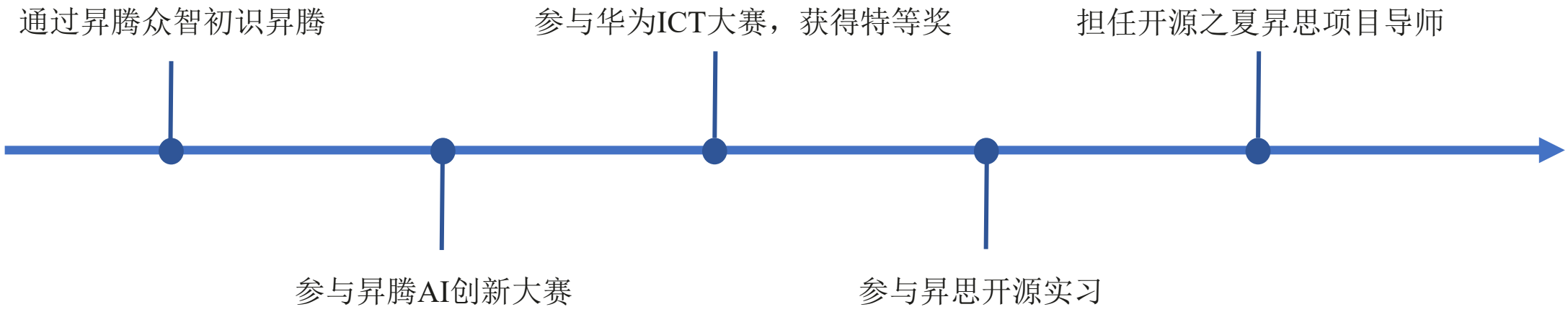
昇腾+MindSpore+MindSpore NLP
极简风的大模型微调实战

昇思
MindSpore

昇腾相关经历介绍

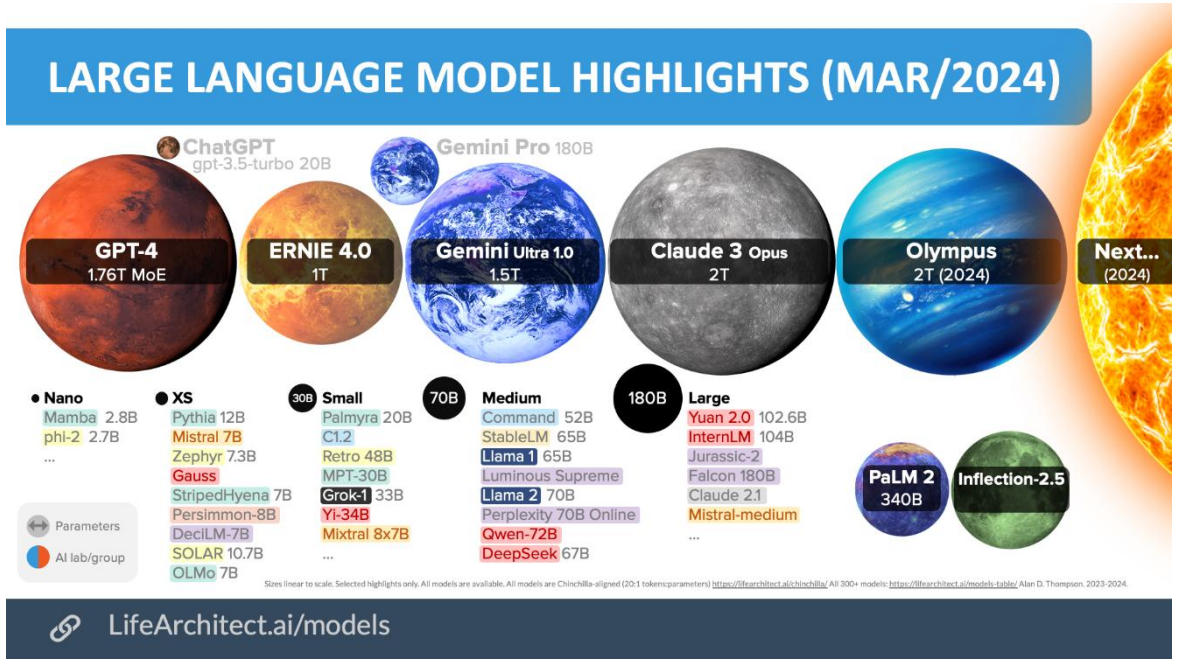


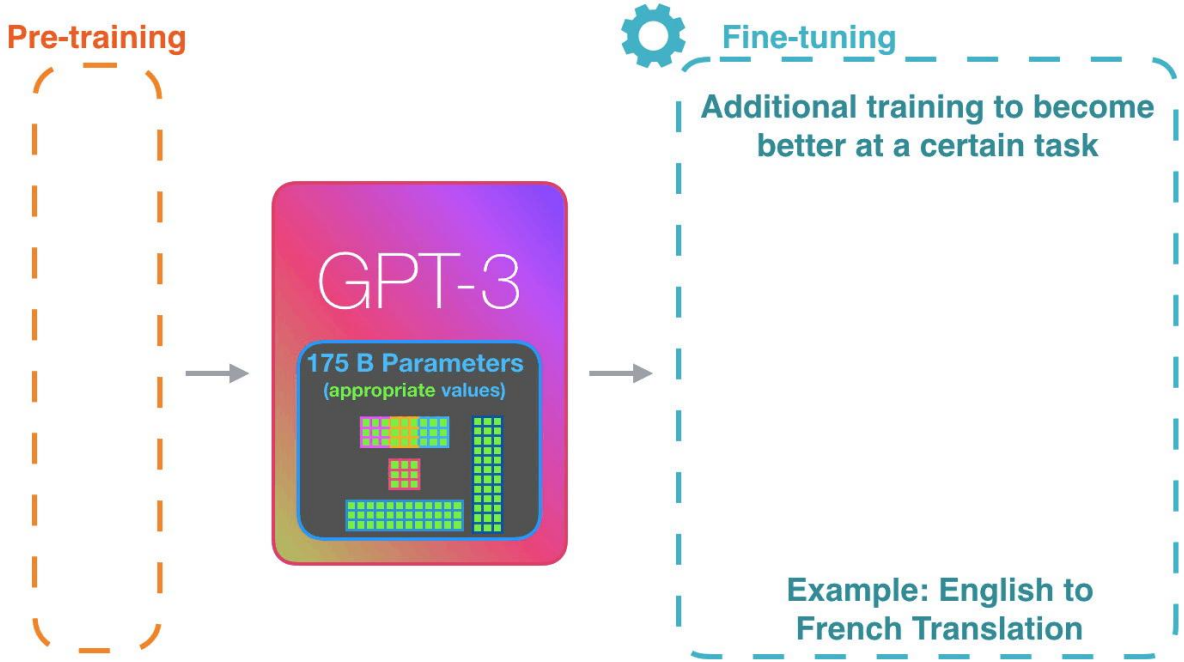
产业实践和大赛（以题目难度逐渐增加排序）		背景介绍	活动时间	题目类型	计划题量
比赛	开源之夏	开源之夏（英文简称 OSPP）是开源软件供应链点亮计划的系列暑期活动，它由中国科学院软件研究所与 openEuler 社区共同举办。开源之夏将联合各大开源社区，针对重要开源软件的开发与维护提供项目，并向全球高校学生开放报名。	6月-9月	1. 模型套件 2. 应用实践	30+
	昇腾AI创新大赛	1. 基于Orange Pi AIpro 开发板进行应用创新，在支持广泛业界生态的同时，牵引使用昇思MindSpore AI框架开发可落地的作品方案。 2. 围绕MindSpore开源，设置算法等相关命题，实现精度/性能极致挑战提升。	5月-10月	AI解决方案征集+算法赛题	30+
	互联网+大赛	中国“互联网+”大学生创新创业大赛，由教育部与政府、各高校共同主办的一项技能大赛。	8月-12月	AI解决方案征集	不限
	华为ICT大赛	“华为ICT大赛”是华为公司面向全球大学生打造的ICT人才竞技交流赛事，旨在为华为ICT学院及有意愿成为华为ICT学院高校的学生提供国际化的竞技与交流平台，提升学生的ICT知识水平，加强学生的实践动手能力以及运用新技术、新平台的创新创造能力。	9月开始持续到次年	AI解决方案征集	不限
	开源软件创新大赛	1. 代码评注赛： 注释类代码评注、技术博客类代码评注 2. 开源项目贡献赛：解决Bug、实现功能、原创工具、创新生态应用	5月-10月	AI解决方案征集	不限

PEFT

(Parameter-Efficient Fine-Tuning)

What is Parameter-Efficient Fine-tuning





随着预训练模型的参数越来越大，让个人对于大模型的全量微调望而却步，近年来研究者们提出了各种各样的**参数高效微调方法**（PEFT），即固定住预训练模型的大部分参数，仅调整模型的一小部分参数来达到与全部参数的微调接近的效果。

Fine-tuning是采用已经在某项任务上预训练过的模型，然后对其进行微调以执行类似任务的过程。例如，对在大规模英文数据集上训练的LLAMA3模型，进行数学运算数据集微调训练，以提升模型的数学能力。

Why use Parameter-Efficient Fine-Tuning?

