

# LLM的垂直化技术 - PEFT与RAG

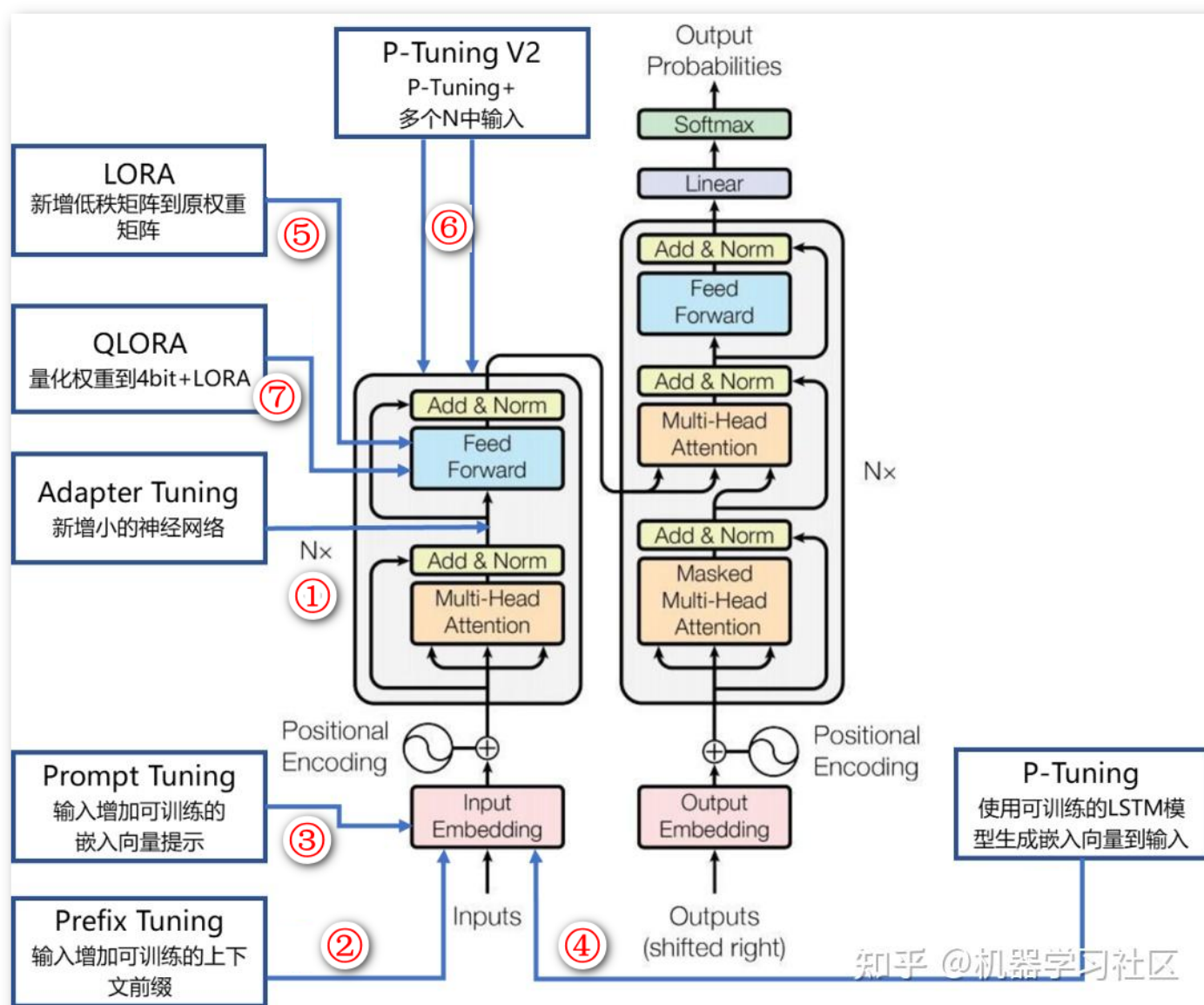
# 1 PEFT

参考资料:

[大模型微调 \(finetune\) 方法总结 - 知乎 \(zhihu.com\)](#)

[一文搞清楚LORA、Prompt Tuning、P-Tuning、Adapter 、Prefix等大模型微调方法 - 知乎 (zhihu.com)] 1 微调

**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。PEFT 包括**LORA**、**QLoRA**、**Adapter Tuning**、**Prefix Tuning**、**Prompt Tuning**、**P-Tuning**及**P-Tuning v2**等。

这些PEFT技术出现的时间轴如下:

## 第一阶段

Adapter Tuning的出现开启了PEFT技术的大门。

该方法首次提出了在预训练模型中插入额外的可训练模块(即适配器)的思路,并证明了仅微调这些附加模块就能取得良好的迁移学习效果。这一阶段奠定了PEFT技术的基础,为后续的发展提供了重要的启示。

### ① Adapter Tuning [2019]:

Adapter Tuning是最早出现的PEFT技术之一,由Houlsby等人在2019年的论文《Parameter-Efficient Transfer Learning for NLP》中首次提出。该方法通过在预训练模型的每一层中插入可训练的适配器模块,并仅微调这些适配器参数,从而实现参数高效的迁移学习。

## 第二阶段

这一阶段涌现出了多种基于提示的PEFT技术,包括Prefix Tuning、Prompt Tuning和P-Tuning。

这些方法的共同点是在预训练模型的输入端或每一层前面添加可学习的提示向量,并只微调这些提示参数。与Adapter Tuning相比,提示类方法**进一步减少了可训练参数的数量**,并展现出了优异的性能。

### ② Prefix Tuning [2021]:

2021年,Li和Liang在论文《Prefix-Tuning: Optimizing Continuous Prompts for Generation》中提出了Prefix Tuning。该方法在预训练模型的每一层前面添加一个连续的向量(即前缀),并在微调过程中只优化这些前缀参数。

### ③ Prompt Tuning [2021]:

Prompt Tuning与Prefix Tuning同年由Lester等人在论文《The Power of Scale for Parameter-Efficient Prompt Tuning》中提出。Prompt Tuning在输入端添加可学习的嵌入向量(即软提示),并只微调这些软提示参数,预训练模型的参数保持不变。

### ④ P-Tuning [2021]:

2021年,Liu等人在论文《GPT Understands, Too》中提出了P-Tuning。该方法在每个输入样本之前插入可学习的连续提示,并在微调过程中只优化这些提示参数。P-Tuning可以看作是Prompt Tuning的一种变体。

## 第三阶段

LORA、P-Tuning v2和QLoRA的出现标志着PEFT技术进入了一个新的阶段。

LORA通过在预训练模型的权重矩阵上添加低秩分解矩阵,实现了参数的高效微调。P-Tuning v2对第二阶段的P-Tuning进行了改进,引入了深度提示编码器和优化的初始化策略。QLoRA则在LORA的基础上结合了量化技术,进一步压缩了微调参数。

#### ⑤ LORA [2021]:

LORA(Low-Rank Adaptation)由Hu等人在2021年的论文《LoRA: Low-Rank Adaptation of Large Language Models》中提出。LORA在预训练模型的权重矩阵上添加低秩分解矩阵,并在微调过程中只训练这些低秩矩阵,从而显著减少可训练参数的数量。

#### ⑥ P-Tuning v2 [2022]:

2022年,Liu等人在论文《P-Tuning v2: Prompt Tuning Can Be Comparable to Fine-tuning Universally Across Scales and Tasks》中对P-Tuning进行了改进,提出了P-Tuning v2。该方法通过引入深度提示编码器和改进的初始化策略,进一步提高了P-Tuning的性能。

#### ⑦ QLoRA [2023]:

QLoRA是LORA的量化版本,由相同的作者在2023年的论文《QLoRA: Efficient Finetuning of Quantized LLMs》中提出。QLoRA在LORA的基础上引入了4-bit量化技术,可以进一步压缩微调参数,并加速推理过程。

