



Universidade Federal do Amazonas – UFAM
Instituto de Computação – IComp
Programa de Pós-Graduação em Informática - PPGI

Tópicos Especiais em Recuperação da Informação | Docentes: Prof Dr Altigran da Silva e Prof Dr André Carvalho

Trabalho Prático 3 (TP3)

LLM sobre Legislação

Acadêmica da UFAM

Discente: Fabrizio Honda Franzoia
Matrícula: 2230647

Manaus - AM, 2024

Mestrado em Informática (PPGI)
PPGINF528

Informações Gerais

- **Objetivo:** desenvolver uma LLM (Large Language Model) capaz de responder perguntas sobre a legislação acadêmica de Graduação da Universidade Federal do Amazonas (UFAM)
- **Etapas:** download e pré-processamento da Legislação, geração de base de dados sintética de instruções, treinamento do modelo de linguagem com LoRA e implementação de RAG
- **Entregáveis:** notebook do Google Colab (modelo treinado e sistema de RAG implementado), link da base de dados sintética e relatório (este documento)



Etapa 1 - Pré-processamento da legislação

1 - Acesso à Legislação

O primeiro passo consistiu em acessar o site onde toda documentação da legislação acadêmica da UFAM está disponível



2 - Download dos arquivos

Em seguida, cada documento foi acessado individualmente e fez-se o download dos arquivos, um por um



3 - Combinação em um único pdf

Utilizando a ferramenta Adobe Acrobat, realizou-se a mesclagem dos documentos em um único arquivo, para facilitar o processo



4 - Conversão de pdf para txt

Utilizou-se a opção de “Reconhecer texto” do Acrobat para transformar as imagens em textos, exportando depois o arquivo como .txt



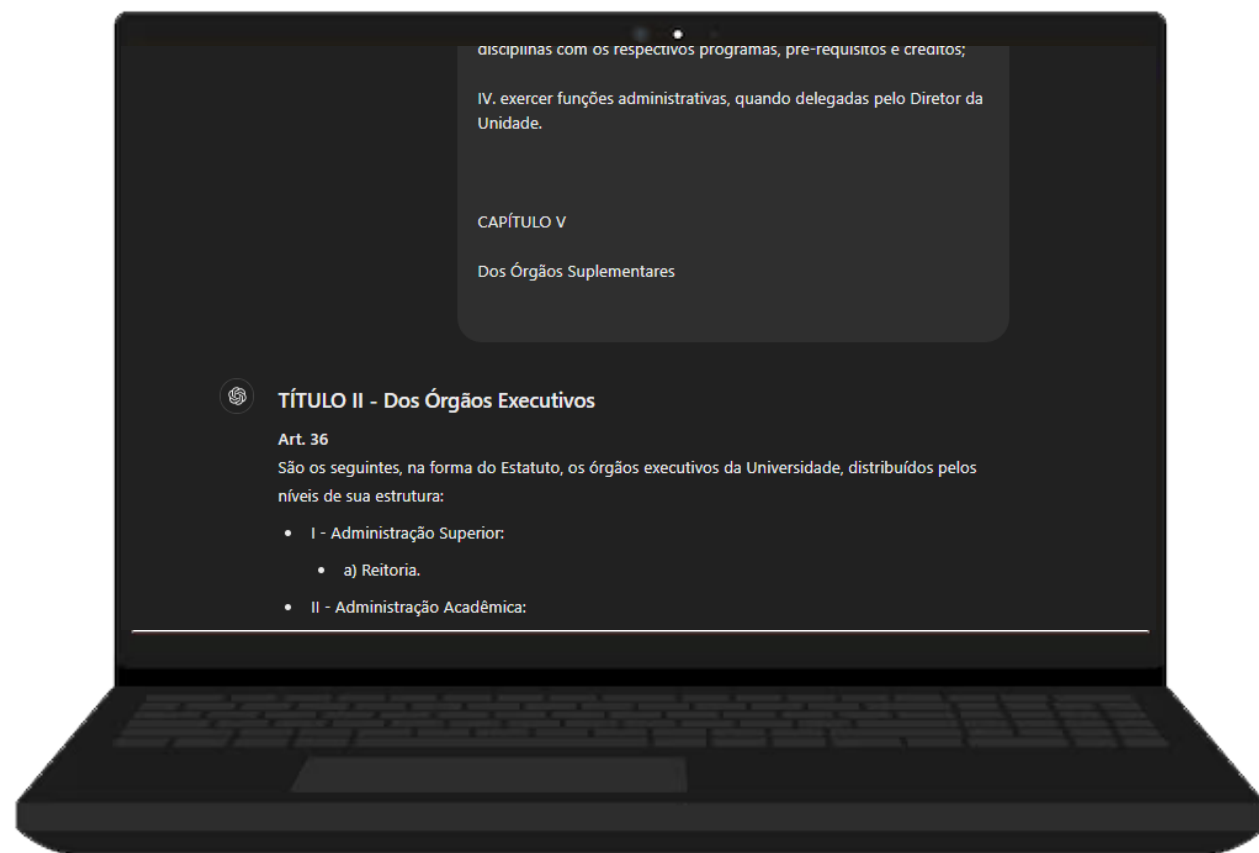
5 - Tratamento dos textos via GPT

Com o Chat-GPT 4o, os textos foram tratados (caracteres formatados de forma inadequada, espaços em branco adicionais, etc)



Etapa 1 - Pré-processamento da legislação

- A estratégia de mesclar todos os arquivos em um único PDF foi adotada visando enviar um único arquivo ao notebook no Colab. O Adobe Acrobat foi escolhido para reconhecer o texto das imagens, pois possui uma opção específica para esta tarefa
- Considerando a qualidade das imagens, foi necessário realizar um tratamento dos textos. Inicialmente, buscou-se realizar via Colab mas o Chat-GPT 4o pareceu ser mais adequado. Como o arquivo .txt era muito grande, pequenas partes de texto foram enviadas ao modelo de cada vez
- Como resultado, o arquivo original continha +317k de caracteres e, após o tratamento, +244k



