

UFAM - UNIVERSIDADE FEDERAL DO AMAZONAS	
Processamento de Linguagem Natural - NLP	Turma: Engenharia de Software
Atividade: Realizar Fine Tuning com LoRA	Entrega: 05/08/2024
Aluno: Davi de Menezes Gonzaga	Matrícula: 21954474

Relatório

Links:

Repositório no GitHub: [FineTuningLoRA](#)

Ferramenta de OCR: [Online2PDF](#)

Base Sintética: [base_sintetica.json](#)

Pré-Processamento:

Para realizar o pré-processamento dos dados, foi necessário utilizar uma ferramenta online gratuita capaz de utilizar tecnologia de OCR (Optical Character Recognition). Isso se fez necessário porque alguns dos PDFs disponibilizados eram apenas páginas digitalizadas, tornando a coleta de informações muito custosa.

Devido a esse problema, precisei pesquisar alternativas para acessar as informações presentes nos PDFs. Durante minha pesquisa, encontrei uma ferramenta gratuita que utiliza OCR para coletar e converter o conteúdo em arquivos TXT. Essa ferramenta é um site online chamado [Online2PDF](#). Apesar de existirem outras ferramentas disponíveis na web, optei por utilizar esse site porque ele permite a submissão de mais de um arquivo por vez para a conversão, tornando o trabalho de conversão mais eficiente.

Após converter todos os PDFs das legislações da UFAM disponibilizados, compilei todos os arquivos TXT em um único arquivo, resultando em um arquivo bastante extenso, e fiz o upload para o [GitHub](#) para facilitar o acesso posterior.

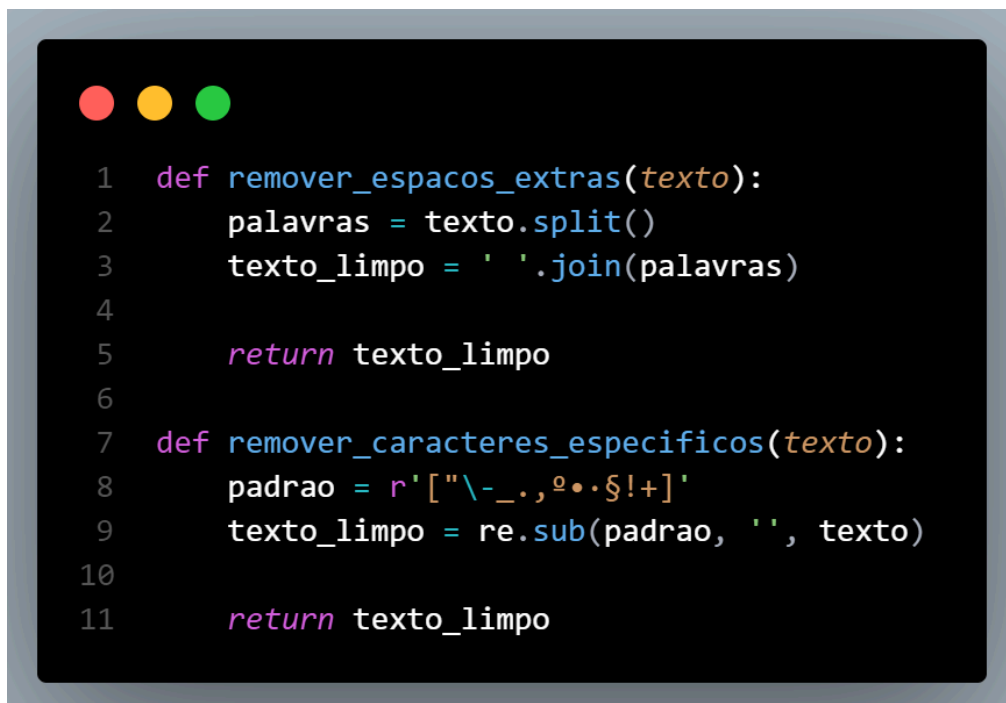
Tratamento dos Dados:

Após a coleta e o pré-processamento dos dados mencionados anteriormente, foi necessário realizar uma etapa de tratamento de dados, uma vez que as informações coletadas continham muitos "lixos", como caracteres irrelevantes e quebras de

linhas desnecessárias, entre outros dados indesejados que acabaram sendo incluídos pela ferramenta de OCR.