## 1.1 Adapter Tuning

Adapter Tuning全称为"Adapter-based Fine-tuning",也简称为"Adapter"。

可以将Adapter Tuning比喻为为预训练模型戴上"任务眼镜"。预训练模型就像一个具备基础知识的学生,而适配器就像一副专门为特定任务定制的眼镜。戴上这副眼镜后,学生可以更好地理解和处理与任务相关的信息,而无需改变自身的基础知识。在训练过程中,我们只需要调整眼镜的焦距(适配器参数),使学生能够更清晰地看到任务的关键信息。同时,学生的基础知识(预训练模型参数)保持不变,可以在不同任务之间迁移和重用。这样,学生就可以高效、灵活地适应各种不同的任务,而不会因为专注于一个任务而忘记了基础知识。

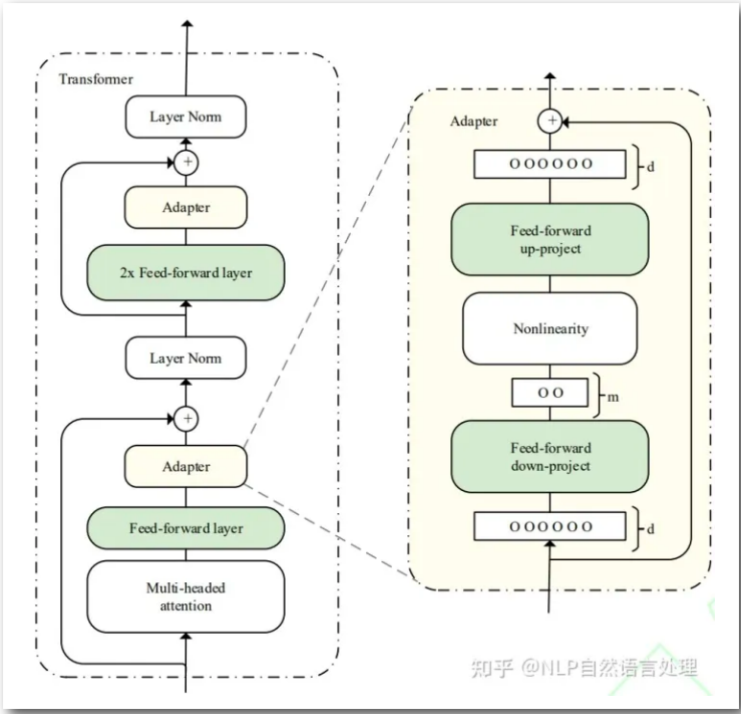
### 1.1.1 出现背景与解决的问题

在Adapter Tuning出现之前,迁移学习主要采用全量微调的方式,即在预训练模型的基础上,使用新任务的数据对所有参数进行微调。这种方法虽然有效,但存在两个主要问题:

- 其一,全量微调需要更新预训练模型的所有参数,计算和存储开销大;
- 其二,不同任务之间的适配能力有限,每个新任务都需要单独进行全量微调,缺乏通用性。

Adapter Tuning通过在预训练模型中插入轻量级的适配器模块,并只微调这些附加参数,从而在降低计算开销的同时,提高了模型的泛化能力和跨任务适配能力。

1.1.2 算法思路



Adapter Tuning的核心思想是在预训练模型的每一层中插入一个轻量级的适配器模块, 该模块由两个全连接层和一个残差连接组成。

具体而言,对于预训练模型的第*i*层,其输出表示为 $h_i$ , Adapter Tuning在其后插入一个适配器模块,计算公式如下:

$$z_i = W_{down}(ReLU(W_{up}h_i))$$
$$h'_i = h_i + z_i$$

其中, $W_{up}$ 和 $W_{down}$ 分别为上投影矩阵和下投影矩阵,ReLU为激活函数。通过这种方式,适配器模块可以学习到针对特定任务的特征表示,同时也能够保留预训练模型中已经学到的通用知识。在微调过程中,只需要更新适配器模块的参数,预训练模型的参数保持不变。

Q : Adapter 参数量 大概是多少?

Q : 这不就跟 LoRA 差不多了么?

1.1.3 微调数据组织与参数更新

在Adapter Tuning中,输入数据首先经过预训练模型的Embedding层得到词嵌入表示, 然后依次通过预训练模型的每一层和插入其中的适配器模块, 最后经过输出层得到预测结果。在训练过程中,损失函数(如交叉熵损失)的梯度只会回传到适配器模块的参数, 通过反向传播算法更新这些参数, 而预训练模型的参数保持不变。

Adapter Tuning通过引入轻量级的适配器模块,实现了参数高效的迁移学习。这一技术的提出,为后续一系列PEFT方法的发展奠定了重要的基础。

# 1.2 Prefix Tuning

Prefix Tuning（前缀微调）是一种参数高效的微调技术，由Li和Liang在2021年提出。它通过在每个Transformer层的输入端添加可学习的连续前缀向量（continuous prefix vectors），在微调过程中只优化这些前缀，而保持预训练模型的参数不变，从而实现了对下游任务的适应。

可以将Prefix Tuning比喻为一种“即插即用”的适配器。假设我们有一台通用的机器(预训练语言模型),它具有强大的功能(如语言理解、文本生成等),但缺乏专门针对特定任务(如情感分类、问答等)的专业技能。为了让这台机器能够胜任特定任务,我们可以为其设计一个专用的“插件”(前缀向量)。这个插件不会改变机器的内部结构和原有功能,但可以通过与机器的协同工作,使其在特定任务上表现出色。

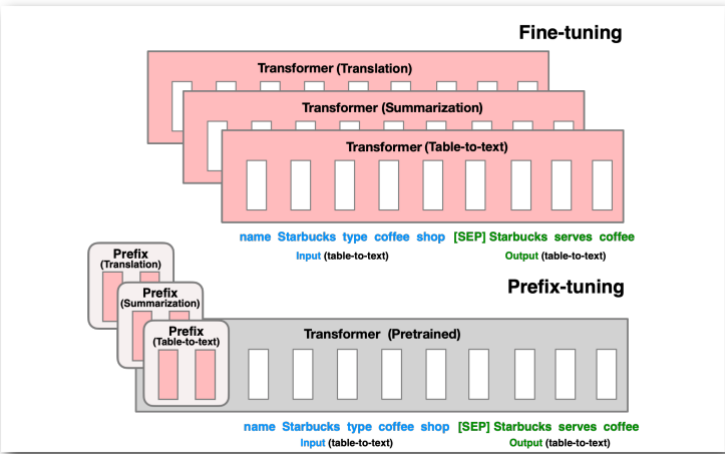
## 1.2.1 出现背景与解决的问题

在Prefix Tuning出现之前，微调预训练语言模型主要有两种方式：全参数微调和Adapter Tuning。

- 全参数微调需要调整模型的所有参数，计算和存储成本高，而且在小型数据集上容易过拟合。
- Adapter Tuning通过在每个Transformer层中添加适配器模块，只微调适配器参数，降低了计算和存储成本，提高了微调效率。然而，Adapter Tuning仍然需要**为每个任务单独添加和训练适配器**，灵活性和可扩展性有限。

Prefix Tuning 通过引入可学习的前缀向量，进一步简化了微调过程。它在输入端添加前缀，而不需要修改预训练模型的架构，使得微调更加灵活和高效。同时，前缀向量的参数量远小于适配器和全模型参数，进一步降低了计算和存储开销。

## 1.2.2 算法思路



Prefix Tuning的核心思想是在每个Transformer层的输入端引入可学习的连续前缀向量。

具体而言，对于一个有 $L$ 层的预训练语言模型，Prefix Tuning为**每一层**分配一个长度为 $P$ 的前缀向量  $\mathbf{P}_l \in \mathbb{R}^{P \times d}$ ，其中 $l \in \{1, 2, \dots, L\}$ ， $d$ 为模型的隐藏状态维度。



