Language Models are Few-Shot Learners

摘要

在自然语言处理领域，传统模型通常需要大量为特定任务订制的标注数据集，这不仅容易导致过拟合，还限制了跨任务的迁移能力。大规模自然语言模型，如 GPT-2 和 GPT-3，通过预训练以及少量样本学习，能在众多自然语言处理任务中表现优秀。通过进一步提升模型规模和训练方案，这种模型带来了基础研究和实际应用的重大进展。同时，少样本学习提高了模型的跨领域和跨场景迁移能力，为不同场景下的实际问题提供了有效解决方案。本文基于 GPT-2 的 Transformer 模型结构和 GPT-3 的训练超参数，搭建并预训练了 GPT 模型，并通过改进注意力机制等方法，在训练速度上提升了 456.39%。原文在零样本设置下，在 8 个经过测试的语言建模数据集中的 7 个数据集上取得了最先进的结果。这些发现表明，构建能够从自然发生的演示中学习执行任务的语言处理系统是一种很有前景的方法。

关键词： 大语言模型；Transformers；少样本学习；迁移学习

1 引言

近年来，预训练语言表示模型在自然语言处理（NLP）系统中得到了广泛应用，并以越来越灵活和任务无关的方式进行下游迁移任务的应用。最初，通过学习单层表示的词向量 [17] [20]，将其输入到任务特定的架构中进行处理；接着，使用带有多个表示层和上下文状态的循环神经网络（RNN）来形成更强的表示 [3] [15] [21]，尽管这些方法现在仍然应用于特定任务的架构；近期，预训练的循环或 Transformer 语言模型 [26] 已被直接进行微调，完全去除了对任务特定架构的依赖 [22] [5] [8]。这一最新范式在许多具有挑战性的 NLP 任务中取得了显著进展，如阅读理解、问答、文本蕴涵等，并且随着新架构和算法的提出，取得了持续的进展 [13] [24]。

然而，这种方法的一个主要局限在于，尽管架构本身是任务无关的，但仍需依赖于特定任务的数据集进行微调。为了在特定任务上实现良好的性能，通常需要使用包含数千到数十万个示例的专用数据集进行微调。这一要求在实际应用中带来了限制，因为每个新任务都需要大量的带标签示例数据集。此外，存在各种潜在有用的语言任务，例如语法纠正、生成抽象概念示例或评论短篇故事。对于许多这些任务，收集大型监督训练数据集是相当困难的，尤其是在每个新任务都需重复这一过程的情况下。

对比之下，大规模自然语言模型，如 GPT-2 [23] 和 GPT-3 [2]，通过预训练以及少量样本学习，在众多自然语言处理任务中显示出了出色的表现。少样本学习可在多种任务中实现无需进行额外的任务专用训练，使模型有机会解决于不同场景和领域中的实际问题。这不仅出现在传统训练方法上，更在实验设备、设计方案和分析问题中，带来了高效性和新视野。

连续训练和学习量化方案，在提升模型的过程中进一步提高了其在解决复杂问题方面的能力。通过对处理策略和拟合方案的进一步探索，大规模自然语言模型显示了构建和适用方案方面的最优效果。

2 相关工作

2.1 元学习

为了解决现有预训练语言模型的局限性，近年来元学习（meta-learning）成为了一种潜在的解决方案。在元学习框架下，模型在训练时通过学习广泛的技能和模式识别能力，在推理时可以快速适应新任务或识别任务需求。近期的研究 [23] 提出了“上下文学习”的方法，通过将预训练语言模型的文本输入作为任务说明，模型根据自然语言的指令和少量任务示例进行条件化，并直接预测任务的下一步。这种方法虽然展示了初步的潜力，但效果仍然远远不及传统的微调方法。例如，GPT-2 [23] 在“Natural Questions”数据集上的正确率仅为 4%，在 CoQA 上的 F1 分数为 55，仍落后于最先进的微调方法。因此，元学习作为一种解决语言任务的新方法，仍然需要进一步的改进和优化。

2.2 模型规模增大

与此同时，近年来基于 Transformer 的语言模型的规模不断扩展，给这一领域带来了新的突破。从 1 亿参数 [22] 到 3 亿参数 [5]、15 亿参数 [23]、80 亿参数 [25] 和 110 亿参数 [24]，每一次模型规模的增加都带来了生成文本和执行下游任务的性能提升。研究表明，模型规模的增大与许多下游任务（如对数损失）之间存在正相关关系 [10]。在上下文学习的背景下，随着模型规模的增大，模型能够吸收更多的技能和任务，从而提高其在快速适应任务时的能力。

本文的工作集中于对 GPT 模型的复现，并在此基础上通过优化训练过程提高训练速度。原文在多个 NLP 数据集和任务上对模型进行了评估，这些任务涵盖了不同条件下的模型适应能力测试，包括零样本学习、单样本学习和少样本学习。实验结果表明，改进后的 GPT 模型不仅能高效完成训练任务，还在部分评估任务中表现出色，展示了优异的性能。

