Folha de Dicas: Ajuste Fino Avançado de IA Generativa para LLMs

Pacote/Método	Descrição	Exemplo de Código
GPU compatível com CUDA	Disponível no sistema usando PyTorch, um popular framework de aprendizado profundo. Se uma GPU estiver disponível, ela atribui a variável device a "cuda" (CUDA, a plataforma de computação paralela e modelo de interface de programação de aplicativos desenvolvido pela NVIDIA). Se uma GPU não estiver disponível, ela atribui a variável device a "cpu" (o que significa que o código será executado na CPU).	`device = torch.device("cuda" if torch.cuda.is_available() else "cpu")` `device`

collate_fn	Mostra que a função collate_fn é usada em conjunto com carregadores de dados para personalizar a maneira como os lotes são criados a partir de amostras individuais. Uma função collate_batch no PyTorch é usada com carregadores de dados para personalizar a criação de lotes a partir de amostras individuais. Ela processa um lote de dados, incluindo rótulos e sequências de texto. Aplica a função text_pipeline para pré-processar o texto. Os dados processados são então convertidos em tensores do PyTorch e retornados como uma tupla contendo o tensor de rótulo, tensor de texto e tensor de offsets representando as posições iniciais de cada sequência de texto no tensor combinado. A função também garante que os tensores retornados sejam movidos para o dispositivo especificado (GPU) para computação eficiente.	`from torch.nn.utils.rnn import pad_sequence`
		`def collate_batch(batch):`
		`  label_list, text_list = [], []`
		`  for _label, _text in batch:`
		`  label_list.append(_label)`
		`  text_list.append(torch.tensor(text_pipeline(_text), dtype=torch.int64)`
		`  label_list = torch.tensor(label_list, dtype=torch.int64)`
		`  text_list = pad_sequence(text_list, batch_first=True)`
		`  return label_list.to(device), text_list.to(device)`

Função de treinamento	Auxilia no treinamento do modelo, atualizando iterativamente os parâmetros do modelo para minimizar a função de perda. Melhora o desempenho do modelo em uma tarefa específica.	`def train_model(model, optimizer, criterion, train_dataloader, valid_dataloade`
		`  cum_loss_list = []`
		`  acc_epoch = []`
		`  acc_old = 0`
		`  model_path = os.path.join(save_dir, file_name)`
		`  acc_dir = os.path.join(save_dir, os.path.splitext(file_name)[0] + "_acc")`
		`  loss_dir = os.path.join(save_dir, os.path.splitext(file_name)[0] + "_loss"`
		`  time_start = time.time()`
		`  for epoch in tqdm(range(1, epochs + 1)):`
		`  model.train()`
		`  cum_loss = 0`
		`  for idx, (label, text) in enumerate(train_dataloader):`
		`  optimizer.zero_grad()`
		`  label, text = label.to(device), text.to(device)`
		`  predicted_label = model(text)`
		`  loss = criterion(predicted_label, label)`
		`  loss.backward()`
		`  optimizer.step()`
		`  cum_loss += loss.item()`
		`  print(f"Epoch {epoch}/{epochs} - Loss: {cum_loss}")`
		`  cum_loss_list.append(cum_loss)`
		`  accu_val = evaluate_no_tqdm(valid_dataloader,model)`
		`  acc_epoch.append(accu_val)`
		`  if model_path and accu_val > acc_old:`
		`  acc_old = accu_val`
		`  if save_dir is not None:`
		`  pass`
		`  time_end = time.time()`
		`  print(f"Tempo de treinamento: {time_end - time_start}")`

Pacote/Método	Descrição	Exemplo de Código

Configuração DPO	Envolve argumentos de treinamento, processamento de etapas de registro, estratégias de avaliação e agendamento para atualizações do modelo. Configurar DPO envolve estabelecer medidas administrativas e técnicas para cumprir com as leis e regulamentos de proteção de dados.	`# Configuração DPO`
		`training_args = DPOConfig(`
		`beta=0.1,`
		`output_dir="dpo",`
		`num_train_epochs=5,`
		`per_device_train_batch_size=1,`
		`per_device_eval_batch_size=1,`
		`remove_unused_columns=False,`
		`logging_steps=10,`
		`gradient_accumulation_steps=1,`
		`learning_rate=1e-4,`
		`eval_strategy="epoch",`
		`warmup_steps=2,`
		`fp16=False,`
		`save_steps=500,`
		`report_to='none'`