- 大模型在下游任务上往往泛化性不够好
- 高效参数微调相比全量微调训练参数更少
- 高效参数微调忘的少，相比全量微调不容易过拟合
- 更适合online training

这张图展示了 PEFT 微调过程的基本原理，特别是如何利用用户与模型输出的交互来更新模型：

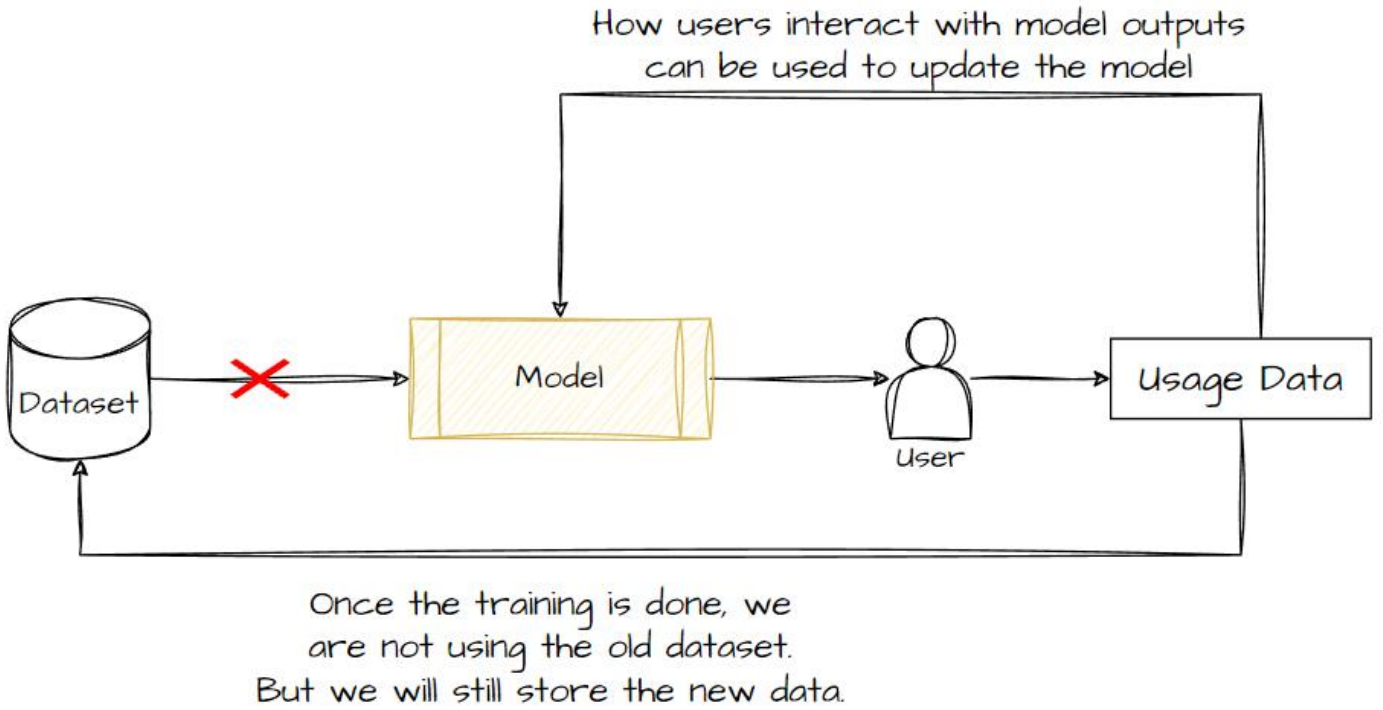
初始训练: 模型使用原始数据集进行初始训练。

用户交互: 用户与模型的输出进行交互，产生新的使用数据。

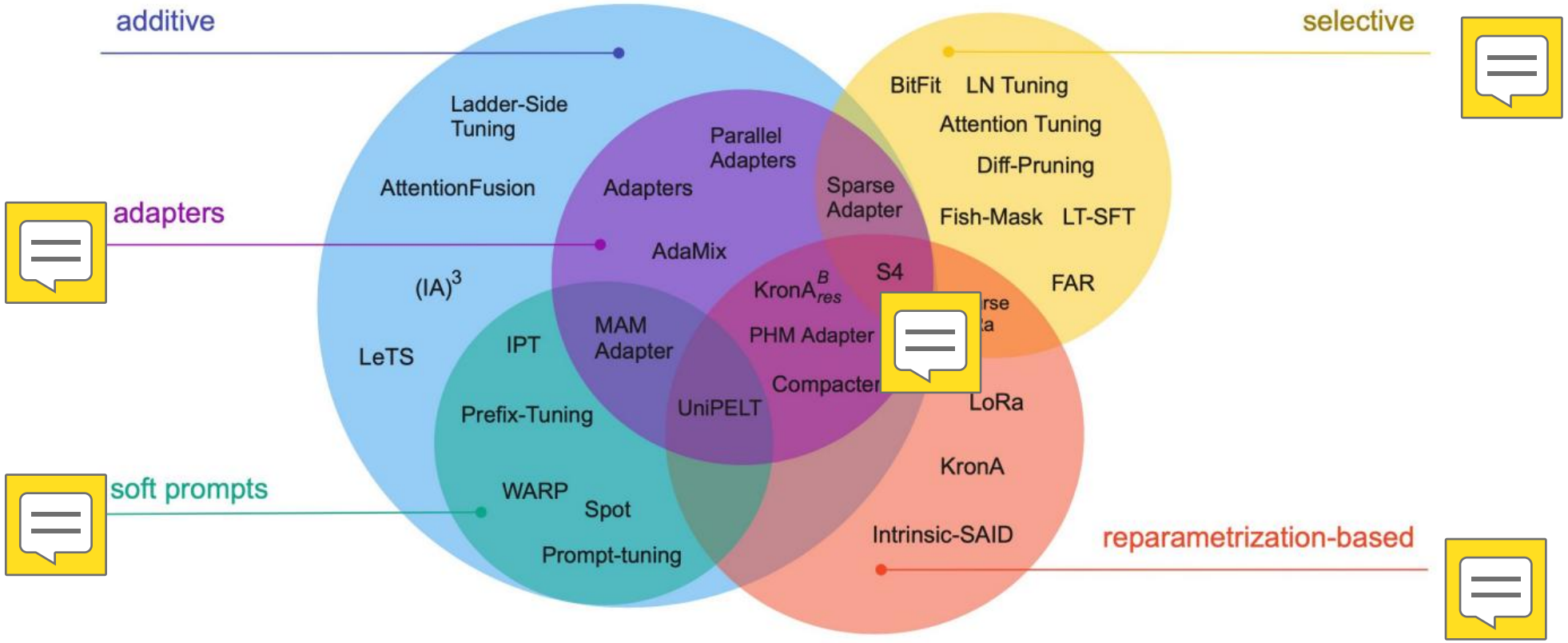
数据收集: 用户的交互数据被收集并存储。

模型更新: 使用数据被用于更新模型，使模型能够更好地适应新的任务和用户需求。

通过这种方式，PEFT 能够在不改变模型原始权重的情况下，通过用户交互数据动态地调整模型的行为，使其更好地适应新任务和用户需求。

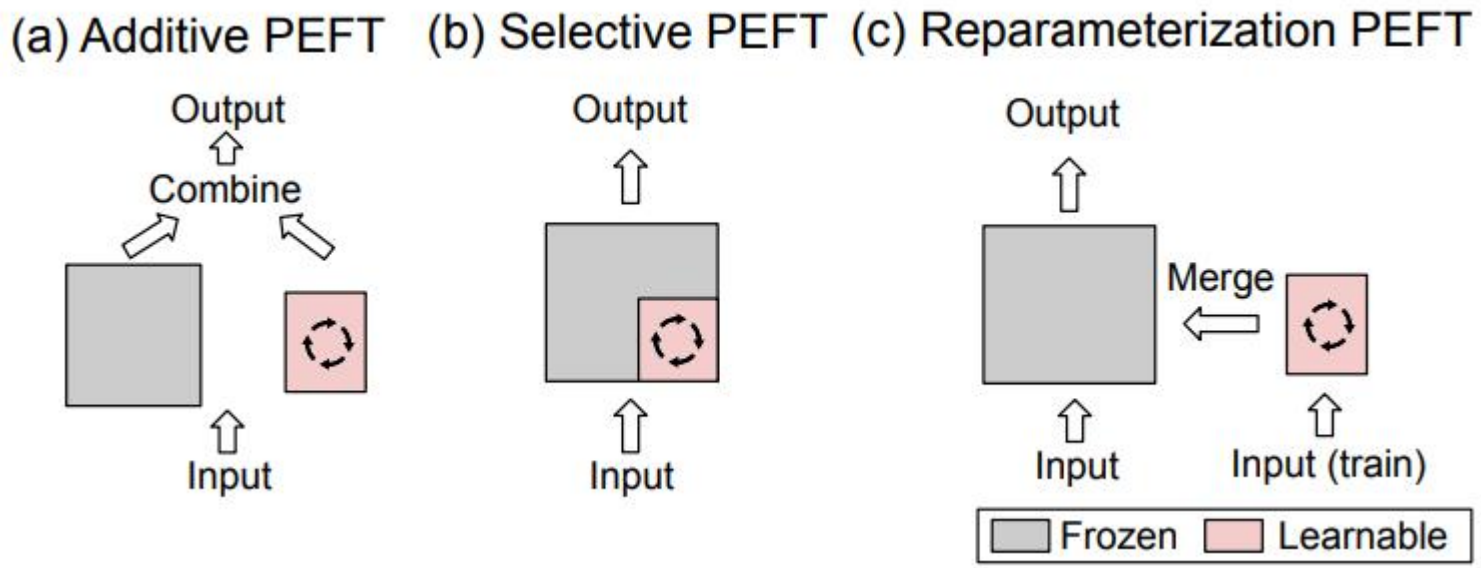


PEFT Methods



PEFT Methods

- **Additive PEFT(加性微调):**在模型的特定位置添加可学习的模块或参数。
- **Selective PEFT(选择性微调):**在微调过程中只更新模型中的一部分参数，而保持其余参数固定。
- **Reparameterization PEFT(重参数化微调):**通过构建原始模型参数的低秩表示，在训练过程中增加可学习参数，以实现参数高效微调。



