PEFT（Parameter-Efficient Fine-Tuning ）

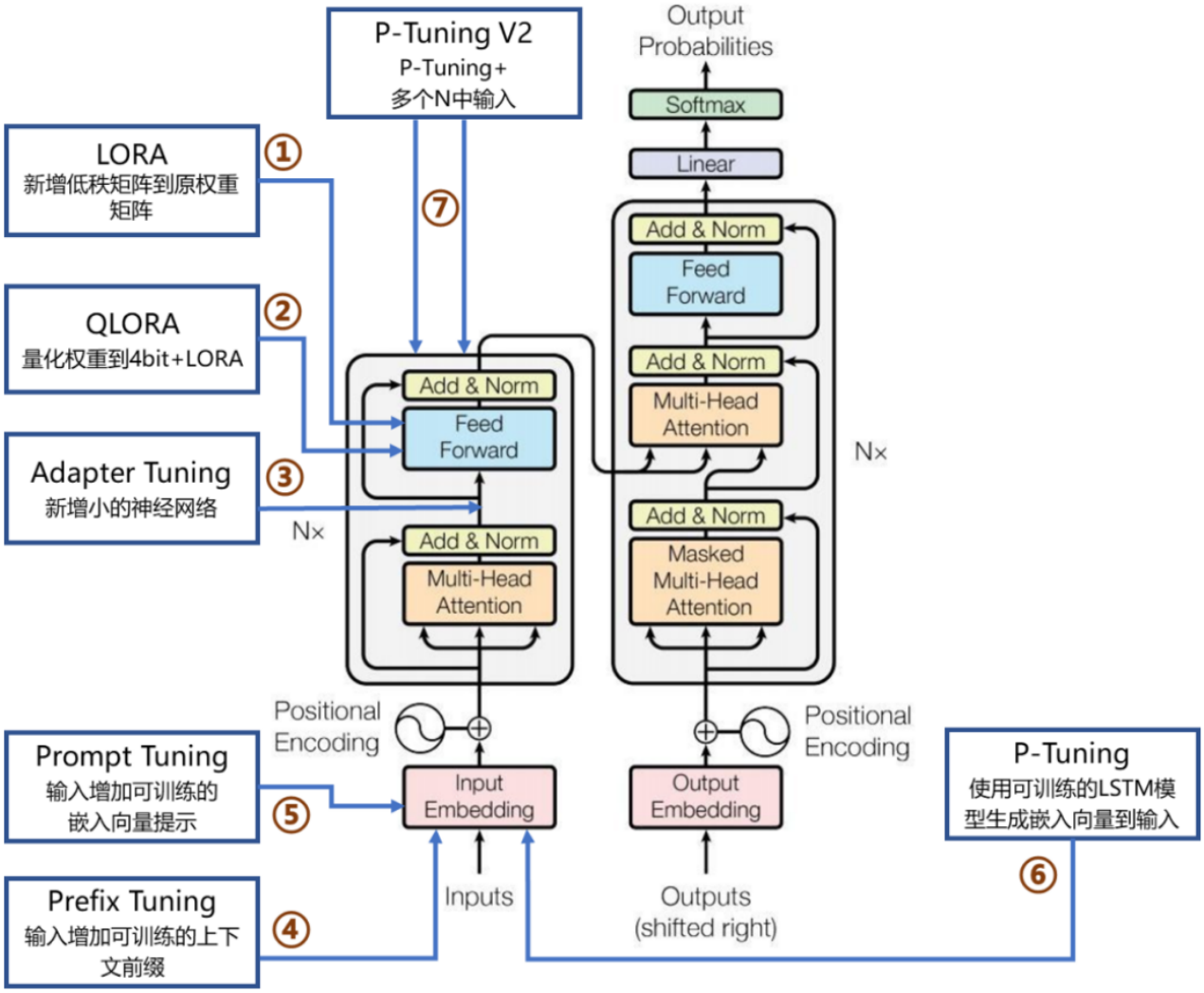
大模型全量微调（Fine-tuning）通过在预训练的大型模型基础上调整所有层和参数，使其适应特定任务。这一过程使用较小的学习率和特定任务的数据进行，可以充分利用预训练模型的通用特征，但可能需要更多的计算资源。

