Tech Challenge – Fase 3

Integrantes:

• Breno Robert de Oliveira Ribeiro

• Daniel Correia dos Reis

• Felipe Moura Monteiro

• João Alberto Dutra da Silveira Duarte

• Romulo Figueiredo Romano

Link do vídeo: https://www.youtube.com/watch?v=SD1KwzfZ15U

Link do Github: https://github.com/DanielCorreiaReis/tech-challenge-fase3

Definição do Problema

Usuários da Amazon enfrentam dificuldades em encontrar rapidamente informações sobre títulos de produtos. O desafio é realizar o fine-tuning, de modo que ele seja capaz de responder a perguntas dos usuários com base nas descrições dos produtos, extraindo as informações relevantes diretamente do contexto fornecido.

O objetivo

O projeto tem por finalidade realizar o fine-tuning de um modelo de linguagem pré-treinado para a geração de descrições de produtos (livros) a partir de seus títulos. O modelo deve ser capaz de gerar descrições coerentes e informativos para melhor a apresentação dos livros.

Os critérios de sucesso

Configurar e treinar o modelo da melhor maneira para que ele possa trazer a melhor descrição para o usuário.

Processo de Seleção e Preparação do Dataset

Aquisição e Formatação dos Dados

Os dados foram extraídos de um conjunto json (trn.json), contendo pares de título e descrição dos livros. O dataset passou por uma limpeza e formatação, garantindo que apenas entradas válidas fossem utilizadas.

Filtro aplicado: O código verifica se os campos ‘title’ e ‘content’ estão presentes e não vazios antes de processar cada item.

Arquivo de saída: Os dados filtrados e reestruturados são salvos em amazon_titles_search.json, posteriormente convertidos para formatted_amazon_titles_search_data.json para o fine-tuning.

```
[ ]  def process_dataset(item):
       return {
           "input": f"PLEASE PROVIDE ME A DESCRIPTION FOR THIS PRODUCT TITLE.\n[|
           Title|] {item['title']} [|eTitle|]\n\n\[|Description|] {item['content']}
           [|eDescription|]"
       }
```

```
import json
processed_data = []

with open(r'/content/drive/MyDrive/fiap/trn.json', 'r', encoding='utf-8') as
file:
  for line in file:
    item = json.loads(line)
    # Verifica se os campos title e content existem e não estão vazios
    if item.get('title') and item.get('content'):
          processed_data.append(process_dataset(item))

output_file = r'/content/drive/MyDrive/fiap/amazon_titles_search.json'
with open(output_file, 'w', encoding='utf-8') as file:
  json.dump(processed_data, file, ensure_ascii=False, indent=4)

print(f"Todos os dados reformatados foram salvos em '{output_file}'.")
```

Processo de Fine-Tuning do Modelo

- Carregamento do Modelo

  O modelo utilizado foi unsloth/llama-3-8b-bnb-4bit, carregado com as seguintes configurações:

  - max_seq_length=4096
  - load_in_4bit=True (redução de consumo de memória)
  - tokenizer.pad_token = tokenizer.eos_token (evita erros de padding)

```
[ ]  # Importando dependencias para preparar modelo pra teste
     import unsloth
     from unsloth import FastLanguageModel, is_bfloat16_supported
     import torch
     import json
     from datasets import load_dataset
     from trl import SFTTrainer
     from transformers import TrainingArguments

     DATA_PATH = "/content/drive/MyDrive/fiap/amazon_titles_search.json"
     OUTPUT_PATH_DATASET = "/content/drive/MyDrive/fiap/
     formatted_amazon_titles_search_data.json"
     max_seq_length = 2048
     dtype = None
     load_in_4bit = True
     fourbit_models = [
         "unsloth/mistral-7b-v0.3-bnb-4bit",
         "unsloth/mistral-7b-instruct-v0.3-bnb-4bit",
         "unsloth/llama-3-8b-bnb-4bit",
         "unsloth/llama-3-8b-Instruct-bnb-4bit",
         "unsloth/llama-3-70b-bnb-4bit",
         "unsloth/Phi-3-mini-4k-instruct",
         "unsloth/Phi-3-medium-4k-instruct",
         "unsloth/mistral-7b-bnb-4bit",
         "unsloth/gemma-7b-bnb-4bit",
     ]
```

- Configuração do Treinamento

  Para o fine-tuning, foi utilizada a técnica LoRA (Low-Rank Adaptation) com os seguintes parâmetros:

  - r=16 (redução das matrizes de ajuste)
  - lora_alpha=16 (intensidade do ajuste)
  - bias="none" (não treina viés das camadas)
  - use_gradient_checkpointing="unsloth" (economia de memória durante o treino)
  - random_state=3407 (reprodutibilidade)

```python
model = FastLanguageModel.get_peft_model(
    model,
    r = 16,
    target_modules = ["q_proj", "k_proj", "v_proj", "o_proj",
                      "gate_proj", "up_proj", "down_proj",],
    lora_alpha = 16,
    lora_dropout = 0,
    bias = "none",

    use_gradient_checkpointing = "unsloth",
    random_state = 3407,
    use_rslora = False,
    loftq_config = None,
)
```

- Pré-processamento para o Treinamento

Os dados foram formatados com um prompt estruturado, baseado no padrão Alpaca.

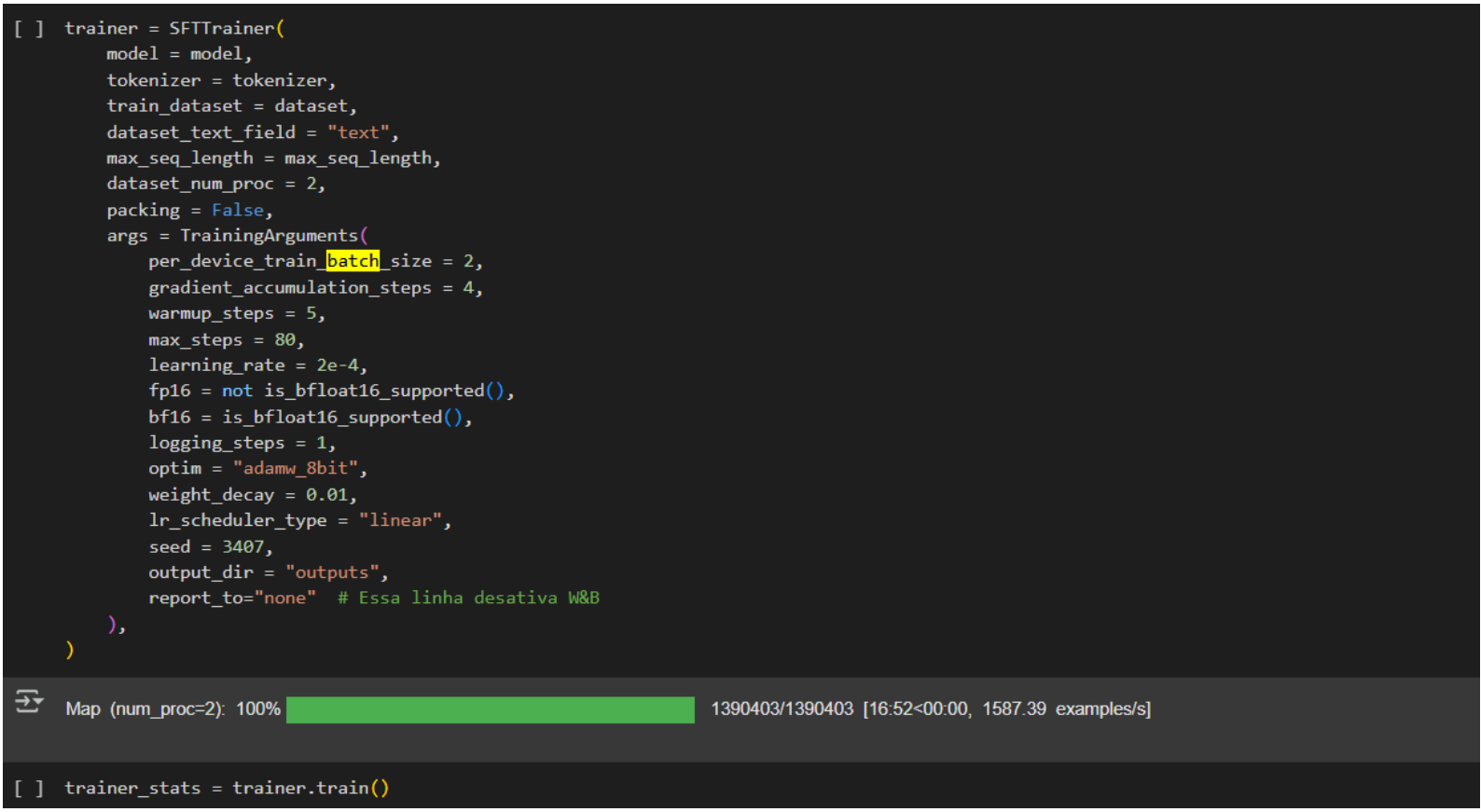
Após essa etapa, o dataset final foi carregado com a biblioteca datasets e convertido para um formato adequado ao fine-tuning.

```
alpaca_prompt = """Below is an instruction that describes a task, paired with
an input that provides further context. Write a response that appropriately
completes the request.

### Instruction:
{}

### Input:
{}

### Response:
{}"""

EOS_TOKEN = tokenizer.eos_token
def formatting_prompts_func(examples):
    instructions = examples["instruction"]
    inputs       = examples["input"]
    outputs      = examples["output"]
    texts = []
    for instruction, input, output in zip(instructions, inputs, outputs):

        text = alpaca_prompt.format(instruction, input, output) + EOS_TOKEN
        texts.append(text)
    return { "text" : texts, }
pass

from datasets import load_dataset, Features, Value

OUTPUT_PATH_DATASET = "/content/drive/MyDrive/fiap/
formatted_amazon_titles_search_data.json"

# Define the features of your dataset explicitly
features = Features({
    'instruction': Value(dtype='string'),
    'input': Value(dtype='string'),
    'output': Value(dtype='string'),
})

dataset = load_dataset('json', data_files=OUTPUT_PATH_DATASET, split='train',
features=features)
dataset = dataset.map(formatting_prompts_func, batched = True)
```

