评分：_____

上海大学
Shanghai University

人工智能导论

课程实践报告

自监督预训练 + PEFT 微调：轻量级 GPT 垃圾短信分类器构建

学　　号　25725187

姓　　名　韩旭

目录

一、项目介绍

1.1 研究背景

近年来，近年来，以 GPT 系列为代表的大语言模型（LLM）在自然语言处理（NLP）领域取得了突破性进展。其成功的核心在于采用了“预训练-微调”（Pre-training and Fine-tuning）范式。首先，在海量的无标签文本数据上对模型进行预训练，使其学习到通用的语言知识；然后，在特定下游任务的少量有标签数据上进行微调，将通用知识适配于专门应用。本项目旨在完整复现这一经典范式，涵盖从预训练到微调的全过程。

1.2 核心目标

本项目的核心目标是实践并验证针对特定分类任务的 LLM 微调技术。具体而言，本项目旨在：本项目的核心目标是实践并验证“预训练-微调”技术流程。具体而言，本项目旨在：

在一个小规模的无标签文本数据集上，从零开始预训练一个小型 GPT 模型，使其掌握基本的语言规律。选择一个更大、更强的公开预训练 GPT-2 模型作为微调阶段的基础模型。在一个公开的二元文本分类数据集（SMS 垃圾短信分类）上，对该模型进行参数高效微调。实现一个高效的微调策略：仅更新模型的一小部分参数，而非训练整个模型，从而在保留模型泛化能力的同时，降低计算资源需求和训练时间。评估微调后模型的性能，并将其封装成一个可交互的 Web 应用，以直观展示其最终分类能力。

1.3 所用到的 AI 技术

大语言模型（LLM）：采用基于 Transformer 架构的 GPT-2 Small（124M 参数）模型。该模型具备强大的文本理解和表征能力。

迁移学习与微调（Transfer Learning & Fine-tuning）：将 GPT-2 在通用语料上学到的知识，迁移到垃圾短信分类这一特定任务上。通过在目标任务数据上进行有监督训练，微调部分模型参数，使其适应新任务。

参数高效微调（Parameter-Efficient Fine-Tuning, PEFT）：为防止模型在小数据集上发生灾难性遗忘并提高训练效率，本项目冻结了大部分预训练参数，仅对与分类任务最相关的顶层网络层进行训练。

自然语言处理（NLP）：包括使用 tiktoken 进行高效分词、文本填充（Padding）与截断（Truncation）等数据预处理技术。

深度学习框架：使用 PyTorch 构建模型、定义损失函数（交叉熵损失）和优化器（AdamW），并完成整个训练流程。

二、实践过程

2.1 语言模型的预训练

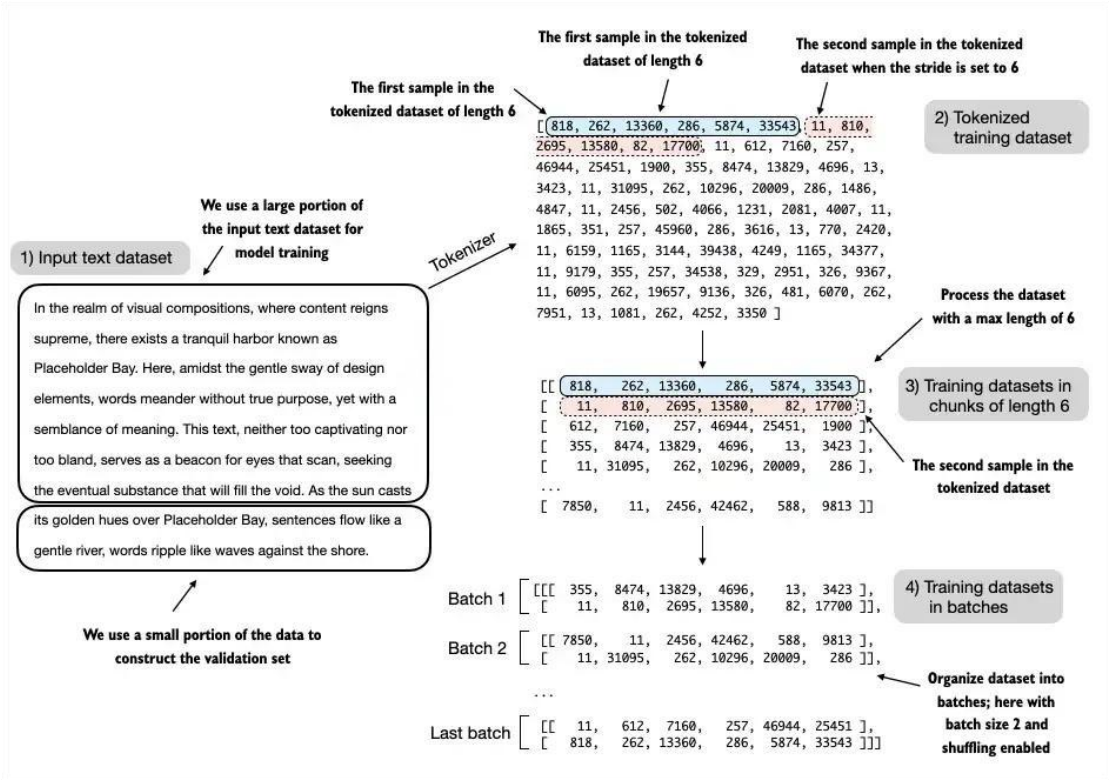
此步骤旨在模拟 LLM 知识获取的第一阶段，即在一个无标签文本语料上教会一个随机初始化的模型掌握语言本身。