在微调过程中，输入序列  $\mathbf{X} \in \mathbb{R}^{N \times d}$  ( $N$ 为序列长度) 与对应层的前缀向量  $\mathbf{P}_l$  拼接，形成新的输入序列  $\mathbf{X}'_l = [\mathbf{P}_l; \mathbf{X}] \in \mathbb{R}^{(P+N) \times d}$ 。然后，将拼接后的序列  $\mathbf{X}'_l$  输入到相应的Transformer层中进行前向传播：

$$\mathbf{H}_l = \text{Transformer}_l(\mathbf{X}'_l)$$

其中  $\mathbf{H}_l \in \mathbb{R}^{(P+N) \times d}$  为第  $l$  层的输出。

在训练过程中，冻结预训练模型的所有参数，只更新前缀向量  $\mathbf{P}_l$ 。

梯度反向传播时，只计算并更新前缀向量的梯度，而预训练模型的参数保持不变。

### 1.2.3 微调数据组织与参数更新

在Prefix Tuning中，微调数据的组织与常规微调方法类似。

对于每个任务，准备相应的输入序列和标签，并将其传递给添加了前缀向量的预训练模型。

在训练过程中，通过以下步骤更新前缀向量的参数：

- 前向传播：将输入序列与对应层的前缀向量拼接，然后通过预训练模型的Transformer层，计算损失函数。
- 反向传播：根据损失函数计算前缀向量的梯度，并使用优化算法（如Adam）更新前缀向量的参数。
- 冻结预训练模型参数：确保只更新前缀向量的参数，而预训练模型的参数保持不变。

Q：我考虑，这个微调过程中，模型可以学到新的信息么？或者充分学习到新的信息么？我的意思是，假设我需要在专用领域进行数据微调，这种方式能够调动的参数量比较少，可以使用这些参数更好的记住或者压缩我的“知识”么？

A：确实由于参数量的局限性，但我们可以通过下述方法进行改进：

- 增加前缀向量的长度：通过增加前缀向量的长度  $P$ ，可以提高前缀向量学习和存储新知识的能力。但是，过长的前缀向量可能会增加计算和存储开销，需要在性能和效率之间进行权衡。
- 结合其他微调技术：将Prefix Tuning与其他微调技术（如Adapter Tuning或Fine-tuning）相结合，可以在保持预训练模型稳定性的同时，提高模型学习新知识的能力。例如，可以在Prefix Tuning的基础上，再对模型的一部分参数进行微调，以更好地适应专业领域的需求。
- 领域数据预训练：在进行Prefix Tuning之前，可以先在大规模的领域数据上对预训练模型进行二次预训练（domain-specific pretraining）。这样，模型可以在Prefix Tuning之前就学习到更多的领域知识，减轻前缀向量的学习压力。
- 迁移学习：如果在相似领域上已经有了高质量的微调模型，可以考虑将其作为初始化参数，再进行Prefix Tuning。通过迁移学习，可以将已有模型中的领域知识传递给新模型，加速微调过程并提高性能。

## 1.3 Prompt Tuning

Prompt Tuning（提示微调）是由Lester等人在2021年提出的一种参数高效的微调技术。它通过在输入序列的开头添加可学习的软提示（soft prompts），在微调过程中只优化这些软提示，同时冻结预训练模型的所有参数，从而实现了对下游任务的适应。

可以将Prompt Tuning比喻为一种“软件插件”的开发过程。假设我们有一个功能强大的通用软件（预训练语言模型），它已经具备了处理各种任务的基本能力。但是，为了让这个软件更好地适应特定的任务（如情感分析、问答等），我们需要为其开发一些专门的插件（软提示向量）。

在开发插件的过程中，我们不需要修改软件的源代码（预训练模型的参数），而只需要编写插件的代码（优化软提示向量）。通过不断地调试和优化插件代码，我们可以使插件与软件更好地协同工作，从而在特定任务上取得更好的性能。一旦插件开发完成，我们就可以将其嵌入到软件中，使软件能够快速、高效地处理该任务。

值得注意的是，同一个软件可以搭配不同的插件，以适应不同的任务需求。这就像Prompt Tuning可以为同一个预训练模型开发多个软提示向量，使其能够在多个任务上取得良好的性能。这种灵活性和通用性，使得Prompt Tuning成为一种极具吸引力的参数高效微调范式。

### 1.3.1 出现背景与解决的问题

在探索预训练语言模型的适应和应用过程中，研究者们发现，通过设计恰当的文本提示(prompt)，可以在few-shot或even zero-shot的设定下，利用预训练模型完成各种自然语言处理任务。这种基于prompt的学习范式(prompt-based learning)展现出了预训练模型的强大能力和灵活性。然而，传统的prompt-based learning通常依赖于手工设计的离散文本提示，其性能受到prompt质量的限制，且难以适应不同的任务和领域。

Prompt Tuning的提出，正是为了解决上述问题。通过将离散的文本提示替换为连续的可学习嵌入向量，Prompt Tuning在保持预训练模型参数不变的情况下，实现了对模型的软提示(soft prompting)和连续调优。

在Prompt Tuning出现之前，微调预训练语言模型的主要方法有全参数微调、Adapter Tuning和Prefix Tuning。

- 全参数微调需要调整模型的所有参数，计算和存储成本高，且在小型数据集上容易过拟合。
- Adapter Tuning通过在每个Transformer层中添加适配器模块，只微调适配器参数，降低了计算和存储成本，但仍需为每个任务单独添加和训练适配器。
- Prefix Tuning通过在每个Transformer层的输入端添加可学习的前缀向量，进一步简化了微调过程，但前缀向量的长度需要针对不同任务进行调整，灵活性有限。



Prompt Tuning通过引入可学习的软提示，提供了一种更加灵活、高效的微调方式。软提示在输入序列的开头添加，作为任务描述或指令，引导模型生成符合任务要求的输出。与Prefix Tuning相比，Prompt Tuning **只需在输入端添加软提示**，避免了在每个Transformer层中引入额外的参数，进一步降低了计算和存储开销。

### 1.3.2 算法思路

Prompt Tuning的核心思想是在输入序列的开头添加一组可学习的软提示向量。具体而言，对于一个给定的任务，Prompt Tuning引入一组长度为 $M$ 的软提示向量 $\mathbf{P} \in \mathbb{R}^{M \times d}$ ，其中 $d$ 为模型的隐藏状态维度。

在微调过程中，软提示向量 $\mathbf{P}$ 与输入序列 $\mathbf{X} \in \mathbb{R}^{N \times d}$ （ $N$ 为序列长度）拼接，形成新的输入序列 $\mathbf{X}' = [\mathbf{P}; \mathbf{X}] \in \mathbb{R}^{(M+N) \times d}$ 。然后，将拼接后的序列 $\mathbf{X}'$ 输入到预训练模型中进行前向传播：

$$\mathbf{H} = \text{Pretrained\_Model}(\mathbf{X}')$$

其中 $\mathbf{H} \in \mathbb{R}^{(M+N) \times d}$ 为预训练模型的输出。

在训练过程中，冻结预训练模型的所有参数，只更新软提示向量 $\mathbf{P}$ 。梯度反向传播时，只计算并更新软提示向量的梯度，而预训练模型的参数保持不变。

### 1.3.3 微调数据组织与参数更新

在Prompt Tuning的微调过程中,我们需要为 **每个任务 准备相应的训练数据和软提示向量**。

- 对于分类任务,训练数据通常包括输入样本和对应的标签;
- 对于生成任务,训练数据则包括输入样本和期望的输出序列。

在准备软提示向量时,我们通常随机初始化一个固定长度的嵌入向量,其长度可以根据任务的复杂度和数据规模进行调整。

在训练过程中,我们将软提示向量与输入样本拼接,得到增广后的输入表示。然后,将增广后的输入送入预训练模型进行前向传播,并根据任务的类型(如分类、生成等)设计相应的损失函数。通过反向传播算法,我们可以计算损失函数对软提示向量参数的梯度,并使用优化算法(如Adam)对其进行更新。在整个微调过程中,预训练模型的参数保持不变,只有软提示向量的参数被优化。

在推理阶段,我们只需将**学习好的软提示向量**与测试样本拼接,并将增广后的输入送入微调后的模型,即可得到预测结果或生成结果。由于软提示向量的长度通常远小于预训练模型的参数量,因此Prompt Tuning在推理阶段的计算开销也相对较小。

与Prefix Tuning相比, Prompt Tuning只在输入端添加软提示,避免了在每个Transformer层中引入额外的参数,使得微调过程更加简洁和高效。同时,软提示的长度可以根据任务的需求进行灵活调整,提高了微调的适应性。

Q: 所以这种方法的知识注入能力一定很差, 因为其可调整的参数数量很少, 对么?