Etapa 2 - Base de Dados Sintética

```
# Função para gerar perguntas e respostas usando T5
def gerar_perguntas_respostas(text, num_perguntas=10):
    perguntas_respostas = []
    topicos = [
        "autonomia", "estrutura", "finalidade", "princípios",
        "constituição básica", "administração", "pesquisa",
        "extensão", "corpo docente", "corpo discente",
        "gestão", "financeira", "patrimonial", "ensino",
        "unidades acadêmicas", "estatuto", "regimento geral",
        "órgãos deliberativos", "representação estudantil",
        "admissão", "matrícula", "avaliação", "diploma",
        "trancamento", "transferência", "provas", "créditos",
        "calendário acadêmico", "regime didático", "currículo",
        "disciplinas", "estágio", "trabalho de conclusão de curso",
        "mobilidade acadêmica", "intercâmbio", "convênios",
        "assistência estudantil", "bolsas", "auxílios",
        "programas de apoio", "monitoria", "iniciação científica",
        "atividades complementares", "extensão universitária",
        "responsabilidade social", "sustentabilidade",
        "ética", "direitos humanos", "diversidade",
        "inclusão", "acessibilidade", "política de cotas",
        "biblioteca", "laboratórios", "infraestrutura",
        "serviços", "segurança", "saúde", "esporte",
        "cultura", "arte", "meio ambiente"
    ]

    templates_perguntas = [
        "Quais são os detalhes sobre {topico} na legislação?",
        "O que a legislação diz sobre {topico}?",
        "Quais são as regras para {topico}?",
        "Quem é responsável por {topico}?",
        "Explique o conceito de {topico} na legislação."
    ]

    # Dividir o texto em partes menores para evitar ultrapassar o limite do modelo
    partes_texto = [text[i:i+2000] for i in range(0, len(text), 2000)]

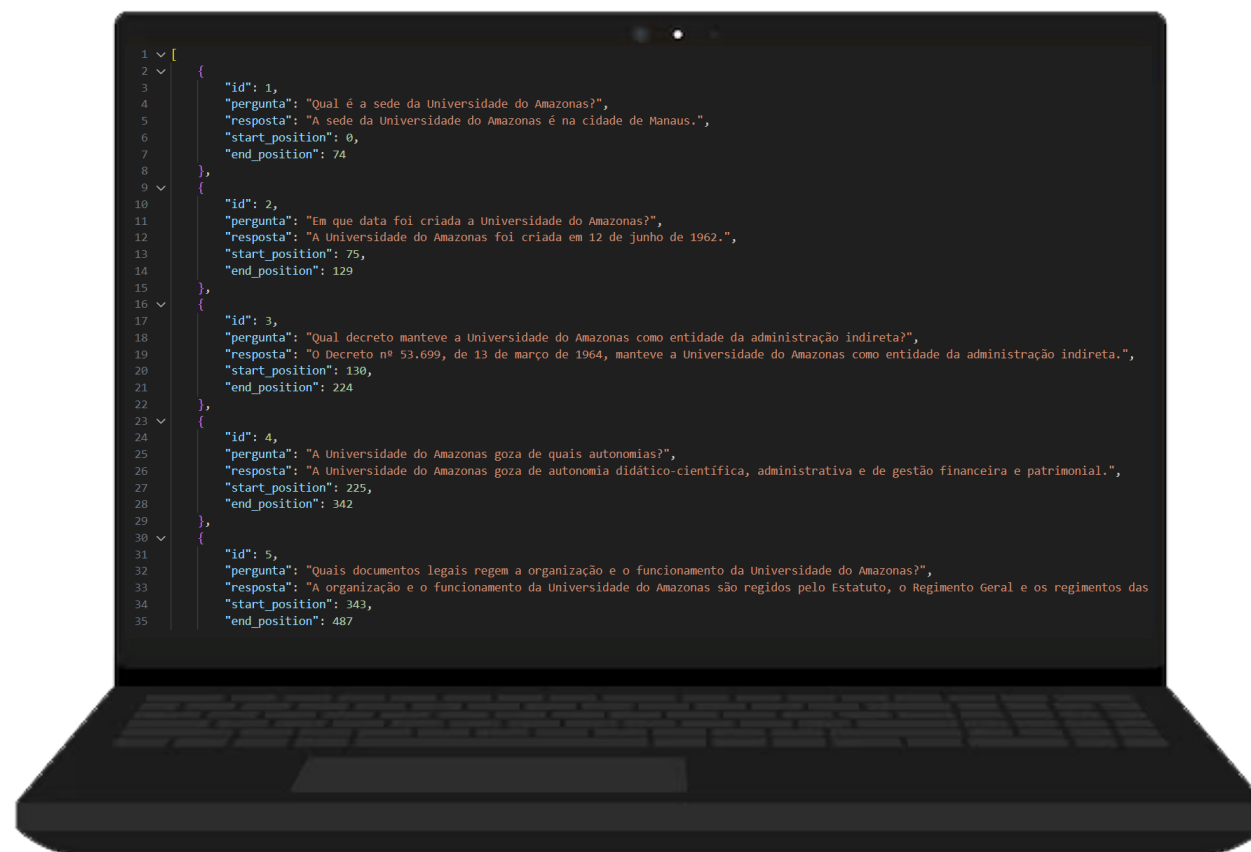
    for _ in tqdm(range(num_perguntas), desc="Gerando perguntas"): # Barra de progresso
        topico = random.choice(topicos)
        parte_aleatoria = random.choice(partes_texto)
        template_pergunta = random.choice(templates_perguntas)
        pergunta = template_pergunta.format(topico=topico)

        input_text = f"pergunta: {pergunta}\ncontexto: {parte_aleatoria}\nresposta:"
        input_ids = tokenizer.encode(input_text, return_tensors='pt', truncation=True, max_length=512).to(device) # Truncar se necessário
```

- De forma equivalente ao tratamento do texto, buscou-se criar a base de dados sintética via Google Colab. Entretanto, as perguntas ficaram, em maioria, genéricas e incoerentes. Por esse motivo, optou-se também por utilizar o Chat-GPT 4o para geração da base
- O código ao lado (que não é mais utilizado) ilustra como as perguntas eram geradas na primeira versão do modelo. Um exemplo de pergunta/resposta gerada por ele:
 - Pergunta 6: Quem é responsável por diploma?
 - Resposta 6: __//, DOU no..s./..

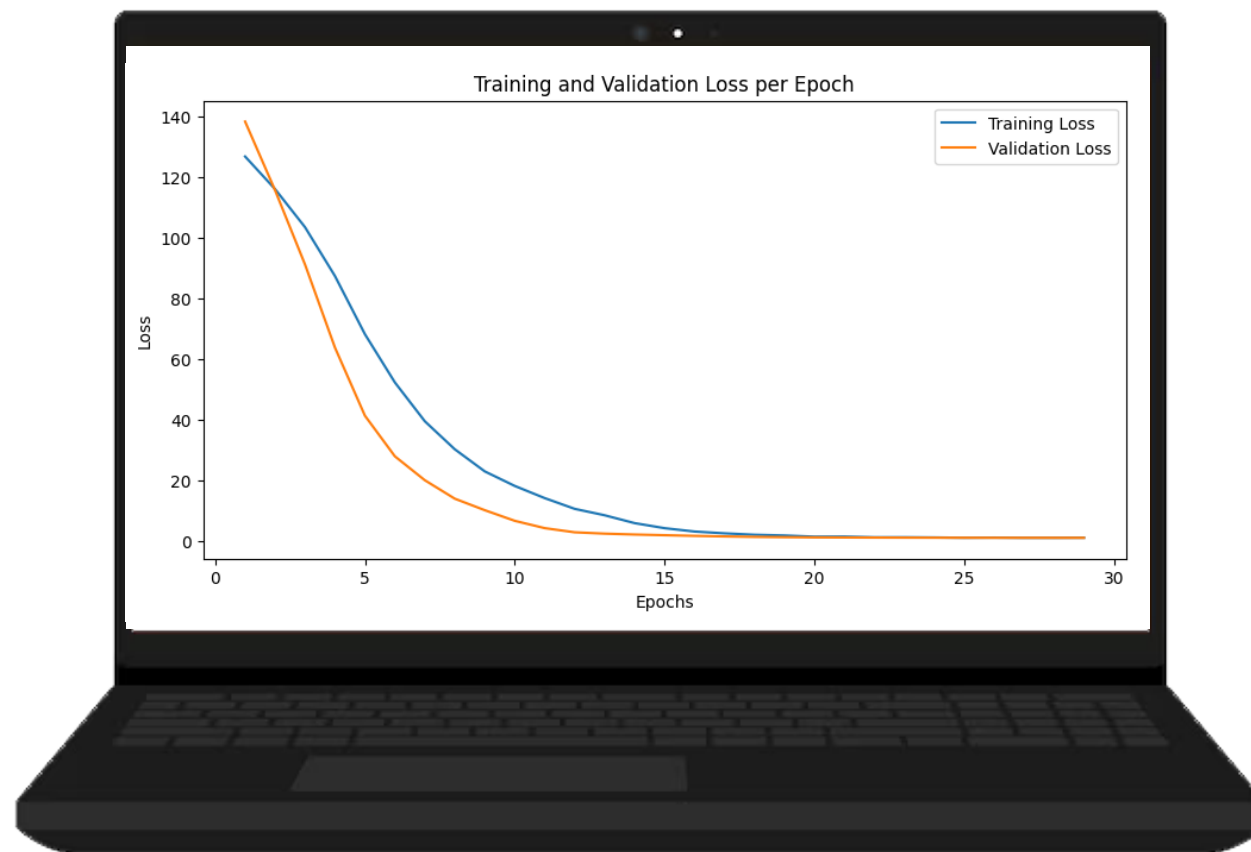
Etapa 2 - Base de Dados Sintética

- Portanto, a geração das perguntas deu-se via solicitação de *prompt* no GPT 4o. O modelo retornou um arquivo JSON com as perguntas e respostas, ilustrado ao lado
- No início, 1000 perguntas eram geradas. Entretanto, devido à limitações do modelo, notou-se que grande parte eram somente cópias de outras perguntas e respostas, apesar do *prompt* solicitar que fossem distintas
- Por conta disso, optou-se por uma base sintética de somente 100 perguntas. Optou-se por essa escolha, de 100 perguntas adequadas, do que 1000 via Google Colab que estavam incoerentes



