

✓ Relembrando o Desafio do Tech Challenge

Objetivo Principal:

1. Executar o fine-tuning de um modelo de linguagem (ex.: LLaMA, BERT, GPT) usando o dataset "The AmazonTitles-1.3MM".
2. Receber perguntas dos usuários com base em um contexto (título do produto).
3. Gerar respostas baseadas na descrição do produto após o fine-tuning.
4. Documentar e Apresentar:
5. Explicar os parâmetros, ajustes e resultados.
6. Criar um vídeo demonstrando o modelo em ação.

Dataset:

O dataset contém títulos de produtos e suas descrições provenientes da Amazon.

 Passo 1: Conectar o Google Drive

 Passo 2: Copiar o Dataset do Google Drive

 Passo 3: Salvar o Dataset Processado no Google Drive

```
!pip install transformers datasets
!pip install transformers datasets bitsandbytes peft accelerate loralib
!pip install sentencepiece
!pip install bitsandbytes-cuda117
```

 [Mostrar saída oculta](#)

```
import pandas as pd
import json
import os
import random
from google.colab import drive

import torch
from transformers import LlamaTokenizer, LlamaForCausalLM, GenerationConfig, Trainer
from peft import LoraConfig, get_peft_model, prepare_model_for_kbit_training

from transformers import AutoTokenizer, BitsAndBytesConfig, AutoModelForQuestionAnswering, pipeline

import gzip
import pandas as pd
import json

from huggingface_hub import notebook_login

from google.colab import drive

# Montar o Google Drive
drive.mount('/content/drive')

Mounted at /content/drive

!unzip "/content/drive/MyDrive/Fase3/LF-Amazon-1.3M.raw.zip" -d "/content/drive/MyDrive/Fase3/dataset"

Archive: /content/drive/MyDrive/Fase3/LF-Amazon-1.3M.raw.zip
  creating: /content/drive/MyDrive/Fase3/dataset/LF-Amazon-1.3M/
  inflating: /content/drive/MyDrive/Fase3/dataset/LF-Amazon-1.3M/lbl.json.gz
  inflating: /content/drive/MyDrive/Fase3/dataset/LF-Amazon-1.3M/trn.json.gz
  inflating: /content/drive/MyDrive/Fase3/dataset/LF-Amazon-1.3M/filter_labels_test.txt
  inflating: /content/drive/MyDrive/Fase3/dataset/LF-Amazon-1.3M/tst.json.gz
  inflating: /content/drive/MyDrive/Fase3/dataset/LF-Amazon-1.3M/filter_labels_train.txt

# Caminho do arquivo original
file_path = "/content/drive/MyDrive/Fase3/dataset/LF-Amazon-1.3M/trn.json.gz"

# Função para carregar o arquivo JSON compactado
def load_gzipped_json(path):
    data = []
```

```
with gzip.open(path, 'rt', encoding='utf-8') as f: # Abre como texto com utf-8
    for line in f:
        data.append(json.loads(line))
return pd.DataFrame(data)

# Carregar o dataset
df_train = load_gzipped_json(file_path)
print(df_train.head())

# Inspecionar as primeiras linhas do arquivo
try:
    with gzip.open(file_path, 'rt', encoding='utf-8') as infile:
        for i in range(10): # Inspecionar as primeiras 10 linhas
            line = infile.readline().strip()
            try:
                record = json.loads(line)
                print(f"Linha {i+1}: {record}")
            except json.JSONDecodeError as e:
                print(f"Linha {i+1} malformada: {e}")
except Exception as e:
    print(f"Erro ao abrir o arquivo: {e}")
```

	uid	title \	content \	target_ind \	target_rel
0	0000031909	Girls Ballet Tutu Neon Pink	High quality 3 layer ballet tutu. 12 inches in...	[12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 2...	[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, ...
1	0000032034	Adult Ballet Tutu Yellow		[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 16, 33, 36, 37,...	[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, ...
2	0000913154	The Way Things Work: An Illustrated Encycloped...		[116, 117, 118, 119, 120, 121, 122]	[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
3	0001360000	Mog's Kittens	Judith Kerr’s best–selling adventu...	[146, 147, 148, 149, 495]	[1.0, 1.0, 1.0, 1.0, 1.0]
4	0001381245	Misty of Chincoteague		[151]	[1.0]
Linha 1: {'uid': '0000031909', 'title': 'Girls Ballet Tutu Neon Pink', 'content': 'High quality 3 layer ballet tutu. 12 inches in l...					
Linha 2: {'uid': '0000032034', 'title': 'Adult Ballet Tutu Yellow', 'content': '', 'target_ind': [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 16,					
Linha 3: {'uid': '0000913154', 'title': 'The Way Things Work: An Illustrated Encyclopedia of Technology', 'content': '', 'target_ind					
Linha 4: {'uid': '0001360000', 'title': 'Mog's Kittens', 'content': 'Judith Kerr’s best–selling adventures of that endea					
Linha 5: {'uid': '0001381245', 'title': 'Misty of Chincoteague', 'content': '', 'target_ind': [151], 'target_rel': [1.0]}					
Linha 6: {'uid': '0001371045', 'title': 'Hilda Boswell's treasury of children's stories: A new anthology of stories for the young',					
Linha 7: {'uid': '0000230022', 'title': 'The Simple Truths of Service: Inspired by Johnny the Bagger', 'content': '', 'target_ind':					
Linha 8: {'uid': '0000031895', 'title': 'Girls Ballet Tutu Neon Blue', 'content': 'Dance tutu for girls ages 2-8 years. Perfect for					
Linha 9: {'uid': '0000174076', 'title': 'Evaluating Research in Academic Journals - A Practical Guide to Realistic Evaluation (5th f					
Linha 10: {'uid': '0001713086', 'title': 'Dr. Seuss ABC (Dr.Seuss Classic Collection) (Spanish Edition)', 'content': '', 'target_ind					

🧹 Limpeza do Dataset

```
df_train = df_train.dropna(subset=['content', 'title']) # Remover linhas com valores ausentes em 'content' e 'title'
df_train = df_train.drop_duplicates(subset=['title', 'content']) # Remover duplicatas com base em 'title' e 'content'
df_train['title'] = df_train['title'].str.lower()
df_train['content'] = df_train['content'].str.lower()

df_train = df_train.dropna(subset=['content'])
df_train = df_train.dropna(subset=['title'])
print(df_train.head())
```

	uid	title \	content \
0	0000031909	Girls Ballet Tutu Neon Pink	High quality 3 layer ballet tutu. 12 inches in...
1	0000032034	Adult Ballet Tutu Yellow	
2	0000913154	The Way Things Work: An Illustrated Encycloped...	
3	0001360000	Mog's Kittens	
4	0001381245	Misty of Chincoteague	

```

2
3 Judith Kerr&#8217;s best&#8211;selling adventu...
4

                                target_ind \
0 [12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 2...
1 [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 16, 33, 36, 37,...
2 [116, 117, 118, 119, 120, 121, 122]
3 [146, 147, 148, 149, 495]
4 [151]

                                target_rel
0 [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, ...
1 [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, ...
2 [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
3 [1.0, 1.0, 1.0, 1.0, 1.0]
4 [1.0]

import json
import re
import gzip

# Variável para contar registros limpos
total_cleaned = 0

# Função de limpeza de texto
def clean_text(text):
    text = text.strip() # Remover espaços extras, mantendo maiúsculas
    text = re.sub(r'^a-zA-Z0-9.,!?:\s', '', text) # Manter letras (maiúsculas e minúsculas), números e pontuações comuns
    text = re.sub(r'\s+', ' ', text) # Substituir múltiplos espaços por um único espaço
    return text

# Caminhos dos arquivos
file_path = "/content/drive/MyDrive/Fase3/dataset/LF-Amazon-1.3M/trn.json.gz"
cleaned_file_path = "/content/cleaned_trn.json"

try:
    with gzip.open(file_path, 'rt', encoding='utf-8') as infile, open(cleaned_file_path, 'w') as outfile:
        outfile.write('[') # Início do array JSON
        first_record = True # Para gerenciar vírgulas

        for i, line in enumerate(infile):
            try:
                record = json.loads(line.strip())

                # Verificar e limpar os campos
                title = record.get('title', '').strip()
                content = record.get('content', '').strip()

                # Ignorar se 'content' ou 'title' estiver vazio
                if title and content:
                    cleaned_record = {
                        'prompt': clean_text(title),
                        'response': clean_text(content)
                    }

                    if not first_record:
                        outfile.write(',') # Separador entre registros

                    json.dump(cleaned_record, outfile)
                    total_cleaned += 1
                    first_record = False

            except json.JSONDecodeError as e:
                print(f"Linha {i+1} ignorada devido a erro: {e}")

        outfile.write(']') # Fim do array JSON

except Exception as e:
    print(f"Erro ao processar: {e}")

print(f"Total de registros limpos: {total_cleaned}")
print(f"Arquivo limpo salvo em: {cleaned_file_path}")

➡ Total de registros limpos: 1390403
Arquivo limpo salvo em: /content/cleaned_trn.json

#import json

# Caminho do arquivo limpo
cleaned_file_path = "/content/cleaned_trn.json"

```

```
# Carregar o arquivo limpo e contar os registros
try:
    with open(cleaned_file_path, 'r') as infile:
        data = json.load(infile) # Carrega o conteúdo JSON como uma lista de dicionários

    # Mostrar a quantidade de registros
    print(f"Total de registros após a limpeza: {len(data)}")

    # Mostrar os primeiros 5 registros
    print("Exemplo dos primeiros registros:")
    for i, record in enumerate(data[:5]):
        print(f"\nRegistro {i + 1}:")
        print(json.dumps(record, indent=2, ensure_ascii=False))

except Exception as e:
    print(f"Erro ao abrir o arquivo limpo: {e}")
```

```
↗ Total de registros após a limpeza: 1390403
Exemplo dos primeiros registros:

Registro 1:
{
  "prompt": "Girls Ballet Tutu Neon Pink",
  "response": "High quality 3 layer ballet tutu. 12 inches in length"
}

Registro 2:
{
  "prompt": "Mog's Kittens",
  "response": "Judith Kerr8217;s best8211;selling adventures of that endearing and exasperating cat Mog have entertained children fo
}

Registro 3:
{
  "prompt": "Girls Ballet Tutu Neon Blue",
  "response": "Dance tutu for girls ages 28 years. Perfect for dance practice, recitals and performances, costumes or just for fun!
}

Registro 4:
{
  "prompt": "The Prophet",
  "response": "In a distant, timeless place, a mysterious prophet walks the sands. At the moment of his departure, he wishes to offe
}

Registro 5:
{
  "prompt": "Rightly Dividing the Word",
  "response": "This text refers to thePaperbackedition."
}
```

✓ 🌟 Gráfico e Tabela para Visualização

O gráfico irá mostrar:

Distribuição do tamanho dos prompts e respostas. Quantidade de textos dentro e fora dos limites definidos.

