工程实践与科技创新 IV-J 作业

韩金汝　陈昱衡　521021910982　521021910939

2024 年 6 月 9 日

目录

1 第一部分：微调 GPT-2

1.1 实验介绍

本次实验的目标是实现对大语言模型（如 GPT-2 [16]）的微调训练以及更多细节上的探索。具体内容包括第一部分对 GPT-2 [16] 模型的微调和以及第二部分对 GPT2 模型的进一步探索。

GPT-2 模型 [16] 是 OpenAI 团队在 2019 年基于 GPT-1 模型的基础上提出的改进模型 [15]。GPT-2 模型的训练没有和 GPT-1 一样采用固定句式的训练方法（如图 1所示），而是通过使用更大的模型、更大的训练数据集，来试图训练模型“理解”所给 **Prompt** 中包含的语义信息，从而识别任务进行回答预测。

与 Google 团队所提出的语言模型 Bert [8] 不同的是，Bert 模型采用的 Transformer [20] 架构的编码器，通过“完型填空”的自监督训练和两句话的蕴含关系（如问答，图 2所示）训练来训练模型，但是可以通过连接分类头来适用于不同的下游任务。而 GPT [16, 15, 5] 系列工作采用的是 Transformer [20] 架构的解码器。并且 GPT-2 [16] 和 GPT-3 [5] 模型并不包含针对特定下游任务分类头。OpenAI 团队希望通过不断扩大模型的体量和训练数据的规模是的模型可以充分理解任务类型。

本次实验中，我们针对 **GPT-2-Small** 进行微调和探索。GPT-2-Small 模型和 GPT-1 模型体量一致，包含 12 层的 Transformer 块，模型宽度为 768，拥有 1.17 亿个可学习参数。我们在机器翻译、情感分类、风格迁移和模型对其等方面对 GPT-2 模型进行了微调并在不同数据集上进行训练和测试。

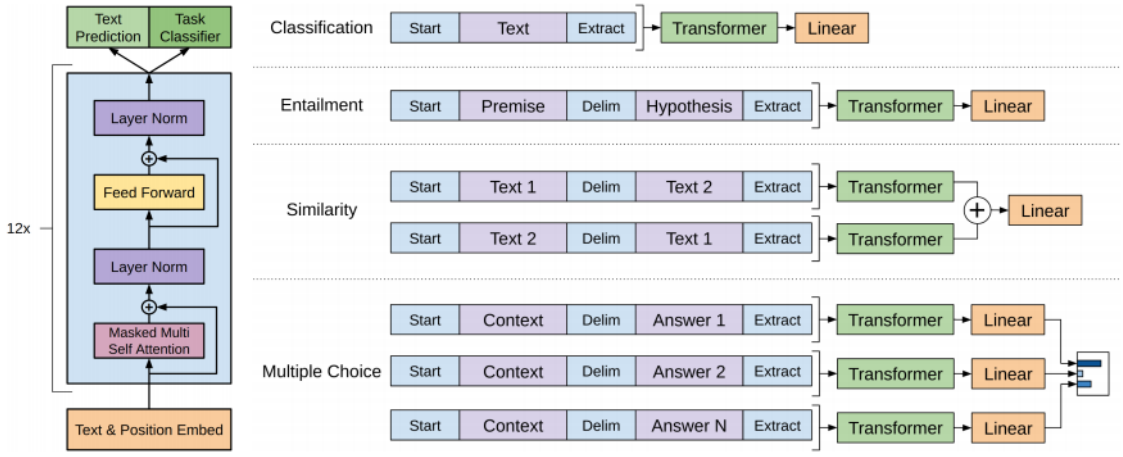

图 1: GPT1 训练方式示意图 [15]

1.2 机器翻译

对 GPT-2 原模型进行机器翻译的简单测试，本实验采用的机器翻译数据集为英语德语翻译数据集 wmt16 [2]（GPT-2 [16] 使用的是 wmt14 数据集 [3]）。

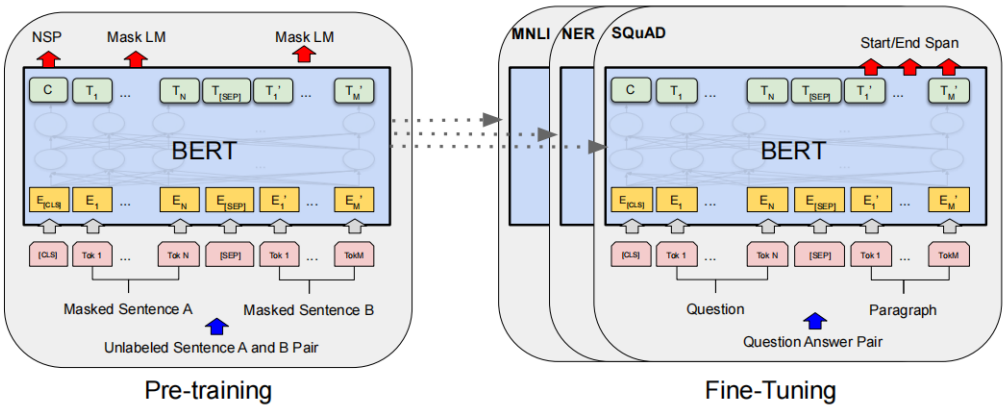

图 2: bert 训练方式 [8]

> "cs": "Následný postup na základě usnesení Parlamentu: viz zápis",
> "en": "Action taken on Parliament's resolutions: see Minutes"

由于我们使用的 GPT-2-Small 的模型参数过少，对任务的理解能力较差，实际测试中也发现模型无法理解 Prompt 中的翻译任务要求。因此我们尝试使用 GPT-1 [15] 中采用固定格式输入的训练方式，给予模型固定的 Prompt 格式（如加入额外的特殊 token，如 <ANSWER> 等），希望通过微调，模型可以理解特定 Prompt 格式的语义任务信息。

对于测试集和验证集，我们使用如下格式使用 wmt16 数据集构建训练语料。

```
"sentence to translate:" + ex["en"] + ", you answer please:" + ex["de"]
```

没有采用特殊的标记的原因是，实验中发现在训练语料有限的情况下，模型无法正确理解新加入的 **token**，为其分配了不正确的注意力，使得在很大一部分测试样例中，模型给出了一连串新加入 token 的输出。因此我们决定不扩大 **vocab**，使用原有的词汇尽量构造模型容易理解的 Prompt。

1.3 情感分析

由于在机器翻译任务中，固定 Prompt 格式的方法效果不佳。在进行实验设计时，对于情感分析任务我们设计的方案和机器翻译一致。但是在进行简单的模型能力测试时，发现如果给出类似 Can you tell me the emotion in the sentence "What a funny day!"? Just choose your answer from the following options: positive, negative, neutral. Please give me your answer: 的提示，模型的回答经常性地表示困惑，如 "What a funny day!"I'm sorry, but I'm not sure what you mean by "what a funny day!"，这显然不利于我们的训练，因为我们无法有效地从 GPT-2 的模型输出中有效地提取出明确答案。我们进行了其它方面的测试，发现 GPT-2-Small 模型对于问题的回答通常做得不好。比