Classe DPOTraining	Projetada para equipar profissionais com o conhecimento e habilidades para gerenciar e supervisionar estratégias de proteção de dados em conformidade com as leis e regulamentos relevantes.	`tokenizer.pad_token = tokenizer.eos_token`
		`# Criar um treinador de DPO`
		`# Este treinador irá lidar com o ajuste f`
		`trainer = DPOTrainer(`
		`# O modelo a ser ajustado`
		`model,`
		`# O modelo de referência (não utilizado neste caso porque LoRA foi usa`
		`ref_model=None,`
		`# A configuração de treinamento DPO`
		`args=training_args,`
		`# O parâmetro beta para a função de perda DPO`
		`beta=0.1,`
		`# O conjunto de dados de treinamento`
		`train_dataset=train_dataset,`
		`# O conjunto de dados de avaliação`
		`eval_dataset=eval_dataset,`
		`# O tokenizer para o modelo`
		`tokenizer=tokenizer,`
		`# A configuração PEFT (Parallel Efficient Finetuning)`
		`peft_config=peft_config,`
		`# O comprimento máximo do prompt`
		`max_prompt_length=512,`
		`# O comprimento máximo da sequência`
		`max_length=512,`
		`)`

Recuperar e plotar a perda de treinamento versus perda de avaliação	Ajuda a recuperar o histórico de logs e salvá-lo para o log do dataframe. Também plota as perdas de treinamento e avaliação para a época.	`# Recuperar log_history e salvá-lo em um dataframe`
		`log = pd.DataFrame(trainer.state.log_history)`
		`log_t = log[log['loss'].notna()]`
		`log_e = log[log['eval_loss'].notna()]`
		`# Plotar perdas de treinamento e avaliaçã`
		`plt.plot(log_t[“epoch”], log_t[“loss”], label = “train_loss”)`
		`plt.plot(log_e[“epoch”], log_e[“eval_loss”], label = “eval_loss”)`
		`plt.legend()`
		`plt.show()`

Pacote/Método	Descrição	Exemplo de Código

Percorrer o conjunto de dados IMDB	Este trecho de código percorre o conjunto de dados IMDB obtendo, carregando e explorando o conjunto de dados. Ele também realiza operações básicas, visualiza os dados e analisa e interpreta o conjunto de dados.	`class IMDBDataset(Dataset):`
		`  def __init__(self, root_dir, train=True):`
		`  """`
		`  root_dir: O diretório base do conjunto de dados IMDB.`
		`  train: Um sinalizador booleano indicando se deve usar dados de treinam`
		`  """`
		`  self.root_dir = os.path.join(root_dir, "train" if train else "test")`
		`  self.neg_files = [os.path.join(self.root_dir, "neg", f) for f in os.li`
		`  self.pos_files = [os.path.join(self.root_dir, "pos", f) for f in os.li`
		`  self.files = self.neg_files + self.pos_files`
		`  self.labels = [0] * len(self.neg_files) + [1] * len(self.pos_files)`
		`  self.pos_inx=len(self.pos_files)`
		`  def __len__(self):`
		`  return len(self.files)`
		`  def __getitem__(self, idx):`
		`  file_path = self.files[idx]`
		`  label = self.labels[idx]`
		`  with open(file_path, 'r', encoding='utf-8') as file:`
		`  content = file.read()`
		`  return label, content`

Iteradores para conjuntos de dados de treinamento e teste	Este trecho de código indica um caminho para o diretório do conjunto de dados IMDB, combinando nomes temporários e de subdiretórios. Este código configura os iteradores de dados de treinamento e teste, recupera o índice inicial dos dados de treinamento e imprime os itens do conjunto de dados de treinamento nos índices.	`root_dir = tempdir.name + '/' + 'imdb_dataset'`
		`train_iter = IMDBDataset(root_dir=root_dir, train=True) # Para`
		`dados de treinamento`
		`test_iter = IMDBDataset(root_dir=root_dir, train=False) # Para dados de teste`
		`start=train_iter.pos_inx`
		`for i in range(-10,10):`
		`  print(train_iter[start+i])`

