🚀 Relembrando o Desafio do Tech Challenge

TechChallenge Fase 3 Pós-Tech em IA para DEVS FIAP

INTEGRANTES DO GRUPO 18

Beatriz Cardoso Cunha

Francisco Giuan Miranda Ferreira

Maurício Lachaitis da Silva

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
```

```
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->torch->bitsandbytes) (3.0
Downloading bitsandbytes-0.45.0-py3-none-manylinux_2_24_x86_64.whl (69.1 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 69.1/69.1 MB 32.5 MB/s eta 0:00:00
Downloading loralib-0.1.2-py3-none-any.whl (10 kB)
Installing collected packages: loralib, bitsandbytes
Successfully installed bitsandbytes-0.45.0 loralib-0.1.2
Requirement already satisfied: sentencepiece in /usr/local/lib/python3.10/dist-packages (0.2.0)
Collecting bitsandbytes-cuda117
  Downloading bitsandbytes_cuda117-0.26.0.post2-py3-none-any.whl.metadata (6.3 kB)
Downloading bitsandbytes_cuda117-0.26.0.post2-py3-none-any.whl (4.3 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 4.3/4.3 MB 33.2 MB/s eta 0:00:00
Installing collected packages: bitsandbytes-cuda117
Successfully installed bitsandbytes-cuda117-0.26.0.post2
```

```
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')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
#!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  \
0  0000031909                           Girls Ballet Tutu Neon Pink
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

                                             content  \
0  High quality 3 layer ballet tutu. 12 inches in...
1
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]
3                          [1.0, 1.0, 1.0, 1.0, 1.0]
4                                              [1.0]
Linha 1: {'uid': '0000031909', 'title': 'Girls Ballet Tutu Neon Pink', 'content': 'High quality 3 layer ballet tutu. 12 inches in le
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_inc
Linha 4: {'uid': '0001360000', 'title': "Mog's Kittens", 'content': 'Judith Kerr&#8217;s best&#8211;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 E
Linha 10: {'uid': '0001713086', 'title': 'Dr. Seuss ABC (Dr.Seuss Classic Collection) (Spanish Edition)', 'content': '', 'target_inc
```

🪄 Limpeza do Dataset

O que queremos fazer?

Um modelo de Q&A flexível

```
#Limpeza de dataset
import json
import re
import gzip

# 📁 Caminhos dos arquivos no Google Drive
file_path = "/content/drive/MyDrive/Fase3/dataset/LF-Amazon-1.3M/trn.json.gz"
cleaned_file_path = "/content/drive/MyDrive/Fase3/trn_cleaned.json"
formatted_file_path = "/content/drive/MyDrive/Fase3/trn_formatted.json"

# 🔍 Função de limpeza de texto
def clean_text(text):
    """Limpa o texto removendo caracteres especiais e múltiplos espaços."""
    text = text.strip()  # Remover espaços extras
    text = re.sub(r'[^a-zA-Z0-9.,!?;:\'\"\s]', '', text)  # Manter letras, números e pontuações comuns
    text = re.sub(r'\s+', ' ', text)  # Substituir múltiplos espaços por um único espaço
    return text

# 🛠 Função para limpeza e formatação do dataset
def clean_and_format_dataset(input_path, cleaned_output_path, formatted_output_path):
    """Limpa e formata o dataset no padrão Q&A."""
    total_cleaned = 0
    formatted_data = []

    try:
        # Ler o dataset compactado linha por linha
        with gzip.open(input_path, 'rt', encoding='utf-8') as infile, open(cleaned_output_path, 'w', encoding='utf-8') as cleaned_outfi
            cleaned_outfile.write('[')  # Início do array JSON
            first_record = True

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

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

                if title and content:
                    cleaned_title = clean_text(title)
                    cleaned_content = clean_text(content)

                    cleaned_record = {
                        'title': cleaned_title,
                        'content': cleaned_content
                    }

                    # Escrever o registro limpo no arquivo
                    if not first_record:
                        cleaned_outfile.write(',\n')  # Separador entre registros
                    json.dump(cleaned_record, cleaned_outfile, ensure_ascii=False)

                    # Formatar para o padrão Q&A
                    formatted_item = {
                        "instruction": "Answer the user's question based on the product information provided.",
                        "input_text": cleaned_title,
                        "response": cleaned_content
                    }

                    formatted_data.append(formatted_item)
                    total_cleaned += 1
                    first_record = False

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

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

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

    print(f"Total de registros limpos e formatados: {total_cleaned}")
    print(f"Arquivo limpo salvo em: {cleaned_output_path}")
    print(f"Arquivo formatado salvo em: {formatted_output_path}")

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

# 🚀 Executar a função
clean_and_format_dataset(file_path, cleaned_file_path, formatted_file_path)
```

```
Total de registros limpos e formatados: 1390403
Arquivo limpo salvo em: /content/drive/MyDrive/Fase3/trn_cleaned.json
Arquivo formatado salvo em: /content/drive/MyDrive/Fase3/trn_formatted.json
```

```
#import json

# Caminho do arquivo limpo
cleaned_file_path = "/content/drive/MyDrive/Fase3/trn_formatted.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:
{
  "instruction": "Answer the user's question based on the product information provided.",
  "input_text": "Girls Ballet Tutu Neon Pink",
  "response": "High quality 3 layer ballet tutu. 12 inches in length"
}

Registro 2:
{
  "instruction": "Answer the user's question based on the product information provided.",
  "input_text": "Mog's Kittens",
  "response": "Judith Kerr8217;s best8211;selling adventures of that endearing and exasperating cat Mog have entertained children fo
}

Registro 3:
{
  "instruction": "Answer the user's question based on the product information provided.",
  "input_text": "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:
{
  "instruction": "Answer the user's question based on the product information provided.",
  "input_text": "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:
{
  "instruction": "Answer the user's question based on the product information provided.",
  "input_text": "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 comprimentos 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.

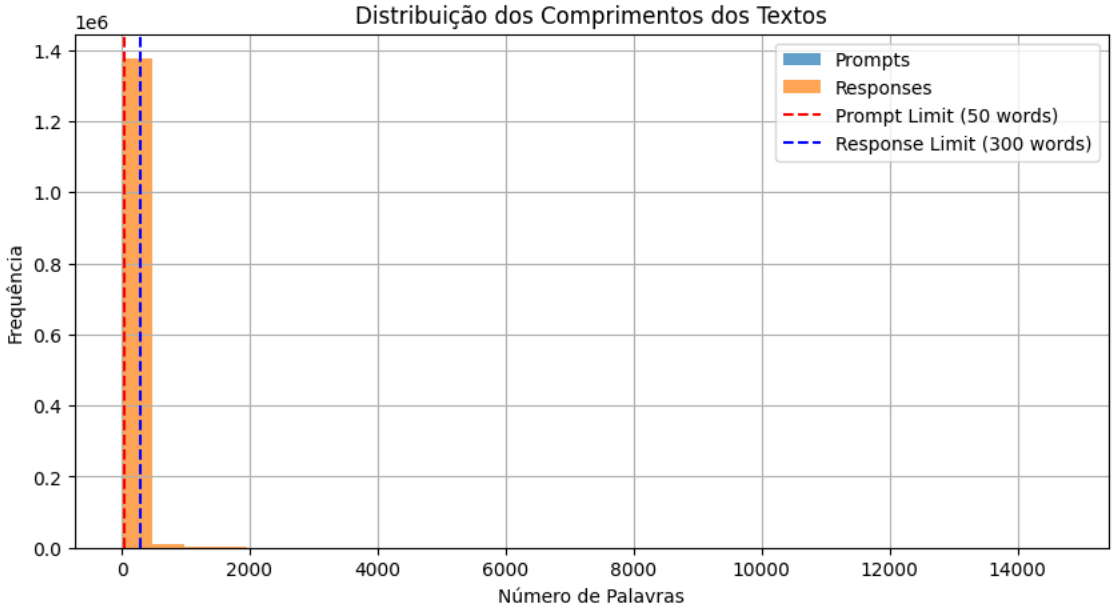
Se muitos textos ultrapassam esses limites, pode ser necessário aumentar o max_length ou ajustar os dados de entrada.

