Pós Tech FIAP - IA para Devs - TCC3

Alunos
Luciano Giles Soares - RM 359834
Luiz Ricardo Zinsly Calmon - RM 359894
Rafael Alves Cardoso - RM 360124
Silvio Cezer Saczuck - RM 360204
William Judice Yizima - RM 360214

GitHub do TCC3: https://github.com/WilliamYizima/POS-4IADT-3
Vídeo da apresentação: https://youtu.be/QrHZkkCkyHE

Objetivo do Projeto

Este projeto tem como objetivo aplicar o conhecimento obtido durante a fase do curso para executar o **fine-tuning de um foundation model (Llama 3.1-8B)** utilizando o dataset **AmazonTitles-1.3MM**. O modelo treinado deve ser capaz de **responder perguntas dos usuários com base nos títulos de produtos**, utilizando descrições aprendidas durante o treinamento.

Estrutura do Notebook e Análise das Células:

Célula 1: Importações Essenciais
from IPython import get_ipython: Importa funcionalidades para interagir com o ambiente IPython (o kernel do Colab). from IPython.display import display: Importa funcionalidades para exibir objetos no notebook.

Célula 2: Montagem do Google Drive
from google.colab import drive: Importa a biblioteca necessária para montar o Google Drive no ambiente Colab. drive.mount('/content/drive'): Monta o Google Drive no diretório /content/drive, permitindo que o notebook acesse arquivos armazenados no seu Drive.

Célula 3: Instalação do Unsloth e Dependências
%%capture: Suprime a saída desta célula. import os: Importa a biblioteca os para interagir com o sistema operacional. O código verifica se está em um ambiente Colab ("COLAB_" not in "".join(os.environ.keys())). Se não estiver no Colab (o que não é o caso neste contexto), instala o Unsloth via pip. Se estiver no Colab: Instala as dependências essenciais (bitsandbytes, accelerate, xformers, peft, trl, triton, cut_cross_entropy, unsloth_zoo) com pip install --no-deps. O no-deps evita conflitos de versão. Instala outras dependências (sentencepiece, protobuf, datasets, huggingface_hub, hf_transfer) necessárias para manipulação de dados e interação com o Hugging Face Hub. Instala o Unsloth via pip (neste caso, sem dependências diretas, assumindo que as anteriores cobrem o necessário). Nota: A linha comentada !pip install "unsloth[colab-new] @ git+https://github.com/unslothai/unsloth.git" é uma forma alternativa de instalar a versão mais recente do Unsloth diretamente do GitHub, focada no Colab.

Célula 4: Instalação de Xformers (Versão Específica)
Importa version do torch e Version do packaging.version para verificar a versão do PyTorch. Define a versão do xformers a ser instalada com base na versão do PyTorch (xformers==0.0.27 para PyTorch < 2.4.0, xformers sem versão específica para versões mais recentes). Instala o xformers (com a versão definida), trl, peft, accelerate, bitsandbytes e triton com pip install --no-deps. Esta célula parece redundante após a Célula 3 em um ambiente Colab, mas garante que a versão correta do xformers seja usada.

Célula 5: Importações e Configurações Adicionais
Importa classes e funções das bibliotecas instaladas (SFTTrainer, TrainingArguments, is_bfloat16_supported, load_dataset, Dataset, pandas, json, yaml, unicodedata, re, os). os.environ["WANDB_DISABLED"] = "true": Desabilita a integração com o Weights & Biases (WandB), uma ferramenta comum para monitoramento de treinamento, para evitar a solicitação de uma API Key.