Função yield_tokens	Gera tokens a partir da coleção de amostras de dados de texto. O trecho de código processa cada texto em 'data_iter' através do tokenizador e gera tokens para criar uma geração de tokens eficiente, adequada para tarefas como treinamento de modelos de aprendizado de máquina.	`tokenizer = get_tokenizer("basic_english")`
		`def yield_tokens(data_iter):`
		`  """Gera tokens para cada amostra de dados."""`
		`  for _, text in data_iter:`
		`  yield tokenizer(text)`

Carregar modelo pré-treinado e sua avaliação em dados de teste	Este trecho de código ajuda a baixar um modelo pré-treinado de uma URL, carregá-lo em uma arquitetura específica e avaliá-lo em um conjunto de dados de teste para avaliar seu desempenho.	`urlopened = urlopen('https://cf-courses-data.s3.us.cloud-object-storage.appdom`
		`model_ = Net(vocab_size=vocab_size, num_class=2).to(device)`
		`model_.load_state_dict(torch.load(io.BytesIO(urlopened.read()), map_location=d`
		`evaluate(test_dataloader, model_)`

Carregando o modelo Hugging Face	Este trecho de código inicia um tokenizador usando um modelo pré-treinado 'bert-base-cased'. Ele também baixa um modelo pré-treinado para a tarefa de modelo de linguagem mascarada (MLM) e como carregar as configurações do modelo a partir de um modelo pré-treinado.	`# Instanciar um tokenizador usando o modelo BERT base cased`
		`tokenizer = AutoTokenizer.from_pretrained("bert-base-cased")`
		Baixe o modelo pré-treinado de huggingf
		`model = BertForMaskedLM.from_pretrained(‘bert-base-cased’)`
		Você também pode começar o treinamento
		config =
		`AutoConfig.from_pretrained(“google-bert/bert-base-cased”)`

Pacote/Método	Descrição	Exemplo de Código

		# model = BertForMaskedLM.from_config(config)</pre></td>
		</tr>
		<tr class="even">
		<td>Treinando um modelo BERT para a tarefa MLM</td>
		<td>Este trecho de código treina o modelo com os parâmetros e conjunto de dado
		<td><pre>training_args = TrainingArguments(
		output_dir="./trained_model", # Especifique o diretório de saída para o m
		overwrite_output_dir=True,
		do_eval=False,
		learning_rate=5e-5,
		num_train_epochs=1, # Especifique o número de épocas de treinamento
		per_device_train_batch_size=2, # Defina o tamanho do lote para o treiname
		save_total_limit=2, # Limite o número total de pontos de verificação salv
		logging_steps = 20
		)
		dataset = load_dataset("imdb", split="train")
		trainer = SFTTrainer(
		model,
		args=training_args,
		train_dataset=dataset,
		dataset_text_field="text",
		)</pre></td>
		</tr>
		<tr class="odd">
		<td>Carregar o modelo e o tokenizador</td>
		<td>Útil para tarefas onde você precisa classificar rapidamente o sentimento d
		<td><pre>tokenizer = DistilBertTokenizer.from_pretrained("distilbert-base-unca
		model = DistilBertForSequenceClassification.from_pretrained("distilbert-base-u
		</pre></td>
		</tr>
		<tr class="even">
		<td>torch.no_grad()</td>
		<td>O gerenciador de contexto torch.no_grad() desabilita o cálculo de gradient
		<td><pre># Realizar inferência
		with torch.no_grad():
		outputs = model(**inputs)</pre></td>
		</tr>
		<tr class="odd">
		<td>Logits</td>
		<td>As previsões brutas e não normalizadas do modelo. Vamos extrair os logits
		<td><pre>logits = outputs.logits
		logits.shape</pre></td>
		</tr>
		<tr class="even">
		<td>Tokenizador GPT-2</td>
		<td>Ajuda a inicializar o tokenizador GPT-2 usando um modelo pré-treinado para
		<td><pre># Carregar o tokenizador e o modelo
		tokenizer = GPT2Tokenizer.from_pretrained("gpt2")</pre></td>
		</tr>
		<tr class="odd">
		<td>Carregar modelo GPT-2</td>
		<td>Este trecho de código inicializa e carrega o modelo GPT-2 pré-treinado. Es
		<td><pre># Carregar o tokenizador e o modelo
		model = GPT2LMHeadModel.from_pretrained("gpt2")</pre></td>
		</tr>