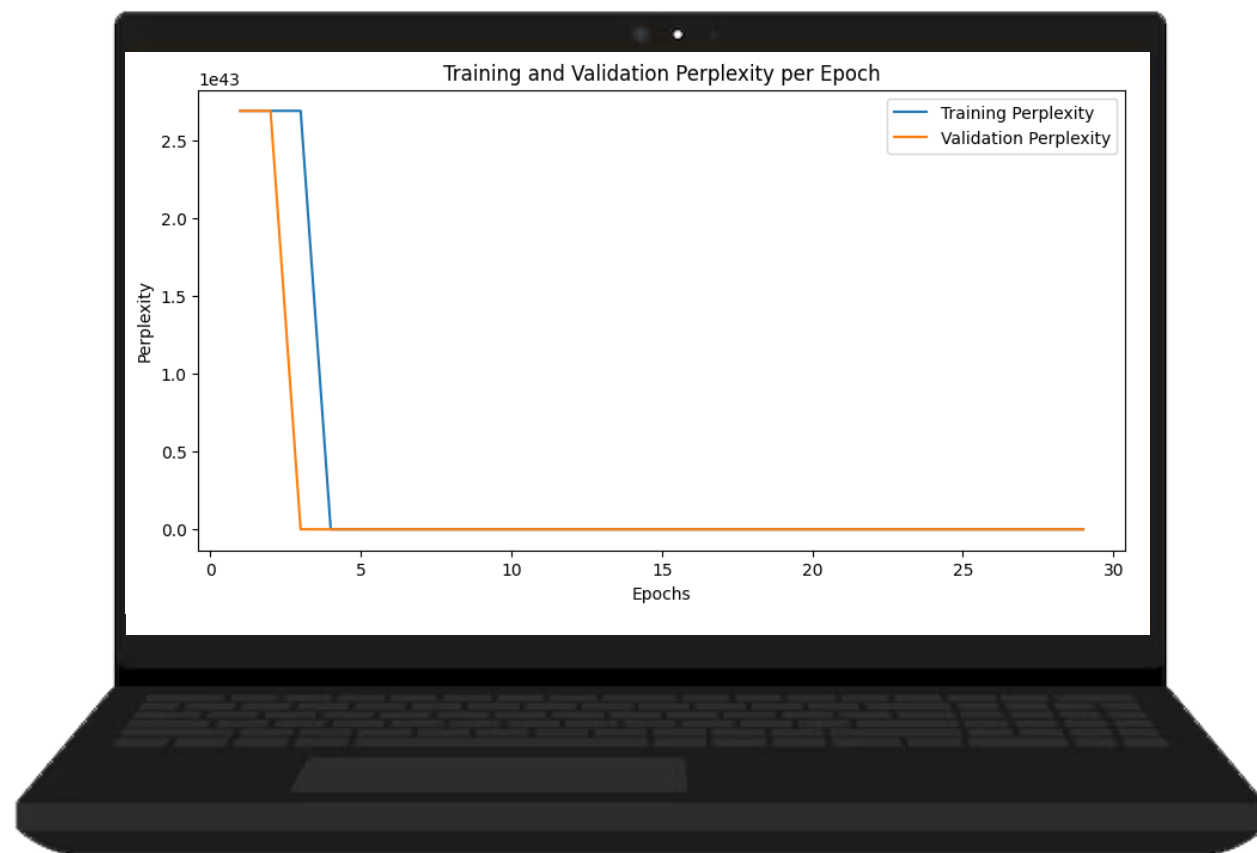
Etapa 3 - Treinamento do modelo com LoRA

- Nas primeiras versões do modelo, estava-se utilizando o *t5-small*. Apesar de ser em inglês, os resultados estavam sendo satisfatórios no treinamento. Porém, ao decorrer da elaboração do projeto, optou-se pelo *PTT5 (unicamp-dl/ptt5-base-portuguese-vocab)* por ser especificamente para o idioma português-brasileiro
- Ao longo do treinamento, a melhor solução foi utilizar 30 épocas com um *learning rate* de 3^{-5} . Utilizou-se um otimizador para minimizar a função de perda e um agendador para ajustar a taxa de aprendizado do modelo
- Tanto o *loss* de treino (para ajustar os pesos) quanto o de validação iniciaram com valores altos ($+100$), mas foram decaindo ao longo do treinamento. Os valores finais foram, respectivamente, ≈ 1.07 e ≈ 1.15



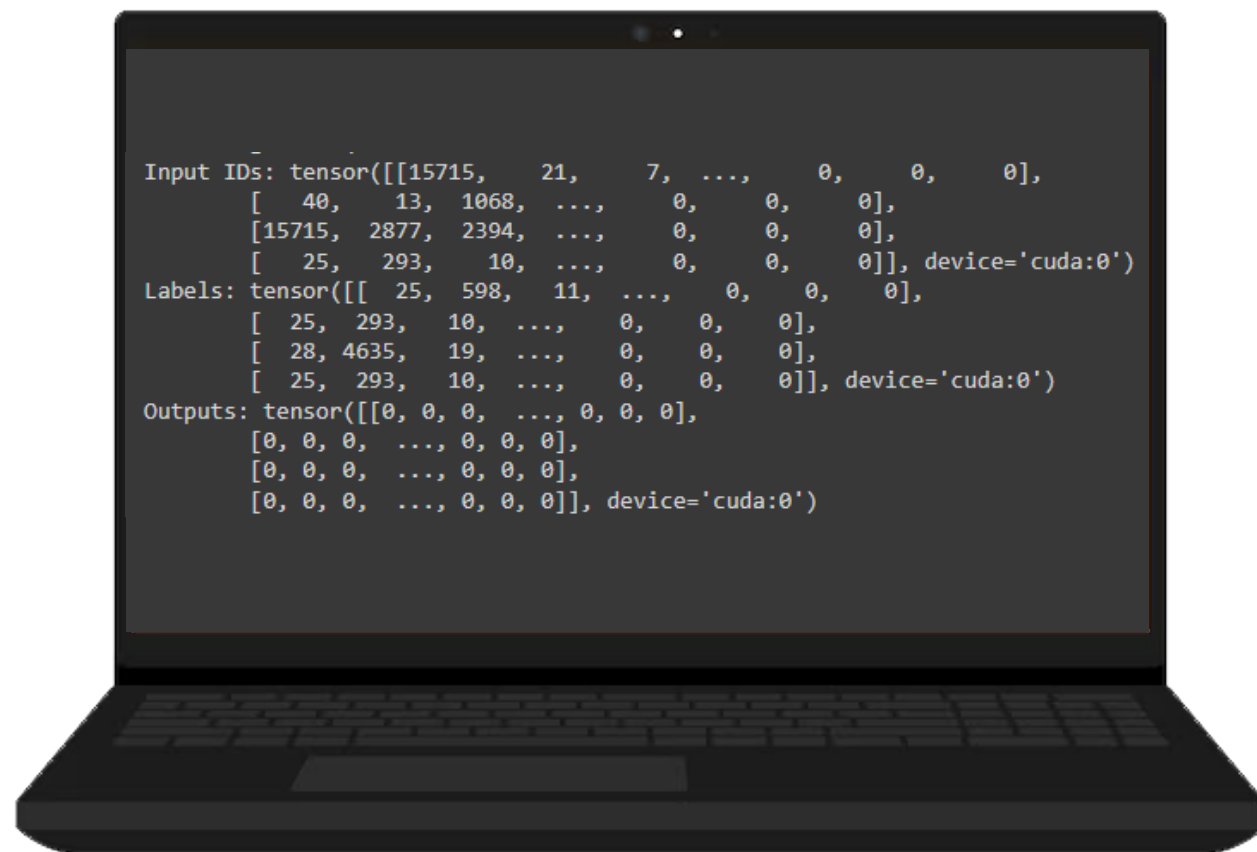
Etapa 3 - Treinamento do modelo com LoRA