Processar textos longos aumenta:

Custo: GPT cobra por token processado. Tempo de Treinamento: Modelos como BERT e LLaMA levam mais tempo com entradas maiores.

Limitações: Alguns modelos têm limites (ex.: 512 ou 1024 tokens).

🇧🇷 Objetivos da Análise

Determinar Distribuição de Comprimentos:

A análise dos cumprimentos dos prompts e responses te dá uma visão clara de quantas palavras eles contêm. Isso ajuda a identificar se os textos são geralmente curtos ou longos.

Definir um max_length Adequado:

Durante o treinamento, você precisa definir um max_length para a tokenização (ex.: 512 tokens).

Se os prompts ou responses forem muito longos, podem ser truncados durante o treinamento, perdendo informações importantes.

Avaliar Impacto de Truncar Textos:

Comparar os limites definidos (prompt_limit = 50 e response_limit = 150) com a distribuição real.

Se muitos textos ultrapassam esses limites, pode ser necessário aumentar o max_length ou ajustar os dados de entrada.

```

import json
import matplotlib.pyplot as plt
import pandas as pd

# Caminho do arquivo
file_path = "/content/cleaned_trn.json"

# Variáveis para análise
prompt_lengths = []
response_lengths = []
prompt_limit = 50
response_limit = 300

# Analisar o comprimento dos textos
try:
    with open(file_path, 'r') as file:
        data = json.load(file)

        for record in data:
            prompt_lengths.append(len(record['prompt'].split()))
            response_lengths.append(len(record['response'].split()))
except Exception as e:
    print(f"Erro ao carregar os dados: {e}")

# Criar tabela com os dados
df = pd.DataFrame({
    'Prompt Length': prompt_lengths,
    'Response Length': response_lengths
})
df['Prompt Category'] = df['Prompt Length'].apply(lambda x: 'Long' if x > prompt_limit else 'Short')
df['Response Category'] = df['Response Length'].apply(lambda x: 'Long' if x > response_limit else 'Short')

# Calcular proporções
prompt_counts = df['Prompt Category'].value_counts()
response_counts = df['Response Category'].value_counts()
prompt_percent = (prompt_counts / len(df)) * 100
response_percent = (response_counts / len(df)) * 100

# Criar DataFrame de resumo
summary_df = pd.DataFrame({
    'Category': ['Short Prompts', 'Long Prompts', 'Short Responses', 'Long Responses'],
    'Count': [prompt_counts.get('Short', 0), prompt_counts.get('Long', 0),
              response_counts.get('Short', 0), response_counts.get('Long', 0)],
    'Percentage': [prompt_percent.get('Short', 0), prompt_percent.get('Long', 0),
                  response_percent.get('Short', 0), response_percent.get('Long', 0)]
})

# Exibir a tabela no Colab
print("Tabela de análise com porcentagens:")
display(summary_df)

# Gráficos
# 1. Distribuição dos comprimentos
plt.figure(figsize=(10, 5))
plt.hist(prompt_lengths, bins=30, alpha=0.7, label='Prompts')
plt.hist(response_lengths, bins=30, alpha=0.7, label='Responses')
plt.axvline(prompt_limit, color='red', linestyle='--', label='Prompt Limit')
plt.axvline(response_limit, color='blue', linestyle='--', label='Response Limit')
plt.title('Distribuição dos Comprimentos dos Textos')
plt.xlabel('Número de Palavras')
plt.ylabel('Frequência')
plt.legend()
plt.show()

# 2. Quantidade de textos curtos/longos com porcentagens
plt.figure(figsize=(8, 5))
response_counts.plot(kind='bar', color=['orange', 'green'])
plt.title('Quantidade de Respostas Curtas/Longas')
plt.ylabel('Frequência')
for i, count in enumerate(response_counts):
    plt.text(i, count, f"{response_percent[i]:.2f}%", ha='center', va='bottom')
plt.show()

prompt_counts.plot(kind='bar', color=['orange', 'green'])
plt.title('Quantidade de Prompts Curtos/Longos')
plt.ylabel('Frequência')
for i, count in enumerate(prompt_counts):
    plt.text(i, count, f"{prompt_percent[i]:.2f}%", ha='center', va='bottom')
plt.show()

```


Tabela de análise com porcentagens:

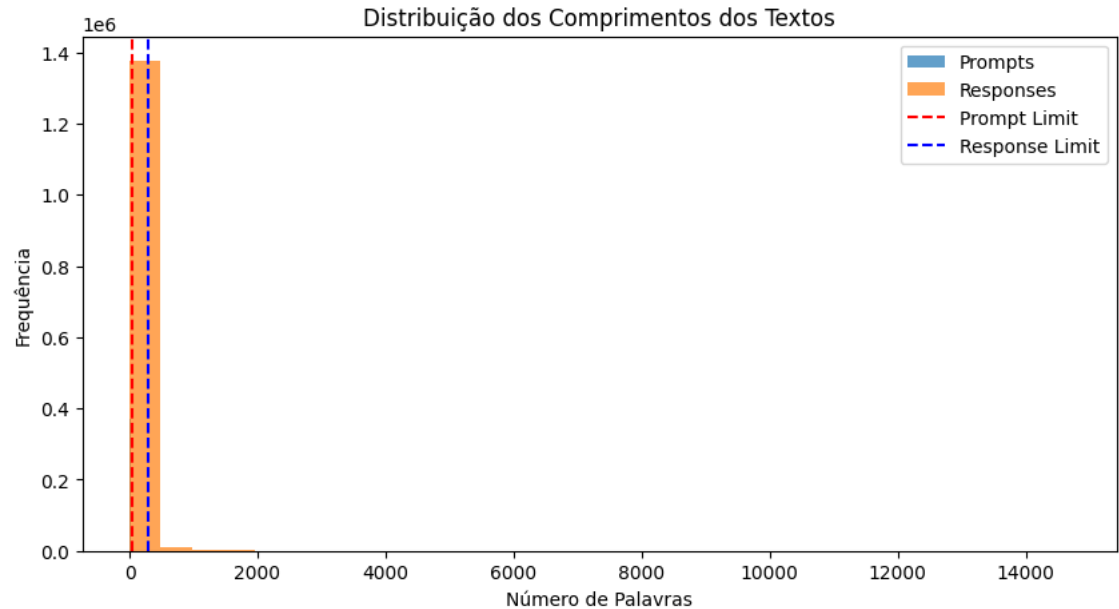
1 to 4 of 4 entriesFilter

index	Category ▲	Count	Percentage
1	Long Prompts	2557	0.18390351574327732
3	Long Responses	46313	3.330904780844115
0	Short Prompts	1387846	99.81609648425672
2	Short Responses	1344090	96.66909521915589

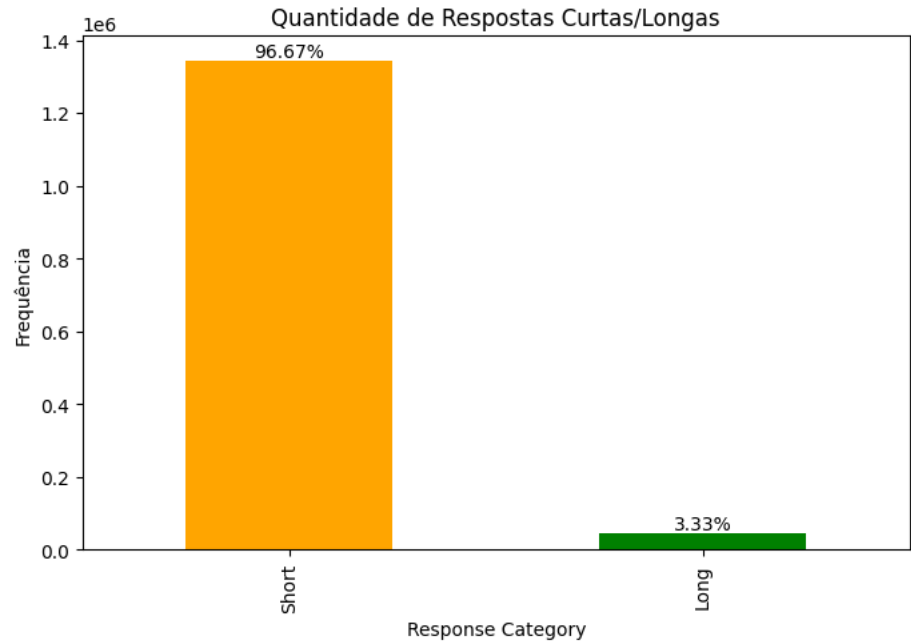
Show 25 per page



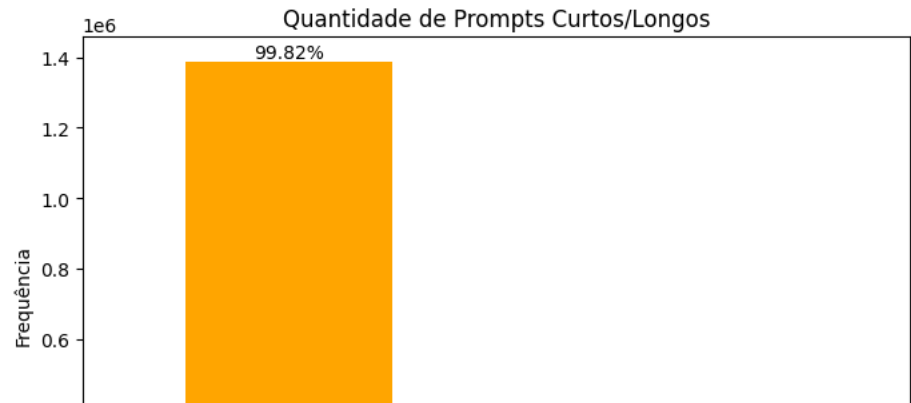
Like what you see? Visit the [data table notebook](#) to learn more about interactive tables.