Pacote/Método	Descrição	Exemplo de Código

		<tr class="even">
		<td>Gerar texto</td>
		<td>Este trecho de código gera sequências de texto com base na entrada e não c
		<td><pre># Gerar texto
		output_ids = model.generate(
		inputs.input_ids,
		attention_mask=inputs.attention_mask,
		pad_token_id=tokenizer.eos_token_id,
		max_length=50,
		num_return_sequences=1
		)
		output_ids
		ou
		with torch.no_grad():
		outputs = model(**inputs)
		outputs
		</pre></td>
		</tr>
		<tr class="odd">
		<td>Decodificar o texto gerado</td>
		<td>Este trecho de código decodifica o texto a partir dos IDs de token gerados
		<td><pre># Decodificar o texto gerado
		generated_text = tokenizer.decode(output_ids[0], skip_special_tokens=True)
		print(generated_text)
		</pre></td>
		</tr>
		<tr class="even">
		<td>Função pipeline() do Hugging Face</td>
		<td>A função pipeline() da biblioteca Transformers do Hugging Face é uma API d
		<td><pre>transformers.pipeline(
		task: str,
		model: Optional = None,
		config: Optional = None,
		tokenizer: Optional = None,
		feature_extractor: Optional = None,
		framework: Optional = None,
		revision: str = 'main',
		use_fast: bool = True,
		model_kwargs: Dict[str, Any] = None,
		**kwargs
		)</pre></td>
		</tr>
		<tr class="odd">
		<td>expected_outputs</td>
		<td>Tokenize instruções e as instructions_with_responses. Em seguida, conte o
		<td><pre>expected_outputs = []
		instructions_with_responses = formatting_prompts_func(test_dataset)
		instructions = formatting_prompts_func_no_response(test_dataset)
		for i in tqdm(range(len(instructions_with_responses))):
		tokenized_instruction_with_response = tokenizer(instructions_with_response
		tokenized_instruction = tokenizer(instructions[i], return_tensors="pt")
		expected_output = tokenizer.decode(tokenized_instruction_with_response['in
		expected_outputs.append(expected_output)
		</pre></td>
		</tr>

Pacote/Método	Descrição	Exemplo de Código

		<tr class="odd">
		<td>ListDataset</td>
		<td>Herda de Dataset e cria um Dataset do torch a partir de uma lista. Esta cl
		<td><pre>class ListDataset(Dataset):
		    def __init__(self, original_list):
		        self.original_list = original_list
		    def __len__(self):
		        return len(self.original_list)
		    def __getitem__(self, i):
		        return self.original_list[i]
		instructions_torch = ListDataset(instructions)
		</pre></td>
		</tr>
		<tr class="even">
		<td>gen_pipeline</td>
		<td>Este trecho de código pega os IDs de token da saída do modelo, decodifica
		<td><pre>gen_pipeline = pipeline("text-generation",
		                        model=model,
		                        tokenizer=tokenizer,
		                        device=device,
		                        batch_size=2,
		                        max_length=50,
		                        truncation=True,
		                        padding=False,
		                        return_full_text=False)
		</pre></td>
		</tr>
		<tr class="odd">
		<td>torch.no_grad()</td>
		<td>Este código gera texto a partir da entrada dada usando um pipeline enquant
		<td><pre>with torch.no_grad():
		    # Devido à limitação de recursos, aplique a função apenas em 3 registros u
		    pipeline_iterator= gen_pipeline(instructions_torch[:3],
		                                    max_length=50, # isso é definido como 50 d
		                                    num_beams=5,
		                                    early_stopping=True,)
		generated_outputs_base = []
		for text in pipeline_iterator:
		    generated_outputs_base.append(text[0]["generated_text"])
		</pre></td>
		</tr>
		<tr class="even">
		<td>load_dataset</td>
		<td>O conjunto de dados é carregado usando a função load_dataset da biblioteca
		<td><pre>dataset_name = "imdb"
		ds = load_dataset(dataset_name, split = "train")
		N = 5
		for sample in range(N):
		    print('text',ds[sample]['text'])
		    print('label',ds[sample]['label'])
		ds = ds.rename_columns({"text": "review"})
		ds
		ds = ds.filter(lambda x: len(x["review"]) > 200, batched=False)
		</pre></td>
		</tr>