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

# 📁 Caminho do arquivo no Google Drive
file_path = "/content/drive/MyDrive/Fase3/trn_formatted.json"

# 📏 Limites definidos para a análise
prompt_limit = 50
response_limit = 300

# 📊 Variáveis para armazenar os comprimentos dos textos
prompt_lengths = []
response_lengths = []

# 🔍 Analisar o comprimento dos prompts e responses
try:
    with open(file_path, 'r', encoding='utf-8') as file:
```

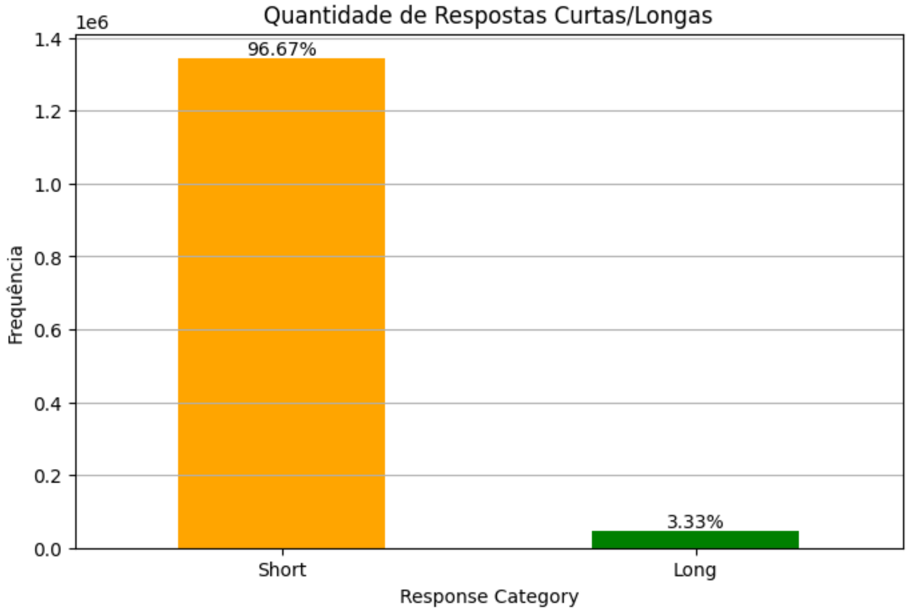
```
        data = json.load(file)

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

# 📈 Criar DataFrame com os dados coletados
df = pd.DataFrame({
    'Prompt Length': prompt_lengths,
    'Response Length': response_lengths
})

# 🏷️ Categorizar os prompts e responses como 'Short' ou 'Long'
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 com os resultados
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 com os resultados no Colab
print("📊 Tabela de análise com porcentagens:")
display(summary_df)

# 📈 Gráficos
# 1️⃣ Distribuição dos comprimentos dos prompts e responses
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=f'Prompt Limit ({prompt_limit} words)')
plt.axvline(response_limit, color='blue', linestyle='--', label=f'Response Limit ({response_limit} words)')
plt.title('Distribuição dos Comprimentos dos Textos')
plt.xlabel('Número de Palavras')
plt.ylabel('Frequência')
plt.legend()
plt.grid(True)
plt.show()

# 2️⃣ Quantidade de respostas curtas/longas 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.xticks(rotation=0)
plt.grid(axis='y')
plt.show()

# 3️⃣ Quantidade de prompts curtos/longos com porcentagens
plt.figure(figsize=(8, 5))
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.xticks(rotation=0)
plt.grid(axis='y')
plt.show()
```

📊 Tabela de análise com porcentagens:

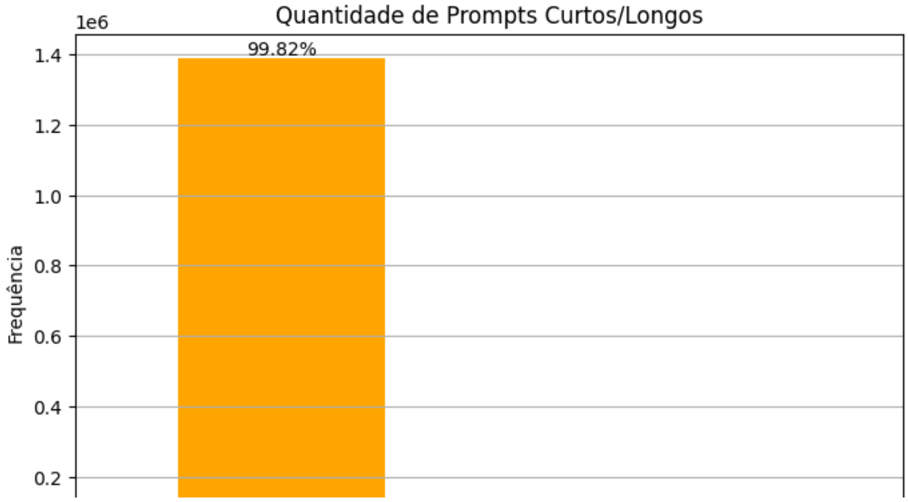
	Category	Count	Percentage
0	Short Prompts	1387846	99.816096
1	Long Prompts	2557	0.183904
2	Short Responses	1344085	96.668736
3	Long Responses	46318	3.331264




```
<ipython-input-9-0ff583a59085>:76: 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-9-0ff583a59085>:87: 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')
```



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.