3 本文方法

由于原文中没有给出 GPT-3 的模型结构和相关说明，故本文以 GPT-2 [23] 的模型结构为基础进行了 GPT 模型的复现。

3.1 本文方法概述

完整的 Transformer [26] 的模型结构如图 1 所示。Transformer 是一种用于自然语言处理的深度学习模型架构，由 Vaswani 等 [26] 在 2017 年提出。它通过自注意力机制和多头注意力来捕捉序列中不同位置之间的依赖关系，并通过位置编码来保留序列的顺序信息。Transformer 的编码器（Encoder）-解码器（Decoder）结构使其能够高效地处理输入和生成输出。由于其完全并行化的特性，Transformer 在训练速度上具有显著优势。自从引入以来，Transformer 在

机器翻译、文本生成等 NLP 任务中取得了巨大成功，并成为许多后续模型（如 BERT [5] 和 GPT 系列 [23] [2]）的基础。

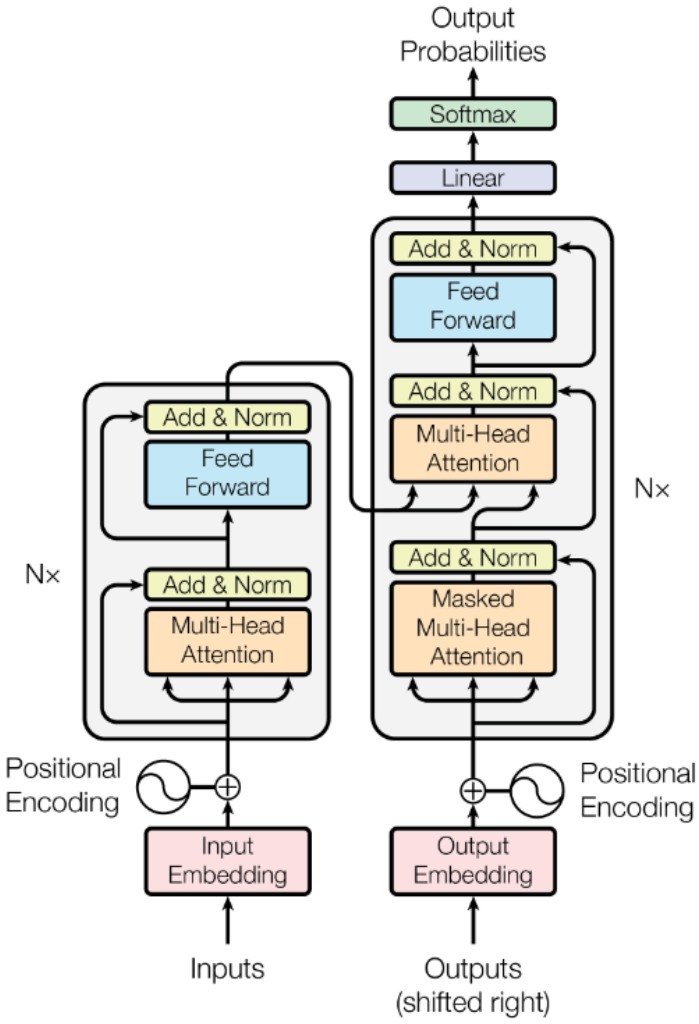

图 1. 完整的 Transformer 模型结构

GPT 模型只使用了图 1 中的右侧解码器部分，且模型在解码过程中并未包括与编码器之间的交叉注意力计算模块。这与传统的 Transformer 架构有所不同，在标准的 Transformer 中，解码器和编码器之间通过交叉注意力层进行信息交互，以便解码器能够利用编码器的输出进行生成。而在 GPT 模型中，由于其是一个自回归生成模型，解码器部分的每一步生成仅依赖于先前的步骤，不需要与编码器交互。因此，GPT 模型仅保留了解码器的自注意力机制。

此外，GPT 模型还在设计上做了一些改进，尤其是在层归一化（Layer Normalization）[1] 的位置上。传统的 Transformer 架构通常将层归一化应用于每个子层的输出端，但在 GPT 模型中，层归一化被移至每个子块的输入端。这种设计灵感来自于预激活残差网络（Pre-activation Residual Networks） [7]，其中层归一化和激活函数先于残差连接进行计算。这种设计有助于在训练过程中梯度传播，减缓梯度消失问题，从而促进更稳定的训练。

在 GPT 模型的架构末端，还增加了一个额外的层归一化模块，放置在最后一个自注意力模块之后。这一额外的层归一化层进一步增强了模型的稳定性，有助于减少模型训练中的不稳定因素，同时优化模型的最终输出表现。