PEFT（Parameter-Efficient Fine-Tuning ）技术旨在通过最小化微调参数的数量和计算复杂度，来提高预训练模型在新任务上的性能，从而缓解大型预训练模型的训练成本。

这样一来，即使计算资源受限，也可以利用预训练模型的知识来迅速适应新任务，实现高效的迁移学习。因此，PEFT技术可以在提高模型效果的同时，大大缩短模型训练时间和计算成本，让更多人能够参与到深度学习研究中来。

今天学习LORA、QLoRA、Adapter Tuning、Prefix Tuning、Prompt Tuning、P-Tuning及P-Tuning v2。

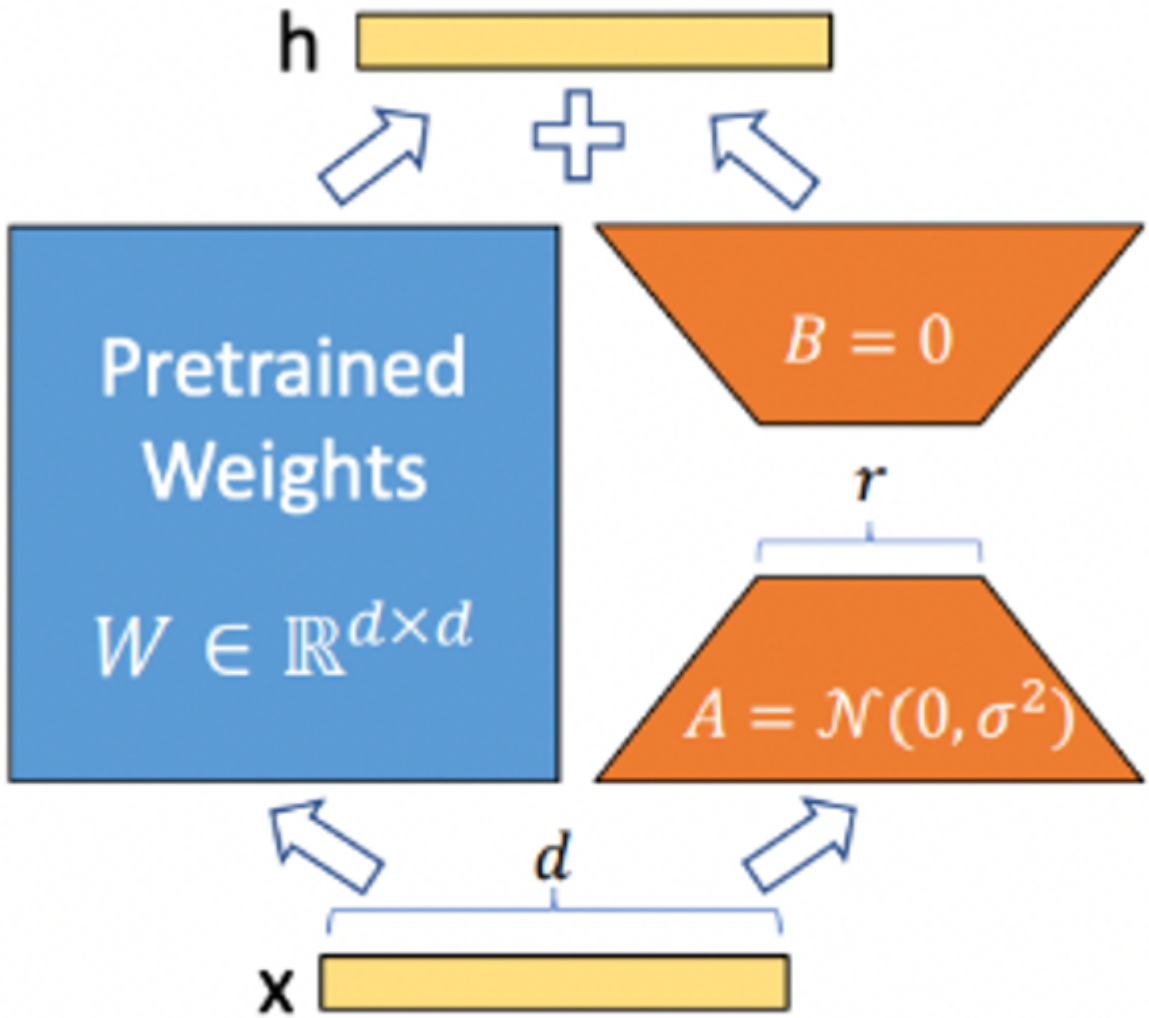
下图示例了这**7个主流PEFT方法**在Transformer网络架构的作用位置和简要说明：