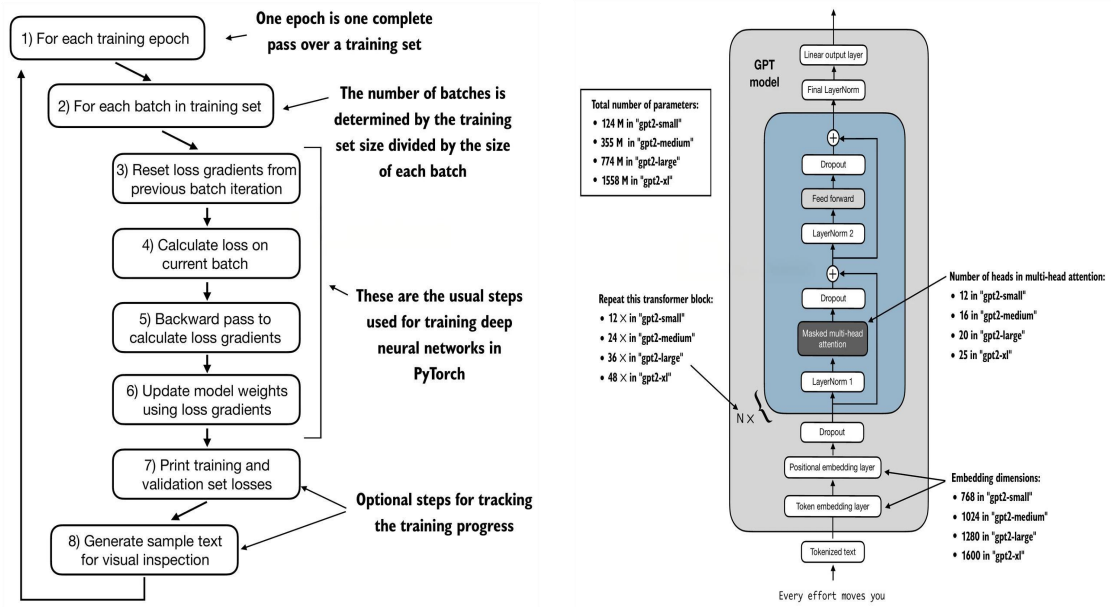
数据集与目标任务：选用英文短篇小说《The Verdict》的纯文本文件作为训练语料。预训练的目标任务是下一个词预测。这是一种无监督学习，模型在没有任何人工标签的情况下，通过文本自身的上下文进行学习。

模型初始化：构建一个参数量较小的 GPT 模型。至关重要的是，该模型的权重被随机初始化，意味着在训练开始前，它不具备任何语言知识。

训练过程与损失函数：采用自回归的方式进行训练。模型接收一段文本序列作为输入，并被要求预测序列中每一个位置的下一个词。使用交叉熵损失函数来衡量模型预测的词与真实文本中的下一个词之间的差距。通过反向传播和 AdamW 优化器不断调整模型权重，以最小化这个差距。

预训练结果：经过多个周期的训练后，模型能够根据给定的前文（Context），生成符合小说风格和语法逻辑的连贯文本。模型已成功从原始文本中学习到了语言的内在规律。





2.2 微调数据的准备与加载

步骤一：数据准备与加载。本项目选用“SMS Spam Collection”数据集，该数据集包含约 5500 条被标记为“ham”（正常）或“spam”（垃圾）的短信。

	Label	Text
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...
...	...	...
5567	spam	This is the 2nd time we have tried 2 contact u...
5568	ham	Will ü b going to esplanade fr home?
5569	ham	Pity, * was in mood for that. So...any other s...
5570	ham	The guy did some bitching but I acted like i'd...
5571	ham	Rofl. Its true to its name

5572 rows × 2 columns

数据平衡处理：原始数据存在严重的类别不平衡问题。为避免模型偏向多数类，采用随机下采样策略，从正常短信中随机抽取与垃圾短信相同数量的样本，构建了一个类别均衡的数据集，使其包含每个类别的 747 个实例。

```
def create_balanced_dataset(df):

    # Count the instances of "spam"
    num_spam = df[df["Label"] == "spam"].shape[0]

    # Randomly sample "ham" instances to match the number of "spam" instances
    ham_subset = df[df["Label"] == "ham"].sample(num_spam, random_state=123)

    # Combine ham "subset" with "spam"
    balanced_df = pd.concat([ham_subset, df[df["Label"] == "spam"]])

    return balanced_df


balanced_df = create_balanced_dataset(df)
print(balanced_df["Label"].value_counts())
```

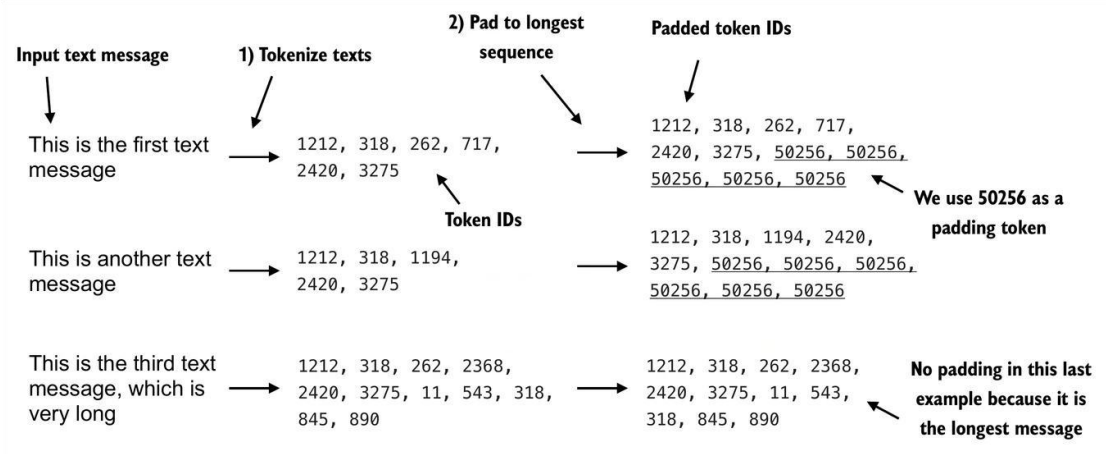
```
Label
ham     747
spam    747
Name: count, dtype: int64
```