这些设计上的创新，可以进一步提升 GPT 模型的训练效率、稳定性以及生成效果。通

过精心调整层归一化的位置，并剔除交叉注意力机制，GPT 能够更高效地进行自回归生成任务，尤其是在大规模训练和推理时表现出更好的计算性能和稳定性。

修改后的 GPT 模型的结构如图 2 所示。

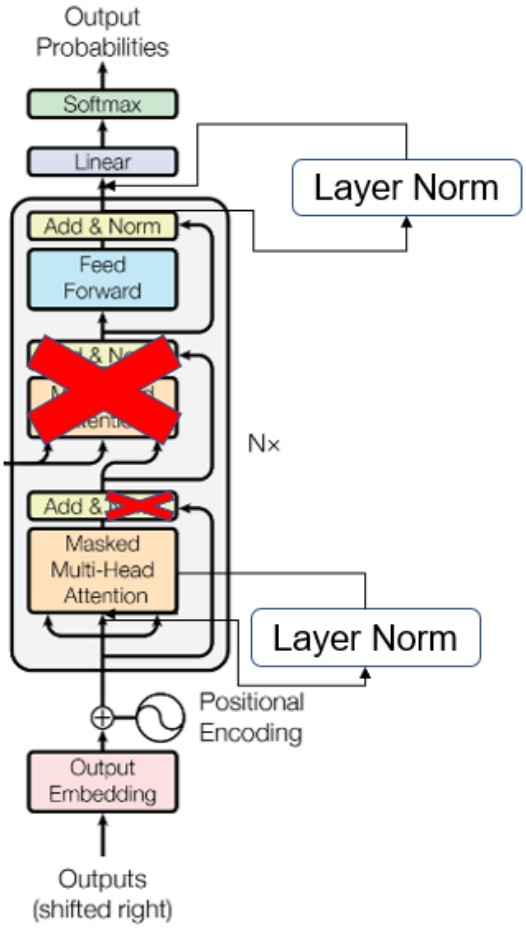

图 2. GPT 模型的结构

3.2 归一化的设计

原始 Transformer 模型是 Post Norm，而 GPT 系列是 Pre Norm [19]，如图 3 所示。使用 Post Norm 的 Transformer 模型训练鲁棒性不高，比如训练 Transformer 时对学习率预热（Warmup）通常都有用，但不同大小的模型和下游任务，预热的拐点各有不同，因此对参数初始化和超参选择要求较高 [28]。

Pre Norm 与残差神经网络 [6] 类似，残差块直连在一起，保证每一层的输出与输入之间都有一条跨越非线性层的直连通路，这个直连通路的梯度是 1，保证了每一层都会有至少为 1 的梯度回传下来。由于 Post Norm 的残差块之间有层归一化，打破了直连通路的稳定梯度，会导致浅层网络梯度弥散导致学习不充分。

Pre Norm 在更深层的网络中，梯度累计速度会有 $\sqrt{\frac{1}{L}}$ 的衰减，其中 L 为当前网络层数。而 Post Norm 则一直累积，容易发生梯度爆炸现象。因此，GPT 系列的模型使用 Pre Norm 进行归一化。

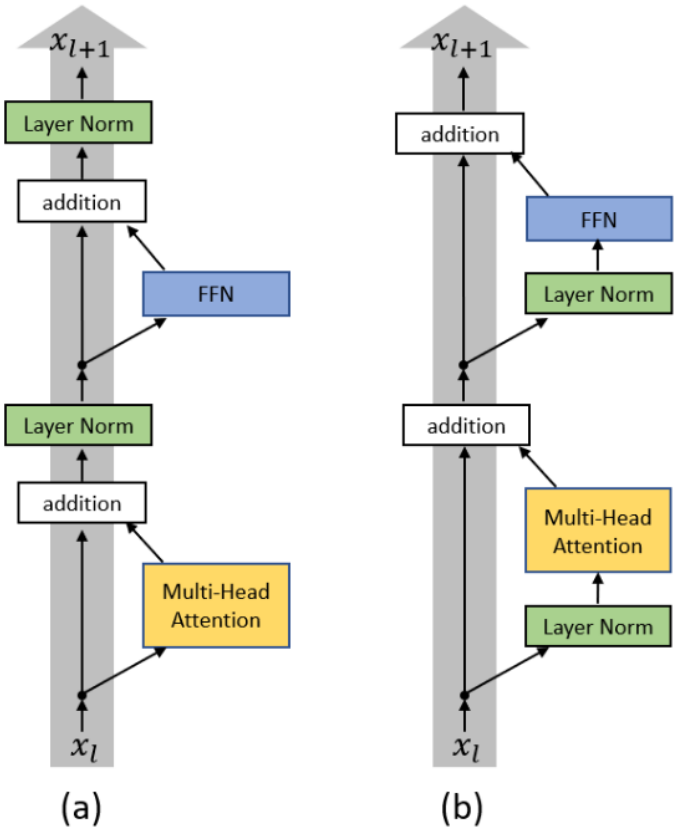

图 3. (a) Post Norm 和 (b) Pre Norm 的示意图

4 复现细节

4.1 与已有开源代码对比

与 Hugging Face [27] 的 Transformers 库中已开源的 GPT-2 代码 [9] 相比，本文复现的 GPT-2 模型结构部分的代码不到 100 行，而前者对 GPT-2 的实现代码接近 2000 行，包含了各个组件更多的技巧策略的实现以及框架，本文的复现代码则主要聚焦于对 GPT-2 模型核心结构的复现，代码量小，结构简单，易于理解和修改。

