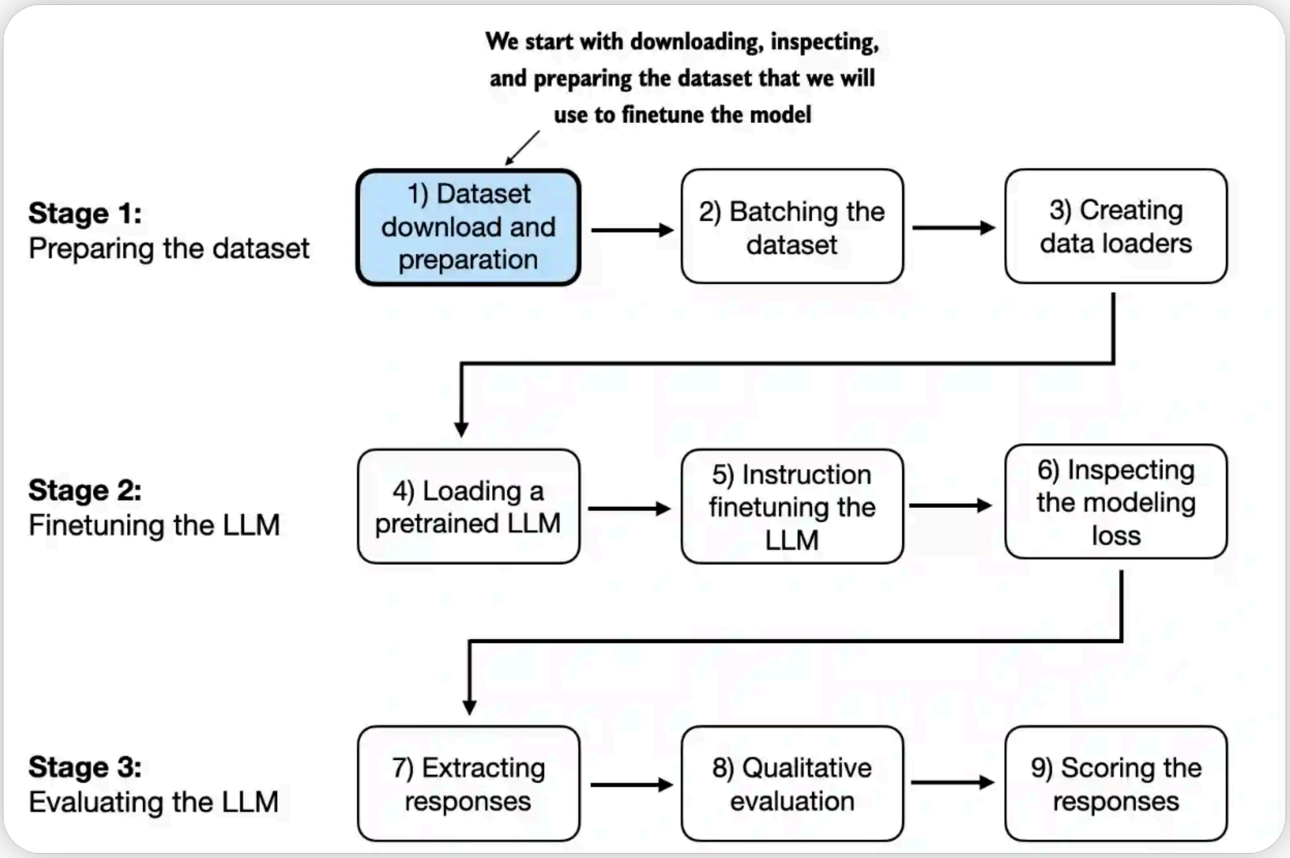