如我们使用 [16] 中 GPT-2-XL（拥有 15.42 亿参数的 GPT-2 模型）模型回答正确的问题来测试 GPT-2-Small，发现其完全无法回答正确，如 What are the common built-in data types in Python? ，GPT-2-Small 给出回答 The following are the most common: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28。所以我们尝试构造肯定语气的问题 Choose one word to decribe the emotion in the sentence "What a funny day!" from the following options: positive, negative, neutral. Please give me your answer: ，GPT-2-Small 给出 "What a funny day!"I'm sorry, but I'm not sure what to say. 的回复。这表明相比于疑问语气的提示，GPT-2 更能理解肯定语气的问题，但是其仍然无法给出正确答案。

综合以上测试结果，我们放弃了和机器翻译类似的构造提示进行问题的方式，转而尝试使用分类头来进行情感预测。由于 GPT-2 采用的是 Transformer 架构的解码器，相比于编码器多了一层 **Masked Multi-Head Attention** 层，用来掩盖当前词后面的序列信息。所以和 RNN 架构类似，GPT-2 模型的最后一层隐含状态的最后一个 token 对应的状态编码了整个序列的信息。所以我们取最后一个 token 的最后一层隐含层状态作为下游任务的分类头的输入。

1.4 风格迁移

我们尝试微调 GPT-2 模型使其能够生成一些具备特定风格的输出。在本次实验中，我们考虑使用歌词作为训练语料。与机器翻译和情感分析不同的是，风格迁移任务没有具体的正确答案，因此我们既不需要构造提示词，也不需要构造额外的分类头。我们只需要利用 GPT-2 原始训练方式，即自回归地根据当前词来预测下一个词即可。在测试阶段，我们对测试集中的歌词进行截断处理，去掉末尾固定长度的歌词，将剩余部分作为提示输入给 GPT-2，然后将 GPT-2 的输出和参考歌词使用 BLEU 的打分方式进行评价，目的是评测微调后的 GPT-2 模型的输出的语义、风格是否和人类近似，以此作为评判标准。

1.5 模型对齐

模型对齐是为了使大语言模型能够产生具有特定人类倾向或者喜好的输出，因为使语言模型变得更大并不意味着它们本身就能更好地遵循用户的意图。例如，大型语言模型可以生成不真实、有毒或对用户没有帮助的输出。ChatGPT [14] 系列模型就是在 GPT3 等模型的基础上，使用如图 3所示的三个步骤对预训练模型进行对齐操作。对于本次实验，我们主要关注第三步的使用（人类反馈）强化学习（RLHF）来对模型进行微调。

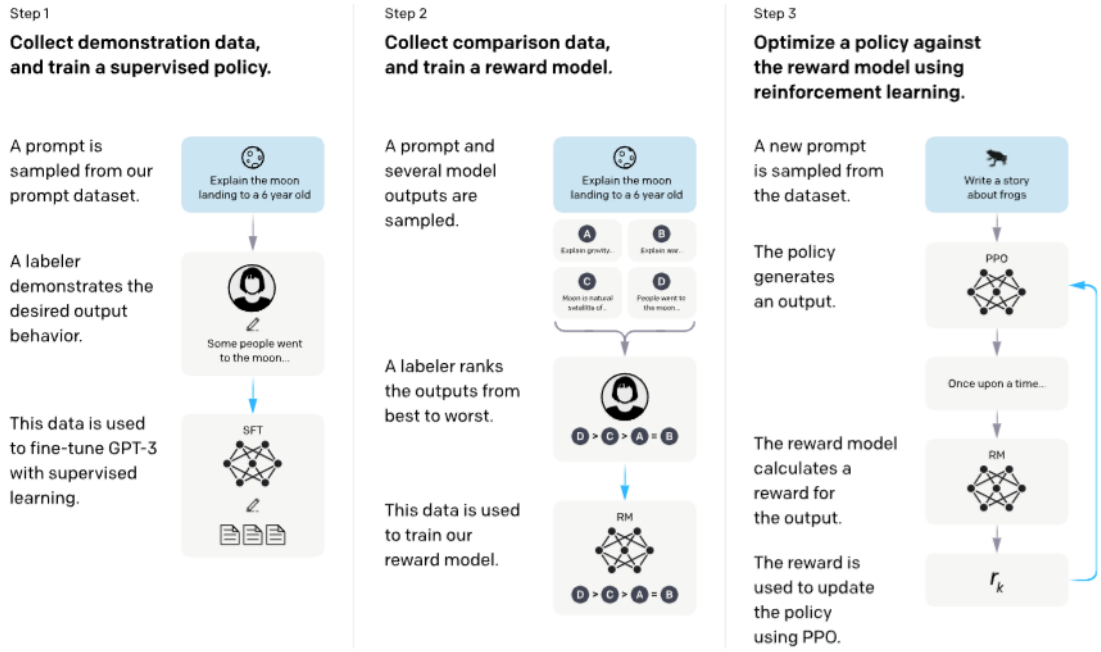

图 3: RLHF 训练过程 [14]

1.6 实现细节

本节主要针对实验的实现细节、使用的数据集以及训练方法（参数调整，模型调整）等方面进行讨论。

1.6.1 机器翻译

我们所考虑的机器翻译任务的主要难点在于构造合适的提示词。具体的细节在章节 1.2中已经详细讨论。我们使用的数据集为 wmt16 [2]，尝试的翻译方向是从英语到德语。强调翻译方向是因为在 [16] 中的实验结果表明 GPT-2-XL 英语翻法语的效果要比法语翻英语的差很多，给出的原因是 GPT-2 模型主要是在英文的数据集上进行训练的，所以生成英语的能力要比生成法语的能力强很多。但是在本次实验中由于计算能力、训练时间等因素影响，只测试了英语翻译德语，并最终使用 BLEU 方法进行评测。

1.6.2 情感分析

对于情感分析任务，在章节 1.3中已经提到需要额外构建一个分类头。对于尝试的第一个来自 Kaggle 的情感分析数据集 [19]，其包含了三种情感：Positive，Negative 和 Neutral。我们尝试了两种下游分类头的结构。第一种是简单的线性分类头，只包含一个 768×3 的线性层。第二种是 MLP，具体包含 $768 \times 1532, 1532 \times 192, 192 \times 48, 48 \times 3$ 和每个线性层之间的 ReLU 激活层。使用这种结构的目的是为了通过增加分类头的模型大小来增强其情感分类的学习能力。

1.6.3 风格迁移

对于风格迁移任务，本次实验中我们从 Kaggle 上选取了两个数据集，分别是五百万首歌的一般数据集 [13]（一般是指不特别针对某个歌手或者某种音乐风格），以及泰勒斯威夫特的歌词数据集 [17]。具体地，我们设置符号化后的序列长度最多为 1024，训练集和验证集中包含完整的歌词信息。在测试阶段，我们只截取符号化测试集中每首歌的前 100 个符号，并指定模型为我们最多再生成 100 个符号。最终使用 BLEU 方法为微调后的模型和原模型进行打分。