Pacote/Método	Descrição	Exemplo de Código

		<tr class="odd">
		<td>build_dataset</td>
		<td>Incorpora os passos necessários para construir um objeto de conjunto de da
		<td><pre>del(ds)
		dataset_name="imdb"
		ds = load_dataset(dataset_name, split="train")
		ds = ds.rename_columns({"text": "review"})
		def build_dataset(config, dataset_name="imdb", input_min_text_length=2, input_
		"""
		Construir conjunto de dados para treinamento. Isso constrói o conjunto de
		personalizar esta função para treinar o modelo em seu próprio conjunto de
		Args:
		dataset_name (`str`):
		O nome do conjunto de dados a ser carregado.
		Returns:
		dataloader (`torch.utils.data.DataLoader`):
		O dataloader para o conjunto de dados.
		"""
		tokenizer = AutoTokenizer.from_pretrained(config.model_name)
		tokenizer.pad_token = tokenizer.eos_token
		# carregar imdb com datasets
		ds = load_dataset(dataset_name, split="train")
		ds = ds.rename_columns({"text": "review"})
		ds = ds.filter(lambda x: len(x["review"]) > 200, batched=False)
		input_size = LengthSampler(input_min_text_length, input_max_text_length)
		def tokenize(sample):
		sample["input_ids"] = tokenizer.encode(sample["review"])[: input_size(
		sample["query"] = tokenizer.decode(sample["input_ids"])
		return sample
		ds = ds.map(tokenize, batched=False)
		ds.set_format(type="torch")
		return ds
		</pre></td>
		</tr>
		<tr class="even">
		<td>Inicializar PPOTrainer</td>
		<td>A otimização de política proximal (PPO) é um algoritmo de aprendizado por
		Melhora os métodos de gradiente de política para chatbots usando uma função ob
		config: Configurações de configuração para treinamento PPO, como taxa de apren
		model: O modelo principal a ser ajustado usando PPO.
		tokenizer: Tokenizador correspondente ao modelo, usado para processar o texto
		dataset: Conjunto de dados a ser usado para treinamento, fornecendo os dados d
		data_collator: Colador de dados para gerenciar o agrupamento e a formatação do
		<td><pre>ppo_trainer = PPOTrainer(config, model, ref_model, tokenizer, dataset
		print("ppo_trainer object ",ppo_trainer)
		device = ppo_trainer.accelerator.device
		if ppo_trainer.accelerator.num_processes == 1:
		device = 0 if torch.cuda.is_available() else "cpu"
		print(device)
		</pre></td>
		</tr>
		<tr class="odd">
		<td>output_length_sampler</td>
		<td>Inicializado com LengthSampler(output_min_length,
		output_max_length). Este objeto é usado para amostrar comprimentos de saída pa

Pacote/Método	Descrição	Exemplo de Código

		<td><pre>output_min_length = 4
		output_max_length = 16
		output_length_sampler = LengthSampler(output_min_length, output_max_length)</p
		</tr>
		<tr class="even">
		<td>max_new_tokens</td>
		<td>Defina o parâmetro max_new_tokens no dicionário generation_kwargs para o v
		<td><pre>generation_kwargs["max_new_tokens"] = gen_len
		generation_kwargs</pre></td>
		</tr>
		<tr class="odd">
		<td>Função de geração de texto</td>
		<td>Tokeniza o texto de entrada, gera uma resposta e a decodifica.</td>
		<td><pre>gen_kwargs = {"min_length": -1, "top_k": 0.0, "top_p": 1.0, "do_sampl
		def generate_some_text(input_text,my_model):
		# Tokenize o texto de entrada
		input_ids = tokenizer(input_text, return_tensors='pt').input_ids.to(device)
		generated_ids = my_model.generate(input_ids,**gen_kwargs)
		# Decodificar o texto gerado
		generated_text_ = tokenizer.decode(generated_ids[0], skip_special_tokens=True)
		return generated_text_

Gerar texto com modelo PPO	Gerar texto usando o modelo treinado com PPO.	input_text = "Era uma vez em uma terra distante" generated_text=generate_some_text(input_text,model_1) generated_text
Análise de sentimento	Analise o sentimento do texto gerado usando sentiment_pipe.	pipe_outputs = sentiment_pipe(generated_text, **sent_kwargs) pipe_outputs
Gerar texto com modelo de referência	Gerar texto usando o modelo de referência.	generated_text = generate_some_text(input_text,ref_model) generated_text
compare_models_on_dataset	Este trecho de código inicializa os parâmetros de	def compare_models_on_dataset(model, ref_model, dataset, tokenizer, sentiment_ gen_kwargs = {