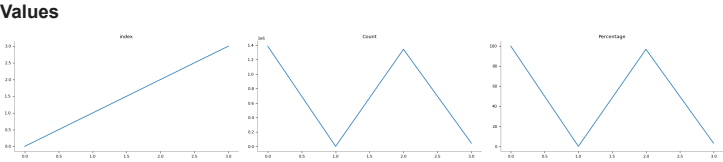
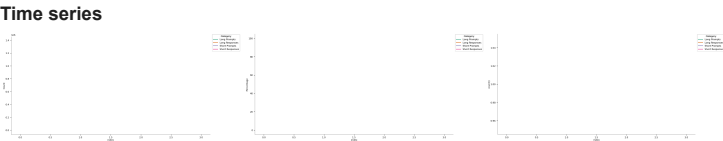
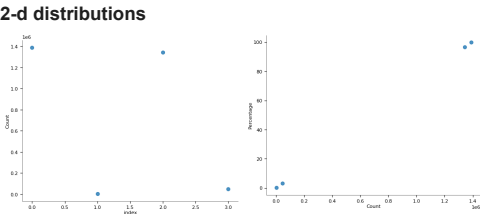
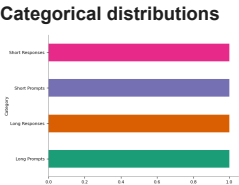
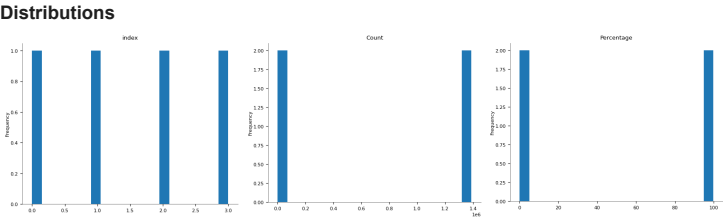
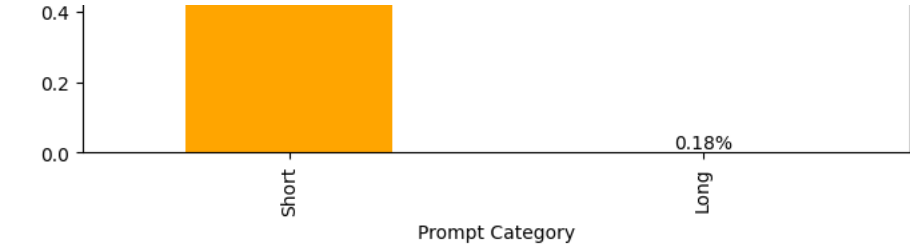


<ipython-input-35-e5cda0f83b2f>:71: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future version, plt.text(i, count, f"{response_percent[i]:.2f}%", ha='center', va='bottom')



<ipython-input-35-e5cda0f83b2f>:79: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future version, plt.text(i, count, f"{prompt_percent[i]:.2f}%", ha='center', va='bottom')





Faceted distributions

<string>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend` to `True`.

<string>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend` to `True`.

<string>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend` to `True`.

Próximas etapas:

Gerar código com summary_df

Ver gráficos recomendados

New interactive sheet

Comece a programar ou gere código com IA.

Limites de Tokens por Modelo

Análise dos Comprimentos dos Prompts e Responses

Index	Categoria	Contagem	Porcentagem
0	Short Prompts	1.387.846	99,82%
1	Long Prompts	2.557	0,18%
2	Short Responses	1.344.090	96,67%
3	Long Responses	46.313	3,33%

- GPT: Geralmente até 4096 tokens (pode variar por versão).
- BERT: 512 tokens.
- LLAMA: Geralmente 2048 tokens ou mais.

Explicação dos Limites

Conversão de Palavras para Tokens:

1 palavra ≈ 1.5 tokens (média para textos em português).

Cálculo dos Tokens:

- 50 palavras → 75 tokens.
- 300 palavras → 450 tokens.

Recomendações de max_length:

Componente	max_length Sugerido
Prompts	75 tokens
Responses	450 tokens
Total	525 tokens

Comece a programar ou [gere código](#) com IA.

Recomendação de Otimizações possíveis:

Ação	Objetivo
Resumização	Reduzir responses longas para até 300 palavras.
Filtragem	Tratar ou excluir responses muito longas (> 500 tokens).
Tokenização Eficiente	Garantir que a tokenização está otimizada para o português.
Data Augmentation	Aumentar dados para equilibrar a proporção de respostas.

Essas estratégias podem ajudar a melhorar a **eficiência do modelo**, reduzir **custos computacionais**, e manter a **qualidade** dos resultados.

▼ Próximos Passos

- Confirmar o Dataset Final:
- Agora temos o arquivo cleaned_trn.json com:

```
-- Limpeza completa.
```

```
-- Remoção de duplicatas.
```
- Reduzir aleatoriamente para 40k Registros

Motivos para Reduzir o Dataset

Limitações de Recursos Computacionais:

Memória da GPU: Mesmo com GPUs poderosas, como a A100 (40 GB), um dataset grande pode facilmente esgotar a memória disponível.

Tempo de Treinamento: Reduzir o número de registros ajuda a diminuir o tempo de treinamento, tornando o processo mais ágil e viável.

Custos Computacionais:

Se você estiver usando serviços pagos, como RunPod ou Colab Pro, cada hora de treinamento tem um custo. Reduzir o tamanho do dataset ajuda a controlar os gastos.

Overfitting:

Em muitos casos, usar um dataset muito grande pode levar a overfitting (quando o modelo se ajusta demais aos dados de treino e não generaliza bem). Um tamanho menor, mas diversificado, pode melhorar a generalização.

Validação Rápida:

Um dataset menor facilita rodar múltiplas iterações de treino e validação rápida, permitindo ajustar os hiperparâmetros de forma eficiente.

```
import json
import random

# Caminho do arquivo limpo completo
cleaned_path = "/content/cleaned_trn.json"
reduced_file_path = "/content/reduced_40k_trn.json"

# Número desejado de registros#
desired_size = 40000

try:
    # Carregar os dados limpos existentes
    with open(cleaned_path, 'r') as file:
        data = json.load(file)

    print(f"Total de registros disponíveis: {len(data)}")

    # Embaralhar os dados
    random.shuffle(data)

    # Reduzir ao tamanho desejado
    data_reduced = data[:desired_size]
    print(f"Total de registros após redução: {len(data_reduced)}")

    # Salvar o novo conjunto reduzido
    with open(reduced_file_path, 'w', encoding='utf-8') as outfile:
        json.dump(data_reduced, outfile, ensure_ascii=False, indent=2)

    # Verificar o tamanho final
    print(f"Tamanho do dataset reduzido: {len(data_reduced)} registros")
    print(f"Dataset reduzido salvo em: {reduced_file_path}")

except Exception as e:
    print(f"Erro ao processar os dados: {e}")
```

```
↵ Total de registros disponíveis: 1390403
Total de registros após redução: 40000
Tamanho do dataset reduzido: 40000 registros
Dataset reduzido salvo em: /content/reduced_40k_trn.json
```

Próximos Passos

- Carregar o Modelo:
 - Upload do arquivo reduced_40k_trn.json no modelo para iniciar o processo de fine-tuning.
- Configurar o Fine-Tuning - Hiperparâmetros:
 - Hiperparâmetros recomendados:
 - Número de Épocas: 3 a 5
 - Batch Size: 32

- Learning Rate: Entre 1e-5 (0.00001) e 5e-5 (0.00005).

- Executar o Fine-Tuning:
- Iniciar o treinamento e monitorar o progresso.
- Comparar o Desempenho:
- Usar os dados de teste (tst.json) e as respostas esperadas (lbl.json) para comparar o desempenho antes e depois do fine-tuning.
- Avaliar as Métricas:

Métrica	Descrição
Acurácia	Percentual de respostas corretas em relação ao total de exemplos testados.
Precisão	Percentual de respostas corretas entre as respostas que o modelo classificou como corretas.
Recall	Percentual de respostas corretas identificadas corretamente em relação ao total de respostas reais.
F1-Score	Média harmônica entre precisão e recall (bom para dados desbalanceados).
Score BLEU/ROUGE	Métricas usadas para avaliar similaridade entre textos gerados e textos de referência.

