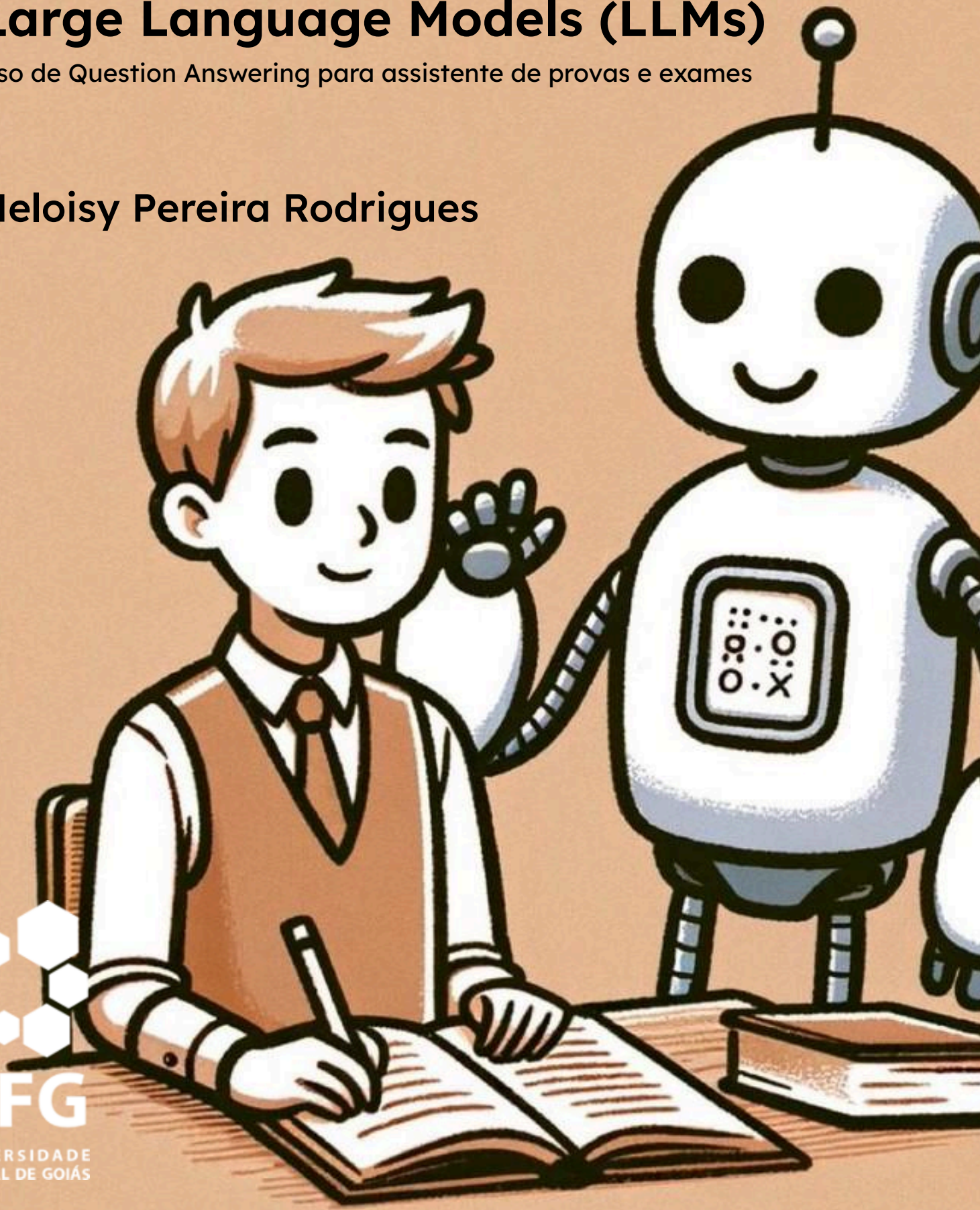


# Large Language Models (LLMs)

Uso de Question Answering para assistente de provas e exames

Heloisy Pereira Rodrigues



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

HELOISY PEREIRA RODRIGUES

## **LARGE LANGUAGE MODELS (LLMS)**

Uso de Question Answering para assistente de provas e exames

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): **HELOISY PEREIRA RODRIGUES**

Título do trabalho:

**LARGE LANGUAGE MODELS (LLMS)**

**Uso de Question Answering para assistente de provas e exames**

### 2. Informações de acesso ao documento (este campo deve ser preenchido pelo orientador) Concorda com a liberação total do documento [ X ] SIM [ ] NÃO<sup>1</sup>

[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 **Heloisy Pereira Rodrigues, Discente**, em 15/02/2024, às 19:14, 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 12/09/2024, às 11:10, 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](https://sei.ufg.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0), informando o código verificador **4383396** e o código CRC **FF42B50F**.

---

Referência: Processo nº 23070.008386/2024-55

SEI nº 4383396

HELOISY PEREIRA RODRIGUES

## **LARGE LANGUAGE MODELS (LLMS)**

Uso de Question Answering para assistente de provas e exames

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.

RODRIGUES, HELOISY PEREIRA  
LARGE LANGUAGE MODELS (LLMS) [manuscrito] : Uso de Question Answering para assistente de provas e exames / HELOISY PEREIRA RODRIGUES. - 2024.  
149 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. question answering. I. FEDERSON, FERNANDO MARQUES, orient.  
II. Título.

CDU 004

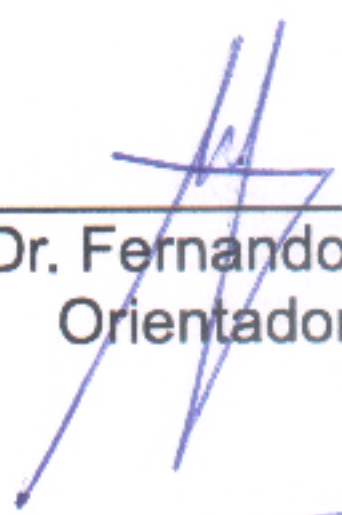
HELOISY PEREIRA RODRIGUES

## **LARGE LANGUAGE MODELS (LLMS)**

Uso de Question Answering para assistente de provas e exames

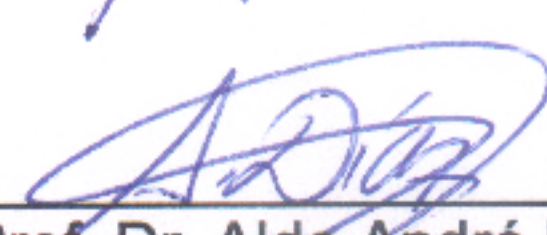
Relatório final de Trabalho de Conclusão de Curso, apresentado à Universidade Federal de Goiás, como parte das exigências para a obtenção do título de Bacharel em Inteligência Artificial.

Data da Aprovação: 08 de fevereiro de 2024.



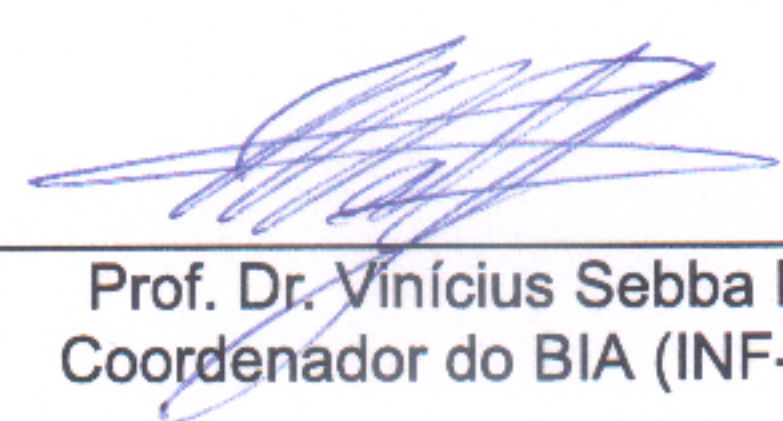
---

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



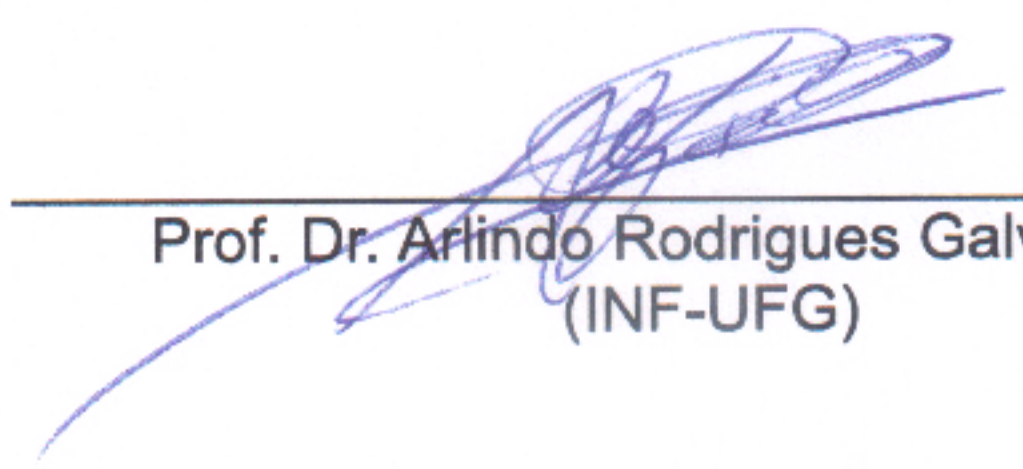
---

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. Arlindo Rodrigues Galvão Filho  
(INF-UFG)

HELOISY PEREIRA RODRIGUES

## **LARGE LANGUAGE MODELS (LLMS)**

Uso de Question Answering para assistente de provas e exames

### **RESUMO**

Este Relatório de Conclusão de Curso tem como objetivo reunir os resultados da minha jornada para me tornar um especialista em **Large Language Models (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, question answering.

### **ABSTRACT**

This Course Completion Report aims to bring together the results of my journey to become an expert in **Large Language Models (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.

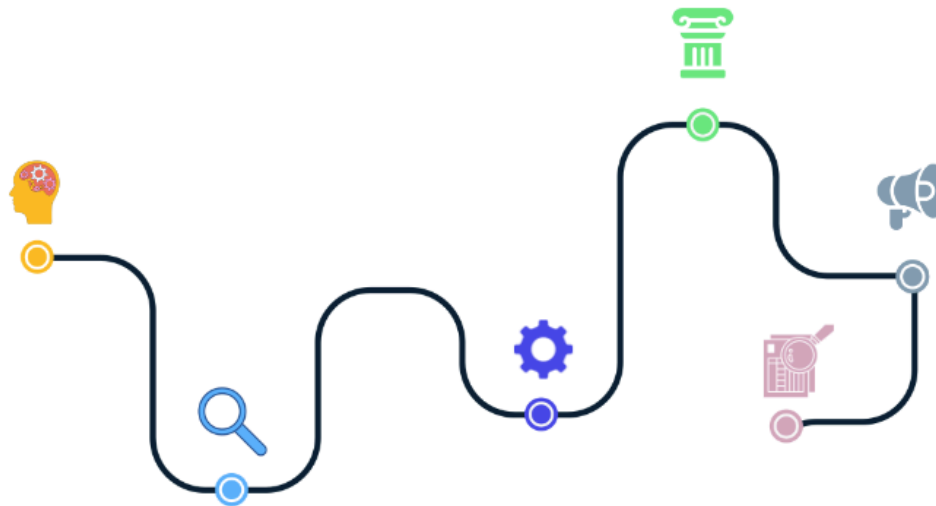
Keywords: artificial intelligence, large language models, question answering.

Goiânia

2024

# Minha Jornada

Heloisy Pereira Rodrigues



Especialista em: Large Language Models (LLMs)

## Semanas 1 e 2

Definição da área de conhecimento para a especialização e revisão de NLP

## Semanas 3 e 4

Estudo de artigos e consolidação de conhecimento sobre a área escolhida (LLMs)

## Semana 5

Estudo sobre ferramentas para o desenvolvimento de LLMs (frameworks, datasets e etc.)

## Semanas 6 e 7

Definição do campo de aplicação, dataset e preparação do ambiente para fine-tuning

## Semanas 8 e 9

Realização de fine-tunings

## Semana 10

Observações sobre os resultados com a realização de fine-tunings

---

## MINHA JORNADA

**Nome:** Heloisy Pereira Rodrigues

**Especialidade:** *Large Language Models* (LLMs)

### Objetivo deste documento

Durante o processo da disciplina Residência em IA<sup>1</sup>, foram gerados diversos resultados na construção da minha especialização. A cada semana, um conjunto de resultados foi formalizado por um Termo de Aceite de Entrega e avaliado por uma banca, considerando o planejado e o realizado para o período. Este documento tem como objetivo descrever esses resultados obtidos, fazendo referência aos Termos de Aceite de Entrega e seus documentos associados.

### Minha Jornada

A minha jornada começou na **Semana 1** com a busca e o entendimento de qual área dentro da Inteligência Artificial eu abordaria para intensificar meus estudos e me tornar uma especialista. Nesta pesquisa, ao analisar áreas (consolidadas e abrangentes) com base na *The 2023 International Conference on Computational Science and Computational Intelligence* (CSCI'23), tive um indicativo de que gostaria de trabalhar na área de **Large Language Models** (LLMs), que está inserida na área de *Natural Language Processing* (NLP). Vale ressaltar que esse indicativo foi inspirado tanto nas aulas da Disciplina de Processamento de Linguagem Natural, ministradas pelo Prof. Dr. Arlindo Galvão, quanto no contato que tive com o tema em um Projeto de Pesquisa e Desenvolvimento ligado ao Centro de Excelência em Inteligência Artificial (CEIA) do qual faço parte. A partir disso, durante a **Semana 2**, revisei conceitos clássicos de NLP, elaborando um *notebook* com uma breve explicação teórica de alguns termos relevantes e códigos demonstrativos para o meu

---

<sup>1</sup> Dez semanas, entre setembro de 2023 e janeiro de 2024.

desenvolvimento posterior na área escolhida. As referências e o código desenvolvido neste *notebook* de revisão estão localizados no **Apêndice 1**.

Nas **Semanas 3 e 4**, o meu foco foi pesquisar artigos relacionados a LLMs e estudá-los, o que me deu a certeza de que eu seguiria nessa área. Um material enriquecedor nessa minha jornada foi um *survey* que encontrei, denominado “*A Survey of Large Language Models*”, de Zhao et al. Nele, os autores falam desde os primeiros modelos de linguagem que eram estatísticos até os atuais baseados na arquitetura *Transformer*, além de técnicas e aplicações em que esses modelos podem ser utilizados. Com isso, realizei algumas marcações e observações nesse artigo, além de considerações de outros materiais de artigos e blogs estudados durante esse período. Isso tudo pode ser conferido mais profundamente no **Apêndice 2**.

Com a consolidação da minha área de especialização e alguns estudos teóricos realizados, durante a **Semana 5**, procurei em blogs e comunidades virtuais o que estava sendo utilizado para o desenvolvimento de LLMs. A primeira parte dessa busca foi por *datasets* (conjunto de dados), pensando na possibilidade de treinar um modelo. Pesquisei por *datasets* que eu poderia utilizar e, em especial, pelos que já haviam sido utilizados por modelos pré-treinados. A segunda parte foi a busca por *frameworks* que me possibilitassem fazer *data scraping*, ou seja, extrair os dados da web, caso eu precisasse captar dados para um novo treino. Os mais utilizados são o BeautifulSoup e o Scrapy. A terceira parte foi a busca por *frameworks* para o pré-processamento dos dados; os mais comentados nos blogs foram o spaCy e o NLTK (*Natural Language Toolkit*). A quarta parte foi a busca por ferramentas e técnicas que me permitissem realizar efetivamente o treino do modelo. A quinta (e última) parte da busca foi por *frameworks* para a avaliação de LLMs, a fim de entender a performance durante o treinamento e em tarefas específicas. Assim, elaborei um documento abordando os prós e contras dessas ferramentas (**Apêndice 3**).

Após todo esse mapeamento (teoria, técnicas e ferramentas), era necessário alguma aplicação para que eu pudesse colocar em prática todo esse conhecimento adquirido durante essas cinco **Semanas**. Comecei a pensar no que eu gostaria de trabalhar como projeto pessoal. Então, lembrei-me dos meus tempos de pré-vestibular. O quanto era difícil encontrar gabaritos para algumas provas que eu estava estudando. Imaginei que isso seria uma dificuldade comum a qualquer estudante para um processo seletivo, tanto de

vestibulares quanto de concursos e afins. Com isso em mente, durante a **Semana 6**, pensei em criar um monitor que forneceria gabaritos para as mais variadas provas. Neste sentido, o meu campo de aplicação dentro de LLMs foi direcionado para *Question and Answering* (Q&A).

No estudo anterior, eu havia mapeado *datasets* para treino, mas eles não eram voltados para essa tarefa específica. Portanto, iniciei um novo trabalho, começando a procurar *datasets* relacionados a processos seletivos. Apesar da busca, não encontrei nada em português que oferecesse um volume de dados suficiente para a realização do treinamento de um modelo. Eu poderia desenvolver *crawlers* para coletar esses dados da internet, mas julguei esse esforço desnecessário. Concluí que com um *dataset* pelo menos semelhante ao que eu desejava, eu poderia realizar um *fine-tuning* e quando tivesse mais tempo para a captação de dados, bastaria seguir os mesmos passos do processo já realizado.

Na pesquisa, encontrei um *dataset* em inglês chamado Cosmos QA, composto por dados que exigem compreensão de leitura baseada em senso comum, formulados como questões de múltipla escolha. Como as questões dos processos seletivos são semelhantes, selecionei esse *dataset* como ponto de partida. No entanto, surgiu um problema: por ser em língua inglesa, era necessário traduzi-lo. A partir disso, na **Semana 7**, criei um algoritmo para realizar a tradução por meio da API da Open AI, utilizando o modelo GPT 3.5-turbo, um modelo de grande porte. Apesar de alguns problemas durante a tradução, implementei um detector de falhas na tradução, geralmente decorrentes de erros na API, e refiz a tradução eliminando estas falhas.

Com a conclusão da tradução do *dataset* Cosmos QA para o português, disponibilizei o *dataset* traduzido no Hugging Face com o nome Cosmos QA Português, para que mais pessoas pudessem utilizá-lo. Agora, eu tinha o *dataset* e continuava buscando *frameworks* e técnicas, mas faltava decidir qual modelo de grande porte eu utilizaria para o *fine-tuning* em Q&A.

Inicialmente, examinei modelos *open source* e busquei informações sobre seus desempenhos. Fiquei em dúvida entre o Llama2 e o Zephyr (que é um *fine-tuning* do modelo Mistral), pois apresentavam métricas de desempenho semelhantes. No entanto, como o Zephyr obteve melhores resultados que o Llama2 em alguns *benchmarks*, optei por utilizá-lo.

Essas considerações, juntamente com o processo e as dificuldades na tradução, foram documentadas no **Apêndice 4**.

Nos estudos, principalmente em blogs, obtive um *pipeline* para me basear e conseguir realizar o *fine-tuning* do Zephyr 7B beta no Cosmos QA Português. Como já tinha preparado tudo, nas **Semanas 8 e 9**, foi o momento de fazer experimentos. O meu critério avaliativo foi a acurácia nos dados de validação do *dataset*, pois era o que eu tinha como gabarito. Assim, realizei o treinamento do modelo, utilizando algumas técnicas importantes como Quantização, LoRA e o uso da biblioteca PEFT (*Parameter-Efficient Fine-Tuning*). Realizei um teste inicial com o modelo Zephyr quantizado (Zephyr 7B alpha GPTQ) no Google Colab Pro. A acurácia do modelo no *dataset* de validação foi de 72,4%. Um fato interessante é que intensifiquei meus estudos no Zephyr durante esse treinamento e entendi que o modelo chamado de “beta” havia sido treinado por mais épocas. Dessa forma, parti para o primeiro *fine-tuning* no Zephyr 7B beta, utilizando uma RTX 3090, e obtive 79,73% de acurácia. O desempenho foi parecido, mas pequenas porcentagens decidem se você é aprovado ou não no processo seletivo, então considerei algo importante. Percebi que os testes de *fine-tuning* deveriam se voltar para o modelo original, realizados sempre na RTX 3090, dada a necessidade de memória. Portanto, realizei mais dois testes, alterando alguns hiperparâmetros, obtendo 81,03% de acurácia nos dois testes. Os resultados iguais me passaram uma certa desconfiança, mas verifiquei que os dois modelos não acertaram as mesmas questões. Os códigos, de treinamento e avaliação, e mais algumas considerações estão detalhados no **Apêndice 5**.

Ainda não havia feito a inferência dos dados de validação no modelo original, sem *fine-tuning*. Portanto, não era possível comprovar que o *fine-tuning* foi benéfico para o meu propósito de criar um monitor. Na **Semana 10**, desenvolvi um código para realizar a inferência nos dados de validação do Cosmos QA em Português, e o desempenho ficou aquém do que eu esperava, apresentando uma acurácia de 58,27%. Além disso, o modelo sem o *fine-tuning* não apresentava um padrão nas respostas, impossibilitando a criação de um algoritmo para capturar automaticamente as alternativas consideradas corretas. Tive, assim, que anotar todas as inferências antes de verificar a taxa de acertos. Isso demonstra que, ao utilizar as técnicas e recursos disponíveis, os modelos de linguagem podem apresentar um desempenho significativamente melhor quando ajustados para a tarefa e o

caso desejado. Detalhes e observações adicionais podem ser encontrados no **Apêndice 6** deste documento.

Concluídas essas **Semanas**, refleti o quanto o processo foi benéfico para mim, me tornei uma especialista em LLMs. Não somente isso, passei a desenvolver um rigor mais científico e a entender o que a ciência proporciona: seguindo os mesmos processos, as mesmas etapas, tudo que foi observado deve ser exatamente o mesmo. Apesar das inseguranças, de estar iniciando um novo modo de concluir o curso diferente de todo o padrão e de começar a se portar como uma cientista (deixando até mesmo visão empreendedora se distanciar), vejo que eu consegui, eu realmente pesquisei e estudei muito para me especializar em algo escolhido por mim. A Residência em Inteligência Artificial contribuiu para o meu crescimento pessoal e profissional, saio com uma confiança maior, sabendo que consigo realizar projetos (que antes não acreditaria na minha capacidade) e almejando um futuro ainda mais promissor.