来源：大鱼的数据人生

1、lora

论文：**LoRA: Low-Rank Adaptation of Large Language Models**



LoRA是一种用于微调大型预训练语言模型（如GPT-3或BERT）的方法。它的核心思想是在模型的关键层中添加小型、低秩的矩阵来调整模型的行为，而不是直接改变整个模型的结构。

这样做的好处是，可以在不增加太多额外计算负担的情况下，有效调整模型，同时保持其原有的性能。

工作原理如下：

选择要调整的权重矩阵：在大型模型（如GPT）中，我们首先确定要微调的权重矩阵。通常，这些矩阵位于模型的多头自注意力（Multi-head Self-Attention）和前馈神经网络（Feed-Forward Neural Network）部分。（例如llama2选择的就是W_q、W_V注入lora）

引入两个低秩矩阵：接着，我们引入两个低秩矩阵，记为A和B，这两个矩阵的维度比原始权重矩阵小得多，例如，如果原始矩阵的尺寸是d * d，那么，A和B的尺寸可能是d * r和r * d。其中r是一个远小于d的数。其中A会做随机高斯初始化，B则是初始化为0。

计算低秩更新：通过计算这两个低秩矩阵的乘积，生成一个新的矩阵AB，这个新矩阵的秩（即r）远小于原始权重矩阵的秩。这个乘积实际上是一个低秩近似，可以视为对原始权重矩阵的一种调整。

结合原始权重：最后，这个新生成的低秩矩阵AB被加到原始的权重矩阵上。这样，原始的权重矩阵得到了微调，但大部分权重保持不变。这个过程可以用数学公式表示为：新权重 = 原始权重 + AB。

2、Qlora

论文：**QLORA: Efficient Finetuning of Quantized LLMs**

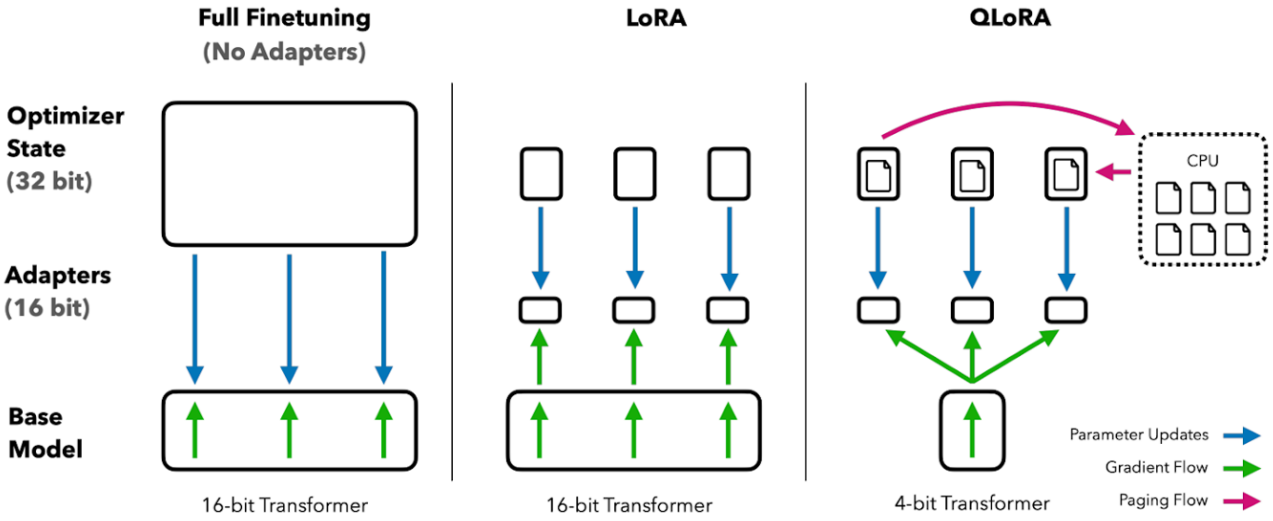


Figure 1: Different finetuning methods and their memory requirements. QLoRA improves over LoRA by quantizing the transformer model to 4-bit precision and using paged optimizers to handle memory spikes.