Treinamento do Modelo

- O treinamento foi realizado utilizando SFTTrainer, com os seguintes parâmetros principais:
    - Batch Size (per_device_train_batch_size=2): Mantém um consumo equilibrado de memória.
    - Passos de Treinamento (max_steps=80): Define um fine-tuning rápido e eficiente.
    - Taxa de Aprendizado (learning_rate=2e-4): Ajustado para um aprendizado estável.
    - Otimização (optim="adamw_8bit"): Reduz o consumo de memória mantendo o desempenho.
- Por que report_to="none"?
    - Esse parâmetro desativa o uso do Weights & Biases (W&B) ou outros sistemas de tracking de experimentos, evitando logs desnecessários e possíveis erros caso a integração com essas ferramentas não esteja configurada.

A execução do treinamento gera um modelo ajustado salvo no diretório outputs, pronto para inferência.



Teste do Modelo Treinado

Após o treinamento, o modelo foi carregado e testado para avaliar sua capacidade de gerar descrições coerentes para títulos de produtos.

- Carregamento do Modelo Ajustado

  O modelo foi carregado a partir do diretório onde foi salvo (/content/drive/MyDrive/fiap/lora_model).

  Esse carregamento permite que o modelo seja usado em inferência, sem a necessidade de manter parâmetros de treinamento ativos.

- Geração de Texto

O modelo foi testado gerando uma descrição baseada em um título de produto, utilizando o formato de prompt definido no treinamento (alpaca_prompt)

  - O prompt segue o formato definido, garantindo que o modelo receba instruções estruturadas.
  - A função generate() gera um texto de no máximo 128 tokens como resposta.
  - A inferência é realizada na GPU (cuda), otimizando a velocidade de geração.

```
alpaca_prompt = """Below is an instruction that describes a task, paired with
an input that provides further context. Write a response that appropriately
completes the request.

### Instruction:
{}

### Input:
{}

### Response:
{}"""


inputs = tokenizer(
[
alpaca_prompt.format(
        "PLEASE PROVIDE ME A DESCRIPTION FOR THIS PRODUCT TITLE",
        news_test[100],
        "",
    )
], return_tensors = "pt").to("cuda")

from transformers import TextStreamer
text_streamer = TextStreamer(tokenizer)
_ = model.generate(**inputs, streamer = text_streamer, max_new_tokens = 128)
```

```
<|begin_of_text|>Below is an instruction that describes a task, paired with an input that provides further context. Write a response that appropriately completes the request.

### Instruction:
PLEASE PROVIDE ME A DESCRIPTION FOR THIS PRODUCT TITLE

### Input:
The Loving Cup (The Poldark novels)

### Response:
The Loving Cup, the second in the Poldark series, is a vivid, engaging, and memorable portrait of a man and a time and a place.<|end_of_text|>
```

Conclusão

O modelo foi treinado com mais de 1 milhão de dados, permitindo gerar descrições coerentes e detalhadas a partir de títulos de livros. O fine-tuning com LoRA e otimizações de memória garantiu um aprendizado eficiente. Nos testes, o modelo demonstrou compreender os títulos e produzir textos naturais, validando o sucesso do treinamento.