## APÊNDICE 1

## Termo de Aceite de Entrega 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]:

Heloisy Pereira Rodrigues

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

A abordagem principal desta semana foi a definição e busca de uma visão do objetivo: ser especialista em Large Language Models. Para isso, o primeiro passo foi classificar quanto as áreas do [CSCI 2023](#), sendo as mais adequadas: Natural language processing, Knowledge representation, Intelligent information systems, Neural networks and applications, Unsupervised and Supervised Learning, Aspects of natural language processing. Além disso, uma pesquisa sobre os artigos escritos envolvendo a temática da especialidade nos CSCI's também foi feita, que resultou num documento de artigos de 2017 a 2022, que estão disponíveis no [link](#).

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

Para a próxima semana, o objetivo é fazer uma pesquisa de artigos fora do CSCI, além de materiais do tipo blogs (Medium), de modo a pensar, brevemente, na história - passos de NLP - até chegar aos LLM. Após essa pesquisa, o foco é condensar as informações mais pertinentes de técnicas/ferramentas que culminaram na existência dos LLM em um documento, para ficar mais fácil o acesso.

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

Como a discussão dos LLM é recente, obtive apenas 1 resultado na pesquisa dos CSCI, então tive que estender a arquitetura Transformer na pesquisa.

## ACEITE DA ENTREGA:

**CEDRIC LUIZ DE CARVALHO:** Go! ▾

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

## Pesquisa de artigos CSCI - Semana 1

[O documento referente a seguir pode ser acessado em:

<https://docs.google.com/document/d/1BL0hvcaZFPmMv0GZp5p9zDsA30H2ItXgi-VDmk4E9ll/edit?usp=sharing>]


Artigos CSCI:

- De LLM:

[Designing of Prompts for Hate Speech Recognition with In-Context Learning](#)

Lawrence Han; Hao Tang

Publication Year: 2022 , Page(s): 319 - 320

✓ Abstract [HTML](#)  

- De Transformer:

---

### Reconstructing Word Representations from Pre-trained Subword Embeddings

Paula Lauren

Publication Year: 2022 , Page(s): 35 - 40

▼ Abstract [HTML](#)  

---

### A Comparative Analysis of State-of-the-Art Time Series Forecasting Algorithms

Cathal Murray; Priyanka Chaurasia; Lynsey Hollywood; Damien Coyle

Publication Year: 2022 , Page(s): 89 - 95

▼ Abstract [HTML](#)  

---

### Preliminary Results of Applying Transformers to Geoscience and Earth Science Data

J. Delgado; U. Ebrezo; Y. Kumar; J. J. Li; P. Morreale

Publication Year: 2022 , Page(s): 284 - 288

▼ Abstract [HTML](#)  

---

### Challenges of Self-Supervised Learning for Unified, Multi-Modal, Multi-Task Transformer Models

Graham Annett; Tim Andersen; Robert Annett

Publication Year: 2022 , Page(s): 293 - 297

▼ Abstract [HTML](#)  

---

### Leveraging K-hop based Graph for the Staffing Recommender System with Parametric Geolocation

Yan Wang; Amar Henni; Azade Fotouhi; Leopold Mvondo Ze

Publication Year: 2022 , Page(s): 664 - 669

▼ Abstract [HTML](#)  

---

### Comparative Analysis of LSTM and CNN for Efficient Malware Detection

Frances Osamor; Briana Wellman

Publication Year: 2022 , Page(s): 965 - 970

▼ Abstract [HTML](#)  

---

### Full/Regular Research Paper submission to (CSCI-RTCW): Multi Class Classification of Online Radicalization Using Transformer Models

Chesta Sofat; Shabeg Singh Gill; Divya Bansal

Publication Year: 2022 , Page(s): 1034 - 1038

▼ Abstract [HTML](#)  

---

### Foreign Object Debris Detection for Airport Pavement Images Based on Self-Supervised Localization and Vision Transformer

### Abstractive Text Summarization via Stacked LSTM

Ireddy Siddhartha; Huixin Zhan; Victor S. Sheng

Publication Year: 2021 , Page(s): 437 - 442

Cited by: Papers (1)

▼ Abstract HTML  

### Zero-Shot Cross-Lingual Transfer in Legal Domain Using Transformer Models

Zein Shaheen; Gerhard Wohlgemann; Dmitry Mouromtsev

Publication Year: 2021 , Page(s): 450 - 456


Cited by: Papers (1)

▼ Abstract HTML  

#### Unified End-to-End Sentence Denoising

Zhantong Liang; Abou Youssef

Publication Year: 2020 , Page(s): 479 - 484

▼ Abstract HTML  

#### Machine Understandable Contracts with Deep Learning

Rares Dolga; Philip Treleaven; Mendes Thame Denny

Publication Year: 2020 , Page(s): 551 - 557

Cited by: Papers (2)

### Solving Arithmetic Word Problems Automatically Using Transformer and Unambiguous Representations

Kaden Griffith; Jugal Kalita

Publication Year: 2019 , Page(s): 526 - 532

Cited by: Papers (6)

▼ Abstract HTML  

## Termo de Aceite de Entrega 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:** 26 de out. de 2023

**Participantes da Entrega** [matriculados em Residência em IA]:

Heloisy Pereira Rodrigues

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

Nesta semana o foco foi estudar e revisar conceitos de NLP que antecedem os LLM's: as representações de palavras e o início de modelos de linguagem. Para que os conceitos sejam facilmente retomados, criei um notebook com essa revisão e exemplos no link: [Revisão NLP](#).

Além disso, um grupo de estudos foi montado e criado um repositório comum em que depositamos contribuições: [Repositório NLP LLM](#). Com a contribuição de um colega (Alex), encontrei um Survey e um Overview de LLM (se encontram no repositório), que servirão de base para a próxima semana.

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

Leitura dos [materiais catalogados](#) a seguir e entendimento dos artigos com base nas “partes” de uma pesquisa:

- “A Survey of Large Language Models”
- “A Comprehensive Overview of Large Language Models”

Enquanto faço essa leitura, o foco também é contribuir na criação do vocabulário específico: [Vocabulário NLP LLM](#).

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

## ACEITE DA ENTREGA:

**CEDRIC LUIZ DE CARVALHO:** Go! ▾

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

## Notebook de Revisão - Semana 2

[O documento referente a seguir pode ser acessado em :

<https://colab.research.google.com/drive/1ubQ7zdzj-9Ccn-gZ95IWC7izXV6ZxdAyN?usp=sharing> ]

Breve revisão sobre a representação de palavras, com um pouco de modelo de língua.

# Representações Clássicas

## One Hot Encoder

Dada as sentenças:

> **The cat sat on the mat.**

> **The dog chased the cat.**

> **The mat was soft and fluffy.**

Com o One Hot, cada palavra nessas frases tem que ser representada como um único vetor.

Exemplo:

```
import numpy as np
```

```
# Defina as frases
```

```
frases = [
```

```
    'O Cedric é o professor.',
```

```
'A Luana é a mestranda.',  
'Há banca na residência.'  
]  
  
# Crie um conjunto de vocabulário  
vocabulario = set()  
for frase in frases:  
    palavras = frase.lower().split()  
    for palavra in palavras:  
        vocabulario.add(palavra)  
  
# Crie um dicionário para mapear palavras em números inteiros  
palavra_para_inteiro = {palavra: i for i, palavra in enumerate(vocabulario)}  
  
# Crie um vetor binário para cada palavra em cada frase  
vetores = []  
for frase in frases:  
    palavras = frase.lower().split()  
    vetores_frase = []  
    for palavra in palavras:  
        vetor_binario = np.zeros(len(vocabulario))  
        vetor_binario[palavra_para_inteiro[palavra]] = 1  
        vetores_frase.append(vetor_binario)  
    vetores.append(vetores_frase)  
  
# Imprima os vetores one-hot codificados para cada palavra em cada frase  
for i in range(len(frases)):  
    print(f"\n Frase {i + 1}:")  
    for j in range(len(vetores[i])):  
        print(f"{frases[i].split()[j]}: {vetores[i][j]}")
```

Logo percebemos que gera matrizes esparsas. Vale ressaltar também que não há relação semântica entre as palavras.

## ## Bag of Words

A ideia principal do BoW é criar um "saco" contendo todas as palavras únicas de um conjunto de documentos (um "vocabulário") e, em seguida, representar cada documento como um vetor onde cada elemento representa a contagem de uma palavra do vocabulário.

Exemplo:

```
from sklearn.feature_extraction.text import CountVectorizer

# Exemplo de documentos
documentos = [
    "Este é um exemplo de bag of words.",
    "Neste exemplo, estamos criando uma representação simples.",
    "Cada documento é transformado em um vetor de contagem de palavras."
]

# Inicialize o CountVectorizer
vectorizer = CountVectorizer()

# Ajuste o vetorizador aos documentos e transforme os documentos em vetores BoW
vetores_bow = vectorizer.fit_transform(documentos)

# Obtenha o vocabulário
```

```
vocabulario = vectorizer.get_feature_names_out()
```

```
# Exiba os vetores BoW e o vocabulário
```

```
print("Vetores BoW:")
```

```
print(vetores_bow.toarray())
```

```
print("\nVocabulário:")
```

```
print(vocabulario)
```

O BoW enfrenta os mesmos problemas do One Hot Encoder.

```
## Term Frequency-Inverse Document Frequency (TF-IDF)
```

Avalia a importância de uma palavra em um documento em relação a um conjunto de documentos. Para isso, leva em consideração duas métricas, são elas:

1. TF (Term Frequency - Frequência do Termo): Mede a frequência com que um termo específico ocorre em um documento. Quanto mais vezes uma palavra aparece em um documento, maior será o seu valor TF para essa palavra nesse documento.
2. IDF (Inverse Document Frequency - Frequência Inversa nos Documentos): Mede a importância relativa de um termo em relação a um conjunto de documentos. Palavras que aparecem em muitos documentos têm um valor IDF menor, enquanto palavras raras têm um valor IDF maior.

Ao multiplicarmos essas duas métricas possuímos o produto para cada termo em um documento. Assim, palavras que ocorrem frequentemente em um documento, mas raramente em outros, terão uma alta pontuação TF-IDF e, portanto, são consideradas importantes para a interpretação desse documento.

Exemplo:

```
from sklearn.feature_extraction.text import TfidfVectorizer

# Exemplo de documentos
documentos = [
    "Este é um exemplo de TF-IDF.",
    "O TF-IDF é útil na análise de texto.",
    "TF-IDF é uma técnica de processamento de linguagem natural.",
]

# Inicialize o TfidfVectorizer
vectorizer = TfidfVectorizer()

# Ajuste o vetorizador aos documentos e transforme os documentos em vetores TF-IDF
vetores_tfidf = vectorizer.fit_transform(documentos)

# Obtenha o vocabulário
vocabulario = vectorizer.get_feature_names_out()

# Exiba os vetores TF-IDF e o vocabulário
print("Vetores TF-IDF:")
print(vetores_tfidf.toarray())
print("\nVocabulário:")
print(vocabulario)
```

Continua gerando um vetor grande dado que os vetores são gerados a partir do conjunto de dados e não tendo relação semântica, apesar de que é melhor que o BoW visto que não gera apenas valores binários e sim números com mais informações sobre os dados.

## # Representações densas - Word Embeddings

São vetores espaciais que mantêm a semelhança semântica entre o texto. Cada palavra é representada por um vetor.

Tenta prever o contexto através da palavra central ou vice-versa.

## ## Word2Vec

Word2Vec é uma rede neural que aprende a prever a probabilidade de uma palavra dada o seu contexto (CBOW) ou o contexto dado uma palavra (skip-gram).

Word2Vec consiste em duas arquiteturas: Continuous Bag of Words (CBOW) e Skip-Gram.

[image.png]

1. CBOW (Continuous Bag of Words): Nesta arquitetura, o modelo tenta prever uma palavra alvo com base em um contexto de palavras vizinhas. Por exemplo, dado o contexto "Ele gosta de \_\_\_\_", o modelo tentaria prever a palavra alvo, como "programar", com base nas palavras ao lado.
2. Skip-Gram: Nesta arquitetura, o modelo tenta prever o contexto em torno de uma palavra alvo. Por exemplo, dada a palavra alvo "programar", o modelo tentaria prever as palavras circundantes, como "Ele gosta de \_\_\_\_".

```
from gensim.models import Word2Vec  
from nltk.tokenize import word_tokenize
```

```
import nltk
nltk.download('punkt')

# Exemplo de texto
texto = "O gato roeu a roupa do rei de Roma. A rainha ficou brava e brigou com o rato."

# Tokenize o texto em palavras
palavras = word_tokenize(texto)

# Crie um modelo Word2Vec
modelo = Word2Vec([palavras], vector_size=100, window=5, min_count=1, sg=0)

# Treine o modelo
modelo.train([palavras], total_examples=1, epochs=10)

# Obtenha o vetor de uma palavra
vetor = modelo.wv['rainha']

# Encontre palavras similares
palavras_similares = modelo.wv.most_similar('rainha')

print("Vetor da palavra 'rainha':", vetor)
print("Palavras similares a 'rainha':", palavras_similares)

import matplotlib.pyplot as plt
from sklearn.manifold import TSNE
import numpy as np

# Obtenha os vetores de todas as palavras
```

```
vetores_palavras = np.array([modelo.wv[palavra] for palavra in modelo.wv.index_to_key])

# Reduza a dimensionalidade usando t-SNE com perplexidade definida
tsne = TSNE(n_components=2, perplexity=10) # Ajuste a perplexidade conforme necessário
vetores_tsne = tsne.fit_transform(vetores_palavras)

# Extraia as palavras correspondentes aos vetores
palavras = modelo.wv.index_to_key

# Plote as palavras no gráfico
plt.figure(figsize=(10, 6))
for i in range(len(palavras)):
    x, y = vetores_tsne[i]
    plt.scatter(x, y)
    plt.text(x, y, palavras[i])

plt.title("Visualização de Palavras em Espaço Vetorial 2D (t-SNE)")
plt.show()
```

Treinando mais vezes que o código anterior, para observar a importância do treino:

```
from gensim.models import Word2Vec
from nltk.tokenize import word_tokenize
import nltk
nltk.download('punkt')

# Exemplo de texto
texto = "O gato roeu a roupa do rei de Roma. A rainha ficou brava e brigou com o rato."

# Tokenize o texto em palavras
```

```
palavras = word_tokenize(texto)

# Crie um modelo Word2Vec
modelo = Word2Vec([palavras], vector_size=100, window=5, min_count=1, sg=0)

# Treine o modelo
modelo.train([palavras], total_examples=1, epochs=200)

# Obtenha o vetor de uma palavra
vetor = modelo.wv['rainha']

# Encontre palavras similares
palavras_similares = modelo.wv.most_similar('rainha')

print("Vetor da palavra 'rainha':", vetor)
print("Palavras similares a 'rainha':", palavras_similares)

import matplotlib.pyplot as plt
from sklearn.manifold import TSNE
import numpy as np

# Obtenha os vetores de todas as palavras
vetores_palavras = np.array([modelo.wv[palavra] for palavra in modelo.wv.index_to_key])

# Reduza a dimensionalidade usando t-SNE com perplexidade definida
tsne = TSNE(n_components=2, perplexity=10) # Ajuste a perplexidade conforme necessário
vetores_tsne = tsne.fit_transform(vetores_palavras)

# Extraia as palavras correspondentes aos vetores
```

```
palavras = modelo.wv.index_to_key
```

```
# Plote as palavras no gráfico
```

```
plt.figure(figsize=(10, 6))
```

```
for i in range(len(palavras)):
```

```
    x, y = vetores_tsne[i]
```

```
    plt.scatter(x, y)
```

```
    plt.text(x, y, palavras[i])
```

```
plt.title("Visualização de Palavras em Espaço Vetorial 2D (t-SNE)")
```

```
plt.show()
```

Limitações:

1. Interpretabilidade:

Redes neurais não são explicativas, contém fatores estocástico

2. Não Leva em Conta a Morfologia: Palavras (por exemplo, "correr" e "corria") como palavras diferentes, não considerando sua similaridade morfológica.

3. Perda de Ordem das Palavras (Vetor estático): Sentenças com as mesmas palavras, mas em ordens diferentes, são representadas da mesma forma.

## GloVe

O Word2Vec conseguia capturar o contexto local, mas não global. O contexto global é captado usando múltiplas frases no corpus. Assim, o GloVe é uma técnica que visa criar representações vetoriais contínuas para palavras, de modo a capturar informações semânticas e relacionamentos entre palavras. Ele se concentra na co-ocorrência de palavras em grandes conjuntos de texto para criar essas representações.

Então é treinado com base na contagem de co-ocorrência de palavras em um grande corpus de texto. O que significa que, durante o treinamento, o modelo considera quais palavras ocorrem juntas em um contexto específico.

Exemplo:

> Não consegui baixar para rodar neste notebook, pois estava dando erro de conexão, creio que no servidor da Stanford, mas segue o que serviu de estudo: [GloVe on Colab](<https://colab.research.google.com/drive/1Alfud-zxMIUKp2L7KfZvvoVtsA6GkRuS?usp=sharing>)

```
urllib.request.urlretrieve('https://nlp.stanford.edu/data/glove.6B.zip','glove.6B.zip')
```

## ## Avaliação Word Embeddings

1. Avaliação Intrínseca: A avaliação intrínseca foca na qualidade dos embeddings de palavras em si

1. Avaliação em alguma tarefa intermediária
2. Rápido de calcular
3. Não é nítido a correlação com a tarefa final

2. Avaliação Extrínseca: A avaliação extrínseca, por outro lado, está preocupada com a utilidade dos embeddings de palavras em tarefas do mundo real.

1. Avaliação em uma tarefa final
2. Custoso para calcular

### 3. Não é claro o impacto dos sub sistemas

Ambos os tipos de avaliação são complementares e desempenham um papel importante na escolha e ajuste de embeddings de palavras.

#### # Representações Contextuais

##### ## Embeddings from Language Models (ELMo)

Em contraste com os word embeddings tradicionais, que geram uma única representação fixa para cada palavra, o ELMo gera representações de palavras que levam em consideração o contexto em que as palavras ocorrem em uma frase ou documento. Assim, podem gerar diferentes embeddings para a mesma palavra usada em um contexto diferente em frases diferentes.

O ELMo é basicamente 2 BI-LSTM empilhadas e captura nuances semânticas e sintáticas, bem como as relações entre palavras em contextos específicos.

```
# import necessary libraries
import tensorflow_hub as hub
import tensorflow.compat.v1 as tf
tf.disable_eager_execution()

# Load pre trained ELMo model
elmo = hub.Module("https://tfhub.dev/google/elmo/3", trainable=True)

embeddings = elmo(
    [
        "A manga da camisa está dobrada",
```

```
"Ontem comi uma manga deliciosa"
],
signature="default",
as_dict=True)["elmo"]

init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init)

# Imprima os embeddings para a palavra "manga" nas duas frases
print('Embeddings para a palavra "manga" na primeira frase')
print(sess.run(embeddings[0][1])) # Índice 1 representa a palavra "manga" na primeira frase

print('Embeddings para a palavra "manga" na segunda frase')
print(sess.run(embeddings[1][4])) # Índice 4 representa a palavra "manga" na segunda frase

# Referências

[1] Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean, Efficient Estimation of Word
Representations in Vector Space (2013)

[2] Quoc V. Le, Tomas Mikolov, Distributed Representations of Sentences and Documents
(2014)

[3] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global
Vectors for Word Representation.

[4] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton
Lee, Luke Zettlemoyer, Deep contextualized word representations
```

## APÊNDICE 2

## Termo de Aceite de Entrega 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]:

Heloisy Pereira Rodrigues

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

Para esta entrega, me dediquei na leitura do paper “A Survey of Large Language Models”. No estudo, marquei no pdf coisas que achei importante, com diferentes cores, sendo elas:

- amarelo = importante saber
- vermelho = necessário revisar
- azul = possibilidades de investigação de fatores importantes dos LLM
- verde = ferramentas que posso utilizar

Segue o link do pdf com as marcações: [A Survey of Large Language Models](#)

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

Para a seguinte semana, o planejado é revisar os pontos do pdf marcados em vermelho, decidir uma área de atuação dentro dos LLM e montar um cronograma dos próximos passos para a área desejada.

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

Nas páginas 1 e 2, usei o verde para diferenciar, mas a lógica de usar as cores para “temas” definidos só veio depois.

## ACEITE DA ENTREGA:

**CEDRIC LUIZ DE CARVALHO:** Go! ▾

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

## PDF de “A Survey of Large Language Models” com marcações - Semana 3

[O documento referente a seguir pode ser acessado em:

<https://drive.google.com/file/d/1gAsDrcAAfq9meJYLXm3dyfHFAITBzaJ/view?usp=sharing>]



# A Survey of Large Language Models

Wayne Xin Zhao, Kun Zhou\*, Junyi Li\*, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jian-Yun Nie and Ji-Rong Wen

**Abstract**—Ever since the Turing Test was proposed in the 1950s, humans have explored the mastering of language intelligence by machine. Language is essentially a complex, intricate system of human expressions governed by grammatical rules. It poses a significant challenge to develop capable artificial intelligence (AI) algorithms for comprehending and grasping a language. As a major approach, *language modeling* has been widely studied for language understanding and generation in the past two decades, evolving from statistical language models to neural language models. Recently, pre-trained language models (PLMs) have been proposed by pre-training Transformer models over large-scale corpora, showing strong capabilities in solving various natural language processing (NLP) tasks. Since the researchers have found that model scaling can lead to an improved model capacity, they further investigate the scaling effect by increasing the parameter scale to an even larger size. Interestingly, when the parameter scale exceeds a certain level, these enlarged language models not only achieve a significant performance improvement, but also exhibit some special abilities (e.g., in-context learning) that are not present in small-scale language models (e.g., BERT). To discriminate the language models in different parameter scales, the research community has coined the term *large language models (LLM)* for the PLMs of significant size (e.g., containing tens or hundreds of billions of parameters). Recently, the research on LLMs has been largely advanced by both academia and industry, and a remarkable progress is the launch of ChatGPT (a powerful AI chatbot developed based on LLMs), which has attracted widespread attention from society. The technical evolution of LLMs has been making an important impact on the entire AI community, which would revolutionize the way how we develop and use AI algorithms. Considering this rapid technical progress, in this survey, we review the recent advances of LLMs by introducing the background, key findings, and mainstream techniques. In particular, we focus on four major aspects of LLMs, namely pre-training, adaptation tuning, utilization, and capacity evaluation. Furthermore, we also summarize the available resources for developing LLMs and discuss the remaining issues for future directions. This survey provides an up-to-date review of the literature on LLMs, which can be a useful resource for both researchers and engineers.