Pacote/Método	Descrição	Exemplo de Código

	geração, prepara o conjunto de dados e converte as consultas em tensores para a entrada do modelo. Ele também gera um modelo, ajuda a decodificar o texto e usa análise de sentimento para calcular pontuações de sentimento para textos concentrados antes e depois de aplicar o modelo. Este código também compila resultados, armazena consultas originais e converte o dicionário em um dataframe pandas para fácil análise.	<pre>"min_length": -1,
		"top_k": 0.0,
		"top_p": 1.0,
		"do_sample": True,
		"pad_token_id": tokenizer.eos_token_id
		}
		bs = 16
		game_data = dict()
		dataset.set_format("pandas")
		df_batch = dataset[:].sample(bs)
		game_data["query"] = df_batch["query"].tolist()
		query_tensors = df_batch["input_ids"].tolist()
		response_tensors_ref, response_tensors = [], []
		for i in range(bs):
		gen_len = output_length_sampler()
		output = ref_model.generate(
		torch.tensor(query_tensors[i]).unsqueeze(dim=0).to(device),
		max_new_tokens=gen_len,
		**gen_kwargs
		).squeeze()[-gen_len:]
		response_tensors_ref.append(output)
		output = model.generate(torch.tensor(query_tensors[i]).unsqueeze(dim
		max_new_tokens=gen_len,
		**gen_kwargs
		).squeeze()[-gen_len:]
		response_tensors.append(output)
		game_data["response (before)"] = [tokenizer.decode(response_tensors_ref[i]
		game_data["response (after)"] = [tokenizer.decode(response_tensors[i]) for
		texts_before = [q + r for q, r in zip(game_data["query"], game_data["respo
		game_data["rewards (before)"] = [output[1]["score"] for output in sentimen
		texts_after = [q + r for q, r in zip(game_data["query"], game_data["respon
		game_data["rewards (after)"] = [output[1]["score"] for output in sentiment
		df_results = pd.DataFrame(game_data)
		return df_results</pre>

Tokenizando dados	Este trecho de código define uma função 'compare_models_on_dataset' para comparar o desempenho de dois modelos, inicializando parâmetros de geração e definindo o tamanho do lote, preparando o conjunto de dados no formato pandas e amostrando as consultas do lote.	<pre># Instanciar um tokenizer usando o modelo BERT base cased
		tokenizer = AutoTokenizer.from_pretrained("bert-base-cased")</pre>
		Defina uma função para tokenizar exempl
		<pre>def tokenize_function(examples):
		# Tokenize o texto usando o tokenizador
		# Aplique o preenchimento para garantir que todas as sequências tenham o m
		# Aplique a truncagem para limitar o comprimento máximo da sequência
		return tokenizer(examples[“text”], padding=”max_length”, truncation=True)</pre>
		Aplique a função tokenize ao conjunto d
		<pre>tokenized_datasets = dataset.map(tokenize_function, batched=True)</pre>

Loop de treinamento	A função train_model treina um modelo usando um conjunto de dados de treinamento fornecido através de um dataloader. Começa configurando uma barra de progresso para ajudar a monitorar visualmente o progresso do treinamento. O modelo é definido em modo de treinamento, o que é necessário para certos comportamentos do modelo, como dropout, funcionarem corretamente durante o treinamento. A função processa os dados em lotes para cada época, o que envolve várias etapas para cada lote: transferir os dados para o dispositivo correto (como uma GPU), executar	<pre>def train_model(model,tr_dataloader):
		# Criar uma barra de progresso para acompanhar o progresso do treinamento
		progress_bar = tqdm(range(num_training_steps))
		# Definir o modelo em modo de treinamento
		model.train()
		tr_losses=[]
		# Loop de treinamento
		for epoch in range(num_epochs):
		total_loss = 0
		# Iterar sobre os lotes de dados de treinamento
		for batch in tr_dataloader:
		# Mover o lote para o dispositivo apropriado
		batch = {k: v.to(device) for k, v in batch.items()}
		# Passagem para frente pelo modelo
		outputs = model(**batch)
		# Calcular a perda
		loss = outputs.loss
		# Passagem para trás (calcular gradientes)
		loss.backward()
		total_loss += loss.item()
		# Atualizar os parâmetros do modelo
		optimizer.step()
		# Atualizar o programador de taxa de aprendizado
		lr_scheduler.step()
		# Limpar os gradientes
		optimizer.zero_grad()</pre>