Additive PEFT

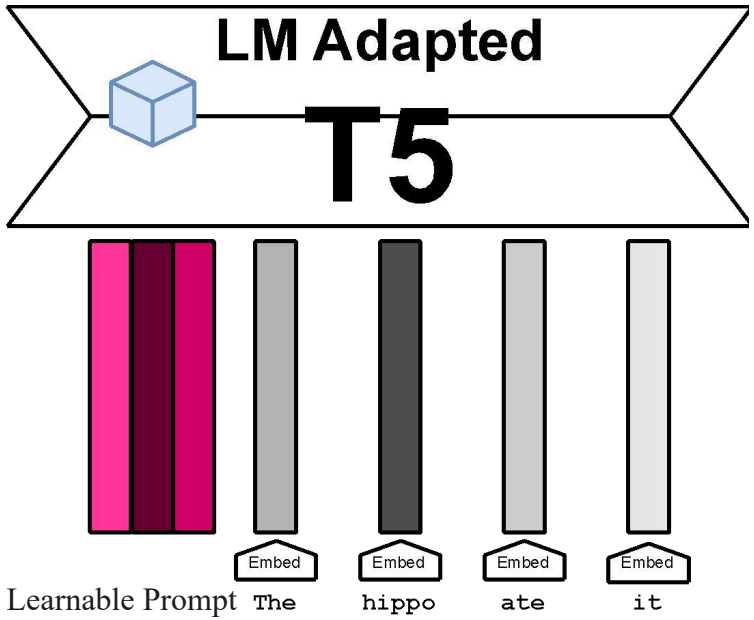
Methods: prompt tuning, prefix tuning and adapters

Prompt Tuning(Soft Prompt Method)

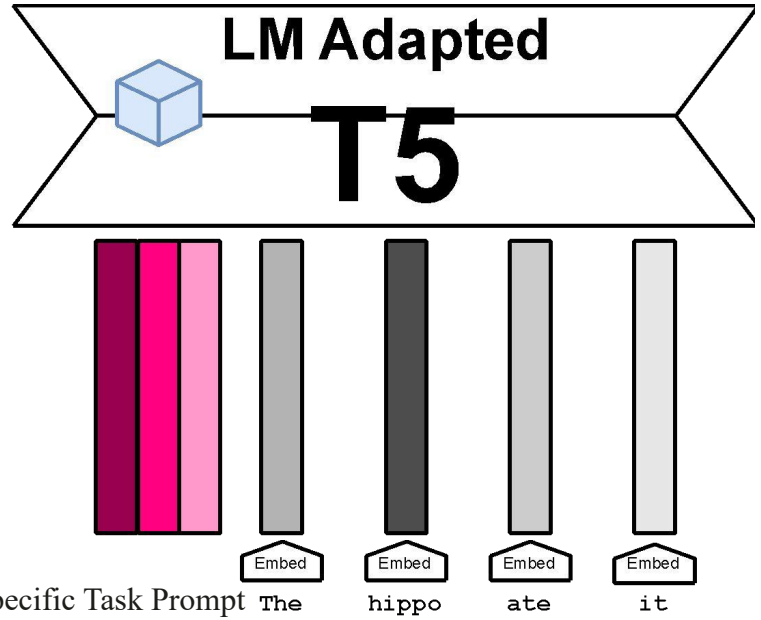
Prompt
[X] is located in [Y]. *(original)*
[X] is located in which country or state? [Y].
[X] is located in which country? [Y].
[X] is located in which country? In [Y].