Célula 6: Carregamento do Modelo Base com Unsloth
from unsloth import FastLanguageModel: Importa a classe principal do Unsloth para carregar modelos rapidamente. import torch: Importa a biblioteca PyTorch. Define max_seq_length (comprimento máximo da sequência de entrada), dtype (tipo de dado, None para detecção automática) e load_in_4bit (habilita quantização em 4 bits para reduzir o uso de memória VRAM). Define uma lista fourbit_models com nomes de modelos otimizados para 4 bits pelo Unsloth. model, tokenizer = FastLanguageModel.from_pretrained(...): Carrega o modelo pré-treinado unsloth/Meta-Llama-3.1-8B utilizando a função otimizada do Unsloth. Configura o comprimento máximo da sequência, tipo de dado e ativa a quantização em 4 bits. Importa CHAT_TEMPLATES e get_chat_template para configurar o tokenizador com um template de chat específico. tokenizer = get_chat_template(tokenizer, chat_template = "gemma-3", ...): Configura o tokenizador para usar o template de chat "gemma-3". Nota: O template "gemma-3" pode não ser o ideal para o modelo Llama 3.1. Seria mais apropriado usar um template compatível com Llama 3.1.

Célula 7: Função prepara_dataset
Define a função prepara_dataset(inpfile, outfile, tam_regs=None) para pré-processar os datasets de treinamento e teste. Carrega dados de um arquivo JSON de entrada (inpfile) linha por linha, tratando possíveis erros de decodificação JSON. Filtra os dados para incluir apenas itens com 'title' e 'content'. Converte os dados em um DataFrame pandas. Define a função normalizar para remover caracteres de controle e excesso de espaços, normalizando o texto. Aplica a função normalizar às colunas 'title' e 'content'. Remove linhas onde 'title' ou 'content' estão vazios após a normalização. Remove linhas duplicadas com base em 'title' e 'content'. Reseta o índice do DataFrame. Cria as colunas 'instruction', 'input' e 'output' no formato necessário para o fine-tuning, seguindo a estrutura de "instruction-input-output". Se tam_regs for especificado, amostra um subconjunto do DataFrame com tam_regs registros (com um random_state para reprodutibilidade). Formata os dados em uma lista de dicionários com as colunas 'instruction', 'input' e 'output'. Salva os dados formatados em um arquivo JSON de saída (outfile). Imprime informações sobre o número de registros processados em cada etapa. Retorna o número final de registros processados.

Célula 8: Definição dos Caminhos dos Arquivos

Define as variáveis com os caminhos dos arquivos JSON originais de treinamento (dataset_treino) e teste (dataset_teste) no Google Drive. Define as variáveis com os caminhos de saída para os datasets processados de treinamento (dataset_treino_output) e teste (dataset_teste_output).

Célula 9: Execução do Pré-processamento para Treinamento

Chama a função prepara_dataset para processar o arquivo de treinamento (dataset_treino) e salvar o resultado em dataset_treino_output. Limita o número de registros para 250000 .

Célula 10: Execução do Pré-processamento para Teste

Chama a função prepara_dataset para processar o arquivo de teste (dataset_teste) e salvar o resultado em dataset_teste_output. Limita o número de registros para 2500.

Célula 11: Configuração do Modelo para Fine-tuning (LoRA)

model = FastLanguageModel.get_peft_model(...): Configura o modelo carregado anteriormente para fine-tuning utilizando o método LoRA (Low-Rank Adaptation). Define o rank (r) das matrizes LoRA (16). Define os módulos alvo (target_modules) do modelo onde as adaptações LoRA serão aplicadas (módulos relacionados à atenção e feed-forward). Define lora_alpha (escala das atualizações LoRA) e lora_dropout. Define bias como "none". Habilita use_gradient_checkpointing = "unsloth" para otimizar o uso de memória VRAM. Define random_state para reprodutibilidade. Outras configurações (use_rslora, loftq_config) são mantidas como padrão.

Célula 12: Definição do Template Alpaca Prompt

Define a string alpaca_prompt que representa a estrutura do template de prompt a ser usado para formatar os dados para o treinamento. Este template segue o formato "instruction-input-response".

Célula 13: Função formatting_prompts_func

Define a função formatting_prompts_func que será usada para formatar o dataset no template do prompt. Obtém as colunas 'instruction', 'input' e 'output' do dataset. Itera sobre os exemplos e formata cada um utilizando o alpaca_prompt e adicionando o EOS_TOKEN (End-Of-Sentence Token) do tokenizador no final. Retorna um dicionário com a coluna 'text' contendo os prompts formatados.