```python
import json
import random

# 📁 Caminhos dos arquivos
formatted_file_path = "/content/drive/MyDrive/Fase3/trn_formatted.json"       # Caminho do arquivo formatado
reduced_file_path = "/content/drive/MyDrive/Fase3/trn_reduced_40k.json"       # Caminho para salvar o arquivo reduzido

# 🎯 Número desejado de registros
desired_size = 40000

try:
    # 📥 Carregar os dados formatados existentes
    with open(formatted_file_path, 'r', encoding='utf-8') as file:
        data = json.load(file)

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

    # ⚠️ Verificar se o dataset é maior que o tamanho desejado
    if total_records < desired_size:
        print(f"⚠️ O dataset contém apenas {total_records} registros. Não é possível reduzir para {desired_size}.")
    else:
        # 🔀 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)

        # ✅ Mensagem de confirmação
        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/drive/MyDrive/Fase3/trn_reduced_40k.json
```

📝 Testar o Modelo Pré-Treinado

```
from huggingface_hub import login

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

```
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM

# 📁 Caminhos dos arquivos
test_data_path = "/content/drive/MyDrive/Fase3/trn_reduced_40k.json"       # Dataset reduzido
model_name = "meta-llama/Llama-3.2-1B"                                     # Modelo pré-treinado (ajuste conforme necessário)

# 🔄 Carregar o tokenizer e o modelo pré-treinado
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(model_name, torch_dtype=torch.float16, device_map="auto")

# 📝 Função para gerar respostas
def generate_response(prompt, max_length=200):
    inputs = tokenizer(prompt, return_tensors="pt").to(model.device)
    outputs = model.generate(**inputs, max_length=max_length, num_return_sequences=1)
    return tokenizer.decode(outputs[0], skip_special_tokens=True)

# 📥 Carregar alguns exemplos do dataset de teste
import json

with open(test_data_path, 'r', encoding='utf-8') as file:
    test_data = json.load(file)

# 🧪 Testar com 3 exemplos
for i in range(3):
    prompt = test_data[i]['input_text']
    print(f"🔹 **Prompt {i+1}:** {prompt}")

    response = generate_response(prompt)
    print(f"🧠 **Resposta do Modelo Pré-Treinado:** {response}\n")
```

```
/usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as :
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
tokenizer_config.json: 100%   50.5k/50.5k [00:00<00:00, 3.80MB/s]
tokenizer.json: 100%   9.09M/9.09M [00:00<00:00, 45.5MB/s]
special_tokens_map.json: 100%   301/301 [00:00<00:00, 26.3kB/s]
config.json: 100%   843/843 [00:00<00:00, 75.5kB/s]
model.safetensors: 100%   2.47G/2.47G [00:58<00:00, 42.2MB/s]
generation_config.json: 100%   185/185 [00:00<00:00, 15.1kB/s]
Setting `pad_token_id` to `eos_token_id`:None for open-end generation.
🔹 **Prompt 1:** Weave Aide Straight Edge Pomade 4 oz.
Setting `pad_token_id` to `eos_token_id`:None for open-end generation.
🧠 **Resposta do Modelo Pré-Treinado:** Weave Aide Straight Edge Pomade 4 oz. - 1.13 oz.
Weave Aide Straight Edge Pomade is a 4 oz. tube of pomade that will straighten, define, and hold your hair in place. The Weave Aide

🔹 **Prompt 2:** Buck Wear Inc. RedMultiTasking Short Sleeve Tee
Setting `pad_token_id` to `eos_token_id`:None for open-end generation.
🧠 **Resposta do Modelo Pré-Treinado:** Buck Wear Inc. RedMultiTasking Short Sleeve Tee. 100% Cotton. Size: S, M, L, XL, 2XL. Color

🔹 **Prompt 3:** Trefethen's Index Cards: Forty Years of Notes about People, Words and Mathematics
🧠 **Resposta do Modelo Pré-Treinado:** Trefethen's Index Cards: Forty Years of Notes about People, Words and Mathematics
by Michael Trefethen
Trefethen's Index Cards: Forty Years of Notes about People, Words and Mathematics Cover  - Comment on this title and you could win 1
Michael Trefethen's index cards contain notes on people, words, and mathematics, mostly from the past forty years. The index cards a
```

```
#📄 Código Ajustado com Configuração de pad_token e eos_token
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM
```

```
import json

# 📁 Caminhos dos arquivos
test_data_path = "/content/drive/MyDrive/Fase3/trn_reduced_40k.json"       # Dataset reduzido
model_name = "meta-llama/Llama-3.2-1B"                                     # Modelo pré-treinado

# 🔄 Carregar o tokenizer e o modelo pré-treinado
tokenizer = AutoTokenizer.from_pretrained(model_name)

# Configurar pad_token como eos_token para evitar avisos
if tokenizer.pad_token is None:
    tokenizer.pad_token = tokenizer.eos_token

model = AutoModelForCausalLM.from_pretrained(model_name, torch_dtype=torch.float16, device_map="auto")

# 📝 Função para gerar respostas com penalidade para evitar repetições
def generate_response(prompt, max_length=700, repetition_penalty=1.2):
    inputs = tokenizer(prompt, return_tensors="pt").to(model.device)
    outputs = model.generate(
        **inputs,
        max_length=max_length,
        num_return_sequences=1,
        pad_token_id=tokenizer.pad_token_id,
        eos_token_id=tokenizer.eos_token_id,
        repetition_penalty=repetition_penalty
    )
    return tokenizer.decode(outputs[0], skip_special_tokens=True)

# 📥 Carregar alguns exemplos do dataset de teste
with open(test_data_path, 'r', encoding='utf-8') as file:
    test_data = json.load(file)

# 🧪 Testar com 3 exemplos
for i in range(3):
    prompt = test_data[i]['input_text']
    print(f"🔹 **Prompt {i+1}:** {prompt}")

    response = generate_response(prompt)
    print(f"🧠 **Resposta do Modelo Pré-Treinado:** {response}\n")
```

```
🔹 **Prompt 1:** Weave Aide Straight Edge Pomade 4 oz.
🧠 **Resposta do Modelo Pré-Treinado:** Weave Aide Straight Edge Pomade 4 oz. Bottle
This is the perfect straight edge for those who want a little bit of shine without having to worry about their hair looking too oily
The Weave Aide Straight Edge pomade works great as an all-purpose styling product, but it also gives your style that extra touch of
It contains natural oils like jojoba oil and coconut oil which help add moisture back into your strands while adding some volume at
You can use this formula either alone or mixed together depending how much hold & shine do not need - just make sure there isn't too
Just apply directly onto dampened hairs before blow drying them until completely dry; let air-dry naturally if needed later during (

🔹 **Prompt 2:** Buck Wear Inc. RedMultiTasking Short Sleeve Tee
🧠 **Resposta do Modelo Pré-Treinado:** Buck Wear Inc. RedMultiTasking Short Sleeve Tee.
Red Multi-Tasking short sleeve tee with Buck logo screen print at left chest and a 3D buckle in the center back neck of this classic

🔹 **Prompt 3:** Trefethen's Index Cards: Forty Years of Notes about People, Words and Mathematics
🧠 **Resposta do Modelo Pré-Treinado:** Trefethen's Index Cards: Forty Years of Notes about People, Words and Mathematics
In this book Trefethen describes the origins and development of his "index cards," which are used for recording thoughts on a variety
The index card is an indispensable tool to me when I'm writing down things that come into my head at random times throughout the day
This volume contains entries covering forty-five years' worth of observations ranging across numerous disciplines including philosop
```