Hard Prompt:字符串文本提示

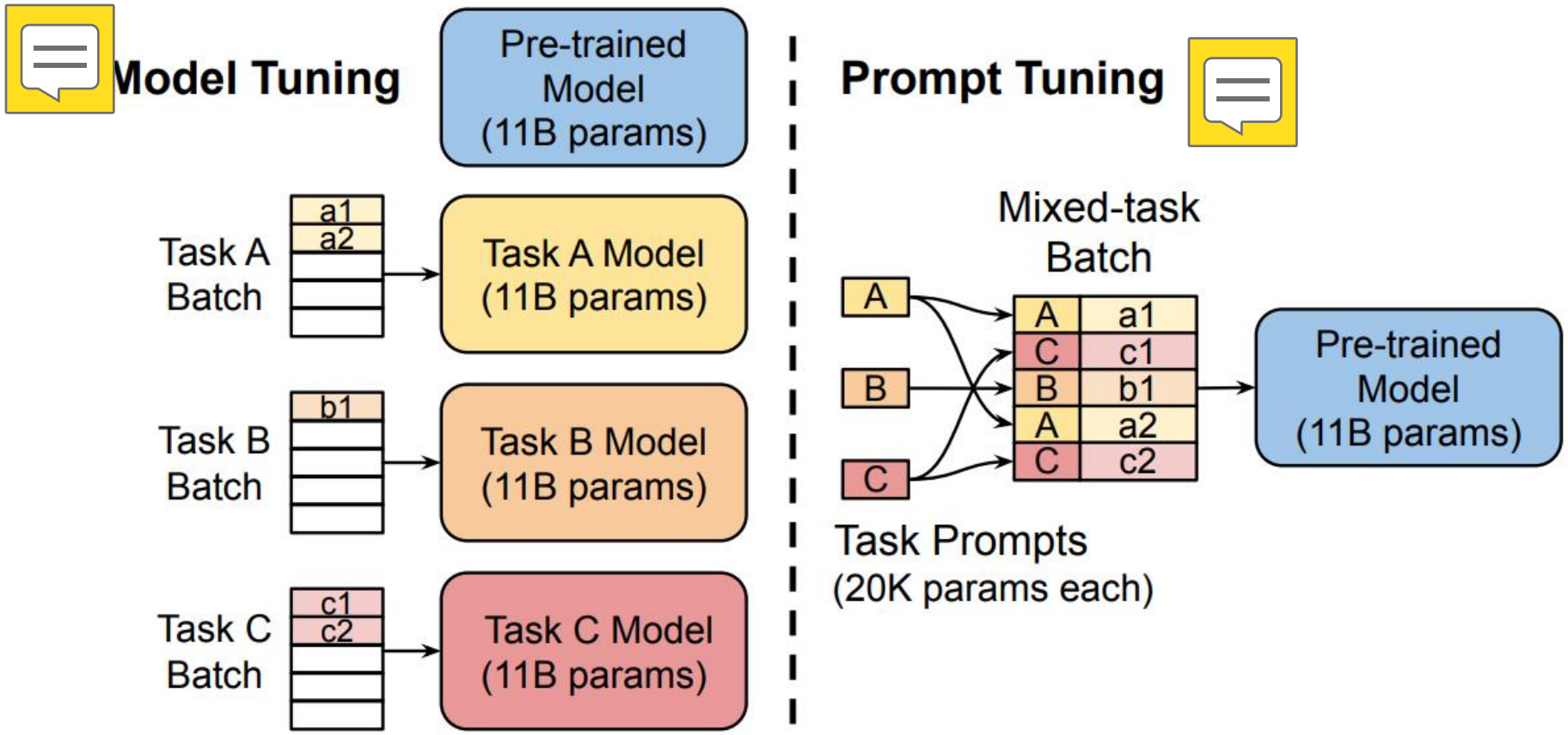
Soft Prompt:可学习的参数提示



Downstream Task



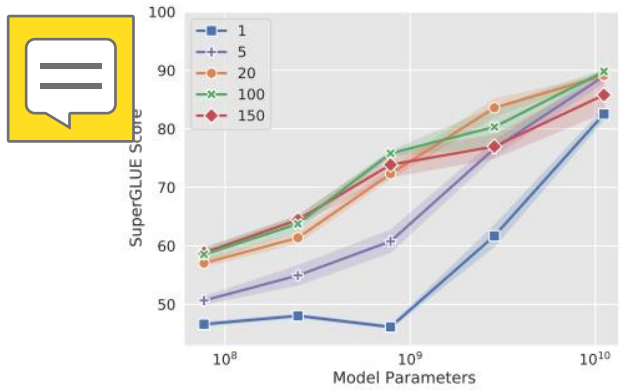
Prompt Tuning



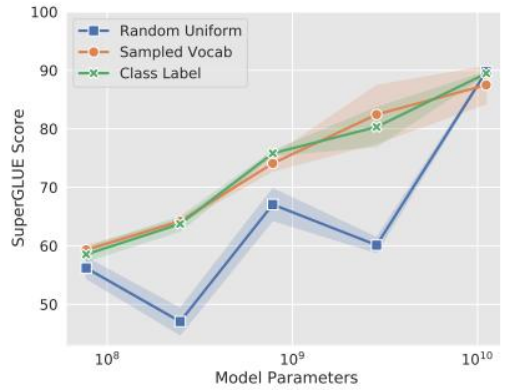
传统多任务训练推理

Prompt训练推理

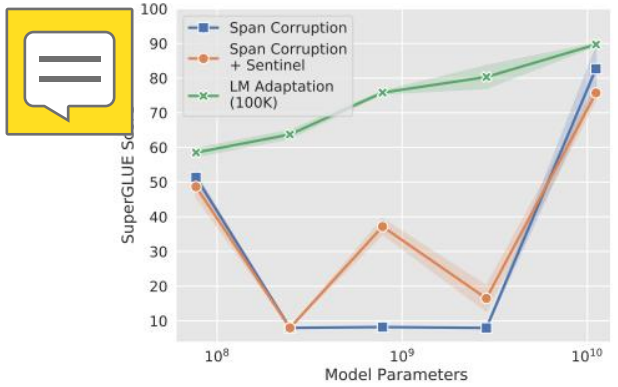
Efficiency of Prompt Tuning



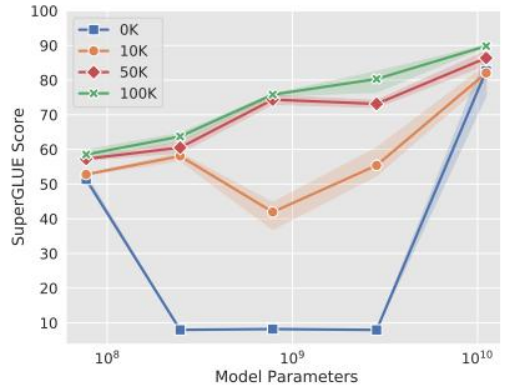
(a) Prompt length



(b) Prompt initialization



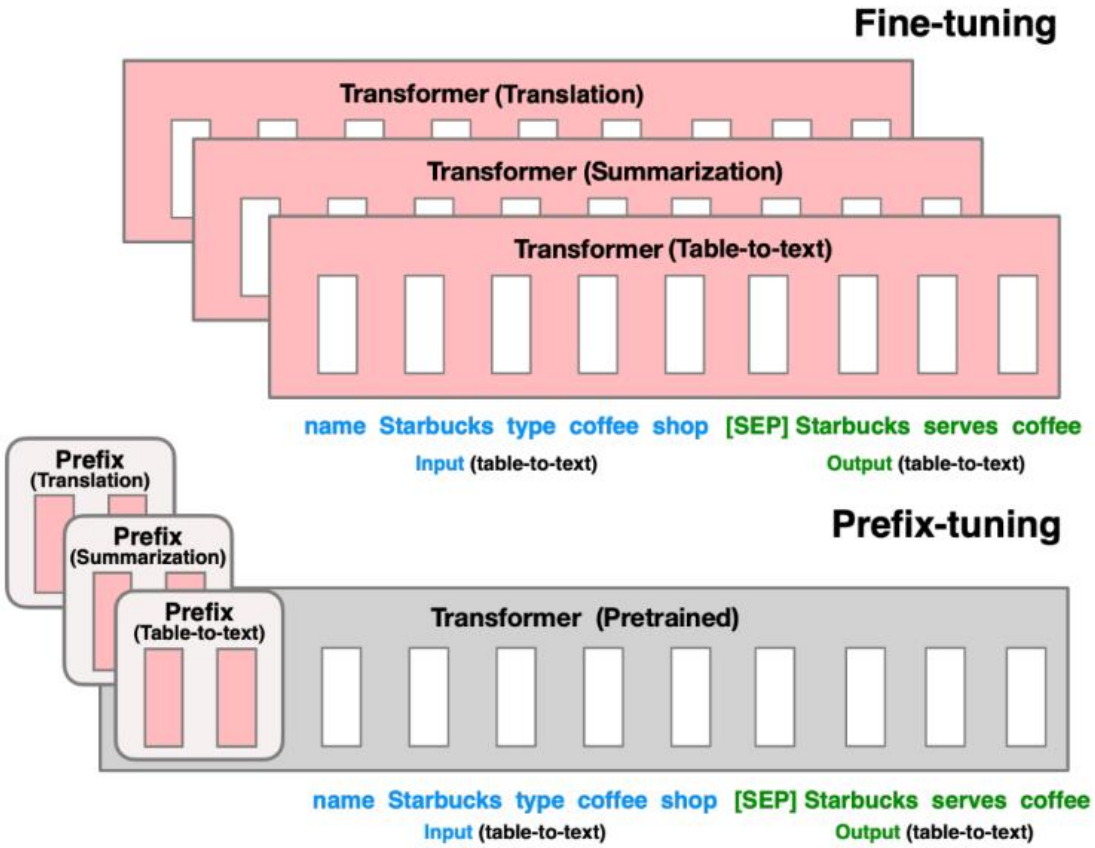
(c) Pre-training method



(d) LM adaptation steps

- Prompt参数长度超过20后，模型整体效果提升不明显
- 采用Class Label初始化Prompt参数效果比较好（从下游任务中取label对应的 token中取。）

Prefix Tuning



Prefix-Tuning方法在每个Transformer模块输入中加入Prefix Learnable Parameter,更好的学习提示表征

Efficiency of Prefix Tuning

	E2E					WebNLG									DART					
	BLEU	NIST	MET	R-L	CIDEr	BLEU			MET			TER ↓			BLEU	MET	TER ↓	Mover	BERT	BLEURT
						S	U	A	S	U	A	S	U	A						
GPT-2$_{\text{MEDIUM}}$																				
FINE-TUNE	68.2	8.62	**46.2**	71.0	2.47	**64.2**	27.7	46.5	**0.45**	0.30	0.38	**0.33**	0.76	0.53	46.2	**0.39**	**0.46**	**0.50**	**0.94**	**0.39**
FT-TOP2	68.1	8.59	46.0	70.8	2.41	53.6	18.9	36.0	0.38	0.23	0.31	0.49	0.99	0.72	41.0	0.34	0.56	0.43	0.93	0.21
ADAPTER(3%)	68.9	8.71	46.1	71.3	2.47	60.4	**48.3**	54.9	0.43	**0.38**	**0.41**	0.35	**0.45**	**0.39**	45.2	0.38	**0.46**	**0.50**	**0.94**	**0.39**
ADAPTER(0.1%)	66.3	8.41	45.0	69.8	2.40	54.5	45.1	50.2	0.39	0.36	0.38	0.40	0.46	0.43	42.4	0.36	0.48	0.47	**0.94**	0.33
PREFIX(0.1%)	**69.7**	**8.81**	46.1	**71.4**	**2.49**	62.9	45.6	**55.1**	0.44	**0.38**	**0.41**	0.35	0.49	0.41	**46.4**	0.38	**0.46**	**0.50**	**0.94**	**0.39**
GPT-2$_{\text{LARGE}}$																				
FINE-TUNE	68.5	8.78	46.0	69.9	2.45	**65.3**	43.1	55.5	**0.46**	0.38	**0.42**	**0.33**	0.53	0.42	**47.0**	**0.39**	0.46	**0.51**	**0.94**	**0.40**
Prefix	**70.3**	**8.85**	**46.2**	**71.7**	**2.47**	63.4	**47.7**	**56.3**	0.45	**0.39**	**0.42**	0.34	**0.48**	**0.40**	46.7	**0.39**	**0.45**	**0.51**	**0.94**	**0.40**
SOTA	68.6	8.70	45.3	70.8	2.37	63.9	52.8	57.1	0.46	0.41	0.44	-	-	-	-	-	-	-	-	-

只需要约0.1%的训练参数，即可在E2E、WebNLG和DART三个table-to-text任务上取得比全量微调更好的效果。

Advantages of Prefix/Prompt Tuning

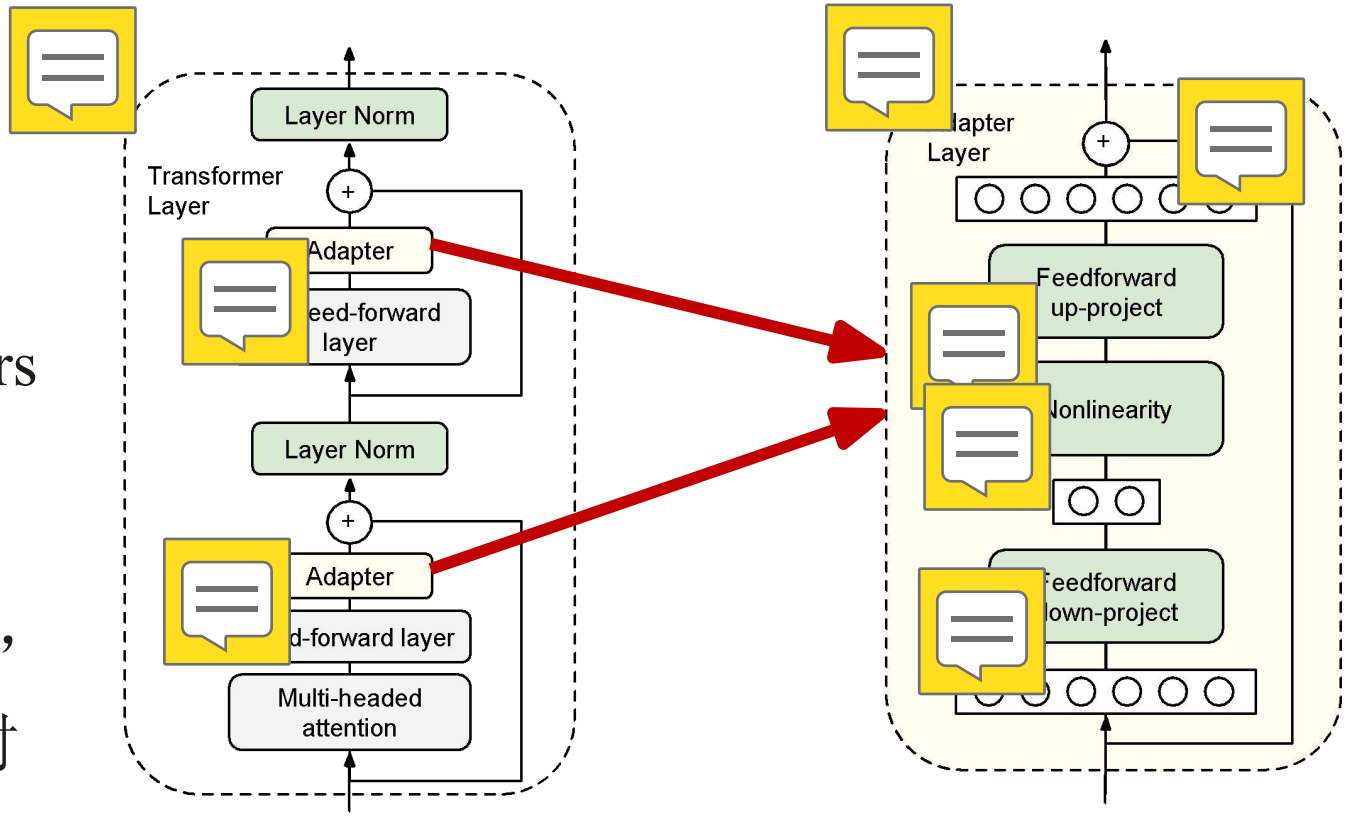
- 需要学习的参数量非常少，占总参数量的0.1%不到。
  - 即使每个任务训练相应的Prompt 参数，总的参数量也很低。
- 预训练的LLM冻结，只需要训练很少的参数，因此训练显存占用少，可以在消费级显卡上训练较大模型。