本文对 GPT 模型的核心结构的复现主要代码如图 4 和图 5 所示。

```python
class CausalSelfAttention(nn.Module):
    def __init__(self,config):
        super().__init__()
        assert config.n_embd %config.n_head==0
        self.c_attn=nn.Linear(config.n_embd,3*config.n_embd)
        self.c_proj=nn.Linear(config.n_embd,config.n_embd)
        self.c_proj.NANOGPT_SCALE_INIT = 1
        self.n_head=config.n_head
        self.n_embd=config.n_embd
        self.register_buffer( name: 'bias',torch.tril(torch.ones(config.block_size,config.block_size)).view(1,1,config.block_size,config.block_size))

    def forward(self,x):
        B,T,C=x.size()
        qkv=self.c_attn(x)
        q,k,v=qkv.split(self.n_embd,dim=2)
        k=k.view(B,T,self.n_head,C//self.n_head).transpose(1,2)
        q=q.view(B,T,self.n_head,C//self.n_head).transpose(1,2)
        v=v.view(B,T,self.n_head,C//self.n_head).transpose(1,2)

        # att=(q@k.transpose(-2,-1)*1.0/math.sqrt(k.size(-1)))
        # att=att.masked_fill(self.bias[:,:,:T,:T]==0,float('-inf'))
        # att=F.softmax(att,dim=-1)
        # y=att@v
        y=F.scaled_dot_product_attention(q,k,v,is_causal=True)#Flash Attention

        y=y.transpose( dim0: 1, dim1: 2).contiguous().view(B,T,C)
        y=self.c_proj(y)
        return y
```

图 4. 因果自注意力模块的实现

```python
class MLP(nn.Module):
    def __init__(self,config):
        super().__init__()
        self.c_fc=nn.Linear(config.n_embd,4*config.n_embd)
        self.gelu=nn.GELU(approximate='tanh')
        self.c_proj=nn.Linear(4*config.n_embd,config.n_embd)
        self.c_proj.NANOGPT_SCALE_INIT=1
    def forward(self,x):
        x=self.c_fc(x)
        x=self.gelu(x)
        x=self.c_proj(x)
        return x

class Block(nn.Module):
    def __init__(self,config):
        super().__init__()
        self.ln_1=nn.LayerNorm(config.n_embd)
        self.attn=CausalSelfAttention(config)
        self.ln_2=nn.LayerNorm(config.n_embd)
        self.mlp=MLP(config)
    def forward(self,x):
        x=x+self.attn(self.ln_1(x))
        x=x+self.mlp(self.ln_2(x))
        return x
```

图 5. 前向传播模块的实现

图 4 展示了因果自注意力模块（Causal Self-Attention）的代码实现。该模块是 GPT 模型的核心组件之一，用于计算输入序列的自注意力，同时通过加入因果掩码（Causal Mask）确保仅关注先前的时间步，从而实现自回归的生成任务。

图 5 展示了 GPT 模型中前向传播模块的实现。前向传播模块包括两个主要部分：多头自注意力（如图 4 所示的因果自注意力模块）和全连接前馈网络（MLP）。与原文 [23] 一致，每个子模块前后分别加入了层归一化（LayerNorm）以提高训练的稳定性和模型的收敛速度。

4.2 模型结构的参数和训练的超参数

GPT-3 中部分模型结构的尺寸参数如表 1 所示，由于硬件限制，本文只选取了 125M 的参数规模的模型进行了复现和训练。

表 1. GPT-3 中部分模型结构的尺寸参数

模型参数	层数	维度	注意力头数
125 M	12	768	12
350 M	24	1024	16
760 M	24	1536	16
1.3 B	24	2048	24

根据原文附录 B [2] 中的模型训练细节，本文复现的模型训练超参数如表 2 所示。

表 2. 模型训练的超参数

超参数名称	值
β_1	0.9
β_2	0.95
ε	10^{-8}
优化器	AdamW
学习率	6.0×10^{-4}

本文使用 AdamW 优化器进行训练，其中 $\beta_1 = 0.9$，$\beta_2 = 0.95$，$\varepsilon = 10^{-8}$ ，本文将梯度的全局范数裁剪到 1.0，并使用余弦衰减将学习率降低到其值的 10%。在最初进行线性学习率预热。本文还根据模型大小，在训练的最初阶段时，将批量大小从一个小值线性地逐步增加到完整值。在训练期间，数据在没有替换的情况下进行采样（直到达到一轮训练的终点），以最大限度地减少过拟合。所有模型都使用 0.1 的权重衰减以进行正则化 [14]。

4.3 创新点

本文复现工作的主要创新点集中在提高训练速度的同时，最大限度地保证训练精度。通过一系列优化技术的应用，不仅加速了训练过程，还确保了在多 GPU 环境下的高效协同与资源利用。

4.3.1 混合精度训练

在深度学习训练中，传统的浮点精度计算（单精度浮点数）虽然能够提供足够的模型精度，但其在计算和存储上的开销较大，尤其是在大规模模型训练时，资源消耗非常高。混合

精度训练 [16] 通过结合半精度（FP16 或者 BF16 等）和单精度（FP32）浮点数计算，既能保持训练的稳定性，又能够显著减少计算资源的使用。混合精度训练将计算中不需要高精度的部分（例如梯度更新、前向推理等）使用半精度浮点数执行，从而减轻了计算负担，并通过适当的调整保持训练精度。与此同时，计算过程中对梯度的累加和反向传播使用单精度浮点数来保证数值稳定性。通过这种方法，训练的内存使用量和计算时间得到显著降低，而不牺牲模型的精度。