A primeira etapa do tratamento de dados envolveu a separação do conjunto de dados em subconjuntos para facilitar o processo. Para essa fase, escolhi separar os dados usando o caractere "\n" (quebra de linha). Cada um dos subconjuntos criados foi então passado por funções específicas de tratamento de texto.

Essas funções incluíram uma responsável por remover caracteres específicos que não agregavam valor ao contexto ou que eram erros resultantes da conversão de PDF para TXT, além de uma função dedicada a eliminar espaços desnecessários.

A screenshot of a code editor with a dark background and light-colored text. The editor has three colored window control buttons (red, yellow, green) in the top-left corner. The code is written in Python and consists of two functions. The first function, `remover_espacos_extras`, takes a string `texto` and returns it with extra spaces removed by splitting on spaces and joining back with single spaces. The second function, `remover_caracteres_especificos`, takes a string `texto` and returns it with a set of unwanted characters removed using a regular expression. The code is numbered from 1 to 11.

```
1 def remover_espacos_extras(texto):
2     palavras = texto.split()
3     texto_limpo = ' '.join(palavras)
4
5     return texto_limpo
6
7 def remover_caracteres_especificos(texto):
8     padrao = r'["\ _.,@*~$!+]'
9     texto_limpo = re.sub(padrao, '', texto)
10
11     return texto_limpo
```

Após tratar todo o texto das legislações da UFAM coletado, tomei uma decisão de projeto importante: decidi separar o texto em vários blocos. Optei por essa abordagem devido ao limite de tokens da LLM que será utilizada na próxima fase, que envolve a criação da base sintética de instruções.

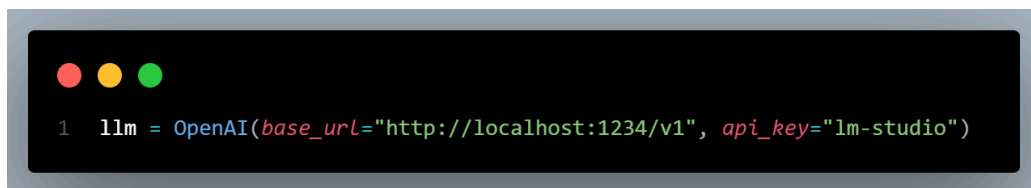
Devido ao tamanho dessa base de dados, não seria viável utilizar todo o conteúdo como contexto para a criação das perguntas. O limite de tokens do servidor local da LLM não seria suficiente para suportar o tamanho do prompt.

Criação da Base Sintética de Instruções:

Após concluir todas as etapas de processamento das legislações da UFAM para a criação da base de dados, foi possível criar uma base sintética de instruções. Para isso, optei por utilizar o software LM Studio, apresentado pelo Professor André em sala de aula. A escolha desse software foi motivada pela falta de acesso à API do Chat GPT.