**Index Terms**—Large Language Models; Emergent Abilities; Adaptation Tuning; Utilization; Alignment; Capacity Evaluation

arXiv:2303.18223v12 [cs.CL] 11 Sep 2023

## 1 INTRODUCTION

*“The limits of my language mean the limits of my world.”*  
—Ludwig Wittgenstein

LANGUAGE is a prominent ability in human beings to express and communicate, which develops in early childhood and evolves over a lifetime [1, 2]. Machines, however, cannot naturally grasp the abilities of understanding and communicating in the form of human language, unless equipped with powerful artificial intelligence (AI) algorithms. It has been a longstanding research challenge to achieve this goal, to enable machines to read, write, and communicate like humans [3].

Technically, *language modeling (LM)* is one of the major approaches to advancing language intelligence of machines. In general, LM aims to model the generative likelihood of word sequences, so as to predict the probabilities of

future (or missing) tokens. The research of LM has received extensive attention in the literature, which can be divided into four major development stages:

- *Statistical language models (SLM)*. SLMs [4–7] are developed based on *statistical learning* methods that rose in the 1990s. The basic idea is to build the word prediction model based on the Markov assumption, e.g., predicting the next word based on the most recent context. The SLMs with a fixed context length  $n$  are also called  $n$ -gram language models, e.g., bigram and trigram language models. SLMs have been widely applied to enhance task performance in information retrieval (IR) [8, 9] and natural language processing (NLP) [10–12]. However, they often suffer from the curse of dimensionality: it is difficult to accurately estimate high-order language models since an exponential number of transition probabilities need to be estimated. Thus, specially designed smoothing strategies such as back-off estimation [13] and Good–Turing estimation [14] have been introduced to alleviate the data sparsity problem.

- *Neural language models (NLM)*. NLMs [15–17] characterize the probability of word sequences by neural networks, e.g., recurrent neural networks (RNNs). As a remarkable contribution, the work in [15] introduced the concept of *distributed representation of words* and built the word prediction function conditioned on the aggregated context features (i.e., the distributed word vectors). By extending the idea of learning effective features for words or sentences, a general neural network approach was developed to build

- Version: v12 (major update on September 10, 2023).
- GitHub link: <https://github.com/RUCAIBox/LLMSurvey>
- Chinese version link: [https://github.com/RUCAIBox/LLMSurvey/blob/main/assets/LLM\\_Survey\\_Chinese.pdf](https://github.com/RUCAIBox/LLMSurvey/blob/main/assets/LLM_Survey_Chinese.pdf)
- \* K. Zhou and J. Li contribute equally to this work.
- The authors are mainly with Gaoling School of Artificial Intelligence and School of Information, Renmin University of China, Beijing, China; Jian-Yun Nie is with DIRO, Université de Montréal, Canada.  
Contact e-mail: batmanfly@gmail.com
- The authors of this survey paper reserve all the copyrights of the figures/tables, and any use of these materials for publication purpose must be officially granted by the survey authors.

leads to the rethinking of the possibilities of artificial general intelligence (AGI). OpenAI has published a technical article entitled “*Planning for AGI and beyond*”, which discusses the short-term and long-term plans to approach AGI [40], and a more recent paper has argued that GPT-4 might be considered as an early version of an AGI system [41]. The research areas of AI are being revolutionized by the rapid progress of LLMs. In the field of NLP, LLMs can serve as a general-purpose language task solver (to some extent), and the research paradigm has been shifting towards the use of LLMs. In the field of IR, traditional search engines are challenged by the new information seeking way through AI chatbots (i.e., ChatGPT), and *New Bing* [8] presents an initial attempt that enhances the search results based on LLMs. In the field of CV, the researchers try to develop ChatGPT-like vision-language models that can better serve multimodal dialogues [42-45], and GPT-4 [46] has supported multimodal input by integrating the visual information. This new wave of technology would potentially lead to a prosperous ecosystem of real-world applications based on LLMs. For instance, Microsoft 365 is being empowered by LLMs (i.e., Copilot) to automate the office work, and OpenAI supports the use of plugins in ChatGPT for implementing special functions.

Despite the progress and impact, the underlying principles of LLMs are still not well explored. Firstly, it is mysterious why emergent abilities occur in LLMs, instead of smaller PLMs. As a more general issue, there lacks a deep, detailed investigation of the key factors that contribute to the superior abilities of LLMs. It is important to study when and how LLMs obtain such abilities [47]. Although there are some meaningful discussions about this problem [31, 47], more principled investigations are needed to uncover the “secrets” of LLMs. Secondly, it is difficult for the research community to train capable LLMs. Due to the huge demand of computation resources, it is very costly to carry out repetitive, ablating studies for investigating the effect of various strategies for training LLMs. Indeed, LLMs are mainly trained by industry, where many important training details (e.g., data collection and cleaning) are not revealed to the public. Thirdly, it is challenging to align LLMs with human values or preferences. Despite the capacities, LLMs are also likely to produce toxic, fictitious, or harmful contents. It requires effective and efficient control approaches to eliminating the potential risk of the use of LLMs [46].

Faced with both opportunities and challenges, it needs more attention on the research and development of LLMs. In order to provide a basic understanding of LLMs, this survey conducts a literature review of the recent advances in LLMs from four major aspects, including *pre-training* (how to pre-train a capable LLM), *adaptation* (how to effectively adapt pre-trained LLMs for better use), *utilization* (how to use LLMs for solving various downstream tasks) and *capability evaluation* (how to evaluate the abilities of LLMs and existing empirical findings). We thoroughly comb the literature and summarize the key findings, techniques, and methods of LLMs. For this survey, we also create a GitHub project website by collecting the supporting resources for LLMs, at the link <https://github.com/RUCAIBox/LLMSurvey>. We

3. <https://www.bing.com/new>

are also aware of several related review articles on PLMs or LLMs [32, 36, 38, 39, 43, 48-54]. These papers either discuss PLMs or some specific (or general) aspects of LLMs. Compared with them, we focus on the techniques and methods to develop and use LLMs and provide a relatively comprehensive reference to important aspects of LLMs.

The remainder of this survey is organized as follows: Section 2 introduces the background for LLMs and the evolution of GPT-series models, followed by the summarization of available resources for developing LLMs in Section 3. Sections 4, 5, 6, and 7 review and summarize the recent progress from the four aspects of pre-training, adaptation, utilization, and capacity evaluation, respectively. Then, Section 8 discusses the practical guide for prompt design, and Section 9 reviews the applications of LLMs in several representative domains. Finally, we conclude the survey in Section 10 by summarizing the major findings and discuss the remaining issues for future work.

## 2 OVERVIEW

In this section, we present an overview about the background of LLMs and then summarize the technical evolution of the GPT-series models.

### 2.1 Background for LLMs

Typically, *large language models* (LLMs) refer to Transformer language models that contain hundreds of billions (or more) of parameters [4], which are trained on massive text data [32], such as GPT-3 [55], PaLM [56], Galactica [35], and LLaMA [57]. LLMs exhibit strong capacities to understand natural language and solve complex tasks (via text generation). To have a quick understanding of how LLMs work, this part introduces the basic background for LLMs, including scaling laws, emergent abilities and key techniques.

**Scaling Laws for LLMs.** Currently, LLMs are mainly built upon the *transformer architecture* [22], where multi-head attention layers are stacked in a very deep neural network. Existing LLMs adopt similar Transformer architectures and pre-training objectives (e.g., language modeling) as small language models. However, LLMs significantly extend the model size, data size, and total compute (orders of magnification). Extensive research has shown that scaling can largely improve the model capacity of LLMs [26, 55, 56]. Thus, it is useful to establish a quantitative approach to characterizing the scaling effect. Next, we introduce two representative scaling laws for Transformer language models [30, 34].

• *KM scaling law* [5]. In 2020, Kaplan et al. [30] (the OpenAI team) firstly proposed to model the power-law relationship of model performance with respect to three major factors, namely model size ( $N$ ), dataset size ( $D$ ), and the amount of

4. In existing literature, there is no formal consensus on the minimum parameter scale for LLMs, since the model capacity is also related to data size and total compute. In this survey, we take a slightly loose definition of LLMs, and mainly focus on discussing language models with a model size larger than 10B.

5. Since there was not a model trained following this law in the original paper, we took the last names of the two co-first authors to name this scaling law.

cial form of prompting) endows LLMs with the ability to perform well on new tasks, even outperforming full-data fine-tuned models in some cases. Furthermore, to enhance the ability of complex reasoning, advanced prompting techniques have been proposed, exemplified by the chain-of-thought (CoT) strategy, which includes the intermediate reasoning steps into prompts. However, existing prompting approaches still have several deficiencies described as follows. Firstly, it involves considerable human efforts in the design of prompts. It would be quite useful to automatically generate effective prompts for solving various tasks. Secondly, some complex tasks (e.g., formal proof and numerical computation) require specific knowledge or logic rules, which may not be well expressed in natural language or demonstrated by examples. Thus, it is important to develop more informative, flexible task formatting methods for prompts<sup>54</sup>. Thirdly, existing prompting strategies mainly focus on single-turn performance. It is useful to develop interactive prompting mechanisms (e.g., through natural language conversations) for solving complex tasks, which have been demonstrated to be very useful by ChatGPT.

**Safety and Alignment.** Despite their capacities, LLMs pose similar safety challenges as small language models. For example, LLMs exhibit a tendency to generate hallucinations [518], which are texts that seem plausible but may be factually incorrect. What is worse, LLMs might be elicited by intentional instructions to produce harmful, biased, or toxic texts for malicious systems, leading to the potential risks of misuse [53, 61]. To have a detailed discussion of the safety issues of LLMs (e.g., privacy, overreliance, disinformation, and influence operations), the readers can refer to the GPT-3/4 technical reports [46, 53]. As the major approach to averting these issues, reinforcement learning from human feedback (RLHF) [61, 103] has been widely used by incorporating humans in the training loop for developing well-aligned LLMs. To improve the model safety, it is also important to include safety-relevant prompts during RLHF, as shown by GPT-4 [46]. However, RLHF heavily relies on high-quality human feedback data from professional labelers, making it difficult to be properly implemented in practice. Therefore, it is necessary to improve the RLHF framework for reducing the efforts of human labelers and seek a more efficient annotation approach with guaranteed data quality, e.g., LLMs can be employed to assist the labeling work. It is also meaningful to establish the proper learning mechanism for LLMs to obtain human feedback via chatting and directly utilize it for self-improvement. In addition, privacy concerns are also important to consider when fine-tuning LLMs with domain-specific data, and federated learning libraries [683] can be useful in privacy-restricted scenarios.

**Application and Ecosystem.** As LLMs have shown a strong capacity in solving various tasks, they can be applied in a broad range of real-world applications (i.e., following task-specific natural language instructions). As a remarkable progress, ChatGPT has potentially changed the way how

humans access information, which has been implemented in the release of *New Bing*. In the near future, it can be foreseen that LLMs would have a significant impact on information-seeking techniques, including both search engines and recommender systems. Furthermore, the development and use of intelligent information assistants would be highly promoted with the technology upgrade from LLMs. In a broader scope, this wave of technical innovation would lead to an ecosystem of LLM-empowered applications (e.g., the support of plugins by ChatGPT), which has a close connection with human life. Lastly, the rise of LLMs sheds light on the exploration of artificial general intelligence (AGI). It is promising to develop more smart intelligent systems (possibly with multi-modality signals) than ever. However, in this development process, AI safety should be one of the primary concerns, i.e., making AI lead to good for humanity but not bad [40].

#### CODA

It is not an easy job to write this long survey and update its content with timely work. First of all, we would like to sincerely thank the support from the readers and our team members. We work very hard on this survey, and hope that it can present a comprehensive, timely reference for LLMs.

**Survey Writing.** This survey was planned during a discussion meeting held by our research team, and we aimed to summarize the recent advances of large language models as a highly readable report for our team members. The first draft was finished on March 13, 2023, in which our team members tried their best to include the related studies about LLMs in a relatively objective, comprehensive way. Then, we have extensively revised the writing and contents in several passes. Due to the space limit, we can only include a fraction of existing LLMs in Figure 2 and Table 1, by setting the selection criterion. However, we set a more relaxed criterion for model selection on our GitHub page (<https://github.com/RUCAIBox/LLMSurvey>), which will be regularly maintained. We release the initial version on March 31, 2023, the major revision on June 29, 2023, and the latest version (v12) on September 10, 2023.

**Seeking for Advice.** Despite all our efforts, this survey is still far from perfect: we are likely to miss important references or topics, and might also have non-rigorous expressions or discussions. We will continuously update this survey, and improve the quality as much as we can. For us, survey writing is also a learning process for LLMs by ourselves. For readers with constructive suggestions to improve this survey, you are welcome to leave comments on the GitHub page of our survey or directly email our authors. We will make revisions following the received comments or suggestions in a future version, and acknowledge the readers who have contributed constructive suggestions in our survey.

**Update log.** In this part, we regularly maintain a update log for the submissions of this survey to arXiv:

- First release on March 31, 2023: the initial version.
- Update on April 9, 2023: add the affiliation information, revise Figure 2 and Table 1 and clarify the correspond-

<sup>54</sup> It seems that an alternative approach to this issue is to invoke external tools, e.g., the plugins for ChatGPT, when the task is difficult to solve via text generation.

## Termo de Aceite de Entrega 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]:

Heloisy Pereira Rodrigues

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

Prosseguindo no objetivo de me tornar especialista em Large Language Models, para essa semana, cumpri com o objetivo planejado, ou seja realizei as seguintes tarefas:

- Estudo dos pontos destacados a serem revisados do paper [A Survey of Large Language Models](#)
- Compilação de alguns desses pontos no documento [Estudo dos pontos em aberto do Survey](#)
- Definição de um tema a seguir seguido durante a especialização: Privacidade dos dados no uso de Large Language Models (na minha empresa), seja na criação de prompts seguros, seja na realização de treino ou fine tuning.
- Criação de um cronograma a ser seguido para as próximas entregas: [Cronograma Residência em IA](#)

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

Fazer a leitura e as considerações iniciais do paper “BEYOND MEMORIZATION: VIOLATING PRIVACY VIA INFERENCE WITH LARGE LANGUAGE MODELS”

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

### ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: **Go!** ▾

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

## Estudo de conceitos e técnicas relacionados - Semana 4

[O documento referente a seguir pode ser acessado em:

<https://docs.google.com/document/d/1D6Sp5yOT0eVQY5Q8EiE6hOkwvQQljZlXZ7OHyLn7CF0/edit?usp=sharing>

]

## Leitura dos links:

[Tranformers — Um guia introdutório](#)

[Simplifying Transformers: State of the Art NLP Using Words You Understand— Part 1 — Intro](#)

[Simplifying Transformers: State of the Art NLP Using Words You Understand — part 2— Input](#)

[Simplifying Transformers: State of the Art NLP Using Words You Understand — part 3— Attention](#)

[Simplifying Transformers: State of the Art NLP Using Words You Understand — Part 4 — Feed-Forward- Layer](#)

[Simplifying Transformers: State of the Art NLP Using Words You Understand — Part 5 — Decoder and Final Output](#)

[In Context Learning \(ICL\)](#)

[Two minutes NLP — A Taxonomy of Tokenization Methods](#)

## Considerações:

- Codificação e Decodificação:

O Transformer é comumente utilizado em tarefas de sequência para sequência, como tradução automática. Ele possui duas partes principais: a codificação (Encoder) e a decodificação (Decoder).

O Encoder processa a entrada (por exemplo, uma frase em inglês) e o Decoder gera a saída correspondente (por exemplo, a tradução em francês).

- Autoatenção (Self Attention):

Este é o coração do Transformer. A autoatenção permite que o modelo dê diferentes pesos a diferentes partes da entrada, dependendo da relevância para cada palavra.

Imagine uma palavra em uma frase, e a autoatenção permite que o modelo olhe para todas as outras palavras na frase para entender melhor o contexto.

- Multi-Head Attention:

Para melhorar a robustez, a autoatenção é geralmente estendida para várias "cabeças" (ou perspectivas). Cada cabeça aprende diferentes representações da relação entre palavras. A saída de múltiplas cabeças é combinada para fornecer uma representação mais rica e abrangente.

- Camadas Residuais e Normalização:

Cada subcamada (como a autoatenção) é seguida por uma camada residual, que ajuda a prevenir problemas de treinamento profundo.

A normalização é aplicada para garantir que os dados estejam em uma escala adequada.

- Feedforward e Camadas Lineares:

Cada subcamada (tanto no Encoder quanto no Decoder) possui uma camada feedforward para realizar transformações lineares nos dados.

Camadas lineares são usadas para ajustar as dimensões dos vetores e preparar os dados para as próximas etapas.

- Embeddings Posicionais:

Como o Transformer não possui noção intrínseca de ordem nas sequências, são adicionados embeddings posicionais para indicar a posição relativa de cada palavra na frase.

- Função de Ativação e Softmax:

Funções de ativação, como a ReLU, são aplicadas para introduzir não linearidades.

A função Softmax é usada para converter os pesos da autoatenção em uma distribuição de probabilidade.

Como a autoatenção realmente funciona?

Dada a limitação da atenção, cada vetor recebe um peso, reduzindo ruídos e focalizando a atenção do modelo. Isso resulta em uma versão contextualizada das palavras, considerando suas relações na frase. As operações matemáticas realizadas incluem a multiplicação do embedding de cada palavra,  $V_i$ , pelo embedding de todas as outras palavras, incluindo o próprio. Esta multiplicação quantifica a influência entre as palavras, sendo máxima para palavras idênticas e nula para opostas.

Após obtermos os pesos  $S_{ij}$ , normalizamos para que a soma seja 1, gerando os valores normalizados  $W_{ij}$ . Para obter o valor contextualizado  $Y_i$ , multiplicamos cada valor normalizado  $S_{ij}$  pelos embeddings originais da palavra  $V_j$ . Esse processo, apesar de limitado na quantidade de palavras que podem ser consideradas simultaneamente, é aplicado a todos os embeddings, determinando as relações entre cada palavra da frase.

Apesar dos embeddings não serem assim, um bom exemplo é:

$$\text{Amendoim} = (1, 0, 0, 0, 1, 1, 0, 1)$$

$$\text{Moletom} = (0, 1, 1, 1, 0, 0, 1, 0)$$

$$\text{Amendoim} \cdot \text{Amendoim} =$$

$$1*1 + 0*0 + 0*0 + 0*0 + 1*1 + 1*1 + 0*0 + 1*1 = 4$$

$$\text{Amendoim} \cdot \text{Moletom} =$$

$$1*0 + 0*1 + 0*1 + 0*1 + 1*0 + 1*0 + 0*1 + 1*0 = 0$$

O Aprendizado em Contexto (ICL) permite que modelos de linguagem aprendam novas tarefas sem ajustes finos, incorporando exemplos no estímulo. A inserção de contextos relevantes no início ou no final do estímulo melhora o desempenho. O tamanho do modelo e a janela de contexto afetam a eficácia do ICL. Zero-shot learning, sem dados rotulados, é preferido, mas ICL pode usar poucos exemplos.

- Tokenização por Palavra: Divide o texto em palavras individuais.  
Exemplo: "Hello, world!" seria tokenizado em ["Hello", ",", "world", "!"].
- Tokenização por Subpalavra: Divide o texto em partes menores, úteis para lidar com palavras compostas ou diferentes formas de uma palavra.  
Exemplo: "Unsupervised" poderia ser tokenizado em ["Un", "super", "vised"].
- Tokenização por Caractere: Divide o texto em caracteres individuais.  
Exemplo: "Machine" seria tokenizado em ["M", "a", "c", "h", "i", "n", "e"].
- Tokenização com N-gramas: Cria tokens com base em subsequências contínuas de N elementos (palavras ou caracteres).  
Exemplo: Para N=2, "Natural Language Processing" seria tokenizado em ["Natural", "Language", "Processing", "Natural Language", "Language Processing"].
- Tokenização com Expressões Regulares: Usa padrões definidos por expressões regulares para identificar tokens.  
Exemplo: Pode ser usado para extrair menções de hashtags em tweets (#tokenization).
- Tokenização com Aprendizado de Máquina: Utiliza modelos de aprendizado de máquina para identificar padrões e separar tokens.  
Exemplo: Modelos treinados para reconhecer entidades específicas ou contextos linguísticos.

- Tokenização para Redes Neurais: Projetada para atender às necessidades de modelos de linguagem, geralmente dividindo o texto em unidades menores, como subpalavras.

Exemplo: WordPiece, SentencePiece.

## APÊNDICE 3

## Termo de Aceite de Entrega 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]:

Heloisy Pereira Rodrigues

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

No intuito de me tornar especialista em LLM, havia pensado em me especializar em uma parte do tema direcionado a proteção dos dados de treinamento, mas esse direcionamento estava muito ligado à LGPD. Como esse não era o meu foco, voltei ao início, após a leitura e entendimento geral do Survey de LLM's, resolvi então agora ir por parte de cada assunto importante retratado lá, voltando à questão principal: ser especialista em LLM.

Dito isso, abandonei a estratégia pensada na semana passada e fiz um novo cronograma, disponibilizado no link a seguir: [Cronograma Residência em IA v2](#) voltando a programação normal, partindo dos tópicos do survey.

Além disso, para essa semana fiquei em busca de mapear ferramentas e recursos disponíveis para uso futuro, quando for para prática dos conhecimentos adquiridos na especialização, seja um treinamento de um modelo, seja um fine tuning. Essa documentação é encontrada no link a seguir: [Ferramentas e recursos catalogados - LLM Residência IA](#).

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

Seguindo o novo cronograma, para a próxima semana pretendo estudar mais sobre pré-treino, o que é necessário para fazê-lo e como arquiteta-lo.

