4 参数高效微调

在现代人工智能的蓬勃发展中，大语言模型以其卓越的性能和广泛的应用场景，成为了众多领域的研究热点。尽管这些模型在大规模数据上的预训练使其具备了丰富的世界知识和多任务能力，但它们在特定领域或任务中的表现仍存在显著的局限性。为了更好地适应新的领域和处理新的任务，通常需要进行下游任务适配。然而，传统的下游任务适配方法通常存在性能不足或计算成本过高等问题。例如，传统的微调方法通常更新模型中的所有参数，需要大量的计算资源和存储空间。因此，在资源受限的环境中，全量微调大语言模型变得不切实际，因此，需要开发参数高效微调技术来克服这些挑战。本章将深入探讨当前主流的参数高效微调技术，首先将简要介绍参数高效微调的概念和分类学，然后将详细介绍参数高效微调的三类主要方法，包括参数附加方法、参数选择方法和低秩适配方法，探讨它们具体的技术实现和优势。最后，本章还将通过具体案例展示参数高效微调在垂直领域的实际应用。

4.1 参数高效微调简介

在面对新领域或任务时，大语言模型需要进行下游任务适配以更好地处理新任务。本节将介绍参数高效微调技术，该技术可以实现在较低成本下完成下游任务适配。首先，我们将回顾下游任务适配的主流范式，包括上下文学习和指令微调，分析它们的优缺点以及在实际应用中的局限性。接着，我们将详细介绍参数高效微调的概念及其重要性，阐述其在降低成本和提高效率方面的显著优势。随后，我们将对主流的 PEFT 方法进行分类，包括参数附加方法、参数选择方法和低秩适配方法，介绍每种方法的基本原理和代表性工作。

4.1.1 下游任务适配

通常，大语言模型通过在大规模数据集上进行预训练，能够积累丰富的世界知识，并获得处理多任务的能力 [42]。但由于开源大语言模型训练数据有限，因此仍存在知识边界，导致其在垂直领域（如医学、金融、法学等）上的知识不足，进而影响在垂直领域的性能表现。因此，需要进行下游任务适配才能进一步提高其在垂直和细分领域上的性能。主流的下游任务适配方法有两种：a) **上下文学习（In-context learning）** [8]；b）**指令微调（Instruction Tuning）** [52]。

1. 上下文学习

在之前的内容中，我们已经介绍过上下文学习的相关内容。它的核心思想是将不同类型的任务都转化为生成任务，通过设计 Prompt 来驱动大语言模型完成这些下游任务。小样本上下文学习（Few-shot in-context learning）将数据集中的样本-标签对转化为自然语言指令（Instruction）和样例（Demonstrations），并拼接上需要测试的样本一同输入给大语言模型，将模型输出作为最终预测结果。上下文学习完全不需要对模型进行参数更新，因此能快速将单个模型应用到多种不同的任务上。

尽管在实际应用中，**上下文学习能有效利用大语言模型的能力，但它缺点也很明显**：1）上下文学习的性能和微调依旧存在差距，并且提示词设计需要花费大量的人力成本，不同提示词的最终任务性能有较大差异；2）上下文学习虽然完全不需要训练，但在推理阶段的代价也会随提示词中样例的增多快速增加。因此，微调大语言模型在许多场景和任务中依旧是必要的，尤其是在垂直领域。

2. 指令微调

指令微调（Instruction Tuning）是另一种进行下游任务适配的方法。指令微调不仅侧重于模型在特定数据集上的表现，并且通过对模型进行任务指令的学习，使其能够更好地理解和执行各种自然语言处理任务的指令。指令微调需要首先构建指令数据集，然后在指令数据集上进行监督微调。

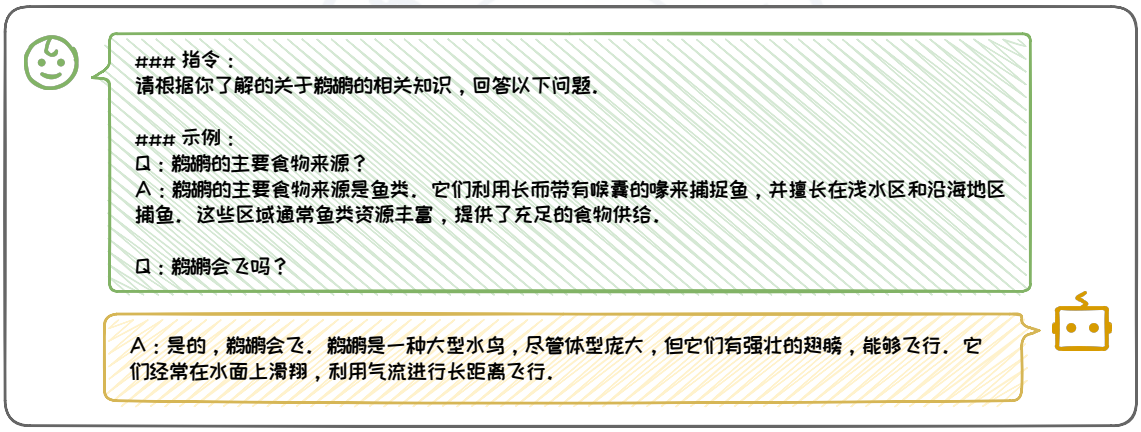

图 4.1: 指令数据样例

- **指令数据收集**：指令数据通常包含指令（任务描述）、示例（可选）以及问题和回答，如图 4.1 所示。构造这种指令数据一般有**两种方式** [52]：1）数据集成。通过使用模板将带标签的自然语言数据集，转换为指令格式的 < 输入，输出 > 对。如 Flan [46] 和 P3 [36] 数据集基于数据集成策略构建；2）大语言模型生成。通过对人工收集或少量手写指令，使用诸如 GPT-3.5-Turbo 或 GPT4 的闭源大语言模型进行指令扩展，并将收集到的指令馈送到 GPT 以获

得输出。如 InstructWild [32] 和 Self-Instruct [45] 数据集采用这种方法生成。

- **监督微调**：通过上述方法采集指令数据，组成指令数据集，以完全监督的方式对预训练模型进行微调，在给定指令和输入的情况下，通过顺序预测输出中的每个 token 来训练模型。经过微调的大型语言模型能够显著提升指令遵循（Instruction-following）能力，这有助于增强其推理水平，泛化到未见过的任务，以及适应新的领域。尽管指令微调能有效帮助大语言模型理解新领域的数据知识，提高大语言模型在下游任务上的性能。然而，监督微调需要较大的计算资源，以 LLaMA2-7B [39] 模型为例，直接进行全量微调需要近 60GB 内存，普通的消费级 GPU（如 RTX4090（24GB））无法完成模型训练。因此，为了在资源受限的环境中有效微调大语言模型，研究参数高效的微调技术显得尤为重要。

4.1.2 参数高效微调

参数高效微调（Parameter-Efficient Fine-Tuning，PEFT）旨在避免微调全部参数，减少在微调过程中需要更新的参数数量和计算开销，从而提高微调大语言模型的效率。主流的 PEFT 方法可以分为三类：参数附加方法（Additional Parameters Methods），参数选择方法（Parameter Selection Methods）以及低秩适配方法（Low-Rank Adaptation Methods）。图 4.2 给出了主流 PEFT 方法的分类学。

1. 参数附加方法

参数附加方法通过在现有模型结构中添加新的、较小的可训练模块来实现高效微调。这些模块通常称为适应层（Adapter Layer），它们被插入到现有模型的不同层之间，用于捕获特定任务的信息。通过固定原始模型参数，仅需微调新增的适应层，能够显著减少需要更新的参数量。典型的方法包括：**适配器微调（Adapter-tuning）** [18]、**提示微调（Prompt-tuning）** [23]、**前缀微调（Prefix-tuning）** [24]

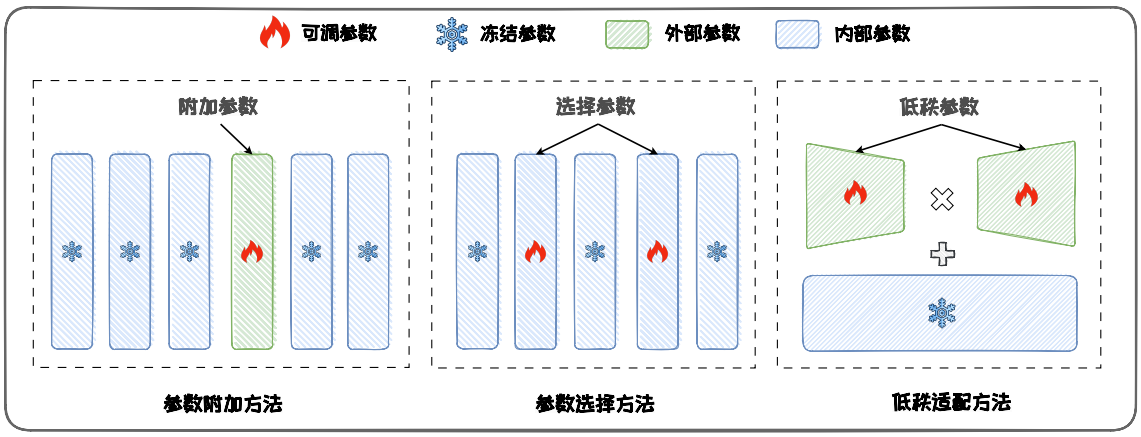

图 4.2: 高效参数微调方法分类学

和 **代理微调（Proxy-tuning）** [27] 等。参数附加方法将在 4.2 节具体介绍。

2. 参数选择方法

参数选择方法通过仅选择并微调模型的一部分参数，而冻结其余参数来实现高效微调。这种方法依赖于模型中某些参数对特定任务的高敏感性，这些参数的微调对模型性能的提升具有决定性作用。因此，选择性地微调这些关键参数，可以在降低计算负担的同时提升模型的性能。典型的方法包括：**BitFit** [49]、**Child-Tuning** [48] 以及 **FishMask** [38] 等。参数选择方法的将在 4.3 节具体介绍。

3. 低秩适配方法

低秩适配方法通过低秩矩阵分解来近似原始权重更新矩阵，并通过仅微调低秩矩阵达到全量微调的效果。由于低秩矩阵的参数远小于原始的参数更新矩阵，因此大幅节省了微调时的内存开销。LoRA [19] 是原始的低秩适配方法，后续有 **AdaLoRA** [51]、**DyLoRA** [41] 以及 **DoRA** [29] 等变体被提出，进一步改进了 LoRA 性能。低秩适配方法将在 4.4 节具体介绍。