QLoRA（Quantized Low-Rank Adaptation）是一种高效的模型微调方法，它在LoRA（Low-Rank Adaptation）的基础上引入了深度量化过程。QLoRA的核心特点包括：

- **量化技术**：QLoRA使用一种新颖的高精度技术将预训练模型量化为4-bit。这种技术包括一种低精度存储数据类型（4-bit NormalFloat，简写为NF4）和一种计算数据类型（16-bit BrainFloat）。这样做可以在保持整个模型精度损失极小的同时，减少存储需求。
- **双重量化**：将量化常数进行量化的方法，平均每个参数节省约0.37位（对于65B模型约3GB）。
- **Paged Optimizers**：使用NVIDIA统一内存来避免在处理长序列的小批次时出现梯度变量的内存峰值。

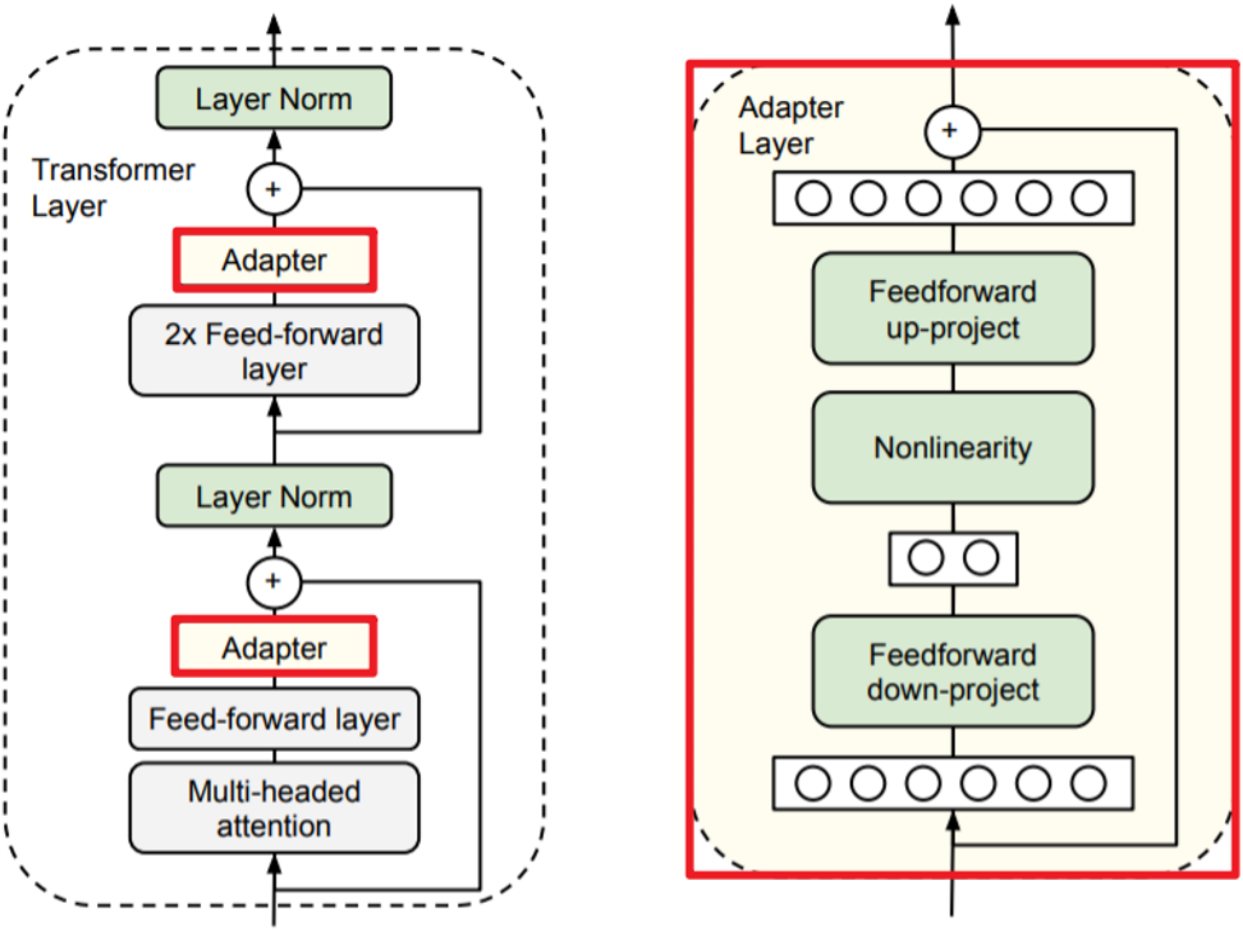
微调过程：