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

### ACEITE DA ENTREGA:

**CEDRIC LUIZ DE CARVALHO:** Go! ▾

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

## Estudo de ferramentas e recursos - Semana 5

[O documento referente a seguir pode ser acessado em:

<https://docs.google.com/document/d/1BnK1qzITte2hrquSNFP2QJepgWVngpja4Y2FGXYOnBA/edit?usp=sharing>]

# Ferramentas e recursos disponíveis

## Web Scraping:

- [BeautifulSoup](#)
  - Prós:
    - Simplicidade e adaptabilidade no código.
    - Comunidade forte e documentação abrangente.
  - Contras:
    - Dificuldade com proxies para evitar bloqueios de IP quando tiver extraído um grande volume de dados.
    - Dependências necessárias; não funciona como parser independente.
- [Scrapy](#)
  - Prós:
    - Fácil de estender para adicionar funcionalidades.
    - Possui ferramenta interativa para testar e depurar.
    - Bom suporte para lidar com diferentes tipos de caracteres.
  - Contras:
    - Pode ter dificuldades com sites que usam JavaScript.
    - A instalação pode variar conforme o sistema operacional.
- [Selenium](#)
  - Prós:
    - Automatiza tarefas online, como fazer login em redes sociais.
    - Funciona bem em páginas web com JavaScript.
    - Suporta vários navegadores.
  - Contras:
    - Pode ser detectado por sistemas de rastreamento como o Google Analytics.
    - Consome mais recursos do sistema e pode ativar respostas de segurança ao carregar o navegador na memória.
- [Requests](#)
  - Prós:
    - Facilita pedir informações de sites.
    - Suporte a API restful e funcionalidades como PUT, GET, DELETE e POST.

- Ajuda a evitar escrever manualmente partes complicadas dos endereços dos sites.
- Contras:
  - Não é seguro para enviar informações sensíveis pela URL.
  - Pode não funcionar bem em sites com muito JavaScript.

## Pré-Processamento dos dados:

### Quality Filtering:

- [NLTK](#)
  - Oferece ferramentas para análise de qualidade linguística.
- [spaCy](#)
  - Pode ajudar na identificação e filtragem de tokens irrelevantes.
- [Regex](#)
  - Útil para remover caracteres indesejados ou padrões específicos.

### Deduplication:

- [Pandas](#)
  - Permite a identificação e remoção de duplicatas em conjuntos de dados tabulares.

### Privacy Reduction:

- [Transformers](#)
  - Modelos pré-treinados podem ser usados para ofuscar ou reduzir informações sensíveis.
- [TextBlob](#)
  - Pode ajudar na análise de sentimentos sem expor detalhes específicos.

### Tokenization:

- [NLTK](#)
  - Oferece tokenizadores eficazes.
- [spaCy](#)
  - Realiza tokenização eficiente, análise morfológica e lematização.

- [Regex](#)
  - Pode ser utilizado para tokenização baseada em padrões.
- [Transformers](#)
  - Fornece ferramentas específicas para tokenização adequada a modelos avançados.

## LLM Training:

- [Megatron-LM](#)
  - Desenvolvido pela equipe de Pesquisa em Aprendizado Profundo Aplicado da NVIDIA.
  - Destinado à pesquisa no treinamento de grandes modelos de linguagem em escala.
  - Suporta paralelismo de modelos (tensor, sequência e pipeline) e pré-treinamento multinodal de modelos transformadores.
  - Oferece suporte a modelos como BERT, GPT e T5.
- [DeepSpeed](#)
  - Biblioteca de aprendizado profundo da Microsoft.
  - Usada para treinar grandes modelos como Megatron-Turing NLG 530B e BLOOM.
  - Inova em treinamento, inferência e compressão.
  - Benefícios incluem treino/inferência de modelos densos ou esparsos com bilhões ou trilhões de parâmetros, alto throughput do sistema, eficiência em milhares de GPUs, baixa latência e compressão extrema.
- [FairScale](#)
  - Extensão PyTorch para treinamento em larga escala e alto desempenho, da pesquisa do Facebook.
  - Visão: Usabilidade, Modularidade e Performance.
  - Suporta FullyShardedDataParallel (FSDP) para escalar o treinamento de grandes redes neurais.
- [ParallelFormers](#)
  - Biblioteca baseada no Megatron-LM, integrada com Hugging Face.
  - Permite a paralelização fácil de modelos na biblioteca Hugging Face para inferência.
- [ColossalAI](#)
  - Fornece componentes paralelos para implementar treinamento distribuído/paralelo.

- Estratégias incluem Data Parallelism, Pipeline Parallelism, Tensor Parallelism (1D, 2D, 2.5D, 3D), Sequence Parallelism, Zero Redundancy Optimizer (ZeRO), e Heterogeneous Memory Management (PatrickStar).
- Inclui Energon-AI para inferência.
- **Alpa**
  - Sistema para treinar e servir redes neurais em grande escala.
  - Características: Automatic Parallelization, Excelente Desempenho, Integração Estreita com Ecosistema de Aprendizado de Máquina.
- **Hivemind**
  - Biblioteca para aprendizado profundo descentralizado usando PyTorch pela internet.
  - Características: Treinamento distribuído sem um nó mestre, Backpropagation tolerante a falhas, Média de parâmetros descentralizada, Treinamento de redes neurais de tamanho arbitrário.
- **OneFlow**
  - Framework de aprendizado profundo projetado para ser fácil de usar, escalável e eficiente.
  - Permite programar modelos com API semelhante ao PyTorch, escalar modelos para execução distribuída e acelerar/deployar modelos com o Static Graph Compiler.
- **Mesh-Tensorflow**
  - Linguagem para aprendizado profundo distribuído, capaz de especificar uma ampla classe de computações de tensor distribuídas.
  - Refere-se à rede interconectada de processadores ou dispositivos de computação.

## Quantization:

- **Bitsandbytes**
  - Fornece funções CUDA de 8 bits para PyTorch 1. Ele contém funções personalizadas para otimizadores de 8 bits, multiplicação de matriz (LLM.int8()) e funções de quantização 1. O pacote é uma camada leve em torno dessas funções personalizadas.
- **GPTQ-for-LLaMA**
  - Quantização de 4 bits do LLaMA usando GPTQ.
  - Criadores recomendam usar o Auto-GPTQ pois migraram esforços para lá
- **Auto-GPTQ**

- Um pacote de quantização de modelos de linguagem leve e fácil de usar, com interfaces de programação amigáveis ao usuário, baseado no algoritmo GPTQ.
- [llama.cpp](#)
  - Executar o modelo LLaMA usando quantização de inteiros de 4 bits em um MacBook.

## Datasets:

- [CommonCrawl](#)
  - multilingual
- [RedPajama](#)
  - multilingual
- [RedefinedWeb](#)
  - inglês
- [The Pile](#)
  - inglês
- [C4](#)
  - multilingual
- [StarcoderData](#)
  - multilingual
- [BookCorpus](#)
  - inglês
- [Alpaca-Data-pt-br](#)
  - português
- [Instructions-pt](#)
  - português
- [Canarim](#)
  - português

## Evaluation:

- [OpenAI Evals](#)

- Open Source de avaliações desafiadoras e suporta a avaliação de sistemas, incluindo correntes de prompts ou agentes que utilizam ferramentas. O objetivo é simplificar a criação de avaliações, exigindo o mínimo de código possível.
- **Language Model Evaluation Harness**
  - Inclui 200+ sub-tarefas, suporte a diversos modelos e APIs comerciais, avaliação em adaptadores, e promove a reprodutibilidade através de prompts publicamente disponíveis e versionamento de tarefas.
- **Holistic Evaluation of Language Models**
  - É um pacote completo, com padronização de conjuntos de dados, acesso unificado a diversos modelos de linguagem, métricas avançadas, avaliação de robustez e imparcialidade
  - Estrutura modular para construção de prompts a partir de dados, além de um servidor proxy para gerenciamento de contas e acesso simplificado aos modelos.
- **IntelliServer**
  - Facilita a avaliação de modelos de linguagem por meio de um microserviço unificado para acessar e testar vários modelos.
- **YiVal**
  - YiVal é uma ferramenta de GenAI-Ops de código aberto para ajustar e avaliar prompts, configurações e parâmetros do modelo usando conjuntos de dados personalizáveis, métodos de avaliação e estratégias de aprimoramento.

## APÊNDICE 4

## Termo de Aceite de Entrega 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]:

Heloisy Pereira Rodrigues

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

Após o Gate da semana passada, com o comentário da professora Luana, para não andar em círculos, resolvi ir para o meu foco inicial que era fine tuning de algum modelo para resolução de questões de concurso/vestibular. Apesar disso, não achei dataset da área, nem em português, mas encontrei um semelhante à ideia em inglês, denominado [Cosmos QA](#) e resolvi fazer a tradução dele com o GPT. O dataset é dividido em treino, validação e teste, somando mais de 35k de perguntas e respostas, de problemas que exigem a compreensão da leitura baseada no senso comum. Vale ressaltar que são problemas de múltipla escolha.  
Faltam poucos dados para a conclusão integral da tradução.

O processo está documentado e pode ser visualizado juntamente com o código utilizado para fazer a tradução a seguir: [Entrega 30/11/2023](#)

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

Término da tradução, resolução do gargalo encontrado na tradução do dataset (GPT fornecia formato errado) e busca/escolha de arquitetura(s) para o fine tuning que condizem com o meu propósito, assim como escolher qual framework para fine tuning usarei. Caso consiga extrapolar, iniciar o pipeline do fine tuning.

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

Agradeço ao Maicon Domingues e ao Luiz Guilherme por me ajudarem no processo. E peço desculpas por mudar o cronograma novamente.

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!** ▾

## Detalhamento do processo - Semana 6

[O documento referente a seguir pode ser acessado em:

<https://docs.google.com/document/u/0/d/1kkgEmSx89AT773id907j1EnuEH85AtpINzjCYCFr-XE/edit>]

## Stage 6

Contextualizando: com a fala da professora Luana, senti novamente que ia rodar em círculos lendo os artigos sobre cada sub tópico do Survey e resolvi colocar a mão na massa. Então voltei para uma vontade que tinha em mim de realizar um fine tuning voltado para questões de prova, para ajudar concurseiros e afins. Diante disso, fui procurar alguns dados que me permitissem fazer o desejado.

Apesar de na semana passada eu ter documentado alguns datasets para fine tuning, nenhum era parecido com esse meu propósito, portanto a pesquisa teve que ser feita novamente. Após algumas pesquisas, encontrei o que parecia ideal para mim, um huggingface, do mesmo criador do [Canarim](#), um modelo com o nome [Canarim-7B-VestibulAide](#) feito para lidar com questões, exercícios e respostas de vestibulares brasileiros. Mandei email para o Domingues, muito solícito, perguntando sobre os dados que ele utilizou, se iria disponibilizá-los, etc. A resposta foi que ele está finalizando um artigo sobre o modelo e irá guardar os dados até publicá-lo, mas me disse que criou uma série de parsers para extrair as informações de provas de diversas fontes, livros “do professor” e o que mais achou interessante. Novamente tive que voltar às pesquisas, mas agora já procurando algo que se assemelhasse, para eu realizar todos os passos do fine tuning e depois, quando tiver tempo futuramente, me dedicar a montar o meu dataset de questões ou até mesmo pegar os dados do Domingues quando ele disponibilizar.

Na nova saga, encontrei o [Cosmos QA](#), retirando e traduzindo a definição do próprio site deles: Cosmos QA é um conjunto de dados em larga escala com 35,6 mil problemas que exigem compreensão de leitura baseada no senso comum, formulados como perguntas de múltipla escolha. Ele se concentra na leitura entre as linhas em uma coleção diversificada de narrativas cotidianas das pessoas, fazendo perguntas sobre as causas ou efeitos prováveis de eventos que exigem raciocínio além dos trechos exatos de texto no contexto. Além disso, ele se baseia nas seguintes questões:

- Causas de eventos: Qual pode (ou não) ser a razão plausível para um evento?
- Efeitos de eventos: O que pode (ou não) acontecer antes (ou depois, ou durante) um evento?
- Fatos sobre entidades: O que pode (ou não) ser um fato plausível sobre alguém ou algo?
- Contrafactuais: O que pode (ou não) acontecer se um evento acontecer (ou não acontecer)?

O modelo adotado por ele se assemelha às questões de prova que tenho interesse futuro, mas é em inglês, exemplo:

### Example 1

**Paragraph:** It's a very humbling experience when you need someone to dress you every morning, tie your shoes, and put your hair up. Every menial task takes an unprecedented amount of effort. It made me appreciate Dan even more. But anyway I shan't dwell on this (I'm not dying after all) and not let it detract from my lovely 5 days with my friends visiting from Jersey.

**Question:** What's a possible reason the writer needed someone to dress him every morning?

**Options:** (click the choice to see if it's correct or not)

- |   |
|---|
| A. The writer doesn't like putting effort into these tasks. |
| B. The writer has a physical disability.                    |
| C. The writer is bad at doing his own hair.                 |
| D. None of the above choices.                               |

No [huggingface](#) ele está assim:

```
{
  "answer0": "If he gets married in the church he wo nt have to get a divorce .",
  "answer1": "He wants to get married to a different person .",
  "answer2": "He wants to know if he does nt like this girl can he divorce her ?",
  "answer3": "None of the above choices .",
  "context": "\"Do i need to go for a legal divorce ? I wanted to marry a woman but she is not in the same religion , so i am not concern of th...\"",
  "id":
  "3BFF0DJK8XA7YNK4QYIGCOG1A95STE##3180JW2OT5AF02OISBX66RFOCTG5J7##A2LTOS0AZ3B28A##Blog_56156##q1_a1##378G7J1SJNCDAAIN46FM2P7T6KZEW2",
  "label": 1,
  "question": "Why is this person asking about divorce ?"
}
```

Perguntei pro Luiz Guilherme se ele achava pertinente também traduzir os dados usando GPT, como foi feito por ele, e foi esse o caminho que tomei.

Fiz um script para realizar a tradução automática, com o GPT 3-5 Turbo e deixei rodando por (muitas e muitas) horas. Quando coloquei pra rodar, durante uma madrugada e uma manhã, e ao rever o código feito na noite anterior vi que havia cometido um deslize e não estava

salvando o contexto traduzido, logo praticamente 1 dia de trabalho foi para o lixo. Ajustei o código e coloquei para rodar novamente.

Um exemplo da tradução está aqui: {'pergunta': 'Os canhões eram barulhentos?', 'A': 'O canhão disparou silenciosamente.', 'B': 'Os canhões são barulhentos o suficiente para surpreender as pessoas.', 'C': 'O canhão falhou em disparar.', 'D': 'O canhão não soou bem.', 'contexto': 'O que eu disparei foi apelidado de "Munchkin" porque era tão pequeno. Enquanto os reinos entravam no campo e começavam as cerimônias, eu não pude deixar de sorrir sabendo a quantidade de pessoas que se assustariam com o som dos canhões. Então, o sinal foi dado e eu acendi o canhão. Foi INCRÍVEL!', 'Label': 'B'}

Até o gate a tradução não terá terminado, pois foi cerca de 10h para cada 5000 dados, e tive que adaptar o código pois as vezes o GPT não retorna a saída 100% como desejado, o que estava dando erro, arrumei de forma a salvar os índices que não era a saída esperada e após a finalização dessa primeira leva de tradução, voltarei traduzindo esses gargalos.

Com isso, subirei no Hugging Face os dados traduzidos.

O link do código utilizado para tradução: [Tradução CosmosQA Exemplo](#)

## Notebook de tradução do Cosmos QA

[O documento referente a seguir pode ser acessado em:

[https://colab.research.google.com/drive/1cnFW3XOVxSJCwuT1\\_uWfU9EzLhhfQADx?usp=sharing](https://colab.research.google.com/drive/1cnFW3XOVxSJCwuT1_uWfU9EzLhhfQADx?usp=sharing)]

```
# v1

import os
import time
import openai
from tqdm import tqdm
import pandas as pd
from datasets import load_dataset

class ChatGPT:
    def __init__(self, api_key_path: str):
        openai.api_key_path = api_key_path
        self.messages = []

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

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

    def add_context(self, context: str):
        context = {'role': 'system', 'content': context}
        self.messages = [context] + self.messages

    def get_completion(self, model="gpt-3.5-turbo", temperature=0):
        retries = 0
        max_retries = 3
```

```
while retries < max_retries:
    try:
        response = openai.ChatCompletion.create(
            model=model,
            messages=self.messages,
            temperature=temperature,
        )
        return response.choices[0].message["content"], response
    except Exception as e:
        model = "gpt-3.5-turbo-1106"
        retries += 1
        time.sleep(2)
```

```
gpt = ChatGPT(r"C:\Users\heloi\Downloads\key.txt")
```

```
cosmosqa = load_dataset("cosmos_qa")["train"]
```

```
dados_cosmosqa_traduzidos = []
```

```
dados_cosmosqa_nao_traduzidos = []
```

```
def traduzir_texto(chat_gpt, prompt):
```

```
    chat_gpt.add_message(role="user", prompt=prompt)
```

```
    # fazendo inferencia
```

```
    resposta, _ = chat_gpt.get_completion()
```

```
    chat_gpt.reset_messages(maintain_context=False)
```

```
    return resposta
```

Fiz a tradução em blocos

```
for index in tqdm(range(0, 5000)):
    # Extrai informações do conjunto de dados COSMOS QA
    context = cosmosqa[index]['context']
    question = cosmosqa[index]['question']
    answers = [cosmosqa[index][f'answer{i}'] for i in range(4)]
    label = cosmosqa[index]['label']

    # Cria a pergunta formatada
    pergunta = f"{question}\n"
    for i, answer in enumerate(answers):
        pergunta += f"{chr(65 + i)} {answer}\n"

    # Cria o prompt de tradução
    prompt = f"Você irá receber uma questão de múltipla escolha, em inglês, de modo que a
    você será passado o enunciado, junto com as alternativas, demarcadas por 'A)', 'B)', 'C)' e
    'D)'. Você deve traduzir a pergunta para o português. Ao traduzir a pergunta, mantenha as
    alternativas 'A)', 'B)', 'C)' e 'D)', com suas respectivas traduções, não fuja desse formato.
    Também, fique atento a possíveis abreviações, expressões populares e polissemias, e
    traduza da melhor forma possível para o português, removendo possíveis ruídos do texto
    também. A pergunta/questão está marcada com a tag <pergunta>.\n Retorne a tradução da
    pergunta em um json, de modo que a pergunta traduzida esteja em uma chave 'pergunta'.
    Segue um exemplo, a ser seguido por você, de retorno: '{{'pergunta': 'enunciado da questao',
    'A': 'letra A', 'B': 'letra B', 'C': 'letra C', 'D': 'letra D', 'contexto': 'contexto da questao'}}'.
    \n\n
    <contexto> {context} <contexto> <pergunta> {pergunta} <pergunta>."
```