1.6.4 模型对齐

数据集方面，我们搜集了一些数据集，包括 [1][7] 以及清华大学最新推出的 ULTRAFEEDBACK [6]，但是上述数据集对模型的性能有一定的要求，例如清华大学的数据集是在 130 亿参数的模型上进行训练和测试的。而 [1][7] 数据集的评判标准是模型的输出是否“有用”，我们认为这是一个很模糊的标准，对于 GPT-2-Small 来说，由于其性能有限，所以可能无法产生出有用的输出，因此可能无法达到很好的训练效果。所以我们转而使用 **IMDB** 数据集，该数据集主要包含正面或者负面的影视评论。我们采用了已经进行了有监督微调的模型 **gpt2-imdb**，并使用 bert 的在情感分类下游任务上微调的模型 **distilbert-imdb** 作为奖励模型，使用 **PPO** 的强化学习策略进行训练。

1.7 实验结果以及分析

1.7.1 机器翻译

我们在章节 1.2中详细地论述了模型在翻译任务中遇到的困难和处理方法。但是经过微调，发现生成的结果仍然不理想。我们在 wmt16 [2] 数据集上做简单的微调，测试阶段随机选取了一些测试集中的一些句子进行人工的评价，得到的模型的输出结果大致如下，英语翻译为德语，

```
sentence to translate:The two wanted to talk about the implementation of the international agreement and about Teheran's destabilising activities in the Middle East., your answer please:Die beiden<|endoftext|>
```

德语翻译为英语，

```
sentence to translate:Obama empfängt Netanyahu, your answer please:das ist einmal einmal.<|endoftext|>
```

可以看到，翻译的结果不很理想。在 [16] 中也有提到，即使是对于 GPT-2-XL 模型来讲，翻译的结果和质量也不佳。由于翻译的语义不正确，所以本次实验不再对翻译结果进行 BLEU 打分。

1.7.2 情感分析

对于在 Kaggle sentiment analysis dataset [19] 上进行训练和测试的两个含有下游分类头的微调模型，使用 **AdamW** 的优化器和交叉熵损失函数，同时设置学习率为 5e-5，训练 10 个 epoch。训练集大小为 27430，测试集大小为 3534。

1. 线性分类头

   对于线性分类头，微调后的模型在测试集上的准确率为 **0.4172**。取出分类头权重和没有微调的 GPT-2 模型拼接后的测试集准确率为 **0.3104**，提高了约 35 个百分点。这意味着线性分类头结合对 GPT-2 模型权重的微调，在情感分析任务上取得了很好的效果。

   同时，考虑到模型规模较大，而训练集规模相比之下很小，所以需要判断模型是否存在过拟合的问题。在训练过程中我们共训练了 10 代，每训练一代在测试集上测试以保存正确率最高的模型。在训练过程中，尽管在训练集上的损失一直下降，但是在第 5 代即在测试集上达到了 0.4172 的正确率且没有进一步提升。说明在之后模型可能存在过拟合问题。

2. MLP 分类头

   对于 MLP 分类头，微调后的模型在测试集上的准确率为 **0.3121**。取出分类头权重和没有微调的 GPT-2 模型进行拼接后的测试集准确率为 **0.2872**，取得了大概 3 个百分点的提升。但是对比线性分类头，MLP 分类头的性能不佳，有较大的性能损失。猜测其中一个可能原因为，由于 GPT-2 模型较大，微调对于 GPT-2 模型本身就是一件很困难的事情。如果分类头过于复杂，在加之本次实验中观察到交叉熵损失的值为个位数，且每个循环的损失降低幅度很小，只有 1e-3 左右的数量级，因此在梯度回传过程中可能发生了梯度消失等问题。而线性分类头可以较好地将梯度回传给 GPT-2 模型进行微调的权重更新。

   同时，考虑到模型规模较大，而训练集规模相比之下很小，所以需要判断模型是否存在过拟合的问题。经过在实验集上的测试，正确率为 **0.3123**，与测试集准确率相差不大，说明模型没有明显的过拟合现象。

1.7.3 风格迁移

风格迁移的评判标准由 BLEU 分数判定。BLEU，即 bilingual evaluation understudy，起初被设计用于评估机器翻译质量的工具，设计思想为机器翻译结果越接近专业人工翻译的结果，则越好。BLEU 算法实际上就是在判断两个句子的相似程度。

1. 一般歌词数据集 [13] 一般歌词数据集包含约 580 万首歌，由于数据量过大，本次实验中选择 10000 首歌作为训练集，1000 首歌作为验证集，1000 首歌作为测试集。每首歌裁剪或者填充到 1024 个 token。