- Em se tratando do cálculo da perplexidade, notou-se que os valores (treinamento e validação) iniciaram extremamente altos, com 1^{43} , e depois caíram drasticamente, convergindo para 0
- Sobre isso, há duas conclusões: (i) o modelo possivelmente estava prevendo bem a próxima palavra, indicando aprendizado; no entanto, (ii) a convergência para zero é um comportamento atípico, provavelmente relevante um superajuste do modelo aos dados de treinamento



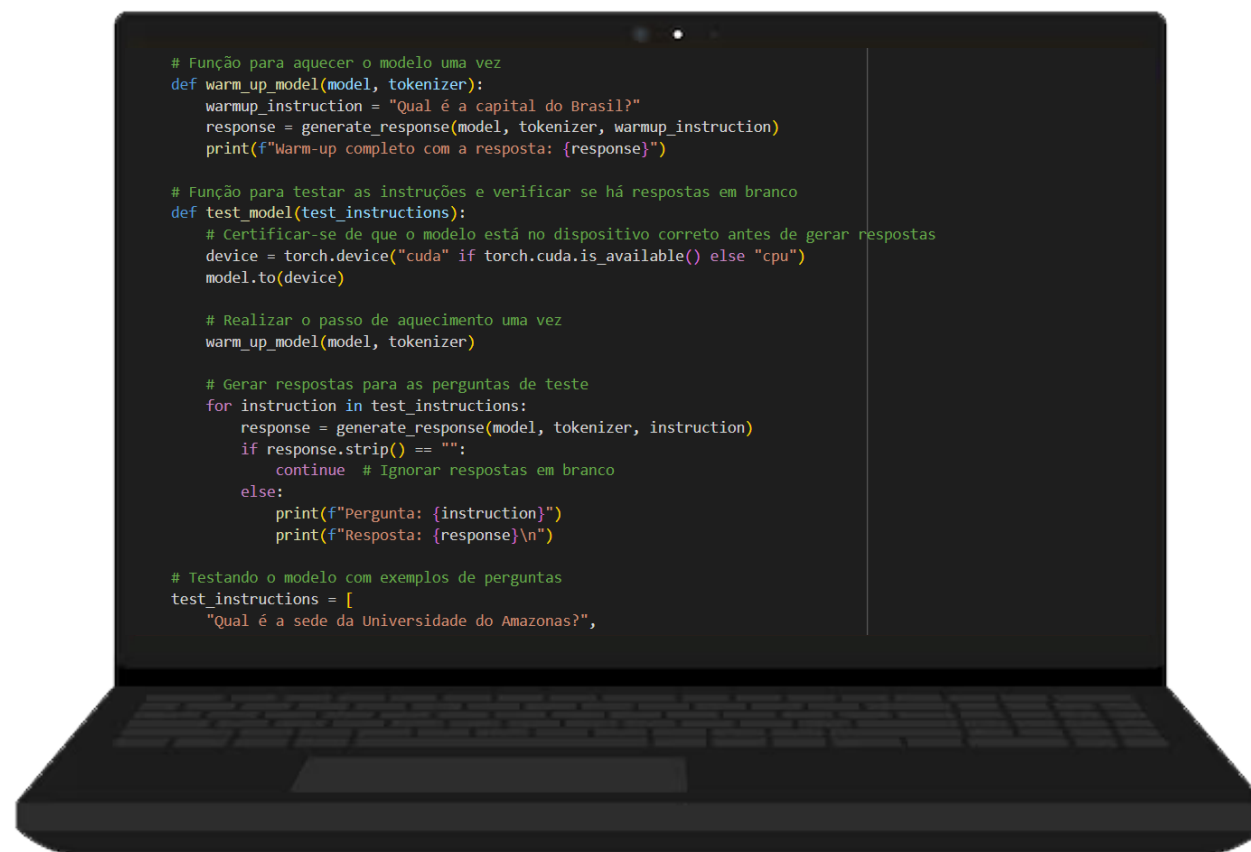
Etapa 3 - Treinamento do modelo com LoRA

- Optou-se por não mensurar acurácia, visto que não é a melhor forma de avaliação em atividades de processamento de texto. De tal modo, utilizou-se o BLEU (Bilingual Evaluation Understudy) para avaliar a qualidade do texto gerado
- Entretanto, o valor do BLEU para o modelo foi de 0.0, notando-se que os tensores gerados estavam sendo vazios – um comportamento não convencional. Ao analisar o código, nota-se que o problema não estava no treinamento, que ocorreu adequadamente



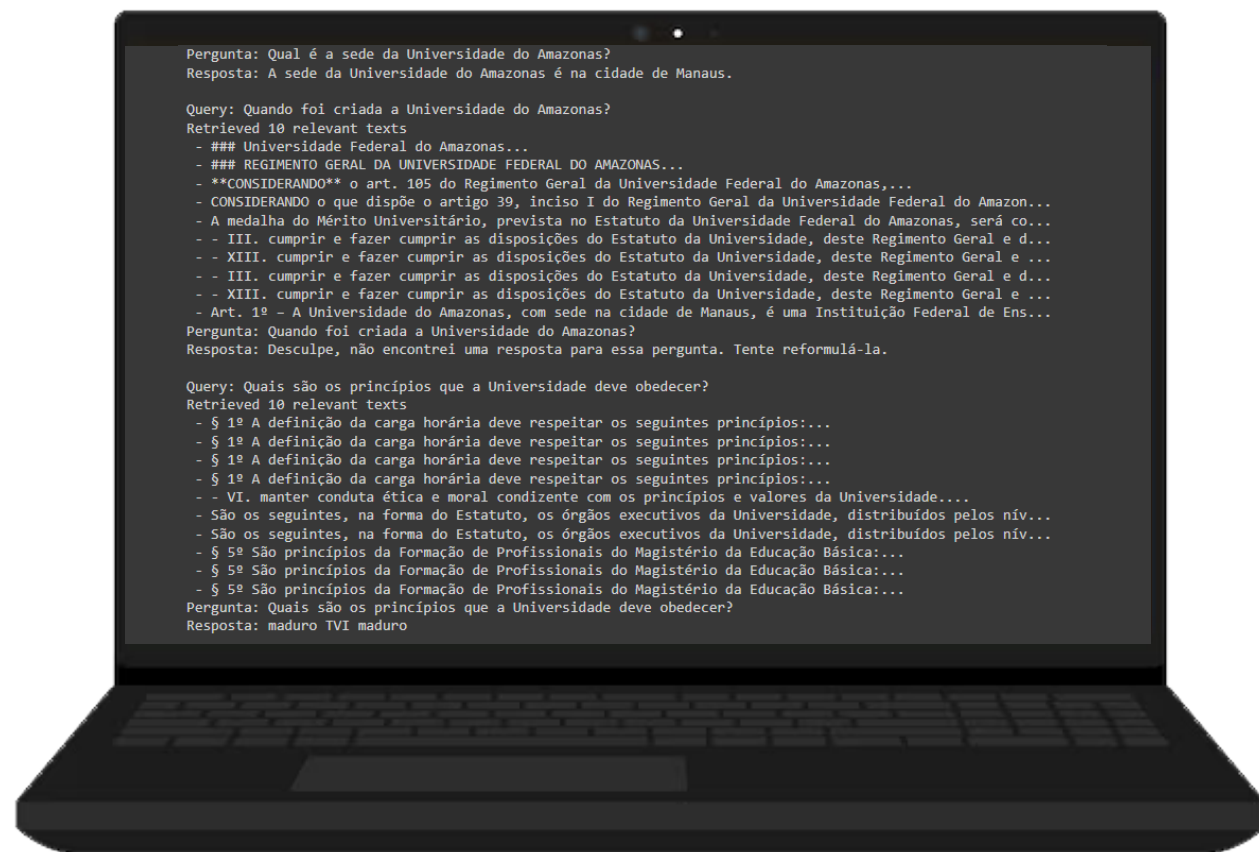
Etapa 3.5 - Testes do Modelo

- Previamente à utilização de RAG, buscou-se realizar testes com o modelo a partir do treinamento, para verificar se estava adequado. As perguntas de teste estavam sendo respondidas corretamente, mas as via *prompt* estavam obtendo valores nulos
- Uma alternativa, que não está mais presente no código, foi o processo de *warm up* (aquecimento), de modo que o modelo pudesse responder as perguntas adequadamente só após estar pronto (“aquecido”). Como o processo de RAG foi utilizado posteriormente, esta etapa acabou não sendo mais necessária e foi descontinuada