A: 您的观察非常正确。Prompt Tuning在注入新知识方面的能力确实有限,主要有以下原因:

1. 可训练参数量少: 软提示向量是Prompt Tuning中唯一可训练的参数,其参数量相比于预训练模型的参数量非常少。这限制了软提示向量学习和存储新知识的能力,特别是当任务需要注入大量领域特定知识时。
2. **有限的知识表示能力:** 软提示向量以一种相对简单的方式(即拼接)与输入序列进行交互。这种交互方式可能无法充分捕捉和表示复杂的领域知识,限制了模型学习和适应新知识的能力。
3. 预训练模型参数固定:在Prompt Tuning中,预训练模型的所有参数都是固定的,只有软提示向量可以更新。这意味着预训练模型本身无法适应和吸收新的领域知识,所有的知识注入都依赖于软提示向量的学习能力。
4. 任务适应性有限:尽管软提示向量可以引导模型生成符合任务要求的输出,但它们可能无法显著改变预训练模型的基础知识和语言理解能力。对于需要深入理解和推理的复杂任务,仅依靠软提示向量可能难以达到理想的性能。

## 1.4 P-Tuning (Prompt-based Tuning)

P-Tuning(Prompt-based Tuning)是由Liu等人在2021年提出的一种参数高效的微调技术。它通过引入一个可学习的、基于LSTM的提示编码器(prompt encoder)来生成软提示,从而实现了对下游任务的适应。与之前的Prompt Tuning和Prefix Tuning不同,P-Tuning使用LSTM网络来学习和生成更加连续、上下文相关的软提示。

P-Tuning可以比喻为给预训练模型配备一个“任务助手”。预训练模型就像一个具备基础知识的学生,而提示编码器就像一个智能助手,可以根据任务描述自动生成最合适的提示和指导。

在面对一个新任务时,智能助手会仔细分析任务的描述和要求,然后根据自己的经验和理解,为学生生成一系列连续、相关的提示和指导。这些提示不仅能够清晰地说明任务的目标和关键点,还能够根据上下文动态调整,为学生提供最有针对性和适应性的帮助。

在训练过程中,我们不断优化智能助手的知识和经验,使其能够为不同的任务生成最有效、最准确的提示和指导。同时,学生的基础知识(预训练模型参数)保持不变,可以在不同任务之间迁移和重用。这样,在智能助手的帮助下,学生就可以快速理解和适应各种不同的任务,并以最佳的方式完成任务目标。

### 1.4.1 出现背景与解决的问题

在之前的软提示学习方法中,如Prompt Tuning和Prefix Tuning, 软提示通常是一组独立的、随机初始化的向量。这种方式存在以下局限性:

- **软提示之间缺乏上下文关联:** 独立的软提示向量无法充分捕捉和利用提示内部的上下文信息和语义关系。

- **软提示的连续性不足:** 随机初始化的软提示向量可能导致生成的提示不连贯、不自然,影响模型的理解和适应能力。
- **软提示的表达能力有限:** 固定长度的软提示向量可能难以适应不同长度和复杂度的任务描述和要求。

P-Tuning通过引入基于LSTM的提示编码器,解决了这些局限性。LSTM网络可以学习和生成连续、上下文相关的软提示,提高了软提示的表达能力和适应性。同时,P-Tuning也降低了对手工设计提示模板的依赖,使得微调过程更加自动化和灵活。

## 1.4.2 算法总体思路

P-Tuning的核心思想是使用一个基于LSTM的提示编码器来生成软提示。具体而言,对于一个给定的任务,P-Tuning引入一个提示编码器 $LSTM_{\theta}$ ,其参数为 $\theta$ 。提示编码器接受一个任务描述 $\mathbf{d}$ 作为输入,生成一个长度为 $M$ 的软提示向量序列 $\mathbf{P} = [\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_M]$ ,其中 $\mathbf{p}_i \in \mathbb{R}^d$ , $d$ 为模型的隐藏状态维度。

$$\mathbf{P} = LSTM_{\theta}(\mathbf{d})$$

生成的软提示向量序列 $\mathbf{P}$ 与输入序列 $\mathbf{X} \in \mathbb{R}^{N \times d}$ ( $N$ 为序列长度)拼接,形成新的输入序列 $\mathbf{X}' = [\mathbf{P}; \mathbf{X}] \in \mathbb{R}^{(M+N) \times d}$ 。然后,将拼接后的序列 $\mathbf{X}'$ 输入到预训练模型中进行前向传播:

$$\mathbf{H} = \text{PretrainedModel}(\mathbf{X}')$$

其中 $\mathbf{H} \in \mathbb{R}^{(M+N) \times d}$ 为预训练模型的输出。

在训练过程中,P-Tuning优化提示编码器的参数 $\theta$ ,同时冻结预训练模型的所有参数。通过最小化任务特定的损失函数,P-Tuning学习生成最优的软提示,引导预训练模型适应下游任务。

## 1.4.3 微调数据组织与参数更新

在P-Tuning中,微调数据的组织与常规微调方法类似。对于每个任务,准备相应的输入序列、标签以及任务描述,并将其传递给添加了提示编码器的预训练模型。

在训练过程中,通过以下步骤更新提示编码器的参数:

- 前向传播:将任务描述输入到提示编码器中,生成软提示向量序列。将软提示向量序列与输入序列拼接,然后通过预训练模型进行前向传播,计算损失函数。
- 反向传播:根据损失函数计算提示编码器参数的梯度,并使用优化算法(如Adam)更新参数。
- 冻结预训练模型参数:确保只更新提示编码器的参数,而预训练模型的参数保持不变。

# 1.5 LoRA

LoRA全称为Low-Rank Adaptation of Large Language Models, 是一种适用于大型语言模型的参数高效微调技术。它通过在预训练模型的每一层注入可训练的低秩分解矩阵,实现了对模型的轻量化调优,在显著降低参数开销的同时,取得了与传统微调方法相当甚至更优的性能。

LoRA的工作原理可以比喻为一个“模型调优助手”。假设我们有一个预训练好的大型语言模型(如BERT),可以看作是一个训练有素的“语言专家”。这个专家已经掌握了大量的语言知识和技能,但可能还不足以完美地完成某些特定领域的任务。

传统的微调方法相当于让这个专家重新学习所有的知识和技能,以适应新的任务。这种“全面再训练”的方式不仅耗时耗力,而且可能会破坏专家原有的一些宝贵经验。

LoRA则采取了一种更加高效和谨慎的调优方式。它为专家配备了一个“调优助手”,这个助手不会直接修改专家的知识 and 技能,而是在专家工作的关键环节(如注意力机制、前馈神经网络等)提供一些“微调建议”。这些建议以低秩矩阵的形式注入到专家的工作流程中,起到“润物细无声”的调优效果。

专家在接受调优助手的建议后,并不需要改变自己的核心知识和技能,只需要在处理具体任务时稍作调整即可。由于调优助手提供的建议非常精简(低秩),因此不会给专家带来过重的认知负担。

通过专家和调优助手的协作,LoRA实现了对预训练语言模型的高效调优。在新任务上,调优后的专家不仅保留了原有的语言能力,还学会了灵活应用这些能力的技巧,从而取得了更优的任务性能。

## 1.5.1 研究背景与动机

近年来,预训练语言模型在自然语言处理领域取得了巨大成功。然而,随着模型规模的不断增大(如GPT-3的参数量高达1750亿),传统的微调方法面临着诸多挑战:

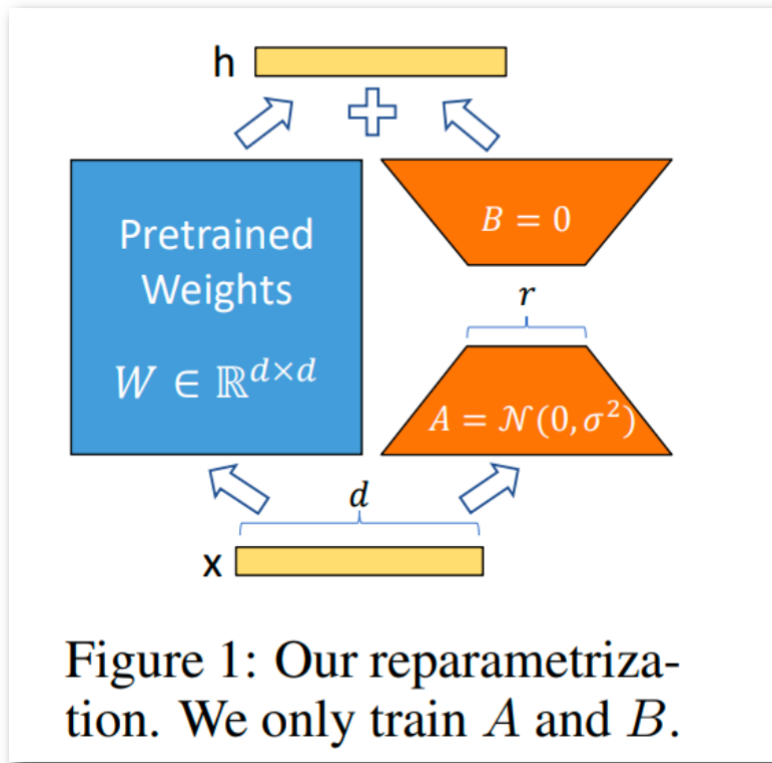
- **参数冗余:** 预训练模型中存在大量参数冗余,全量微调会导致过拟合风险增加,泛化能力下降。
- **计算资源限制:** 全量微调需要更新预训练模型的所有参数,对计算资源和存储空间要求较高,限制了其在实际应用中的可行性。
- **负迁移风险:** 在某些下游任务上,全量微调可能导致预训练模型的知识发生负迁移,反而降低了任务性能。

针对这些挑战,研究者们提出了一系列参数高效微调技术,如Adapter、Prefix Tuning、P-Tuning等。然而,这些方法仍然存在一些局限性,如**超参数敏感**、**模型适用范围有限**等。

LoRA的提出,正是为了进一步提高参数高效微调的性能和适用性。**通过在预训练模型的每一层注入低秩分解矩阵, LoRA在极低的参数开销下(常常少于0.1%), 实现了与全量微调相当甚至更优的性能。**

此外,LoRA具有良好的任务适应性和模型适用性,可以广泛应用于各种类型的预训练模型和下游任务。

## 1.5.2 算法原理与技术细节



LoRA的核心思想是在预训练模型的每一层注入一对低秩矩阵(秩为 $r$ ),用于调整模型的注意力权重矩阵。具体而言,对于Transformer的每个注意力头,其查询矩阵 $\mathbf{W}_q \in \mathbb{R}^{d \times d}$ 被分解为两个低秩矩阵 $\mathbf{A}_q \in \mathbb{R}^{d \times r}$ 和 $\mathbf{B}_q \in \mathbb{R}^{r \times d}$ 的乘积:

$$\mathbf{W}_q = \mathbf{W}_{q,0} + \Delta \mathbf{W}_q = \mathbf{W}_{q,0} + \mathbf{A}_q \mathbf{B}_q$$

其中, $\mathbf{W}_{q,0}$ 为预训练模型中原始的查询矩阵, $\Delta \mathbf{W}_q = \mathbf{A}_q \mathbf{B}_q$ 为注入的低秩调整项。同理,键矩阵 $\mathbf{W}_k$ 和值矩阵 $\mathbf{W}_v$ 也进行类似的低秩分解:

$$\mathbf{W}_k = \mathbf{W}_{k,0} + \mathbf{A}_k \mathbf{B}_k$$

$$\mathbf{W}_v = \mathbf{W}_{v,0} + \mathbf{A}_v \mathbf{B}_v$$

在微调过程中,我们只优化低秩矩阵 $\mathbf{A}_q$ 、 $\mathbf{B}_q$ 、 $\mathbf{A}_k$ 、 $\mathbf{B}_k$ 、 $\mathbf{A}_v$ 、 $\mathbf{B}_v$ 的参数,而预训练模型的原始参数 $\mathbf{W}_{q,0}$ 、 $\mathbf{W}_{k,0}$ 、 $\mathbf{W}_{v,0}$ 保持不变。由于秩 $r$ 通常远小于隐藏层维度 $d$ (如 $r = 8$ 、 $d = 1024$ ),因此LoRA引入的额外参数量极小,常常少于预训练模型参数量的0.1%。

除了注意力矩阵外,LoRA还可以对Transformer的前馈神经网络(FFN)层进行类似的低秩分解调整:

$$\mathbf{W}_1 = \mathbf{W}_{1,0} + \mathbf{A}_1 \mathbf{B}_1$$

$$\mathbf{W}_2 = \mathbf{W}_{2,0} + \mathbf{A}_2 \mathbf{B}_2$$

其中, $\mathbf{W}_1$ 和 $\mathbf{W}_2$ 分别为FFN层的第一个和第二个全连接层的权重矩阵。

通过在每一层注入少量的可训练参数,LoRA实现了对预训练模型的轻量化调优,在极低的参数开销下取得了显著的性能提升。



## 1.5.3 微调流程与数据准备

使用LoRA进行微调的流程如下:

1. 加载预训练模型(如BERT、RoBERTa等),并在每一层注入低秩矩阵。
2. 准备下游任务的训练数据,包括输入文本和对应的标签。
3. 使用训练数据对模型进行微调,其中只优化LoRA引入的低秩矩阵参数,预训练模型的原始参数保持不变。
4. 在验证集上评估微调后的模型性能,根据需要调整超参数(如学习率、训练轮数等)。
5. 使用微调后的模型对测试数据进行预测,并计算评估指标(如准确率、F1分数等)。

在数据准备方面,LoRA与传统的微调方法并无显著差异。对于分类任务,我们需要准备输入文本及其对应的类别标签;对于序列标注任务,我们需要准备输入文本及其对应的标签序列;对于生成任务,我们需要准备输入文本及其对应的目标文本。

在微调过程中,LoRA引入的低秩矩阵参数与预训练模型的原始参数一起参与前向传播和反向传播,但只有低秩矩阵参数会被梯度下降算法更新。这种参数高效的微调方式,使得LoRA在低资源场景下也能取得良好的性能。

## 1.6 P-Tuning v2

P-Tuning v2全称为"Prompt Tuning v2",是一种基于提示学习(Prompt Learning)的参数高效微调技术。与原始的P-Tuning不同,P-Tuning v2不仅在输入层引入连续的可学习提示,而且在预训练模型的每一层都注入了可学习的提示向量,以实现对模型的深度调优。该方法在多个任务上取得了与传统微调方法相当甚至更优的性能。

### 1.6.1 研究背景与动机

尽管原始的P-Tuning通过在输入层引入可学习的提示向量,在一定程度上提高了预训练语言模型在下游任务上的性能,但其局限性也日益凸显。由于P-Tuning只在输入层添加提示,因此其对预训练模型的调优能力有限,难以充分发挥提示学习的潜力。

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。

为了克服这一问题, P-Tuning v2提出了一种更加全面和深入的提示学习方法。其核心思想是在预训练模型的每一层都注入可学习的提示向量, 以实现对模型的深度调优。通过在每一层引入提示, P-Tuning v2可以更加灵活和有效地引导预训练模型适应下游任务, 从而提升模型的性能和泛化能力。

## 1.6.2 算法原理与技术细节

P-Tuning v2的核心思想是在预训练模型的每一层注入一组可学习的提示向量, 并将其与原始的输入表示进行拼接, 以实现对模型的深度调优。具体而言, 对于Transformer的每一层, P-Tuning v2引入了一组长度为 $l$ 的提示向量 $\mathbf{P}^{(i)} \in \mathbb{R}^{l \times d}$ , 其中 $i$ 表示层的编号,  $d$ 表示隐藏状态的维度。