在训练过程中，QLoRA首先将模型用4-bit加载，然后在训练时把数值反量化到bf16后进行训练。这样的设计使得训练所需的显存大大减少。例如，一个33B的LLaMA模型可以在24 GB的显卡上进行训练。

由于量化显著减少了模型的精确度，这通常会带来性能上的损失。然而，对于大型模型，这种方法可以大幅减少内存和计算需求，使得在资源有限的环境下部署和训练成为可能。

量化过程中的关键挑战是如何设计映射和量化策略，以尽量减少因精度损失带来的性能下降。

3、Adapter Tuning



与 LoRA 类似，Adapter Tuning 的目标是在不改变预训练模型的原始参数的前提下，使模型能够适应新的任务。

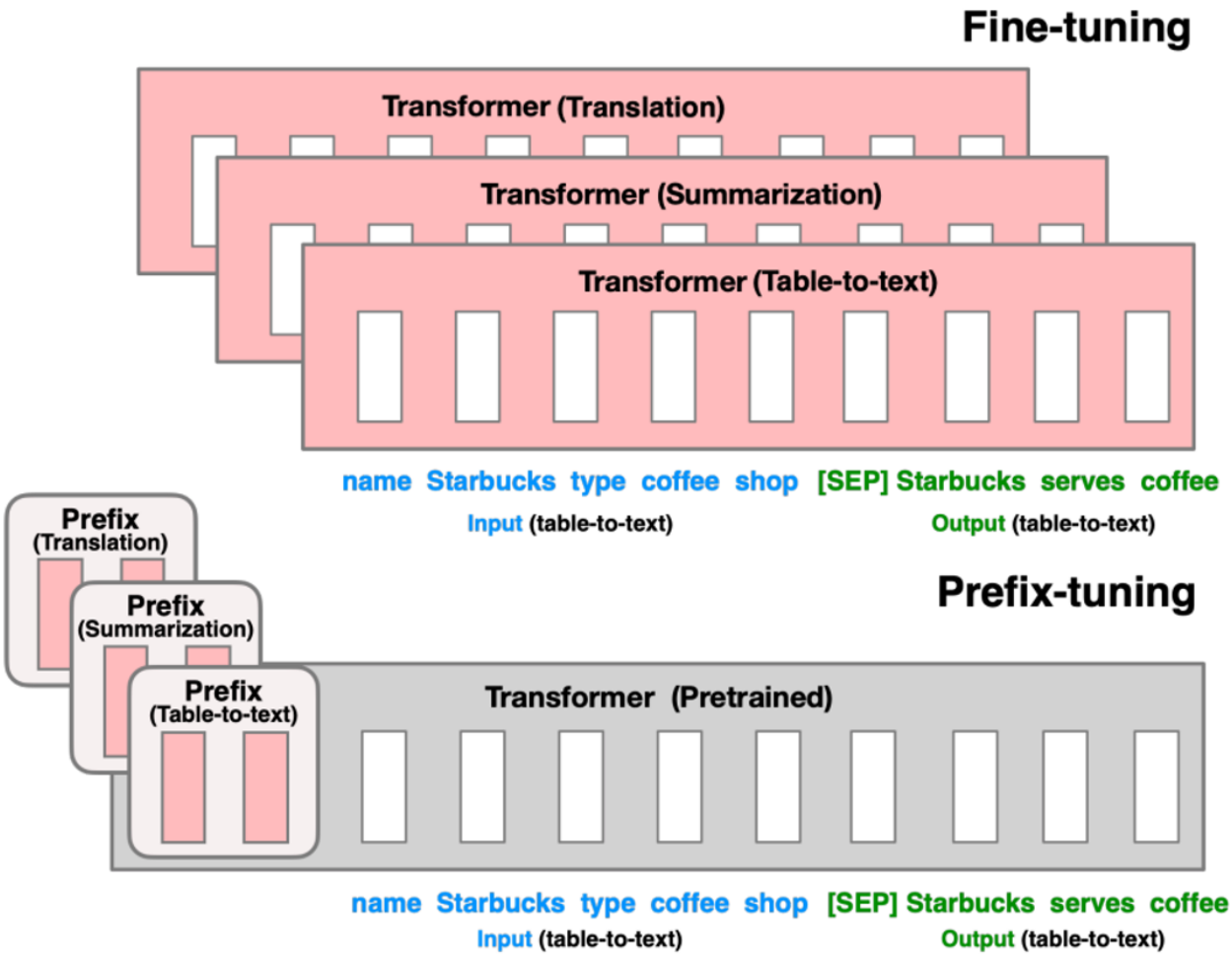
在 Adapter Tuning 中，会在模型的每个层或某些特定层之间插入小的神经网络模块，称为“**adapters**”。这些 adapters 是可以训练的，而原始模型的参数则保持不变。