Célula 14: Carregamento e Formatação de um Dataset de Exemplo (Yahma/Alpaca-cleaned)

%%capture: Suprime a saída. from datasets import load_dataset: Importa a função para carregar datasets do Hugging Face Hub. dataset = load_dataset("yahma/alpaca-cleaned", split = "train"): Carrega um dataset de exemplo (yahma/alpaca-cleaned). Nota: Este dataset parece ser carregado apenas para demonstração ou teste inicial, pois o dataset customizado de produtos Amazon será carregado posteriormente. dataset = dataset.map(formatting_prompts_func, batched = True,): Aplica a função formatting_prompts_func a este dataset de exemplo para formatá-lo.

Célula 15: Habilitação da Inferência Rápida com Unsloth
%%capture: Suprime a saída. FastLanguageModel.for_inference(model): Otimiza o modelo para inferência, tornando-a até 2x mais rápida.

Célula 16: Exemplo de Inferência (Antes do Fine-tuning Completo)
Prepara uma entrada (inputs) para o tokenizador utilizando o template alpaca_prompt com uma instrução e um input de exemplo ("Who was Ayrton Senna?"). O campo de resposta é deixado vazio para o modelo preencher. from transformers import TextStreamer: Importa a classe TextStreamer para exibir a saída gerada pelo modelo em tempo real. text_streamer = TextStreamer(tokenizer): Cria uma instância do TextStreamer. _ = model.generate(**inputs, streamer = text_streamer, max_new_tokens = 128): Gera texto utilizando o modelo carregado (que já tem a configuração LoRA, mas ainda não foi treinado com os dados customizados). A saída é transmitida pelo text_streamer. print (text_streamer): Imprime o objeto TextStreamer (provavelmente para verificar seu conteúdo após a geração).

Célula 17: Carregamento e Formatação do Dataset de Treinamento Customizado
from datasets import load_dataset: Importa a função novamente (já importada na Célula 5). dataset = load_dataset("json", data_files="/content/drive/MyDrive/TCC3-Fiap/LF-Amazon-1.3M/formatted_train_dataset.json", split="train"): Carrega o dataset de treinamento pré-processado do arquivo JSON salvo na Célula 9. dataset = dataset.map(formatting_prompts_func, batched = True,): Aplica a função formatting_prompts_func a este dataset customizado para formatá-lo para o treinamento.

Célula 18: Configuração e Preparação do Treinador (SFTTrainer)
Define o diretório de saída (output_dir) para salvar os resultados do treinamento. Cria uma instância do SFTTrainer, que é uma classe de conveniência para fine-tuning em tarefas de Sequence-to-Sequence. Configura o trainer com o modelo (model), tokenizador (tokenizer), dataset de treinamento formatado (dataset), campo de texto no dataset (dataset_text_field), comprimento máximo da sequência (max_seq_length), número de processos para carregar dados (dataset_num_proc), e desabilita o packing (packing). Configura os argumentos de treinamento (TrainingArguments): per_device_train_batch_size: Tamanho do batch por dispositivo (GPU). gradient_accumulation_steps: Número de passos para acumular gradientes antes de atualizar os pesos. warmup_steps: Número de passos de aquecimento da taxa de aprendizado. num_train_epochs: Número de épocas de treinamento (comentado, usando max_steps). max_steps: Número máximo de passos de treinamento. learning_rate: Taxa de aprendizado inicial. fp16 e bf16: Configura o tipo de precisão (Float16 ou Bfloat16) com base na compatibilidade do hardware. logging_steps: Frequência para registrar métricas. optim: Otimizador a ser usado. weight_decay: Penalidade L2. lr_scheduler_type: Tipo de agendador de taxa de aprendizado. seed: Semente para reprodutibilidade. output_dir: Diretório para salvar os resultados. report_to = "none": Desabilita relatórios para plataformas externas. save_strategy="no": Não salva checkpoints intermediários.

Célula 19: Exibição das Estatísticas de Memória da GPU (Antes do Treinamento)
Obtém as propriedades da GPU. Calcula e imprime a memória da GPU reservada e a memória total.