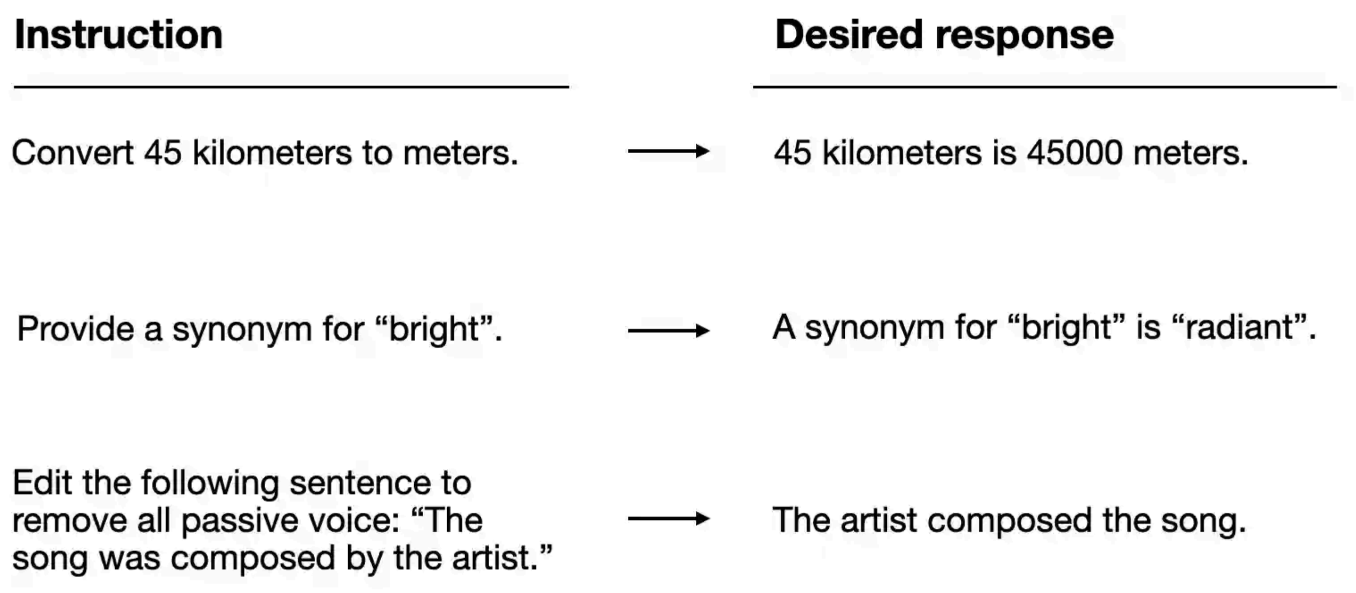
本文包括以下全部步骤来实现指令微调。