	Label	Text
4307	0	Awww dat is sweet! We can think of something t...
4138	0	Just got to <#>
4831	0	The word "Checkmate" in chess comes from the P...
4461	0	This is wishing you a great day. Moji told me ...
5440	0	Thank you. do you generally date the brothas?
...	...	...
5537	1	Want explicit SEX in 30 secs? Ring 02073162414...
5540	1	ASKED 3MOBILE IF 0870 CHATLINES INCLU IN FREE ...
5547	1	Had your contract mobile 11 Mnths? Latest Moto...
5566	1	REMINDER FROM O2: To get 2.50 pounds free call...
5567	1	This is the 2nd time we have tried 2 contact u...

1494 rows × 2 columns

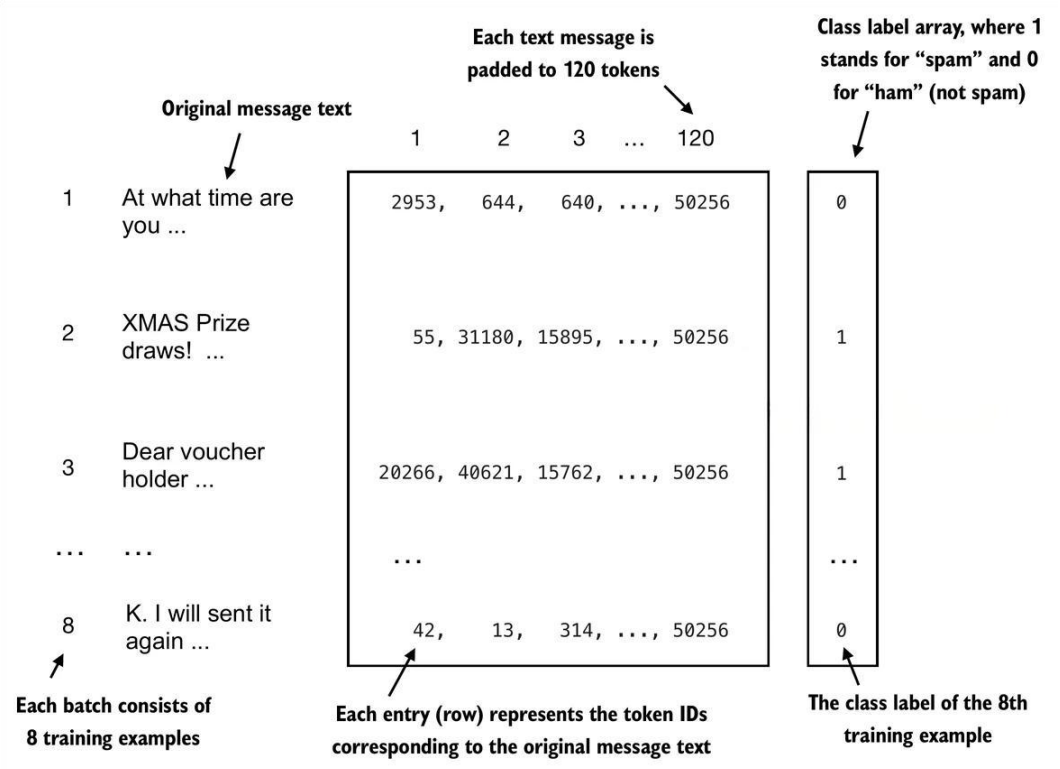
数据集划分：将均衡后的数据集按 7:1:2 的比例划分为训练集、验证集和测试集。

分词与序列化：由于这些文本消息的长度各不相同，如果我们想将多个训练样本合并到一个批次中，我们将所有消息填充至数据集或批次中最长消息的长度。使用与 GPT-2 兼容的 tiktoken 分词器对文本进行编码。为适应模型批处理的需求，对所有文本序列进行填充（Padding），使其长度统一为训练集中最长序列的长度（120 个 Tokens）。



数据加载器：利用 PyTorch 的 Dataset 和 DataLoader 类，构建了用于模型训练和评估的数据管道。设计的 SpamDataset 类会识别训练数据集中的最长序列，并向其他序列添加填充标记，使其长度与该最长序列相匹配。在实际操作中还需要根据最长的训练序列对验证集和测试集进行了填充，长度超过最长训练样本的验证集和测试集样本会被 encoded_text[:self.max_length] 截断。

```python
class SpamDataset(Dataset):
    def __init__(self, csv_file, tokenizer, max_length=None, pad_token_id=50256):
        self.data = pd.read_csv(csv_file)

        # Pre-tokenize texts
        self.encoded_texts = [
            tokenizer.encode(text) for text in self.data["Text"]
        ]

        if max_length is None:
            self.max_length = self._longest_encoded_length()
        else:
            self.max_length = max_length
            # Truncate sequences if they are longer than max_length
            self.encoded_texts = [
                encoded_text[:self.max_length]
                for encoded_text in self.encoded_texts
            ]

        # Pad sequences to the longest sequence
        self.encoded_texts = [
            encoded_text + [pad_token_id] * (self.max_length - len(encoded_text))
            for encoded_text in self.encoded_texts
        ]

    def __getitem__(self, index):
        encoded = self.encoded_texts[index]
        label = self.data.iloc[index]["Label"]
        return (
            torch.tensor(encoded, dtype=torch.long),
            torch.tensor(label, dtype=torch.long)
        )

    def __len__(self):
        return len(self.data)

    def _longest_encoded_length(self):
        max_length = 0
        for encoded_text in self.encoded_texts:
            encoded_length = len(encoded_text)
            if encoded_length > max_length:
                max_length = encoded_length
        return max_length
```