4.3.2 注意力机制的改进

为了进一步提高训练速度和内存利用效率，本文应用了一种新的注意力机制（Flash Attention）[4] 替代因果自注意力机制进行训练。Flash Attention 的核心原理是将输入数据分块处理，利用 GPU 的内存层次结构，减少对高带宽内存（HBM）的读写操作。它将输入数据分成若干块，将这些块从 HBM 加载到快速缓存（SRAM）中，执行注意力操作时仅在 SRAM 中进行计算，减少了频繁的内存读写操作。

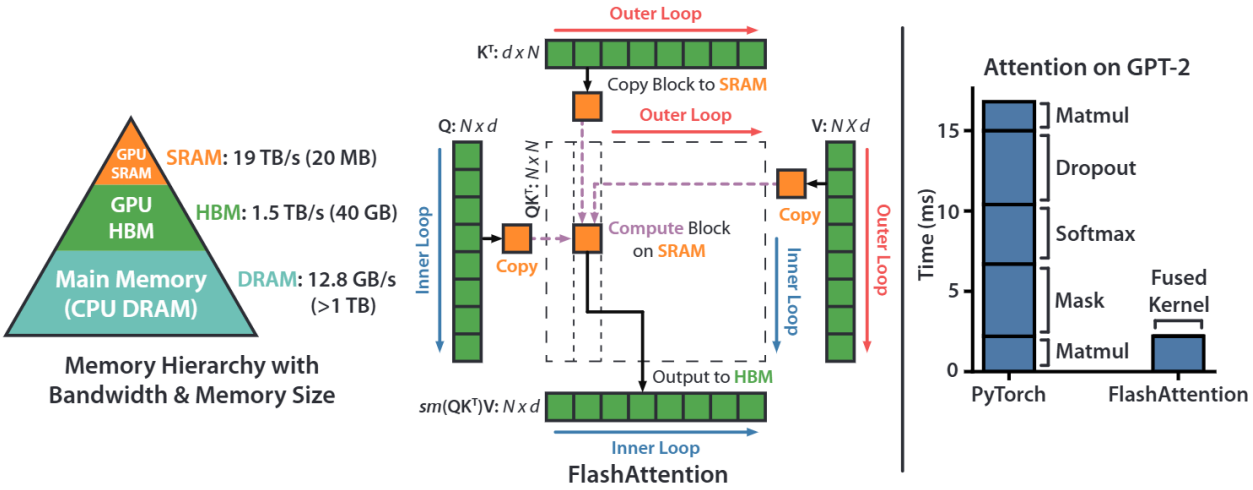

图 6. Flash Attention 机制

如图 6 所示，在左图中 Flash Attention 使用平铺来避免 $N \times N$ 的注意力矩阵 (虚线框) 在相对较慢的 GPU HBM 上实现。在外部循环 (红色箭头) 中，Flash Attention 循环通过 K 和 V 矩阵并将它们加载到 SRAM。在每个块中，Flash Attention 循环遍历 Q 矩阵 (蓝色箭头)，将它们加载到 SRAM，并将注意力计算的输出写回 HBM。而右图是加速 PyTorch 在 GPT-2 上的实现，Flash Attention 不读写 HBM 的 $N \times N$ 注意力矩阵，使得注意力的计算速度大幅提高。

传统的因果自注意力机制通常需要在大内存中对所有输入数据进行操作，而 Flash Attention 通过在硬件层面优化内存访问模式，实现了更高效的计算。此技术相比传统方法可实现 2 到 4 倍的速度提升，并在大规模训练任务中展现出优异的性能。通过这种改进，本文能够在保证模型精度的前提下，显著缩短训练时间，并提升计算资源的使用效率。

4.3.3 多 GPU 并行训练

最后，为了充分利用硬件资源，本文应用了多 GPU 并行训练（DistributedDataParallel (DDP)）[12]。DDP 通过将模型和数据分配到多个 GPU 上，每个 GPU 负责处理部分数据和模型参数，从而实现计算任务的并行化。在训练过程中，每个 GPU 独立进行前向计算、反向传播和梯度计算，并通过集体通信操作（如 NCCL [18]）同步各 GPU 的梯度和模型参数，从而保证了多 GPU 之间的协同更新，如图 7 所示。