4.1.3 参数高效微调的优势

参数高效微调的优势有以下优势：1) **计算效率**：PEFT 技术减少了需要更新的参数数量，从而降低了训练时的计算资源需求；2）**存储效率**：通过减少微调参数的数量，PEFT 显著降低了微调模型的存储空间，特别适用于内存受限的设备。3）**适应性强**：PEFT 能够快速适应不同任务，而无需重新训练整个模型，使得模型在面对变化环境时具有更高的灵活性。

下面我们将通过一个具体案例来深入探讨 PEFT 技术如何显著提升计算效率。具体来说，参考表 4.1 [1]，该表详细展示了在配备 80GB 显存的 A100 GPU 以及 64GB 以上 CPU 内存的高性能硬件环境下，进行模型全量微调与采用参数高效微调方法 LoRA（该方法将在 4.4节中详细介绍）时，GPU 内存的消耗情况对比。通过这一对比，我们可以清晰地看到 PEFT 技术在减少内存消耗方面的显著优势。

表 4.1: 全量参数微调和参数高效微调显存占用对比

模型名	全量参数微调	参数高效微调 (LoRA)
bigscience/T0_3B	47.14GB GPU / 2.96GB CPU	14.4GB GPU / 2.96GB CPU
bigscience/mt0-xxl (12B params)	OOM GPU	56GB GPU / 3GB CPU
bigscience/bloomz-7b1 (7B params)	OOM GPU	32GB GPU / 3.8GB CPU

根据该表格可以看出，对于 80GB 显存大小的 GPU，全量参数微调 7B/12B 参数的模型，会导致显存直接溢出。而在使用 LoRA 后，显存占用被大幅缩减，使得在单卡上微调大语言模型变得可行。

本节首先介绍了两类方法对大语言模型进行下游任务适配的主流范式：上下文学习和指令微调。然而，由于性能和可行性方面的限制，这两类方法难以适应大语言模型的需求。因此，需要研究参数高效微调技术。PEFT 仅对模型的一小部分

[1]表格数据来源：https://github.com/huggingface/peft

参数进行更新，在保证不牺牲性能的前提下有效地减少了模型微调时所需的参数量，从而节约了计算和存储资源。最后，我们给出了主流 PEFT 方法的分类学。在后续小节的内容中，我们将根据本节给出的分类学详细介绍主流的三类 PEFT 方法：参数附加方法 4.2、参数选择方法 4.3 以及低秩适配方法 4.4，并探讨 PEFT 的相关应用与实践 4.5。

4.2 参数附加方法

参数附加方法（Additional Parameter Methods）通过增加并训练新的辅助参数或模块来增强现有的预训练模型。参数附加方法按照附加位置可以分为三类：加在输入、加在模型以及加在输出。

4.2.1 加在输入

加在输入的方法将额外参数附加到模型的输入嵌入（Embedding）中，其中最经典的方法是 Prompt-tuning [23]。Prompt-tuning 在模型的输入中引入可微分的连续张量，通常也被称为软提示（Soft prompt）。软提示作为输入的一部分，与实际的文本数据一起被送入模型。在微调过程中，仅软提示的参数会被更新，其他参数保持不变，因此能达到参数高效微调的目的。

具体地，给定一个包含 n 个 token 的输入文本序列 $\{w_1, w_2, \ldots, w_n\}$，首先通过嵌入层将其转化为**输入嵌入**矩阵 $X \in \mathbb{R}^{n\times d}$，其中 d 是嵌入空间的维度。新加入的软提示参数被表示为**软提示嵌入**矩阵 $P \in \mathbb{R}^{m\times d}$，其中 m 是软提示长度。然后，将软提示嵌入拼接上输入嵌入矩阵，形成一个新矩阵 $[P; X] \in \mathbb{R}^{(m+n)\times d}$，最后输入 Transformer 模型。通过反向传播最大化输出概率似然进行模型训练。训练过程仅软提示参数 P 被更新。图 4.3给出了 Prompt-tuning 的示意图。

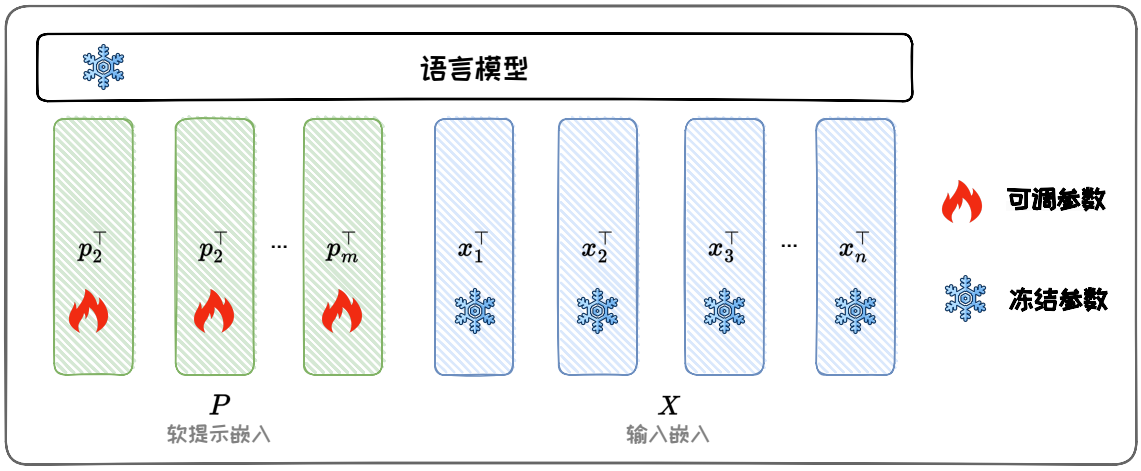

图 4.3: Prompt-tuning 示意图

在实际使用中，软提示的长度可以被设置为 1-200，通常在 20 以上就能有一定的性能保证。此外，软提示的初始化对最终的性能也会有影响，使用词表中的 token 或者在分类任务中使用类名进行初始化会优于随机初始化。值得一提的是，Prompt-tuning 原本被提出的动机不是为了实现参数高效微调，而是自动学习提示词。在第三章我们提到，利用大语言模型的常见方式是通过提示工程，也被称为是硬提示（Hard prompt），这是因为我们使用的离散 prompt 是不可微分的。问题在于大语言模型的输出质量高度依赖于提示词的构建，要找到正确的 “咒语” 使得我们的大语言模型表现最佳，需要花费大量时间。因此，采用可微分的方式，通过反向传播自动优化提示词成为一种有效的方法。

总的来说，Prompt-tuning 有以下优势：(1) **内存效率高**：Prompt-tuning 显著降低了内存需求。例如，T5-XXL 模型对于每个特定任务的模型需要 11B 参数，但经过 Prompt-tuning 的模型只需要 20480 个参数（假设提示长度为 5）；(2) **多任务能力**：可以使用单一冻结模型进行多任务适应。传统的模型微调需要为每个下游任务学习任务特定的完整预训练模型副本，并且推理必须在单独的批次中执行。Prompt-tuning 只需要为每个任务存储一个小的任务特定的提示模块，并且可以使用原始预训练模型进行混合任务推理（在每个任务提示词前加上学到的 soft prompt）。(3) **缩**

放特性：Prompt-tuning 有很好的缩放特性，微调性能会随着模型参数量的增加而增强，在 10B 参数量下的性能接近（多任务）全参数微调的性能。

4.2.2 加在模型

加在模型的方法将额外的参数或模型添加到预训练模型中，其中经典的方法有 Prefix-tuning [24]、Adapter-tuning [18] 和 AdapterFusion [34]。

1. Prefix-tuning

Prefix-tuning 和上一节介绍的 Prompt-tuning 十分类似，区别在于 Prompt-tuning 仅将软提示添加到输入，而 Prefix-tuning 将一系列连续的可训练前缀（prefixes，即 Soft-prompt）插入到输入以及 Transformer 的每一隐层单元中，大幅增加了可学习的参数量。图 4.4 给出了 Prefix-tuning 的示意图。

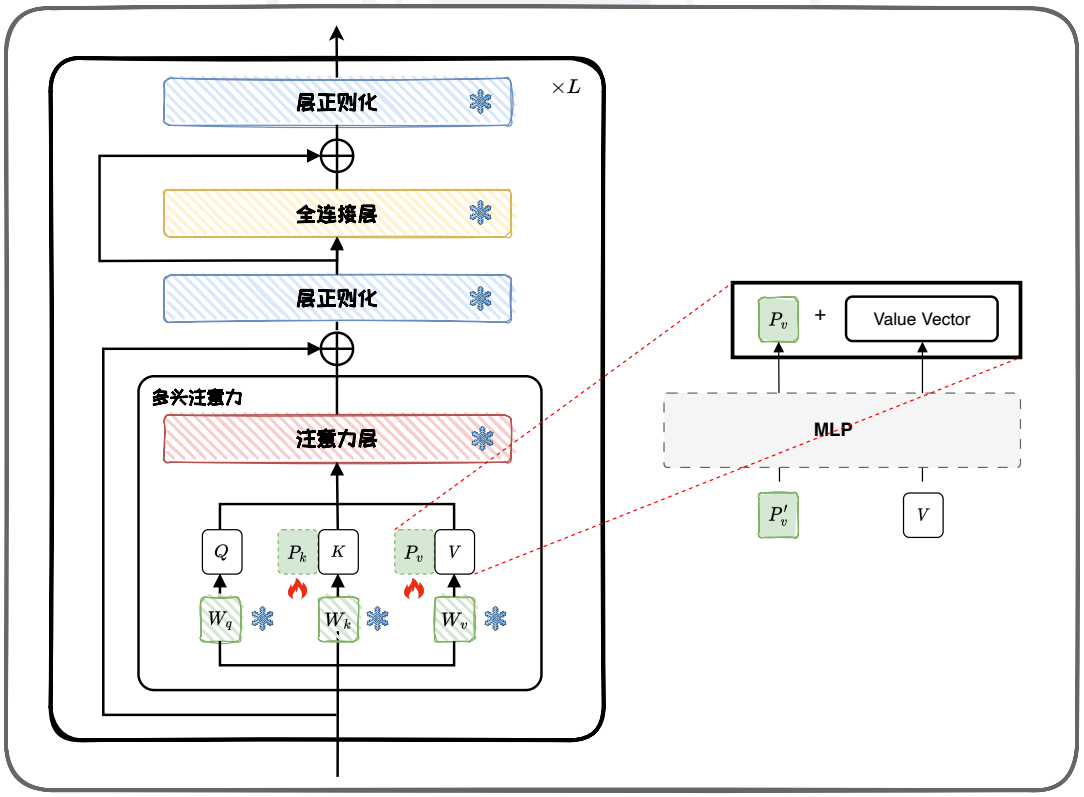

图 4.4: Prefix-tuning 示意图

具体而言，Prefix-tuning 引入了一组可学习的向量，这些向量被添加到所有 Transformer 层中的键 k 和值 v 之前。类似于 Prompt-tuning，Prefix-tuning 也会面临前缀参数更新不稳定的问题，从而导致优化过程难以收敛。因此，在实际应用中，通常需要在输入 Transformer 模型前，先通过一个多层感知机（MLP）进行重参数化。这意味着需要训练的参数包括 MLP 和前缀矩阵两部分。训练完成后，MLP 的参数会被丢弃，仅保留前缀参数。