🚀 Próximos Passos

- Teste Pré-Treino -ok
- 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

🪄 3. Configurar o Modelo LLAMA 3.2 e os Hiperparâmetros

3. Autenticar no Huggingface

```
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
```

```
Requirement already satisfied: peft in /usr/local/lib/python3.10/dist-packages (0.13.2)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-packages (from peft) (1.26.4)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from peft) (24.2)
Requirement already satisfied: psutil in /usr/local/lib/python3.10/dist-packages (from peft) (5.9.5)
Requirement already satisfied: pyyaml in /usr/local/lib/python3.10/dist-packages (from peft) (6.0.2)
Requirement already satisfied: torch>=1.13.0 in /usr/local/lib/python3.10/dist-packages (from peft) (2.5.1+cu121)
Requirement already satisfied: transformers in /usr/local/lib/python3.10/dist-packages (from peft) (4.46.3)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from peft) (4.66.6)
Requirement already satisfied: accelerate>=0.21.0 in /usr/local/lib/python3.10/dist-packages (from peft) (1.1.1)
Requirement already satisfied: safetensors in /usr/local/lib/python3.10/dist-packages (from peft) (0.4.5)
Requirement already satisfied: huggingface-hub>=0.17.0 in /usr/local/lib/python3.10/dist-packages (from peft) (0.26.5)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from huggingface-hub>=0.17.0->peft) (3.16.1)
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub>=0.17.0->peft) (202
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from huggingface-hub>=0.17.0->peft) (2.32.3)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub>=0.17.0-:
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from torch>=1.13.0->peft) (3.4.2)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from torch>=1.13.0->peft) (3.1.4)
Requirement already satisfied: sympy==1.13.1 in /usr/local/lib/python3.10/dist-packages (from torch>=1.13.0->peft) (1.13.1)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from sympy==1.13.1->torch>=1.13.0->pe
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.10/dist-packages (from transformers->peft) (2024.9.11)
Requirement already satisfied: tokenizers<0.21,>=0.20 in /usr/local/lib/python3.10/dist-packages (from transformers->peft) (0.20.3)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->torch>=1.13.0->peft) (3.0.2
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->huggingface-hub>=
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->huggingface-hub>=0.17.0->pef
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->huggingface-hub>=0.17.
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->huggingface-hub>=0.17.
```

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

```
Requirement already satisfied: peft in /usr/local/lib/python3.10/dist-packages (0.13.2)
Collecting peft
  Downloading peft-0.14.0-py3-none-any.whl.metadata (13 kB)
Requirement already satisfied: transformers in /usr/local/lib/python3.10/dist-packages (4.46.3)
Collecting transformers
  Downloading transformers-4.47.0-py3-none-any.whl.metadata (43 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 43.5/43.5 kB 2.1 MB/s eta 0:00:00
Requirement already satisfied: bitsandbytes in /usr/local/lib/python3.10/dist-packages (0.45.0)
Requirement already satisfied: accelerate in /usr/local/lib/python3.10/dist-packages (1.1.1)
Collecting accelerate
  Downloading accelerate-1.2.1-py3-none-any.whl.metadata (19 kB)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-packages (from peft) (1.26.4)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from peft) (24.2)
Requirement already satisfied: psutil in /usr/local/lib/python3.10/dist-packages (from peft) (5.9.5)
Requirement already satisfied: pyyaml in /usr/local/lib/python3.10/dist-packages (from peft) (6.0.2)
Requirement already satisfied: torch>=1.13.0 in /usr/local/lib/python3.10/dist-packages (from peft) (2.5.1+cu121)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from peft) (4.66.6)
Requirement already satisfied: safetensors in /usr/local/lib/python3.10/dist-packages (from peft) (0.4.5)
Requirement already satisfied: huggingface-hub>=0.25.0 in /usr/local/lib/python3.10/dist-packages (from peft) (0.26.5)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from transformers) (3.16.1)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.10/dist-packages (from transformers) (2024.9.11)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from transformers) (2.32.3)
Collecting tokenizers<0.22,>=0.21 (from transformers)
  Downloading tokenizers-0.21.0-cp39-abi3-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (6.7 kB)
Requirement already satisfied: typing_extensions>=4.8.0 in /usr/local/lib/python3.10/dist-packages (from bitsandbytes) (4.12.2)
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub>=0.25.0->peft) (202
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from torch>=1.13.0->peft) (3.4.2)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from torch>=1.13.0->peft) (3.1.4)
Requirement already satisfied: sympy==1.13.1 in /usr/local/lib/python3.10/dist-packages (from torch>=1.13.0->peft) (1.13.1)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from sympy==1.13.1->torch>=1.13.0->pe
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (3
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (2024.8.
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->torch>=1.13.0->peft) (3.0.2
Downloading peft-0.14.0-py3-none-any.whl (374 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 374.8/374.8 kB 10.3 MB/s eta 0:00:00
Downloading transformers-4.47.0-py3-none-any.whl (10.1 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 10.1/10.1 MB 96.7 MB/s eta 0:00:00
Downloading accelerate-1.2.1-py3-none-any.whl (336 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 336.4/336.4 kB 28.5 MB/s eta 0:00:00
Downloading tokenizers-0.21.0-cp39-abi3-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (3.0 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.0/3.0 MB 96.8 MB/s eta 0:00:00
Installing collected packages: tokenizers, accelerate, transformers, peft
  Attempting uninstall: tokenizers
    Found existing installation: tokenizers 0.20.3
    Uninstalling tokenizers-0.20.3:
      Successfully uninstalled tokenizers-0.20.3
  Attempting uninstall: accelerate
    Found existing installation: accelerate 1.1.1
    Uninstalling accelerate-1.1.1:
      Successfully uninstalled accelerate-1.1.1
  Attempting uninstall: transformers
    Found existing installation: transformers 4.46.3
    Uninstalling transformers-4.46.3:
      Successfully uninstalled transformers-4.46.3
  Attempting uninstall: peft
    Found existing installation: peft 0.13.2
    Uninstalling peft-0.13.2:
      Successfully uninstalled peft-0.13.2
Successfully installed accelerate-1.2.1 peft-0.14.0 tokenizers-0.21.0 transformers-4.47.0
WARNING: The following packages were previously imported in this runtime:
  [accelerate,peft,transformers]
You must restart the runtime in order to use newly installed versions.
```

RESTART SESSION

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

```
!pip install wandb
```

```
Requirement already satisfied: wandb in /usr/local/lib/python3.10/dist-packages (0.18.7)
Requirement already satisfied: click!=8.0.0,>=7.1 in /usr/local/lib/python3.10/dist-packages (from wandb) (8.1.7)
Requirement already satisfied: docker-pycreds>=0.4.0 in /usr/local/lib/python3.10/dist-packages (from wandb) (0.4.0)
Requirement already satisfied: gitpython!=3.1.29,>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from wandb) (3.1.43)
Requirement already satisfied: platformdirs in /usr/local/lib/python3.10/dist-packages (from wandb) (4.3.6)
Requirement already satisfied: protobuf!=4.21.0,!=5.28.0,<6,>=3.19.0 in /usr/local/lib/python3.10/dist-packages (from wandb) (4.25.5
Requirement already satisfied: psutil>=5.0.0 in /usr/local/lib/python3.10/dist-packages (from wandb) (5.9.5)
Requirement already satisfied: pyyaml in /usr/local/lib/python3.10/dist-packages (from wandb) (6.0.2)
Requirement already satisfied: requests<3,>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from wandb) (2.32.3)
Requirement already satisfied: sentry-sdk>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from wandb) (2.19.2)
```

```
Requirement already satisfied: setproctitle in /usr/local/lib/python3.10/dist-packages (from wandb) (1.3.4)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from wandb) (75.1.0)
Requirement already satisfied: typing-extensions<5,>=4.4 in /usr/local/lib/python3.10/dist-packages (from wandb) (4.12.2)
Requirement already satisfied: six>=1.4.0 in /usr/local/lib/python3.10/dist-packages (from docker-pycreds>=0.4.0->wandb) (1.17.0)
Requirement already satisfied: gitdb<5,>=4.0.1 in /usr/local/lib/python3.10/dist-packages (from gitpython!=3.1.29,>=1.0.0->wandb) (
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.0.0->wandb)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.0.0->wandb) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.0.0->wandb) (2.2.
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.0.0->wandb) (2024
Requirement already satisfied: smmap<6,>=3.0.1 in /usr/local/lib/python3.10/dist-packages (from gitdb<5,>=4.0.1->gitpython!=3.1.29,
```

```
!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="llama3.2-finetuning", name="run-2")
```

```
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-20241217_100331-x43vxcpw`
Syncing run **[run-2](#)** to [Weights & Biases](#) ([docs](#))
View project at https://wandb.ai/abracord2022-abracord/llama3.2-finetuning
View run at https://wandb.ai/abracord2022-abracord/llama3.2-finetuning/runs/x43vxcpw
Display W&B run