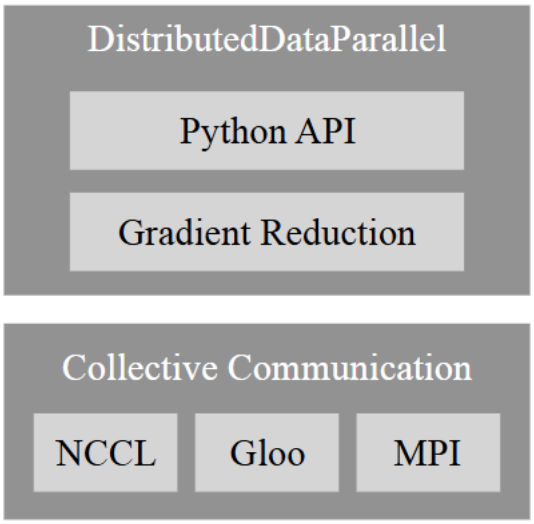

图 7. DistributedDataParallel 的搭建模块

相较于传统的 DataParallel 方法，DDP 采用了更加高效的并行策略。DataParallel 通常依赖于单个进程来处理多 GPU 任务，这可能导致 GPU 之间的通信成为瓶颈。而 DDP 则将每个 GPU 独立运行，并通过高效的分布式同步机制实现梯度的快速交换，减少了通信延迟和资源竞争。因此，DDP 在大规模分布式训练中的表现要优于 DataParallel，尤其在高并发训练任务中，DDP 能够显著提高训练效率，减少训练时间。 在使用 DDP 之前，需要初始化 DDP 环境，设置通信后端和进程组，并选择合适的通信后端来优化 GPU 间的通信 。确保数据在多个 GPU 间均匀分配。这样可以避免单个 GPU 处理过多数据而导致性能瓶颈 。在训练过程中，每个 GPU 独立地进行前向计算和反向传播，然后通过集体通信同步梯度，最后更新模型参数。这确保了训练过程的并行性和高效性。

5 实验结果分析

5.1 实验环境与数据集

本文使用两张 NVIDIA RTX 3090 24GB GPUs，并使用了 Tiny Shakespeare 数据集 [11] 进行模型训练。Tiny Shakespeare 数据集由莎士比亚的戏剧作品中的选定文本组成，数据集总大小约为 1MB，包含大约 10,000 个字符。数据集中包括了莎士比亚的经典戏剧如《哈姆雷特》、《麦克白》和《罗密欧与朱丽叶》等内容，因此在训练过程中，模型需要学习如何在字符级别上捕捉语言规律和上下文依赖关系。该数据集非常适合用来验证和调试基于字符生成的 NLP 模型，特别是在资源有限的情况下。

首先，基于复现的模型结构，本文加载了 GPT-2 的预训练权重进行测试，结果如图 8 所示。可以发现，预训练的 GPT-2 模型能够生成连贯且合理的文本，验证了本文复现 GPT 模型结构的正确性，为后续的训练和应用提供了可靠的基础。

```
decoded output:
> Hello, I'm a language model, not a science. I'm a language designer. I want to write, I want to think. I want
> Hello, I'm a language model, I use an English sentence structure, I like words over sentences.
```

图 8. 预训练模型生成的文本

之后，本文将模型权重随机初始化，并加载 Tiny Shakespeare 数据集进行训练，模型训练的损失下降曲线如图 9 所示。从图中可以看出，模型的损失值随着训练迭代次数的增加稳定下降，这表明复现的模型结构能够成功学习到数据特征，进一步验证了本文复现的模型的正确性。

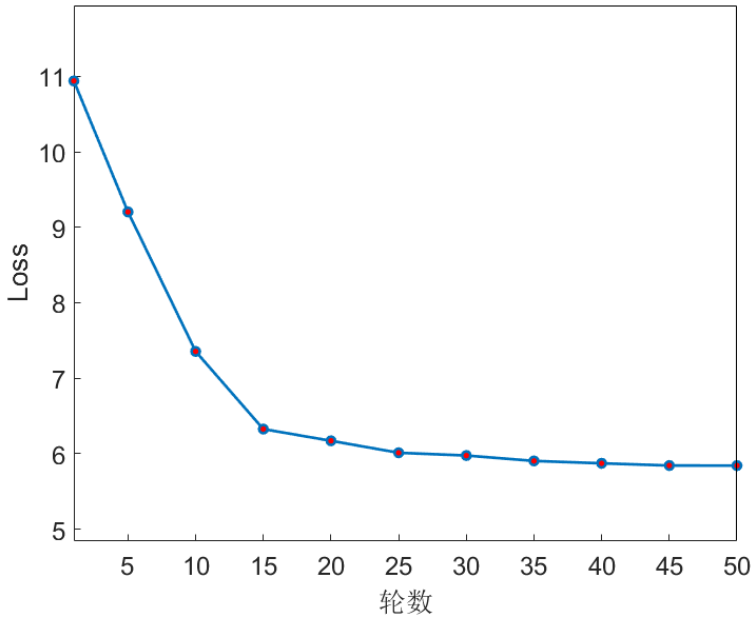

图 9. 模型训练的损失下降曲线

5.2 混合精度训练

在应用混合精度训练后，模型的训练速度从 18911 tokens/s 提升到了 27044 tokens/s，提升了 43.01%，如图 10 所示。混合精度训练通过使用半精度浮点数（FP16）来加速计算，同时保持模型训练的稳定性与精度。由于在计算和内存使用上的优化，训练过程中的硬件资源得到了更高效的利用，从而显著提高了整体训练速度。