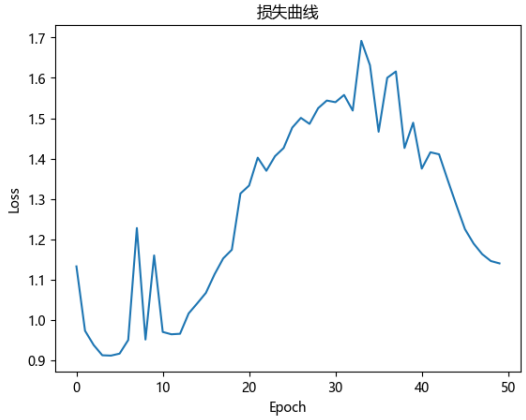

图 4: 泰勒斯威夫特歌词集 [17] 训练损失曲线

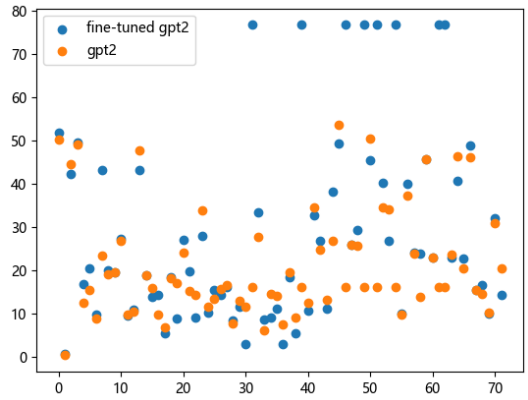

图 5: 泰勒斯威夫特歌词集 [17] 测试 BLEU 分数

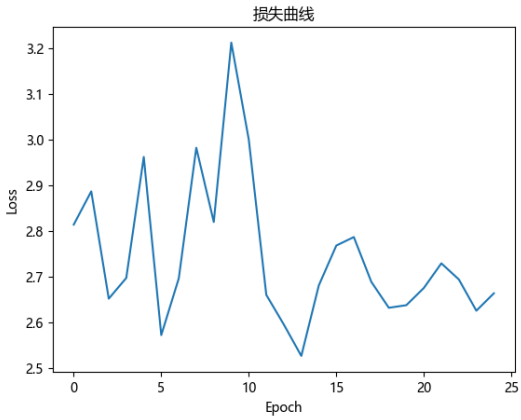

图 6: 一般歌词数据集 [13] 训练损失曲线

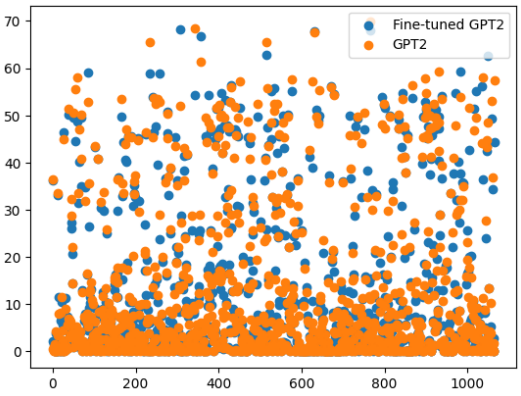

图 7: 微调前后 GPT-2 模型在一般歌词数据集 [13] 上的表现分数

训练过程的损失曲线为图 **??**所示。可以看到，训练的损失并没有很明显的下降，维持在 **2.7** 左右。其中的原因可能是不同歌手、不同曲风的歌词特征并不明显，导致训练时无法准确捕捉到歌词的特征。但是本次实验在收集数据集时，泰勒斯威夫特歌曲集 [17] 已经是较大的单歌手数据集，而同一风格不同歌手的数据集，如 Rap 等，大多包含有不同语言的曲目，因为没有进行选择，而是使用 [13] 进行微调和测试。

最终使用 BLEU 分数对微调后的模型进行评测。结果如图 7所示。可以看到，大多数的生成歌词的 BLEU 分数都较低，微调前后模型的表现并没有明显的优劣之分。统计具体的数值，对于 1000 首测试歌词，有 **508** 首歌词微调后的 BLEU 分数高于原模型。微调后生成歌词的平均 BLEU 分数为 **12.5365**，微调前为 **12.3850**，有小幅度的提升。究其原因，可能是因为训练集中的歌词并没有明显的统一风格，且训练集相对来说较小。

2. 泰勒斯威夫特歌词集 [17] 泰勒斯威夫特的歌词数据集包含约 480 首歌。每首歌裁剪或者填充到 1024 个 token。取 400 首歌作为训练集，其余作为测试集。

   训练过程的损失曲线为图 **??**所示。可以看到，训练的损失呈先下降后上升的趋势。猜测其中的原因是训练的歌词数据集太小，因此模型发生了较为明显的过拟合现象。因此，本次实验的评测模型取第十个循环结束后的模型。

   对于测试集中的每首歌，取符号化后的前 100 个 token 提供给模型，并希望模型生成最多 100 个新的 token。最后采用 BLEU 进行打分。

   最后评测的 72 首歌的 BLEU 分数如图 **??**所示，可以看到，微调后的模型的 BLEU 分数并不能完全超过原 GPT-2 模型。但是可以看到有几首歌的分数微调后的模型远高于原 GPT-2 模型。统计具体的数据为 72 首歌中，微调后平均 BLEU 分数为 **28.2477**，原 GPT-2 模型的平均 BLEU 分数为 **21.4338**，微调后 72 首生成的歌词中，有 **41** 首优于微调之前的模型，调优率为 **0.5694**。

综合以上结果进行分析，可以预见，如果我们使用具有比较明显的统一风格的歌词集，且训练样本足够多，那么微调后的模型有很大希望生成 BLEU 分数较高，也即相对来讲较为符合人类语气、风格的歌词。

1.7.4 模型对齐

在中期报告中未能完成的模型对齐已经完成训练。实验结果我们使用展示部分测试样例的形式呈现。我们从测试集中随机选取了 16 个测试样例分别送入微调前后的 gpt2 模型中，得到微调前后模型的回答，并使用 reward model 得到微调前后的奖励值。由于表格过大，我们分为两个小表格加以呈现。表格 2中呈现的是 prompt 和微调前模型的输出， 3中呈现的是微调后模型的输出和 reward model 给出的微调前后的奖励值。

主观上可以明显感受出微调后的 gpt2 模型的回复更加符合人类的风格并且具有良好的正面的、积极的含义。并且观察较为客观的奖励值，可以发现对于选取的全部测试用例，微调后得到的奖励值均大于微调前。

同时对于全部的测试集，我们统计了微调前后奖励值的中位数和平均值，结果如表 1所示，可以看到，微调前后，奖励值有较为明显的提升，说明集合人类反馈的强化学习方式在模型对齐方面有良好的微调效果。

表 1: 微调前后的奖励值对比

	mean	median
before	0.557807	0.850909
after	2.547478	2.667957

2 第二部分：进一步探索

2.1 不同 token 编码方式：BPE、WordPiece、SentencePiece

Token 编码方式（Tokenization）是自然语言处理（NLP）中的一种技术，用于将文本数据分割成更小的片段，称为 tokens。每个 token 通常代表一个单词、子词、字符或符号，这些 tokens 可以作为模型的输入，便于模型处理和理解文本。

在本次实验中，共探索了三种不同的 token 编码方式在情感分类任务上的性能以及表现，其原理和实验结果如下。

2.1.1 原理及实现

BPE Tokenizer (GPT-2) BPE（Byte Pair Encoding）是一种基于统计的子词分割算法。通过反复合并出现频率最高的字节对，BPE 可以将词汇表的大小减少到一个合理的范围，从而实现对词汇表的有效压缩和编码。其更适用于生成任务比如文本生成、系统对话等。其具体的实现过程为：

1. 统计训练数据中所有字符对的频率。
2. 合并出现频率最高的字符对，将其添加到词汇表中。
3. 重复上述过程，直到达到预定的词汇表大小。

在代码中，GPT2 原始使用的编码方式即为 BPE。

WordPiece Tokenizer (BERT) WordPiece 是一种基于概率的子词分割算法，通过最大化训练数据的似然来生成词汇表。与 BPE 类似，WordPiece 通过反复合并子词来

表 2: 随机选取的一组样例中的 query 和微调前的 response

	query	response (before)
0	This film is	” about the importance of family. ””Lifersose Candidate”” is about”
1	Recap:	 <
2	To finally see	To finally see my hopes and dreams come true.... Thank you everyone!!<|endoftext|>
3	Tasteless. I can’t	even call it a plot device. Give me a
4	While being an impressionable youth when	it comes to religion, and someone who behaves patronisingly towards a community
5	What a surprise; two	advisors suspected that someone had been sleeping with
6	Back in 1997, do I remember	where I rented this movie
7	The following are some	M:F episodes from 1999
8	I’ve always been a	big fan of the 1981 movie world, but now I’m seeing this one
9	If you’re researching UFO	’s, picking your axe and
10	I do miss the	Mara scheming potential of a cartoon that isn’t even more developed in
11	Women have never looked so	” ””so right”” - when I came home”
12	Return to Cabin	music. I loved the scenery especially the top water moments of the train
13	Dwight Frye steals the	show. He is
14	This story	of the dismantling of the USSR is way better than this film.<
15	Ever since I started visiting this site	(living in Germany for about 7 years),