Etapa 4 - Implementação de RAG

- Para implementar RAG, inicialmente optou-se pelo Elasticsearch, mas depois percebeu-se que ele não tem suporte diretamente para o Google Colab. Portanto, para fins de praticidade, escolheu-se o Whoosh como mecanismo de busca, que possui suporte ao Colab
- De forma geral, o modelo funciona da seguinte forma: (i) se a pergunta for idêntica à base sintética, o modelo acessa o JSON e retorna a resposta; já (ii) se a pergunta for outra, o modelo utiliza o RAG para consultar o *corpus* e retornar uma resposta contextualmente relacionada
- No início, o modelo provavelmente tentava encontrar a pergunta completa no *corpus*, retornando em respostas vazias. Uma solução foi implementar a consulta ao *corpus* via palavras-chave
- Os resultados melhoraram, no entanto, apesar de encontrar contextos relevantes no *corpus*, as respostas do modelo são vazias ou totalmente incoerentes e destoantes. Esse fato é curioso e controverso, visto que o contexto é adequado e que o treino do modelo foi bom



Explicação do código

```
# Instalação das bibliotecas necessárias  
!pip install transformers datasets loralib whoosh sacrebleu
```

```
# Importação das bibliotecas necessárias  
import json, os, torch, numpy as np  
import torch.nn as nn  
import matplotlib.pyplot as plt  
from google.colab import files, drive  
from datasets import Dataset, load_metric  
import loralib as lora  
from whoosh.index import create_in, open_dir  
from whoosh.fields import Schema, TEXT  
from whoosh.qparser import QueryParser  
from IPython.display import display, HTML  
from transformers import (  
    AutoTokenizer, AutoModelForSeq2SeqLM,  
    Seq2SeqTrainingArguments, Seq2SeqTrainer,  
    get_linear_schedule_with_warmup  
)
```

Instalação
das bibliotecas



Importação
das bibliotecas



transformers: manipular os modelos pré-treinados da HuggingFace, neste caso, para auxiliar na tokenização e geração de texto

datasets: utilizar os conjuntos de dados de ML e NLP para treinamento do modelo

loralib: para aplicar técnicas de Low-Ranking Adaptation (LoRA)

whoosh: para poder utilizar o mecanismo de busca, visando indexação e busca

Sacrebleu: visando avaliar o modelo via técnica BLEU

json para manipular o arquivo correspondente, *matplotlib* para os gráficos, *files* para enviar os arquivos, *drive* para gerar o .zip, dentre outros

Explicação do código

```
[ ] # Carregamento do JSON no Google Colab (perguntas e respostas sobre a Legislação Acadêmica da UFAM)
display(HTML('<h4>Faça o upload do arquivo .JSON (perguntas e respostas sobre a Legislação Acadêmica da UFAM)</h4>'))
uploaded = files.upload()

# Definição do nome do arquivo JSON carregado
file_name_json = list(uploaded.keys())[0]

# Abertura e leitura do arquivo JSON
with open(file_name_json, 'r', encoding='utf-8') as file:
    data = json.load(file)

# Carregamento do corpus de texto no Google Colab (arquivo .txt com a Legislação Acadêmica da UFAM)
display(HTML('<h4>Faça o upload do arquivo .txt (corpus da Legislação Acadêmica da UFAM)</h4>'))
uploaded_txt = files.upload()
file_name_txt = list(uploaded_txt.keys())[0]

# Especificação do esquema do Whoosh
schema = Schema(content=TEXT(stored=True))

# Criação do diretório de índice, se não existir
if not os.path.exists("indexdir"):
    os.mkdir("indexdir")

# Criação do índice Whoosh
ix = create_in("indexdir", schema)

# Indexação do corpus de texto
writer = ix.writer()
with open(file_name_txt, 'r', encoding='utf-8') as f:
    for line in f:
        writer.add_document(content=line.strip())
writer.commit()

# Verificação da conclusão da indexação
print(f"Indexação completa.")

# Abertura do índice e verificação dos primeiros documentos indexados
ix = open_dir("indexdir")
with ix.searcher() as searcher:
    results = searcher.documents()
    for i, result in enumerate(results):
        if i < 5: # Impressão dos primeiros 5 documentos para verificação
            print(f"Documento {i}: {result['content'][:100]}...")
```

Arquivo JSON e TXT

Área para enviar o arquivo JSON e TXT, respectivamente, referente à base de dados sintética com perguntas e respostas sobre a legislação acadêmica da UFAM e sobre o *corpus* relacionado

Indexação via Whoosh

Prepara o esquema do índice, cria um diretório para armazená-lo e indexa o arquivo (no caso, o *corpus*), possibilitando consultas eficientes

Teste da indexação

Imprime uma mensagem quando a indexação é finalizada e realiza uma verificação, exibindo as 5 primeiras linhas do arquivo indexado

Explicação do código

```
Faça o upload do arquivo .JSON (perguntas e respostas sobre a Legislação Acadêmica da UFAM)
Escolher arquivos legislacao_...a_base.json
• legislacao_academica_base.json(application/json) - 44134 bytes, last modified: 02/08/2024 - 100% done
Saving legislacao_academica_base.json to legislacao_academica_base (2).json

Faça o upload do arquivo .txt (corpus da Legislação Acadêmica da UFAM)
Escolher arquivos legislacao_..._corrigido.txt
• legislacao_academica_corrigido.txt(text/plain) - 363208 bytes, last modified: 31/07/2024 - 100% done
Saving legislacao_academica_corrigido.txt to legislacao_academica_corrigido (2).txt
Indexação completa.
Documento 0: TÍTULO I...
Documento 1: ...
Documento 2: DA UNIVERSIDADE...
Documento 3: ...
Documento 4: Art. 1º - A Universidade do Amazonas, com sede na cidade de Manaus, é uma Instit
```

Resultado

Como observado ao lado, o arquivo JSON e TXT foram enviados, cuja indexação do *corpus* ocorreu com sucesso. Em seguida, as cinco primeiras linhas do arquivo foram exibidas (até 100 caracteres), indicando que ocorreu corretamente

Explicação do código

```
# Transformação dos dados em um formato apropriado para treinamento
train_data = [{"instruction": item["pergunta"], "response": item["resposta"]} for item in data]

# Preparação dos Dados
dataset = Dataset.from_list(train_data)
dataset = dataset.train_test_split(test_size=0.1)

# Carregamento do Modelo e Tokenizer
model_name = "unicamp-dl/ptt5-base-portuguese-vocab" # Modelo em português (derivado do T5)
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForSeq2SeqLM.from_pretrained(model_name, tie_word_embeddings=False) # Desativação do compartilhamento de peso

# Modificação das camadas de atenção para usar LoRA
def modify_with_lora(model):
    for name, module in model.named_modules():
        if isinstance(module, nn.Linear):
            if 'q' in name or 'v' in name: # Aplicação do LoRA apenas às camadas 'q' e 'v'
                lora.mark_only_lora_as_trainable(module)
                lora.Linear(module.in_features, module.out_features, r=8)

modify_with_lora(model)
model.train()

# Função para pré-processamento dos dados
def preprocess_function(examples):
    model_inputs = tokenizer(
        examples["instruction"], max_length=512, padding="max_length", truncation=True
    )
    with tokenizer.as_target_tokenizer():
        labels = tokenizer(
            examples["response"], max_length=512, padding="max_length", truncation=True
        )

    # Ajuste para garantir compatibilidade com o modelo
    model_inputs["labels"] = labels["input_ids"]
    return model_inputs
```