Pitfalls of Prefix and Prompt Tuning

- **可解释性差：**有论文研究过训练后的参数，发现与Prompt相似度最高的词汇组合起来毫无意义。
- **收敛更慢:** 更少的参数想要挖掘更大模型的知识，需要更复杂的空间搜索，相比全量微调收敛更慢。
- 针对不同任务，最适合的Prompt长度不确定
- **微调可能存在不稳定性：**Prompt-tuning和P-tuning的GitHub里都有提到结果在SuperGLUE上无法复现的问题；

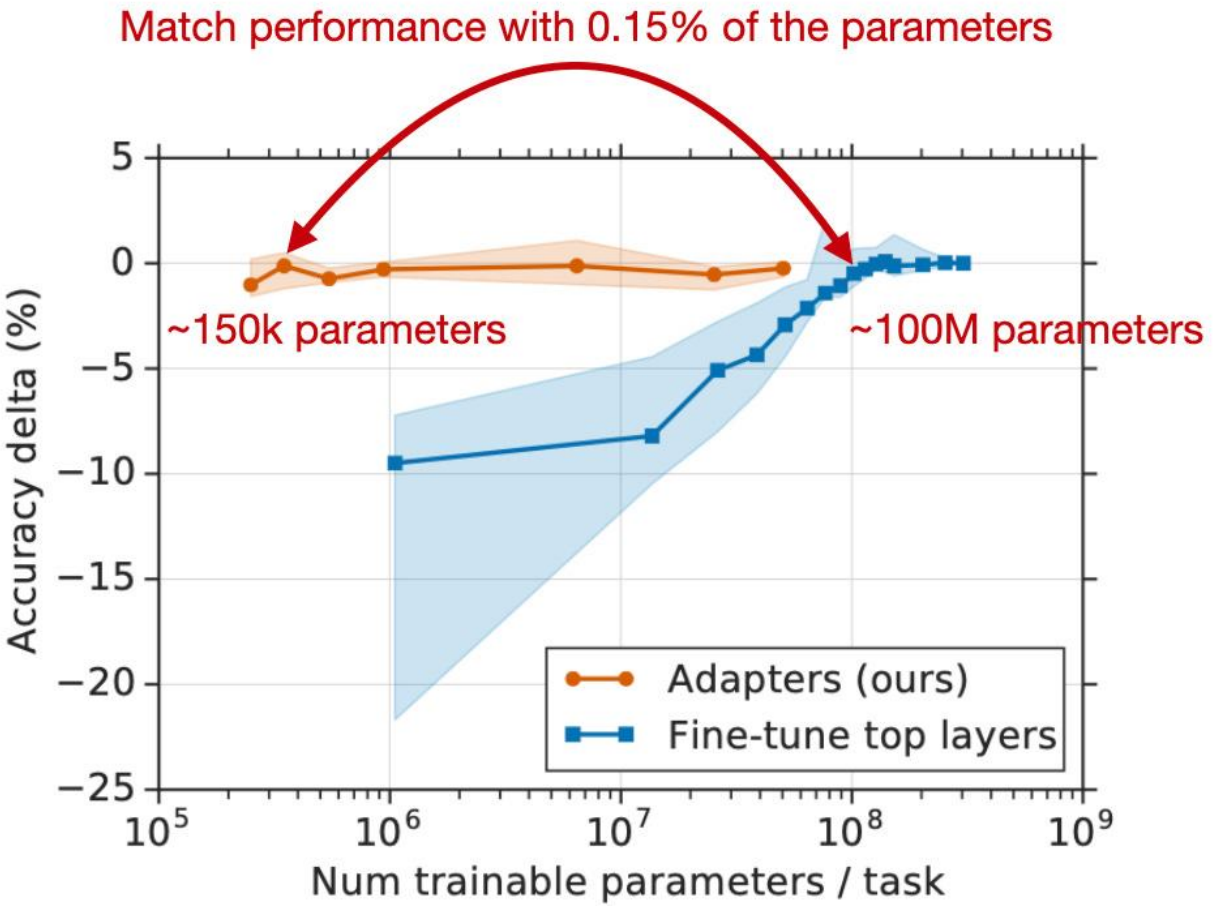
Adapters

- Adapters被加到预训练Transformer Block中间。
- 原始模型权重固定，只训练Adapters模块和Layer Norm层。
- Adapters模块包括一个down-project，一个激活函数，一个up-project,同时残差连接。



$$Adapter(x) = W_{\text{up}}\sigma(W_{\text{down}}x) + x.$$

Adapters



Advantages of Adapter-Based Methods

- 在多任务学习中被证明非常有效，每个任务训练特定的Adapters，然后将他们结合起来解决跨任务的问题。
- 相比于全量微调，收敛更快。
- 相比于全量微调模型，对应微调数据的鲁棒性更强。

Pitfalls or Adapter Methods

- 增强Adapters模块会导致推理速度变慢。
- 加入Adapters模块模型变大，会导致模型出现显存不足的情况。
- 会影响模型的训练推理的并行。

Prefix Tuning in MindSpore NLP

```python
if self.prefix_projection and not config.inference_mode:
    # Use a two-layer MLP to encode the prefix
    self.embedding = nn.Embedding(num_virtual_tokens, token_dim)
    self.transform = nn.Sequential(
        nn.Linear(token_dim, encoder_hidden_size),
        nn.Tanh(),
        nn.Linear(encoder_hidden_size, num_layers * 2 * token_dim),
    )
else:
    self.embedding = nn.Embedding(num_virtual_tokens, num_layers * 2 * token_dim)
```

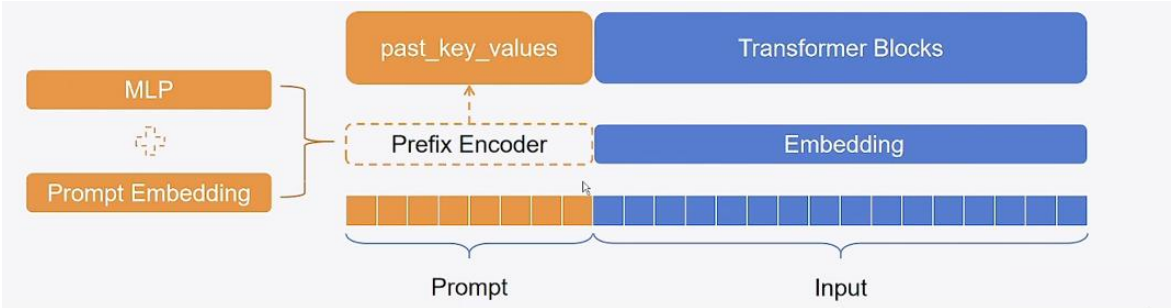
参数初始化：Prefix Encoder模块，为防止直接更新Prefix参数导致训练不稳定，在Prefix层前面加上MLP结果，训练完成只保留Prefix参数。

```python
if self.prefix_projection:
    prefix_tokens = self.embedding(prefix)
    past_key_values = self.transform(prefix_tokens)
else:
    past_key_values = self.embedding(prefix)
return past_key_values
```

得到past_key_values

```python
key_states = project(
    hidden_states, self.k, key_value_states, past_key_value[0]
)
value_states = project(
    hidden_states, self.v, key_value_states, past_key_value[1]
)
if past_key_value is not None:
    if key_value_states is None:
        # self-attn
        # (batch_size, n_heads, key_length, dim_per_head)
        hidden_states = ops.cat([past_key_value, hidden_states], dim=2)
```

在每一层的transformer block的key value前面，拼接past_key_value



在每一层Transformer Block中加入可学习的输入参数

Selective PEFT

Methods: layer freezing, BitFit, diff pruning

Layer Freezing

- 研究表明，Transformer模型的较早层次倾向于捕捉语言现象和基础语言理解；而较后层次则是进行特定任务学习的地方。
- 这意味着我们可以通过冻结较早层次，仅调整较后层次来学习新任务。