总的来说，Prefix-tuning 具有以下优势：（1）**参数效率**：只有前缀参数在微调过程中被更新，这显著减少了需要训练的参数数量；（2）**任务适应性**：前缀参数可以针对不同的下游任务进行定制，提供了一种灵活的微调方法；（3）**保持预训练知识**：由于预训练模型的原始参数保持不变,Prefix-tuning 能够保留预训练过程中学到的知识。

2. Adapter-tuning

Adapter-tuning [18] 提出向预训练语言模型中插入新的可学习的神经网络模块，称为适配器。适配器模块通常采用瓶颈（bottomneck）结构，即引入一个下投影层，将信息压缩到一个低维的表示，然后再通过上投影层扩展回原始维度。

如图 4.5所示，Adapter-tuning 对 Transformer 的每一层新增两个适配器模块，他们分别被放置在**多头注意力层**（Multi-head Attention Layer，图中红色块）和**全连接层**（Feed-forward Network Layer，图中蓝色块）之后。**与 Transformer 的全连接层不同，由于采用了瓶颈结构，适配器的隐藏维度通常比输入小**。因此，适配器有很高的参数效率。在训练时，通过固定原始模型参数，仅对适配器、层正则化（图中绿色框）以及最后的分类层参数（图中未标注）进行微调，可以大幅缩减微调参数量和计算量，从而实现参数高效微调。

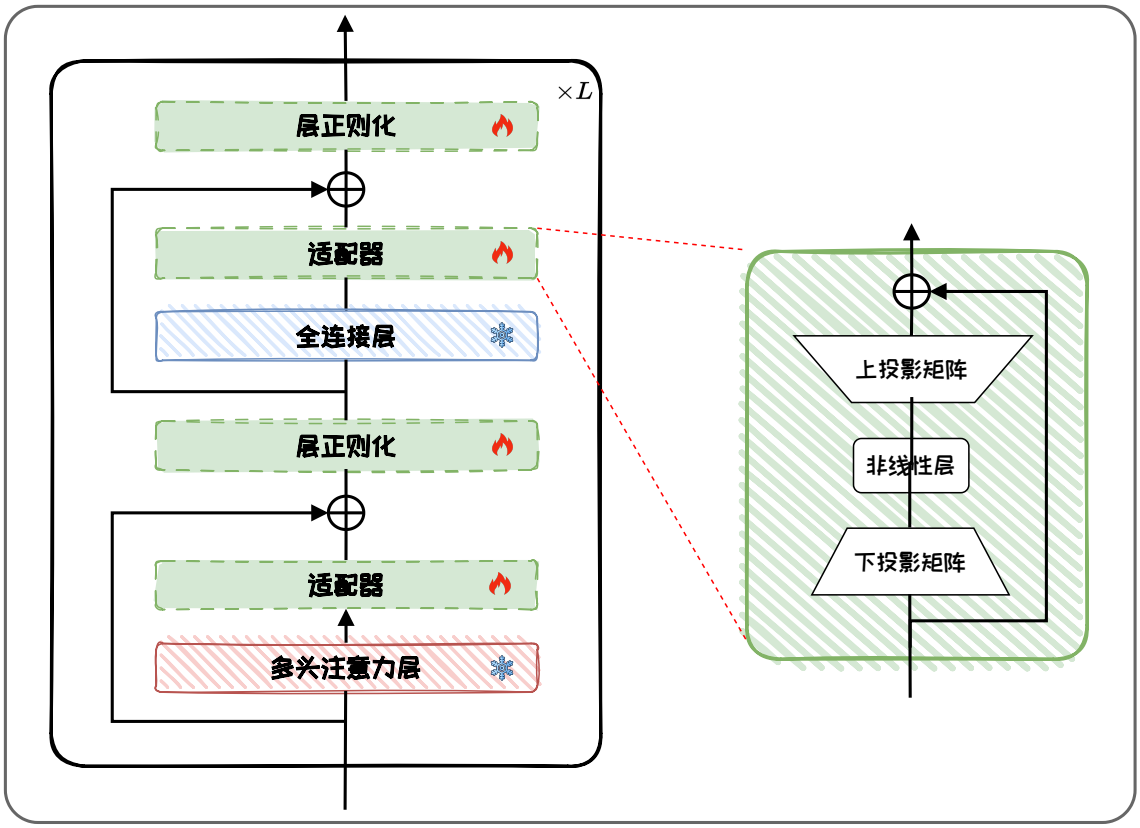

图 4.5: Adapter-tuning 示意图

适配器模块的具体结构如图 4.5 右边所示，适配器模块通常由一个下投影矩阵 $W_d \in \mathbb{R}^{d\times r}$ 和一个上投影矩阵 $W_u \in \mathbb{R}^{r\times d}$ 以及残差连接组成，其中 $r << d$:

$$A^{(l)} = \sigma(W_d * H^{(l-1)})W_u + H^{(l-1)} \tag{4.1}$$

其中，$\sigma(\cdot)$ 是激活函数，如 ReLU 或 Sigmoid。$A^{(l)}$ 是适配器的输出，$H^{(l)}$ 是第 l 层的隐藏状态。

在适配器中，下投影矩阵将输入的 d 维特征压缩到低维 r，再用上投影矩阵投影回 d 维。因此，在每一层中的总参数量为 $2dr + d + r$，其中包括投影矩阵及其偏置项参数。通过设置 $r \ll d$，可以大幅限制每个任务所需的参数量。进一步地，适配器模块的结构还可以被设计得更为复杂，例如使用多个投影层，或使用不同的激活函数和参数初始化策略。此外，Adapter-tuning 还有许多变体，例如通过调

整适配器模块的位置 [14, 55]、剪枝 [15] 等策略来减少可训练参数的数量。

3. AdapterFusion

由于 Adapter-tuning 无需更新预训练模型，而是通过适配器参数来学习单个任务，每个适配器参数保存了解决该任务所需的知识。因此，如果想要结合多个任务的知识，可以考虑将多个任务的适配器参数结合在一起。基于该思路，AdapterFusion 提出一种两阶段学习的方法，先学习多个任务进行知识提取，再“融合”(Fusion) 来自多个任务的知识。具体的两阶段步骤如下：

第一阶段：知识提取。给定 N 个任务，首先对每个任务分别训练适配器模块，用于学习特定任务的知识。该阶段有两种训练方式，分别如下：

- **Single-Task Adapters(ST-A)**：对于 N 个任务，模型都分别独立进行优化，各个任务之间互不干扰，互不影响。
- **Multi-Task Adapters(MT-A)**：N 个任务通过多任务学习的方式，进行联合优化。

第二阶段：知识组合。单个任务的适配器模块训练完成后，AdapterFusion 将不同 LoRA 模块进行融合（Fusion），以实现知识组合。融合操作涉及一个新的融合模块，该模块旨在搜索多个任务适配器模块的最优组合，实现任务泛化。在该阶段，语言模型的参数以及 N 个适配器的参数被固定，仅微调 AdapterFusion 模块的参数，并优化以下损失：

$$\Psi \leftarrow \operatorname{argmin}_{\Psi} L_n\left(D_n; \Theta, \Phi_1, \ldots, \Phi_N, \Psi\right) \tag{4.2}$$

其中，D_n 是第 n 个任务的数据，L_n 是第 n 个任务的损失函数，Θ 表示预训练模型参数，Φ_i 表示第 i 个任务的适配器模块参数，Ψ 是融合模块参数。

图 4.6 给出 AdapterFusion 的示意图。每层的 AdapterFusion 模块包括可学习的 Key（键）、Value（值）和 Query（查询）等对应的投影矩阵。全连接层的输出当作 Query，适配器的输出当作 Key 和 Value，计算注意力得到联合多个适配器的输出