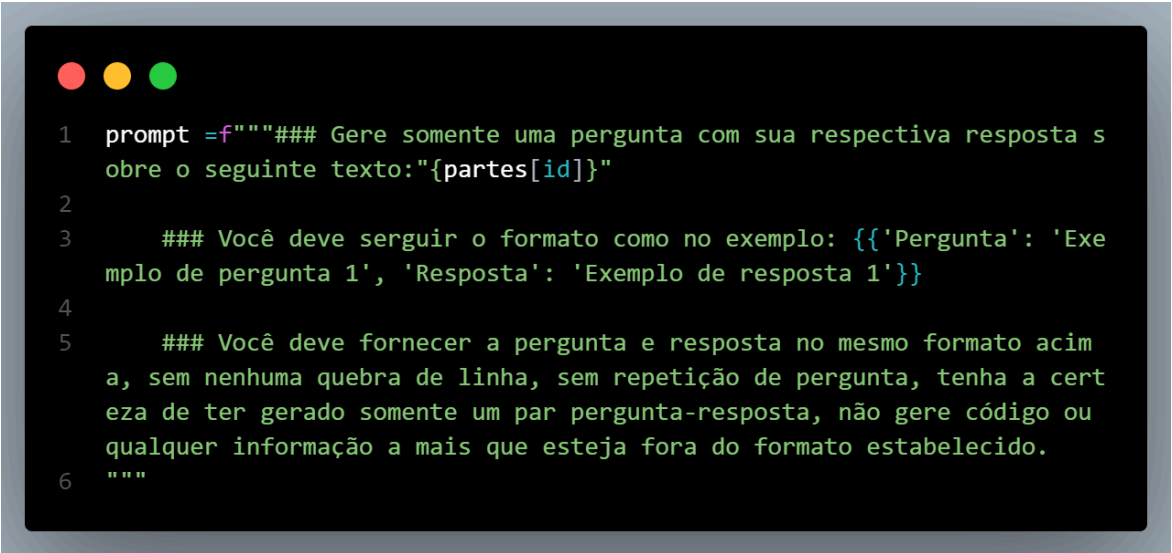
O LM Studio permite montar um servidor local de uma LLM de livre escolha. Para esta fase do trabalho, escolhi utilizar o modelo Llama da META com 8 bilhões de parâmetros, especificamente o *"Meta-Llama-3-8B-Instruct-Q4_K_M.gguf"*.

Ao iniciar o servidor dentro do LM Studio é possível acessar o modelo através da URL: *"http://localhost:1234/v1"*, para realizar as requisições na rota disponibilizada escolhi utilizar a Classe OpenAI da Biblioteca langchain_openai, ambas também apresentadas em Sala de Aula.

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. It contains a single line of Python code: `1 llm = OpenAI(base_url="http://localhost:1234/v1", api_key="lm-studio")`.

```
1 llm = OpenAI(base_url="http://localhost:1234/v1", api_key="lm-studio")
```

Após vários dias de tentativas falhas para criar prompts que gerassem respostas satisfatórias, enfrentei diversos problemas, como o limite máximo de tokens gerados, o tamanho do prompt e, o mais desafiador, fazer com que a LLM escolhida retornasse a resposta no formato desejado. No entanto, após muitas tentativas, consegui chegar a um prompt ideal para o que foi proposto. Abaixo segue o prompt selecionado:



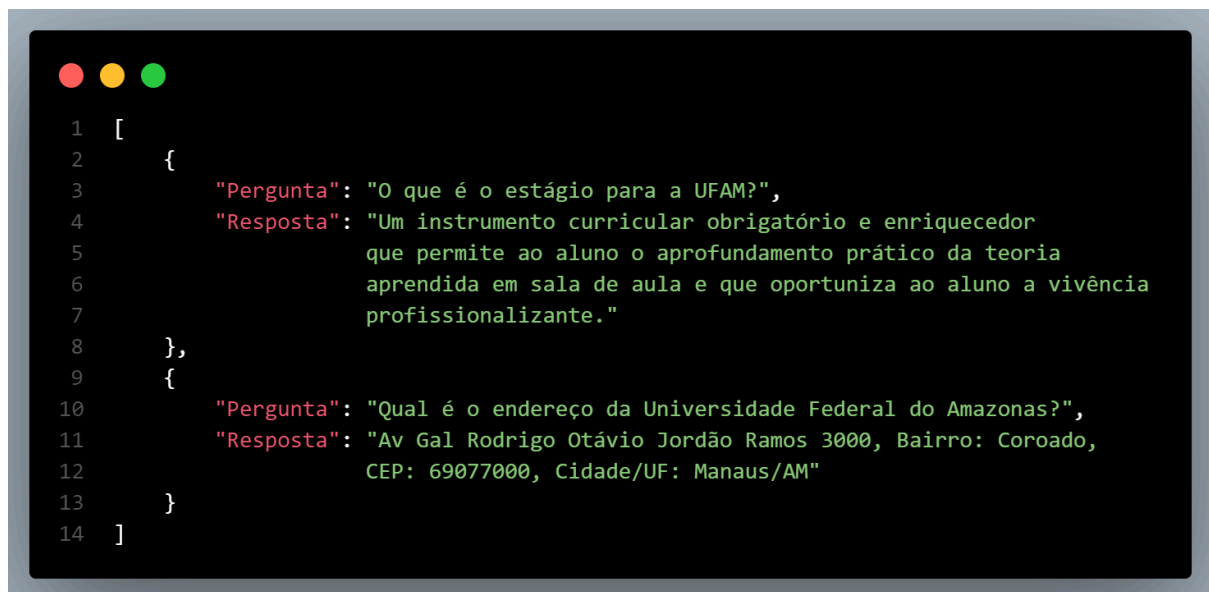
```
1 prompt =f"""### Gere somente uma pergunta com sua respectiva resposta s  
   obre o seguinte texto:"{partes[id]}"  
2  
3     ### Você deve seguir o formato como no exemplo: {'Pergunta': 'Exe  
   mplo de pergunta 1', 'Resposta': 'Exemplo de resposta 1'}}  
4  
5     ### Você deve fornecer a pergunta e resposta no mesmo formato acim  
   a, sem nenhuma quebra de linha, sem repetição de pergunta, tenha a cert  
   eza de ter gerado somente um par pergunta-resposta, não gere código ou  
   qualquer informação a mais que esteja fora do formato estabelecido.  
6 """
```

Anteriormente, mencionei que dividi a base de dados em vários subconjuntos, totalizando 806. Todos esses subconjuntos foram armazenados em uma lista denominada "partes". Para assegurar que toda a base de dados fosse abordada de forma abrangente, decidi percorrer essa lista e fornecer um novo contexto para cada prompt enviado à LLM. Dessa forma, garanti uma maior diversidade de perguntas e a cobertura completa das legislações coletadas.

Assim que a LLM, rodando localmente no LM Studio, retornava uma resposta, era necessário realizar uma etapa adicional de tratamento. Nessa etapa, o objetivo era extrair apenas as perguntas e respostas geradas pelo modelo. Esse processo se mostrou bastante desafiador, pois as respostas não seguiam um padrão consistente. Uma das soluções que encontrei foi reduzir o tamanho do prompt para que o modelo pudesse entender melhor o que era esperado e o formato desejado para as respostas. Com essa abordagem, a qualidade das respostas melhorou significativamente, permitindo a criação de uma expressão regular (regex) que se adequava ao padrão desejado para a coleta das respostas.

Outra dificuldade que encontrei ao longo deste trabalho foi relacionada ao hardware necessário para rodar e treinar modelos de linguagem. Meu computador não é dos mais potentes e, além disso, não possui uma GPU, o que acabou sendo um obstáculo significativo para a realização do trabalho. Cada novo prompt que eu desenvolvía e testava resultava em tempos de resposta consideravelmente longos, especialmente quando era necessário gerar várias respostas para identificar padrões significativos.

Após conseguir ajustar o prompt de forma adequada e identificar e extrair as informações relevantes das respostas geradas pela LLM, apliquei toda essa metodologia à base de dados completa, composta pelos 806 subconjuntos mencionados anteriormente. Como resultado, foram gerados mais de 1.000 pares de perguntas e respostas sobre as legislações da UFAM, conforme ilustrado nos exemplos apresentados abaixo.



```
1  [  
2    {  
3      "Pergunta": "O que é o estágio para a UFAM?",  
4      "Resposta": "Um instrumento curricular obrigatório e enriquecedor  
5                  que permite ao aluno o aprofundamento prático da teoria  
6                  aprendida em sala de aula e que oportuniza ao aluno a vivência  
7                  profissionalizante."  
8    },  
9    {  
10     "Pergunta": "Qual é o endereço da Universidade Federal do Amazonas?",  
11     "Resposta": "Av Gal Rodrigo Otávio Jordão Ramos 3000, Bairro: Coroadó,  
12                CEP: 69077000, Cidade/UF: Manaus/AM"  
13   }  
14 ]
```

Com a [Base Sintética](#) criada, optei por salvá-la em um arquivo JSON, que também está disponível no GitHub.