✂ Explicando Hiperparâmetros Recomendados

🔥 1. Número de Épocas (num_train_epochs)

- O que é: Uma época representa uma passagem completa por todos os dados de treinamento.
- Para que serve: Controla quantas vezes o modelo irá passar pelo dataset completo durante o treinamento.
- Recomendação: Valor recomendado: Entre 3 e 5 épocas.
- Explicação: Um número muito baixo pode resultar em um modelo subtreinado, enquanto um número muito alto pode causar overfitting (o modelo se adapta demais ao treinamento e não generaliza bem para novos dados).

🔥 2. Tamanho do Batch (Batch Size)

- O que é: Número de amostras processadas antes de o modelo atualizar os pesos.
- Para que serve: Batch Size influencia a memória usada pelo treinamento e a estabilidade do gradiente.

Um batch menor usa menos memória, mas pode gerar atualizações menos estáveis.

Um batch maior pode levar a atualizações mais precisas, mas exige mais memória.

- Recomendação: Valor recomendado: 32 (pode variar de 4 a 64 dependendo da memória disponível).
- Explicação:

Batch pequeno: Melhora a capacidade de generalização, mas pode ser instável (oscilações nos gradientes).

Batch grande: Mais estável, mas pode levar a overfitting e requer mais recursos computacionais.

🔥 3. Taxa de Aprendizado (Learning Rate)

- O que é: Define o tamanho dos passos de ajuste dos pesos durante a descida do gradiente.
- Para que serve: Controla a velocidade do aprendizado. Um valor muito alto pode fazer o modelo não convergir (os pesos oscilam demais).

Um valor muito baixo pode tornar o treinamento lento ou ficar preso em mínimos locais.

- Recomendação: Valor recomendado: Entre 1e-5 (0.00001) e 5e-5 (0.00005).
- Explicação: Valores típicos: Em muitos casos, usar o valor padrão sugerido pela ferramenta (como o Playground) é suficiente. Ajuste fino: Para modelos grandes, valores menores ajudam a evitar oscilações.

🏠 Dicas para Escolher os Valores Ideais

Fazer testes com diferentes combinações dos hiperparâmetros e monitorar os gráficos de training loss e validation loss.

- Batch Size: Ajustar conforme a capacidade da sua GPU/CPU. Se tiver memória limitada, use valores menores.
- Learning Rate: Se a training loss não diminuir ou oscilar muito, reduza a taxa de aprendizado.

- Número de Épocas: Se a validation loss começar a aumentar após algumas épocas, pode ser sinal de overfitting; reduza o número de épocas.

✓ Principais Hiperparâmetros

Parâmetro	Valor	Descrição
num_train_epochs	3	Número de épocas de treinamento.
learning_rate	2e-05	Taxa de aprendizado para o otimizador.
per_device_train_batch_size	4	Tamanho do lote (batch size) por GPU durante o treinamento.
fp16	True	Treinamento em precisão mista para reduzir uso de memória.
logging_steps	100	Intervalo de passos para registrar logs.
save_steps	500	Salvar o modelo a cada 500 passos.
output_dir	./llama-fine-tuned	Diretório onde o modelo fine-tuned será salvo.
save_total_limit	2	Manter no máximo 2 checkpoints de modelo salvos.
report_to	['tensorboard', 'wandb']	Relatar logs para TensorBoard e Weights & Biases (W&B) .

✎ Vamos usar o LLAMA 3.2

✎ 1. Configurar o Ambiente no Google Colab

No Google Colab, obter uma GPU disponível:
Em "Ambiente de execução" > "Alterar tipo de ambiente de execução" e selecione GPU.

```
!pip install transformers accelerate bitsandbytes peft datasets
```

↗ Mostrar saída oculta

✎ 2. Carregar e Preparar os Dados

```
import json

# Caminho do dataset reduzido
data_path = "/content/reduced_40k_trn.json"

# Carregar os dados
with open(data_path, 'r') as file:
    data = json.load(file)

print(f"Total de registros: {len(data)}")
print("Exemplo de dados:")
print(data[:2])
```

↗ Total de registros: 40000
Exemplo de dados:
[{'prompt': 'michael ohalloran library of indiana classics', 'response': 'originally published in the early 1900s this volume tells

✎ 3. Configurar o Modelo LLAMA 3.2 e os Hiperparâmetros

3. Autenticar no Google Colab

```
from huggingface_hub import login

# Insira o token gerado
#login("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")
login()
```



Iniciando o Fine Tuning

✓ Solução: Utilizar PEFT (Parameter-Efficient Fine-Tuning)

Para realizar o fine-tuning em modelos quantizados, utilizamos técnicas de PEFT (Parameter-Efficient Fine-Tuning), como LoRA (Low-Rank Adaptation), que adiciona camadas treináveis ao modelo sem modificar os pesos originais.

🔧 Passo a Passo para Resolver com LoRA

```
!pip install peft
```



Mostrar saída oculta

```
!pip install --upgrade peft transformers bitsandbytes accelerate
```



Mostrar saída oculta

🔧 Configuração de Weights & Biases (W&B) e TensorBoard

```
!pip install wandb
```




Mostrar saída oculta

```
!wandb login
```



```
wandb: Logging into wandb.ai. (Learn how to deploy a W&B server locally: https://wandb.me/wandb-server)
wandb: You can find your API key in your browser here: https://wandb.ai/authorize
wandb: Paste an API key from your profile and hit enter, or press ctrl+c to quit:
wandb: Appending key for api.wandb.ai to your netrc file: /root/.netrc
```

```
import wandb
wandb.init(project="llama3-finetuning", name="run-1")
```


 **wandb:** Using wandb-core as the SDK backend. Please refer to <https://wandb.me/wandb-core> for more information.
wandb: Currently logged in as: **abracord2022** (**abracord2022-abracord**). Use `wandb login --relogin` to force relogin
Tracking run with wandb version 0.18.7
Run data is saved locally in /content/wandb/run-20241215_144711-91420g9r
Syncing run **run-1** to [Weights & Biases \(docs\)](#)
View project at <https://wandb.ai/abracord2022-abracord/llama3-finetuning>
View run at <https://wandb.ai/abracord2022-abracord/llama3-finetuning/runs/9l420g9r>



Access Denied

This page can not be accessed from within an iframe


```
!nvcc --version
```

 **nvcc:** NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2023 NVIDIA Corporation
Built on Tue_Aug_15_22:02:13_PDT_2023
Cuda compilation tools, release 12.2, V12.2.140
Build cuda_12.2.r12.2/compiler.33191640_0

```
!pip install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cu121
```

 **Mostrar saída oculta**

```
import torch
print(f"CUDA disponível: {torch.cuda.is_available()}")
print(f"Versão do CUDA: {torch.version.cuda}")
print(f"Versão do PyTorch: {torch.__version__}")
```

 **CUDA disponível:** True
Versão do CUDA: 12.1
Versão do PyTorch: 2.5.1+cu121

```
#utilizando quantização, reinstale a biblioteca bitsandbytes:
!pip install bitsandbytes --no-cache-dir
```


 **Mostrar saída oculta**

```
!pip uninstall bitsandbytes -y
!pip install bitsandbytes --upgrade --no-cache-dir
```

 **WARNING: Skipping bitsandbytes as it is not installed.**
 Collecting bitsandbytes
 Downloading bitsandbytes-0.45.0-py3-none-manylinux_2_24_x86_64.whl.metadata (2.9 kB)
 Requirement already satisfied: torch in /usr/local/lib/python3.10/dist-packages (from bitsandbytes) (2.5.1+cu121)
 Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from bitsandbytes) (1.26.4)
 Requirement already satisfied: typing_extensions>=4.8.0 in /usr/local/lib/python3.10/dist-packages (from bitsandbytes) (4.12.2)
 Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from torch->bitsandbytes) (3.16.1)
 Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from torch->bitsandbytes) (3.4.2)
 Requirement already satisfied: Jinja2 in /usr/local/lib/python3.10/dist-packages (from torch->bitsandbytes) (3.1.4)
 Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from torch->bitsandbytes) (2024.9.0)
 Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.1.105 in /usr/local/lib/python3.10/dist-packages (from torch->bitsandbytes) (12.1.105)
 Requirement already satisfied: nvidia-cuda-runtime-cu12==12.1.105 in /usr/local/lib/python3.10/dist-packages (from torch->bitsandbytes) (12.1.105)
 Requirement already satisfied: nvidia-cuda-cupti-cu12==12.1.105 in /usr/local/lib/python3.10/dist-packages (from torch->bitsandbytes) (12.1.105)
 Requirement already satisfied: nvidia-cudnn-cu12==9.1.0.70 in /usr/local/lib/python3.10/dist-packages (from torch->bitsandbytes) (9.1.0.70)
 Requirement already satisfied: nvidia-cublas-cu12==12.1.3.1 in /usr/local/lib/python3.10/dist-packages (from torch->bitsandbytes) (12.1.3.1)
 Requirement already satisfied: nvidia-cufft-cu12==11.0.2.54 in /usr/local/lib/python3.10/dist-packages (from torch->bitsandbytes) (11.0.2.54)
 Requirement already satisfied: nvidia-curand-cu12==10.3.2.106 in /usr/local/lib/python3.10/dist-packages (from torch->bitsandbytes) (10.3.2.106)
 Requirement already satisfied: nvidia-cusolver-cu12==11.4.5.107 in /usr/local/lib/python3.10/dist-packages (from torch->bitsandbytes) (11.4.5.107)
 Requirement already satisfied: nvidia-cusparselt-cu12==12.1.0.106 in /usr/local/lib/python3.10/dist-packages (from torch->bitsandbytes) (12.1.0.106)
 Requirement already satisfied: nvidia-nccl-cu12==2.21.5 in /usr/local/lib/python3.10/dist-packages (from torch->bitsandbytes) (2.21.5)
 Requirement already satisfied: nvidia-nvtx-cu12==12.1.105 in /usr/local/lib/python3.10/dist-packages (from torch->bitsandbytes) (12.1.105)
 Requirement already satisfied: triton==3.1.0 in /usr/local/lib/python3.10/dist-packages (from torch->bitsandbytes) (3.1.0)
 Requirement already satisfied: sympy==1.13.1 in /usr/local/lib/python3.10/dist-packages (from torch->bitsandbytes) (1.13.1)
 Requirement already satisfied: nvidia-nvjitlink-cu12 in /usr/local/lib/python3.10/dist-packages (from nvidia-cusolver-cu12==11.4.5.107->torch->bitsandbytes) (12.1.105)
 Requirement already satisfied: mpmath<1.4, >=1.1.0 in /usr/local/lib/python3.10/dist-packages (from sympy==1.13.1->torch->bitsandbytes) (1.3.0)
 Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from Jinja2->torch->bitsandbytes) (3.0.2)
 Downloading bitsandbytes-0.45.0-py3-none-manylinux_2_24_x86_64.whl (69.1 MB)
 69.1/69.1 MB 278.7 MB/s eta 0:00:00
 Installing collected packages: bitsandbytes
 Successfully installed bitsandbytes-0.45.0
WARNING: The following packages were previously imported in this runtime:
[bitsandbytes]
You must restart the runtime in order to use newly installed versions.