4、Prefix Tuning

论文：**Prefix-Tuning: Optimizing Continuous Prompts for Generation**

Prefix Tuning提出固定预训练LM，为LM添加可训练，任务特定的前缀，这样就可以为不同任务保存不同的前缀，微调成本也小。

同时，这种Prefix实际就是连续可微的Virtual Token（Soft Prompt/Continuous Prompt），相比离散的Token，更好优化，效果更好。



这张图展示了两种不同的模型微调方法：Fine-tuning 和 Prefix-tuning。这两种方法都是为了调整预训练的Transformer模型以适应特定任务。

在 Fine-tuning 部分，图展示了三个不同任务的Transformer模型，分别用来做翻译、总结或将表格转换为文本（table-to-text）。

每个任务都有自己的微调模型，这意味着模型的所有权重都在微调过程中针对特定任务进行了更新。这种方法通常需要大量的数据和计算资源，因为整个模型都在学习任务特定的知识。

在 Prefix-tuning 部分，图展示了一种不同的微调策略，对于每个任务，都有一个特定的前缀被添加到输入序列的开始部分。这些前缀相当于任务特定的提示，可以是一组固定的词或是可训练的嵌入向量。

Prefix-tuning 的优势在于它不需要调整模型的全部权重，而是通过在输入中添加前缀来调整模型的行为，这样可以节省大量的计算资源，同时使得一个单一的模型能够适应多种不同的任务。前缀可以是固定的（即手动设计的静态提示）或可训练的（即模型在训练过程中学习的动态提示）。

5、Prompt Tuning

Prompt Tuning是一种微调方法，它在预训练语言模型的输入中添加可学习的嵌入向量作为提示。这些提示被设计成在训练过程中更新，以引导模型输出对特定任务更有用的响应。

Prompt Tuning和Prefix Tuning都涉及在输入数据中加入可学习的向量，这些元素是在输入层添加的，**但两者的策略和目的是不一样的：**

Prompt Tuning：可学习向量（通常称为prompt tokens）旨在模仿自然语言提示的形式，它们被设计为引导模型针对特定任务生成特定类型的输出。这些向量通常被看作是任务指导信息的一部分，倾向于用更少量的向量模仿传统的自然语言提示。

Prefix Tuning：可学习前缀则更多地用于提供输入数据的直接上下文信息，这些前缀作为模型内部表示的一部分，可以影响整个模型的行为。

下面的训练例子说明了两者的区别：

Prompt Tuning示例：

输入序列: [Prompt1] [Prompt2] "这部电影令人振奋。"

问题: 评价这部电影的情感倾向。

答案: 模型需要预测情感倾向（例如“积极”）

提示: 无明确的外部提示，[Prompt1] [Prompt2]充当引导模型的内部提示，因为这里的问题是隐含的，即判断文本中表达的情感倾向。

Prefix Tuning 示例：

输入序列: [Prefix1] [Prefix2] [Prefix3] "I want to watch a movie."