We start with downloading, inspecting, and preparing the dataset that we will use to finetune the model
Stage 1: Preparing the dataset
1) Dataset download and preparation
2) Batching the dataset
3) Creating data loaders
Stage 2: Finetuning the LLM
4) Loading a pretrained LLM
5) Instruction finetuning the LLM
6) Inspecting the modeling loss
Stage 3: Evaluating the LLM
7) Extracting responses
8) Qualitative evaluation
9) Scoring the responses

7 - 1 指令微调简介

示意图：



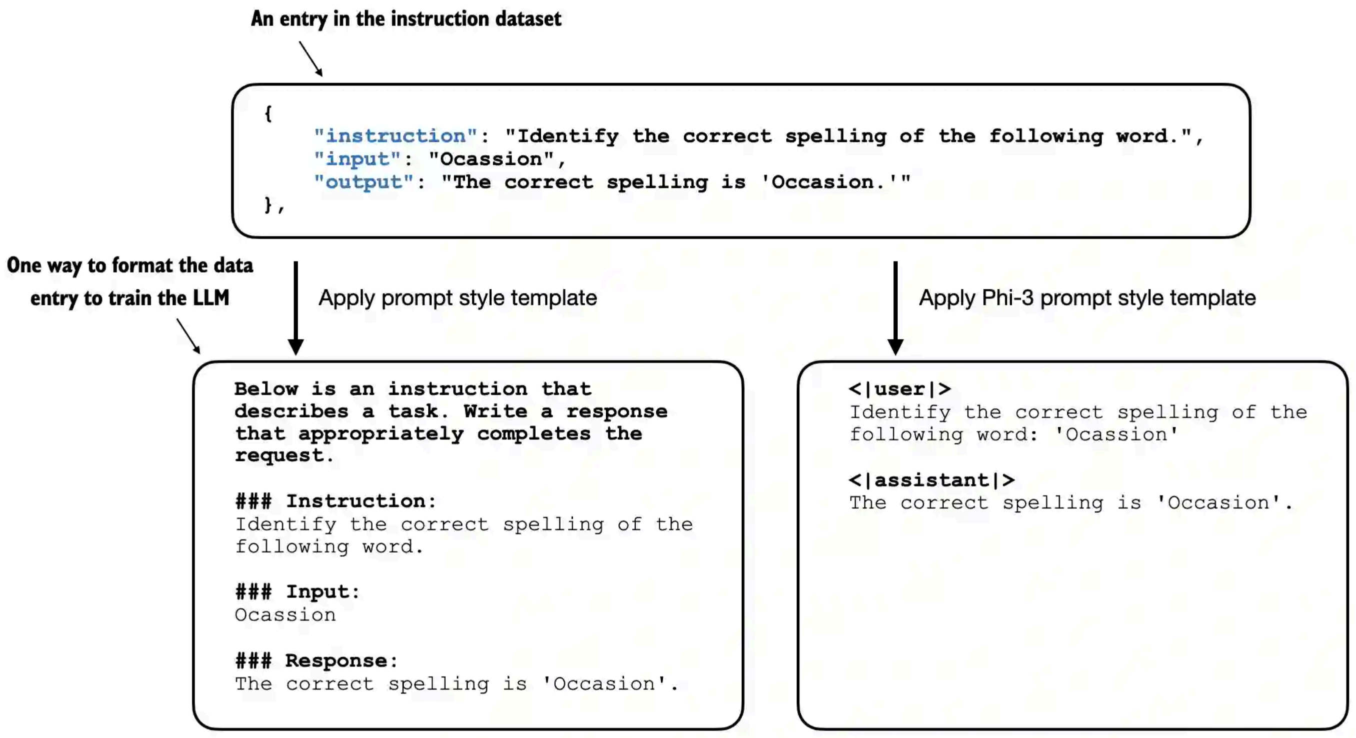
Instruction		Desired response
Convert 45 kilometers to meters.	→	45 kilometers is 45000 meters.
Provide a synonym for “bright”.	→	A synonym for “bright” is “radiant”.
Edit the following sentence to remove all passive voice: “The song was composed by the artist.”	→	The artist composed the song.

- 指令微调通常称为“监督指令微调”，因为它涉及在明确提供输入输出对的数据集上训练模型。

7 - 2 准备用于监督指令微调的数据集

格式化训练数据的方式

> 下左 Alpaca 风格, 下右 Phi-3 风格。


An entry in the instruction dataset
One way to format the data entry to train the LLM
Apply prompt style template
Apply Phi-3 prompt style template