RESTART SESSION

```
import bitsandbytes as bnb
print(f"Versão do bitsandbytes: {bnb.__version__}")
```

 Versão do bitsandbytes: 0.45.0

```
import json
import torch
import gc
import wandb
from transformers import (
    AutoTokenizer,
    AutoModelForCausalLM,
    BitsAndBytesConfig,
    TrainingArguments,
    Trainer,
    DataCollatorForSeq2Seq
)
from peft import get_peft_model, LoraConfig, TaskType
from datasets import Dataset

# Limpeza de cache CUDA
torch.cuda.empty_cache()
gc.collect()

# 1 Login no wandb (interativo)
#wandb login

# 2 Configuração de quantização em 8 bits
quantization_config = BitsAndBytesConfig(load_in_8bit=True)

# 3 Carregar o modelo e tokenizer
model_name = "meta-llama/llama-3.2-1B"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(model_name, device_map="auto", quantization_config=quantization_config)

# Definir o pad_token como eos_token
tokenizer.pad_token = tokenizer.eos_token

# 4 Configurar LoRA
lora_config = LoraConfig(task_type=TaskType.CAUSAL_LM, r=16, lora_alpha=32, lora_dropout=0.1)
model = get_peft_model(model, lora_config)

# 5 Carregar o dataset
data_path = "/content/reduced_40k_trn.json"
with open(data_path, 'r') as file:
```

```
data = json.load(file)
dataset = Dataset.from_list(data)

# 6 Preprocessar o dataset
# Preprocessar o dataset com max_length ajustado - aqui entra o estudo do max_length
def preprocess_data(example):
    combined = f"### Prompt: {example['prompt']}\n### Response: {example['response']}"
    inputs = tokenizer(combined, truncation=True, max_length=700, padding="max_length")
    inputs["labels"] = inputs["input_ids"].copy()
    return inputs

tokenized_dataset = dataset.map(preprocess_data)

# Dividir em treino e validação (80/20) Usando um dataset de 40K atingimos os >30k para o treinamento , exigido no TechChallenge
train_test_split = tokenized_dataset.train_test_split(test_size=0.2)
train_dataset = train_test_split['train']
eval_dataset = train_test_split['test']

# 7 Configurar o Data Collator
data_collator = DataCollatorForSeq2Seq(tokenizer, model=model)

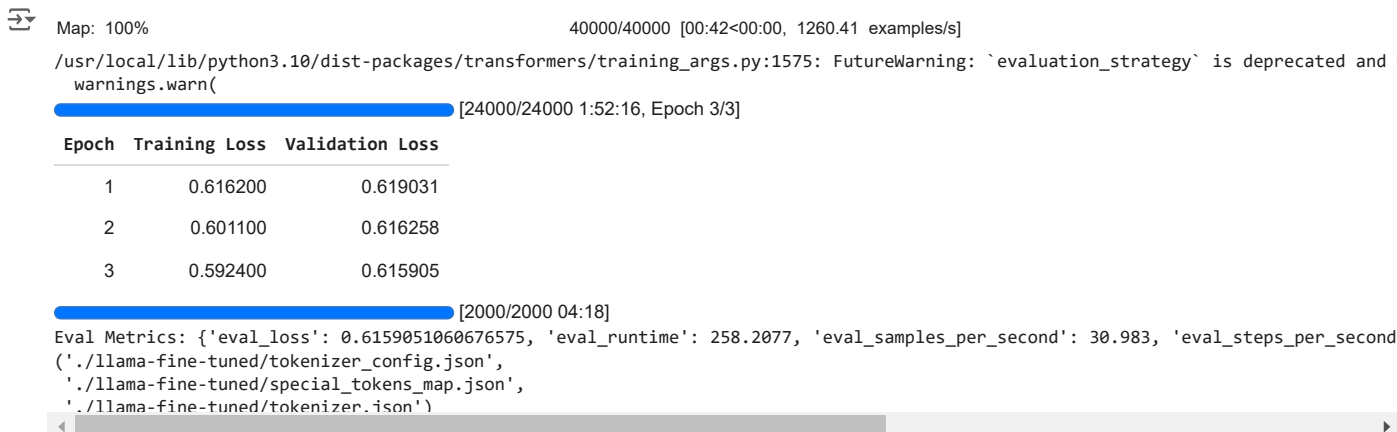
# 8 Configurar o treinamento com métricas
training_args = TrainingArguments(
    output_dir="./results",
    evaluation_strategy="epoch", #usar eval_strategy para v4.47 Transformers
    logging_dir="./logs",
    logging_strategy="epoch",
    num_train_epochs=3,
    per_device_train_batch_size=4,
    per_device_eval_batch_size=4,
    learning_rate=2e-4,
    save_total_limit=2,
    save_strategy="epoch",
    report_to=["wandb"], # Reportar para o wandb
    load_best_model_at_end=True, # Carregar o melhor modelo ao final
    metric_for_best_model="eval_loss", # Métrica para selecionar o melhor modelo
)

# 9 Inicializar o Trainer
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=eval_dataset,
    data_collator=data_collator,
)

# 10 Iniciar o treinamento
trainer.train()

# Avaliar o modelo após o treinamento
eval_metrics = trainer.evaluate()
print(f"Eval Metrics: {eval_metrics}")

# Salvar o modelo e o tokenizer ajustados
trainer.save_model("./llama-fine-tuned")
tokenizer.save_pretrained("./llama-fine-tuned")
```



Quadro de Interpretação de Loss para Treinamento e Validação

Interpretando valores de Training Loss e Validation Loss em treinamentos de modelos de linguagem, como GPT-2 e Llama 3.

Categoria	Training Loss	Validation Loss	Interpretação
✅ Excelente	0.0 - 1.0	0.0 - 1.0	O modelo está aprendendo bem e generalizando para novos dados.
🟢 Bom	1.0 - 2.5	1.0 - 2.5	O modelo está performando bem, com espaço para pequenas melhorias.
🟡 Médio	2.5 - 4.0	2.5 - 4.0	O modelo está aprendendo, mas pode estar subajustado ou com dados complexos.
🔴 Ruim	4.0 ou maior	4.0 ou maior	O modelo não está aprendendo corretamente, provavelmente há algum problema.
⚠️ Overfitting	Muito baixo (< 1.0)	Muito maior (> 2.5)	O modelo está memorizando os dados de treino, mas não generaliza bem.
⚠️ Underfitting	Alto (> 3.0)	Alto (> 3.0)	O modelo não está aprendendo o suficiente, possivelmente devido a poucos dados.

Monitorar o Treinamento 🇧🇷 Training Loss e Validation Loss

💠 1. Training Loss (Perda de Treinamento)

O que é:

É a métrica que mede o erro do modelo nos dados de treinamento a cada iteração/época. Representa o quão bem o modelo está aprendendo os exemplos fornecidos.

Comportamento Esperado: Durante o treinamento, a Training Loss deve diminuir gradualmente conforme o modelo aprende a se ajustar melhor aos dados de treinamento.

💠 2. Validation Loss (Perda de Validação)

O que é:

É a métrica que mede o erro do modelo nos dados de validação (eval_dataset), que o modelo nunca viu durante o treinamento. Serve para verificar se o modelo está generalizando bem ou se está "decorando" (overfitting) os dados de treinamento.

Comportamento Esperado: No início, a Validation Loss deve diminuir junto com a Training Loss.

Se a Validation Loss começar a aumentar enquanto a Training Loss continua diminuindo, é um sinal de overfitting.

🔧 Interpretando os Dados Se a Training Loss e a Validation Loss diminuem juntas:

✅ O modelo está aprendendo corretamente.

Se a Training Loss continua caindo, mas a Validation Loss aumenta:

❗ O modelo está começando a overfitting.

Soluções possíveis:

Reduzir o número de épocas. Usar técnicas de regularização (dropout, weight decay). Aumentar o tamanho dos dados de validação.