Célula 20: Execução do Treinamento
trainer_stats = trainer.train(): Inicia o processo de treinamento utilizando as configurações definidas na Célula 18.

Célula 21: Exibição das Estatísticas de Memória e Tempo (Após o Treinamento)
Calcula e imprime o tempo total de treinamento, a memória da GPU utilizada (pico reservado), a memória utilizada especificamente para o LoRA, e as porcentagens de uso em relação à memória total da GPU.

Célula 22: Salvamento do Modelo e Tokenizador Fine-tuned
model.save_pretrained(...): Salva os pesos LoRA treinados no diretório especificado.
tokenizer.save_pretrained(...): Salva o tokenizador (incluindo configurações de chat) no mesmo diretório.

Célula 23: Carregamento Opcional do Modelo Salvo (Comentado)
Este bloco de código está comentado, mas demonstra como carregar o modelo e tokenizador salvos para inferência após o treinamento.
FastLanguageModel.from_pretrained(...): Carrega o modelo a partir do diretório salvo.
FastLanguageModel.for_inference(model): Otimiza o modelo carregado para inferência.

Célula 24: Exemplo de Inferência (Após o Fine-tuning)
Prepara uma nova entrada para o tokenizador com uma instrução e um input diferente ("Rightly Dividing the Word") utilizando o alpaca_prompt. O campo de resposta é deixado vazio. Cria uma nova instância de TextStreamer. Gera texto utilizando o modelo que foi treinado com os dados customizados. A saída é transmitida pelo text_streamer. Imprime o objeto TextStreamer. Fluxo Geral do Notebook:
Configura o ambiente Colab (monta Drive, instala bibliotecas). Carrega um modelo de linguagem base pré-treinado usando Unsloth (com quantização 4-bit). Define uma função para limpar e formatar dados de produtos Amazon. Processa os datasets de treinamento e teste customizados. Configura o modelo para fine-tuning eficiente usando LoRA. Define o template de prompt para formatar os dados para o treinamento. Carrega o dataset de treinamento customizado e o formata no template do prompt. Configura o treinador (SFTTrainer) com hiperparâmetros para o fine-tuning. Executa o treinamento do modelo nos dados customizados. Salva o modelo e tokenizador fine-tuned. Demonstra como carregar o modelo salvo para inferência. Realiza uma inferência com o modelo treinado para testar sua capacidade de gerar respostas com base em um título. Observações Importantes:
O uso do Unsloth otimiza significativamente o processo de fine-tuning e o uso de memória em GPUs limitadas como as do Google Colab. O pré-processamento dos dados customizados é crucial para garantir que o modelo aprenda a tarefa de forma eficaz. A configuração dos parâmetros LoRA e dos argumentos de treinamento é fundamental para o desempenho e a estabilidade do fine-tuning. O template de prompt utilizado (alpaca_prompt) define como as instruções, inputs e outputs são apresentados ao modelo durante o treinamento e a inferência. É importante que o formato da inferência corresponda ao formato de treinamento. O template de chat "gemma-3" usado na Célula 6 pode ser inadequado para o modelo Llama 3.1. Verificar a documentação do Unsloth ou do modelo Llama 3.1 para o template de chat correto é recomendado.

Pós Tech FIAP - IA para Devs - TCC3

Dataset Utilizado

- **Fonte:** AmazonTitles-1.3MM
- **Arquivo principal:** trn.json
- **Colunas utilizadas:**
    - title: Título do produto
    - content: Descrição do produto
- **Pré-processamento aplicado:**
    - Remoção de registros vazios ou duplicados
    - Normalização de texto (remoção de caracteres de controle, espaços duplicados)
    - Transformação no formato instruction-input-output para fine-tuning

---

Modelo e Ferramentas Utilizadas

- **Foundation Model:** Llama 3.1-8B - Unsloth (bnb-4bit)
- **Ambiente:** Google Colab
- **Bibliotecas:**
    - unsloth, transformers, datasets, trl, bitsandbytes, xformers, sentencepiece, protobuf, pandas
- **Quantização:** 4-bit
- **Método de Fine-tuning:** LoRA (Low-Rank Adaptation)