问题: 根据前缀生成后续的自然语言文本。

答案: 模型生成的文本，如“that is exciting and fun.”

提示: 前缀本身提供上下文信息，没有单独的外部提示

6、P-Tuning

P-Tuning（Prompt-based Tuning）和Prompt Tuning都是调整大型预训练语言模型（如GPT系列）以适应特定任务的技术。

两者都旨在利用预训练的语言模型来执行特定的下游任务，如文本分类、情感分析等。它们都使用某种形式的“提示”或“指导”来引导模型的输出，以更好地适应特定任务。

Prompt Tuning与P-Tuning的区别主要在于：

Prompt Tuning：使用静态的、可训练的虚拟标记嵌入。这些嵌入在初始化后保持固定，除非在训练过程中被更新，相对简单，因为它只涉及调整一组固定的嵌入参数。在处理多种任务时表现良好，但可能在处理特别复杂或需要细粒度控制的任务时受限。

P-Tuning：使用一个可训练的LSTM模型（称为prompt_encoder）来动态生成虚拟标记嵌入，允许根据输入数据的不同生成不同的嵌入，提供了更高的灵活性和适应性，适合需要精细控制和理解复杂上下文的任务，相对复杂，因为它涉及一个额外的LSTM模型来生成虚拟标记嵌入。

在P-Tuning中使用LSTM（长短期记忆网络）作为生成虚拟标记嵌入的工具，充分利用了LSTM的优势，包括：

更好的适应性和灵活性：由于LSTM可以捕捉输入数据中的时间序列特征，它能够更好地理解和适应复杂的、顺序依赖的任务，如文本生成或序列标注。

改进的上下文理解：LSTM由于其循环结构，擅长处理和理解长期依赖关系和复杂的上下文信息。

参数共享和泛化能力：在P-Tuning中，LSTM模型的参数可以在多个任务之间共享，这可以提高模型的泛化能力，并减少针对每个单独任务的训练需求。而在Prompt Tuning中，每个任务通常都有其独立的虚拟标记嵌入，这可能限制了跨任务泛化的能力。

这些特点使得LSTM特别适合于处理复杂任务和需要细粒度控制的应用场景。然而，这些优势也伴随着更高的计算复杂度和资源需求，因此在实际应用中需要根据具体需求和资源限制来权衡使用LSTM的决策。

7、P-Tuning v2

论文：**P-Tuning v2: Prompt Tuning Can Be Comparable to Fine-tuning Universally Across Scales and Tasks**

在P-Tuning中，连续提示被插入到输入序列的embedding里，除了语言模型的输入成之外，其他层的prompt embddding都来自于上一层。这样的设计存在两个问题：

第一、约束了要优化的参数量。由于模型的input text的长度是一定的，一般是512，那么prompt的长度就不能过于长。

第二、当模型层数很深时，tuning时模型的稳定性难以保证；模型层数越深，在第一层输入的prompt对后面的影响是难以预估的，这会影响模型的稳定性。

P-Tuning v2的改进在于，将只在第一层插入连续提示修改为在许多层都插入连续提示，而不仅仅是输入层，层与层之间的连续提示是相互独立的。这样一来，在模型tuning时，可训练的参数就增多了，P-Tuning v2在应对复杂的NLU任务和小型模型方面，相比原始P-Tuning具有更出色的效能。

方法	**对virtual token的处理**	**参与微调的参数**	**适配的下游任务**
P-tuning	MLP+LSTM或MLP	仅Embedding层中virtual token部分	使LLM适配NLU任务
Prefix tuning	MLP	prefix encoder，通过kv cache代入每一层运算	主要用于NLG任务
P-tuning v2	\	同上	NLG/NLU任务

总结

选择哪种微调方法取决于多个因素，包括任务的复杂性、可用的数据量、计算资源和期望的性能。

例如，对于需要细粒度控制的复杂任务，P-Tuning v2或LSTM基础的P-Tuning可能更适合。而对于计算资源有限的情况，可以选择LoRA或Adapter Tuning等方法。