Transformação e
preparação dos
dados + modelo e
tokenizer



A primeira etapa do código de treinamento é transformar os dados no formato pergunta/resposta para o treinamento. Em seguida, os dados são preparados via funções da biblioteca *dataset* e, logo após, o modelo e tokenizador são carregados. O PPT5 foi selecionado, uma versão ajustada do T5 que funciona em português-brasileiro. OBS: a desativação do compartilhamento de peso foi necessário, visto que estava causando conflito posteriormente

Ajuste para LoRA e
pré-processamento



A função *modify_with_lora* faz ajustes para que o modelo utilize LoRA nas camadas de atenção q e v, já a *preprocess_function* realiza a tokenização das instruções e respostas, visando que os dados sejam compatíveis com o modelo no treinamento

Explicação do código

```
# Pré-processamento dos dados
tokenized_datasets = dataset.map(preprocess_function, batched=True)

# Configuração dos Argumentos de Treinamento
training_args = Seq2SeqTrainingArguments(
    output_dir="./results",
    evaluation_strategy="epoch",
    learning_rate=3e-5,
    per_device_train_batch_size=4,
    per_device_eval_batch_size=4,
    weight_decay=0.01,
    save_total_limit=3,
    num_train_epochs=30,
    fp16=True,
    push_to_hub=False,
    gradient_accumulation_steps=2,
    logging_dir='./logs',
    logging_steps=10
)

# Inicialização do Treinador
trainer = Seq2SeqTrainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_datasets["train"],
    eval_dataset=tokenized_datasets["test"],
    tokenizer=tokenizer,
)

# Cálculo do Número Total de Passos de Treinamento
total_steps = len(tokenized_datasets["train"]) // training_args.gradient_accumulation_steps * training_args.num_train_epochs

# Inicialização do Otimizador e do Agendador de Taxa de Aprendizagem
trainer.create_optimizer_and_scheduler(num_training_steps=total_steps)

# Criação do Agendador de Taxa de Aprendizagem
scheduler = get_linear_schedule_with_warmup(
    optimizer=trainer.optimizer,
    num_warmup_steps=0,
    num_training_steps=total_steps
)

trainer.lr_scheduler = scheduler

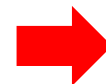
# Treinamento do Modelo
trainer.train()
```

Pré-processamento
e configurações dos
hiperparâmetros



Inicia, de fato, o pré-processamento dos dados e configura os argumentos do modelo para treinamento. A quantidade de épocas como 30 deu-se a partir de testes, que apontaram ser uma quantidade ideal

Inicialização dos
componentes e
treinamento



Inicializa o treinador, faz o cálculo dos passos de treinamento, inicializa/cria o otimizador e o agendador da taxa de aprendizado, e chama a função para treinar o modelo

Explicação do código

```
# Salvamento do Modelo Treinado
trainer.save_model("./legislation-model")
tokenizer.save_pretrained("./legislation-tokenizer")

# Coleta dos Valores de Loss
train_loss = []
eval_loss = []

for log in trainer.state.log_history:
    if 'loss' in log:
        train_loss.append(log['loss'])
    if 'eval_loss' in log:
        eval_loss.append(log['eval_loss'])

# Garantia de que os Arrays tenham o Mesmo Comprimento
min_len = min(len(train_loss), len(eval_loss))
train_loss = train_loss[:min_len]
eval_loss = eval_loss[:min_len]

# Plotagem do Gráfico de Desempenho do Loss por Época
epochs = range(1, min_len + 1)

plt.figure(figsize=(10, 5))
plt.plot(epochs, train_loss, label='Training Loss')
plt.plot(epochs, eval_loss, label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.title('Training and Validation Loss per Epoch')
plt.show()

# Cálculo da Perplexidade
def calculate_perplexity(loss):
    return np.exp(min(loss, 100)) # Limita a perda máxima a 100 para evitar explosão

train_perplexity = [calculate_perplexity(loss) for loss in train_loss]
eval_perplexity = [calculate_perplexity(loss) for loss in eval_loss]

# Plotagem do Gráfico de Desempenho da Perplexidade por Época
plt.figure(figsize=(10, 5))
plt.plot(epochs, train_perplexity, label='Training Perplexity')
plt.plot(epochs, eval_perplexity, label='Validation Perplexity')
plt.xlabel('Epochs')
plt.ylabel('Perplexity')
plt.legend()
plt.title('Training and Validation Perplexity per Epoch')
plt.show()
```

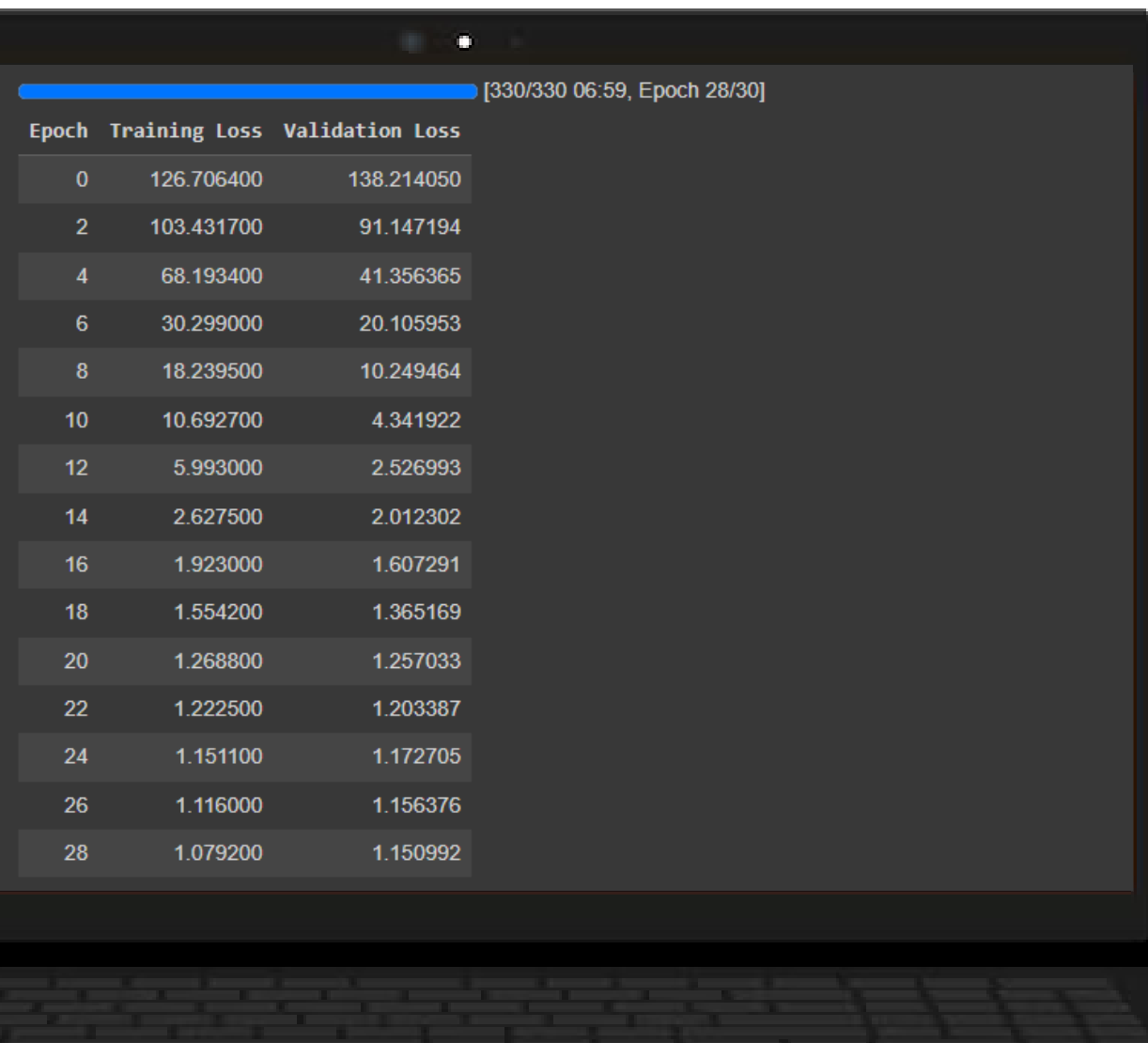
Salvamento do modelo, coleta de loss e garantia de comprimento