```
!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
```

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

```
Found existing installation: bitsandbytes 0.45.0
Uninstalling bitsandbytes-0.45.0:
  Successfully uninstalled bitsandbytes-0.45.0
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: sympy==1.13.1 in /usr/local/lib/python3.10/dist-packages (from torch->bitsandbytes) (1.13.1)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from sympy==1.13.1->torch->bitsandbyte
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 302.1 MB/s eta 0:00:00
Installing collected packages: bitsandbytes
Successfully installed bitsandbytes-0.45.0
```

🚀 Fine-Tuning v1 com Modelo LLaMA 3.2

🛠️ Configurações do Ambiente

- **Modelo**: `meta-llama/Llama-3.2-1B`
- **Framework**: Hugging Face Transformers
- **Quantização**: 4-bit com **BitsAndBytes**
- **Aceleração**: Utilização de GPU A100
- **Tamanho do Dataset**: Reduzido para **40.000 registros**
- **Formato dos Dados**:
  ```json {

  ```
  "instruction": "Answer the user's question based on the product information provided.",
  "input_text": "Título do produto",
  "response": "Conteúdo do produto"
  ```

  }

```
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()

# 1️⃣ Login no wandb (interativo)
#!wandb login

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

# 3️⃣ 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

# 4️⃣ 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)

# 5️⃣ Carregar o dataset
data_path = "/content/drive/MyDrive/Fase3/trn_reduced_40k.json"
with open(data_path, 'r') as file:
    data = json.load(file)
dataset = Dataset.from_list(data)

# 6️⃣ Preprocessar o dataset
# Preprocessar o dataset com max_length ajustado - aqui entra o estudo do max_lenght
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']

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

# 8️⃣ Configurar o treinamento com métricas
training_args = TrainingArguments(
    output_dir="./results",
    eval_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
)

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

# 🔟 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")
```

Map: 100% 40000/40000 [00:42<00:00, 1260.41 examples/s]

```
/usr/local/lib/python3.10/dist-packages/transformers/training_args.py:1575: FutureWarning: `evaluation_strategy` is deprecated and w
  warnings.warn(
```

[24000/24000 1:52:16, Epoch 3/3]

Epoch	Training Loss	Validation Loss
1	0.616200	0.619031
2	0.601100	0.616258
3	0.592400	0.615905

[2000/2000 04:18]

```
Eval Metrics: {'eval_loss': 0.6159051060676575, 'eval_runtime': 258.2077, 'eval_samples_per_second': 30.983, 'eval_steps_per_second
('./llama-fine-tuned/tokenizer_config.json',
 './llama-fine-tuned/special_tokens_map.json',
 './llama-fine-tuned/tokenizer.json')
```

📊 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)
```

🛠️ 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

Resposta do Modelo Pré-Treinado:
 Girls Ballet Tutu Zebra Hot Pink Size 4-6
Girls Ballet Tutu Zebra Hot Pink Size 4-6
Girls Ballet Tutu Zebra Hot Pink Size 4-6
Girls Ballet Tutu Zebra Hot Pink Size 4-6
Girls Ballet Tutu Zebra Hot Pink Size 4-6
Girls Ballet Tutu Zebra Hot Pink Size 4-6
Girls Ballet Tutu Zebra Hot Pink Size 4-6
Girls Ballet Tutu Zebra Hot Pink Size 4-6
Girls Ballet Tutu Zebra Hot Pink Size 4-6
Girls Ballet Tutu Zebra Hot Pink Size 4-6
Girls Ballet Tutu Zebra Hot Pink Size 4-6
Girls Ballet Tutu Zebra Hot Pink Size 4-6
Girls Ballet Tutu Zebra Hot Pink Size 4-6
Girls Ballet Tutu Zebra Hot Pink Size 4-6
Girls Ballet Tutu Zebra Hot Pink Size 4-6
Girls Ballet Tutu Zebra Hot Pink Size 4-6
Girls Ballet Tutu Zebra Hot Pink Size 4-6
Girls Ballet Tutu Zebra Hot Pink Size 4-6
Girls Ballet Tutu Zebra Hot Pink Size 4-6
Girls Ballet Tutu Zebra Hot Pink Size 4-6
Girls Ballet Tutu Zebra Hot Pink Size 4-6
Girls Ballet Tutu Zebra Hot Pink Size 4-

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-finetuning4", name="retrain_llama3_v4")

# 📁 Caminho dos dados
train_data_path = "/content/drive/MyDrive/Fase3/trn_reduced_40k.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 - Precisa corresponder À formatação do dataset
def preprocess_data(example):
    combined = f"### Instruction: {example['instruction']}\n### Input: {example['input_text']}\n### Response: {example['response']}"
    tokens = tokenizer(combined, truncation=True, max_length=1024, padding="max_length", return_tensors="pt")
    return {
        "input_ids": tokens["input_ids"].squeeze(0),          # Remove a dimensão extra
        "attention_mask": tokens["attention_mask"].squeeze(0),
        "labels": tokens["input_ids"].squeeze(0)              # Labels iguais aos input_ids
    }


# 🔹 Configuração de Quantização em 4 Bits
quantization_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_compute_dtype=torch.float16
)


# 🦙 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
tokenized_dataset = dataset.map(preprocess_data, batched=False)

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

# 🛠️ Configurações de Treinamento
training_args = TrainingArguments(
    output_dir="./llama-fine-tuned-v4",
    run_name="retrain_llama3_v4",  # Adicione um nome diferente para o run
    eval_strategy="epoch",
    save_strategy="epoch",
    logging_strategy="epoch",
    num_train_epochs=5,
    per_device_train_batch_size=2,  # Ajuste para evitar OOM  #POdemos ajustar para 8 com GPU A100
    per_device_eval_batch_size=2,
    learning_rate=2e-5,
    warmup_steps=500,
    weight_decay=0.01,
    fp16=True,
    logging_dir="./logs-v4",
    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-v4")
tokenizer.save_pretrained("./llama-fine-tuned-v4")

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

Finishing last run (ID:686i9bb4) before initializing another...
View run **retrain_llama3_v4** at: https://wandb.ai/abracord2022-abracord/llama3-finetuning4/runs/686i9bb4
View project at: https://wandb.ai/abracord2022-abracord/llama3-finetuning4
Synced 5 W&B file(s), 0 media file(s), 0 artifact file(s) and 0 other file(s)
Find logs at: `./wandb/run-20241217_102306-686i9bb4/logs`
Successfully finished last run (ID:686i9bb4). Initializing new run:
Tracking run with wandb version 0.18.7
Run data is saved locally in `/content/wandb/run-20241217_102847-v0kz45pd`
Syncing run **retrain_llama3_v4** to Weights & Biases (docs)
View project at https://wandb.ai/abracord2022-abracord/llama3-finetuning4
View run at https://wandb.ai/abracord2022-abracord/llama3-finetuning4/runs/v0kz45pd

Map: 100% 40000/40000 [01:04<00:00, 669.21 examples/s]

Map: 100% 40000/40000 [00:34<00:00, 1269.15 examples/s]

[100000/100000 5:49:29, Epoch 5/5]

Epoch	Training Loss	Validation Loss
1	0.485000	0.425871
2	0.424700	0.422027
3	0.421700	0.419682
4	0.419700	0.418271
5	0.418500	0.417775

✅ Treinamento concluído e modelo salvo com sucesso!