Pacote/Método	Descrição	Exemplo de Código

	os dados pelo modelo para obter saídas e calcular a perda, atualizar os parâmetros do modelo usando os gradientes calculados, ajustar a taxa de aprendizado e limpar os gradientes antigos.	`# Atualizar a barra de progresso`
		`progress_bar.update(1)`
		`tr_losses.append(total_loss/len(tr_dataloader))`
		`#plotar perda`
		`plt.plot(tr_losses)`
		`plt.title("Perda de treinamento")`
		`plt.xlabel("Época")`
		`plt.ylabel("Perda")`
		`plt.show()`

Função evaluate_model	Funciona de maneira semelhante à função train_model, mas é usada para avaliar o desempenho do modelo em vez de treiná-lo. Utiliza um dataloader para processar dados em lotes, configurando o modelo em modo de avaliação para garantir precisão nas medições e desativando cálculos de gradiente, uma vez que não está em treinamento. A função calcula previsões para cada lote, atualiza uma métrica de precisão e, finalmente, imprime a precisão geral após processar todos os lotes.	`def evaluate_model(model, evl_dataloader):`
		`# Criar uma instância da métrica de Precisão para classificação multiclass`
		`metric = Accuracy(task="multiclass", num_classes=5).to(device)`
		`# Definir o modelo em modo de avaliação`
		`model.eval()`
		`# Desativar o cálculo de gradientes durante a avaliação`
		`with torch.no_grad():`
		`# Iterar sobre os lotes de dados de avaliação`
		`for batch in evl_dataloader:`
		`# Mover o lote para o dispositivo apropriado`
		`batch = {k: v.to(device) for k, v in batch.items()}`
		`# Passagem para frente pelo modelo`
		`outputs = model(**batch)`
		`# Obter os rótulos de classe previstos`
		`logits = outputs.logits`
		`predictions = torch.argmax(logits, dim=-1)`
		`# Acumular as previsões e rótulos para a métrica`
		`metric(predictions, batch["labels"])`
		`# Calcular a precisão`
		`accuracy = metric.compute()`
		`# Imprimir a precisão`
		`print("Precisão:", accuracy.item())`

llm_model	Este trecho de código define a função 'llm_model' para gerar texto usando o modelo de linguagem da plataforma mistral.ai, especificamente o modelo 'mitral-8x7b-instruct-v01'. A função ajuda a personalizar os parâmetros de geração e interage com os serviços de aprendizado de máquina da IBM Watson.	`def llm_model(prompt_txt, params=None):`
		`model_id = 'mistralai/mixtral-8x7b-instruct-v01'`
		`default_params = {`
		`"max_new_tokens": 256,`
		`"min_new_tokens": 0,`
		`"temperature": 0.5,`
		`"top_p": 0.2,`
		`"top_k": 1`
		`}`
		`if params:`
		`default_params.update(params)`
		`parameters = {`
		`GenParams.MAX_NEW_TOKENS: default_params["max_new_tokens"],  # isso co`
		`GenParams.MIN_NEW_TOKENS: default_params["min_new_tokens"], # isso con`
		`GenParams.TEMPERATURE: default_params["temperature"], # isso controla`
		`GenParams.TOP_P: default_params["top_p"],`
		`GenParams.TOP_K: default_params["top_k"]`
		`}`
		`credentials = {`
		`"url": "https://us-south.ml.cloud.ibm.com"`
		`}`
		`project_id = "skills-network"`
		`model = Model(`
		`model_id=model_id,`
		`params=parameters,`
		`credentials=credentials,`
		`project_id=project_id`
		`)`
		`mixtral_llm = WatsonxLLM(model=model)`
		`response  = mixtral_llm.invoke(prompt_txt)`
		`return response`

RewardTrainer	O RewardTrainer orquestra o processo de treinamento, lidando com tarefas como	`# Inicializar RewardTrainer`
		`trainer = RewardTrainer(`
		`model=model,`
		`args=training_args,`

Pacote/Método	Descrição	Exemplo de Código

	agrupamento, otimização, avaliação e salvamento de pontos de verificação do modelo. É particularmente útil para treinar modelos que precisam aprender com sinais de feedback, melhorando sua capacidade de gerar respostas de alta qualidade.	`tokenizer=tokenizer,`
		`train_dataset=dataset_dict['train'],`
		`eval_dataset=dataset_dict['test'],`
		`peft_config=peft_config,`
		`)`