▼ Pós Treinamento

📁 Carregar os Dados de Teste e o Ground Truth

```
import json
import gzip

# Caminho do arquivo de teste
test_data_path = "/content/drive/MyDrive/Fase3/dataset/LF-Amazon-1.3M/tst.json.gz"

# Carregar dados de teste
test_data = []
try:
    with gzip.open(test_data_path, 'rt', encoding='utf-8') as file:
        for i, line in enumerate(file):
            test_data.append(json.loads(line))
            if i < 2: # Visualizar os dois primeiros registros
                print(f"Linha {i + 1}: {test_data[-1]}")
```

```
except Exception as e:
    print(f"Erro ao carregar os dados de teste: {e}")

print(f"\nTotal de registros de teste: {len(test_data)}")
```

↻ Linha 1: {'uid': '0000032069', 'title': 'Adult Ballet Tutu Cheetah Pink', 'content': '', 'target_ind': [0, 1, 2, 4, 7, 8], 'target_r
Linha 2: {'uid': '0000589012', 'title': "Why Don't They Just Quit? DVD Roundtable Discussion: What Families and Friends need to Know

Total de registros de teste: 970237

```
# Limpar e Transformar tst.json.gz no mesmo critério do trn.json
import json
import gzip

# Caminho do arquivo original de teste
test_data_path = "/content/drive/MyDrive/Fase3/dataset/LF-Amazon-1.3M/tst.json.gz"

# Caminho do arquivo de teste limpo
cleaned_test_path = "/content/cleaned_tst.json"

# Lista para armazenar os dados limpos
cleaned_data = []

# Carregar e processar os dados
try:
    with gzip.open(test_data_path, 'rt', encoding='utf-8') as infile:
        for line in infile:
            record = json.loads(line)
            title = record.get('title', '').strip()
            content = record.get('content', '').strip()

            # Manter apenas registros com 'title' e 'content' não vazios
            if title and content:
                cleaned_record = {
                    'prompt': title,
                    'response': content
                }
                cleaned_data.append(cleaned_record)

# Salvar os dados limpos em um novo arquivo JSON
with open(cleaned_test_path, 'w') as outfile:
    json.dump(cleaned_data, outfile, ensure_ascii=False, indent=2)

print(f"Total de registros limpos: {len(cleaned_data)}")
print(f"Arquivo limpo salvo em: {cleaned_test_path}")

except Exception as e:
    print(f"Erro ao processar o arquivo de teste: {e}")
```

↻ Total de registros limpos: 599743
Arquivo limpo salvo em: /content/cleaned_tst.json

🚩 1. Carregar os Dados de Teste e Ground Truth
import json

```
# Caminho do arquivo de teste limpo
test_data_path = "/content/cleaned_tst.json"
```

```
# Carregar os dados de teste
with open(test_data_path, 'r') as file:
    test_data = json.load(file)
```

```
print(f"Total de registros de teste: {len(test_data)}")
print(test_data[:2]) # Visualizar os dois primeiros registros
```

↻ Total de registros de teste: 599743
[{'prompt': 'Girls Ballet Tutu Zebra Hot Pink', 'response': 'Tutu'}, {'prompt': 'Ballet Dress-Up Fairy Tutu', 'response': 'This ador

✓ 🧠 Carregar os Modelos Pré-Treinado e Fine-Tuned

```
# 🧠 2. Carregar os Modelos Pré-Treinado e Fine-Tuned
from transformers import AutoTokenizer, AutoModelForCausalLM
```

```
# Caminhos dos modelos atualizados
pretrained_model_path = "meta-llama/llama-3.2-1B" # nosso modelo pré-treinado utilizado
fine_tuned_model_path = "./llama-fine-tuned"

# Carregar tokenizer
tokenizer = AutoTokenizer.from_pretrained(fine_tuned_model_path)

# Carregar modelo pré-treinado
pretrained_model = AutoModelForCausalLM.from_pretrained(pretrained_model_path, device_map="auto")

# Carregar modelo fine-tuned
fine_tuned_model = AutoModelForCausalLM.from_pretrained(fine_tuned_model_path, device_map="auto")

print("✅ Modelos carregados com sucesso!")
```

✅ Modelos carregados com sucesso!

Gerar Respostas para Ambos os Modelos

```
# 3. Gerar Respostas para Ambos os Modelos
def generate_response(model, prompt, tokenizer, max_length=700):
    inputs = tokenizer(prompt, return_tensors="pt", truncation=True, max_length=max_length)
    outputs = model.generate(**inputs, max_length=max_length, pad_token_id=tokenizer.eos_token_id)
    return tokenizer.decode(outputs[0], skip_special_tokens=True)

# Exemplo de geração de respostas
sample_prompt = test_data[0]['prompt']

pretrained_response = generate_response(pretrained_model, sample_prompt, tokenizer)
fine_tuned_response = generate_response(fine_tuned_model, sample_prompt, tokenizer)

print(f"Prompt: {sample_prompt}")
print(f"Resposta Pré-Treinada: {pretrained_response}")
print(f"Resposta Fine-Tuned: {fine_tuned_response}")
```

```
-----
RuntimeError                                Traceback (most recent call last)
<ipython-input-9-c749531944f8> in <cell line: 10>()
      8 sample_prompt = test_data[0]['prompt']
      9
--> 10 pretrained_response = generate_response(pretrained_model, sample_prompt, tokenizer)
     11 fine_tuned_response = generate_response(fine_tuned_model, sample_prompt, tokenizer)
     12

-----
13 frames
/usr/local/lib/python3.10/dist-packages/torch/nn/functional.py in embedding(input, weight, padding_idx, max_norm, norm_type,
scale_grad_by_freq, sparse)
    2549     # remove once script supports set_grad_enabled
    2550     _no_grad_embedding_renorm_(weight, input, max_norm, norm_type)
-> 2551     return torch.embedding(weight, input, padding_idx, scale_grad_by_freq, sparse)
    2552
    2553

RuntimeError: Expected all tensors to be on the same device, but found at least two devices, cuda:0 and cpu! (when checking
argument for argument index in method wrapper_CUDA_index_select)
```

Próximas etapas: [Explicar o erro](#)

Solução

Para resolver esse problema, devemos garantir que todos os elementos (modelo, tokenizer, e inputs) estejam no mesmo dispositivo. Vamos ajustar o código para mover os dados para o mesmo dispositivo do modelo.

```
import torch

# Função para gerar respostas
def generate_response(model, prompt, tokenizer, device):
    # Colocar o modelo no dispositivo correto (GPU ou CPU)
    model.to(device)

    # Tokenizar o prompt e mover os inputs para o dispositivo correto
    inputs = tokenizer(prompt, return_tensors="pt").to(device)

    # Gerar a resposta com o modelo
```

```
with torch.no_grad():
    outputs = model.generate(**inputs, max_length=307)

# Decodificar a resposta gerada
return tokenizer.decode(outputs[0], skip_special_tokens=True)

# Definir o dispositivo (GPU se disponível, caso contrário CPU)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Gerar respostas usando o modelo pré-treinado e fine-tuned
sample_prompt = test_data[0]['prompt']

pretrained_response = generate_response(pretrained_model, sample_prompt, tokenizer, device)
fine_tuned_response = generate_response(fine_tuned_model, sample_prompt, tokenizer, device)

# Exibir as respostas geradas
print("Prompt:", sample_prompt)
print("\nResposta do Modelo Pré-Treinado:\n", pretrained_response)
print("\nResposta do Modelo Fine-Tuned:\n", fine_tuned_response)
```

```

➤ Setting `pad_token_id` to `eos_token_id`:128001 for open-end generation.
Setting `pad_token_id` to `eos_token_id`:128001 for open-end generation.
Prompt: Girls Ballet Tutu Zebra Hot Pink

```

[illegible]

Resposta do Modelo Fine-Tuned:
Girls Ballet Tutu Zebra Hot Pink
Girls Ballet Tutu Zebra Hot Pink
Girls Ballet Tutu Zebra Hot Pink

● Problema:

Embora o modelo fine-tuned tenha sido ajustado, ele também está gerando respostas **repetitivas** e não está adicionando informações novas ou contextuais ao prompt.

▼ 🔍 Possíveis Causas

1. 🕒 **Treinamento Insuficiente:**
 - **3 épocas** de treinamento podem não ter sido suficientes para que o modelo aprenda a gerar descrições detalhadas e variadas.
2. 📏 **Max Length:**
 - O `max_length` pode estar limitando a geração de respostas mais completas.
 - Sugerimos ajustar para um valor ligeiramente maior, como **350 ou 400 tokens**.
3. 📊 **Dados de Treinamento:**
 - Se os dados de treinamento forem limitados em **variedade** ou **complexidade**, o modelo terá dificuldades em gerar respostas criativas e diversificadas.
4. ⚙️ **Parâmetros de Geração:**
 - `temperature` e `top_p` podem estar muito baixos, fazendo com que o modelo gere respostas previsíveis e repetitivas.
 - Ajustar esses parâmetros pode ajudar a diversificar a geração.

Próximos Passos

✓ 1. Re-treinar o Modelo:

- Aumentar o número de épocas e ajustar os parâmetros de treinamento.

✓ 2. Ajustar Parâmetros de Geração:

- Experimentar diferentes valores para `temperature` (ex.: 0.7 a 1.0) e `top_p` (ex.: 0.9).

✓ 3. Aumentar `max_length`:

- Configurar `max_length` para **1024 tokens** para acomodar respostas mais completas.

✓ 4. Gerar Novas Respostas:

- Comparar novamente as respostas geradas com o **Ground Truth**.

✓ 5. Avaliar Métricas:

- Calcular **métricas BLEU e ROUGE** para verificar se houve melhora no desempenho.