try:

```
    # Traduz a pergunta
    resposta = traduzir_texto(gpt, prompt).replace("\n", ' ')
```

```
if resposta and resposta[-1] != "":  
    resposta = resposta[:-1]  
    resposta_traduzida = eval(resposta)  
    resposta_traduzida['Label'] = f"{chr(65 + label)}"  
    print(resposta_traduzida)  
    dados_cosmosqa_traduzidos.append([resposta_traduzida])
```

```
except Exception as e:  
    print(f"Erro durante a tradução: {e}")  
    nao_traduzido = f"{cosmosqa[index]} não traduzido"  
    dados_cosmosqa_nao_traduzidos.append([nao_traduzido])  
    continue
```

```
df = pd.DataFrame([item[0] for item in dados_cosmosqa_traduzidos])
```

```
df
```

```
df.to_csv("cosmos0_5000.csv",index=False)
```

```
df2 = pd.DataFrame({'index': dados_cosmosqa_nao_traduzidos})
```

```
df2.to_csv("numeros_erro0_5000.csv",index=False)
```

```
# v2
```

```
import os
```

```
import time
```

```
import openai
```

```
class ChatGPT:
    def __init__(self, api_key_path: str):
        openai.api_key_path = api_key_path
        self.messages = []

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

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

    def add_context(self, context: str):
        context = {'role': 'system', 'content': context}
        self.messages = [context] + self.messages

    def get_completion(self, model="gpt-3.5-turbo", temperature=0):
        retries = 0
        max_retries = 3

        while retries < max_retries:
            try:
                response = openai.ChatCompletion.create(
                    model=model,
                    messages=self.messages,
                    temperature=temperature,
                )
                return response.choices[0].message["content"], response
```

```
except Exception as e:
    model = "gpt-3.5-turbo-1106"
    retries += 1
    time.sleep(2)

from tqdm import tqdm
import json

from datasets import load_dataset

cosmosqa = load_dataset("cosmos_qa")["train"]

gpt = ChatGPT(r"C:\Users\helo\Downloads\keycosmos.txt")

def traduzir_texto(chat_gpt, prompt):

    chat_gpt.add_message(role="user", prompt=prompt)

    # fazendo inferencia
    resposta, _ = chat_gpt.get_completion()

    chat_gpt.reset_messages(maintain_context=False)

    return resposta

dados_cosmosqa_traduzidos = []
dados_cosmosqa_nao_traduzidos = []

for index in tqdm(range(5001, 10000)):
    # Extrai informações do conjunto de dados COSMOS QA
    context = cosmosqa[index]["context"]
```

```
question = cosmosqa[index]['question']  
answers = [cosmosqa[index][f'answer{i}']] for i in range(4)  
label = cosmosqa[index]['label']
```

```
# Cria a pergunta formatada
```

```
pergunta = f"{question}\n"
```

```
for i, answer in enumerate(answers):
```

```
    pergunta += f"{chr(65 + i)} {answer}\n"
```

```
# Cria o prompt de tradução
```

```
prompt = f"Você irá receber uma questão de múltipla escolha, em inglês, de modo que a  
você será passado o enunciado, junto com as alternativas, demarcadas por 'A)', 'B)', 'C)' e  
'D)'. Você deve traduzir a pergunta para o português. Ao traduzir a pergunta, mantenha as  
alternativas 'A)', 'B)', 'C)' e 'D)', com suas respectivas traduções, não fuja desse formato.  
Também, fique atento a possíveis abreviações, expressões populares e polissemias, e  
traduza da melhor forma possível para o português, removendo possíveis ruídos do texto  
também. A pergunta/questão está marcada com a tag <pergunta>.\n Retorne a tradução da  
pergunta em um json, de modo que a pergunta traduzida esteja em uma chave 'pergunta'.  
Segue um exemplo, a ser seguido por você, de retorno: '{{'pergunta': 'enunciado da questao',  
'A': 'letra A', 'B': 'letra B', 'C': 'letra C', 'D': 'letra D', 'contexto': 'contexto da questao'}}'. \n \n  
<contexto> {context} <contexto> <pergunta> {pergunta} <pergunta>."
```

```
try:
```

```
    # Traduz a pergunta
```

```
    resposta = traduzir_texto(gpt, prompt).replace('\n', ' ')
```

```
    if resposta and resposta[-1] != "}":
```

```
        resposta = resposta[:-1]
```

```
    resposta_traduzida = eval(resposta)
```

```
    resposta_traduzida['Label'] = f"{chr(65 + label)}"
```

```
    print(resposta_traduzida)
```

```
    dados_cosmosqa_traduzidos.append([resposta_traduzida])
```

```
except Exception as e:
    print(f"Erro durante a tradução: {e}")
    nao_traduzido = f"{cosmosqa[index]} não traduzido"
    dados_cosmosqa_nao_traduzidos.append([nao_traduzido])
    continue

import pandas as pd
df = pd.DataFrame([item[0] for item in dados_cosmosqa_traduzidos])
df

df.to_csv("cosmos5001_10000.csv",index=False)

df2 = pd.DataFrame({'index': dados_cosmosqa_nao_traduzidos})
df2

df2.to_csv("numeros_erro5001_10000.csv",index=False)

dados_cosmosqa_traduzidos = []
dados_cosmosqa_nao_traduzidos = []

for index in tqdm(range(15000, 20000)):
    # Extrai informações do conjunto de dados COSMOS QA
    context = cosmosqa[index]['context']
    question = cosmosqa[index]['question']
    answers = [cosmosqa[index][f'answer{i}'] for i in range(4)]
    label = cosmosqa[index]['label']

    # Cria a pergunta formatada
    pergunta = f"{question}\n"
    for i, answer in enumerate(answers):
```

```
pergunta += f"{chr(65 + i)} {answer}\n"
```

```
# Cria o prompt de tradução
```

```
prompt = f"Você irá receber uma questão de múltipla escolha, em inglês, de modo que a  
você será passado o enunciado, junto com as alternativas, demarcadas por 'A)', 'B)', 'C)' e  
'D)'. Você deve traduzir a pergunta para o português. Ao traduzir a pergunta, mantenha as  
alternativas 'A)', 'B)', 'C)' e 'D)', com suas respectivas traduções, não fuja desse formato.  
Também, fique atento a possíveis abreviações, expressões populares e polissemias, e  
traduza da melhor forma possível para o português, removendo possíveis ruídos do texto  
também. A pergunta/questão está marcada com a tag <pergunta>.\n Retorne a tradução da  
pergunta em um json, de modo que a pergunta traduzida esteja em uma chave 'pergunta'.  
Segue um exemplo, a ser seguido por você, de retorno: '{{'pergunta': 'enunciado da questao',  
'A': 'letra A', 'B': 'letra B', 'C': 'letra C', 'D': 'letra D', 'contexto': 'contexto da questao'}}'. \n \n  
<contexto> {context} <contexto> <pergunta> {pergunta} <pergunta>."
```

```
try:
```

```
# Traduz a pergunta
```

```
resposta = traduzir_texto(gpt, prompt).replace("\n", ' ')
```

```
if resposta and resposta[-1] != "}":
```

```
    resposta = resposta[:-1]
```

```
resposta_traduzida = eval(resposta)
```

```
resposta_traduzida['Label'] = f"{chr(65 + label)}"
```

```
print(resposta_traduzida)
```

```
dados_cosmosqa_traduzidos.append([resposta_traduzida])
```

```
except Exception as e:
```

```
    print(f"Erro durante a tradução: {e}")
```

```
    nao_traduzido = f"{cosmosqa[index]} não traduzido"
```

```
    dados_cosmosqa_nao_traduzidos.append([nao_traduzido])
```

```
    continue
```

```
df = pd.DataFrame([item[0] for item in dados_cosmosqa_traduzidos])
```

```
df
```

```
df.to_csv("cosmos15000_20000.csv",index=False)
```

```
df2 = pd.DataFrame({'index': dados_cosmosqa_nao_traduzidos})
```

```
df2
```

```
df2.to_csv("numeros_erro15000_20000.csv",index=False)
```

```
rodei ate 2808
```

```
dados_cosmosqa_traduzidos = []
```

```
dados_cosmosqa_nao_traduzidos = []
```

```
for index in tqdm(range(17641, 20000)):
```

```
    # Extrai informações do conjunto de dados COSMOS QA
```

```
    context = cosmosqa[index]['context']
```

```
    question = cosmosqa[index]['question']
```

```
    answers = [cosmosqa[index][f'answer{i}'] for i in range(4)]
```

```
    label = cosmosqa[index]['label']
```

```
    # Cria a pergunta formatada
```

```
    pergunta = f"{question}\n"
```

```
    for i, answer in enumerate(answers):
```

```
        pergunta += f"{chr(65 + i)} {answer}\n"
```

```
    # Cria o prompt de tradução
```

```
    prompt = f"Você irá receber uma questão de múltipla escolha, em inglês, de modo que a  
você será passado o enunciado, junto com as alternativas, demarcadas por 'A)', 'B)', 'C)' e  
'D)'. Você deve traduzir a pergunta para o português. Ao traduzir a pergunta, mantenha as
```

alternativas 'A)', 'B)', 'C)' e 'D)', com suas respectivas traduções, não fuja desse formato. Também, fique atento a possíveis abreviações, expressões populares e polissemias, e traduza da melhor forma possível para o português, removendo possíveis ruídos do texto também. A pergunta/questão está marcada com a tag <pergunta>.\n Retorne a tradução da pergunta em um json, de modo que a pergunta traduzida esteja em uma chave 'pergunta'. Segue um exemplo, a ser seguido por você, de retorno: '{'pergunta': 'enunciado da questao', 'A': 'letra A', 'B': 'letra B', 'C': 'letra C', 'D': 'letra D', 'contexto': 'contexto da questao'}'. \n \n <contexto> {context} <contexto> <pergunta> {pergunta} <pergunta>."

try:

```
# Traduz a pergunta
resposta = traduzir_texto(gpt, prompt).replace('\n', ' ')
if resposta and resposta[-1] != "}":
    resposta = resposta[:-1]
resposta_traduzida = eval(resposta)
resposta_traduzida['Label'] = f"{{chr(65 + label)}}"
print(resposta_traduzida)
dados_cosmosqa_traduzidos.append([resposta_traduzida])
```

except Exception as e:

```
print(f"Erro durante a tradução: {e}")
nao_traduzido = f"{{cosmosqa[index]}} não traduzido"
dados_cosmosqa_nao_traduzidos.append([nao_traduzido])
continue
```

```
df = pd.DataFrame([item[0] for item in dados_cosmosqa_traduzidos])
df
```

```
df.to_csv(r"C:\Users\heloi\Downloads\cosmos17641_20000.csv", index=False)
```

```
df2 = pd.DataFrame({'index': dados_cosmosqa_nao_traduzidos})
```

df2

```
df2.to_csv(r"C:\Users\heloi\Downloads\numeros_erro17641_20000.csv",index=False)
```

dados de teste

```
cosmosqa = load_dataset("cosmos_qa")["test"]
```

```
dados_cosmosqa_traduzidos = []
```

```
dados_cosmosqa_nao_traduzidos = []
```

```
gpt = ChatGPT(r"C:\Users\heloi\Downloads\key.txt")
```

```
for index in tqdm(range(0, len(cosmosqa))):
```

```
    # Extrai informações do conjunto de dados COSMOS QA
```

```
    context = cosmosqa[index]['context']
```

```
    question = cosmosqa[index]['question']
```

```
    answers = [cosmosqa[index][f'answer{i}'] for i in range(4)]
```

```
    label = cosmosqa[index]['label']
```

```
    # Cria a pergunta formatada
```

```
    pergunta = f"{question}\n"
```

```
    for i, answer in enumerate(answers):
```

```
        pergunta += f"{chr(65 + i)} {answer}\n"
```

```
    # Cria o prompt de tradução
```

```
    prompt = f"Você irá receber uma questão de múltipla escolha, em inglês, de modo que a  
você será passado o enunciado, junto com as alternativas, demarcadas por 'A)', 'B)', 'C)' e  
'D)'. Você deve traduzir a pergunta para o português. Ao traduzir a pergunta, mantenha as  
alternativas 'A)', 'B)', 'C)' e 'D)', com suas respectivas traduções, não fuja desse formato.  
Também, fique atento a possíveis abreviações, expressões populares e polissemias, e
```

traduza da melhor forma possível para o português, removendo possíveis ruídos do texto também. A pergunta/questão está marcada com a tag <pergunta>.\n Retorne a tradução da pergunta em um json, de modo que a pergunta traduzida esteja em uma chave 'pergunta'. Segue um exemplo, a ser seguido por você, de retorno: '{'pergunta': 'enunciado da questao', 'A': 'letra A', 'B': 'letra B', 'C': 'letra C', 'D': 'letra D', 'contexto': 'contexto da questao'}'. \n \n <contexto> {context} <contexto> <pergunta> {pergunta} <pergunta>."

try:

```
# Traduz a pergunta
resposta = traduzir_texto(gpt, prompt).replace("\n", ' ')
if resposta and resposta[-1] != "}":
    resposta = resposta[:-1]
resposta_traduzida = eval(resposta)
resposta_traduzida['Label'] = f"{chr(65 + label)}"
print(resposta_traduzida)
dados_cosmosqa_traduzidos.append([resposta_traduzida])
```

except Exception as e:

```
print(f"Erro durante a tradução: {e}")
nao_traduzido = f"{cosmosqa[index]} não traduzido"
dados_cosmosqa_nao_traduzidos.append([nao_traduzido])
continue
```

```
df = pd.DataFrame([item[0] for item in dados_cosmosqa_traduzidos])
```

```
df
```

```
df.to_csv(r"C:\Users\heloi\Downloads\cosmosteste.csv", index=False)
```

```
df2 = pd.DataFrame({'index': dados_cosmosqa_nao_traduzidos})
```

```
df2
```

```
df2.to_csv(r"C:\Users\helo\Downloads\numeros_errosteste.csv",index=False)
```

```
cosmosqa = load_dataset("cosmos_qa")["validation"]
```

```
dados_cosmosqa_traduzidos = []
```

```
dados_cosmosqa_nao_traduzidos = []
```

```
for index in tqdm(range(0, len(cosmosqa))):
```

```
    # Extrai informações do conjunto de dados COSMOS QA
```

```
    context = cosmosqa[index]['context']
```

```
    question = cosmosqa[index]['question']
```

```
    answers = [cosmosqa[index][f'answer{i}'] for i in range(4)]
```

```
    label = cosmosqa[index]['label']
```

```
    # Cria a pergunta formatada
```

```
    pergunta = f"{question}\n"
```

```
    for i, answer in enumerate(answers):
```

```
        pergunta += f"{chr(65 + i)} {answer}\n"
```

```
    # Cria o prompt de tradução
```

```
    prompt = f"Você irá receber uma questão de múltipla escolha, em inglês, de modo que a  
você será passado o enunciado, junto com as alternativas, demarcadas por 'A)', 'B)', 'C)' e  
'D)'. Você deve traduzir a pergunta para o português. Ao traduzir a pergunta, mantenha as  
alternativas 'A)', 'B)', 'C)' e 'D)', com suas respectivas traduções, não fuja desse formato.  
Também, fique atento a possíveis abreviações, expressões populares e polissemias, e  
traduza da melhor forma possível para o português, removendo possíveis ruídos do texto  
também. A pergunta/questão está marcada com a tag <pergunta>.\n Retorne a tradução da  
pergunta em um json, de modo que a pergunta traduzida esteja em uma chave 'pergunta'.  
Segue um exemplo, a ser seguido por você, de retorno: '{{'pergunta': 'enunciado da questao',  
'A': 'letra A', 'B': 'letra B', 'C': 'letra C', 'D': 'letra D', 'contexto': 'contexto da questao'}}'. \n \n  
<contexto> {context} <contexto> <pergunta> {pergunta} <pergunta>."
```

```
try:
    # Traduz a pergunta
    resposta = traduzir_texto(gpt, prompt).replace("\n", ' ')
    if resposta and resposta[-1] != "}":
        resposta = resposta[:-1]
    resposta_traduzida = eval(resposta)
    resposta_traduzida['Label'] = f"{chr(65 + label)}"
    print(resposta_traduzida)
    dados_cosmosqa_traduzidos.append([resposta_traduzida])

except Exception as e:
    print(f"Erro durante a tradução: {e}")
    nao_traduzido = f"{cosmosqa[index]} não traduzido"
    dados_cosmosqa_nao_traduzidos.append([nao_traduzido])
    continue
```

## Termo de Aceite de Entrega 7

### 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:** 7 de dez. de 2023

**Participantes da Entrega** [matriculados em Residência em IA]:

Heloisy Pereira Rodrigues

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

Continuando na missão de me tornar especialista em Large Language Models, no decorrer dessa semana eu finalizei a tradução dos dados “Cosmos QA”, resolvendo os gargalos que ficaram anteriormente. Além disso, escolhi a arquitetura que realizarei o Fine Tuning, Zephyr 7b beta, assim como a lista dos frameworks que serão utilizados para a realização dessa tarefa.

O dataset upado no Hugging Face está aqui: [Cosmos QA Português](#)

Todo o processo está relatado no documento, assim como as dificuldades encontradas: [Entrega 07/12/2023](#)

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

Criação do pipeline de fine tuning e realização do fine tuning do Zephyr 7b beta nos dados Cosmos QA Português.

**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! ▾

## Detalhamento do processo - Semana 7

[O documento referente a seguir pode ser acessado em:

<https://docs.google.com/document/d/1U6tjzKbnQCMuY34sfE4gkDNQwmohcDSDMravqEHwak/edit?usp=sharing>]

## Tradução do dataset

Na semana passada havia iniciado a tradução do dataset [Cosmos QA](#), que se concluiu no sábado, porém apresentou um erro de sincronização da coluna 'id' dos DataFrames. Logo tive que refazer o código, e ajustei em forma de threads para pegar 2 threads e concluir mais rápido a requisição e tradução via GPT. Diminuí bastante o tempo dessa forma, mas para minha surpresa ao ver os DataFrames traduzidos, houve casos de duplicação, creio que estavam compartilhando algum processo/variável que posso não ter separado corretamente via thread. Novamente, coube a mim colocar uma nova tradução para rodar, com 1 thread só. Aproveitando as traduções já feitas, salvei o que já tinha produzido, excluí as duplicatas (não deixei nem a primeira incidência, pois não poderia afirmar com certeza que ela era a verdadeira) e coloquei pra rodar novamente. Agora sim, a tradução foi realizada corretamente, novamente, demorou cerca de 3 dias mas deu certo. Relatei que havia alguns gargalos na tradução com o GPT na hora da devolução do formato da resposta pra mim, esses gargalos eu tive que resolver na mão adicionando no DataFrame traduzido manualmente, esses gargalos foram basicamente dois:

- Dava erro no formato dicionário devido às palavras do inglês com “s”, ele teve dificuldade em organizar as aspas.
- Quando havia alguma fala dentro da questão/contexto/alternativa e ele precisava usar aspas, novamente obteve dificuldade.

Assim, printei os dicionários com erro e fui ajustando as aspas e adicionando ao DataFrame traduzido. Com a tradução finalizada, upei e disponibilizei o dataset Cosmos QA Português no meu Hugging Face no seguinte link: [Cosmos QA Português](#).

O novo código para tradução upei aqui: [Código para tradução 2](#).

## Arquitetura

Buscando entender qual arquitetura usar para realizar meu fine tuning, começo pela listagem de modelos Open Source presentes no [Github](#) do “A Survey of Large Language Models” [1]. Dos modelos disponíveis citados, me interessei no Mistral, pois no decorrer dos meus estudos, lembrei-me de ver alguns resultados bons, porém ele é para a língua inglesa. Sabemos que alguns modelos de linguagem performam bem, mesmo sem ter visto dados de uma língua no treinamento, mas não quero fazer um fine tuning assim, sem dados em português no treinamento do modelo. Assim, parti para investigar um modelo que é um fine tuning do Mistral, o recentemente lançado [Zephyr](#), que é um modelo de linguagem que não teve apenas inglês em seu treinamento e sim um conjunto de dados em inglês e outras

línguas (incluindo português). Além disso, seu desempenho em alguns testes foram satisfatórios e até melhores que modelos como o Llama2 [2]. Portanto fixei o modelo a ser realizado o fine tuning: Zephyr (beta).

Falando um pouco mais sobre o Zephyr, temos os seguintes pontos:

- fine tuning do Mistral 7B com DPO (Direct Preference Optimization)
- seus dados foram uma combinação de conjuntos de dados sintéticos disponíveis publicamente
- possui 7B de parâmetros
- não foi treinado com RLHF e preferências humanas, segurança, etc, logo, se instruído a dar respostas problemáticas, o modelo as dará.
- é uma série de modelos, começa com o alfa, depois tem o beta

## Frameworks

Para captar os (meus) dados e o LLM usarei o Hugging Face com a biblioteca [transformers](#), dado que é uma biblioteca comum na área (possuindo vários modelos e datasets) e de fácil manipulação.

Como não temos um alto poder computacional, utilizar quantização é uma opção viável para reduzir o consumo de memória, além do fato que o fine tuning total do modelo é extremamente custoso e opta-se por usar Parameter-Efficient Fine-Tuning ([PEFT](#)), que permite adaptar o modelo sem ajustar todos os parâmetros dele. Para a quantização, no meu palpite inicial de pipeline, de acordo com um tutorial encontrado [4][5] utilizarei tanto as bibliotecas [bitsandbytes](#) quanto [gptq](#).

Agora pensando quanto ao treinamento em si, tenho a opção de usar tanto o Trainer da biblioteca transformers quanto uma nova biblioteca chamada [trl](#) que possui o SFTTrainer, que é uma versão light do Trainer para realização de fine tuning de modelos de linguagem em dataset customizado.

## Referências

[1] ZHAO, W. et al. A Survey of Large Language Models. [s.l: s.n.]. Disponível em: <<https://arxiv.org/pdf/2303.18223.pdf>>.

[2] TUNSTALL, L. et al. ZEPHYR: DIRECT DISTILLATION OF LM ALIGNMENT 1 INTRODUCTION. [s.l: s.n.]. Disponível em: <<https://arxiv.org/pdf/2310.16944.pdf>>.

[3] RAFAILOV, R. et al. Direct Preference Optimization: Your Language Model is Secretly a Reward Model. [s.l: s.n.]. Disponível em: <<https://arxiv.org/pdf/2305.18290.pdf>>.

[4] PAZUZZU. In-depth LLM finetuning guide: Properly combining LoRA and GPTQ quantization to efficiently.... Disponível em:  
<<https://medium.com/@pazuzzu/in-depth-llm-fine-tuning-guide-efficiently-fine-tune-and-use-zephyr-7b-beta-assistant-using-lora-e23d8151e067>>.

[5] NAYAK, P. Finetuning using Zephyr 7B Quantized model on a custom task of customer support chatbot. Disponível em:  
<<https://medium.aiplanet.com/finetuning-using-zephyr-7b-quantized-model-on-a-custom-task-of-customer-support-chatbot-7f4fff56059d>>.