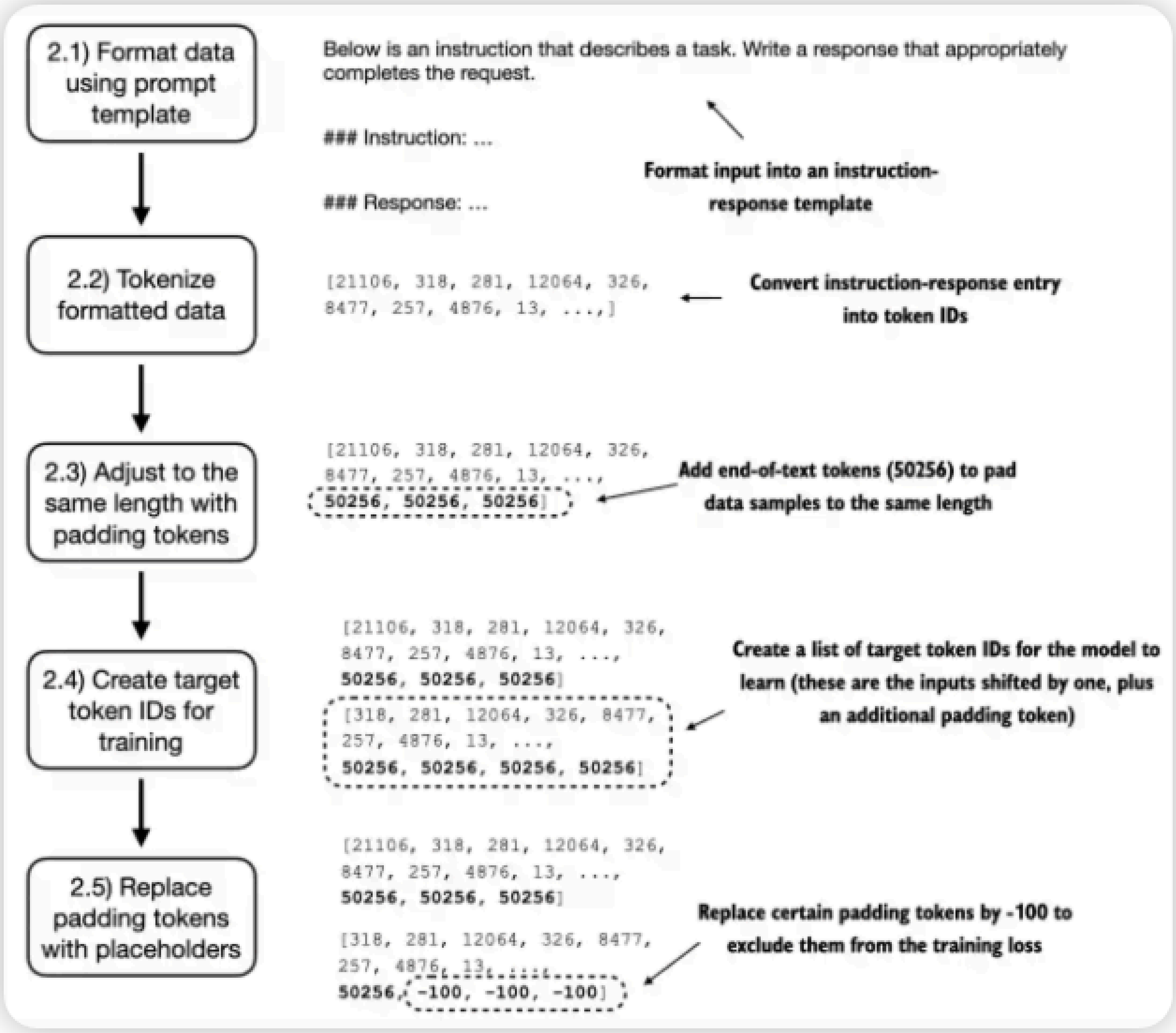
```
{
    "instruction": "Identify the correct spelling of the following word.",
    "input": "Ocassion",
    "output": "The correct spelling is 'Occasion.'"
},
```

```
Below is an instruction that
describes a task. Write a response
that appropriately completes the
request.

### Instruction:
Identify the correct spelling of the
following word.

### Input:
Ocassion

### Response:
The correct spelling is 'Occasion'.
```

```
<|user|>
Identify the correct spelling of the
following word: 'Ocassion'

<|assistant|>
The correct spelling is 'Occasion'.
```