在前向传播过程中, 对于第 $i$ 层的输入表示 $\mathbf{H}^{(i)} \in \mathbb{R}^{n \times d}$ , P-Tuning v2首先将其与提示向量 $\mathbf{P}^{(i)}$ 进行拼接, 得到增强后的输入表示 $\hat{\mathbf{H}}^{(i)} \in \mathbb{R}^{(n+l) \times d}$ :

$$\hat{\mathbf{H}}^{(i)} = \text{Concat}(\mathbf{H}^{(i)}, \mathbf{P}^{(i)})$$

然后, 增强后的输入表示 $\hat{\mathbf{H}}^{(i)}$ 再经过自注意力层(Self-attention)、前馈神经网络层(FFN)等常规的Transformer计算, 得到该层的输出表示 $\mathbf{H}^{(i+1)}$ 。

在训练过程中, P-Tuning v2只优化每一层的提示向量 $\mathbf{P}^{(i)}$ , 而预训练模型的原始参数保持不变。这种参数高效的微调方式不仅大大减小了训练开销, 而且通过在每一层引入提示, 实现了对预训练模型的深度调优, 显著提升了模型在下游任务上的性能表现。

在推理过程中, 我们只需将学习到的提示向量 $\mathbf{P}^{(i)}$ 逐层拼接到输入表示中, 再经过常规的Transformer计算, 即可得到最终的输出结果。由于提示向量的长度 $l$ 通常远小于输入序列长度 $n$ , 因此P-Tuning v2带来的推理开销也非常有限。

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。

## 1.7 QLoRA

QLoRA全称为Quantized Low-Rank Adaptation, 是一种基于量化和低秩分解的参数高效微调技术。它在LoRA的基础上, 通过对低秩矩阵进行量化, 进一步压缩了微调参数的存储和计算开销, 同时保持了与LoRA相当的性能表现。

QLoRA的工作原理可以比喻为一个“模型调优助手+数据压缩专家”的组合。在LoRA的基础上, QLoRA引入了一个“数据压缩专家”, 负责对调优助手提供的建议(即低秩矩阵)进行压缩和编码。

具体而言, 调优助手首先为语言专家(即预训练模型)提供一些微调建议, 这些建议以低秩矩阵的形式表示。然而, 为了进一步降低存储和计算开销, 数据压缩专家对这些低秩矩阵进行量化, 即将其中的浮点数参数转换为定点数表示。

量化后的低秩矩阵虽然在数值精度上有所损失,但其参数的位宽(如8位)远小于原始的浮点数表示(如32位),因此可以显著减小存储和计算开销。同时,由于量化是在参数级别进行的,且采用了对称量化的方式,因此量化带来的精度损失对模型性能的影响较小。

在推理过程中,语言专家首先对输入进行处理,得到中间结果;然后,调优助手使用量化后的低秩矩阵对中间结果进行微调,得到最终的输出。这种分离的计算方式可以防止量化误差在网络中累积,提高量化微调的性能表现。

通过调优助手和数据压缩专家的协作,QLoRA实现了对预训练语言模型的高效微调和压缩。在资源受限的场景下,QLoRA可以在保持性能的同时,进一步降低微调的存储和计算开销,提高模型的实用性和可用性

## 1.7.1 研究背景与动机

尽管LoRA已经实现了极低的参数开销(常常少于预训练模型参数量的0.1%),但在某些资源极度受限的场景下(如嵌入式设备、移动端应用等),其参数量和计算量仍然可能超出设备的承载能力。

此外,随着预训练语言模型的规模不断增大(如GPT-3的参数量达到1750亿),即使是LoRA引入的少量参数,其绝对数量也可能达到亿级别,给存储和计算带来挑战。

为了进一步降低参数高效微调技术的资源消耗,研究者们开始探索将量化技术与LoRA相结合的可能性。

**量化是一种经典的模型压缩方法,通过将模型参数从高精度浮点数(如FP32)转换为低精度整数(如INT8),可以显著减小参数的存储和计算开销。**

QLoRA的提出,正是为了将量化的优势引入到LoRA中,实现更加高效和轻量化的模型微调。通过对LoRA的低秩矩阵进行量化,QLoRA在保持性能的同时,进一步压缩了微调参数的存储和计算开销,使其更加适用于资源受限的场景。

## 1.7.2 算法原理与技术细节

QLoRA的核心思想是在LoRA的基础上,对低秩矩阵进行量化。具体而言,对于Transformer的每个注意力头,其查询矩阵 $\mathbf{W}_q \in \mathbb{R}^{d \times d}$ 被分解为两个低秩矩阵 $\mathbf{A}_q \in \mathbb{R}^{d \times r}$ 和 $\mathbf{B}_q \in \mathbb{R}^{r \times d}$ 的乘积,并进行量化:

$$\mathbf{W}_q = \mathbf{W}_{q,0} + \Delta \mathbf{W}_q = \mathbf{W}_{q,0} + \text{Quantize}(\mathbf{A}_q) \cdot \text{Quantize}(\mathbf{B}_q)$$

其中, $\mathbf{W}_{q,0}$ 为预训练模型中原始的查询矩阵, $\Delta \mathbf{W}_q = \text{Quantize}(\mathbf{A}_q) \cdot \text{Quantize}(\mathbf{B}_q)$ 为量化后的低秩调整项。键矩阵 $\mathbf{W}_k$ 和值矩阵 $\mathbf{W}_v$ 也进行类似的量化低秩分解。

QLoRA中的量化操作通常采用对称量化(Symmetric Quantization)的方式,即将浮点数参数 $x$ 量化为定点数 $\text{Quantize}(x)$ :

$$\text{Quantize}(x) = \text{round}\left(\frac{x}{\Delta}\right) \cdot \Delta$$

其中,  $\Delta$  为量化步长, 通常设置为  $\Delta = \frac{\max(|x|)}{2^{b-1}-1}$ ,  $b$  为量化位宽(如8位)。

在推理过程中, 我们首先对输入进行前向传播, 得到中间结果; 然后对中间结果应用量化后的低秩调整项, 得到最终的输出。这种分离的计算方式避免了量化带来的精度损失在网络中累积, 提高了量化微调的性能表现。

在训练过程中, 我们采用伪量化(Fake Quantization)的方式, 即在前向传播时对参数进行量化, **在反向传播时则使用原始的浮点数参数计算梯度**。这种方式可以在训练过程中模拟量化的效果, 同时保留参数的连续性, 便于梯度的计算和优化。

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。

除了注意力矩阵外, QLoRA还可以对Transformer的前馈神经网络(FFN)层进行类似的量化低秩分解调整:

$$\begin{aligned} \mathbf{W}_1 &= \mathbf{W}_{1,0} + \text{Quantize}(\mathbf{A}_1) \cdot \text{Quantize}(\mathbf{B}_1) \\ \mathbf{W}_2 &= \mathbf{W}_{2,0} + \text{Quantize}(\mathbf{A}_2) \cdot \text{Quantize}(\mathbf{B}_2) \end{aligned}$$

其中,  $\mathbf{W}_1$  和  $\mathbf{W}_2$  分别为FFN层的第一个和第二个全连接层的权重矩阵。

通过对LoRA中的低秩矩阵进行量化, QLoRA实现了更加高效和轻量化的模型微调, 在资源受限场景下具有广阔的应用前景。

### 1.7.3 微调流程与数据准备

使用QLoRA进行微调的流程与LoRA基本一致, 主要区别在于对低秩矩阵进行量化:

1. 加载预训练模型(如BERT、RoBERTa等), 并在每一层注入低秩矩阵。
2. 对低秩矩阵进行量化, 即将其转换为定点数表示。
3. 准备下游任务的训练数据, 包括输入文本和对应的标签。
4. 使用训练数据对模型进行微调, 其中只优化量化后的低秩矩阵参数, 预训练模型的原始参数保持不变。
5. 在验证集上评估微调后的模型性能, 根据需要调整超参数(如学习率、训练轮数、量化位宽等)。
6. 使用微调后的模型对测试数据进行预测, 并计算评估指标(如准确率、F1分数等)。

在数据准备方面, QLoRA与LoRA和传统的微调方法并无显著差异。对于不同类型的任务(如分类、序列标注、生成等), 我们需要准备相应的输入文本和标签数据。



## 1.7.4 未来展望

QLoRA为参数高效微调技术的发展提供了新的思路,其量化与低秩分解相结合的设计值得进一步探索和优化。未来,QLoRA的研究可以在以下几个方向上拓展:

- 量化方案优化:探索更加高效和精确的量化方案(如非对称量化、learned步长量化等),以进一步提高QLoRA的性能和效率。
- 适用模型拓展:将QLoRA的思想拓展到其他类型的预训练模型(如CNN、RNN等)以及下游任务(如语音识别、图像分类等),扩大其应用范围。
- 硬件友好设计:针对不同硬件平台(如CPU、GPU、ASIC等)的特性,设计更加硬件友好的QLoRA量化方案,如采用powers-of-two量化步长,以便于硬件加速。
- 量化感知训练:在预训练阶段引入量化感知的目标函数和优化策略,使得预训练模型更加适合后续的量化微调,提高QLoRA的性能上限。

总之,QLoRA在LoRA的基础上,进一步压缩了微调参数的存储和计算开销,在资源受限场景下具有广阔的应用前景。相信通过研究者的不断探索和优化,QLoRA以及类似的量化微调技术将在未来得到更加广泛和深入的应用,为预训练语言模型的实用化部署提供有力支持。

## 1.8 最后

我感觉 Prefix Tuning、Prompt Tuning、P-Tuning及P-Tuning v2 比较适用于 更好的优化 Prompt, 以获得更好从LLM中提取知识, 而 LORA、QLoRA、Adapter Tuning 更偏向于"注入知识", 相比之下,Prefix Tuning、Prompt Tuning、P-Tuning及P-Tuning v2 更偏向于现在的权宜之计, 实现并不优雅, 一个理想的LLM应该是直接端到端的, 能够理解人类自然语言的, 而不需要将我们的 hard-prompt 转化为 soft-prompt

你指出Prefix Tuning、Prompt Tuning、P-Tuning及P-Tuning v2更适用于优化Prompt,以便更好地从语言模型中提取知识,而LoRA、QLoRA、Adapter Tuning更偏向于向模型中注入新的知识。这一区分非常精准,揭示了这两类方法在目标和机制上的本质差异。

Prompt优化类方法旨在通过学习连续的嵌入向量,来自动构建最优的Prompt,替代人工设计的离散Prompt(即hard prompt)。这些方法的重点是如何以最佳方式询问语言模型, 便更好地利用其已有的知识。这种方法大多针对**特定类型的任务进行设计**,如分类、序列标注、生成等. 而知识注入类方法则旨在通过额外的可训练模块,来补充语言模型在特定任务上所欠缺的知识。这些方法的重点是如何在不破坏原有知识的情况下,为语言模型灌输新的知识。

现有的PEFT方案局限性有哪些? 未来PEFT的发展方向是什么?

任务适应性的局限:



现有的PEFT方法大多针对**特定类型的任务进行设计**,如分类、序列标注、生成等。它们在这些任务上表现出色,但在面对一些复杂、多样化的任务时,如多轮对话、开放域问答、推理解释等,它们的适应性和泛化能力还有待提高。

未来的PEFT方法应该更加注重任务普适性,能够灵活适应不同类型、不同领域的任务需求。这可能设计更加通用的微调模块,如具有递归结构的Adapter、可组合的Prompt等,以及更加智能的任务表示和转换机制,如基于元学习、强化学习的自适应微调等。

# 2 RAG

## 参考资料 - RAG

Backbone : [大语言模型的检索增强生成技术 \(RAG\)](#)

[从 RAG 到 Self-RAG —— LLM 的知识增强 - 知乎 \(zhihu.com\)](#)

Information : [RAG 工具 langchain](#)

Information : [落地应用 1 - TorchV Bot](#)

Information : [落地应用 2 - Mr History](#)

检索增强生成(RAG)是一种将外部知识库中的信息与语言模型(LLM)相结合的技术,它能够显著提升LLM在知识密集型任务上的表现。本文系统梳理了RAG技术的发展历程,概括了RAG的三种范式: 朴素RAG(Naive RAG)、进阶RAG(Advanced RAG)和 模块化RAG(Modular RAG)。

进阶RAG在朴素RAG的基础上引入了更复杂的架构元素,如查询重写、文本块重排序和提示摘要等,使得LLM的性能和可解释性都得到提升。模块化RAG则进一步增强了RAG的灵活性,允许针对不同任务定制不同的检索和生成模块。

RAG在内容检索方面采用了结构化数据与非结构化数据相结合的混合方法。最新的RAG研究还探索了一些创新的概念,例如LLM的自我检索以及自适应的信息检索时机控制。

尽管RAG取得了长足进展,但在提高其鲁棒性和处理长文本的能力方面仍有很大的研究空间。此外,RAG正在将其应用拓展到多模态领域,如图像、视频和代码的理解与生成。这突显了RAG在人工智能落地方面的重要现实意义。

随着RAG生态系统的不断发展,我们迫切需要与之相匹配的评估方法。只有确保性能评估的准确性和代表性,才能全面把握RAG对AI研究与开发的贡献。RAG是LLM领域的一个重要分支,工程师们了解RAG的核心理念和关键技术,有助于在实际项目中更好地应用LLM,开发出更智能、更可靠的AI系统

## 1 RAG 简介

### 1.1 技术背景

检索增强生成 [Retrieval-Augmented Generation, RAG] 是一种将外部知识库中的信息与语言模型 [Language Model, LLM] 相结合的技术。简而言之,  $RAG = externalDatabase + LLM$

RAG的诞生背景可以追溯到大语言模型的局限性。尽管LLM在各种基准测试中表现出色,但它们在处理特定领域或高度专业化的查询时仍然存在不足。这些局限性包括生成 **<不正确的信息(即"幻觉" hallucination)>**、**<知识过时>** 以及 **<推理过程不透明>**。

为了应对这些挑战,RAG应运而生。它通过在**LLM生成过程中集成外部数据检索**,增强了LLM提供准确和相关响应的能力。RAG的主要用途在于增强LLM处理知识密集型任务的能力,尤其是那些需要**<最新信息>**或**<领域特定知识>**的任务。通过动态检索外部知识库中的信息,RAG可以帮助LLM生成更准确、更相关、更新颖的回答。此外,RAG还提高了LLM的**<可解释性>**,因为生成的内容可以追溯到检索到的证据。

总的来说,RAG**弥合了LLM固有知识与外部知识库之间的鸿沟**,使得LLM能够更有效地应对现实世界中的各种应用场景,如问答系统、对话助手和知识库查询等。同时,它**避免了为特定任务应用重新训练LLM的需要**。开发人员可以附加一个外部知识库,丰富输入,从而提高模型的输出精度。

RAG的主要优势在于由于其高实用性和低进入门槛,RAG已成为LLM系统中最流行的架构之一,许多对话产品几乎完全建立在RAG之上。

## 1.2 发展脉络

RAG技术的发展大致可以分为四个阶段:

### 1. 探索阶段(2017-2020年):

这一阶段与Transformer架构的出现相吻合。研究的重点是通过预训练模型[Pre-Training Models, PTM]吸收额外的知识,从而增强语言模型的能力。这一时期的RAG研究主要集中在优化预训练方法上。

### 2. 沉寂阶段(2020-2022年):

在ChatGPT出现之前,RAG相关研究进展缓慢,没有太多突破性的进展。

### 3. 繁荣阶段(2022年至今):

ChatGPT的问世将LLM推到了聚光灯下。研究社区的关注点转移到了如何利用LLM的能力来提高可控性和满足不断变化的需求上。在这一阶段,绝大多数RAG工作都集中在推理阶段,少数工作关注微调过程。