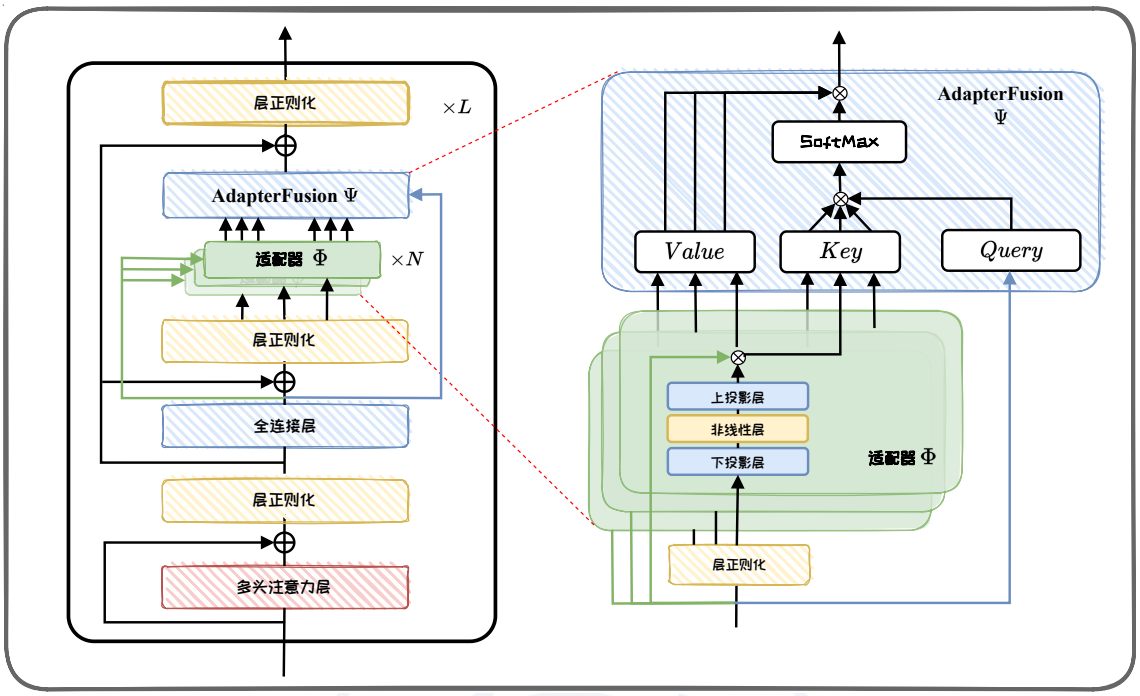

图 4.6: AdapterFusion 示意图

结果。此外，AdapterFusion 还能进一步在不同的适配器模块间参数共享，进一步减少参数数量。

本节介绍了 Adapter-tuning 和 AdapterFusion 这两种典型的适配器方法。总的来说，适配器方法的优势包括：（1）**模型参数的最小化改变**：适配器方法只训练少量的额外参数，这使得模型的微调更加高效，同时减少了对计算资源的需求；（2）**保持预训练模型的稳定性**：由于原始模型参数不变，适配器方法能够保持预训练模型的泛化能力和稳定性；（3）**任务适应性**：适配器层可以根据不同的下游任务进行定制，使得模型能够更好地适应特定的任务需求。

4.2.3 加在输出

1. Proxy-tuning

在微调大语言模型时，通常会面临以下问题：首先，大语言模型的**参数数量可能会非常庞大**，例如 LLaMA 系列最大的模型拥有 70B 参数，即使采用了 PEFT 技术，也难以在普通消费级 GPU 上完成下游任务适应；其次，用户可能**无法直接访问大语言模型的权重**（黑盒模型），这进一步增加了微调的难度。

为了应对这些问题，代理微调（Proxy-tuning）[27] 提供了一种轻量级的解码时（decoding-time）算法，允许我们在不直接修改大语言模型权重的前提下，通过仅访问模型输出词汇表预测分布，来实现对大语言模型的进一步定制化调整。

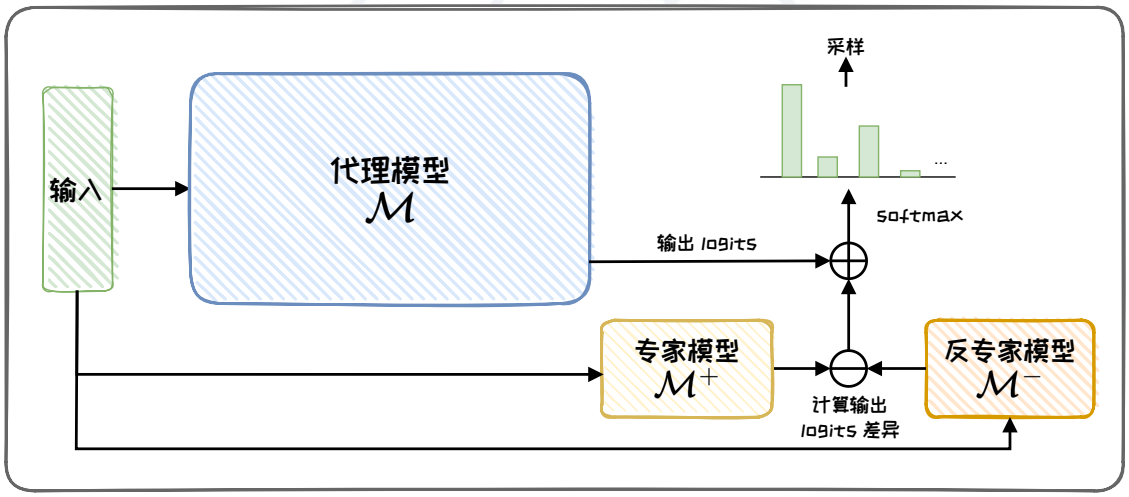

图 4.7: Proxy-tuning 示意图

如图 4.7 所示，给定待微调的**代理模型** $\mathcal{M}$ 以及较小的**反专家模型**（anti-expert model）$\mathcal{M}^-$，这两个模型仅需要相同的词汇表即可。我们对 $\mathcal{M}^-$ 进行微调，得到微调后的**专家模型**（expert model）$\mathcal{M}^+$。在每一个自回归生成的时间步中，代理微调首先计算专家模型 $\mathcal{M}^+$ 和反专家模型 $\mathcal{M}^-$ 之间的 $logits$ 分布差异，然后将其加到代理模型 $\mathcal{M}$ 下一个词预测的 $logits$ 分布中。

具体来说，在代理微调的计算阶段，针对每一时间步 t 的输入序列 $x_{<t}$，从基础

模型 $\mathcal{M}$、专家模型 $\mathcal{M}^+$ 和反专家模型 $\mathcal{M}^-$ 中获取相应的输出分数 $s_{\mathcal{M}}, s_{\mathcal{M}^+}, s_{\mathcal{M}^-}$。通过下式调整目标模型的输出分数 $\tilde{s}$:

$$\tilde{s} = s_{\mathcal{M}} + s_{\mathcal{M}^+} - s_{\mathcal{M}^-}, \tag{4.3}$$

然后，使用 $softmax(\cdot)$ 对其进行归一化，得到输出概率分布，

$$p_{\tilde{\mathcal{M}}}\left(X_t \mid x_{<t}\right) = softmax(\tilde{s}) \tag{4.4}$$

最后，在该分布中采样得到下一个词的预测结果。在实际使用中，通常专家模型是较小的模型（例如，LLaMA-7B），而代理模型则是更大的模型（例如，LLaMA-13B 或 LLaMA-70B）。通过代理微调，我们将较小模型中学习到的知识，以一种解码时约束的方式迁移到比其大得多的模型中，大幅节省了计算成本。同时，由于仅需要接触到模型的输出分布，而不需要原始的模型权重，因此该方法对于黑盒模型（如 ChatGPT/GPT-4）同样适用。

本节介绍了三种主要的参数附加方法，分别通过加在输入、加在模型以及加在输出三种方式实现。总的来说，这几类方法都是提升预训练语言模型性能的有效手段，它们各有优势。加在输入的方法通过在输入序列中添加可学习的张量，对模型本身的结构修改较小，有更好的灵活性。加在模型的方法由于保持了原始预训练模型的参数，在泛化能力上表现更佳。加在输出的方法则能够以更小的代价驱动更大参数量的黑盒模型，带来更实际的应用前景。

4.3 参数选择方法

参数选择方法（Parameter Selection Methods）通过选择预训练模型中的参数子集实现高效微调。和参数附加方法不同的是，参数选择方法无需向模型添加额外的参数，避免了在推理阶段引入额外的计算成本。通常，参数选择方法分为两类：基于规则的方法和基于学习的方法。

4.3.1 基于规则的方法

基于规则的方法根据人类专家的经验，确定哪些参数应该被更新。下面我们介绍代表性的方法 BitFit [49]。BitFit 通过仅优化神经网络中的每一层的偏置项（biases）以及任务特定的分类头来实现参数高效微调。由于偏置项在模型总参数中所占比例极小（约 0.08%-0.09%），BitFit 有极高的参数效率。尽管只微调少量参数，BitFiT 依然能在 GLUE Benchmark [43] 上与全量微调相媲美，甚至在某些任务上表现更好。此外，BitFit 方法相比全量微调允许使用更大的学习率，因此该方法整体优化过程更稳定。然而，该方法仅在小模型（如 BERT、RoBERT 等）上进行验证性能，在更大模型上的性能表现如何尚且未知。

除 BitFit 以外，还有一些其他方法通过优化特定的 Transformer 层提高参数效率。例如，Lee 等人 [22] 提出，仅对 BERT 和 RoBERTa 的最后四分之一层进行微调，便能实现完全参数微调 90% 的性能；PaFi [26] 选择具有最小绝对值的模型参数作为可训练参数。FishMask [38] 提出一种被称为 FISH（Fisher-Induced Sparse Unchanging）的掩码方法，通过该掩码选择一小部分参数。

4.3.2 基于学习的方法

基于学习的方法在模型训练过程中自动选择可训练的参数子集，例如通过掩码学习 [22] 和参数剪枝 [26] 等方法。

1. Child-Tuning

Child-Tuning [48] 通过在反向传播过程中策略性地选择子网络，只更新模型中子网络的部分参数，屏蔽非子网络的梯度，从而达到微调的效果。具体来说，假设 $\mathbf{w}_t$ 是第 t 轮迭代的参数矩阵，我们引入了一个与 $\mathbf{w}_t$ 同维度的 0-1 掩码矩阵 $\mathbf{M}_t$ ，

用于选择第 t 轮迭代的子网络 $\mathbf{C}_t$，仅更新该子网络的参数，其定义如下：

$$\mathbf{M}_t^{(i)} = \begin{cases} 1, & \text{if } \mathbf{w}_t^{(i)} \in \mathbf{C}_t \\ 0, & \text{if } \mathbf{w}_t^{(i)} \notin \mathbf{C}_t \end{cases} \tag{4.5}$$

其中，$\mathbf{M}_t^{(i)}$ 和 $\mathbf{w}_t^{(i)}$ 分别是矩阵 $\mathbf{M}_t$ 和 $\mathbf{w}_t$ 在第 t 轮迭代的第 i 个元素。此时，梯度更新公式为：

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \left(\frac{\partial \mathcal{L}(\mathbf{w}_t)}{\partial \mathbf{w}_t} \odot \mathbf{M}_t \right) \tag{4.6}$$