5.3 注意力机制的改进

在采用了 Flash Attention 优化后的注意力机制后，训练速度从 27044 tokens/s 提升到 55382 tokens/s，提升了 104.78%，如图 10 所示。Flash Attention 通过减少内存带宽的使用，优化了数据访问模式，显著提高了计算效率。该改进使得自注意力机制的计算过程变得更加高效，为大规模训练任务提供了强大的支持。

5.4 多 GPU 并行训练

在引入多 GPU 并行训练（DistributedDataParallel）后，训练速度从 55382 tokens/s 提升到 105219 tokens/s，进一步提升了 89.99%，如图 10 所示。多 GPU 训练通过将模型和数据分布到多个 GPU 上，能够显著提升训练效率。在每个 GPU 上独立进行前向传播和反向传播，并通过高效的梯度同步机制，确保了模型参数的一致更新，从而大幅提升了整体训练速度。

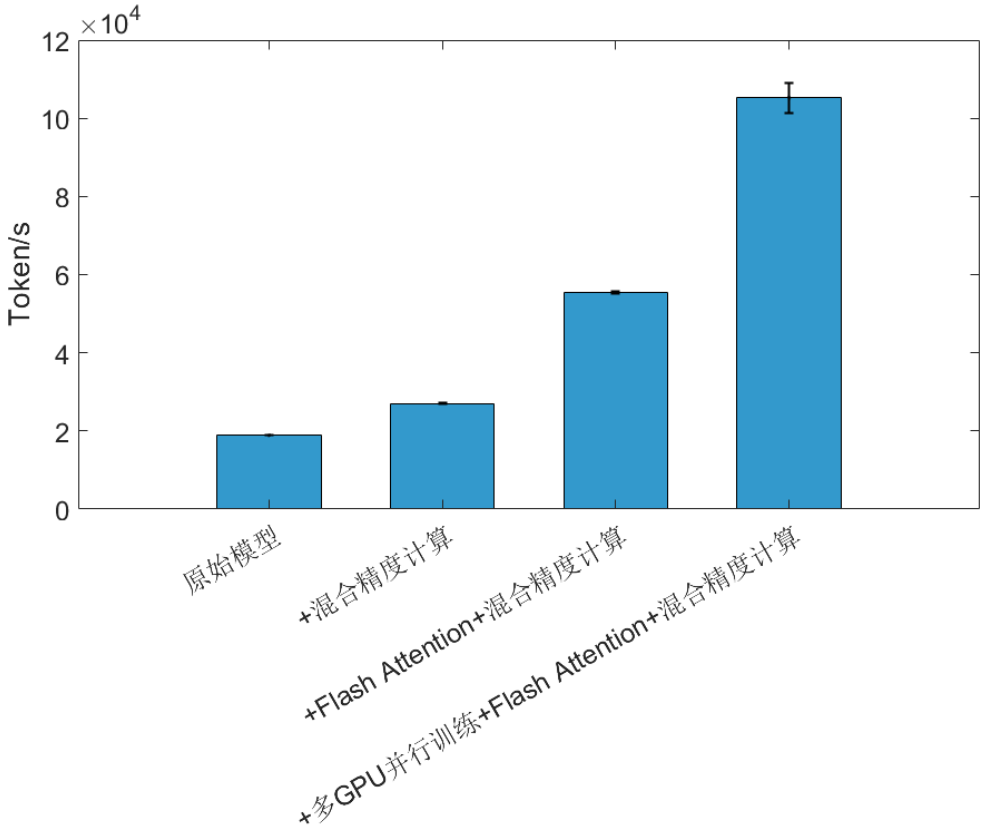

图 10. 模型训练速度对比

在加入上述创新后，总体的训练速度提升了 456.39%，而模型训练的损失下降曲线如图 11 所示。可以观察到，模型在训练过程中依然能够稳定地进行梯度下降，且损失下降趋势平滑，几乎不受上述创新方法的负面影响。这表明，混合精度训练、改进的注意力机制和多 GPU 并行训练等创新方法在训练过程中不会对模型的学习产生不良影响，反而能够有效提高训练效率。因此，可以得出结论，所采用的创新方法对于提升训练速度和优化硬件资源利用率具有积极效果，且在保证模型训练稳定性和精度的同时，能够在较短时间内实现较好的收敛表现。