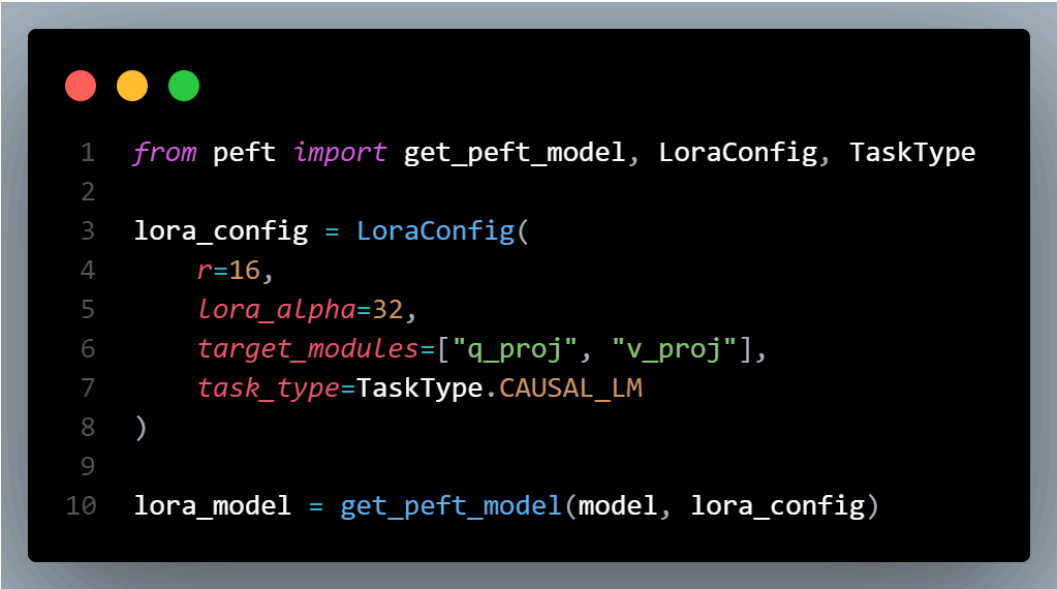
Fine-Tuning com Low-Rank Adaptation - LoRA:

Com a base sintética criada anteriormente, agora é possível realizar o fine-tuning utilizando a técnica Low-Rank Adaptation (LoRA). Para isso, escolhi usar o modelo pré-treinado Llama 2 com 7 bilhões de parâmetros, especificamente o "sharpbai/Llama-2-7b-hf".

Para baixar o modelo pré-treinado, utilizei a classe *AutoModelForCausalLM* da biblioteca *transformers*. Em seguida, converti a base de dados sintética em um dataset compatível, utilizando a classe *Dataset* da biblioteca *datasets*, para garantir a integração adequada com o modelo. Após a conversão, realizei a tokenização dos dados utilizando o tokenizador correspondente ao modelo Llama 2. Este processo é essencial para preparar os dados de entrada de forma que sejam compreendidos pelo modelo durante o fine-tuning.

Após o tratamento da base sintética para torná-la compatível com o modelo Llama 2, foi necessário configurar a técnica Low-Rank Adaptation (LoRA). Para isso, utilizei as classes *LoraConfig* e *get_peft_model*, disponíveis na biblioteca *peft*. A *LoraConfig* permite definir os parâmetros específicos para a adaptação de baixo-ranque, como o número de rótulos e as dimensões dos componentes baixos-rank. A função *get_peft_model* é então utilizada para integrar a configuração LoRA com o modelo pré-treinado.