表 3: 随机选取的一组样例中微调后的 response 以及微调前后的 reward 对比

	response (after)	rewards (before)	rewards (after)
0	amazing, also it's whimsical and wonderful fun! It is a win	1.469438195	2.894922495
1	I loved it.	-0.145162717	2.479050636
2	this in English now. edit: splendidly edited	2.229227543	2.560649872
3	wait to see it again. Great job! Great	-2.335234642	1.97157073
4	it comes to the art's quality I found it a very very impressive movie	-1.294993877	2.745095491
5	of course, great, great direction,	0.964190841	2.671462774
6	that beauty! You're	0.662935317	2.14386487
7	of the essential character development sequences	0.418358445	1.781858087
8	fan of theirs, and have a great time together. I really liked it	2.029317141	2.731583118
9	case you'll love it!	-0.714467049	2.349456787
10	show and enjoy it, Yes it's definitely a great adult entertainment;	-1.569046021	2.768733025
11	good in the quite seeing viewer loves. Great	0.737627327	2.574943304
12	was the best and answered the best after that as was brilliant and expressive	2.245842457	2.791222334
13	show, hilarious string	1.470099568	2.664450884
14	is compelling, but exciting, and ultimately well worth watching. I	1.195120573	2.816143513
15	). One really amazing, beautiful story I loved	1.56165278	2.814643621

构建词汇表，但它还考虑了子词的概率分布。其更适合理解任务，比如文本分类、问答系统等。

以下是 WordPiece 编码的简单实现过程：

1. 初始化词汇表，包括所有单字符的词。
2. 计算训练数据中所有可能的子词对的频率和概率。
3. 合并使得训练数据的似然增加最多的子词对。
4. 重复上述过程，直到达到预定的词汇表大小。

在实验中具体的代码实现为将 GPT2 的 tokenizer 直接替换为 BERT 的 tokenizer。

SentencePiece Tokenizer SentencePiece 是一种基于无监督学习的子词分割算法，它将输入文本视为一个字节序列，而不是词序列，从而避免了对语言的依赖。SentencePiece 使用一种称为“最大匹配算法”的技术来生成子词。其适用于多语言模型和需要处理未分词文本的场景。

以下是 SentencePiece 编码的简单实现过程：

1. 将输入文本分割为字节序列。
2. 使用最大匹配算法对字节序列进行分割，生成子词。
3. 统计子词的频率，生成词汇表。
4. 使用词汇表对输入文本进行编码，生成 Token 序列。

在实验中具体的代码实现为将 GPT2 的 tokenizer 直接替换为 AlBERT 的 tokenizer。

2.1.2 结果及分析

采用 3 种不同的编码方式在情感分类任务上的准确率如表4所示。

表 4: 不同 Token 编码方式的准确率

Token 编码方式	**准确率**
BPE Tokenizer (GPT-2)	0.6658
WordPiece Tokenizer (BERT)	**0.6975**
SentencePiece Tokenizer	0.6868

在情感分类任务中，不同的 Token 编码方式对模型的表现有显著影响。表格显示了三种常见 Token 编码方式的准确率，其中 WordPiece Tokenizer (BERT) 的准确率最高，达到 0.6975；SentencePiece Tokenizer 的准确率为 0.6868；BPE Tokenizer (GPT-2) 的准确率最低，为 0.6658。

分析 BPE Tokenizer 通过反复合并出现频率最高的字符对，生成合理的词汇表。这种方法对于常见词和短语的优化效果显著，特别适合生成任务。然而，在情感分类这种需

要细粒度理解任务中，BPE 可能不足以捕捉低频词和新词，从而影响模型的准确率。因此，BPE Tokenizer 在情感分类任务中的表现相对较差。

WordPiece Tokenizer 通过最大化训练数据的似然来生成词汇表，能够对词汇进行细粒度的控制，尤其在处理低频词和新词方面表现突出。情感分类任务通常依赖于捕捉细微的词汇和短语的情感倾向，因此 WordPiece Tokenizer 在该任务中表现最佳，其准确率最高，达到 0.6975。

SentencePiece Tokenizer 不依赖于特定语言，能够处理未分词的文本，适用于多语言模型。这种灵活性使得 SentencePiece 在多语言任务中具有很好的适应性。然而，相比 WordPiece，SentencePiece 在捕捉特定语言的细微情感倾向时稍显不足，因此在情感分类任务中的准确率为 0.6868，略低于 WordPiece。

2.2 不同的位置编码方式：SPE，LPE

2.2.1 原理及实现

正余弦位置编码 (Sinusoidal Positional Encoding) GPT2 种默认使用 SPE 进行位置编码，其生成公式如下：

$$\mathrm{PE}_{(\mathrm{pos},2i)} = \sin\left(\frac{\mathrm{pos}}{10000^{\frac{2i}{d}}}\right) \tag{1}$$

$$\mathrm{PE}_{(\mathrm{pos},2i+1)} = \cos\left(\frac{\mathrm{pos}}{10000^{\frac{2i}{d}}}\right) \tag{2}$$

其中，pos 是位置。i 是维度索引。d 是嵌入维度。

可学习的位置编码 (Learnable Positional Encoding) LPE 可以通过模型学习的位置编码，通常在模型初始化时随机生成，并在训练过程中不断更新。

在具体的实现过程中，在 Hugging Face 的 `transformers` 库中，修改模型的源代码。具体做法为修改 position embedding 模块中的位置编码为使用 `torch` 中的 `nn.Embedding`。

2.2.2 结果及分析

表 5: 不同位置编码方式在情感分类任务中的准确率

位置编码方式	**准确率**
SPE (Sinusoidal Positional Encoding)	0.6658
LPE (Learnable Positional Encoding)	**0.7213**

表格5显示了两种常见位置编码方式的准确率，其中 LPE（Learnable Positional Encoding）的准确率更高，达到 0.7213；SPE（Sinusoidal Positional Encoding）的准确率为 0.6658。

分析 正弦和余弦位置编码（SPE）是一种固定的位置编码方法，不需要通过训练学习。其优势在于能够通过正弦和余弦函数生成不同频率的编码，确保序列中每个位置有唯一的表示。然而，由于其固定的性质，SPE 在捕捉复杂的上下文依赖关系时可能有所欠缺，从而影响情感分类任务的准确率。

可学习的位置编码（LPE）是一种通过模型学习的位置编码方法。在训练过程中，模型能够不断调整位置编码，从而更好地捕捉序列中各位置之间的复杂关系。由于 LPE 具有更高的灵活性和适应性，因此在情感分类任务中的表现优于 SPE，其准确率达到了 0.7213。

在情感分类任务中，可学习的位置编码（LPE）由于其灵活性和适应性，能够更好地捕捉序列中各位置之间的复杂关系，从而在准确率上优于固定的正弦和余弦位置编码（SPE）。因此，在需要捕捉复杂上下文依赖关系的任务中，LPE 可能是更好的选择。