```
#Movendo a Pasta Após o Treinamento
!mv /content/llama-fine-tuned-v4 /content/drive/MyDrive/Fase3/models
```

```
import shutil
import os
```

```
# Caminho do checkpoint
checkpoint_path = "/content/drive/MyDrive/Fase3/models/checkpoint-100000"

# Caminho de destino para salvar o modelo final
final_model_path = "/content/drive/MyDrive/Fase3/models/llama-fine-tuned-v4"

# Verifica se o checkpoint existe
if os.path.exists(checkpoint_path):
    # Copia o conteúdo do checkpoint para a pasta do modelo final
    shutil.copytree(checkpoint_path, final_model_path, dirs_exist_ok=True)
    print(f"✅ Modelo salvo com sucesso em: {final_model_path}")
else:
    print(f"❌ Checkpoint não encontrado em: {checkpoint_path}")
```

```
✅ Modelo salvo com sucesso em: /content/drive/MyDrive/Fase3/models/llama-fine-tuned-v4
```

```
from google.colab import drive

# Caminhos das pastas onde os modelos devem estar
output_path_v1 = "/content/drive/MyDrive/Fase3/dataset/models/llama fine tuned"
output_path_v2 = "/content/drive/MyDrive/Fase3/dataset/models/Llama Fine Tune 2"
output_path_v4 = "/content/drive/MyDrive/Fase3/models/llama-fine-tuned-v4"

# Verificar se os modelos estão nos diretórios
print("🔍 Verificando o modelo v1:")
!ls "{output_path_v1}"

print("\n🔍 Verificando o modelo v2:")
!ls "{output_path_v2}"

print("\n🔍 Verificando o modelo v4:")
!ls "{output_path_v4}"
```

```
🔍 Verificando o modelo v1:
 adapter_config.json        'special_tokens_map (1).json'   training_args.bin
 adapter_model.safetensors  'tokenizer (1).json'
'README (1).md'              tokenizer_config.json

🔍 Verificando o modelo v2:
 adapter_model.safetensors     model.safetensors          tokenizer.json
'config (1).json'             'optimizer (1).pt'         'trainer_state (1).json'
 config.json                   optimizer.pt               trainer_state.json
'generation_config (1).json'   rng_state.pth              training_args.bin
 generation_config.json        scheduler.pt
'merges (1).txt'               special_tokens_map.json

🔍 Verificando o modelo v4:
adapter_config.json        optimizer.pt  rng_state.pth  trainer_state.json
adapter_model.safetensors  README.md     scheduler.pt   training_args.bin
```

📝 Código para Comparar o Llama 3.2-1B e o Modelo v4 Fine-Tuned

📝 Comparar as Respostas para Ambos os Modelos

```
!ls -l /content/drive/MyDrive/Fase3/models/llama-fine-tuned-v4
```

```
total 20060
-rw-------+ 1 root root      719 Dec 17 16:20 adapter_config.json
-rw-------+ 1 root root  6824216 Dec 17 16:20 adapter_model.safetensors
-rw-------+ 1 root root 13685562 Dec 17 16:20 optimizer.pt
-rw-------+ 1 root root     5097 Dec 17 16:20 README.md
-rw-------+ 1 root root    14244 Dec 17 16:20 rng_state.pth
-rw-------+ 1 root root     1064 Dec 17 16:20 scheduler.pt
-rw-------+ 1 root root     2627 Dec 17 16:20 trainer_state.json
-rw-------+ 1 root root     5304 Dec 17 16:20 training_args.bin
```

```
from transformers import AutoTokenizer

model_name = "meta-llama/Llama-3.2-1B"
tokenizer = AutoTokenizer.from_pretrained(model_name)

# Salvar o tokenizer na pasta do modelo v4
```

```
tokenizer.save_pretrained("/content/drive/MyDrive/Fase3/models/llama-fine-tuned-v4")
```

```
('/content/drive/MyDrive/Fase3/models/llama-fine-tuned-v4/tokenizer_config.json',
 '/content/drive/MyDrive/Fase3/models/llama-fine-tuned-v4/special_tokens_map.json',
 '/content/drive/MyDrive/Fase3/models/llama-fine-tuned-v4/tokenizer.json')
```

```
tokenizer = AutoTokenizer.from_pretrained("/content/drive/MyDrive/Fase3/models/llama-fine-tuned-v4")
```

```
#Precisamos tratar o arquivo tst.json
import json

# 📁 Caminho do arquivo de teste original e do arquivo corrigido
test_data_path = "/content/drive/MyDrive/Fase3/dataset/tst.json"
corrected_test_data_path = "/content/drive/MyDrive/Fase3/dataset/tst_corrected.json"

# 🕵️ Verificar as primeiras linhas do arquivo para determinar o formato
print("🔍 Inspecionando as primeiras 5 linhas do arquivo:")
with open(test_data_path, 'r', encoding='utf-8') as file:
    for i in range(5):
        print(file.readline().strip())

# 📥 Carregar o arquivo em formato JSON Lines (caso seja esse o formato)
test_data = []
try:
    with open(test_data_path, 'r', encoding='utf-8') as file:
        for line in file:
            try:
                test_data.append(json.loads(line.strip()))
            except json.JSONDecodeError as e:
                print(f"Erro ao decodificar a linha: {e}")
except Exception as e:
    print(f"Erro ao ler o arquivo: {e}")

print(f"✅ Total de registros carregados: {len(test_data)}")

# 💾 Salvar o arquivo corrigido (se necessário)
with open(corrected_test_data_path, 'w', encoding='utf-8') as outfile:
    json.dump(test_data, outfile, ensure_ascii=False, indent=2)

print(f"✅ Arquivo corrigido salvo em: {corrected_test_data_path}")
```

```
🔍 Inspecionando as primeiras 5 linhas do arquivo:
{"uid": "0000032069", "title": "Adult Ballet Tutu Cheetah Pink", "content": "", "target_ind": [0, 1, 2, 4, 7, 8], "target_rel": [1.0
{"uid": "0000589012", "title": "Why Don't They Just Quit? DVD Roundtable Discussion: What Families and Friends need to Know About Ac
{"uid": "0000031852", "title": "Girls Ballet Tutu Zebra Hot Pink", "content": "TUtu", "target_ind": [13, 16, 18, 20, 23, 32, 33, 11
{"uid": "0000032050", "title": "Adult Ballet Tutu Purple", "content": "", "target_ind": [1, 2, 4, 7, 8, 31, 32, 35, 41, 46, 53, 60,
{"uid": "0001203088", "title": "Hilda Boswell's Omnibus - A Treasury of Favorites", "content": "", "target_ind": [150], "target_rel
✅ Total de registros carregados: 970237
✅ Arquivo corrigido salvo em: /content/drive/MyDrive/Fase3/dataset/tst_corrected.json
```