本文选择 Alpaca 模型的格式化方式。

7 - 3 将数据组织成训练批次

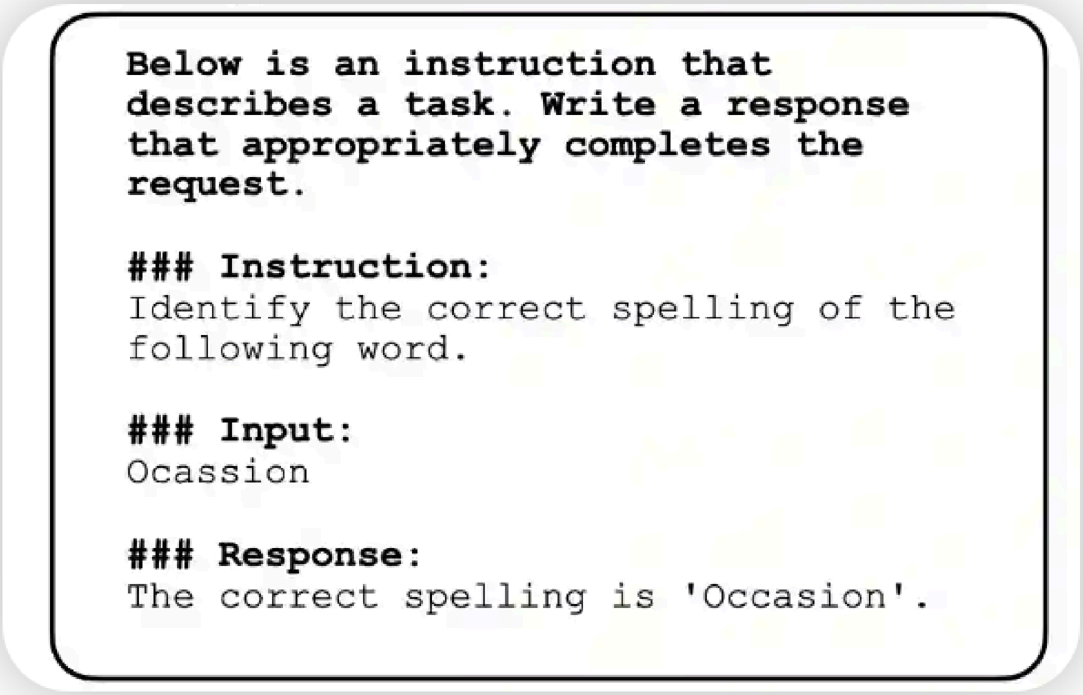

2.1) Format data using prompt template
Below is an instruction that describes a task. Write a response that appropriately completes the request.
### Instruction: ...
### Response: ...
Format input into an instruction-response template
2.2) Tokenize formatted data
[21106, 318, 281, 12064, 326, 8477, 257, 4876, 13, ...,]
Convert instruction-response entry into token IDs
2.3) Adjust to the same length with padding tokens
[21106, 318, 281, 12064, 326, 8477, 257, 4876, 13, ..., 50256, 50256, 50256]
Add end-of-text tokens (50256) to pad data samples to the same length
2.4) Create target token IDs for training
[21106, 318, 281, 12064, 326, 8477, 257, 4876, 13, ..., 50256, 50256, 50256]
[318, 281, 12064, 326, 8477, 257, 4876, 13, ..., 50256, 50256, 50256, 50256]
Create a list of target token IDs for the model to learn (these are the inputs shifted by one, plus an additional padding token)
2.5) Replace padding tokens with placeholders
[21106, 318, 281, 12064, 326, 8477, 257, 4876, 13, ..., 50256, 50256, 50256]
[318, 281, 12064, 326, 8477, 257, 4876, 13, ..., 50256, -100, -100, -100]
Replace certain padding tokens by -100 to exclude them from the training loss

将数据集处理为inputs_tensor, targets_tensor的步骤

0. 数据集

```
// 一个entry
{
        "instruction": "Identify the correct spelling of the following word.",
        "input": "Ocassion",
        "output": "The correct spelling is 'Occasion.'"
},
```

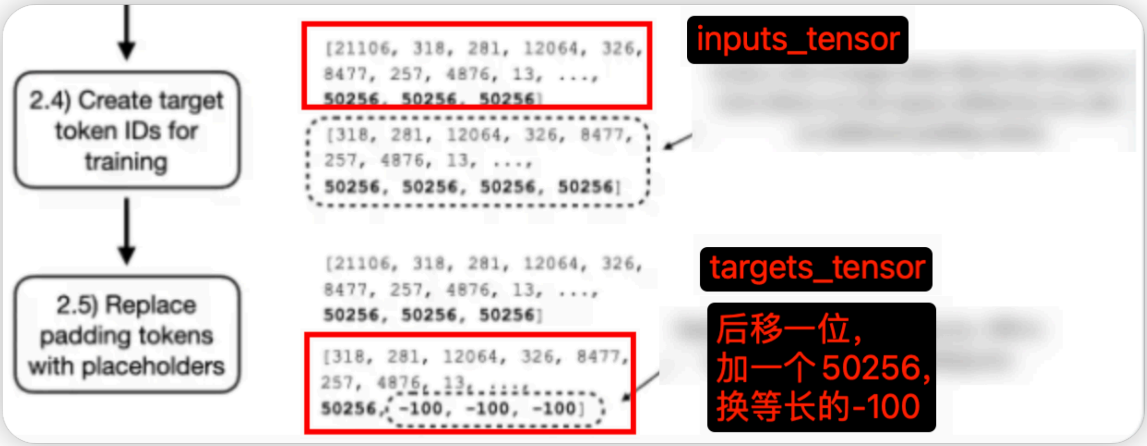
1. 格式化数据



```
Below is an instruction that
describes a task. Write a response
that appropriately completes the
request.

### Instruction:
Identify the correct spelling of the
following word.

### Input:
Ocassion

### Response:
The correct spelling is 'Occasion'.
```