Salva o modelo e o tokenizador, coleta os valores de *loss* do treino (para ajuste de pesos) e de validação. Além disso, garante que os *arrays* de teste e validação tenham o mesmo comprimento

Plotagem de gráfico de *loss* e cálculo + plotagem da perplexidade

Plota e exibe o gráfico de *loss* de treino e validação a cada época (30, no total). Além disso, faz o cálculo da perplexidade do modelo e exibe em um gráfico ao longo das épocas

Explicação do código



[330/330 06:59, Epoch 28/30]

Epoch	Training Loss	Validation Loss
0	126.706400	138.214050
2	103.431700	91.147194
4	68.193400	41.356365
6	30.299000	20.105953
8	18.239500	10.249464
10	10.692700	4.341922
12	5.993000	2.526993
14	2.627500	2.012302
16	1.923000	1.607291
18	1.554200	1.365169
20	1.268800	1.257033
22	1.222500	1.203387
24	1.151100	1.172705
26	1.116000	1.156376
28	1.079200	1.150992

..... Resultado



Ao executar a célula, o treinamento do modelo é realizado, a partir das perguntas e respostas do JSON sobre a legislação acadêmica da UFAM, utilizando o PTT5. Observa-se que o *loss* é bem expressivo no início, mas vai diminuindo e ficando adequado ao longo das épocas. O gráfico de *loss* e perplexidade também é exibido, que podem ser consultado nos Slides 7 e 8, respectivamente

Explicação do código

```
# Verificação de disponibilidade da GPU
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Carregamento da métrica BLEU
bleu_metric = load_metric("sacrebleu")

# Função para calcular a métrica BLEU
def compute_bleu(predictions, references):
    decoded_preds = tokenizer.batch_decode(predictions, skip_special_tokens=True)
    decoded_labels = tokenizer.batch_decode(references, skip_special_tokens=True)
    decoded_labels = [[label] for label in decoded_labels]
    result = bleu_metric.compute(predictions=decoded_preds, references=decoded_labels)
    return result["score"]

# Avaliação do modelo
model.eval()
predictions = []
references = []

batch_size = 4 # Tamanho do lote para processamento (reduzido para evitar problemas de memória)

for i in range(0, len(tokenized_datasets), batch_size):
    batch = tokenized_datasets.select(range(i, min(i + batch_size, len(tokenized_datasets))))
    input_ids = torch.tensor(batch["input_ids"]).to(device)
    labels = torch.tensor(batch["labels"]).to(device)
    attention_mask = torch.tensor(batch["attention_mask"]).to(device)

    with torch.no_grad():
        outputs = model.generate(input_ids, attention_mask=attention_mask, max_length=512, num_beams=5, early_stopping=True)

    predictions.extend(outputs)
    references.extend(labels)

# Impressão das previsões e referências para verificação
decoded_preds = tokenizer.batch_decode(predictions, skip_special_tokens=True)
decoded_labels = tokenizer.batch_decode(references, skip_special_tokens=True)

for i in range(5):
    print(f"Prediction: {decoded_preds[i]}")
    print(f"Reference: {decoded_labels[i]}")

# Cálculo da pontuação BLEU
bleu_score = compute_bleu(predictions, references)
print("BLEU Score:", bleu_score)
```

..... BLEU →

Apesar da métrica BLEU (Bilingual Evaluation Understudy) ser utilizada majoritariamente para tradução, também pode ser usada para analisar a geração de texto no formato de perguntas e respostas (como neste caso). A BLEU realiza a correspondência de n-gramas entre as previsões geradas pelo modelo e as referências humanas, exibindo uma pontuação ao final. No código ao lado, as previsões e referências resultantes são impressas para analisar o desempenho do modelo, bem como a pontuação do BLEU

Explicação do código

```
Prediction:  
Reference: A sede da Universidade do Amazonas é na cidade de Manaus.  
Prediction:  
Reference: A Universidade do Amazonas foi criada em 12 de junho de 1962.  
Prediction:  
Reference: O Decreto no 53.699, de 13 de março de 1964, manteve a Universi  
Prediction:  
Reference: A Universidade do Amazonas goza de autonomia didático-científic  
Prediction:  
Reference: A organização e o funcionamento da Universidade do Amazonas são  
BLEU Score: 0.0
```

..... Resultado →

Apesar do bom treinamento do modelo, os resultados ao aplicar o BLEU foram insatisfatórios: um score de 0.0 e respostas vazias. Ao analisar o código, realizando depurações, notou-se que, por algum motivos, os tensores resultantes eram vazios, ocasionando em respostas nulas.

Explicação do código

```
# Carregamento do modelo e do tokenizer treinados
model_dir = "./legislation-model" # Diretório onde o modelo treinado foi salvo
tokenizer_dir = "./legislation-tokenizer" # Diretório onde o tokenizer treinado foi salvo

tokenizer = AutoTokenizer.from_pretrained(tokenizer_dir)
model = AutoModelForSeq2SeqLM.from_pretrained(model_dir)

# Carregamento do arquivo JSON
with open(file_name_json, 'r', encoding='utf-8') as file:
    qa_data = json.load(file)

# Função para buscar a resposta no JSON
def search_in_json(query):
    for item in qa_data:
        if query.lower() == item["pergunta"].lower():
            return item["resposta"]
    return None

# Função para extrair palavras-chave de uma consulta
def extract_keywords(query):
    keywords = query.lower().split()
    return " OR ".join(keywords)

# Função para recuperar textos relevantes usando Whoosh com verificação de relevância
def retrieve_relevant_texts(query, index_name, top_k=10):
    ix = open_dir(index_name)
    qp = QueryParser("content", schema=ix.schema)

    # Extração de palavras-chave da consulta
    keywords = extract_keywords(query)
    q = qp.parse(keywords)

    with ix.searcher() as searcher:
        results = searcher.search(q, limit=top_k)
        retrieved_texts = [result['content'] for result in results]
        print(f"Query: {query}")
        print(f"Retrieved {len(retrieved_texts)} relevant texts")
        for text in retrieved_texts:
            print(f" - {text[:100]}...") # Impressão de uma prévia de cada texto recuperado
    return retrieved_texts
```

Carregamento

Carrega o modelo, o tokenizador e o arquivo JSON com as perguntas e respostas

Atividades do modelo

Nesse trecho, o modelo realiza as seguintes atividades: (i) procura a resposta exata para uma pergunta, se ela for idêntica à que está contida na base sintética (visando focar somente em questões fora da base pra analisar o comportamento do modelo); (ii) extrai as palavras-chave das perguntas, visando consultar as ocorrências no *corpus*; e (iii) a partir das palavras-chave, consulta o *corpus* e retorna trechos relevantes, que estão associados ao contexto da pergunta realizada