Child-Tuning 提供了两种生成子网络掩码 $\mathbf{M}$ 的方式，由此产生两种变体模型：Child-Tuning$_F$ 和 Child-Tuning$_D$。Child-Tuning$_F$ 是一种**任务无关**的变体，它在没有任何下游任务数据的情况下选择子网络。在每次迭代时，Child-Tuning$_F$ 从伯努利分布中抽取 0-1 掩码，生成梯度掩码 $\mathbf{M}_t$：

$$\mathbf{M}_t \sim \text{Bernoulli}(p_F) \tag{4.7}$$

其中，p_F 是伯努利分布的概率，表示子网络的比例。此外，Child-Tuning$_F$ 通过引入噪声来对全梯度进行正则化，从而防止小数据集上的过拟合，并提高泛化能力。

Child-Tuning$_D$ 是一种**任务驱动**的变体，它利用下游任务数据来选择与任务最相关的子网络。具体来说，Child-Tuning$_D$ 使用费舍尔信息矩阵（FIM）来估计特定任务相关参数的重要性。正式地，模型参数 w 的费舍尔信息矩阵定义如下：

$$F(w) = E\left[\left(\frac{\partial \log p(y|x;w)}{\partial w} \right) \left(\frac{\partial \log p(y|x;w)}{\partial w} \right)^{\top} \right] \tag{4.8}$$

其中，x 和 y 分别表示输入和输出，FIM 是对数似然梯度关于参数 w 的协方差。在实际应用中，我们使用经验费舍尔信息矩阵的对角元素来点估计与任务相关的参数重要性。具体地，对于给定的任务训练数据 $\mathcal{D}$，模型参数 w 的第 i 个分量 $w^{(i)}$ 的费舍尔信息估计为：

$$F^{(i)}(w) = \frac{1}{|\mathcal{D}|} \sum_{j=1}^{|\mathcal{D}|} \left(\frac{\partial \log p(y_j|x_j;w)}{\partial w^{(i)}} \right)^2 \tag{4.9}$$

即计算 $w^{(i)}$ 关于对数似然的梯度平方和。通常，我们假设参数对目标任务越重要，它的费舍尔信息的值就越高。因此，可以根据费舍尔信息来选择子网络，子网络 $\mathbf{C}$ 由具有最高费舍尔信息的参数组成。选择子网络参数的步骤如下：1) 计算每个参数的费舍尔信息值；2) 对这些费舍尔信息值进行排序；3) 选择前 p_D 比例的参数作为子网络 $\mathbf{C}$ 。确定子网络后，生成相应的掩码矩阵完成模型训练。

Child-Tuning 通过梯度屏蔽减少了计算负担，同时减少了模型的假设空间，降低了模型过拟合的风险。然而，子网络的选择需要额外的计算代价，特别是在任务驱动的变体中，费舍尔信息的计算十分耗时。总的来说，Child-Tuning 显著改善大规模预训练语言模型在各种下游任务中的表现，尤其是在训练数据有限的情况下。此外，Child-Tuning 可以很好地与其他 PEFT 方法的集成，进一步提升模型性能。

除 Child-Tuning 外，还有一些其他基于学习的参数选择方法。例如，Zhao 等人 [54] 引入与模型权重相关的二值矩阵来学习，通过阈值函数生成参数掩码（mask），然后在反向传播过程中通过噪声估计器进行更新。类似 FishMask，Fish-Dip [5] 也使用 Fisher 信息来计算掩码，但掩码将在每个训练周期动态重新计算。LT-SFT [2] 受 “彩票假设” [11] 启发，根据参数重要性，根据在初始微调阶段变化最大的参数子集形成掩码。SAM [12] 提出了一个二阶逼近方法，通过解析求解优化函数来帮助决定参数掩码。

基于选择的方法通过选择性地更新预训练模型的参数，在保持大部分参数不变的情况下对模型进行微调。基于选择的方法能够显著减少微调过程中所需要更新的参数，降低计算成本和内存需求。对于资源受限的环境或者需要快速适应新任务的场景尤其适用。然而，这些方法也面临挑战，比如如何选择最佳参数子集，以及如何平衡参数更新的数量和模型性能之间的关系。此外，基于选择的方法可能需要特定的技术来确保训练的稳定性和模型的泛化能力。

4.4 低秩适配方法

低维固有维度假设 [1] 表明，过参数化模型存在于低固有维度上，这表明我们可以通过仅更新与固有秩相关的参数来完成模型学习。基于这一假设，LoRA（Low-Rank Adaptation）提出使用低秩矩阵更新模型中的密集层。在本节中，我们首先将介绍 LoRA 的实现细节以及参数效率分析。接着，将介绍 LoRA 的相关变体。最后，介绍基于 LoRA 插件的任务泛化。

4.4.1 LoRA

方法实现

给定一个由参数矩阵 $W_0 \in \mathbb{R}^{d\times k}$ 定义的密集神经网络层，为了适应下游任务，我们通常需要学习参数更新矩阵 $\Delta W \in \mathbb{R}^{d\times k}$，得到更新后的参数矩阵 $W = W_0 + \Delta W$。在全量微调过程中，ΔW 是基于该层所有 $d \times k$ 个参数的梯度计算而得的，这在计算上代价高昂，并且需要大量的 GPU 内存。为了解决这一问题，LoRA 方法通过将 ΔW 分解为两个小型矩阵 $B \in \mathbb{R}^{d\times r}$ 和 $A \in \mathbb{R}^{r\times k}$，使得：

$$W = W_0 + \alpha BA \tag{4.10}$$

其中，秩 $r \ll min\{d, k\}$，B 和 A 分别用随机高斯分布和零进行初始化，α 是缩放因子，用于控制 LoRA 权重的大小。在训练过程中，固定预训练模型的参数，仅微调 B 和 A 的参数。因此，在训练时，LoRA 涉及的更新参数数量为 $r * (d + k)$，远小于全量微调 $d \times k$。图 4.8 给出了 LoRA 的示意图。在实际使用中，对于基于 Transformer 的大语言模型，密集层通常有两种类型：注意力模块中的投影层和前馈神经网络（FFN）模块中的投影层。在原始研究中，LoRA 被应用于注意力层的权重矩阵。后续工作表明将其应用于 FFN 层可以进一步提高模型性能 [14]。

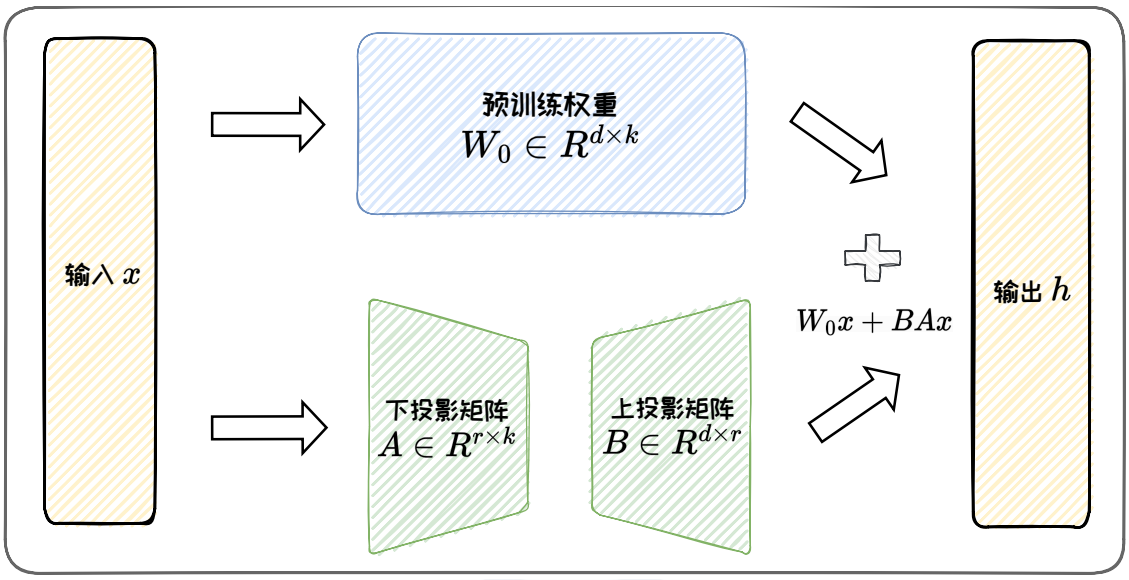

图 4.8: LoRA 示意图

LoRA 仅微调部分低秩参数，因此具有很高的参数效率，同时不会增加推理延迟 [10]。此外，低秩矩阵还可以扩展为低秩张量 [3]，或与 Kronecker 分解结合使用，以进一步提高参数效率 [9, 16]。除了参数效率外，LoRA 还具有可插拔性，因为在训练后可以将 LoRA 参数与模型参数分离。LoRA 的可插拔特性使其能够被多个用户共享和重复使用 [37]，当我们有多个任务的 LoRA 插件时，可以将这些插件组合在一起，并期望获得良好的跨任务泛化性能 [20]。我们将在 4.4.3 提供具体案例详细介绍基于 LoRA 插件的任务泛化。

参数效率分析

下面我们以一个具体的案例分析 LoRA 的参数效率。在 LLaMA2-7B [39] 模型中微调第一个 FFN 层的密集权重矩阵为例，全量微调需要调整 $11,008 \times 4,096 = 45,088,768$ 个参数。而当 $r = 4$ 时，LoRA 只需调整 $(11,008 \times 4) + (4 \times 4,096) = 60,416$ 个参数。对于这一层，与全量微调相比，LoRA 微调的参数不到原始参数量的千分之一。具体来说，模型微调的内存使用主要涉及四个部分：

- 权重内存（Weight Memory）：用于存储模型权重所需的内存；

- 激活内存（Activation Memory）：前向传播内存时中间激活带来的显存占用，主要取决于 batch size 大小以及序列长度等；
- 梯度内存（Gradient Memory）：在反向传播期间需要用来保存梯度的内存，这些梯度仅针对可训练参数进行计算；
- 优化器内存（Optimization Memory）：用于保存优化器状态的内部存在。例如，Adam 优化器会保存可训练参数的 “一阶动量” 和 “二阶动量”。