2.3 不同注意力机制：线性、Nystrom

2.3.1 原理及实现

线性注意力 线性注意力是一种旨在提高计算效率的注意力机制，通过简化计算复杂度来加速模型训练和推理过程。传统的注意力机制如 Scaled-Dot Attention，形式为：

$$\text{Attention}(Q, K, V) = \text{softmax}(QK^T)V$$

这里的 $Q \in \mathcal{R}^{n\times d_k}$, $K \in \mathcal{R}^{m\times d_k}$, $V \in \mathcal{R}^{m\times d_v}$。

简单起见这里没有写出缩放因子。由于在情感分类任务中 encoder 没有输入，因此只需要关心 self-attention 的场景。所以此时 $Q, K, V \in \mathcal{R}^{n\times d}$，而一般情况下都有 $n > d$ 甚至 $n \gg d$。

其复杂度为 $\mathcal{O}(n^2)$，其中 n 是序列的长度。当序列长度较长时，这种计算复杂度会显著增加模型的计算量和内存占用。

线性注意力的关键思想是去掉 Softmax 操作，将注意力机制的复杂度降低到线性级别，即 $\mathcal{O}(n)$。这是因为在计算 $QK^\top$ 这一步的复杂度是 $\mathcal{O}(n^2)$，而如果没有 softmax 则三个矩阵连乘 QK^TV 满足结合律，因此可以首先计算 K^TV 再用 Q 左乘。由于 $d \ll n$，因此复杂度由左乘 Q 那一步主导。

本次实验中主要实现了两种线性注意力。

1. **妙用 Softmax**：[18] 提出在 $QK^\top$ 中，$Q, K \in \mathcal{R}^{n\times d}$，如果 Q 在 d 那一维是归一化的，并且 K 在 n 那一维是归一化的，那么 $QK^\top$ 就是自动满足归一化了，所以它给出的选择是：

   $$\text{Attention}(Q, K, V) = \text{softmax}_2(Q) \cdot \text{softmax}_1(K^\top) \cdot V$$

   其中 softmax_1、softmax_2 分别指在第一个（n）、第二个维度（d）进行 Softmax 运算。也就是说，这时候我们是各自给 Q, K 加 Softmax，而不是 $QK^\top$ 算完之后才加 Softmax。

2. **核函数**：首先将 Scaled-Dot Attention 的定义等价地改写为（向量都是列向量）

$$\text{Attention}(Q,K,V)_i = \frac{\sum_{j=1}^{n} e^{q_i^T k_j} v_j}{\sum_{j=1}^{n} e^{q_i^T k_j}} \tag{3}$$

所以，Scaled-Dot Attention 本质是以 $e^{q_i^T k_j}$ 为权重对 v_j 做加权平均。由此可以导出 Attention 的一般化定义：

$$\text{Attention}(Q,K,V)_i = \frac{\sum_{j=1}^{n} \text{sim}(q_i,k_j) v_j}{\sum_{j=1}^{n} \text{sim}(q_i,k_j)} \tag{4}$$

即把 $e^{q_i^T k_j}$ 换成 q_i, k_j 的一般函数 $\text{sim}(q_i,k_j)$，为了保留 Attention 相似的分布特性，我们要求 $\text{sim}(q_i,k_j) \geq 0$ 恒成立。也就是说，我们如果要定义新的 Attention，那么要保留式 (4) 的形式，并且满足 $\text{sim}(q_i,k_j) \geq 0$。
因此想到核函数的形式，如果 q_i, k_j 的每个元素都是非负的，那么内积也就是非负的。为了完成这点，我们可以给 q_i, k_j 各自加个激活函数 ϕ, ϕ，即

$$\text{sim}(q_i,k_j) = \phi(q_i)^T \phi(k_j) \tag{4}$$

其中 $\phi(\cdot), \phi(\cdot)$ 是值域非负的激活函数。[12] 中选择的是 $\phi(x) = \phi(x) = \text{elu}(x) + 1$。

Nystrom 注意力 [**?**] 作者将 Nyström 方法引入 Transformer 中的自注意力矩阵计算中。标准的自注意力矩阵计算如下：

$$Q = XW_Q, K = XW_K, V = XW_V$$

$$S = \text{softmax}\left(\frac{QK^T}{\sqrt{d_q}}\right)$$

注意到直接计算 S 矩阵的计算量是比较大的，其中矩阵乘法 QK^T 的计算复杂度为 $\mathcal{O}(n^2)$。作者使用 Nyström 方法对 QK^T 进行近似分解：

$$\hat{S} = \text{softmax}\left(\frac{1}{\sqrt{d_q}}\begin{bmatrix} A & B \\ C & CA^{-1}B \end{bmatrix}\right) = \text{softmax}\left(\frac{1}{\sqrt{d_q}}\begin{bmatrix} A \\ C \end{bmatrix}\begin{bmatrix} A^{-1}[A & B] \end{bmatrix}\right)$$

因此 Nyström 方法将自注意力矩阵近似成三个矩阵相乘的形式：

$$\hat{S} = \text{softmax}\left(\frac{1}{\sqrt{d_q}}\begin{bmatrix} A \\ C \end{bmatrix}\right) \times \text{softmax}\left(\frac{1}{\sqrt{d_q}} A^{-1}\right) \times \text{softmax}\left(\frac{1}{\sqrt{d_q}}\begin{bmatrix} A & B \end{bmatrix}\right) = \tilde{F} \times \tilde{A} \times \tilde{B}$$

其可视化如图8所示。

其中矩阵 $\begin{bmatrix} A \\ C \end{bmatrix}$ 是从矩阵 QK^T 中随机选择的 k 列，在实践中采用 Segment-Means(sMEANS) 的方法，即先通过平均池化将矩阵 K 下采样为具有 k 列的矩阵 $\tilde{K}$，再用矩阵 $Q\tilde{K}^T$ 近

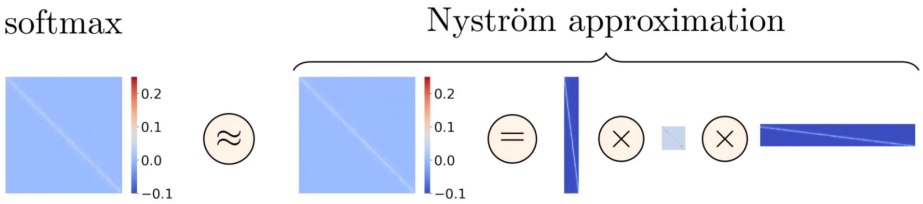

图 8: nystrom 自注意力矩阵分解

似替代矩阵 $\begin{bmatrix} A \\ C \end{bmatrix}$，则有：

$$\tilde{F} = \text{softmax}\left(\frac{Q\tilde{K}^T}{\sqrt{d_q}}\right)$$

矩阵 $\begin{bmatrix} A & B \end{bmatrix}$ 是从矩阵 QK^T 中随机选择的 k 行，采用 sMEANS 的方法，即先通过平均池化将矩阵 Q 下采样为具有 k 行的矩阵 $\tilde{Q}$，再用矩阵 $\tilde{Q}K^T$ 近似替代矩阵 $\begin{bmatrix} A & B \end{bmatrix}$，则有：

$$\tilde{B} = \text{softmax}\left(\frac{\tilde{Q}K^T}{\sqrt{d_q}}\right)$$

矩阵 A 是从矩阵 QK^T 中随机选择的 k 行 k 列交汇处，采用 sMEANS 的方法，即根据平均池化获得的矩阵 $\tilde{K}$ 和矩阵 $\tilde{Q}$，用矩阵 $\tilde{Q}\tilde{K}^T$ 近似替代矩阵 A，则有：

$$\tilde{A} = \text{softmax}\left(\frac{\tilde{Q}\tilde{K}^T}{\sqrt{d_q}}\right)^{-1}$$