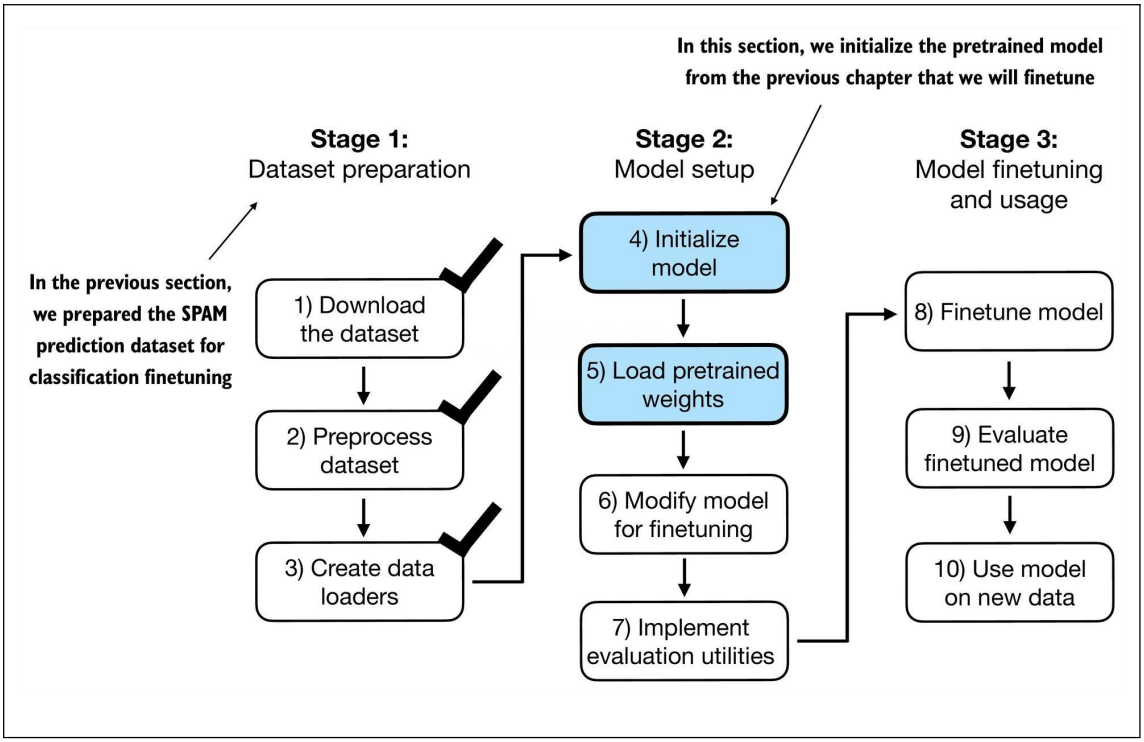


作为验证步骤，我们遍历数据加载器，确保每个批次包含 8 个训练样本，其中每个训练样本包含 120 个标记。我们打印每个数据集中的批次总数如下。

```
Train loader:
Input batch dimensions: torch.Size([8, 120])
Label batch dimensions torch.Size([8])
```

```
130 training batches
19 validation batches
38 test batches
```