Essa configuração é crucial para adaptar o modelo às características da base sintética sem a necessidade de ajustar todos os parâmetros do modelo, permitindo um fine-tuning mais eficiente e eficaz.

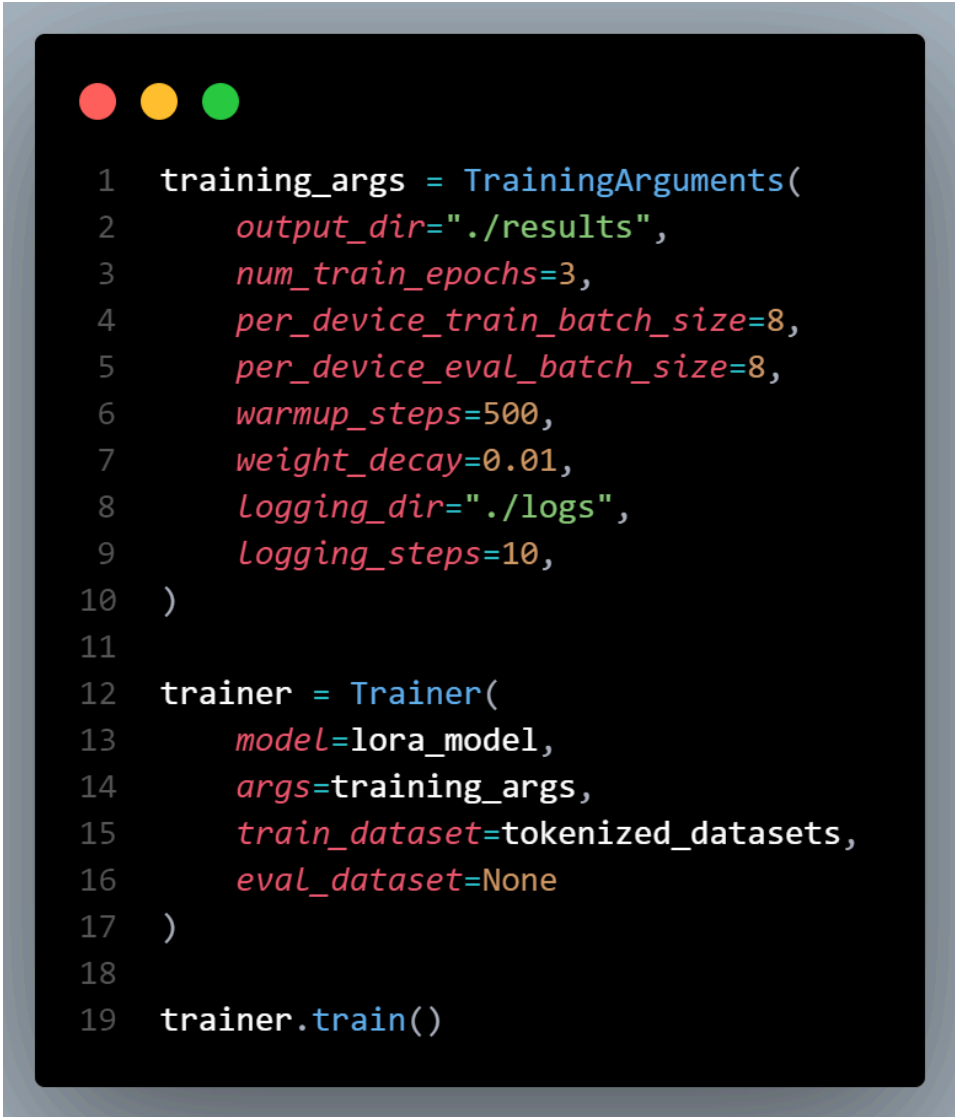


```
1 from peft import get_peft_model, LoraConfig, TaskType
2
3 lora_config = LoraConfig(
4     r=16,
5     lora_alpha=32,
6     target_modules=["q_proj", "v_proj"],
7     task_type=TaskType.CAUSAL_LM
8 )
9
10 lora_model = get_peft_model(model, lora_config)
```