文献 [33] 提供了关于在 LLaMA2-7B 模型上使用批量大小为 1，在单个 NVIDIA RTX4090（24GB）GPU 上进行全量微调和 LoRA 微调的全面实验对比。根据这项研究，全量微调大约需要 60GB 的内存，超出了 RTX4090 GPU 的显存容量。相比之下，LoRA 微调只需要大约 23GB 的内存。LoRA 显著减少了内存使用，并使得在单个 NVIDIA RTX4090（24GB）GPU 上进行 LLaMA2-7B 微调成为可能。具体来说，**由于可训练参数较少，优化器内存和梯度内存分别减少了约 25GB 和 14GB**。另外，虽然 LoRA 引入了额外的 “增量参数”，导致**激活内存和权重内存略微增加（总计约 2GB）**，但考虑到整体内存的减少，这种增加是可以忽略不计的。此外，减少涉及到的参数计算可以加速反向传播。与全量微调相比，LoRA 的速度提高了 1.9 倍。

4.4.2 LoRA 相关变体

虽然 LoRA 在一些下游任务上能够实现较好的性能，但在许多下游任务（如数学推理 [4, 6, 53]）上，LoRA 与全量微调之间仍存在性能差距。为弥补这一差距，许多方法被提出，以进一步提升 LoRA 在下游任务中的适应性能。现有方法主要从以下几个角度进行改进：(1) 打破低秩瓶颈；(2) 动态秩分配；(3) 训练过程优化。接下来，将分别介绍这三种类型变种的代表性方法。

1. 打破低秩瓶颈

LoRA 的低秩更新特性使其在参数效率上具有优势；然而，这也限制了大规模语言模型记忆新知识和适应下游任务的能力 [4, 13, 21, 53]，即存在低秩瓶颈。Biderman 等人 [4] 的实验研究表明，全量微调的秩显著高于 LoRA 的秩（10-100 倍），并且增加 LoRA 的秩可以缩小 LoRA 与全量微调之间的性能差距。因此，一些方法被提出，旨在打破低秩瓶颈 [25, 35, 47]。

例如，ReLoRA [25] 提出了一种合并和重置（merge-and-reinit）的方法，该方法周期性地将 LoRA 模块合并到大语言模型中，并在微调期间重新初始化 LoRA 模块和优化器状态。具体地，合并的过程如下：

$$W^i \leftarrow W^i + \alpha B^i A^i \tag{4.11}$$

其中，W^i 是原始的权重矩阵，B^i 和 A^i 是低秩分解得到的矩阵，α 是缩放因子。合并后，将重置 B^i 和 A^i 的值重置，通常 B^i 会使用特定的初始化方法（如 Kaiming 初始化）重新初始化，而 A^i 则被设置为零。为了防止在重置后模型性能发散，ReLoRA 还会通过幅度剪枝对优化器状态进行部分重置。合并和重置的过程允许模型在保持总参数量不变的情况下，通过多次低秩 LoRA 更新累积成高秩状态，从而使得 ReLoRA 能够训练出性能接近全秩训练的模型。

2. 动态秩分配

然而，LoRA 的秩并不总是越高越好，冗余的 LoRA 秩可能会导致性能和效率的退化。并且，微调时权重的重要性可能会因 Transformer 模型中不同层而存在差异，因此需要为每个层分配不同的秩 [7, 30, 40, 51]。

例如，AdaLoRA [51] 通过将参数更新矩阵参数化为奇异值分解（SVD）的形式，再通过奇异值剪枝动态调整不同层中 LoRA 模块的秩。具体地，AdaLoRA 使用奇异值分解重新表示 ΔW，即

$$W = W_0 + \Delta W = W_0 + P\Lambda Q \tag{4.12}$$

其中，$P \in \mathbb{R}^{d\times r}$ 和 $Q \in \mathbb{R}^{r\times k}$ 是正交的，Λ 是一个对角矩阵，其中包含 $\{\lambda_i\}_{1\le i\le r}$ 的奇异值。在训练过程中，W_0 的参数被固定，仅更新 P、Λ 和 Q 的参数。根据梯度权重乘积大小的移动平均值构造奇异值的重要性得分，对不重要的奇异值进行迭代剪枝。此外，为了增强稳定训练性，AdaLoRA 引入一个额外的惩罚项确保 P 和 Q 之间的正交性:

$$R(P,Q) = ||P^TP - I||_F^2 + ||QQ^T - I||_F^2. \tag{4.13}$$

其中，I 是单位矩阵，$||\cdot||_F$ 代表 Frobenius 范数。

3. 训练过程优化

实际上，LoRA 的收敛速度比全量微调要慢。此外，它还对超参数敏感，并且容易过拟合。这些问题影响了 LoRA 的效率并阻碍了其下游适应性能。为了解决这些问题，一些工作尝试对 LoRA 的训练过程进行优化 [31, 44]。代表性方法 DoRA (权重分解低秩适应) [29] 提出约束梯度更新，侧重于更新参数的方向变化。它将预训练权重 $W_0 \in \mathbb{R}^{d\times k}$ 分解为方向和大小两个组件，并仅将 LoRA 应用于方向组件以增强训练稳定性。具体地，DoRA 将 $W_0 \in \mathbb{R}^{d\times k}$ 重新表示为:

$$W_0 = m\frac{V}{\|V\|_c} = \|W_0\|_c \frac{W_0}{\|W_0\|_c}, \tag{4.14}$$

其中，$m \in \mathbb{R}^{1\times k}$ 是大小向量, $V \in \mathbb{R}^{d\times k}$ 是方向矩阵，$\|\cdot\|_c$ 是矩阵在每一列上的向量范数。随后，DoRA 仅对方向矩阵 V 施加 LoRA 进行参数化，定义为:

$$W' = \underline{m}\frac{V + \underline{\Delta V}}{\|V + \underline{\Delta V}\|_c} = \underline{m}\frac{W_0 + \underline{BA}}{\|W_0 + \underline{BA}\|_c}, \tag{4.15}$$

其中，ΔV 是由 LoRA 学习的增量方向更新，下划线参数表示可训练参数。

4.4.3 基于 LoRA 插件的任务泛化

LoRA 的一个优良的特性是插件化。我们可以在不同任务上训练的各种 LoRA 模块，这些模块可以插件化的方式保存、共享与使用。此外，我们还可以将多个任

务的 LoRA 模块组合，然后将不同任务的能力迁移到新任务。LoRAHub [20] 较早地提供了一个可用的多 LoRA 组合框架，用于新任务泛化。如图 4.9 所示，LoRAHub

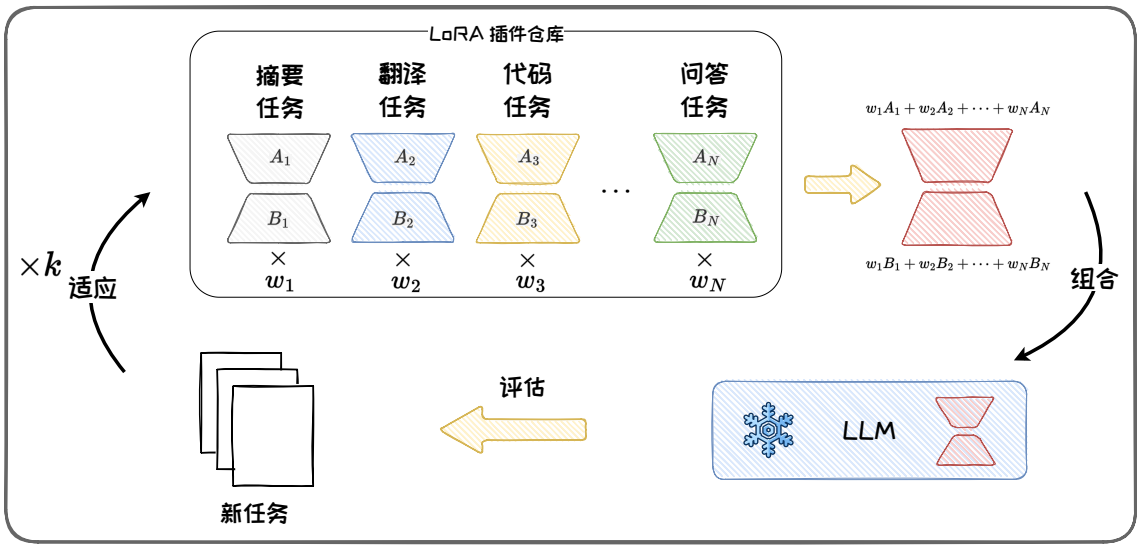

图 4.9: LoRAHub 示意图

包括两个阶段：**组合阶段**和**适应阶段**。**在组合阶段**，LoRAHub 将已学习的 LoRA 模块通过逐元素线性加权组合为单一模块：

$$\hat{m} = (w_1A_1 + w_2A_2 + \cdots + w_NA_N)\,(w_1B_1 + w_2B_2 + \cdots + w_NB_N) \tag{4.16}$$

其中，w_i 是第 i 个 LoRA 模块的权重，$\hat{m}$ 是组合后的模块，$A_{i=1}^{N}$ 和 $B_{i=1}^{N}$ 分别是 N 个 LoRA 分解矩阵。**在适应阶段**，给定一些新任务的示例，通过无梯度方法 Shiwa [28] 自适应地学习权重组合。适应和组合经过 k 次迭代，直至找到最优的权重组合，以完成对新任务的适应。

本节介绍了低秩适配方法 LoRA。LoRA 对参数矩阵进行低秩分解，通过仅训练低秩矩阵，大幅降低了训练涉及的参数量。此外，还介绍了从打破低秩瓶颈、动态秩分配以及训练过程优化等不同角度改进 LoRA 的变体。最后，介绍了基于插件 LoRA 的任务泛化方法 LoRAHub，通过对已学习的 LoRA 模块进行加权组合，LoRAHub 可以融合多任务能力并迁移到新任务上，提供了一个高效的跨任务学习

范式。

4.5 实践与应用

PEFT 技术在许多领域中展现了其强大的应用潜力。例如，在自然语言处理（NLP）领域，PEFT 被用于对大语言模型进行下游任务适配。在计算机视觉（CV）领域，PEFT 技术也被用于图片生成模型。本章将详细探讨 PEFT 的实践与应用。在实践部分，我们将介绍目前最流行的 PEFT 框架，Hugging Face 开发的开源库 HF-PEFT，并提供其使用方法和相关技巧。在应用部分，我们将展示 PEFT 技术在不同垂直领域中的应用案例，包括表格数据处理和金融领域的 Text-to-SQL 生成任务。这些案例不仅证明了 PEFT 在提升大模型特定任务性能方面的有效性，也为未来的研究和应用提供了有益的参考。

