:
        if not os.path.exists(path):
            os.makedirs(path)
```

requirements.txt

```
boto3==1.34.15
datasets==2.16.1
denoiser==0.1.5
elasticsearch==8.11.1
faster-whisper==0.10.0
ftfy==6.1.3
librosa==0.10.1
llvmlite==0.41.1
numba==0.58.1
numpy==1.26.2
openai==1.6.1
pandas==2.1.4
pyarrow==14.0.2
scikit-learn==1.3.2
scipy==1.11.4
sounddevice==0.4.6
soundfile==0.12.1
tokenizers==0.15.0
torch==2.1.2
torchaudio==2.1.2
tqdm==4.66.1
transformers==4.36.2
```

README.md

Auto Instruct for LLMs

Gerador de datasets instrutivos para treino de LLMs a partir de ligações de call center.

Por Alex Echeverria, Tese de Conclusão de Curso do Bacharelado em Inteligência Artificial.

Como usar esta ferramenta:

Siga os passos abaixo:

- **Instalando os requirements:**
 - Requisito de python: 3.10.0
 - Recomento o uso de conda para criar environments.
 - A partir de um shell / terminal, crie um environment conda com o seguinte comando:
 - `conda create --name AutoInstruct4LLMs python=3.10`
 - Instale as bibliotecas necessárias, usando o comando:
 - `pip install -r requirements.txt`
 - Instale o ffmpeg, com o seguinte comando:
 - `conda install conda-forge::ffmpeg`
- **Preparando os dados:**
 - Coloque todos os arquivos de áudio que possui no diretório `./dados/bronze/{nome_operação}` (crie um, se não houver). (nome operação é o nome da empresa de call center que deseja aplicar a ferramenta)
 - Insira sua chave da OpenAI (token) em um txt de nome `key.txt`, na raiz do diretório (direto em `./AutoInstruct4LLMs`)
- **Configurando o Elasticsearch:**
 - Crie uma instância elasticsearch (recomendo usar docker), determine o nome do índice como sendo `{OPERACAO}_index`.
 - Salve a sua senha do elasticsearch na variável `password_elastic` (linha 36 do código `main.py`) e salve o arquivo `http_ca.crt` também na raiz do repositório.
- **Executando o script principal:**
 - Execute o script principal em `main.py`, substituindo o valor de `OPERACAO` pelo nome da operação que deseja usar.
- **Obtendo o dataset:**
 - Após a finalização da execução, o dataset estará disponível em `./dados/gold/demandas_{OPERACAO}/demandas_respostas_final_instruct.csv`