则使用 Nyström 方法近似自注意力矩阵的算法流程如下：

Algorithm 1: Pipeline for Nyström approximation of softmax matrix in self-attention

Input: Query Q and Key K.

Output: Nyström approximation of softmax matrix.

Compute landmarks from input Q and landmarks from input K, $\tilde{Q}$ and $\tilde{K}$ as the matrix form;

Compute $\tilde{F} = \text{softmax}\left(\frac{Q\tilde{K}^T}{\sqrt{d_q}}\right)$, $\tilde{B} = \text{softmax}\left(\frac{\tilde{Q}K^T}{\sqrt{d_q}}\right)$;

Compute $\tilde{A} = \text{softmax}\left(\frac{\tilde{Q}\tilde{K}^T}{\sqrt{d_q}}\right)^{-1}$;

return $\tilde{F} \times \tilde{A} \times \tilde{B}$;

2.3.2 结果及分析

实验在情感分类任务上对比了不同的注意力机制，结果如表6所示，其中的 GPU 显存为在固定序列长度为 256、batchsize 为 16 时单张 GPU 的显存消耗，时间为在上述

设置下 RTX4090 训练 1 个 epoch 的平均时间。

Method	Accuracy	GPU Memory (M)	Time (s)
GPT2	0.6658	9507	295
Linear (softmax)	0.5990	9499	290
Linear (kernel)	0.4544	9883	280
Nystrom	0.4069	9759	320

表 6: 不同注意力机制的准确率、GPU 显存占用和训练时间

分析 从实验结果可以看出：从实验结果可以看出：

1. **GPT2**: 使用标准的注意力机制，准确率最高，达到了 0.6658。GPU 显存占用为 9507MB，训练时间为 295 秒。说明标准的注意力机制在处理此类任务时具有较好的性能和效率。
2. **Linear(softmax)**: 使用线性注意力机制，其中 softmax 计算按以下公式进行：

$$\text{Attention}(Q, K, V) = \text{softmax}_2(Q)\text{softmax}_1(K^T)V$$

   这里，softmax_1 和 softmax_2 分别指在第一个 (n) 和第二个维度 (d) 进行的 Softmax 运算。这时我们各自给 Q，K 加 Softmax，而不是先计算 QK^T 再加 Softmax，准确率为 0.5990，略低于标准 GPT2。GPU 显存占用略有降低，训练时间没有显著减少，由于计算复杂度降低，理论上应提高效率，但实际中计算开销并未显著减少，可能是因为其他部分开销较大。
3. **Linear(kernel)**: 使用线性注意力机制，基于核函数计算：

$$\text{sim}(q_i, k_j) = \phi(q_i)^T\phi(k_j)$$

   其中 $\phi(x) = \text{elu}(x)+1$。准确率显著下降到 0.4544，GPU 显存占用增加到 9883MB，训练时间为 280 秒，猜测准确率较低可能是因为核函数没有很好地捕捉到数据特征，导致模型性能下降。虽然理论上计算复杂度降低，但实际实现中开销并没有显著减少，反而增加了内存占用。
4. **Nystrom**: 使用 Nystrom 方法进行近似计算，准确率最低，为 0.4069，GPU 显存占用为 9759MB，训练时间增加到 320 秒，Nystrom 方法的近似计算虽然理论上能降低复杂度，但在实际实现中由于近似计算的开销以及内存访问的开销，导致实际效果不佳，且准确率较低，训练时间也没有减少反而增加。

总体来说，实验结果显示在处理情感分类任务时，标准的注意力机制仍然是最优选择。虽然理论上有些方法可以降低计算复杂度，但在实际应用中，近似方法和核方法并没有显著提高效率，反而降低了模型性能。这可能是因为这些方法在捕捉长距离依赖关系和特征表达方面存在不足，导致准确率下降。

2.4 模型结构探索：Mamba

2.4.1 原理

状态空间模型（State space models，SSMs）[11][9] 从线性时不变控制系统中汲取灵感，将刺激信号 $x(t) \in \mathcal{R}^L$ 映射到响应信号 $y(t) \in \mathcal{R}$。基本上，这些模型被表述为线性常微分方程（ODE）如下：

$$h'(t) = Ah(t) + Bx(t),$$
$$y(t) = Ch(t) + Dx(t),$$

其中 $A \in \mathcal{C}^{N\times N}$ 表示演化参数，$B \in \mathcal{C}^{N\times 1}$ 和 $C \in \mathcal{C}^{1\times N}$ 是状态投影参数。$D \in \mathcal{C}^1$ 是跳过连接。

为了更好地融入深度学习算法，最近的 S4[11] 和 Mamba[9] 模型通过离散化过程将连续时间 ODE 转换为离散函数 [10]。这种转换对于灵活处理不规则采样或缺失数据，以及计算效率操作非常重要。数学上，上述连续 ODE（方程 1 和 2）通过零阶保持器（ZOH）离散化如下：

$$\bar{A} = \exp(\Delta A),$$
$$\bar{B} = (\Delta A)^{-1}(\exp(\Delta A) - I)\Delta B,$$

$$\bar{C} = C,$$

其中 Δ 是离散化的时间尺度参数。给定输入信号 $x_k \in R^{L\times D}$，方程 1 和方程 2 的离散版本可以重写为：

$$h_k = \bar{A}h_{k-1} + \bar{B}x_k,$$

$$y_k = \bar{C}h_k,$$

基于 SSMs 的线性特性，输出信号可以通过输入序列的离散卷积并行计算：

$$\mathcal{K} = (C\bar{B}, C\bar{A}\bar{B}, ..., C\bar{A}^{L-1}\bar{B}),$$

$$y_k = x_k * \mathcal{K},$$

其中 L 是输入序列 x 的长度，$\mathcal{K} \in \mathcal{R}^L$ 是结构卷积核。虽然这种计算需要 $O(L^2)$ 次乘法，其复杂度可以通过快速傅里叶变换（FFT）[4] 减少到 $O(L\log(L))$。