🚀 **Vamos continuar aprimorando o modelo!** 😊

```
!pip install transformers peft bitsandbytes accelerate wandb
```

↔ [Mostrar saída oculta](#)

▼ 🚀 Retreinar

```
import torch
torch.cuda.empty_cache()

import json
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM, TrainingArguments, Trainer, BitsAndBytesConfig
from peft import get_peft_model, LoraConfig, TaskType
from datasets import Dataset
import wandb

# 🚀 Inicializar WandB com um novo nome
wandb.init(project="llama3-finetuning", name="retrain_llama3_v2")

# 📁 Caminho dos dados
train_data_path = "/content/reduced_40k_trn.json"

# 📖 Carregar dados de treinamento
with open(train_data_path, 'r') as file:
    train_data = json.load(file)
dataset = Dataset.from_list(train_data)

# ✂ Função de Preprocessamento
def preprocess_data(example):
    combined = f"### Prompt: {example['prompt']}\n### Response: {example['response']}"
    tokens = tokenizer(combined, truncation=True, max_length=1000, padding="max_length")
    return {
        "input_ids": tokens["input_ids"],
        "attention_mask": tokens["attention_mask"],
        "labels": tokens["input_ids"] # Certifique-se de incluir os labels corretos
    }

# 💎 Configuração de Quantização em 4 Bits
quantization_config = BitsAndBytesConfig(load_in_4bit=True)

# 🐾 Carregar Modelo e Tokenizer
model_name = "meta-llama/Llama-3.2-1B"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(model_name, device_map="auto", quantization_config=quantization_config)

# Definir pad_token como eos_token
tokenizer.pad_token = tokenizer.eos_token

# 🛠 Configurar LoRA para Fine-Tuning
lora_config = LoraConfig(task_type=TaskType.CAUSAL_LM, r=16, lora_alpha=32, lora_dropout=0.1)
model = get_peft_model(model, lora_config)

# 🧹 Limpar Cache da GPU
torch.cuda.empty_cache()

# 🇺🇸 Aplicar Preprocessamento
```

```

# 🚦 Pré-processamento
tokenized_dataset = dataset.map(preprocess_data, batched=True)

# 📌 Adicionar Labels
tokenized_dataset = tokenized_dataset.map(lambda x: {"labels": x["input_ids"]})

# 🚀 Configurações de Treinamento
training_args = TrainingArguments(
    output_dir="./llama-fine-tuned-v2",
    eval_strategy="epoch",
    save_strategy="epoch",
    logging_strategy="epoch",
    num_train_epochs=5,
    per_device_train_batch_size=2, # Ajuste para evitar OOM
    per_device_eval_batch_size=2,
    learning_rate=2e-5,
    warmup_steps=500,
    weight_decay=0.01,
    fp16=True,
    logging_dir="./logs-v2",
    logging_steps=10,
    save_total_limit=2,
    load_best_model_at_end=True,
    report_to="wandb"
)

# 🚀 Inicializar o Trainer
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_dataset,
    eval_dataset=tokenized_dataset,
)

# 🚦 Iniciar o Treinamento
trainer.train()

# 💾 Salvar o Modelo Fine-Tuned
trainer.save_model("./llama-fine-tuned-v2")
tokenizer.save_pretrained("./llama-fine-tuned-v2")

print("✅ Treinamento concluído e modelo salvo com sucesso!")

```

Finishing last run (ID:zhoubchs) before initializing another...

View run **retrain_llama3_v2** at: <https://wandb.ai/abracord2022-abracord/llama3-finetuning/runs/zhoubchs>

View project at: <https://wandb.ai/abracord2022-abracord/llama3-finetuning>

Synced 5 W&B file(s), 0 media file(s), 0 artifact file(s) and 0 other file(s)

Find logs at: ./wandb/run-20241215_175822-zhoubchs/logs

Successfully finished last run (ID:zhoubchs). Initializing new run:

Tracking run with wandb version 0.18.7

Run data is saved locally in /content/wandb/run-20241215_175855-u41e4s4q

Syncing run **retrain_llama3_v2** to [Weights & Biases \(docs\)](#)

View project at <https://wandb.ai/abracord2022-abracord/llama3-finetuning>

View run at <https://wandb.ai/abracord2022-abracord/llama3-finetuning/runs/u41e4s4q>

Map: 100% 40000/40000 [00:28<00:00, 1388.26 examples/s]

Map: 100% 40000/40000 [00:02<00:00, 14592.79 examples/s]

ValueError Traceback (most recent call last)

[<ipython-input-16-556523691bb7>](#) in <cell line: 83>()

```

81
82 # 🚦 Iniciar o Treinamento
---> 83 trainer.train()
84
85 # 💾 Salvar o Modelo Fine-Tuned

```

22 frames

[/usr/local/lib/python3.10/dist-packages/transformers/models/llama/modeling_llama.py](#) in forward(self, hidden_states, attention_mask, position_ids, past_key_value, output_attentions, use_cache, cache_position, position_embeddings, **kwargs)

```

518
519
--> 520     bsz, q_len, _ = hidden_states.size()
521
522     query_states = self.q_proj(hidden_states)

```

ValueError: not enough values to unpack (expected 3, got 2)

Próximas etapas: [Explicar o erro](#)

```
print(tokenized_dataset[0])
```

```
vo. MiracleEar: ME 1; ME 2; ME 3; ME 4. Hansaton: Auriga; Veneto; Lumeo; Cemia; Velvet; Espria; Loona.", 'input_ids': 128000, 'at
```

🔍 Diagnóstico

input_ids: Esperado ser uma lista de tokens, mas está como um valor único 128000.

attention_mask: Também deveria ser uma lista, mas está como 1.

labels: Devem ser uma cópia exata de input_ids, mas estão com um valor incorreto.

✓ Adicionar format_dataset em title e content

🎯 Como o foco é no título e na descrição, precisamos treinar o modelo com esse enfoque! Assim evitaremos aquelas respostas sem contexto.

```
import json
import gzip

# Caminho do arquivo original
data_path = "/content/drive/MyDrive/Fase3/dataset/LF-Amazon-1.3M/trn.json.gz"
# Caminho para salvar o dataset formatado
output_path = "/content/formatted_trn.json"

def format_dataset(data_path, output_path):
    """Formats the dataset for fine-tuning a Q&A model, skipping items with empty content.

    Args:
        data_path: Path to the original dataset.
        output_path: Path to save the formatted dataset.
    """
    formatted_data = []
    total_lines = 0
    valid_lines = 0

    try:
        # Ler o dataset compactado linha por linha
        with gzip.open(data_path, 'rt', encoding='utf-8') as f:
            for line in f:
                total_lines += 1
                try:
                    item = json.loads(line)
                    title = item.get('title', '').strip()
                    content = item.get('content', '').strip()

                    # Garantir que title e content não estão vazios
                    if title and content:
                        formatted_item = {
                            "instruction": "Describe the product based on the given title.",
                            "input_text": f"{title}",
                            "response": f"The product '{title}' is described as: {content}"
                        }
                        formatted_data.append(formatted_item)
                        valid_lines += 1

                except json.JSONDecodeError as e:
                    print(f"Linha {total_lines} ignorada devido a erro de decodificação: {e}")

        # Salvar o dataset formatado
        with open(output_path, 'w', encoding='utf-8') as f:
            json.dump(formatted_data, f, ensure_ascii=False, indent=2)

        print(f"\n✅ Dataset formatado salvo em: {output_path}")
        print(f"📊 Total de linhas processadas: {total_lines}")
        print(f"📄 Total de registros formatados: {valid_lines}")

    except Exception as e:
        print(f"❌ Erro ao processar o arquivo: {e}")

# Executar a função
format_dataset(data_path, output_path)
```



```
✅ Dataset formatado salvo em: /content/formatted_trn.json
📊 Total de linhas processadas: 2248619
📄 Total de registros formatados: 1390403
```