```
import json

# Caminho do arquivo de teste
test_data_path = "/content/drive/MyDrive/Fase3/dataset/tst.json"

# Lista para armazenar os dados formatados
formatted_test_data = []

# Carregar e formatar os dados
with open(test_data_path, 'r', encoding='utf-8') as file:
    for line in file:
        record = json.loads(line.strip())
        title = record.get('title', '').strip()
        content = record.get('content', '').strip()

        # Incluir apenas registros onde 'title' e 'content' não estão vazios
        if title and content:
            formatted_test_data.append({
                "input_text": title,
                "response": content
            })

# Salvar o dataset formatado
formatted_test_data_path = "/content/drive/MyDrive/Fase3/dataset/formatted_tst.json"
with open(formatted_test_data_path, 'w', encoding='utf-8') as outfile:
    json.dump(formatted_test_data, outfile, ensure_ascii=False, indent=2)
```

```
print(f"✅ Total de registros formatados: {len(formatted_test_data)}")
print(f"📁 Dataset de teste formatado salvo em: {formatted_test_data_path}")
```

```
✅ Total de registros formatados: 599743
📁 Dataset de teste formatado salvo em: /content/drive/MyDrive/Fase3/dataset/formatted_tst.json
```

```
import os
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM

# 🔧 Definir CUDA_LAUNCH_BLOCKING para depuração
os.environ["CUDA_LAUNCH_BLOCKING"] = "1"

# 📁 Caminho do modelo v4
model_path = "/content/drive/MyDrive/Fase3/models/llama-fine-tuned-v4"

# Tentar carregar o modelo e o tokenizer
try:
    tokenizer = AutoTokenizer.from_pretrained(model_path)
    model = AutoModelForCausalLM.from_pretrained(model_path, device_map="auto")

    # Definir dispositivo como CPU para evitar erros relacionados à GPU
    device = torch.device("cpu")
    model.to(device)

    # 📝 Função para gerar respostas
    def generate_response(prompt, max_length=600):
        inputs = tokenizer(prompt, return_tensors="pt").to(device)
        with torch.no_grad():
            outputs = model.generate(**inputs, max_length=max_length)
        return tokenizer.decode(outputs[0], skip_special_tokens=True)

    # 🔹 Exemplo de prompt para teste
    prompt = "Girls Ballet Tutu Zebra Hot Pink"
    print(f"🔹 **Prompt:** {prompt}")
    response = generate_response(prompt)
    print(f"🧠 **Resposta do Modelo v4:** {response}")

except Exception as e:
    print(f"Erro ao carregar o modelo ou gerar respostas: {e}")
```

```
Setting `pad_token_id` to `eos_token_id`:128001 for open-end generation.
🔹 **Prompt:** Girls Ballet Tutu Zebra Hot Pink
🧠 **Resposta do Modelo v4:** Girls Ballet Tutu Zebra Hot Pink
Girls Ballet Tutu Zebra Hot Pink
Girls Ballet Tutu Zebra Hot Pink
Girls Ballet Tutu Zebra Hot Pink
Girls Ballet Tutu Zebra Hot Pink
Girls Ballet Tutu Zebra Hot Pink
Girls Ballet Tutu Zebra Hot Pink
Girls Ballet Tutu Zebra Hot Pink
```

```
def generate_response(prompt, max_length=200):
    inputs = tokenizer(prompt, return_tensors="pt").to(device)
    with torch.no_grad():
        outputs = model.generate(
            **inputs,
            max_length=max_length,
            temperature=0.7,     # Aumenta a diversidade
            top_p=0.9,           # Nucleus Sampling
            num_beams=5          # Beam Search
        )
    return tokenizer.decode(outputs[0], skip_special_tokens=True)

# 🔹 Testar com um prompt ajustado
prompt = "Describe the product: Girls Ballet Tutu Zebra Hot Pink"
print(f"🔹 **Prompt:** {prompt}")
response = generate_response(prompt)
print(f"🧠 **Resposta do Modelo v4:** {response}")
```

```
Setting `pad_token_id` to `eos_token_id`:128001 for open-end generation.
🔹 **Prompt:** Describe the product: Girls Ballet Tutu Zebra Hot Pink
🧠 **Resposta do Modelo v4:** Describe the product: Girls Ballet Tutu Zebra Hot Pink
Girls Ballet Tutu Zebra Hot Pink
Girls Ballet Tutu Zebra Hot Pink Girls Ballet Tutu Zebra Hot Pink Girls Ballet Tutu Zebra Hot Pink Girls Ballet Tutu Zebra Hot Pink
```

```
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM

# 🚀 Carregar o modelo v4 e o tokenizer
model_path_v4 = "/content/drive/MyDrive/Fase3/models/llama-fine-tuned-v4"
tokenizer = AutoTokenizer.from_pretrained(model_path_v4)
model = AutoModelForCausalLM.from_pretrained(model_path_v4, torch_dtype=torch.float16, device_map="auto")

# 📝 Função para gerar respostas ajustada
def generate_response(prompt, max_length=600):
    inputs = tokenizer(prompt, return_tensors="pt").to(model.device)
    with torch.no_grad():
        outputs = model.generate(
            **inputs,
            max_length=max_length,
            temperature=0.9,       # Aumenta a aleatoriedade
            top_p=0.85,            # Nucleus Sampling para diversidade
            repetition_penalty=1.2, # Penaliza repetições
            num_return_sequences=1
        )
    return tokenizer.decode(outputs[0], skip_special_tokens=True)

# 🔹 Testar com um prompt ajustado
prompt = "Describe the product in detail: Girls Ballet Tutu Zebra Hot Pink"
print(f"🔹 **Prompt:** {prompt}")
response = generate_response(prompt)
print(f"🧠 **Resposta do Modelo v4:** {response}")
```

```
Setting `pad_token_id` to `eos_token_id`:128001 for open-end generation.
🔹 **Prompt:** Describe the product in detail: Girls Ballet Tutu Zebra Hot Pink
🧠 **Resposta do Modelo v4:** Describe the product in detail: Girls Ballet Tutu Zebra Hot Pink & Black
Description ## Description 100% Cotton. Sizes S - XL
```

```
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM

# 🚀 Carregar o modelo v4 e o tokenizer
model_path_v4 = "/content/drive/MyDrive/Fase3/models/llama-fine-tuned-v4"
tokenizer = AutoTokenizer.from_pretrained(model_path_v4)
model = AutoModelForCausalLM.from_pretrained(model_path_v4, torch_dtype=torch.float16, device_map="auto")