-

2. 字符串标记化 Tokenizer 格式化后的数据
3. 使用填充令牌等长化（同批次等长）
4. 为训练创建目标令牌
5. 将目标向量中的填充令牌替换为占位符



```
[21106, 318, 281, 12064, 326,
8477, 257, 4876, 13, ...,
50256, 50256, 50256]

[318, 281, 12064, 326, 8477,
257, 4876, 13, ...,
50256, 50256, 50256, 50256]

[21106, 318, 281, 12064, 326,
8477, 257, 4876, 13, ...,
50256, 50256, 50256]

[318, 281, 12064, 326, 8477,
257, 4876, 13, ...,
50256, -100, -100, -100]
```

2.4) Create target token IDs for training
2.5) Replace padding tokens with placeholders
inputs_tensor
targets_tensor
后移一位，
加一个 50256,
换等长的-100

-
- 加一个 `endoftext(50256)` ，不想忽略文本结尾标记的第一个实例，因为它可以帮助在响应完成时向 LLM 发出信号
- targets_tensor 和 inputs_tensor 的 `endoftext(50256)` 等长的 `占位符-100` 可以忽略批次中用于将训练示例填充到相等长度的附加文本结尾标记。

- Pytorch 默认-100为占位符：在求 `cross_entropy` 时会忽略改样例。

7 - 4 为指令数据集创建数据加载器

和 6-文本分类微调 > 6 - 3 创建数据加载器 类似。

7 - 5 加载预训练模型

和6-文本分类微调 > 6 - 4 使用预训练权重初始化模型类似。

都是使用hugging face上openai官方的预训练模型。

```
# 模型参数获取

from transformers import GPT2Model

gpt_hf = GPT2Model.from_pretrained("openai-community/gpt2-medium",
cache_dir="checkpoints")

BASE_CONFIG = {
    "vocab_size": 50257,     # Vocabulary size
    "context_length": 1024, # Context length
    "drop_rate": 0.0,        # Dropout rate
    "qkv_bias": True         # Query-key-value bias
    "emb_dim": 1024,
    "n_layers": 24,
    "n_heads": 16
}
```

```
# 加载函数

import numpy as np

def assign_check(left, right):
    if left.shape != right.shape:
        raise ValueError(f"Shape mismatch. Left: {left.shape},
Right: {right.shape}")
    return torch.nn.Parameter(right.clone().detach())

def load_weights(gpt, gpt_hf):

    d = gpt_hf.state_dict()

    gpt.pos_emb.weight = assign_check(gpt.pos_emb.weight,
d["wpe.weight"])
    gpt.tok_emb.weight = assign_check(gpt.tok_emb.weight,
d["wte.weight"])

    for b in range(BASE_CONFIG["n_layers"]):
        q_w, k_w, v_w = np.split(d[f"h.{b}.attn.c_attn.weight"], 3,
axis=-1)
        gpt.trf_blocks[b].att.W_query.weight =
assign_check(gpt.trf_blocks[b].att.W_query.weight, q_w.T)
        gpt.trf_blocks[b].att.W_key.weight =
assign_check(gpt.trf_blocks[b].att.W_key.weight, k_w.T)
        gpt.trf_blocks[b].att.W_value.weight =
assign_check(gpt.trf_blocks[b].att.W_value.weight, v_w.T)

        q_b, k_b, v_b = np.split(d[f"h.{b}.attn.c_attn.bias"], 3,
axis=-1)
        gpt.trf_blocks[b].att.W_query.bias =
assign_check(gpt.trf_blocks[b].att.W_query.bias, q_b)
        gpt.trf_blocks[b].att.W_key.bias =
assign_check(gpt.trf_blocks[b].att.W_key.bias, k_b)
        gpt.trf_blocks[b].att.W_value.bias =
assign_check(gpt.trf_blocks[b].att.W_value.bias, v_b)


        gpt.trf_blocks[b].att.out_proj.weight =
assign_check(gpt.trf_blocks[b].att.out_proj.weight, d[f"h.
{b}.attn.c_proj.weight"].T)
        gpt.trf_blocks[b].att.out_proj.bias =
assign_check(gpt.trf_blocks[b].att.out_proj.bias, d[f"h.
{b}.attn.c_proj.bias"])

        gpt.trf_blocks[b].ff.layers[0].weight =
assign_check(gpt.trf_blocks[b].ff.layers[0].weight, d[f"h.
{b}.mlp.c_fc.weight"].T)
        gpt.trf_blocks[b].ff.layers[0].bias =
assign_check(gpt.trf_blocks[b].ff.layers[0].bias, d[f"h.
{b}.mlp.c_fc.bias"])
        gpt.trf_blocks[b].ff.layers[2].weight =
assign_check(gpt.trf_blocks[b].ff.layers[2].weight, d[f"h.
{b}.mlp.c_proj.weight"].T)
        gpt.trf_blocks[b].ff.layers[2].bias =
assign_check(gpt.trf_blocks[b].ff.layers[2].bias, d[f"h.
{b}.mlp.c_proj.bias"])

        gpt.trf_blocks[b].norm1.scale =
assign_check(gpt.trf_blocks[b].norm1.scale, d[f"h.
{b}.ln_1.weight"])
        gpt.trf_blocks[b].norm1.shift =
assign_check(gpt.trf_blocks[b].norm1.shift, d[f"h.{b}.ln_1.bias"])
        gpt.trf_blocks[b].norm2.scale =
assign_check(gpt.trf_blocks[b].norm2.scale, d[f"h.
{b}.ln_2.weight"])
        gpt.trf_blocks[b].norm2.shift =
assign_check(gpt.trf_blocks[b].norm2.shift, d[f"h.{b}.ln_2.bias"])

        gpt.final_norm.scale = assign_check(gpt.final_norm.scale,
d[f"ln_f.weight"])
        gpt.final_norm.shift = assign_check(gpt.final_norm.shift,
d[f"ln_f.bias"])
        gpt.out_head.weight = assign_check(gpt.out_head.weight,
d["wte.weight"])
```

```
import torch
from previous_chapters import GPTModel

model = GPTModel(BASE_CONFIG)

device = torch.device("cuda" if torch.cuda.is_available() else
"cpu")
load_weights(model, gpt_hf)
```