BitFit: Bias-terms Fine-tuning

只对模型的bias进行微调，在大模型中，bias主要存在于Q,K,V,MLP,LayerNorm中。

$$\mathbf{Q}^{m,\ell}(\mathbf{x}) = \mathbf{W}_q^{m,\ell}\mathbf{x} + \mathbf{b}_q^{m,\ell}$$
$$\mathbf{K}^{m,\ell}(\mathbf{x}) = \mathbf{W}_k^{m,\ell}\mathbf{x} + \mathbf{b}_k^{m,\ell}$$
$$\mathbf{V}^{m,\ell}(\mathbf{x}) = \mathbf{W}_v^{m,\ell}\mathbf{x} + \mathbf{b}_v^{m,\ell}$$

$$\mathbf{h}_2^\ell = \text{Dropout}\left(\mathbf{W}_{m_1}^\ell \cdot \mathbf{h}_1^\ell + \mathbf{b}_{m_1}^\ell\right) \quad (1)$$
$$\mathbf{h}_3^\ell = \mathbf{g}_{LN_1}^\ell \odot \frac{(\mathbf{h}_2^\ell + \mathbf{x}) - \mu}{\sigma} + \mathbf{b}_{LN_1}^\ell \quad (2)$$
$$\mathbf{h}_4^\ell = \text{GELU}\left(\mathbf{W}_{m_2}^\ell \cdot \mathbf{h}_3^\ell + \mathbf{b}_{m_2}^\ell\right) \quad (3)$$
$$\mathbf{h}_5^\ell = \text{Dropout}\left(\mathbf{W}_{m_3}^\ell \cdot \mathbf{h}_4^\ell + \mathbf{b}_{m_3}^\ell\right) \quad (4)$$
$$\text{out}^\ell = \mathbf{g}_{LN_2}^\ell \odot \frac{(\mathbf{h}_5^\ell + \mathbf{h}_3^\ell) - \mu}{\sigma} + \mathbf{b}_{LN_2}^\ell \quad (5)$$

	Train size	%Param	QNLI 105k	SST-2 67k	MNLI_m 393k	MNLI_mm 393k	CoLA 8.5k	MRPC 3.7k	STS-B 7k	RTE 2.5k	QQP 364k	Avg.
(V)	Full-FT†	100%	**93.5**	**94.1**	**86.5**	**87.1**	**62.8**	**91.9**	89.8	71.8	**87.6**	**84.8**
(V)	Full-FT	100%	91.7±0.1	93.4±0.2	85.5±0.4	85.7±0.4	62.2±1.2	90.7±0.3	**90.0±0.4**	**71.9±1.3**	87.5±0.4	84.1
(V)	Diff-Prune†	0.5%	**93.4**	**94.2**	**86.4**	**86.9**	63.5	91.3	89.5	71.5	**86.6**	**84.6**
(V)	BitFit	0.08%	91.4±2.4	93.2±0.4	84.4±0.2	84.8±0.1	**63.6±0.7**	**91.7±0.5**	**90.3±0.1**	**73.2±3.7**	85.4±0.1	84.2
(T)	Full-FT‡	100%	91.1	94.1	86.7	**86.0**	59.6	88.9	86.6	**71.2**	71.7	81.2
(T)	Full-FT†	100%	**93.4**	**94.9**	86.7	85.9	**60.5**	**89.3**	**87.6**	70.1	**72.1**	**81.8**
(T)	Adapters‡	3.6%	90.7	94.0	84.9	85.1	59.5	89.5	**86.9**	71.5	**71.8**	81.1
(T)	Diff-Prune†	0.5%	**93.3**	94.1	**86.4**	**86.0**	**61.1**	**89.7**	86.0	70.6	71.1	**81.5**
(T)	BitFit	0.08%	92.0	**94.2**	84.5	84.8	59.7	88.9	85.5	**72.0**	70.5	80.9

在Bert-Base/Bert-Large这种模型里，bias参数仅占模型全部参数量的0.08%～0.09%。在GLUE数据集上对Bert-Large的bias参数进行BitFit微调，能够在参数量远小于Adapter、Diff-Pruning的情况下，效果与Adapter、Diff-Pruning想当，甚至在某些任务上略优于Adapter、Diff-Pruning。

Intuition for DiffPruning

在先前讨论的方法中，我们手动决定哪些参数应该保持不变，哪些参数应该进行调整。为什么不通过学习来决定这个选择呢？

$$\theta_\tau = \theta + \delta_\tau$$

Diff pruning 将特定任务的微调表述为学习一个 diff 向量 δτ，该向量被添加到预先训练的模型参数θ中，该参数保持固定。我们首先对特定任务的模型参数进行重新参数化， diff向量用L0-norm惩罚的可微近似值进行重构。

BitFit in MindSpore NLP

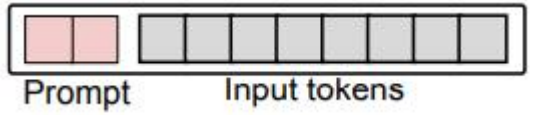
```
for n, p in model.named_parameters():
    if "bias" not in n :
        p.requires_grad = False
    else :
        p.requires_grad = True
```

冻结除bias外的所有参数，训练时只更新bias参数

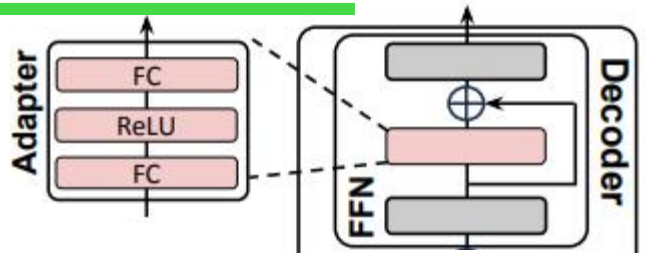
Reparameterization PEFT

Methods: LoRA, AdaLoRA,$(IA)^3$

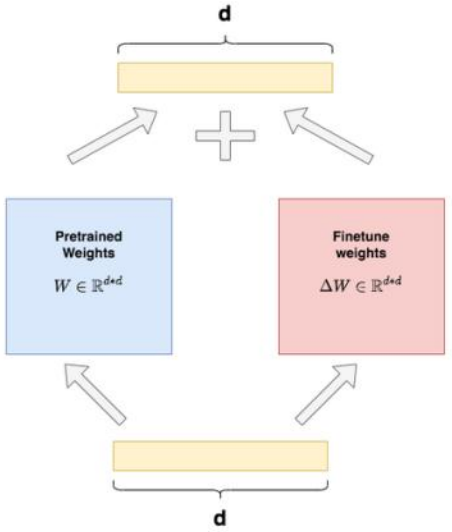
LoRA - Low-Rank Adaptation



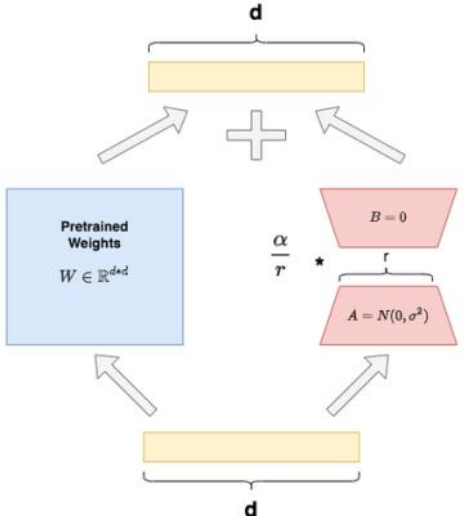
Prompt Tuning方法难以训练



Adapter方法推理变慢



全量微调



LoRA

$$h = W_0 x + \Delta W x = W_0 x + BAx$$

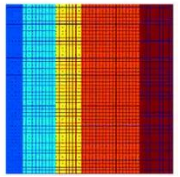
$$W_0 \in \mathbb{R}^{d \times k} \quad B \in \mathbb{R}^{d \times r} \quad A \in \mathbb{R}^{r \times k}$$

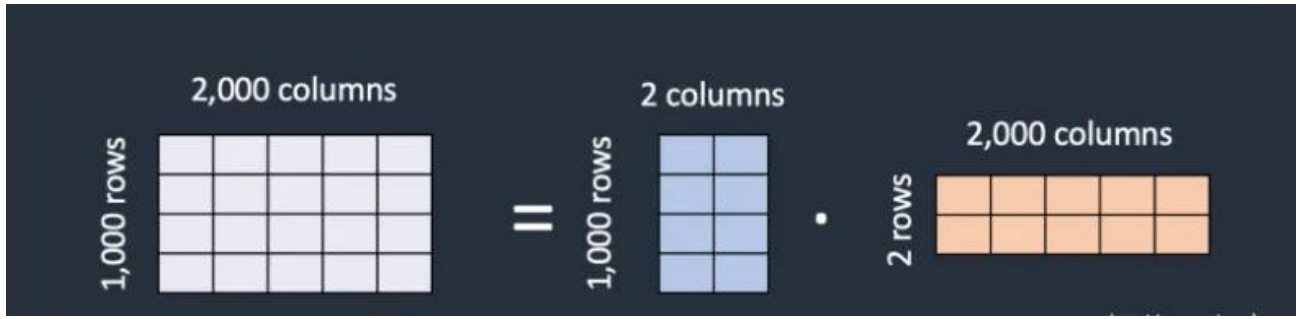
[M]S 昇思 MindSpore

LoRA

*The data should be **low-dimensional (low-rank)**:*

$$A = [\mathbf{a}_1 \mid \cdots \mid \mathbf{a}_n] \in \mathbb{R}^{m \times n}, \quad \operatorname{rank}(A) \ll m.$$





$$h = W_0 x + \Delta W x = W_0 x + BAx$$

$$W_0 \in \mathbb{R}^{d \times k} \quad B \in \mathbb{R}^{d \times r} \quad A \in \mathbb{R}^{r \times k}$$