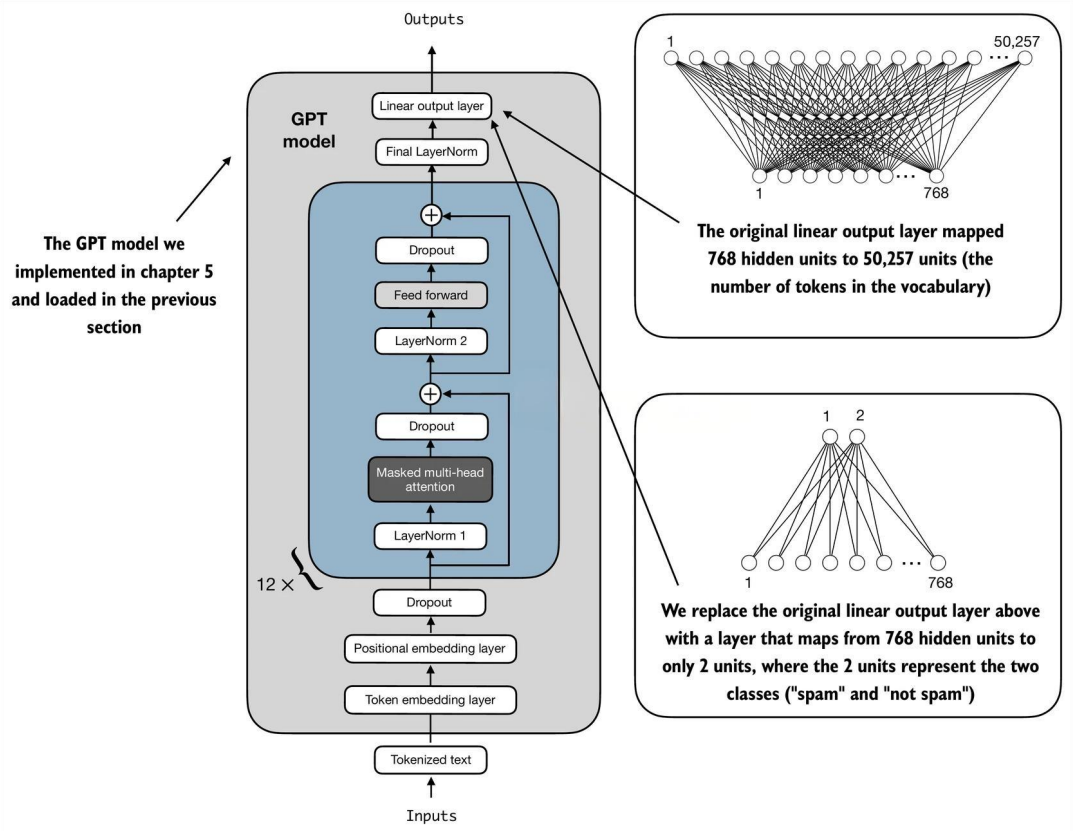
步骤二：模型构建与适配



加载预训练模型：加载拥有 1.24 亿参数的 GPT-2 Small 模型及其预训练权重。

参数冻结：默认情况下，模型的所有参数（requires_grad=False）均被冻结，以防止在微调过程中破坏其已有的语言知识。

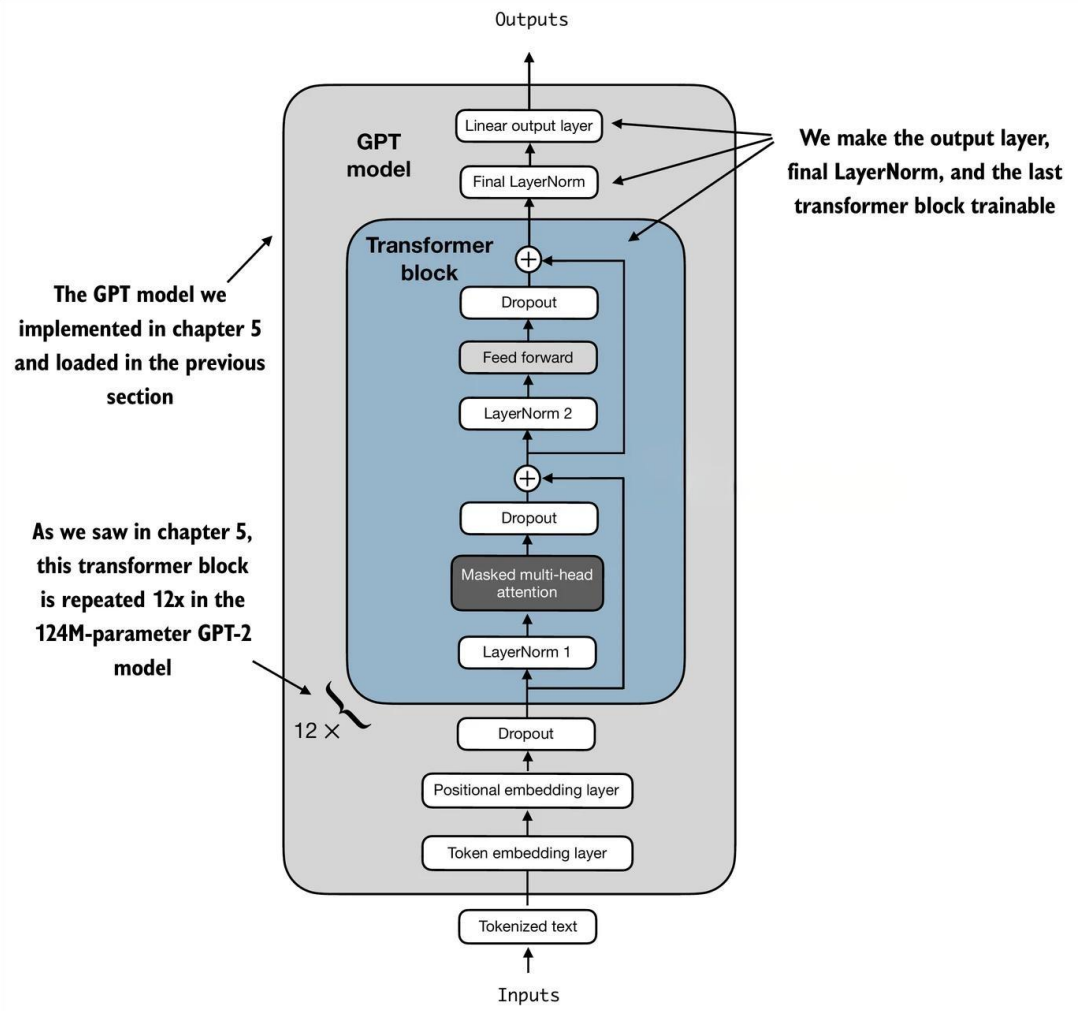
修改分类头：GPT-2 的原始输出层（out_head）用于预测词汇表中的下一个词。为适配二元分类任务，将其替换为一个新的线性层，输入维度为模型的嵌入维度（768），输出维度为类别数（2）。