## Notebook de tradução do Cosmos QA ajustado

[O documento referente a seguir pode ser acessado em:

[https://colab.research.google.com/drive/1e3msYJ7ij3-JsRblkSfTiZd425bhre\\_V?usp=sharing](https://colab.research.google.com/drive/1e3msYJ7ij3-JsRblkSfTiZd425bhre_V?usp=sharing)]

```
import os
import time

import openai

class ChatGPT:
    def __init__(self, api_key_path: str):
        openai.api_key_path = api_key_path
        self.messages = []

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

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

    def add_context(self, context: str):
        context = {'role': 'system', 'content': context}
        self.messages = [context] + self.messages

    def get_completion(self, model="gpt-3.5-turbo", temperature=0):
        retries = 0
        max_retries = 3

        while retries < max_retries:
            try:
                response = openai.ChatCompletion.create(
```

```
        model=model,
        messages=self.messages,
        temperature=temperature,
    )
    return response.choices[0].message["content"], response
except Exception as e:
    model = "gpt-3.5-turbo-1106"
    retries += 1
    time.sleep(2)

from tqdm import tqdm
import json
from datasets import load_dataset
import pandas as pd

from datasets import load_dataset

cosmosqa = load_dataset("cosmos_qa")["train"]

gpt = ChatGPT(r"C:\Users\heloi\Downloads\key.txt")

dados_cosmosqa_traduzidos = []
dados_cosmosqa_nao_traduzidos = []

def traduzir_texto(chat_gpt, prompt):

    chat_gpt.add_message(role="user", prompt=prompt)

    # fazendo inferencia
    resposta, _ = chat_gpt.get_completion()
```

```
chat_gpt.reset_messages(maintain_context=False)
```

```
return resposta
```

Refazendo treino sem duplicatas

```
result_df = pd.read_csv("treino_semdupli.csv")
```

```
import threading
```

```
from tqdm import tqdm
```

```
# Função para traduzir uma pergunta
```

```
def traduzir_pergunta(index, cosmosqa, gpt, dados_cosmosqa_traduzidos,  
dados_cosmosqa_nao_traduzidos, result_df):
```

```
    context = cosmosqa[index]['context']
```

```
    question = cosmosqa[index]['question']
```

```
    answers = [cosmosqa[index][f'answer{i}'] for i in range(4)]
```

```
    label = cosmosqa[index]['label']
```

```
    id = cosmosqa[index]['id']
```

```
# Verifica se 'id' já está no DataFrame
```

```
if id in result_df['id'].values:
```

```
    return
```

```
pergunta = f"{question}\n"
```

```
for i, answer in enumerate(answers):
```

```
    pergunta += f"{chr(65 + i)} {answer}\n"
```

prompt = f"Você irá receber uma questão de múltipla escolha, em inglês, de modo que a você será passado o enunciado, junto com as alternativas, demarcadas por 'A)', 'B)', 'C)' e 'D)'. Você deve traduzir a pergunta para o português. Ao traduzir a pergunta, mantenha as

alternativas 'A)', 'B)', 'C)' e 'D)', com suas respectivas traduções, não fuja desse formato. Também, fique atento a possíveis abreviações, expressões populares e polissemias, e traduza da melhor forma possível para o português, removendo possíveis ruídos do texto também. A pergunta/questão está marcada com a tag <pergunta>.\n Retorne a tradução da pergunta em um json, de modo que a pergunta traduzida esteja em uma chave 'pergunta'. Segue um exemplo, a ser seguido por você, de retorno: '{'pergunta': 'enunciado da questao', 'A': 'letra A', 'B': 'letra B', 'C': 'letra C', 'D': 'letra D', 'contexto': 'contexto da questao'}'. \n \n <contexto> {context} <contexto> <pergunta> {pergunta} <pergunta>. Não se esqueça de traduzir o contexto, ele é muito importante."

try:

```
resposta = traduzir_texto(gpt, prompt).replace('\n', ' ')
if resposta and resposta[-1] != "}":
    resposta = resposta[:-1]
resposta_traduzida = eval(resposta)
resposta_traduzida['Label'] = f"{chr(65 + label)}"
resposta_traduzida['id'] = id

# Adiciona ao DataFrame
dados_cosmosqa_traduzidos.append(resposta_traduzida)
print(resposta_traduzida)
except Exception as e:
    print(f"Erro durante a tradução: {e}")
    nao_traduzido = id
    dados_cosmosqa_ao_nao_traduzidos.append(nao_traduzido)
```

# Lista para armazenar os threads

```
threads = []
```

# Número desejado de threads (neste caso, 2)

```
num_threads = 1
```

```
# Iteração sobre os índices usando threads
for index in tqdm(range(0, len(cosmosqa))):
    # Cria um thread para cada tradução
    thread = threading.Thread(target=traduzir_pergunta, args=(index, cosmosqa, gpt,
dados_cosmosqa_traduzidos, dados_cosmosqa_ao_traduzidos, result_df))
    # Inicia o thread
    thread.start()
    # Adiciona o thread à lista
    threads.append(thread)

# Aguarda até que tenhamos o número desejado de threads em execução
if len(threads) == num_threads:
    # Aguarda até que todas as threads concluam antes de continuar
    for t in threads:
        t.join()
    # Limpa a lista de threads para o próximo conjunto
    threads = []

# Aguarda as threads restantes concluírem
for thread in threads:
    thread.join()
```



## APÊNDICE 5

## Termo de Aceite de Entrega 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]:

Heloisy Pereira Rodrigues

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

Na saga da especialista em LLM, para esta semana o foco foi realizar o fine tuning do Zephyr no dataset Cosmos QA Português, então criei um pipeline para essa realização e o segui para conseguir fazer o fine tuning do modelo. Para fazer esse fine tuning, tive que adaptar um pouco os dados Hugging Face (fiz um script para isso), e os subi num repositório privado no Hugging Face para ficar mais fácil o acesso. Partindo dos dados prontos, coloquei o script para rodar e realizar o fine tuning, após isso subi o [modelo “fine tunado”](#) no Hugging Face. Fiz duas abordagens de fine tuning, que estão detalhadas no documento, assim como o que fiz para analisar perante o Cosmos QA e os scripts usados: [Stage 8 Heloisy](#).

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

Para a próxima semana quero realizar o treino com mais épocas, ajustar alguns hiperparâmetros e ler o paper do Zephyr.

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

### ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: Go! ▾

LUANA GUEDES BARROS MARTINS: Go! ▾

## Detalhamento do processo - Semana 8

## Stage 8

### Processo

Na intenção de criar um pipeline para o fine tuning do Zephyr 7b beta, fui pesquisando em diversos códigos do github, medium, hugging face, um esquema já encaminhado, para facilitar o processo. Diante disso, captando um pouquinho de cada código, olhando as documentações e adaptando para os meus dados, fui montando o pipeline.

Minha intenção inicial era usar a DGX para fazer o fine tuning, logo fiz o que estava ao meu alcance, pedi para o meu colega João Gabriel agendar alguma GPU e pedi auxílio para o Luiz Guilherme com um tutorial de como usá-la, afinal nunca havia mexido com ela. Luiz me indicou pedir a criação de um usuário para acessá-la, então pedi ao Anderson. Como ele não me respondeu, fui pensar outras estratégias para realizar um fine tuning.

Lembrei-me de um fine tuning do Zephyr feito no colab que encontrei na semana passada. Para ser feito no Colab, era necessário já usar um modelo do Zephyr previamente quantizado, senão não rodaria no Colab (até fiz esse teste, mas só apresentou erro de memória, não contornável nem com o Colab Pro, mudança de alguns parâmetros etc) e no pipeline encontrado ele usou o [Zephyr-7B-alpha-GPTQ](#). Apesar de querer usar o beta, mantive para ver se conseguiria rodar. Assim, segui os mesmos passos e consegui rodar [Fine tuning Zephyr 7b alpha GPTQ](#) no Colab. O treinamento foi bem sucedido, mas não cheguei a completar uma época (pois os recursos foram limitados - antes não havia assinado o Colab Pro), porém a inferência estava demorando muito, cerca de 80 segundos para cada amostra. Na inferência, das poucas amostras que deixei, percebi que o modelo estava com dificuldade em parar de gerar resposta, ele gerava a resposta que eu desejava e depois aumentava tendo um pouco de alucinação. Fui pesquisar o que poderia estar acontecendo e achei uma [issue](#) no Hugging Face relatando o mesmo problema e a solução que a pessoa encontrou, logo salvei para aplicar no próximo teste. Finalizando esse teste via colab, upei o modelo [heloisy/zephyr-support-qa](#) no Hugging Face também e vi em uma análise superficial que os resultados estavam até então satisfatórios, comparando a resposta gerada com a resposta verdadeira do dataset, porém numa amostra bem pequena o que não poderia ser considerado e dado o tempo que estava consumindo para inferir, resolvi procurar outro caminho.

Pensando onde rodar em uma GPU com melhor memória, até procurei algumas para alugar na internet, mas no projeto usamos uma 3090 e como não estávamos usando ela nesses dias, resolvi usá-la. Criei um pipeline e coloquei para rodar durante a noite, para minha surpresa... de madrugada acabou a energia. Já havia rodado quase 15h, porém há males

que vem para bens, a loss estava começando bem alta e fui revisar o código, alterei alguns parâmetros e coloquei para rodar novamente. Assim, fiz o fine tuning de acordo com o pipeline, que detalho um pouco mais a posteriori e subi o modelo no Hugging Face, que pode ser acessado pelo link: [heloisy/zephyr-7b-cosmosqa-ptbr text](#). Uma observação importante é que tive que modificar o dataset [CosmosQA Português](#) para colocar novas colunas que me permitissem colocar os dados no formato esperado para o Fine Tuning, esse código pode ser visualizado: [Ajustando Cosmos QA Português text](#).

## Pipeline - usado na 3090

Para a criação do código do pipeline de realização de Fine Tuning, um ponto crucial é decidir a abordagem que será feita, que no meu caso é chamado Supervised Fine Tuning (SFT), ou seja, dou um conjunto de dados com rótulos explícitos, geralmente para uma task mais particular, que no meu caso é Multiple Choice. Vale ressaltar que a biblioteca TRL ajuda a criar o SFT e treinar o modelo com poucas linhas de código por meio de uma API de fácil uso.

O primeiro passo é quantizar o modelo e essa semana estudei um pouco mais sobre quantização e decidi abordar, de antemão, a biblioteca bitsandbytes, isso porque achei a opção mais fácil para quantizar um modelo para 8/4 bits. É necessário fazer quantização para reduzir o armazenamento e uso de memória pelo modelo, pois com a quantização, é possível representar dados com menos informações (menos bits), mas sem perder (muito) a precisão. Acaba que o tempo de inferência também fica menor, pois realiza cálculos mais rápidos com menos bits. Eu quis ir para a quantização de 4 bits pois não sabia se a de 8 bits rodaria na 3090, acabou que depois lendo mais a documentação da biblioteca, vi que (teoricamente, pois não testei) a linha RTX 30xx consegue a quantização de 8 bits. Uma observação importante é que quando quantizado um modelo para 4 bits não é possível enviar os pesos quantizados para o hub do hugging face.

Depois da quantização do modelo feita, precisamos usar os adapters da biblioteca Parameter-Efficient Fine-Tuning (PEFT), que possuem métodos como: LoRA (que utilizei), PrefixTuning etc. Usa-se a PEFT também para o fine tuning ser mais eficiente em níveis de memória e tempo computacional, o LoRA é utilizado pois ele modifica apenas uma fração do modelo pré-treinado e mantém a maioria dos outros parâmetros congelados. Isso ocorre pois ele adiciona uma quantidade de parâmetros treináveis, os chamados adaptadores, para cada camada do LLM, congelando os parâmetros originais do modelo, e no fine tuning ele atualiza os pesos apenas do adaptador. Combinando o LoRA com a Quantização temos o QLoRA. LoraConfig permite controlar como o LoRA é aplicado ao modelo base através de parâmetros.

Seguindo, passamos alguns parâmetros de configuração para o treinamento e junta-se tudo no SFTTrainer. Faço o treinamento e subo para o Hugging Face “fine tunado”. O código final coloquei aqui: [Fine tuning Zephyr beta 3090](#).

## Resultados

Após os fine tuning realizados, fiz a inferência do “Validation” do CosmosQA Português, a parte “Teste” vou inferir depois, para submeter na plataforma deles, que possui um [leaderboard](#), mas pra isso tenho que enviar o código devidamente explicado, e ainda preciso organizar isso para mandar para eles. Quanto ao resultado do fine tuning do Zephyr alpha GPTQ feito no Colab, tive que fazer por amostra de 10%, visto que estava demorando para rodar as inferências, mas para essa amostra apresentou 72,4% de acertos. Já no fine tuning do Zephyr beta, rodei todo o dataset de validação e apresentou 79,73% de acertos. O código de visualização de como foi feito esse cálculo está disponível aqui: [Resultados - Acurácia no CosmosQA Validation](#).

## Referências

Quantization. Disponível em: <<https://huggingface.co/docs/transformers/quantization>>.

LoRA. Disponível em: <[https://huggingface.co/docs/peft/conceptual\\_guides/lora](https://huggingface.co/docs/peft/conceptual_guides/lora)>.

SANTOS, O. QLoRa: A Great Method for Efficiently Fine-Tuning Large Language Models (LLMs). Disponível em:

<<https://medium.com/@santosomar/qlora-a-great-method-for-efficiently-fine-tuning-large-language-models-llms-1f332cf37e96>>.

NAYAK, P. Finetuning using Zephyr 7B Quantized model on a custom task of customer support chatbot. Disponível em:

<<https://medium.aiplanet.com/finetuning-using-zephyr-7b-quantized-model-on-a-custom-task-of-customer-support-chatbot-7f4fff56059d>>.

PAZUZZU. In-depth LLM finetuning guide: Properly combining LoRA and GPTQ quantization to efficiently.... Disponível em:

<<https://medium.com/@pazuzzu/in-depth-llm-fine-tuning-guide-efficiently-fine-tune-and-use-zephyr-7b-beta-assistant-using-lora-e23d8151e067>>.

PEFT. Disponível em: <<https://huggingface.co/docs/peft/index>>.

## Notebook de Fine-Tuning do Zephyr 7b alpha no Colab

[O documento referente a seguir pode ser acessado em:

[https://colab.research.google.com/drive/1D6Df1X7-0BOMilRYa0xwxCrrEMNx\\_7Em?usp=sharing](https://colab.research.google.com/drive/1D6Df1X7-0BOMilRYa0xwxCrrEMNx_7Em?usp=sharing)]

---

```
!pip install -qU transformers datasets trl peft accelerate bitsandbytes auto-gptq optimum
```

```
from huggingface_hub import login  
login(token="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")
```

```
import torch  
from datasets import load_dataset, Dataset  
from peft import LoraConfig, prepare_model_for_kbit_training, get_peft_model  
from transformers import AutoModelForCausalLM, AutoTokenizer, GPTQConfig,  
TrainingArguments  
from trl import SFTTrainer  
import pandas as pd
```

```
MODEL_ID = "TheBloke/zephyr-7B-alpha-GPTQ"  
DATASET_ID = "heloisy/cosmos_qa_ptbr"  
CONTEXT_FIELD = ""  
INSTRUCTION_FIELD = "instruction"  
TARGET_FIELD = "response"  
BITS = 4  
DISABLE_EXLLAMA = True  
DEVICE_MAP = "auto"  
USE_CACHE = False  
LORA_R = 16  
LORA_ALPHA = 16  
LORA_DROPOUT = 0.05  
BIAS = "none"  
TARGET_MODULES = ["q_proj", "v_proj"]  
TASK_TYPE = "CAUSAL_LM"  
OUTPUT_DIR = "zephyr-support-qa"  
BATCH_SIZE = 8  
GRAD_ACCUMULATION_STEPS = 1  
OPTIMIZER = "paged_adamw_32bit"  
LR = 2e-4  
LR_SCHEDULER = "cosine"  
LOGGING_STEPS = 50  
SAVE_STRATEGY = "epoch"
```

```
NUM_TRAIN_EPOCHS = 2
MAX_STEPS = 1000
FP16 = True
PUSH_TO_HUB = True
DATASET_TEXT_FIELD = "text"
MAX_SEQ_LENGTH = 1024
PACKING = False
```

```
data = load_dataset(DATASET_ID, split='train')
data
```

```
df = data.to_pandas()
df.head()
```

```
def process_data_sample_treino1(example):
```

```
    fixed_text = "<|system|>\nVocê é um monitor que ajuda a responder aos usuários as  
    respostas corretas das questões, como um gabarito. Sempre ajude o usuário respondendo a  
    alternativa correta.\n\n <|user|>\n"
```

```
    alta = 'Alternativa A' + example['A']
```

```
    altb = 'Alternativa B' + example['B']
```

```
    altc = 'Alternativa C' + example['C']
```

```
    altd = 'Alternativa D' + example['D']
```

```
    processed_example = fixed_text + "\n".join([example['contexto'], example['pergunta'], alta,  
    altb, altc, altd])
```

```
    processed_label = "\n<|assistant|>\n" + 'Alternativa ' + example['Label']
```

```
    return processed_example, processed_label
```

```
df[['INSTRUCT_FIELD', 'TARGET_FIELD']] = df.apply(lambda x:  
pd.Series(process_data_sample_treino1(x)), axis=1)
```

```
df.head()
```

```
def process_data_sample_treino2(example):
```

```
    processed_example = example['INSTRUCT_FIELD'] + example['TARGET_FIELD']
```

```
    return processed_example
```

```
df[DATASET_TEXT_FIELD] = df.apply(lambda x: process_data_sample_treino2(x), axis=1)
df.head()
```

```
df[DATASET_TEXT_FIELD][0]
```

```
processed_data = Dataset.from_pandas(df[[DATASET_TEXT_FIELD]])
processed_data
```

```
tokenizer = AutoTokenizer.from_pretrained("TheBloke/zephyr-7B-alpha-GPTQ")
tokenizer.pad_token = tokenizer.eos_token
```

```
bnb_config = GPTQConfig(bits=4,
                        disable_exllama=True,
                        device_map="auto",
                        use_cache=False,
                        lora_r=16,
                        lora_alpha=16,
                        tokenizer=tokenizer
                        )
```

```
#
model = AutoModelForCausalLM.from_pretrained("TheBloke/zephyr-7B-alpha-GPTQ",
                                             quantization_config=bnb_config,
                                             device_map="auto",
                                             use_cache=False,
                                             )
```

```
print("\n=====
=====\n")
print("\t\t\tDOWNLOADED MODEL")
print(model)
print("\n=====
=====\n")
```

```
model.config.use_cache=False
model.config.pretraining_tp=1
model.gradient_checkpointing_enable()
```

```
model = prepare_model_for_kbit_training(model)

print("\n=====
====\n")
print("\t\t\t\tMODEL CONFIG UPDATED")
print("\n=====
====\n")

peft_config = LoraConfig(
    r=LORA_R,
    lora_alpha=LORA_ALPHA,
    lora_dropout=LORA_DROPOUT,
    bias=BIAS,
    task_type=TASK_TYPE,
    target_modules=TARGET_MODULES
)

model = get_peft_model(model, peft_config)
print("\n=====
====\n")
print("\t\t\t\tPREPARED MODEL FOR FINETUNING")
print(model)
print("\n=====
====\n")

training_arguments = TrainingArguments(
    output_dir=OUTPUT_DIR,
    per_device_train_batch_size=BATCH_SIZE,
    gradient_accumulation_steps=GRAD_ACCUMULATION_STEPS,
    optim=OPTIMIZER,
    learning_rate=LR,
    lr_scheduler_type=LR_SCHEDULER,
    save_strategy=SAVE_STRATEGY,
    logging_steps=LOGGING_STEPS,
    num_train_epochs=NUM_TRAIN_EPOCHS,
    max_steps=MAX_STEPS,
    fp16=FP16,
    push_to_hub=PUSH_TO_HUB)
```



```
    altb = 'Alternativa B)' + example['B']
    altc = 'Alternativa C)' + example['C']
    altd = 'Alternativa D)' + example['D']
    processed_example = fixed_text + "\n".join([example['contexto'], example['pergunta'], alta,
    altb, altc, altd])
    return processed_example
```

```
dfteste[['INSTRUCT_FIELD']] = dfteste.apply(lambda x:
pd.Series(process_data_sample_teste(x), axis=1)
```

```
from peft import AutoPeftModelForCausalLM
from transformers import GenerationConfig
from transformers import AutoTokenizer
import torch
```

```
# Criar uma lista de strings contendo os exemplos de teste pré-processados
test_examples = dfteste['INSTRUCT_FIELD'].tolist()
test_examples_dict = dict(zip(dfteste['id'], test_examples))
```

```
tokenizer = AutoTokenizer.from_pretrained("heloisy/zephyr-support-qa")
```

```
model = AutoPeftModelForCausalLM.from_pretrained(
    "heloisy/zephyr-support-qa",
    low_cpu_mem_usage=True,
    return_dict=True,
    torch_dtype=torch.float16,
    device_map="cuda")
```

```
generation_config = GenerationConfig(
    do_sample=True,
    top_k=1,
    temperature=0.1,
    max_new_tokens=256,
    pad_token_id=tokenizer.eos_token_id
```

```
)

import time
from tqdm import tqdm
# Inicialize uma lista para armazenar as saídas geradas
generated_outputs = []

# Medir o tempo de geração
st_time = time.time()

# Iterar sobre os exemplos e gerar as saídas
for example_id, example_text in tqdm(test_examples_dict.items()):
    st_time_sample = time.time()
    inputs = tokenizer(example_text, return_tensors="pt").to("cuda")
    outputs = model.generate(**inputs, generation_config=generation_config)
    decoded_output = tokenizer.decode(outputs[0], skip_special_tokens=True)
    print(f"Tempo sample: {time.time() - st_time_sample} segundos")
    print(f"ID: {example_id}\nGenerated Output: {decoded_output}\n")
    generated_outputs.append((example_id, decoded_output))

df_results = pd.DataFrame(generated_outputs, columns=['ID', 'Generated Output'])

# Salvar o DataFrame em um arquivo CSV
df_results.to_csv('generated_outputs.csv', index=False)

# Imprimir o tempo total de geração para o conjunto de teste
print(f"Tempo total de geração para o conjunto de teste: {time.time() - st_time} segundos")

Tive que interromper pois demoraria demais rodar todos os exemplos, deixei 10% para ter
uma ideia do comportamento/acertos

df_results = pd.DataFrame(generated_outputs, columns=['ID', 'Generated Output'])

# Salvar o DataFrame em um arquivo CSV
df_results.to_csv('generated_outputs.csv', index=False)

df_results
```

## Notebook Ajustando Cosmos QA Português para Fine-Tuning

[O documento referente a seguir pode ser acessado em:

[https://colab.research.google.com/drive/1x5zD66g1HZA1RPFaXFSlyWtQmHUS\\_ikU?usp=sharing](https://colab.research.google.com/drive/1x5zD66g1HZA1RPFaXFSlyWtQmHUS_ikU?usp=sharing)]

Aqui está um exemplo do que foi feito para o "train", mas foi feito para o restante do dataset, só substituir o split.

```
from datasets import load_dataset, Dataset
import pandas as pd

data = load_dataset(DATASET_ID, split='train')
data
```

```
df = data.to_pandas()
df.head()
```

```
def process_data_sample_treino1(example):
```

```
    fixed_text = "<|system|>\nVocê é um monitor que ajuda a responder aos usuários as  
respostas corretas das questões, como um gabarito. Sempre ajude o usuário respondendo a  
alternativa correta.\n\n <|user|>\n"
```

```
    alta = 'Alternativa A)' + example['A']
```

```
    altb = 'Alternativa B)' + example['B']
```

```
    altc = 'Alternativa C)' + example['C']
```

```
    altd = 'Alternativa D)' + example['D']
```

```
    processed_example = fixed_text + "\n".join([example['contexto'], example['pergunta'], alta,  
altb, altc, altd])
```

```
    processed_label = "\n<|assistant|>\n" + 'Alternativa ' + example['Label']
```

```
    return processed_example, processed_label
```

```
df[['INSTRUCT_FIELD', 'TARGET_FIELD']] = df.apply(lambda x:  
pd.Series(process_data_sample_treino1(x)), axis=1)
```

```
df.head()
```

```
def process_data_sample_treino2(example):
```

```
    processed_example = example['INSTRUCT_FIELD'] + example['TARGET_FIELD']
```

```
    return processed_example
```

```
df[DATASET_TEXT_FIELD] = df.apply(lambda x: process_data_sample_treino2(x), axis=1)
```

```
df.head()
```

## Código de Fine-tuning Zephyr 7b beta utilizado na RTX 3090 v1

[O documento referente a seguir pode ser acessado em:

[https://colab.research.google.com/drive/1DNkQwh-qX1gLFqe6kTbXIX\\_6ZV31BDcZ?usp=sharing](https://colab.research.google.com/drive/1DNkQwh-qX1gLFqe6kTbXIX_6ZV31BDcZ?usp=sharing)]

```
!pip install -q bitsandbytes datasets accelerate loralib
!pip install -q git+https://github.com/huggingface/transformers.git@main
git+https://github.com/huggingface/peft.git
!pip install optimum auto-gptq
```

```
import torch
from datasets import load_dataset
from peft import LoraConfig, PeftModel, prepare_model_for_kbit_training
from transformers import (
    AutoModelForCausalLM,
    AutoTokenizer,
    BitsAndBytesConfig,
    TrainingArguments,
)
import os
from huggingface_hub import login
```

# Definição de várias constantes e variáveis com informações sobre o projeto, modelo, dataset, etc.

```
PROJECT_NAME = 'zephyr-7b-cosmosqa-ptbr_text'
HUGGING_FACE_REPO_NAME = f'heloisy/{PROJECT_NAME}'
DATASET_ID = "heloisy/cosmos_qa_ptbr_text"
DATASET_TEXT_FIELD = "text"
DEVICE_MAP = "auto"
MODEL_ID = "HuggingFaceH4/zephyr-7b-beta"
OUTPUT_DIR = "./zephyr-7b-cosmosqa-ptbr_text"
TASK_TYPE = "CAUSAL_LM"
LORA_R = 16
LORA_ALPHA = 16
LORA_DROPOUT = 0.05
BATCH_SIZE = 16
NUM_TRAIN_EPOCHS = 2
LOGS_DIR = "./logszephyr2"
MAX_STEPS = 1000
WARMUP = 100
OPTIMIZER = 'paged_adamw_8bit'
GRAD_ACC = 2
LR = 4e-5
MAXLEN = 512
```

```
PUSH_TO_HUB = True

# Definição da função principal
def main():
    # Autenticação no Hugging Face Hub usando um token
    token = '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'
    login(token=token)

    # Verifica se o diretório de saída não existe e, se não existir, cria-o
    if not os.path.exists(OUTPUT_DIR):
        print(f"O diretório {OUTPUT_DIR} não existe. Criando...")
        try:
            os.makedirs(OUTPUT_DIR)
        except Exception as e:
            print(f"Não foi possível criar o diretório {OUTPUT_DIR}. Erro: {e}")
            exit(1)

    # Verifica se o diretório de logs não existe e, se não existir, cria-o
    if not os.path.exists(LOGS_DIR):
        print(f"O diretório {LOGS_DIR} não existe. Criando...")
        try:
            os.makedirs(LOGS_DIR)
        except Exception as e:
            print(f"Não foi possível criar o diretório {LOGS_DIR}. Erro: {e}")
            exit(1)

    # Carrega um conjunto de dados usando o ID especificado
    dataset = load_dataset(DATASET_ID)

    # Cria um tokenizador usando o modelo especificado e faz algumas configurações
    relacionadas ao preenchimento
    tokenizer = AutoTokenizer.from_pretrained(MODEL_ID, use_fast=True)
    tokenizer.pad_token = tokenizer.unk_token
    tokenizer.pad_token_id = tokenizer.unk_token_id
    tokenizer.padding_side = 'right'

    # Configura o tipo de dados de cálculo para ponto flutuante de 16 bits
    compute_dtype = getattr(torch, "float16")

    # Configura a quantização de bits e bytes para o modelo
```

```
bnb_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_compute_dtype=compute_dtype,
    bnb_4bit_use_double_quant=True,
)

# Carrega um modelo de linguagem causal pré-treinado com a configuração de
quantização
model = AutoModelForCausalLM.from_pretrained(
    MODEL_ID, quantization_config=bnb_config, device_map={"":
torch.cuda.current_device()})

# Prepara o modelo para o treinamento com quantização
model = prepare_model_for_kbit_training(model)

# Configura algumas propriedades do modelo
model.config.pad_token_id = tokenizer.pad_token_id
model.config.use_cache = False
model.config.pretraining_tp = 1

# Configura a configuração do modelo Lora para treinamento
peft_config = LoraConfig(
    lora_alpha=LORA_ALPHA,
    lora_dropout=LORA_DROPOUT,
    r=LORA_R,
    bias="none",
    task_type=TASK_TYPE,
    target_modules=["q_proj", "k_proj", "v_proj", "o_proj", "gate_proj", "up_proj"]
)

# Configura as opções de treinamento
training_arguments = TrainingArguments(
    output_dir=OUTPUT_DIR,
    num_train_epochs=NUM_TRAIN_EPOCHS,
    evaluation_strategy="steps",
    do_eval=True,
    per_device_train_batch_size=BATCH_SIZE,
    per_device_eval_batch_size=BATCH_SIZE,
```

```
    gradient_accumulation_steps=GRAD_ACC,
    optim=OPTIMIZER,
    save_steps=100,
    logging_steps=100,
    learning_rate=LR,
    logging_dir=LOGS_DIR,
    max_steps=MAX_STEPS,
    warmup_steps=WARMUP,
    lr_scheduler_type="linear",
)

# Cria um treinador SFTTrainer para treinar o modelo
trainer = SFTTrainer(
    model=model,
    train_dataset=dataset['train'],
    eval_dataset=dataset['validation'],
    peft_config=peft_config,
    dataset_text_field=DATASET_TEXT_FIELD,
    max_seq_length=MAXLEN,
    tokenizer=tokenizer,
    args=training_arguments,
)

# Inicia o treinamento do modelo
trainer.train()

# Envia o modelo treinado para o Hugging Face Hub
trainer.push_to_hub(HUGGING_FACE_REPO_NAME)

# Executa a função principal quando o script é executado diretamente
if __name__ == "__main__":
    main()
```

## Termo de Aceite de Entrega 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]:

Heloisy Pereira Rodrigues

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

Para esta semana, me especializando em Large Language Models, rodei duas versões de fine tuning do Zephyr 7b beta, alterando alguns parâmetros dada a versão da semana passada. As alterações foram documentadas no documento: [Stage 9](#)  
Além disso, li o paper do [Zephyr](#) e adicionei considerações pertinentes ao documento anteriormente citado.

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

Para a próxima semana quero rodar a inferência no modelo Zephyr 7b beta original, ver se realmente compensa tal gasto computacional, e embasar hipóteses de melhoras nos fine tunings (relativamente rápidos) que realizei.

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

A acurácia no dataset de validação do fine tuning v3 terá um atraso de cerca de 1h do horário de entrega, peço desculpas pois fugiu do meu controle visto que estavam utilizando a GPU e só pude usá-la a partir de segunda a noite.

## ACEITE DA ENTREGA:

**CEDRIC LUIZ DE CARVALHO:** Go! ▾

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

## Detalhamento do processo - Semana 9

[O documento referente a seguir pode ser acessado em:

<https://docs.google.com/document/d/1fl0qscfBx9B6lr-L45Gb8FegXFH0hyOJrsWcRI9ekgw/edit?usp=sharing>]

## Stage 9

Para essa semana, meu foco foi rodar novamente o fine tuning do modelo Zephyr 7b beta, alterando alguns parâmetros e por mais épocas, visto que antes tinha sido aproximadamente 1 época apenas. Tive problemas no início pois estavam utilizando a GPU, logo tive que esperar desocupar. Mas na segunda pela noite comecei o treinamento. Assim, rodei 2 versões a mais, sendo elas:

Fine tuning v2:

- alterei o LORA\_R para 32
- alterei o num\_epochs para 3, porém fui desconectada (via ssh da máquina que estava rodando o código) durante o treinamento, não concluindo as 3 épocas, apenas 2 e um pouco a mais.
- alterei o lr\_scheduler\_type para “cosine” visto que ao ler o paper do Zephyr eles treinaram desde o início com ele.
- ao rodar o [script de acurácia](#) na validação do Cosmos QA Português, ele apresentou 81,03% de acertos
- como não foi possível finalizar o treinamento, ele não salvou no Hugging Face, mas está localmente salvo

Fine tuning v3:

- voltei o LORA\_R para 16
- alterei o num\_epochs para 2
- alterei o lr\_scheduler\_type para “cosine” visto que ao ler o paper do Zephyr eles treinaram desde o início com ele
- eu quis mudar apenas o lr\_scheduler\_type e treinar por mais tempo do que a versão 1.
- a inferência dos testes da validação no dataset Cosmos QA Português do modelo gerado está em 75% agora as 10:22 do dia 21/12/2023, faltando cerca de 2h e 30 para finalizar, vou adicionar o resultado aqui assim que terminar.
- ao rodar o script de acurácia na validação do Cosmos QA Português, ele também apresentou 81,03% de acertos, achei suspeito, porém com respostas diferentes da v2, o que voltei a achar normal.
- foi salvo no Hugging Face: [heloisy/zephyr-7b-b-cosmosqa-ptbr-text-v3](https://huggingface.co/heloisy/zephyr-7b-b-cosmosqa-ptbr-text-v3)

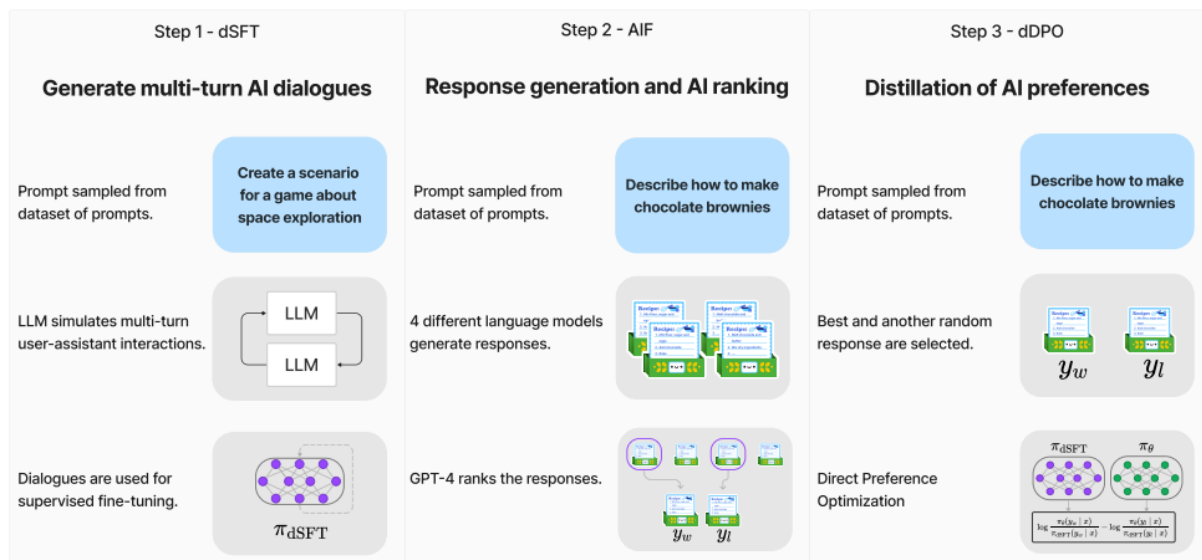
## Leitura do paper Zephyr

O foco do Zephyr é ser um modelo de linguagem menor, se comparado às dezenas e centenas de bilhões de parâmetros de outros modelos, e alinhado às intenções do usuário. Isso porque há vários modelos que são bons em algumas tarefas específicas, benchmarks de tasks etc, mas em contato com o usuário não performam adequadamente, ou seja, não atendem exclusivamente ao comando do usuário, podendo escrever mais do que o pedido, por exemplo. Vale ressaltar que os cientistas fizeram esse alinhamento com preferências humanas sem anotações humanas, mas como fizeram isso? Vou falar o que entendi.

Essa missão tem três fases,

- Fine Tuning Supervisionado Destilado (dSFT): Inicia com um modelo bruto e o treina para responder a prompts do usuário com base em instruções geradas pelo modelo de ensino.
- Feedback de IA através de Preferências (AIF): Utiliza preferências do modelo de ensino sobre as saídas geradas por outros modelos.
- Otimização Direta de Preferência Destilada (dDPO): Refina o modelo dSFT maximizando a probabilidade de escolher a resposta preferida sobre as outras.

A imagem no paper nos ajuda a entender um pouco mais:



Essa destilação é baseada no modelo professor e aluno, que ao invés de treinar o aluno com base nos livros, que pode ter falas/conhecimentos desnecessários, ele o treina com base em suas anotações mais pertinentes.

No AI Feedback o Zephyr é um estudante e os usuários são os professores. Cada vez que o estudante (Zephyr) responde a uma pergunta, o professor (usuário) dá um feedback sobre a resposta, isso é passado depois para ajudar no alinhamento desejado.

Já no dDPO, imagine que você está treinando um cachorro para buscar uma bola. Você tem vários treinadores (os modelos de professores) que são especialistas em treinar cães. Cada treinador tem uma maneira diferente de ensinar o cão a buscar a bola e dá feedback diferente ao cão. O dDPO é como se o cão (o Zephyr) estivesse aprendendo com o feedback de todos esses treinadores ao mesmo tempo.

Então o Zephyr é capaz de aprender com vários modelos de professores ao mesmo tempo, melhorando sua capacidade de responder e se alinhar com a instrução humana.

Traduzi o paper com o Google Tradutor, está disponível aqui: [Paper Zephyr Traduzido](#)

## Referências

TUNSTALL, L. et al. Zephyr: Direct Distillation of LM Alignment. Disponível em:  
<<https://arxiv.org/abs/2310.16944>>.

Zephyr-7B : HuggingFace's Hyper-Optimized LLM Built on Top of Mistral 7B - Unite.AI.  
Disponível em:

<<https://www.unite.ai/zephyr-7b-huggingfaces-hyper-optimized-llm-built-on-top-of-mistral-7b/>>. Acesso em: 21 dez. 2023.

## APÊNDICE 6

## Termo de Aceite de Entrega 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]:

Heloisy Pereira Rodrigues

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

No último episódio da minha saga de me tornar especialista em Large Language Models fiquei responsável de comprovar e embasar hipóteses de como um fine tuning pode ajudar a melhorar a performance de um modelo de linguagem. Para isso, realizei a inferência do dataset de validação Cosmos QA Português no modelo Zephyr 7b beta (o original, sem fine tunings) e submeti ao mesmo processo de medição da acurácia realizada nos modelos que eu fiz ajuste fino anteriormente. No decorrer do caminho para aferir essa acurácia, tive que anotar os dados da resposta gerada na inferência, dado que não havia um padrão (logo não podia captar por um script), e assim possibilitar este acontecimento. Como esperado, o modelo sem fine tuning apresentou um desempenho abaixo dos modelos com fine tuning, com cerca de 25% a menos de acurácia. Apresentei os possíveis porquês da melhora, como: transferência de aprendizado, uso de Parameter-Efficient Fine-Tuning (PEFT), Low-Rank Adaptation (LoRA), maior capacidade de generalização e adaptação ao vocabulário. A explicação dessas hipóteses de por que alterar o modelo, fazendo o fine tuning, assim como o detalhamento do processo foi relatado no documento: [Stage 10](#).

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

Documentar e apresentar um bom TCC.

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

## ACEITE DA ENTREGA:

**CEDRIC LUIZ DE CARVALHO:** Go! ▾

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

## Detalhamento do processo - Semana 10

[O documento referente a seguir pode ser acessado em:

<https://docs.google.com/document/d/11KCZx4dUpkMC8bayil54ps18DhAhZ8mUQC3IWM3YVkl/edit?usp=sharing>]

## Processo

No decorrer dessa semana, fiquei responsável de inferir no Zephyr 7b beta (o modelo original) o dataset de validação do Cosmos QA Português. Essa escolha foi feita para ter a mesma medida - acurácia - se comparado aos fine tunings realizados. Assim, fiz os passos de sempre. Acessei a GPU RTX 3090 e coloquei o [código de inferência](#) para rodar, após isso salvei os resultados e olhei superficialmente como estavam, a fim de fazer um script para captar as respostas, como foi feito anteriormente nos códigos para avaliação dos fine tuning realizados. Apesar disso, percebi que não havia um padrão nas respostas e logo me veio a triste realidade, era necessário anotar os dados. Então fiz um [script](#) de uma ferramenta que me permitisse anotar os dados da seguinte forma: imprimia a resposta gerada pelo Zephyr e eu escrevia, via teclado, qual a alternativa que ele havia dado como correta. Foi um trabalho manual, realizado por mim, de quase 3000 dados. Durante esse processo de anotação, observei as seguintes coisas:

- O modelo cria novas alternativas nas respostas, não respondendo a pergunta

Generated Output: <|system|>

Você é um monitor que ajuda a responder aos usuários as respostas corretas das questões, como um gabarito. Sempre ajude o usuário respondendo a alternativa correta.

<|user|>

Eu uso muitos chapéus ... As confusões habituais foram ainda mais divertidas por causa da menina de três anos cansada e temperamental, filha da minha nova nora. Ela não estava aceitando essa bobagem e continuava correndo pelo caminho, ou se grudando na perna da nora.

Por que a criança da nora grudou na perna da nora?

Alternativa A)O Velcro é projetado para ser adesivo, a criança está grudada.

Alternativa B)Nenhuma das opções acima.

Alternativa C)A menina das flores está em um casamento, um evento emocionante, a criança está sobrecarregada e procurando a mãe para conforto.

Alternativa D)Crianças temperamentais costumam se divertir se comportando mal.

Alternativa E)A criança está aprendendo a caminhar e está apenas segurando a perna da mãe.

Alternativa F)A criança está apenas segurando a perna da mãe, mas está cansada e está se movendo lentamente.

Alternativa G)A criança está apenas segurando a perna da mãe, mas está tentando escapar.

Alternativa H)A criança está apenas segurando a perna da mãe, mas está tentando escapar, mas está cansada e está se movendo lentamente.

Alternativa I)A criança está apenas segurando a perna da mãe, mas está tentando escapar, mas está sobrecarregada e procurando a mãe para conforto.

Alternativa J)A criança está apenas segurando a perna da mãe, mas está tentando escapar, mas está sobrecarregada e está se movendo lentamente.

Alternativa K)A crianç

- O modelo fornece a resposta gerada sem dizer qual alternativa era, citando assim só o texto

Generated Output: <|system|>

Você é um monitor que ajuda a responder aos usuários as respostas corretas das questões, como um gabarito. Sempre ajude o usuário respondendo a alternativa correta.

<|user|>

Todos eles são muito atraentes e tão legais! E os caras são muito mais fáceis de sair quando você não os conhece muito bem. Quero dizer, eu adoro as meninas quando sou bom amigo delas, mas todo o julgamento e avaliação que acontece ao conhecê-las é tão irritante. Os caras são tão tranquilos.

Por que eu saí com aqueles caras?

Alternativa A) Porque eles eram atraentes e legais

Alternativa B) Porque eles eram bons amigos de meninas que eu conheço

Alternativa C) Nenhuma das opções acima.

Alternativa D) Porque outros caras te avaliam e julgam quando você sai com eles.

<|assistant|>

A alternativa correta é: Porque eles eram atraentes e legais.

Explicação: O usuário explica que todas as meninas são atraentes e legais, e que são mais fáceis de sair quando se não conhece muito bem. Essa descrição se aplica aos caras mencionados, pois o usuário afirma que saiu com eles por essa razão.

- **O modelo explica a alternativa dada como correta**

Generated Output: <|system|>

Você é um monitor que ajuda a responder aos usuários as respostas corretas das questões, como um gabarito. Sempre ajude o usuário respondendo a alternativa correta.

<|user|>

Nada mais sobre isso, eu ainda não ouvi. Não faço ideia de como aconteceu, especialmente se a casa estava realmente vazia como meu primo disse. Meu irmão estava em casa e disse que a casa inteira tremeu e coisas caíram da prateleira do quarto dele. E estamos do outro lado da cidade de tudo isso, então deve ter sido bem grande. Por que eles estão preocupados com isso?

Alternativa A) Eles queriam ser os primeiros com informações sobre o que aconteceu.

Alternativa B) Eles queriam ver o evento acontecendo.

Alternativa C) A casa tremeu e coisas caíram das prateleiras.

Alternativa D) Eles queriam poder contar ao irmão deles que a casa caiu.

<|assistant|>

O monitor deve responder: "Os outros estão preocupados porque a casa do seu irmão tremeu e coisas caíram das prateleiras, o que pode indicar um evento de grande magnitude, provavelmente um terremoto. Eles podem querer ser os primeiros a obter informações sobre o que aconteceu, ou talvez simplesmente queiram ver o evento acontecendo, mas a alternativa mais provável é que eles querem poder contar ao irmão deles que a casa caiu, pois estão preocupados com a segurança dele."

- **O modelo explica todas as alternativas**

Generated Output: <|system|>

Você é um monitor que ajuda a responder aos usuários as respostas corretas das questões, como um gabarito. Sempre ajude o usuário respondendo a alternativa correta.

<|user|>

Taylor estava chateado e eu não sabia para onde ir, então eventualmente foi decidido que eu iria até lá, cheguei até Greenwood e depois desci do ônibus e atravessei a rua para pegar o outro 48. Então cheguei ao ponto do 73 e as

peças gostavam de conversar comigo. Uma senhora idosa muito simpática começou a conversar comigo e sorriu sobre o ônibus. Um senhor idoso parou na minha frente e disse várias coisas, rindo e eu não o entendia.

O que poderia ser diferente se Taylor não estivesse chateado?

Alternativa A)O narrador teria pegado o ônibus

Alternativa B)Nenhuma das opções acima.

Alternativa C)O narrador não teria ido até lá

Alternativa D)O narrador teria ido até lá e pegado o outro 48.

<|assistant|>

Se Taylor não estivesse chateado, o narrador pode ter feito outra coisa ao invés de ir até lá. Então, a alternativa C é incorreta.

Se o narrador não estivesse chateado, ele pode ter decidido ficar em algum lugar ou fazer outra atividade, então a alternativa A é incorreta.

Se o narrador não estivesse chateado, ele pode ter continuado com sua rotina normal e não ter ido até lá, então a alternativa D é incorreta.

Portanto, se Taylor não estivesse chateado, a alternativa B é a correta. Nenhuma das outras opções acima se aplica.

- **O modelo cria uma única alternativa a mais (letra E)**

Generated Output: <|system|>

Você é um monitor que ajuda a responder aos usuários as respostas corretas das questões, como um gabarito.

Sempre ajude o usuário respondendo a alternativa correta.