有相关工作证明，微调过程中产生的权重更新量是低秩的。LoRA就假设模型在下游任务上微调得到的增量参数矩阵ΔW是低秩的。

假设假设模型有一个具有 1,000 行和 2,000 列的矩阵。这就是要在模型文件中存储的 2,000,000 个数字（1,000 x 2,000）。LoRA 将该矩阵分解为一个 1,000x2 的矩阵和一个 2x2,000 的矩阵。这只需要 6,000 个数字（1,000 x 2 + 2 x 2,000），是原始矩阵大小的 333 倍。这就是 LoRA 参数数量小的原因。

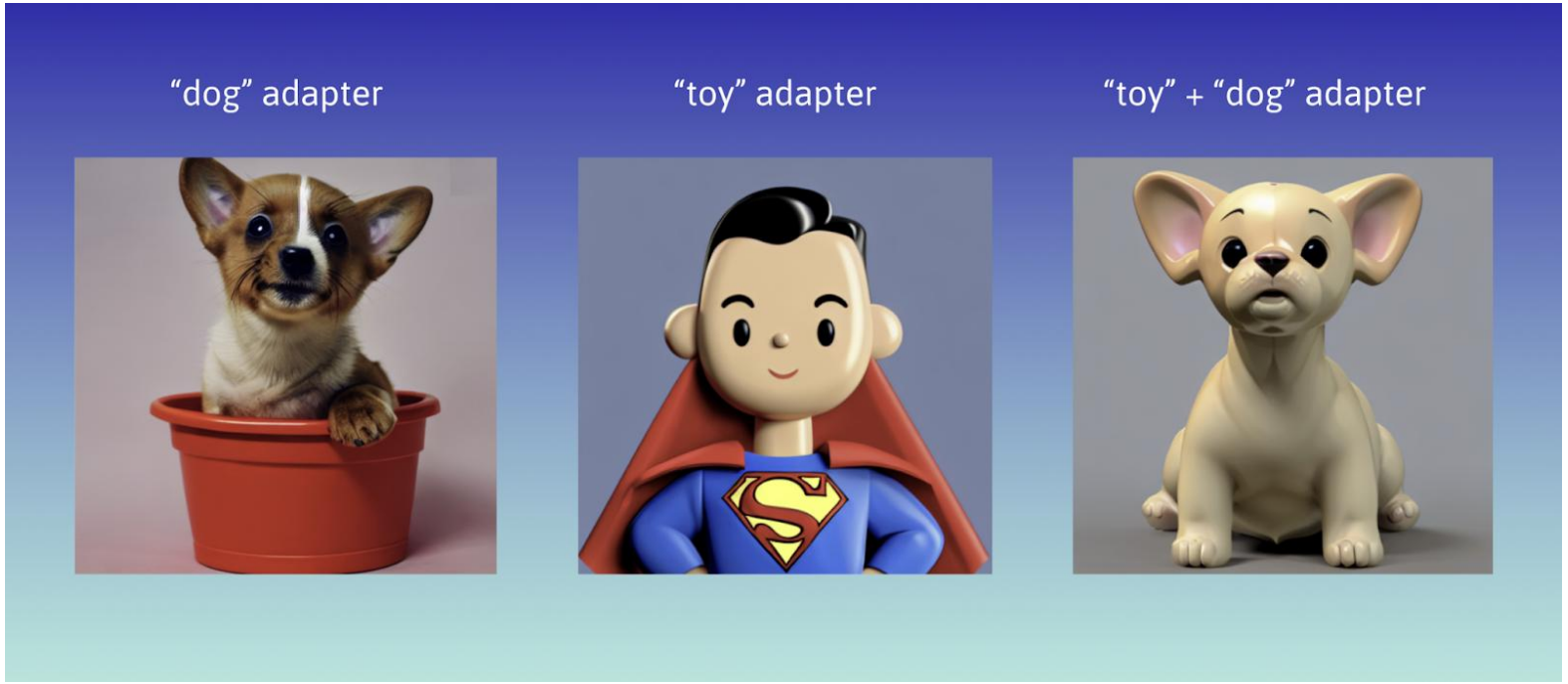
Efficiency of LoRA

Model&Method	# Trainable Parameters	WikiSQL Acc. (%)	MNLI-m Acc. (%)	SAMSum R1/R2/RL
GPT-3 (FT)	175,255.8M	**73.8**	89.5	52.0/28.0/44.5
GPT-3 (BitFit)	14.2M	71.3	91.0	51.3/27.4/43.5
GPT-3 (PreEmbed)	3.2M	63.1	88.6	48.3/24.2/40.5
GPT-3 (PreLayer)	20.2M	70.1	89.5	50.8/27.3/43.5
GPT-3 ($\text{Adapter}^{\text{H}}$)	7.1M	71.9	89.8	53.0/28.9/44.8
GPT-3 ($\text{Adapter}^{\text{H}}$)	40.1M	73.2	**91.5**	53.2/29.0/45.1
GPT-3 (LoRA)	4.7M	73.4	**91.7**	**53.8/29.8/45.9**
GPT-3 (LoRA)	37.7M	**74.0**	**91.6**	53.4/29.2/45.1

Table 4: Performance of different adaptation methods on GPT-3 175B. We report the logical form validation accuracy on WikiSQL, validation accuracy on MultiNLI-matched, and Rouge-1/2/L on SAMSum. LoRA performs better than prior approaches, including full fine-tuning. The results on WikiSQL have a fluctuation around $\pm 0.5\%$, MNLI-m around $\pm 0.1\%$, and SAMSum around $\pm 0.2/\pm 0.2/\pm 0.1$ for the three metrics.

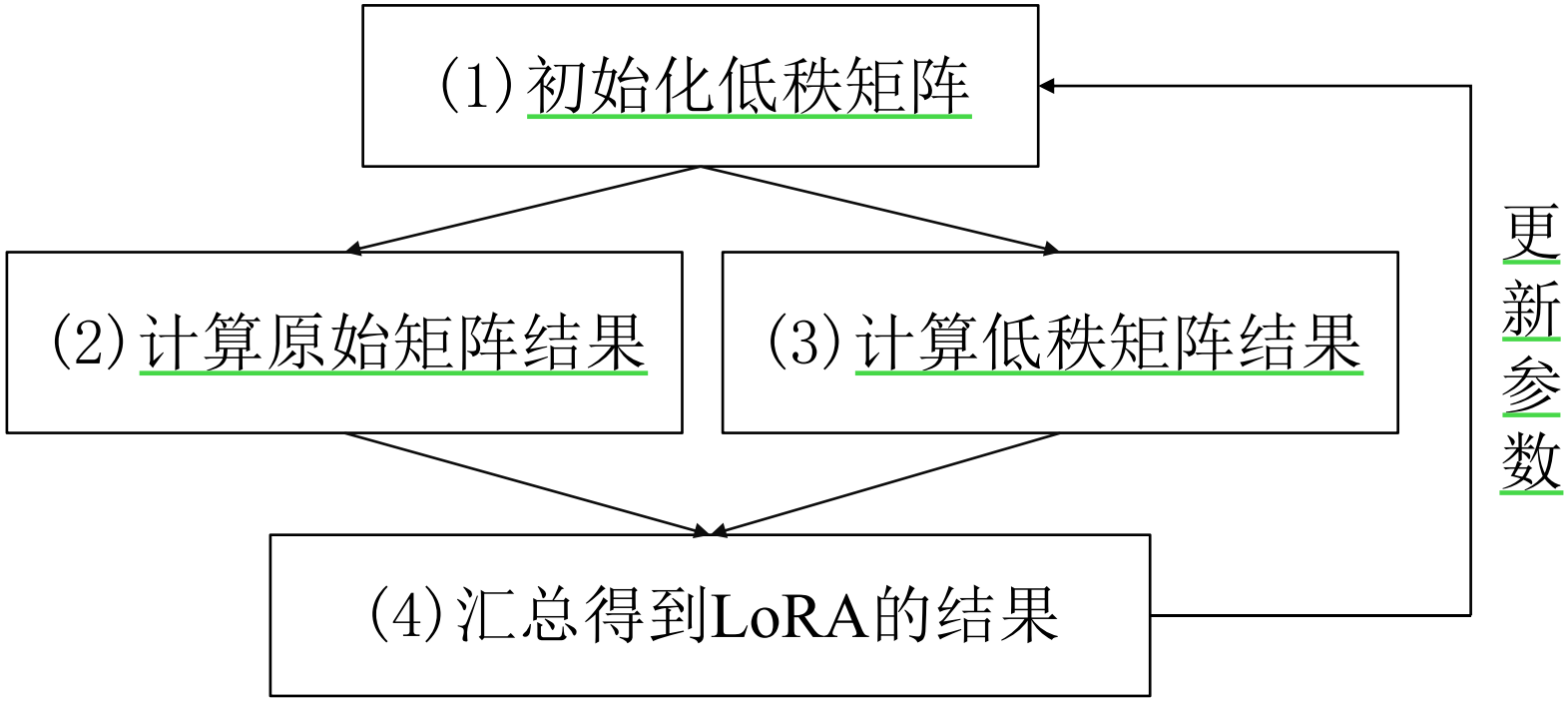
LoRA在训练参数在训练参数不到0.1%的情况下，超过了全量微调的效果

LoRA in Stable Diffusion



在Stable Diffusion（SD）模型的应用中，LoRA被用作一种插件，允许用户在不修改SD模型的情况下，利用少量数据训练出具有特定画风、IP或人物特征的模型。这种技术在社区使用和个人开发者中非常受欢迎。例如，可以通过LoRA模型改变SD模型的生成风格，或者为SD模型添加新的人物/IP。