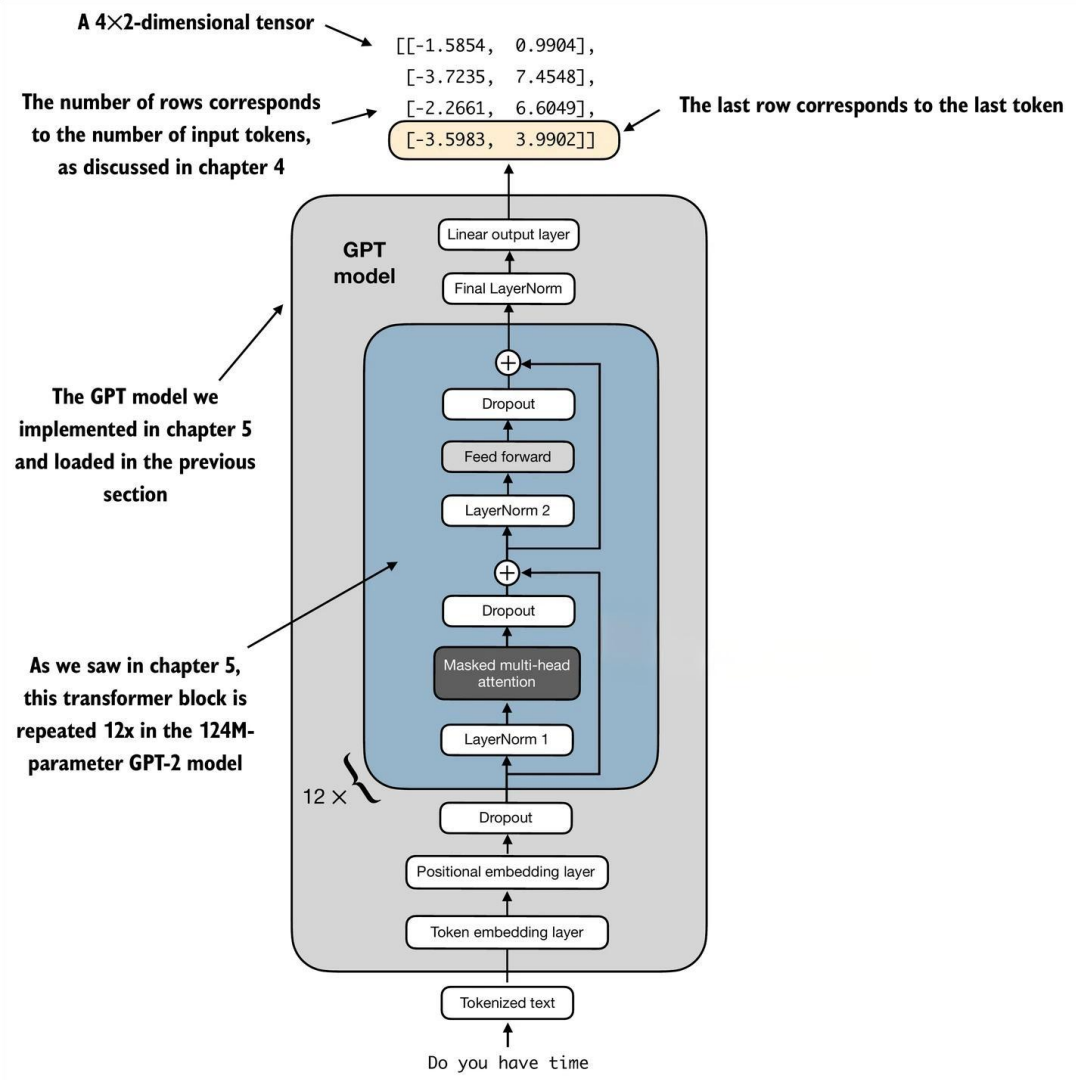
选择性解冻：为提升性能，本项目解冻了最后一个 Transformer 块（trf_blocks[-1]）及最终的层归一化模块（final_norm）的参数。最后一个 Transformer 块的输出包含了对整个输入序列最丰富的语义信息，对其进行微调能更好地适应新任务。我们替换输出层，它原本将层输入映射到 50257 维（词汇表的大小）。由于我们对模型进行微调是为了进行二元分类（预测两类：“垃圾邮件”和“非垃圾邮件”），我们可以像下面这样替换输出层，该输出层默认情况下是可训练的。

```
torch.manual_seed(123)

num_classes = 2
model.out_head = torch.nn.Linear(in_features=BASE_CONFIG["emb_dim"], out_features=num_classes)
```

我们也使最后一个 Transformer 模块和连接最后一个 Transformer 模块与输出层的最终 LayerNorm 模块可训练。



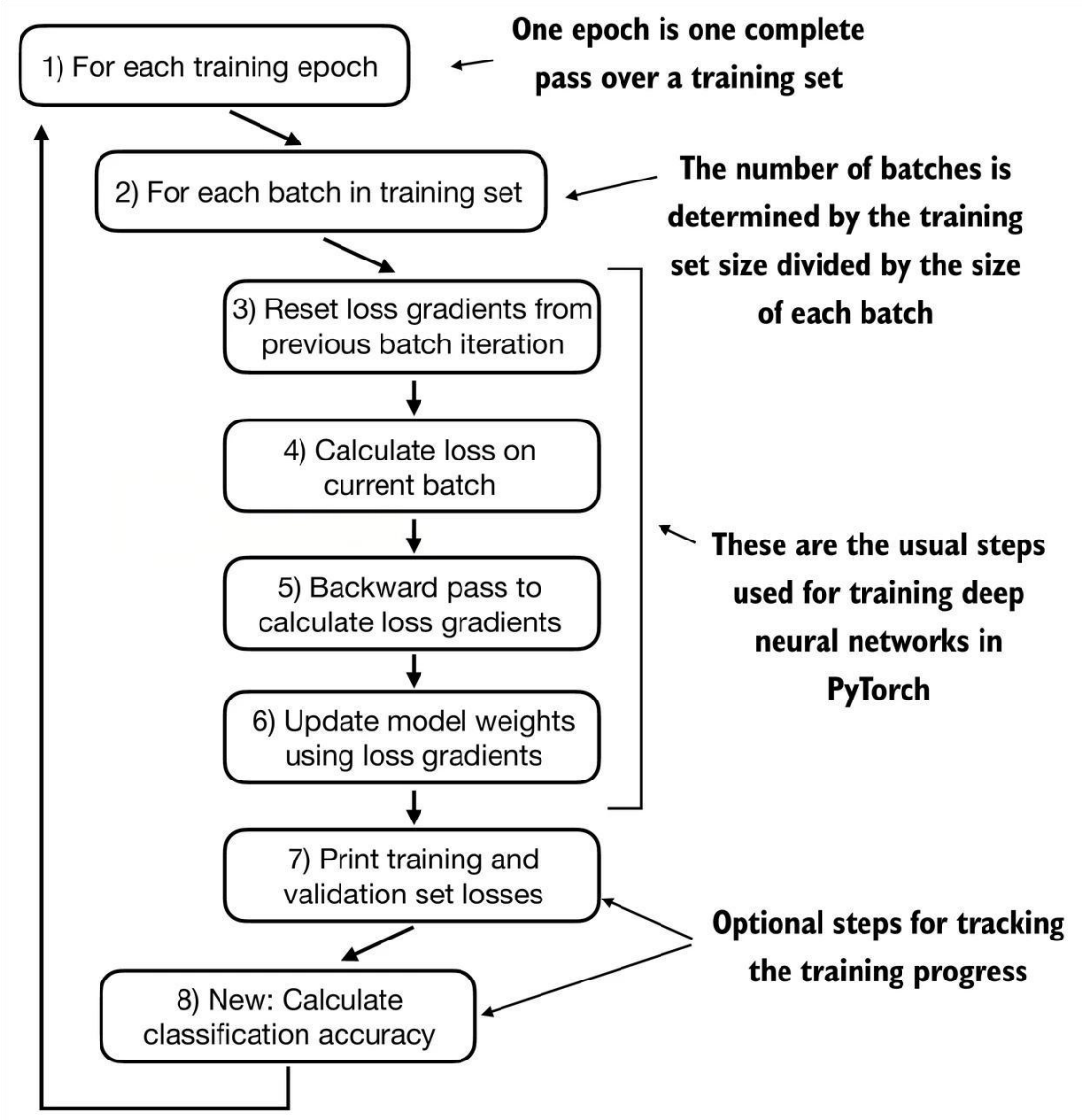
GPT 类模型使用一种因果注意力掩码，该因果掩码使得当前标记仅关注其当前位置和前一个标记的位置。由于我们向模型输入了一个包含 4 个输入词元的文本样本，因此输出由 4 个二维输出向量组成。基于因果注意力机制，第 4 个（最后一个）词元包含的信息量最大，因为它是唯一一个包含了所有其他词元信息的词元。因此，我们对最后一个标记尤为感兴趣，我们将对其进行微调，以用于垃圾邮件分类任务。