4.5.1 PEFT 实践

1. PEFT 主流框架

目前最流行的 PEFT 框架是由 Hugging Face 开发的开源库 HF-PEFT [2]，它旨在提供最先进的参数高效微调方法。HF-PEFT 框架的设计哲学是高效和灵活性。HF-PEFT 集成了多种先进的微调技术，如 LoRA、Apdater-tuning、Prompt-tuning 和 IA3 等。HF-PEFT 支持与 Hugging Face 的其他工具如 Transformers、Diffusers 和 Accelerate 无缝集成，并且支持从单机到分布式环境的多样化训练和推理场景。HF-PEFT 特别适用于大模型，能够在消费级硬件上实现高性能，并且可以与模型量化技术兼容，进一步减少了模型的内存需求。HF-PEFT 支持多种架构模型，包括 Transformer 和 Diffusion，并且允许用户手动配置，在自己的模型上启用 PEFT。

[2]https://github.com/huggingface/peft

HF-PEFT 的另一个显著优势是它的易用性。它提供了详细的文档、快速入门指南和示例代码[3]，可以帮助用户了解如何快速训练 PEFT 模型，以及如何加载已有的 PEFT 模型进行推理。此外，HF-PEFT 拥有活跃的社区支持，并鼓励社区贡献，是一个功能强大、易于使用且持续更新的库。

2. HF-PEFT 框架使用

使用 HF-PEFT 框架进行模型微调可以显著提升模型在特定任务上的性能，同时保持训练的高效性。通常，使用 HF-PEFT 框架进行模型微调的步骤如下：

1. **安装与配置**：首先，在环境中安装 HF-PEFT 框架及其依赖项，主要是 Hugging Face 的 Transformers 库。
2. **选择模型与数据**：根据任务需求，挑选合适的预训练模型，并准备相应的训练数据集。
3. **确定微调策略**：选择适合任务的微调方法，例如 LoRA 或适配器技术。
4. **模型准备**：加载预训练模型并为选定的微调方法进行配置，包括任务类型、推理模式、参数值等。
5. **模型训练**：定义完整的训练流程，包括损失函数、优化器设置，并执行训练，同时可应用数据增强和学习率调度等技术。

通过上述步骤，可以高效地使用 HF-PEFT 框架进行模型微调，以适应特定的任务需求，并提升模型性能。

3. PEFT 相关技巧

HF-PEFT 库提供了多种参数高效微调技术，用于在不微调所有模型参数的情况下，有效地将预训练语言模型适应各种下游应用。以下是一些 PEFT 技术常用的参数设置方法：

Prompt Tuning

[3]https://huggingface.co/docs/peft/

- num_virtual_tokens: 表示为每个任务添加的 virtual tokens 的数量，通常设置在 10-20 之间。
- prompt_tuning_init：表示 prompt 参数的初始化方式。可以选择随机初始化（RANDOM）、文本初始化（TEXT），或者其他方式。文本初始化方式下，可以使用特定文本对 prompt embeddings 进行初始化以加速收敛。

Prefix Tuning

- num_virtual_tokens: 与 Prompt Tuning 中相同，表示构造的 virtual tokens 的数量，设置和 Prompt Tuning 类似。
- encoder_hidden_size: 表示用于 Prefix 编码的多层感知机（MLP）层的大小，通常与模型的隐藏层大小相匹配。

LoRA

- r: 秩的大小，用于控制更新矩阵的复杂度。通常可以选择较小的值如 4、8、16，对于小数据集，可能需要设置更小的 r 值。
- lora_alpha: 缩放因子，用于控制 LoRA 权重的大小，通常与 r 成反比，以保持权重更新的一致性。
- lora_dropout: LoRA 层的 dropout 比率，用于正则化以防止过拟合，可以设置为一个较小的值，比如 0.01。
- target_modules: 指定模型中 LoRA 中要应用的模块，如注意力机制中的 query、key、value 矩阵。可以选择特定的模块进行微调，或者微调所有线性层。

需要注意的是，具体的参数设置可能会根据所使用的模型、数据集和具体任务有所不同，因此在实际应用中可能需要根据实验结果进行调整。

4.5.2 PEFT 应用

本节我们将介绍两个比较有代表性的案例，它们将 PEFT 方法应用到垂直领域，提升大模型在垂直领域的性能。其中包括将大模型适配到表格数据以及金融领域的 Text-to-SQL 生成任务。

1. 表格数据

表格数据在现实世界中的应用非常广泛，涉及医疗保健、气候科学和金融等多个领域。然而，为训练监督学习算法以进行分类任务获取足够的标记数据常常面临挑战，尤其是在医疗保健领域，许多罕见疾病的存在限制了传统机器学习的应用。尽管深度学习模型在图像和文本分析方面取得了显著进展，但这些技术在表格数据上的应用效果并不理想。表格数据的特点——缺乏局部性、包含多种数据类型和相对较少的特征数量——使得传统的深度学习方法难以直接应用。

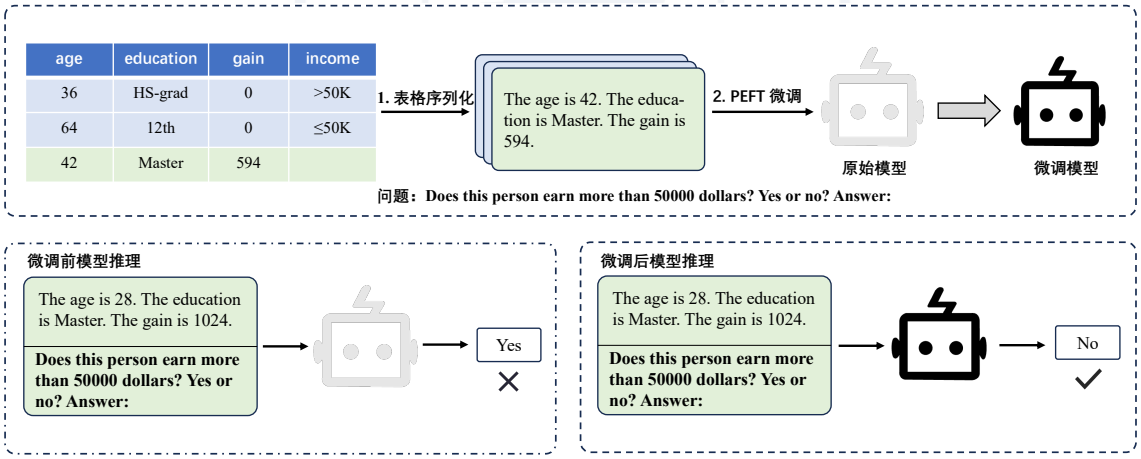

图 4.10: TabLLM 框架图

大语言模型的参数中编码了大量先验知识，即使在没有大量标记数据的情况下也能取得良好的性能。然而，在少量数据上进行全参数微调容易导致大模型过拟合。因此，采用 PEFT 技术是微调表格数据任务的理想方法。由于 PEFT 只微调部分参数，有效降低了过拟合的风险，使大语言模型在表格数据上的性能更加稳健。

例如，TabLLM [17] 提出基于大语言模型的少样本表格数据分类框架，图 4.10 给出该方法的框架图。该框架首先通过将表格数据序列化为自然语言字符串，并附上分类问题的简短描述来提示大语言模型。在少样本设置中，使用 LoRA 在一些带标签的样本对大语言模型进行微调。微调后，TabLLM 在多个基准数据集上的表现超过了以前的基于深度学习的表格分类方法。此外，微调后的 TabLLM 还表现出了强大的小样本学习能力，在极少样本设置下，TabLLM 的性能超过了梯度提升树等强大的传统基线。

2. 金融领域

在金融领域，表格数据的应用非常广泛，包括股票市场数据、财务报表、交易记录等。对于这些表格数据，生成结构化的 SQL 查询以便于数据库操作和数据分析是一个常见需求。然而，编写和调试复杂的 SQL 查询通常需要数据科学家花费大量时间，往往需要数分钟甚至更长。另外，SQL 涉及复杂的查询逻辑和严格的语法规则，对于初学者，熟练掌握并使用 SQL 可能需要数月或者更长的时间来学习和实践。Text-to-SQL 旨在将自然语言文本翻译成 SQL 查询，从而将 SQL 生成时间从分钟级缩短到秒级，使数据科学家能够专注于分析和建模，加快数据驱动决策的速度，同时让广大普通人也能操作和管理数据库，显著提高了数据分析的效率，让更多的人能够挖掘和利用数据的价值。但是在金融垂直领域，Text-to-SQL 数据标注专业性强，标注数据获取成本高，在实际场景中难以获得足够的数据。利用这些少量数据进行全参数微调时则容易导致大模型过拟合，因此采用 PEFT 技术进行部分参数微调十分适合金融垂直领域 Text-to-SQL 任务。

如图 4.11 所示，FinSQL [50] 提出针对金融垂直领域 Text-to-SQL 训练推理框架。该框架包含提示构造、参数高效微调和输出校准三个部分。首先，提示构造通过不同策略增强原始数据，并且构造高效检索器，检索与用户查询相关表格和字段。然后，采用 PEFT 技术对基座模型进行微调，通过 LoRAHub 融合多个 LoRA

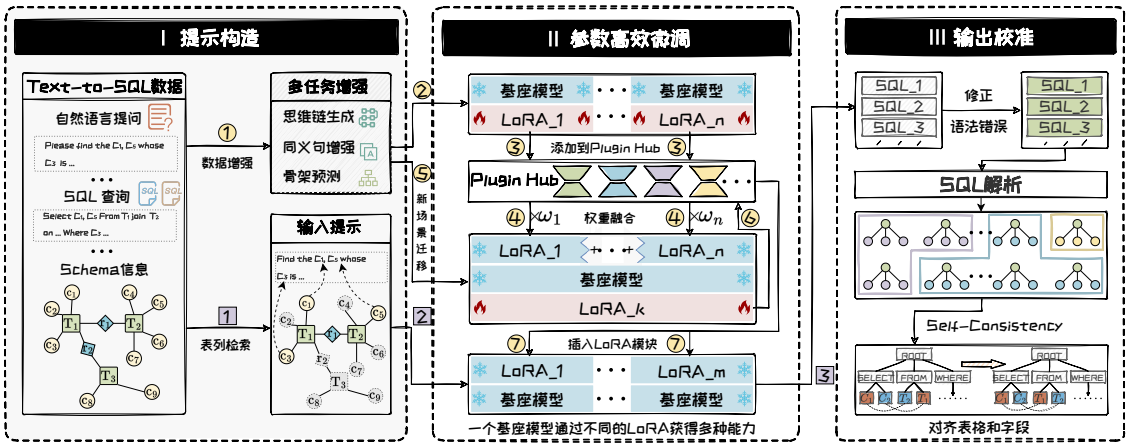

图 4.11: FinSQL 示意图

模块，提高少样本场景的性能。最后，输出校准模块会修正生成 SQL 中的语法错误，并用 Self-Consistency 方法来选择一致性 SQL。FinSQL 相比于基线方法，不仅显著提升了准确性和效率，还展现了在处理复杂金融查询时的强大能力，为金融领域的数据分析和决策支持提供了强有力的技术支撑。

本节介绍了参数高效微调技术的实践与应用。首先，我们介绍了 PEFT 主流框架 HF-PEFT 及其使用方法，并介绍了 PEFT 的相关技巧。最后，展示了 PEFT 技术如何助力大语言模型在垂直领域的性能提升，包括表格数据的分类任务和金融领域的 Text-to-SQL 任务，使大语言模型适配垂域任务的同时保证训练效率。

参考文献

[1] Armen Aghajanyan, Sonal Gupta, and Luke Zettlemoyer. “Intrinsic Dimensionality Explains the Effectiveness of Language Model Fine-Tuning”. In: *ACL/IJCNLP*. 2021.