Explicação do código

```
# Função de geração de respostas com textos contextuais
def generate_response_with_retrieval(model, tokenizer, query, index_name):
    # Busca inicial da resposta no JSON
    json_response = search_in_json(query)
    if json_response:
        return json_response

    # Busca no corpus indexado se não encontrado no JSON
    relevant_texts = retrieve_relevant_texts(query, index_name)
    if not relevant_texts:
        return "Desculpe, não encontrei uma resposta para essa pergunta. Tente reformulá-la."

    # Limitação do contexto para evitar sobrecarregar o modelo
    context = "\n\n".join(relevant_texts[:3])

    # Verificação da relevância do contexto
    if len(context.strip()) == 0:
        return "Desculpe, não encontrei uma resposta para essa pergunta. Tente reformulá-la."

    # Utilização do modelo treinado para gerar a resposta com base no contexto
    input_text = f"Contexto: {context}\n\nPergunta: {query}\n\nResposta:"
    inputs = tokenizer.encode(input_text, return_tensors="pt", max_length=1024, truncation=True)
    outputs = model.generate(inputs, max_length=150, num_beams=5, early_stopping=True)
    response = tokenizer.decode(outputs[0], skip_special_tokens=True)

    # Verificação da resposta gerada
    if response.strip() == "":
        response = "Desculpe, não encontrei uma resposta para essa pergunta. Tente reformulá-la."

    return response

# Função para testar as instruções
def test_model_with_retrieval(test_instructions):
    for instruction in test_instructions:
        response = generate_response_with_retrieval(model, tokenizer, instruction, "indexdir")
        print(f"Pergunta: {instruction}")
        print(f"Resposta: {response}\n")

# Teste do modelo com exemplos de perguntas
test_instructions = [
    "Qual é a sede da Universidade do Amazonas?",
    "Quando foi criada a Universidade do Amazonas?",
    "Quais são os princípios que a Universidade deve obedecer?",
    "O que rege a organização e o funcionamento da Universidade?",
    "Qual é a finalidade da Universidade?"
]

# Execução da função de teste
test_model_with_retrieval(test_instructions)

# Função para perguntar ao usuário
def ask_user_question():
    while True:
        user_question = input("Digite sua pergunta (ou 'sair' para encerrar): ")
        if user_question.lower() == 'sair':
            break

        response = generate_response_with_retrieval(model, tokenizer, user_question, "indexdir")
        print(f"Resposta: {response}\n")

# Permissão para que o usuário informe uma pergunta via prompt
ask_user_question()
```

Resposta a
partir do
contexto



Após o retorno de conteúdos relevantes dentro do *corpus* que estão associados à pergunta realizada, o modelo busca uma resposta a partir desse contexto. Caso não encontre algo relacionado, uma mensagem é impressa

Perguntas e
Respostas



Nessa parte do código, algumas questões de teste são incluídas para analisar o comportamento do modelo, além de uma opção para o usuário fazer suas próprias perguntas (o sistema é finalizado quando “sair” for digitado)

Explicação do código

```
Query: O que rege a organização e o funcionamento da Universidade?
Retrieved 10 relevant texts
- Art. 68 - A Universidade adotará regime financeiro e contábil que atenda às suas peculiaridades de o...
- Art. 3º - A organização e o funcionamento da Universidade reger-se-ão pelas normas constantes dos se...
- Art. 53 - A organização e o funcionamento da extensão obedecerão aos dispositivos estatutários e reg...
- propor ao Conselho Universitário, através da Reitoria, medidas que visem à melhoria do seu funcionam...
- e) opinar sobre normas complementares, a serem baixadas pelo Conselho de Ensino, Pesquisa e Extens...
- #### CAPÍTULO III - Da Organização Curricular...
- #### CAPÍTULO III - Da Organização Curricular...
- Art. 51 - A organização e o funcionamento da pesquisa, na Universidade, obedecerão às normas estatut...
- O presente Regimento Geral disciplina os aspectos de organização e funcionamento comuns aos vários ó...
- IX. deliberar sobre suspensão temporária, parcial ou total do funcionamento da Universidade;...
Pergunta: O que rege a organização e o funcionamento da Universidade?
Resposta: maduro maduro Undertaker maduro maduro Undertaker maduro maduro maduro maduro maduro maduro

Query: Qual é a finalidade da Universidade?
Retrieved 10 relevant texts
- DA FINALIDADE...
- III - nome da instituição de ensino na qual o estudante esteja matriculado;...
- Art. 3º Os diplomas expedidos pelas universidades serão por elas próprios registrados, e aqueles con...
- VI - identificação do sítio eletrônico da IES no qual poderá ser consultada a relação de diplomas re...
- Art. 50 - A Universidade empreenderá esforços no sentido de interiorizar as atividades de pesquisa, ...
- VIII - outras atividades de natureza semelhante e relacionadas à comunidade escolar na qual se inser...
- IV - acompanhante: aquele que acompanha a pessoa com deficiência, o qual pode ou não desempenhar as ...
- Art. 4º - A Universidade tem por finalidade cultivar o saber em todos os campos do conhecimento puro...
```

..... Resultado



Os resultados da utilização de RAG indicam que as consultas no *corpus* estão sendo realizadas adequadamente, retornando contextos relevantes a partir das palavras-chave das perguntas. Além disso, o programa retorna as respostas adequadas quando a pergunta está presente no JSON. Entretanto, quando o modelo responde às perguntas fora do JSON, traz informações incoerentes e não relacionadas ao contexto anteriormente apresentado.

Análise dos resultados

- Em relação ao processamento dos documentos, que eventualmente tornaram-se um único arquivo TXT, e à geração da base sintética com perguntas e respostas sobre a legislação acadêmica da UFAM, ocorreu adequadamente. Apesar da base não conter 1000 perguntas, continha 10% disso com questões distintas e relevantes
- De forma geral, quanto ao treinamento, o modelo apresentou bons resultados: um baixo *loss* depois de 30 épocas, com, respectivamente, ≈ 1.07 de treino e ≈ 1.15 de validação. A perplexidade também teve uma queda, entretanto, convergiu para 0 – um valor atípico, indicando um possível superajuste do modelo aos dados



Análise dos resultados

- Para avaliar o desempenho do modelo, fez-se uso do BLEU, contudo, atipicamente os tensores retornados eram nulos. Já no que se refere à implementação de RAG, a indexação no mecanismo de busca Whoosh mostrou-se funcional. A consulta via palavras-chave ao *corpus* também estava sendo realizada corretamente
- No entanto, apesar do bom treinamento e funcionamento de RAG, o modelo estranhamente não conseguiu responder corretamente às perguntas. Não somente isso, mas trouxe informações incoerentes e destoantes à legislação acadêmica da UFAM. Por exemplo, em uma das respostas, repetiu o termo “maduro” várias vezes. Isso pode estar relacionado à situação política atual da Venezuela, indicando que o modelo acessou à alguma fonte externa ao *corpus* – que não foi identificado no código. Portanto, tanto o BLEU quanto o RAG demonstraram uma dificuldade do modelo em retornar respostas adequadas



Análise dos resultados

- Desse modo, apesar das consultas relevantes ao *corpus* que são realizadas a partir das palavras-chave das perguntas, o modelo não conseguiu utilizar essas informações para retornar uma boa resposta. O código foi revisado, mas não identificou-se como nem onde essa inconsistência ocorre
- Para reverter essa situação, alternativas possíveis seriam: utilizar outro modelo (não o PTT5) para treinamento, inserir mais componentes na implementação do LLM, adicionar mais verificações para acompanhar as atividades do modelo, dentre outros



Links importantes

- **Repositório no GitHub:**
<https://github.com/fabhonda/ufam-ppginf528-nlp-tp3>
- **Notebook no Google Colab:**
<https://colab.research.google.com/drive/1F1p-uryzjoZziLzxnKIMUhbntep5qsyZ?usp=sharing>
- **Modelo no Drive (.zip, se necessário):**
https://drive.google.com/file/d/1PJoXVOB_A_bXem5vDDzvTOATIpT9Z9NrR/view?usp=sharing