# Definir pad_token como eos_token para evitar a mensagem
tokenizer.pad_token = tokenizer.eos_token
model.config.pad_token_id = tokenizer.pad_token_id  # Definir pad_token_id no modelo

# 📝 Função para gerar respostas ajustada
def generate_response(prompt, max_new_tokens=150):
    inputs = tokenizer(prompt, return_tensors="pt").to(model.device)
    with torch.no_grad():
        outputs = model.generate(
            **inputs,
            max_new_tokens=max_new_tokens,
            pad_token_id=tokenizer.pad_token_id,  # Passar pad_token_id explicitamente
            temperature=0.7,
            top_p=0.9,
            repetition_penalty=1.3,
            num_return_sequences=1
        )
    return tokenizer.decode(outputs[0], skip_special_tokens=True)

# 🔹 Testar com um prompt ajustado
prompt = "Describe the product in detail, including material, usage, and target audience: Girls Ballet Tutu Zebra Hot Pink."
print(f"🔹 **Prompt:** {prompt}")
response = generate_response(prompt)
print(f"🧠 **Resposta do Modelo v4:** {response}")
```

```
mv /content/wandb /content/drive/MyDrive/Fase3/wandb
```

```
drive.flush_and_unmount()
```

Devido aos sucessivos erros vamos reiniciar o ambiente

```
#🛠️ Instalação de Bibliotecas
```

```
# Atualizar o pip
!pip install --upgrade pip

# Bibliotecas essenciais
!pip install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cu118
!pip install transformers datasets peft
!pip install accelerate
!pip install wandb
!pip install bitsandbytes
!pip install sentencepiece
!pip install protobuf
!pip install rouge_score
!pip install evaluate
!pip install matplotlib pandas

# Se precisar de descompactadores
!apt-get install -y gzip
```

```
Requirement already satisfied: pip in /usr/local/lib/python3.10/dist-packages (24.3.1)
Looking in indexes: https://download.pytorch.org/whl/cu118
Requirement already satisfied: torch in /usr/local/lib/python3.10/dist-packages (2.5.1+cu121)
Requirement already satisfied: torchvision in /usr/local/lib/python3.10/dist-packages (0.20.1+cu121)
Requirement already satisfied: torchaudio in /usr/local/lib/python3.10/dist-packages (2.5.1+cu121)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from torch) (3.16.1)
Requirement already satisfied: typing-extensions>=4.8.0 in /usr/local/lib/python3.10/dist-packages (from torch) (4.12.2)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from torch) (3.4.2)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from torch) (3.1.4)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from torch) (2024.9.0)
Requirement already satisfied: sympy==1.13.1 in /usr/local/lib/python3.10/dist-packages (from torch) (1.13.1)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from sympy==1.13.1->torch) (1.3.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from torchvision) (1.26.4)
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in /usr/local/lib/python3.10/dist-packages (from torchvision) (11.0.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->torch) (3.0.2)
Requirement already satisfied: transformers in /usr/local/lib/python3.10/dist-packages (4.47.1)
Requirement already satisfied: datasets in /usr/local/lib/python3.10/dist-packages (3.2.0)
Requirement already satisfied: peft in /usr/local/lib/python3.10/dist-packages (0.14.0)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from transformers) (3.16.1)
Requirement already satisfied: huggingface-hub<1.0,>=0.24.0 in /usr/local/lib/python3.10/dist-packages (from transformers) (0.26.
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-packages (from transformers) (1.26.4)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from transformers) (24.2)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-packages (from transformers) (6.0.2)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.10/dist-packages (from transformers) (2024.9.11)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from transformers) (2.32.3)
Requirement already satisfied: tokenizers<0.22,>=0.21 in /usr/local/lib/python3.10/dist-packages (from transformers) (0.21.0)
Requirement already satisfied: safetensors>=0.4.1 in /usr/local/lib/python3.10/dist-packages (from transformers) (0.4.5)
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.10/dist-packages (from transformers) (4.66.6)
Requirement already satisfied: pyarrow>=15.0.0 in /usr/local/lib/python3.10/dist-packages (from datasets) (17.0.0)
Requirement already satisfied: dill<0.3.9,>=0.3.0 in /usr/local/lib/python3.10/dist-packages (from datasets) (0.3.8)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (from datasets) (2.2.2)
Requirement already satisfied: xxhash in /usr/local/lib/python3.10/dist-packages (from datasets) (3.5.0)
Requirement already satisfied: multiprocess<0.70.17 in /usr/local/lib/python3.10/dist-packages (from datasets) (0.70.16)
Requirement already satisfied: fsspec<=2024.9.0,>=2023.1.0 in /usr/local/lib/python3.10/dist-packages (from fsspec[http]<=2024.9.
Requirement already satisfied: aiohttp in /usr/local/lib/python3.10/dist-packages (from datasets) (3.11.10)
Requirement already satisfied: psutil in /usr/local/lib/python3.10/dist-packages (from peft) (5.9.5)
Requirement already satisfied: torch>=1.13.0 in /usr/local/lib/python3.10/dist-packages (from peft) (2.5.1+cu121)
Requirement already satisfied: accelerate>=0.21.0 in /usr/local/lib/python3.10/dist-packages (from peft) (1.2.1)
Requirement already satisfied: aiohappyeyeballs>=2.3.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (2.4.4
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (1.3.1)
Requirement already satisfied: async-timeout<6.0,>=4.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (4.0.3
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (24.2.0)
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (1.5.0)
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (6.1.0)
Requirement already satisfied: propcache>=0.2.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (0.2.1)
Requirement already satisfied: yarl<2.0,>=1.17.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (1.18.3)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->transformers)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (2.2.3
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (2024.
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from torch>=1.13.0->peft) (3.4.2)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from torch>=1.13.0->peft) (3.1.4)
Requirement already satisfied: sympy==1.13.1 in /usr/local/lib/python3.10/dist-packages (from torch>=1.13.0->peft) (1.13.1)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from sympy==1.13.1->torch>=1.13.0->
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas->datasets) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas->datasets) (2024.2)
```

```
from huggingface_hub import login

# Substitua 'seu_token_aqui' pelo token gerado
login()
```

n, make sure you are properly logged in by executing `huggingface-cli login`, and if you

```
import os
os.environ["HF_TOKEN"] = "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"
```

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

```
WARNING: The following packages were previously imported in this runtime:
  [accelerate,peft,transformers]
You must restart the runtime in order to use newly installed versions.
```

RESTART SESSION

```
# 🔹 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)}")
```

📊 Preparando para monitorar em tempo real

🚀 Usando TensorBoard e Weights & Biases (W&B)

```
#3️⃣ 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,
    }
)

# simulate training
epochs = 5
offset = random.random() / 5
for epoch in range(2, epochs):
    acc = 1 - 2 ** -epoch - random.random() / epoch - offset
    loss = 2 ** -epoch + random.random() / epoch + offset

    # log metrics to wandb
    wandb.log({"acc": acc, "loss": loss})
```

```
# [optional] finish the wandb run, necessary in notebooks
#wandb.finish()
```

```
#Instalar o Weights & Biases
!pip install wandb
```

```
#Configurar o W&B:
#obter o token em: https://wandb.ai/authorize

import wandb
# Substitua pelo seu token real, entre aspas
wandb.login(key="277119a7a9f72b15b3b8b00081bd412a7a7eee6c")
```