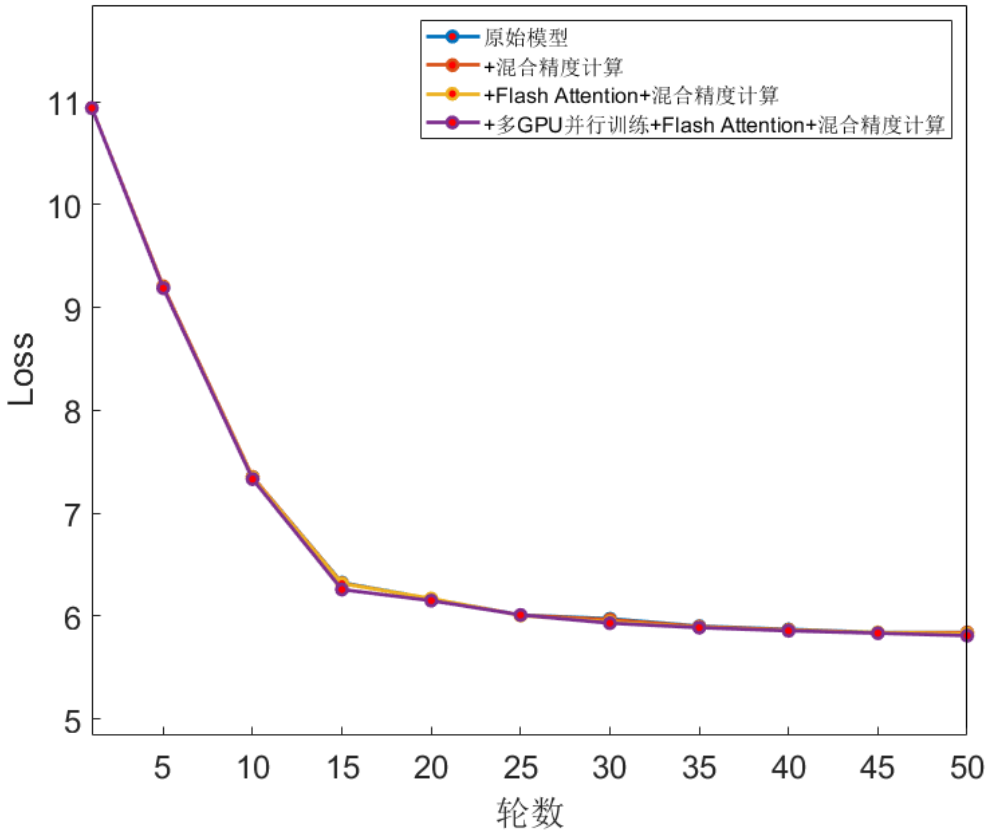

图 11. 改进后模型训练的损失下降曲线

6 总结与展望

本文对 GPT-3 模型进行了复现，并选取了合适参数规模的模型进行训练。针对训练速度，本文通过混合精度训练、改进的注意力机制以及多 GPU 并行训练等技术手段进行了优化。实验结果表明，通过这些改进，训练速度得到了显著提升，尤其是在注意力机制优化和多 GPU 并行训练的结合下，训练速度实现了大幅度提高。

未来的研究将聚焦于在不显著增加模型规模的前提下进一步提升模型的表现。本文计划通过探索更加高效的训练算法、优化内存管理、减少冗余计算等方式，持续提高模型的训练效率与推理速度。同时，针对更大规模的多 GPU 训练集群，将进一步优化分布式训练策略，最大化硬件资源的利用，确保能够在更大规模的任务中保持高效性和稳定性。

参考文献

[1] Jimmy Lei Ba. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[2] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot

learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901, 2020.

[3] Andrew M Dai and Quoc V Le. Semi-supervised sequence learning. *Advances in neural information processing systems*, 28, 2015.

[4] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Proc. Conf. on Neural Information Processing Systems*, 35:16344–16359, 2022.

[5] Jacob Devlin. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proc. IEEE/CVF Conf. on Computer Vision & Pattern Recognition*, pages 770–778, 2016.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *Proc. Euro. Conf. on Computer Vision*, pages 630–645. Springer, 2016.

[8] Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification. *arXiv preprint arXiv:1801.06146*, 2018.

[9] Huggingface. https://github.com/huggingface/transformers/blob/main/src/transformers/models/gpt2/modeling_gpt2.py, 2024.

[10] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

[11] Karpathy. https://github.com/karpathy/char-rnn/tree/master/data/tinyshakespeare, 2015.

[12] Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, et al. Pytorch distributed: Experiences on accelerating data parallel training. *arXiv preprint arXiv:2006.15704*, 2020.

[13] Yinhan Liu. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 364, 2019.

[14] I Loshchilov. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[15] Bryan McCann, James Bradbury, Caiming Xiong, and Richard Socher. Learned in translation: Contextualized word vectors. *Advances in neural information processing systems*, 30, 2017.

[16] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, et al. Mixed precision training. *arXiv preprint arXiv:1710.03740*, 2017.

[17] Tomas Mikolov. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 3781, 2013.

[18] NVIDIA Collective Communications Library (NCCL). https://developer.nvidia.com/nccl, 2019.

[19] Toan Q Nguyen and Julian Salazar. Transformers without tears: Improving the normalization of self-attention. *arXiv preprint arXiv:1910.05895*, 2019.

[20] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[21] Matthew E Peters, Mark Neumann, Luke Zettlemoyer, and Wen-tau Yih. Dissecting contextual word embeddings: Architecture and representation. *arXiv preprint arXiv:1808.08949*, 2018.

[22] Alec Radford. Improving language understanding by generative pre-training. 2018.

[23] 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.

[24] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67, 2020.

[25] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

[26] A Vaswani. Attention is all you need. *Proc. Conf. on Neural Information Processing Systems*, 2017.

[27] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Perric Cistac, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-Art Natural Language Processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45. Association for Computational Linguistics, October 2020.

[28] Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tieyan Liu. On layer normalization in the transformer architecture. In *International Conference on Machine Learning*, pages 10524–10533. PMLR, 2020.