2.4.2 实现

具体的代码实现为在 GPT2 的基础上将整个 GPT2Block 替换为 MambaBlock，修改后的模型结构如图所示：具体的代码实现为在 GPT2 的基础上将整个 GPT2Block 替换为 MambaBlock，修改后的模型结构如图9所示：

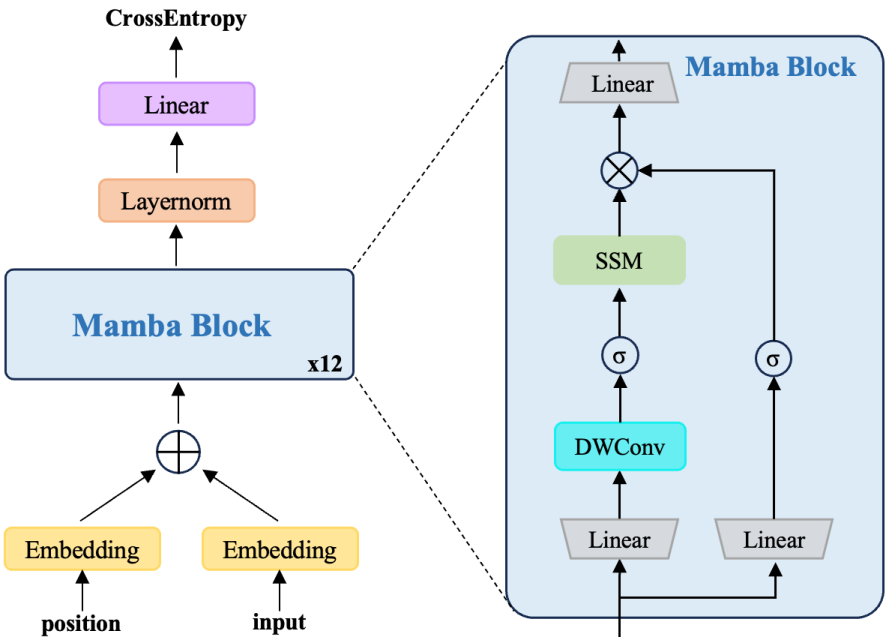

图 9: 使用 Mamba 的模型

2.4.3 结果及分析

原始的 GPT2 模型与使用 Mamba 的模型在测试集上的准确率以及在测试集上的平均推理时间（batch size 为 16，序列长度为 256）如表7所示。

表 7: GPT2 与 Mamba 的准确率以及推理时间对比

模型结构	准确率	推理时间 (s)
GPT2	0.6658	11
Mamba	**0.7054**	**9**

可以看出，Mamba 模型在情感分类任务上取得了更高的准确率，同时在推理时间上也优于使用自注意力机制的 GPT2 模型。这是由于在推理过程中基于自注意力机制的计算为 $O(n^2)$ 的复杂度，而 Mamba 采用 SSM 为线性复杂度。这种推理时间上的差异在序列长度越长的情况下优势越明显。因此，就性能而言，Mamba 在推理速度和准确性方面表现出色。

3 分工

表 8: 组员以及工作

成员	工作占比	具体工作
陈昱衡	50%	第一部分主要内容
韩金汝	50%	第二部分主要内容

4 Bibliography

[1] Yuntao Bai, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova DasSarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Tom Henighan, Nicholas Joseph, Saurav Kadavath, Jackson Kernion, Tom Conerly, Sheer El-Showk, Nelson Elhage, Zac Hatfield-Dodds, Danny Hernandez, Tristan Hume, Scott Johnston, Shauna Kravec, Liane Lovitt, Neel Nanda, Catherine Olsson, Dario Amodei, Tom Brown, Jack Clark, Sam McCandlish, Chris Olah, Ben Mann, and Jared Kaplan. Training a helpful and harmless assistant with reinforcement learning from human feedback, 2022.

[2] Ond rej Bojar, Rajen Chatterjee, Christian Federmann, Yvette Graham, Barry Haddow, Matthias Huck, Antonio Jimeno Yepes, Philipp Koehn, Varvara Logacheva, Christof Monz, Matteo Negri, Aurelie Neveol, Mariana Neves, Martin Popel, Matt Post, Raphael Rubino, Carolina Scarton, Lucia Specia, Marco Turchi, Karin Verspoor, and Marcos Zampieri. Findings of the 2016 conference on machine translation. In Proceedings of the First Conference on Machine Translation, pages 131–198, Berlin, Germany, August 2016. Association for Computational Linguistics.

[3] Ondrej Bojar, Christian Buck, Christian Federmann, Barry Haddow, Philipp Koehn, Johannes Leveling, Christof Monz, Pavel Pecina, Matt Post, Herve Saint-Amand, Radu Soricut, Lucia Specia, and Ale s Tamchyna. Findings of the 2014 workshop on statistical machine translation. In Proceedings of the Ninth Workshop on Statistical Machine Translation, pages 12–58, Baltimore, Maryland, USA, June 2014. Association for Computational Linguistics.

[4] E Oran Brigham. The fast Fourier transform and its applications. Prentice-Hall, Inc., 1988.

[5] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. Advances in neural information processing systems, 33:1877–1901, 2020.

[6] Ganqu Cui, Lifan Yuan, Ning Ding, Guanming Yao, Wei Zhu, Yuan Ni, Guotong Xie, Zhiyuan Liu, and Maosong Sun. Ultrafeedback: Boosting language models with high-quality feedback. arXiv preprint arXiv:2310.01377, 2023.

[7] Dahoas. Full human feedback (hh) reinforcement learning from human feedback (rlhf) dataset. https://huggingface.co/datasets/Dahoas/full-hh-rlhf, 2021. Accessed: 2024-05-19.

[8] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805, 2018.

[9] Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. arXiv preprint arXiv:2312.00752, 2023.

[10] Albert Gu, Tri Dao, Stefano Ermon, Atri Rudra, and Christopher Ré. Hippo: Recurrent memory with optimal polynomial projections. Advances in neural information processing systems, 33:1474–1487, 2020.

[11] Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. In The International Conference on Learning Representations (ICLR), 2022.

[12] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event, volume 119 of Proceedings of Machine Learning Research, pages 5156–5165. PMLR, 2020.

[13] Nikhil Nayak. 5 million song lyrics dataset, 2024. Accessed: 2024-05-18.

[14] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. Advances in neural information processing systems, 35:27730–27744, 2022.

[15] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training, 2018.

[16] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. OpenAI blog, 1(8):9, 2019.

[17] Deep Shah. Song lyrics dataset, 2024. Accessed: 2024-05-18.

[18] Zhuoran Shen, Mingyuan Zhang, Haiyu Zhao, Shuai Yi, and Hongsheng Li. Efficient attention: Attention with linear complexities. In IEEE Winter Conference on Applications of Computer Vision, WACV 2021, Waikoloa, HI, USA, January 3-8, 2021, pages 3530–3538. IEEE, 2021.

[19] Abhishek Shrivastava. Sentiment analysis dataset, 2024. Accessed: 2024-05-18.

[20] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. Advances in neural information processing systems, 30, 2017.