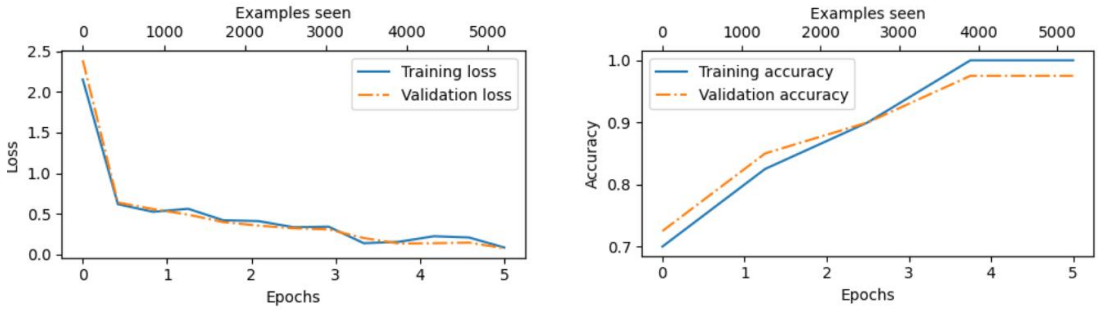
2.3 模型训练与评估

训练过程：采用 AdamW 优化器和最小化交叉熵损失函数，在训练集上对模型进行 5 个周期的训练。我们定义并使用训练函数来提高模型的分类精度。下面这个 train_classifier_simple 函数可以跟踪已看到的训练样本数量（examples_seen），而不是已看到的词元数量。同时在每个 epoch 结束后计算准确率，而不是在每个 epoch 结束后打印示例文本。

```python
def train_classifier_simple(model, train_loader, val_loader, optimizer, device, num_epochs,
                            eval_freq, eval_iter):
    # Initialize lists to track losses and examples seen
    train_losses, val_losses, train_accs, val_accs = [], [], [], []
    examples_seen, global_step = 0, -1

    # Main training loop
    for epoch in range(num_epochs):
        model.train()  # Set model to training mode

        for input_batch, target_batch in train_loader:
            optimizer.zero_grad() # Reset loss gradients from previous batch iteration
            loss = calc_loss_batch(input_batch, target_batch, model, device)
            loss.backward() # Calculate loss gradients
            optimizer.step() # Update model weights using loss gradients
            examples_seen += input_batch.shape[0] # New: track examples instead of tokens
            global_step += 1

            # Optional evaluation step
            if global_step % eval_freq == 0:
                train_loss, val_loss = evaluate_model(
                    model, train_loader, val_loader, device, eval_iter)
                train_losses.append(train_loss)
                val_losses.append(val_loss)
                print(f"Ep {epoch+1} (Step {global_step:06d}): "
                      f"Train loss {train_loss:.3f}, Val loss {val_loss:.3f}")

        # Calculate accuracy after each epoch
        train_accuracy = calc_accuracy_loader(train_loader, model, device, num_batches=eval_iter)
        val_accuracy = calc_accuracy_loader(val_loader, model, device, num_batches=eval_iter)
        print(f"Training accuracy: {train_accuracy*100:.2f}% | ", end="")
        print(f"Validation accuracy: {val_accuracy*100:.2f}%")
        train_accs.append(train_accuracy)
        val_accs.append(val_accuracy)

    return train_losses, val_losses, train_accs, val_accs, examples_seen
```



我使用 matplotlib 绘制训练集和验证集的损失函数图。



从左图的下降斜率可以看出，该模型学习效果良好。此外训练损失和验证损失非常接近，表明该模型没有过拟合训练数据的倾向。右图的准确度图我们可以看到，该模型在第 4 和第 5 个训练轮后都达到了相对较高的训练和验证准确率。