LoRA in MindSpore NLP

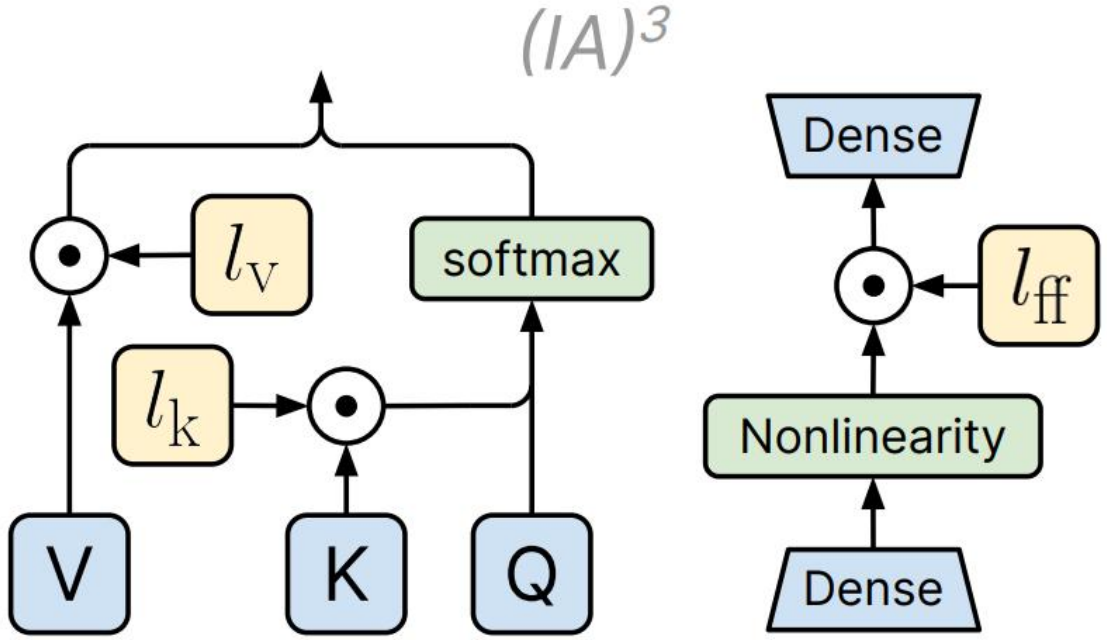


(1)
```
self.lora_A[adapter_name] = nn.Linear(self.in_features, r, bias=False)
self.lora_B[adapter_name] = nn.Linear(r, self.out_features, bias=False)
```

(2)
```
result = self.base_layer(x, *args, **kwargs)
```

(3)+(4)
```
result = result + lora_B(lora_A(dropout(x))) * scaling
```

$(IA)^3$ - Infused Adapter by Inhibiting and Amplifying Inner Activations



$$\text{softmax}\left(\frac{Q(l_k \odot K^T)}{\sqrt{d_k}}\right)(l_v \odot V) \qquad (l_{ff} \odot \gamma(W_1 x))W_2$$

ia3_l向量与attention和mlp的weight相乘，然后得到的权重再与输入相乘，原来的权重都保持冻结，只训练这部分权重

Efficiency of IA3

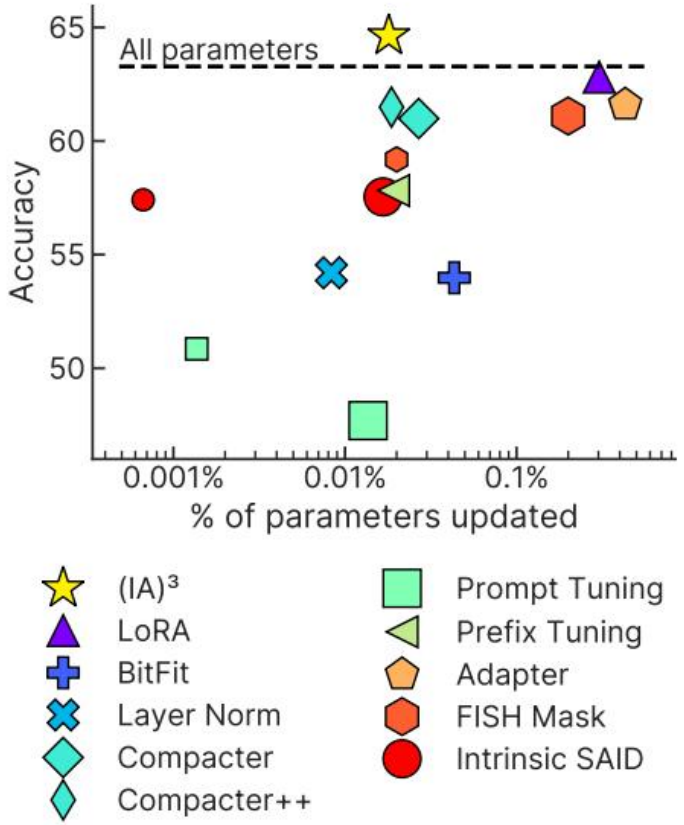


Figure 2: Accuracy of PEFT methods with L_{UL} and L_{LN} when applied to T0-3B. Methods that with variable parameter budgets are represented with larger and smaller markers for more or less parameters.

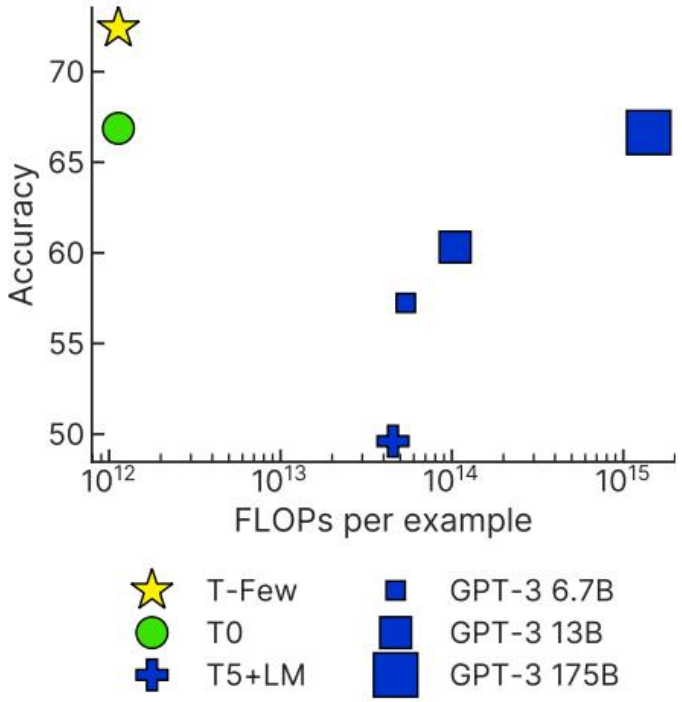
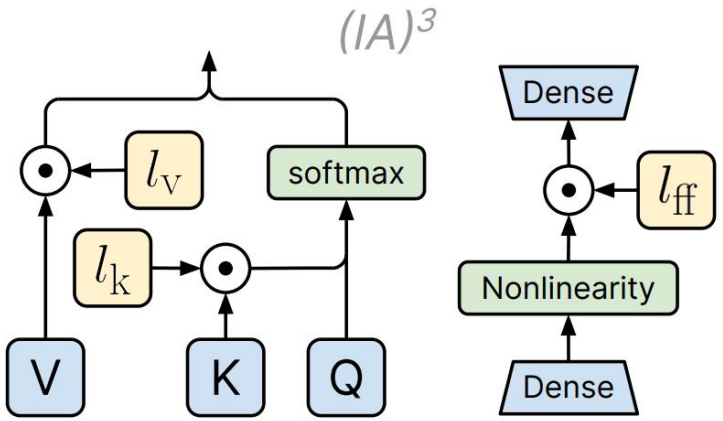


Figure 3: Accuracy of different few-shot learning methods. `T-Few` uses `(IA)`^{3} for PEFT methods of T0, T0 uses zero-shot learning, and T5+LM and the GPT-3 variants use few-shot ICL. The x-axis corresponds to inference costs; details are provided in section 4.2.

$(IA)^3$




```python
def update_layer(self, adapter_name, init_ia3_weights):
    # This code works for linear layers, override for other layer
    # Actual trainable parameters
    if self.is_feedforward:
        weight = torch.randn((1, self.in_features))
    else:
        weight = torch.randn((self.out_features, 1))
    self.ia3_l[adapter_name] = nn.Parameter(weight)
    if init_ia3_weights:
        self.reset_ia3_parameters(adapter_name)
    self.to(self.get_base_layer().weight.device)
    self.set_adapter(self.active_adapters)
```

IA3向量初始化

```python
"""
adapter_names = check_adapters_to_merge(self, adapter_names)
if not adapter_names:
    # no adapter to merge
    return

for active_adapter in adapter_names:
    if active_adapter in self.ia3_l.keys():
        base_layer = self.get_base_layer()
        ia3_l = transpose(self.ia3_l[active_adapter].data, self.fan_in_fan_out)
        orig_dtype = base_layer.weight.data.dtype
        if safe_merge:
            orig_weights = base_layer.weight.data
            orig_weights = torch.mul(orig_weights, ia3_l)
```

IA3权重与原来权重相乘

Advantages of Re-Paramaterization Methods

- 训练往往更加节省内存，因为我们只需要计算少数参数的梯度并更新参数。
- 相比全量微调收敛更快
- 通过简单地替换调优后的权重，就可以直接在任务之间切换。

Homework

作业要求：在华为云平台上，使用MindSpore NLP组件对Roberta-Large模型进行IA3微调训练。

数据集　：　GLUE-MRPC

具体要求：使用MindSpore NLP组件加载Roberta-Large模型，设置IA3算法配置并初始化微调模型，加载数据集进行微调训练，并最终使用微调后的模型在验证集上进行评估。