✓ Retreinar com dataset reformatado

Código para Reduzir e Dividir o Dataset

```
import json
import random
from datasets import Dataset

# Caminho do dataset formatado
formatted_data_path = "/content/formatted_trn.json"
reduced_data_path = "/content/reducedformatted_40k_trn.json"

# Número desejado de registros
desired_size = 40000

# Carregar os dados formatados
with open(formatted_data_path, 'r') as file:
    data = json.load(file)

print(f"Total de registros disponíveis: {len(data)}")

# Embaralhar os dados para evitar viés
random.shuffle(data)

# Reduzir ao tamanho desejado
data_reduced = data[:desired_size]
print(f"Total de registros após redução: {len(data_reduced)}")

# Salvar o novo conjunto reduzido
with open(reduced_data_path, 'w') as file:
    json.dump(data_reduced, file, ensure_ascii=False, indent=2)

print(f"Dataset reduzido salvo em: {reduced_data_path}")


# Criar o Dataset do Hugging Face
dataset = Dataset.from_list(data_reduced)


# Dividir em 80% treino e 20% validação
dataset_split = dataset.train_test_split(test_size=0.2)
train_dataset = dataset_split['train']
eval_dataset = dataset_split['test']


print(f"Total de registros de treino: {len(train_dataset)}")
print(f"Total de registros de validação: {len(eval_dataset)}")


➡ Total de registros disponíveis: 1390403
Total de registros após redução: 40000
Dataset reduzido salvo em: /content/reducedformatted_40k_trn.json
Total de registros de treino: 32000
Total de registros de validação: 8000

import json
import torch
from transformers import (
    AutoTokenizer,
    AutoModelForCausalLM,
    Trainer,
    TrainingArguments,
    BitsAndBytesConfig,
    DataCollatorForSeq2Seq
)
from peft import get_peft_model, LoraConfig, TaskType
from datasets import load_dataset

#  Inicializar WandB com um novo nome
wandb.init(project="llama3-finetuning", name="retrain_llama3_v2")

#  Caminho dos dados
train_data_path = "/content/reducedformatted_40k_trn.json"

#  Carregar dados de treinamento
with open(train_data_path, 'r') as file:
    train_data = json.load(file)
dataset = Dataset.from_list(train_data)

#  Função de Preprocessamento
def preprocess_data(example):
    title = example.get('title', '')
```



```

content = example.get('content', '')
combined = f"### Title: {title}\n### Description: {content}"

tokens = tokenizer(combined, truncation=True, max_length=1024, padding="max_length")

return {
    "input_ids": tokens["input_ids"],
    "attention_mask": tokens["attention_mask"],
    "labels": tokens["input_ids"]
}

# 📌 Configuração de Quantização em 4 Bits
quantization_config = BitsAndBytesConfig(load_in_4bit=True)

# 🐾 Carregar Modelo e Tokenizer
model_name = "meta-llama/Llama-3.2-1B"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(model_name, device_map="auto", quantization_config=quantization_config)

# Definir pad_token como eos_token
tokenizer.pad_token = tokenizer.eos_token

# 🛠 Configurar LoRA para Fine-Tuning
lora_config = LoraConfig(task_type=TaskType.CAUSAL_LM, r=16, lora_alpha=32, lora_dropout=0.1)
model = get_peft_model(model, lora_config)

# 🧹 Limpar Cache da GPU
torch.cuda.empty_cache()

# 🎨 Aplicar Pré-processamento
tokenized_dataset = dataset.map(preprocess_data, batched=False) #batch=false

# 💡 Adicionar Labels
tokenized_dataset = tokenized_dataset.map(lambda x: {"labels": x["input_ids"]})

# 🚀 Configurações de Treinamento
training_args = TrainingArguments(
    output_dir="./llama3-fine-tuned-v2",
    num_train_epochs=5, # Aumentar para 5 épocas
    per_device_train_batch_size=2, # Ajustar para evitar OOM (considerando max_length de 1024)
    per_device_eval_batch_size=2,
    gradient_accumulation_steps=4, # Acumular gradientes para simular batch maior
    eval_strategy="steps", # Trocado para steps para assegurar checkpoint - lembrar de trocar para eval
    eval_steps=500,
    save_steps=500,
    save_total_limit=2, # Salvar os dois últimos checkpoints
    learning_rate=2e-4, # Ajustar taxa de aprendizado
    weight_decay=0.01,
    logging_dir="./logs2",
    logging_steps=100,
    fp16=True, # Ativar treinamento em float16 para otimização
    load_best_model_at_end=True,
    report_to=["wandb"], # Reportar ao WandB
    run_name="llama3_retrain_v2" # Novo nome para evitar sobreposição
)

# **Collator para Seq2Seq**
data_collator = DataCollatorForSeq2Seq(tokenizer, model=model)

# Verificar um exemplo para confirmar a correção
print(tokenized_dataset[0])

# 🏠 Inicializar o Trainer
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_dataset,
    eval_dataset=tokenized_dataset,
)

# 🏁 Iniciar o Treinamento
trainer.train()

# 💾 Salvar o Modelo Fine-Tuned
trainer.save_model("./llama3-fine-tuned-v2")
tokenizer.save_pretrained("./llama3-fine-tuned-v2")

print("✅ Treinamento concluído e modelo salvo com sucesso!")

```

Finishing last run (ID:w6nl9x5o) before initializing another...

Run history:

eval/loss	—
eval/runtime	—
eval/samples_per_second	—
eval/steps_per_second	—
train/epoch	<div><div></div></div>
train/global_step	<div><div></div></div>
train/grad_norm	<div><div></div></div>
train/learning_rate	<div><div></div></div>
train/loss	<div><div></div></div>

Run summary:

eval/loss	0.0
eval/runtime	1497.0983
eval/samples_per_second	26.718
eval/steps_per_second	13.359
train/epoch	0.2
train/global_step	1000
train/grad_norm	0.00015
train/learning_rate	0.00019
train/loss	0

View run **retrain_llama3_v2** at: <https://wandb.ai/abracord2022-abracord/llama3-finetuning/runs/w6nl9x5o>
View project at: <https://wandb.ai/abracord2022-abracord/llama3-finetuning>
Synced 5 W&B file(s), 0 media file(s), 0 artifact file(s) and 0 other file(s)
Find logs at: ./wandb/run-20241215_204423-w6nl9x5o/logs
Successfully finished last run (ID:w6nl9x5o). Initializing new run:
Tracking run with wandb version 0.18.7
Run data is saved locally in /content/wandb/run-20241215_215224-bn4a5o96
Syncing run **retrain_llama3_v2** to [Weights & Biases](#) ([docs](#))
View project at <https://wandb.ai/abracord2022-abracord/llama3-finetuning>
View run at <https://wandb.ai/abracord2022-abracord/llama3-finetuning/runs/bn4a5o96>

Map: 100% 40000/40000 [00:40<00:00, 1300.99 examples/s]
Map: 100% 40000/40000 [00:37<00:00, 1378.59 examples/s]

{'instruction': 'Describe the product based on the given title.', 'input_text': 'Aleratec TunePhonik iMX5 In-Ear Headphones 3.5mm'} [2501/25000 2:07:28 < 19:07:42, 0.33 it/s, Epoch 0.50/5]

Step	Training Loss	Validation Loss
500	0.000000	0.000001
1000	0.000000	0.000000
1500	0.000000	0.000000
2000	0.000000	0.000000

[2072/20000 02:35 < 22:27, 13.31 it/s]

KeyboardInterrupt Traceback (most recent call last)
[ipython-input-31-e53d026d5ed1](#) in <cell line: 101>()
99
100 # 🚦 Iniciar o Treinamento
--> 101 trainer.train()
102
103 # 💾 Salvar o Modelo Fine-Tuned

9 frames
[/usr/local/lib/python3.10/dist-packages/accelerate/utils/operations.py](#) in send_to_device(tensor, device, non_blocking, skip_keys)
154 device = "xpu:0"
155 try:
--> 156 return tensor.to(device, non_blocking=non_blocking)
157 except TypeError: # .to() doesn't accept non_blocking as kwarg
158 return tensor.to(device)

KeyboardInterrupt:

```
#Melhorar o prompt
prompt = "Provide a summary of the best business books recommended by successful entrepreneurs."
```

```
#Ajustar os Parâmetros do Pipeline
```

```
result = pipe_pretrained(prompt, max_new_tokens=25, temperature=0.7, top_p=0.9, repetition_penalty=1.2)[0]['generated_text']
print(f"\nResultado Ajustado:\n{result}")
#temperature: Controle a aleatoriedade (0.7 para menos criatividade).
#top_p: Use nucleação para filtrar tokens prováveis (0.9).
#repetition_penalty: Reduza repetições excessivas (1.2).
```

Resultado Ajustado:
Provide a summary of the best business books recommended by successful entrepreneurs.
Give you an overview of what's inside each book and whether or not it is worth reading in your opinion

```
#AJUSTES
import re
```

```
prompt = "Summarize the top 5 business books recommended by famous entrepreneurs."

# Gerar texto com parâmetros ajustados
result = pipe_pretrained(prompt, max_new_tokens=50, temperature=0.7, top_p=0.9, repetition_penalty=1.2)[0]['generated_text']

# Pós-processar para remover caracteres estranhos
clean_result = re.sub(r'^a-zA-Z0-9\s,!?', '', result)

print(f"\nResultado Pós-processado:\n{clean_result}")
```



Resultado Pós-processado:
Summarize the top 5 business books recommended by famous entrepreneurs.
Many successful people read a lot of good books to learn and improve themselves in many ways, especially when it comes to running th

```
# 🌟 Código Completo com Melhorias
import re

# Prompt ajustado
prompt = "List the titles of the top 5 business books recommended by successful entrepreneurs."

# Geração com parâmetros ajustados
result = pipe_pretrained(prompt, max_new_tokens=50, temperature=0.7, top_p=0.9, repetition_penalty=1.2)[0]['generated_text']

# Pós-processar para remover caracteres estranhos
clean_result = re.sub(r'^a-zA-Z0-9\s,!?', '', result)

# Pós-processamento para remover texto após o primeiro ponto final
clean_result = re.split(r'\.', result)[0] + '.'

print(f"\nResultado Final:\n{clean_result}")
```



Resultado Final:
List the titles of the top 5 business books recommended by successful entrepreneurs.

✓ Plano de Ação

- ◆ 1. Dividir o Dataset Original
- ◆ 2. Filtrar os dados com os labels.txt dados
- ◆ 3. Salvar os Datasets Divididos
- ◆ 4. Continuar com a Tokenização

```
# ◆ 2. Filtragem dos Dados
# Carregar os labels de filtragem
with open("/content/filter_labels_train.txt", 'r') as f:
    train_labels_filter = set(f.read().splitlines())

with open("/content/filter_labels_test.txt", 'r') as f:
    test_labels_filter = set(f.read().splitlines())

# Filtrar o dataset de treino
train_dataset = train_dataset.filter(lambda example: example['prompt'] in train_labels_filter)

# Filtrar o dataset de avaliação
eval_dataset = eval_dataset.filter(lambda example: example['prompt'] in test_labels_filter)

#Verificação das Quantidades após a filtragem:
print(f"Total de exemplos de treino após filtragem: {len(train_dataset)}")
print(f"Total de exemplos de avaliação após filtragem: {len(eval_dataset)}")
```



Próximos Passos

1. Inicializar o Trainer
2. Iniciar o Treinamento
3. Monitorar o Treinamento
4. Salvar o Modelo Treinado

5. Gerar Respostas com o Modelo Fine-Tuned
6. Comparar com os Resultados Esperados

▼ Preparando para monitorar em tempo real

 Usando TensorBoard e Weights & Biases (W&B)

```
# 3 Monitorar o Treinamento
#Instalar o TensorBoard
%load_ext tensorboard

#Iniciar o TensorBoard no Colab:
%tensorboard --logdir ./logs

#Visualizar os Gráficos:
import wandb
import random

# start a new wandb run to track this script
wandb.init(
    # set the wandb project where this run will be logged
    project="huggingface-project",

    # track hyperparameters and run metadata
    config={
        "learning_rate": 0.02,
        "architecture": "CNN",
        "dataset": "CIFAR-100",
        "epochs": 5,
```