在每个周期结束后，使用验证集评估模型性能，以监控训练过程并防止过拟合。我们使用准确率作为核心评估指标。其中左图是未对模型进行微调前的不同数据集的分类准确率，右图是对模型微调后的分类准确率。可以发现。不管是训练集、验证集、还是测试集。分类准确率都是有着显著的提升。可见微调对于模型性能有着显著的积极作用。

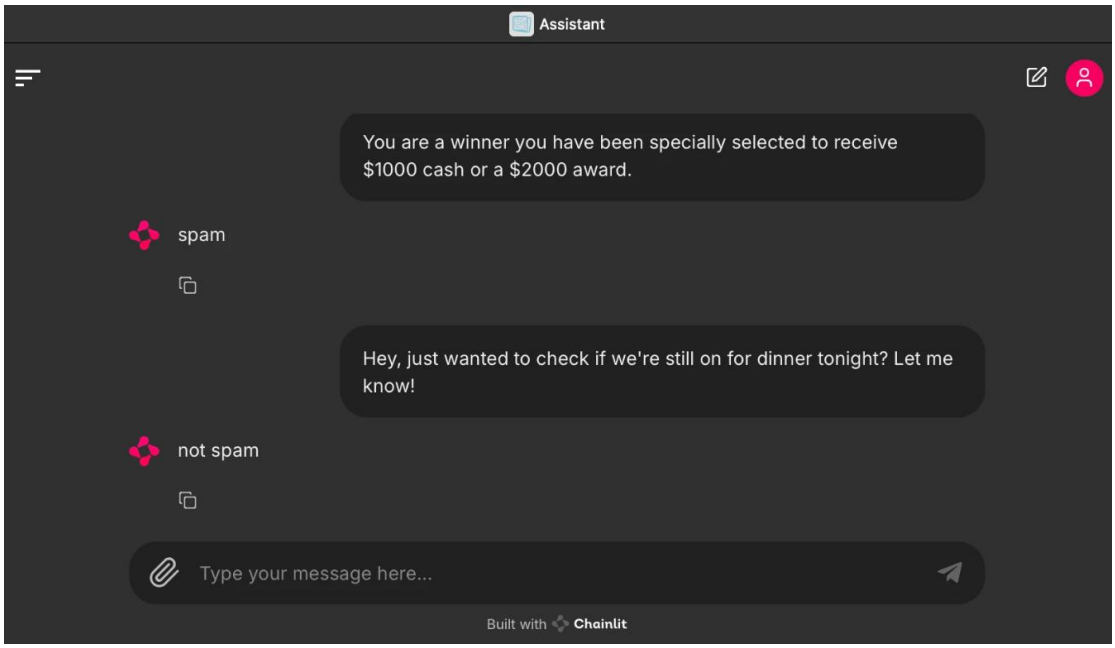
```
Training accuracy: 46.25%
Validation accuracy: 45.00%
Test accuracy: 48.75%
```

```
Training accuracy: 97.21%
Validation accuracy: 97.32%
Test accuracy: 95.67%
```

实验结果：经过 5 个周期的微调，模型性能显著提升。训练损失和验证损失均稳定下降，而准确率则稳步上升并最终收敛。在测试集上的最终准确率达到了 95.67%，远高于随机基线（50%），证明了微调策略的有效性。

2.4 结果展示

构建与基于 GPT 的垃圾邮件分类器交互的用户界面。



三、心得体会

3.1 实践中的难点与解决思路

难点 1：如何让模型从零学会语言？通过自监督学习。通过“下一个词预测”这一巧妙的任务设计，模型可以从海量无标签文本中为自己创造出无穷无尽的“（上下文，目标词）”训练样本，从而完成知识的原始积累。

难点 2：如何在有限资源下应用大模型？通过采用参数高效微调（PEFT）策略，我们无需从头训练一个巨大的模型，也无需更新其所有参数。而是站在巨人的肩膀上，仅微调与任务最相关的少数参数，就能以极低的成本将模型的通用能力适配到特定领域，实现了资源效率和模型性能的最佳平衡。

3.2 知识技能收获

LLM“预训练-微调”全流程：完整地实践了从无到有（预训练）和从有到专（微调）的全过程，对现代 NLP 技术范式有了系统性的认识。

自监督学习与迁移学习：深刻理解了模型如何在无标签数据上进行自监督学习，以及如何将学到的知识通过迁移学习应用于有标签的下游任务。

PyTorch 与 Transformer 应用：熟练掌握了使用 PyTorch 构建和训练 Transformer 模型的全过程。

3.3 反思与感悟

本次实践最深刻的感悟在于分阶段学习的威力。模型首先通过预训练成为一个知识渊博的“通才”，然后通过微调快速转变为特定领域的“专家”。这一流程不仅极大地提升了 AI 模型的性能，更重要的是，它将模型训练的成本分摊开来——少数机构负责昂贵的预训练，而广大开发者和企业则可以低成本地进行微调和应用，这是推动当前 AI 技术浪潮的核心原因。

四、总结展望

4.1 项目成果总结

本项目完整地复现了现代大语言模型的“预训练-微调”范式。首先，通过在无标签文本上进行自回归语言建模，成功地从零开始预训练了一个小型 GPT 模型。随后，基于一个更强大的公开预训练模型 GPT-2，通过参数高效微调策略，实现了一个高精度的垃圾短信分类器，其在测试集上准确率达到 95.67%，并最终被封装为可交互的 Web 应用。

4.2 不足与改进方向

预训练规模局限：预训练受限于数据集和计算资源，其规模较小。未来的研究可以探索在更大、更多样的语料上进行预训练，以获得更强的通用语言模型。

当前只采用了微调顶层模块的策略。未来可以进一步探索如 LoRA、QLoRA 等更先进的 PEFT 方法，以期在更少的训练参数下达到相似甚至更好的效果。

当前 Web 应用只是本地运行的。未来可以学习如何将模型部署到云服务器，通过 API 提供服务，使其成为一个真正可用的在线工具。