[2] Alan Ansell et al. “Composable Sparse Fine-Tuning for Cross-Lingual Transfer”. In: *ACL*. 2022.

[3] Daniel Bershatsky et al. “LoTR: Low Tensor Rank Weight Adaptation”. In: *arXiv preprint arXiv:2402.01376* (2024).

[4] Dan Biderman et al. “LoRA Learns Less and Forgets Less”. In: *arXiv preprint arXiv.2405.09673* (2024).

[5] Sarkar Snigdha Sarathi Das et al. “Unified Low-Resource Sequence Labeling by Sample-Aware Dynamic Sparse Finetuning”. In: *EMNLP*. 2023.

[6] Ning Ding et al. “Parameter-efficient fine-tuning of large-scale pre-trained language models”. In: *Nat. Mac. Intell.* 5.3 (2023), pp. 220–235.

[7] Ning Ding et al. “Sparse Low-rank Adaptation of Pre-trained Language Models”. In: *EMNLP*. 2023.

[8] Qingxiu Dong et al. “A Survey for In-context Learning”. In: *CoRR* abs/2301.00234 (2023).

[9] Ali Edalati et al. “KronA: Parameter Efficient Tuning with Kronecker Adapter”. In: *arXiv preprint arXiv:2212.10650* (2022).

[10] Vlad Fomenko et al. “A Note on LoRA”. In: *arXiv preprint arXiv:2404.05086* (2024).

[11] Jonathan Frankle and Michael Carbin. “The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks”. In: *ICLR*. 2019.

[12] Zihao Fu et al. “On the Effectiveness of Parameter-Efficient Fine-Tuning”. In: *AAAI*. 2023.

[13] Andi Han et al. “SLTrain: a sparse plus low-rank approach for parameter and memory efficient pretraining”. In: *arXiv preprint arXiv:2406.02214* (2024).

[14] Junxian He et al. “Towards a Unified View of Parameter-Efficient Transfer Learning”. In: *ICLR*. 2022.

[15] Shwai He et al. “SparseAdapter: An Easy Approach for Improving the Parameter-Efficiency of Adapters”. In: *Findings of EMNLP*. 2022.

[16] Xuehai He et al. “Parameter-Efficient Model Adaptation for Vision Transformers”. In: *AAAI*. 2023.

[17] Stefan Hegselmann et al. “TabLLM: Few-shot Classification of Tabular Data with Large Language Models”. In: *AISTATS*. 2023.

[18] Neil Houlsby et al. “Parameter-Efficient Transfer Learning for NLP”. In: *ICML*. 2019.

[19] Edward J. Hu et al. “LoRA: Low-Rank Adaptation of Large Language Models”. In: *ICLR*. 2022.

[20] Chengsong Huang et al. “LoraHub: Efficient Cross-Task Generalization via Dynamic LoRA Composition”. In: *arXiv preprint arXiv:2307.13269* (2023).

[21] Ting Jiang et al. “MoRA: High-Rank Updating for Parameter-Efficient Fine-Tuning”. In: *arXiv preprint arXiv:2405.12130* (2024).

[22] Jaejun Lee, Raphael Tang, and Jimmy Lin. “What Would Elsa Do? Freezing Layers During Transformer Fine-Tuning”. In: *arXiv preprint arXiv:1911.03090* (2019).

[23] Brian Lester, Rami Al-Rfou, and Noah Constant. “The Power of Scale for Parameter-Efficient Prompt Tuning”. In: *EMNLP*. 2021, pp. 3045–3059.

[24] Xiang Lisa Li and Percy Liang. “Prefix-Tuning: Optimizing Continuous Prompts for Generation”. In: *ACL*. 2021.

[25] Vladislav Lialin et al. “ReLoRA: High-Rank Training Through Low-Rank Updates”. In: *NIPS Workshop*. 2023.

[26] Baohao Liao, Yan Meng, and Christof Monz. “Parameter-Efficient Fine-Tuning without Introducing New Latency”. In: *ACL*. 2023.

[27] Alisa Liu et al. “Tuning Language Models by Proxy”. In: *arXiv preprint arXiv:2401.08565* (2024).

[28] Jialin Liu et al. “Versatile black-box optimization”. In: *GECCO*. 2020, pp. 620–628.

[29] Shih-Yang Liu et al. “DoRA: Weight-Decomposed Low-Rank Adaptation”. In: *arXiv preprint arXiv:2402.09353* (2024).

[30] Yulong Mao et al. “DoRA: Enhancing Parameter-Efficient Fine-Tuning with Dynamic Rank Distribution”. In: *arXiv preprint arXiv:2405.17357* (2024).

[31] Fanxu Meng, Zhaohui Wang, and Muhan Zhang. “Pissa: Principal singular values and singular vectors adaptation of large language models”. In: *arXiv preprint arXiv:2404.02948* (2024).

[32] Jinjie Ni et al. *Instruction in the Wild: A User-based Instruction Dataset*. https://github.com/XueFuzhao/InstructionWild. 2023.

[33] Rui Pan et al. “LISA: Layerwise Importance Sampling for Memory-Efficient Large Language Model Fine-Tuning”. In: *arXiv preprint arXiv.2403.17919* (2024).

[34] Jonas Pfeiffer et al. “AdapterFusion: Non-Destructive Task Composition for Transfer Learning”. In: *EACL*. Ed. by Paola Merlo, Jörg Tiedemann, and Reut Tsarfaty. 2021.

[35] Pengjie Ren et al. “Mini-Ensemble Low-Rank Adapters for Parameter-Efficient Fine-Tuning”. In: *arXiv preprint arXiv.2402.17263* (2024).

[36] Victor Sanh et al. *Multitask Prompted Training Enables Zero-Shot Task Generalization*. 2021. arXiv: 2110.08207 [cs.LG].

[37] Ying Sheng et al. “S-LoRA: Serving Thousands of Concurrent LoRA Adapters”. In: *arXiv preprint arXiv:2311.03285* (2023).

[38] Yi-Lin Sung, Varun Nair, and Colin Raffel. “Training Neural Networks with Fixed Sparse Masks”. In: *NIPS*. 2021.

[39] Hugo Touvron et al. “Llama 2: Open foundation and fine-tuned chat models”. In: *arXiv preprint arXiv:2307.09288* (2023).

[40] Mojtaba Valipour et al. “Dylora: Parameter efficient tuning of pre-trained models using dynamic search-free low-rank adaptation”. In: *arXiv preprint arXiv:2210.07558* (2022).

[41] Mojtaba Valipour et al. “DyLoRA: Parameter-Efficient Tuning of Pre-trained Models using Dynamic Search-Free Low-Rank Adaptation”. In: *EACL*. 2023.

[42] Zhongwei Wan et al. “Efficient Large Language Models: A Survey”. In: *arXiv preprint arXiv:2312.03863* (2023).

[43] Alex Wang et al. “GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding”. In: *ICLR*. 2019.

[44] Hanqing Wang et al. “MiLoRA: Harnessing Minor Singular Components for Parameter-Efficient LLM Finetuning”. In: *arXiv preprint arXiv:2406.09044* (2024).

[45] Yizhong Wang et al. “Self-Instruct: Aligning Language Models with Self-Generated Instructions”. In: *ACL*. 2023.

[46] Jason Wei et al. “Finetuned Language Models are Zero-Shot Learners”. In: *International Conference on Learning Representations*.

[47] Wenhan Xia, Chengwei Qin, and Elad Hazan. “Chain of LoRA: Efficient Fine-tuning of Language Models via Residual Learning”. In: *arXiv preprint arXiv.2401.04151* (2024). DOI: 10.48550/ARXIV.2401.04151.

[48] Runxin Xu et al. “Raise a Child in Large Language Model: Towards Effective and Generalizable Fine-tuning”. In: *EMNLP*. 2021.

[49] Elad Ben Zaken, Yoav Goldberg, and Shauli Ravfogel. “BitFit: Simple Parameter-efficient Fine-tuning for Transformer-based Masked Language-models”. In: *ACL*. 2022.

[50] Chao Zhang et al. “FinSQL: Model-Agnostic LLMs-based Text-to-SQL Framework for Financial Analysis”. In: *SIGMOD*. 2024.

[51] Qingru Zhang et al. “Adaptive Budget Allocation for Parameter-Efficient Fine-Tuning”. In: *ICLR*. 2023.

[52] Shengyu Zhang et al. “Instruction Tuning for Large Language Models: A Survey”. In: *arXiv preprint arXiv:2308.10792* (2023).

[53] Jiawei Zhao et al. “GaLore: Memory-Efficient LLM Training by Gradient Low-Rank Projection”. In: *arXiv preprint arXiv.2403.03507* (2024).

[54] Mengjie Zhao et al. “Masking as an Efficient Alternative to Finetuning for Pretrained Language Models”. In: *EMNLP*. 2020, pp. 2226–2241.

[55] Yaoming Zhu et al. “Counter-Interference Adapter for Multilingual Machine Translation”. In: *Findings of EMNLP*. 2021.