<|user|>

A sala de estar e o quarto do meu pai não eram apenas adjacentes, mas eram separados apenas por uma cortina para bloquear a luz. Nós, crianças, éramos constantemente advertidos para 'manter o barulho baixo'; mas, pelo que podíamos perceber, a maior parte do barulho vinha da parte de trás do meu pai. Ele soltava gases com frequência e com um som alto e autoritário. O seu peido mais frequente era o que nós, crianças, chamávamos de 'barco a motor'. Qual pode ter sido o motivo do seu pai pedir para vocês manterem o barulho baixo?

Alternativa A)Quase nada impedia o barulho de chegar ao quarto do meu pai.

Alternativa B)Nenhuma das opções acima.

Alternativa C)Nós, crianças, continuamos brincando com o barco a motor à noite.

Alternativa D)Nós soltávamos gases com frequência e com um som alto.

Alternativa E)O barulho vinha da parte de trás do meu pai.

<|assistant|>

A alternativa correta é a opção E, pois o barulho vinha da parte de trás do meu pai. As outras opções não explicam o motivo do pedido de manter o barulho baixo. Alternativa A descreve o fato de que o barulho chegava ao quarto do pai, mas não explica o motivo do pedido. Alternativa B é falsa, pois o texto menciona o barulho. Alternativa C é falsa, pois o texto não menciona nada sobre os outros membros da família brincando com o barco a motor à noite.

Alternativa D é falsa, pois o texto menciona que o pai soltava gases com frequência e com um som alto, mas não explica o motivo do pedido de manter o barulho baixo.

- **O modelo marca como correta a alternativa criada letra E**

Exemplo anterior

- **O modelo cria alternativa nova, na tentativa de encaixar uma resposta, desconsiderando alternativas como “Nenhuma das opções”**

Penúltimo exemplo

## Resultado

Após a anotação desses dados, as anotações foram agrupadas em um csv, para que eu pudesse conferir a acurácia, no mesmo processo feito anteriormente com os testes de fine tuning. Adicionei ela nesse documento: [Resultados - Acurácia no CosmosQA Validation Original](#), que é um compilado de todas as outras já feitas, com o adicional da acurácia gerada agora com o uso do Zephyr 7b beta para a inferência. Agora o resultado da acurácia do modelo original foi 43,18% quando usei as alternativas E/Z, e 58,27% quando exclui as questões que deram como alternativa E/Z. Explicando o que são essas classificações E/Z, basicamente quando o modelo criou alternativa a mais do que era (o dataset contém 4 questões - ABCD) e classificou como a alternativa correta “E” correta, manteve essa classificação, ele fez isso em 481 amostras, cerca de 16% dos dados analisados. Já quando o modelo criou várias alternativas, respondendo ou não a pergunta, classifiquei como “Z”, o que ocorreu em 292 amostras, cerca de 10% dos dados.

A partir daqui já percebemos o quanto o fine tuning é fundamental para que o Large Language Model tenha uma performance melhor, em tarefas determinadas. Isso porque, o desempenho na task (Question Answer) foi consideravelmente menor, as respostas geradas não apresentavam nenhum padrão e o tempo para realização do fine tuning foi consideravelmente rápido, levando menos que 2 dias.

## Por que alterar o modelo foi benéfico?

Fazer fine tuning em modelos de linguagem, é um exemplo clássico da aplicação de transferência de aprendizado. Esse processo começa com um modelo pré-treinado em um vasto corpus de linguagem geral, no meu caso o Zephyr 7b beta, abrangendo uma ampla variedade de tópicos e estilos. O conhecimento assimilado nesta fase é então transferido e adaptado para uma tarefa específica, como QA. [1].

Como utilizei Parameter-Efficient Fine-Tuning (PEFT), que é uma biblioteca que otimiza o processo de fine tuning ao reduzir a carga computacional e de memória necessária [2], consegui realizar o fine tuning de maneira “mais eficiente”, reduzindo o tempo e os recursos necessários. Isso é crucial em large language models como o Zephyr 7b beta, que, sem otimização, exigiriam hardware de alta capacidade, o que é difícil de se conseguir. Ao tornar o fine tuning mais acessível, a biblioteca permite uma experimentação facilitada, contribuindo assim com a busca por configurações de modelo que sejam mais eficazes para a minha tarefa desejada.

Além disso, a adaptação ao vocabulário e estilo linguístico durante o fine tuning é crucial, o que foi bastante notório durante a inferência modelo com fine tuning x modelo sem fine tuning. Esta fase do treinamento permite ao modelo aprender terminologias especializadas, entender melhor as nuances e se adaptar a diferentes formas de formular perguntas e respostas, ou seja, faz sentido o fine tuning colaborar com uma padronização das respostas, diferentemente do modelo original. [3]

Outro ponto importante é a utilização de LoRA (Low-Rank Adaptation), que ao invés de reajustar todos os parâmetros do modelo, LoRA concentra-se em refinar um conjunto selecionado de parâmetros do modelo utilizando técnicas de aproximação de baixa complexidade, otimizando a capacidade de adaptação sem alterar significativamente a estrutura geral do modelo [4]. Isso significa que o modelo retém a maior parte de sua estrutura e conhecimento pré-treinado, enquanto se adapta eficientemente para a tarefa específica.

Como uma das maiores vantagens do fine tuning é a melhoria na capacidade de generalização do modelo [5], treinar no conjunto de dados específico de QA, o modelo ajusta suas representações internas para se alinhar melhor com os tipos de perguntas e respostas encontradas nesse conjunto. Com isso ele fica melhor na generalização e é mais fácil de ele responder novas perguntas do estilo QA que não foram vistas durante o treinamento.

[1] Pan, S. J., & Yang, Q. (2010). A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*.

[2] XU, L. et al. Parameter-Efficient Fine-Tuning Methods for Pretrained Language Models: A Critical Review and Assessment. Disponível em: <<https://arxiv.org/abs/2312.12148>>.

[3] BENGIO, Y. et al. A Neural Probabilistic Language Model. *Journal of Machine Learning Research*, v. 3, p. 1137–1155, 2003.

[4] HU, E. J. et al. LoRA: Low-Rank Adaptation of Large Language Models. *arXiv:2106.09685 [cs]*, 16 out. 2021.

[5] Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.

## Código de Inferência dos dados no Zephyr 7b beta

[O documento referente a seguir pode ser acessado em:

[https://colab.research.google.com/drive/1q\\_EyvBMANrpXMSYcGuZxrNIU\\_4Qa0Gjx?usp=sharing](https://colab.research.google.com/drive/1q_EyvBMANrpXMSYcGuZxrNIU_4Qa0Gjx?usp=sharing)]

```
from datasets import load_dataset
from peft import AutoPeftModelForCausalLM
from transformers import GenerationConfig, AutoModelForCausalLM
from transformers import AutoTokenizer
import torch
from tqdm import tqdm
import pandas as pd
from huggingface_hub import login

def main():
    token = 'meutoken'
    login(token=token)
    # Carregue o dataset de teste
    test_dataset = load_dataset("heloisy/cosmos_qa_ptbr", split="validation")
    dfteste = test_dataset.to_pandas()

    def process_data_sample_teste(example):
        fixed_text = "<|system|>\nVocê é um monitor que ajuda a responder aos usuários as
respostas corretas das questões, como um gabarito. Sempre ajude o usuário respondendo a
alternativa correta.\n\n <|user|>\n"
        alta = 'Alternativa A)' + example['A']
        altb = 'Alternativa B)' + example['B']
        altc = 'Alternativa C)' + example['C']
        altd = 'Alternativa D)' + example['D']
        processed_example = fixed_text + "\n".join([example['contexto'], example['pergunta'],
alta, altb, altc, altd])
        return processed_example
```

```
dfteste[['INSTRUCT_FIELD']] = dfteste.apply(lambda x:
pd.Series(process_data_sample_teste(x)), axis=1)

# Criar uma lista de strings contendo os exemplos de teste pré-processados
test_examples = dfteste['INSTRUCT_FIELD'].tolist()
test_examples_dict = dict(zip(dfteste['id'], test_examples))

tokenizer = AutoTokenizer.from_pretrained("HuggingFaceH4/zephyr-7b-beta")

model = AutoModelForCausalLM.from_pretrained(
    "HuggingFaceH4/zephyr-7b-beta",
    low_cpu_mem_usage=True,
    return_dict=True,
    torch_dtype=torch.float16,
    device_map="cuda")

generation_config = GenerationConfig(
    do_sample=True,
    top_k=1,
    temperature=0.1,
    max_new_tokens=256,
    pad_token_id=tokenizer.unk_token_id
)

import time

# Inicialize uma lista para armazenar as saídas geradas
generated_outputs = []
```

```
# Medir o tempo de geração
st_time = time.time()

# Iterar sobre os exemplos e gerar as saídas
for example_id, example_text in tqdm(test_examples_dict.items()):
    st_time_sample = time.time()
    inputs = tokenizer(example_text, return_tensors="pt").to("cuda")
    outputs = model.generate(**inputs, generation_config=generation_config)
    decoded_output = tokenizer.decode(outputs[0], skip_special_tokens=True)
    print(f"ID: {example_id}\nGenerated Output: {decoded_output}\n")
    print(f"Tempo sample: {time.time() - st_time_sample} segundos")
    generated_outputs.append((example_id, decoded_output))

df_results = pd.DataFrame(generated_outputs, columns=['id', 'Generated Output'])

# Salvar o DataFrame em um arquivo CSV
df_results.to_csv('generated_outputs_zephyr_original.csv', index=False)

# Imprimir o tempo total de geração para o conjunto de teste
print(f"Tempo total de geração para o conjunto de teste: {time.time() - st_time} segundos")

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

## Notebook de ferramenta para anotação

[O documento referente a seguir pode ser acessado em:

<https://colab.research.google.com/drive/1lwU-swfne5CueHwmNP0TKj1sCa4wbfX?usp=sharing>]

```
# Ferramenta para anotar

import pandas as pd
from IPython.display import clear_output

# Função para obter o último índice anotado
def get_last_index():
    try:
        with open('/content/drive/MyDrive/Residência/last_index.txt', 'r') as f:
            return int(f.read())
    except:
        return 0 # Se o arquivo não existir, comece do início

# Função para salvar o último índice anotado
def save_last_index(index):
    with open('/content/drive/MyDrive/Residência/last_index.txt', 'w') as f:
        f.write(str(index))

# Carregar o arquivo CSV
df = pd.read_csv('/content/drive/MyDrive/Residência/geradosoriginal_ anotado.csv')

# Verifique se a coluna 'Anotações' existe, se não, adicione-a
if 'Anotações' not in df.columns:
    df['Anotações'] = ""

# Obter o último índice anotado
last_index = get_last_index()

# Iterar sobre as linhas do DataFrame, começando do último índice
```

```
for index, row in df.iterrows():
    if index >= last_index:
        # Exibindo o "Generated Output"
        print("Generated Output:", row['Generated Output'])

        # Solicitando uma anotação do usuário
        annotation = input("Digite sua anotação para esta entrada (digite 'exit' para sair): ")

        # Verificar se o usuário quer sair
        if annotation.lower() == 'exit':
            save_last_index(index) # Salva o progresso antes de sair
            break

        # Salvando a anotação na nova coluna
        df.at[index, 'Anotações'] = annotation

        # Salvando o último índice anotado
        save_last_index(index + 1) # Salva o próximo índice para continuar depois

        # Limpa o output para a próxima entrada
        clear_output(wait=True)

# Salvando o DataFrame modificado em um novo arquivo CSV
df.to_csv('/content/drive/MyDrive/Residência/geradosoriginal_annotado.csv', index=False)

# Consertando possíveis erros

pd.set_option('display.max_colwidth', None)

df = pd.read_csv('/content/drive/MyDrive/Residência/geradosoriginal_annotado.csv')
```

```
df['Anotações'].value_counts()
```

```
df[df['Anotações'] == 'F']['Generated Output']
```

Verificando esses classificados como "F" na anotação, posso ver que é melhor classificá-los como "Z", logo ajustarei a seguir:

```
df['Anotações'] = df['Anotações'].replace('F', 'Z')
```

```
df['Anotações'].value_counts()
```

```
df['Anotações'] = df['Anotações'].str.upper()
```

```
df['Anotações'].value_counts()
```

```
df[df['Anotações'] == 'H']['Generated Output']
```

Novamente, melhor adequar o classificado como "H" para "Z"

```
df['Anotações'] = df['Anotações'].replace('H', 'Z')
```

```
df[df['Anotações'] == 'Q']['Generated Output']
```

Aqui é um possível erro de digitação, vamos alterar para a alternativa certa

```
df['Anotações'] = df['Anotações'].replace('Q', 'A')
```

```
df[df['Anotações'] == 'NB']['Generated Output']
```

Aqui é um possível erro de digitação, vamos alterar para a alternativa certa

```
df['Anotações'] = df['Anotações'].replace('NB', 'B')
```

```
df['Anotações'].value_counts()
```

Salvando a v2

```
df.to_csv('/content/drive/MyDrive/Residência/geradosoriginal_annotado_ajustado.csv',  
index=False)
```

## Notebook de cálculo de acurácia de todos os modelos trabalhados

[O documento referente a seguir pode ser acessado em:

[https://colab.research.google.com/drive/1-A\\_kqI9WQXYq-oSMdmBQYePJ9LJhV5H9?usp=sharing](https://colab.research.google.com/drive/1-A_kqI9WQXYq-oSMdmBQYePJ9LJhV5H9?usp=sharing)]

```
# Imports
```

```
!pip install datasets
```

```
from datasets import load_dataset
```

```
dataset = load_dataset("heloisy/cosmos_qa_ptbr")["validation"]
```

```
dfval = dataset.to_pandas()
```

```
dfval
```

```
# Fine tuning Zephyr 3090
```

```
import pandas as pd
```

```
df = pd.read_csv('/content/generated_outputs_fine3090.csv')
```

```
df
```

```
for index, row in df.iterrows():
```

```
    decoded_output = row['Generated Output']
```

```
    # Procure a primeira aparição de "Assistente" em decoded_output
```

```
    if "<|assistant|>" in decoded_output:
```

```
        # Divida o texto em linhas
```

```
        lines = decoded_output.split('\n')
```

```
# Encontre a primeira linha após "Assistente" e armazene-a
for i in range(len(lines)):
    if "<|assistant|>" in lines[i]:
        captured_line = lines[i + 1] if i + 1 < len(lines) else None
        break
    else:
        captured_line = None

df.at[index, 'predictions'] = captured_line

# Adicione o resultado ao DataFrame result_df
# result_df = df.append({'example_id': row['example_id'], 'captured_line': captured_line},
ignore_index=True)

# Exiba o DataFrame resultante
df

# Use expressão regular para extrair a letra após "Alternativa"
df['prediction'] = df['predictions'].str.extract(r'Alternativa\s*([A-Z])')
df

df.rename(columns={'ID': 'id'}, inplace=True)

import pandas as pd

# Suponhamos que você já tenha os DataFrames df e dfval

# Verifique se o "id" de df está em dfval
df_merged = df.merge(dfval[['id', 'Label']], on='id', how='inner')
```

```
# Agora, você tem um DataFrame df_merged contendo apenas as linhas onde o "id" existe em dfval
```

```
# Compare a coluna "letra" de df com a "Label" de dfval  
df_merged['comparacao'] = df_merged['prediction'] == df_merged['Label']
```

```
df_merged['comparacao'].value_counts(normalize=True)
```

```
# Fine tuning Zephyr GPTQ Colab
```

```
import pandas as pd  
df = pd.read_csv('/content/generated_outputs_finecolab.csv')  
  
# Use expressão regular para extrair a letra após "Alternativa"  
df['prediction'] = df['predictions'].str.extract(r'Alternativa\s*([A-Z])')  
df
```

```
for index, row in df.iterrows():  
    decoded_output = row['Generated Output']  
  
    # Procure a primeira aparição de "Assistente" em decoded_output  
    if "<|assistant|>" in decoded_output:  
        # Divida o texto em linhas  
        lines = decoded_output.split("\n")  
        # Encontre a primeira linha após "Assistente" e armazene-a  
        for i in range(len(lines)):  
            if "<|assistant|>" in lines[i]:
```

```
        captured_line = lines[i + 1] if i + 1 < len(lines) else None
        break
    else:
        captured_line = None

    df.at[index, 'predictions'] = captured_line

    # Adicione o resultado ao DataFrame result_df
    # result_df = df.append({'example_id': row['example_id'], 'captured_line': captured_line},
    ignore_index=True)

# Exiba o DataFrame resultante
df

df.rename(columns={'ID': 'id'}, inplace=True)

import pandas as pd

# Suponhamos que você já tenha os DataFrames df e dfval

# Verifique se o "id" de df está em dfval
df_merged = df.merge(dfval[['id', 'Label']], on='id', how='inner')

# Agora, você tem um DataFrame df_merged contendo apenas as linhas onde o "id" existe
em dfval

# Compare a coluna "letra" de df com a "Label" de dfval
df_merged['comparacao'] = df_merged['prediction'] == df_merged['Label']

# A coluna 'comparacao' conterá True para as correspondências e False para as não
correspondências
```

```
print(df_merged)
```

```
df_merged['comparacao'].value_counts(normalize=True)
```

```
# Fine tuning Zephyr 3090 v2
```

```
import pandas as pd
```

```
df = pd.read_csv('/content/geradosv2.csv')
```

```
df
```

```
for index, row in df.iterrows():
```

```
    decoded_output = row['Generated Output']
```

```
    # Procure a primeira aparição de "Assistente" em decoded_output
```

```
    if "<|assistant|>" in decoded_output:
```

```
        # Divida o texto em linhas
```

```
        lines = decoded_output.split('\n')
```

```
        # Encontre a primeira linha após "Assistente" e armazene-a
```

```
        for i in range(len(lines)):
```

```
            if "<|assistant|>" in lines[i]:
```

```
                captured_line = lines[i + 1] if i + 1 < len(lines) else None
```

```
                break
```

```
        else:
```

```
captured_line = None

df.at[index, 'predictions'] = captured_line

# Adicione o resultado ao DataFrame result_df
# result_df = df.append({'example_id': row['example_id'], 'captured_line': captured_line},
ignore_index=True)

# Exiba o DataFrame resultante
df

# Use expressão regular para extrair a letra após "Alternativa"
df['prediction'] = df['predictions'].str.extract(r'Alternativa[s]*( [A-Z])')
df

df.rename(columns={'ID': 'id'}, inplace=True)

import pandas as pd

# Suponhamos que você já tenha os DataFrames df e dfval

# Verifique se o "id" de df está em dfval
df_merged = df.merge(dfval[['id', 'Label']], on='id', how='inner')

# Agora, você tem um DataFrame df_merged contendo apenas as linhas onde o "id" existe
em dfval

# Compare a coluna "letra" de df com a "Label" de dfval
df_merged['comparacao'] = df_merged['prediction'] == df_merged['Label']
```

```
df_merged['comparacao'].value_counts(normalize=True)

# Fine tuning Zephyr 3090 v3

import pandas as pd
df = pd.read_csv('/content/geradosv3.csv')

df

for index, row in df.iterrows():
    decoded_output = row['Generated Output']

    # Procure a primeira aparição de "Assistente" em decoded_output
    if "<|assistant|>" in decoded_output:
        # Divida o texto em linhas
        lines = decoded_output.split('\n')
        # Encontre a primeira linha após "Assistente" e armazene-a
        for i in range(len(lines)):
            if "<|assistant|>" in lines[i]:
                captured_line = lines[i + 1] if i + 1 < len(lines) else None
                break
        else:
            captured_line = None

    df.at[index, 'predictions'] = captured_line

# Adicione o resultado ao DataFrame result_df
```

```
# result_df = df.append({'example_id': row['example_id'], 'captured_line': captured_line},
ignore_index=True)

# Exiba o DataFrame resultante
df

# Use expressão regular para extrair a letra após "Alternativa"
df['prediction'] = df['predictions'].str.extract(r'Alternativa\s*([A-Z])')
df

import pandas as pd

# Suponhamos que você já tenha os DataFrames df e dfval

# Verifique se o "id" de df está em dfval
df_merged = df.merge(dfval[['id', 'Label']], on='id', how='inner')

# Agora, você tem um DataFrame df_merged contendo apenas as linhas onde o "id" existe
em dfval

# Compare a coluna "letra" de df com a "Label" de dfval
df_merged['comparacao'] = df_merged['prediction'] == df_merged['Label']

df_merged['comparacao'].value_counts(normalize=True)
```

---

# Inferência Zephyr 3090

```
import pandas as pd
df = pd.read_csv('/content/drive/MyDrive/Residência/geradosoriginal_annotado_ajustado.csv')
```

```
df
```

```
df['Anotações'].value_counts()
```

```
df['Anotações'].value_counts(normalize=True)
```

```
df.rename(columns={'Anotações': 'prediction'}, inplace=True)
```

```
df
```

```
import pandas as pd
```

```
# Suponhamos que você já tenha os DataFrames df e dfval
```

```
# Verifique se o "id" de df está em dfval
```

```
df_merged = df.merge(dfval[['id', 'Label']], on='id', how='inner')
```

```
# Agora, você tem um DataFrame df_merged contendo apenas as linhas onde o "id" existe em dfval
```

```
# Compare a coluna "letra" de df com a "Label" de dfval
```

```
df_merged['comparacao'] = df_merged['prediction'] == df_merged['Label']
```

```
df_merged['comparacao'].value_counts(normalize=True)
```

```
df_merged[df_merged['comparacao'] == False]['prediction'].value_counts(normalize=True)
```

```
df_merged['prediction'].value_counts(normalize=True)
```

Fazendo um experimento, dropando Z e E, para analisar os que ele classificou como opções possíveis, o que era verdade ou erro

```
df_separado = df[(df['prediction'] != 'Z') & (df['prediction'] != 'E')]  
df_separado
```

```
df_merged_separado = df_separado.merge(dfval[['id', 'Label']], on='id', how='inner')
```

# Agora, você tem um DataFrame df\_merged contendo apenas as linhas onde o "id" existe em dfval

```
# Compare a coluna "letra" de df com a "Label" de dfval  
df_merged_separado['comparacao'] = df_merged_separado['prediction'] ==  
df_merged_separado['Label']
```

```
df_merged_separado['comparacao'].value_counts(normalize=True)
```