Em seguida, foi necessário configurar a fase de treinamento do modelo, que corresponde ao fine-tuning propriamente dito. Para isso, utilizei as classes *Trainer* e *TrainingArguments* da biblioteca *transformers*. A classe *Trainer* é responsável por gerenciar o processo de treinamento, incluindo a execução das etapas de treinamento e avaliação. No objeto da classe *Trainer*, passei a base de treino, que é a base sintética criada anteriormente.

Os *TrainingArguments* foram configurados para definir os parâmetros do treinamento, como o número de épocas, a taxa de aprendizado, o tamanho do lote e outros hiperparâmetros importantes. Essas configurações são cruciais para otimizar

o desempenho do fine-tuning e garantir que o modelo se adapte de maneira eficaz aos dados fornecidos.



```
1  training_args = TrainingArguments(  
2      output_dir="./results",  
3      num_train_epochs=3,  
4      per_device_train_batch_size=8,  
5      per_device_eval_batch_size=8,  
6      warmup_steps=500,  
7      weight_decay=0.01,  
8      logging_dir="./logs",  
9      logging_steps=10,  
10 )  
11  
12 trainer = Trainer(  
13     model=lora_model,  
14     args=training_args,  
15     train_dataset=tokenized_datasets,  
16     eval_dataset=None  
17 )  
18  
19 trainer.train()
```

Para essa fase do trabalho, a maior dificuldade continuou sendo o hardware necessário para o treinamento e execução do modelo. Como mencionei anteriormente, não tenho acesso a uma GPU, que é uma ferramenta extremamente importante para o treinamento de modelos de linguagem devido ao seu poder de processamento paralelo. A ausência de uma GPU resultou em tempos de treinamento muito longos e dificuldades para lidar com grandes volumes de dados.

Mesmo as ferramentas online, como Google Colab e Kaggle, oferecem GPUs com limitações de recursos, como 15 GB de memória de GPU, o que é insuficiente para o treinamento eficiente de modelos. Esses limites podem ser um impedimento

significativo, pois o treinamento de modelos de linguagem com milhões de parâmetros requer não apenas GPUs, mas também uma quantidade adequada de memória e capacidade de processamento.

Essa restrição de hardware impactou a eficiência e a velocidade do processo de treinamento, tornando o ajuste fino e a execução do modelo um desafio maior.

Conclusão

Este trabalho envolveu um processo detalhado de pré-processamento, criação de uma base sintética de instrução e fine-tuning de um modelo de linguagem pré-teinado utilizando a técnica Low-Rank Adaptation (LoRA). Iniciei com a conversão de PDFs digitalizados em texto, superando desafios iniciais relacionados a caracteres irrelevantes e quebras de linha através da utilização de ferramentas online como o Online2PDF. Após o tratamento e a formatação adequada dos dados, foi gerada uma base sintética de instruções, armazenada em um arquivo JSON para acessibilidade e uso futuro.

O fine-tuning foi realizado com o modelo Llama 2, utilizando as classes *LoraConfig* e *get_peft_model* da biblioteca *peft* para configurar a técnica LoRA. As fases de treinamento foram gerenciadas por meio das classes *Trainer* e *TrainingArguments* da biblioteca *transformers*, garantindo a adaptação eficiente do modelo aos dados fornecidos.

Entretanto, a falta de acesso a uma GPU e as limitações de recursos das ferramentas online, como Google Colab e Kaggle, representaram desafios significativos, resultando em tempos de treinamento prolongados e limitações na capacidade de processamento. Esses desafios ressaltam a importância de um hardware adequado para o treinamento de modelos de linguagem e a necessidade de estratégias eficientes para a gestão de dados e o ajuste de modelos.