7 - 6 根据指令数据微调模型

和6-文本分类微调 > 6 - 6 计算分类损失和准确率类似。

使用了 MPS 进行训练，删除缓存的方式：

```
# 删除每批次缓存的方式

    for epoch in range(num_epochs):
        for batch in train_loader:
            # 将数据移动到设备
            inputs, labels = batch
            inputs, labels = inputs.to(device), labels.to(device)

            # 训练步骤
            optimizer.zero_grad()
            outputs = model(inputs)
            loss = loss_function(outputs, labels)
            loss.backward()
            optimizer.step()

            # 清理缓存 - mps上
            del inputs, labels, outputs
            torch.mps.empty_cache()
```

7 - 7 提取并保存响应

与6-文本分类微调 > 模型的保存与复用类似。

就是将模型保存，运行结果保存。

7 - 8 评估微调后的 LLM

下载 ollma 的 3.8B小参数本地模型来评估。

在本地终端，下載并执行phi-3 mini模型

```
ollama run phi3
```

- 他是通过一个 prompt 让 ollama来评估得分。

评估 prompt

```python
def generate_model_scores(json_data, json_key, model="llama3"):
    scores = []
    for entry in tqdm(json_data, desc="Scoring entries"):
        prompt = (
            f"Given the input `{format_input(entry)}` "
            f"and correct output `{entry['output']}`, "
            f"score the model response `{entry[json_key]}`"
            f" on a scale from 0 to 100, where 100 is the best 
score. "
            f"Respond with the integer number only."
        )
        score = query_model(prompt, model)
        try:
            scores.append(int(score))
        except ValueError:
            print(f"Could not convert score: {score}")
            continue

    return scores

scores = generate_model_scores(test_data, "model_response")
print(f"Number of scores: {len(scores)} of {len(test_data)}")
print(f"Average score: {sum(scores)/len(scores):.2f}\n")
```

7 - 9 总结

分类微调和指令微调 inputs_tensor, targets_tensor 的区别

	inputs_tensor	**targets_tensor**
分类微调	`tokenizer.encode(text)` 和补长;	类别标签;

指令微调	格式化;
	`tokenizer.encode(text)` 和补长;
		inputs_tensor后移一位;
		加一个 `endoftext` ;
		加 `endoftext` 等长的 `占位符-100`