### 4. 转型阶段(2023年至今):

随着GPT-4等更先进LLM的出现,RAG技术出现了新的变革。研究重点转向了RAG与微调相结合的混合方法,同时也有一小部分工作继续专注于优化预训练方法。

# 1.3 工作流程

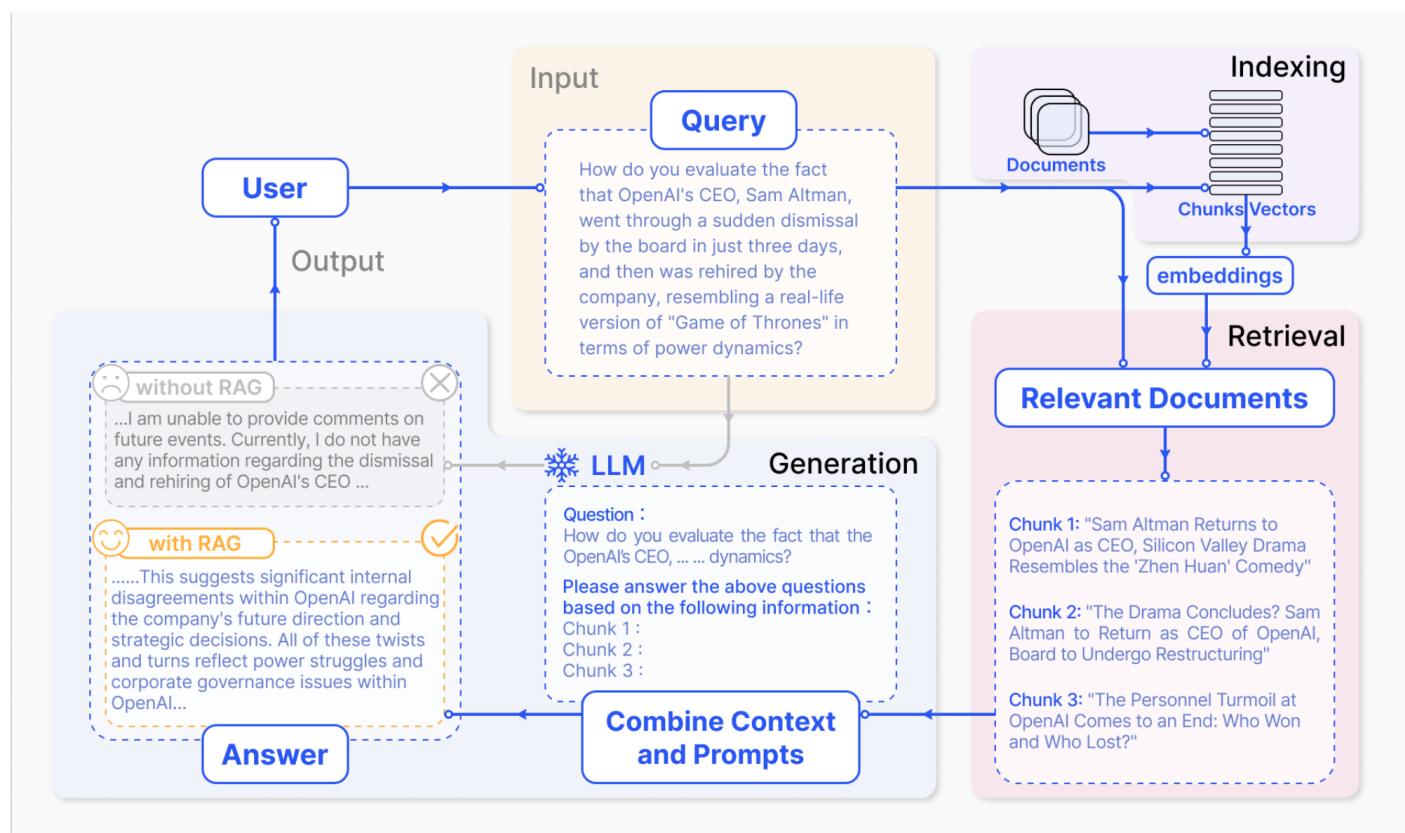


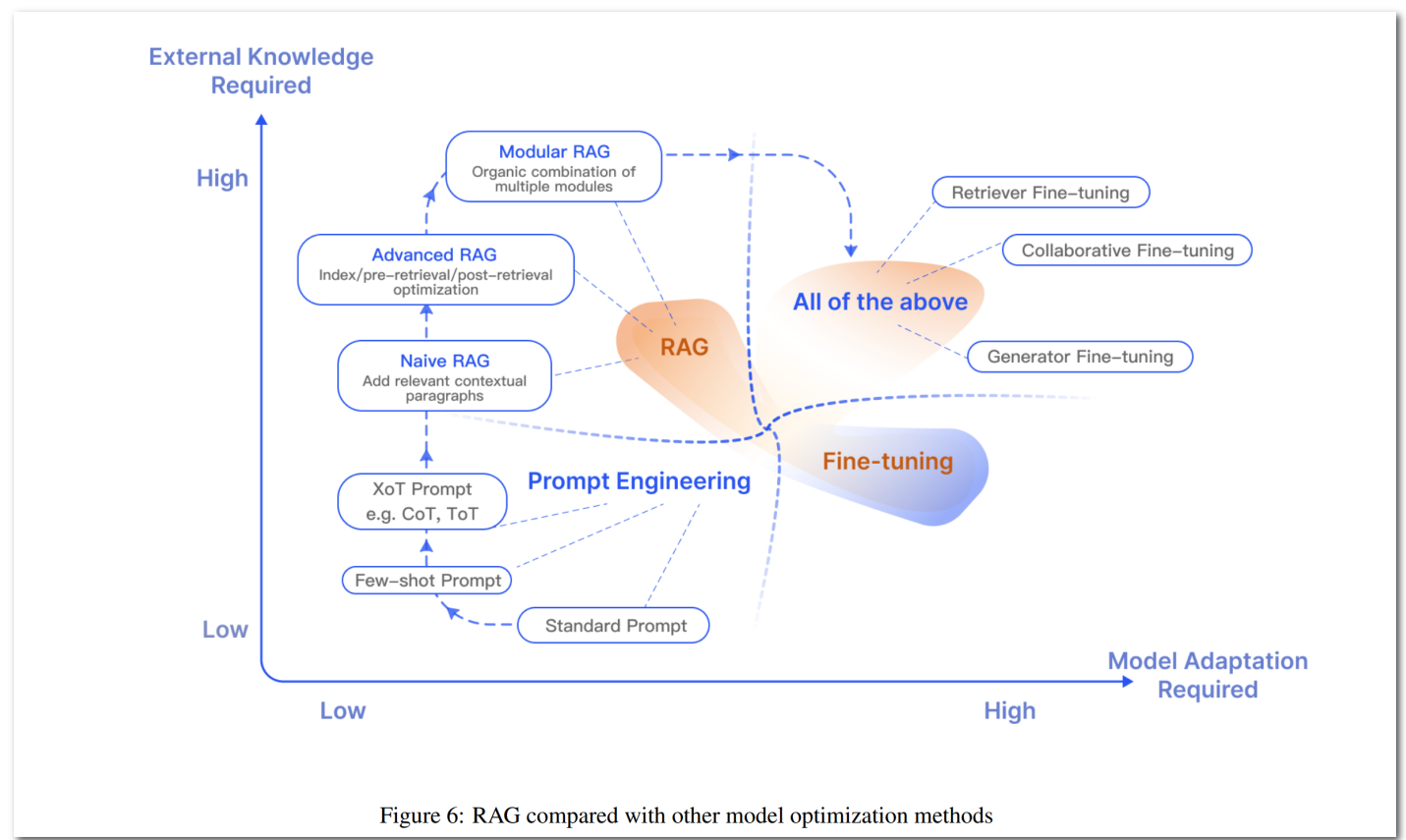
Figure 2: A representative instance of the RAG process applied to question answering

RAG是一种通过**集成外部知识库来增强LLM**的范式。它采用协同方法,结合**信息检索机制**和**上下文学习 (ICL)**来提高LLM的性能。在这个框架中,用户发起的查询会通过搜索算法**检索**相关信息。然后将这些信息编入LLM的Prompt中,为生成过程提供额外的上下文。如上图所示。

RAG工作流程包括三个关键步骤。首先,语料库被划分为离散的块,利用编码器模型构建向量索引。其次,RAG根据查询和索引块之间的向量相似性来识别和检索块。最后,模型根据从检索到的块中获得的上下文信息综合出响应。这些步骤构成了RAG过程的基本框架,支撑其信息检索和上下文感知生成能力。

RAG的核心组件包括**检索器[Retriever]**和**生成器[Generator]**。**检索器负责从外部知识库中检索相关信息,生成器则根据检索到的信息生成最终的回答。**随着RAG技术的不断发展,各种辅助组件被引入到RAG的管道中,以优化检索-生成的过程。不同类型的RAG具有不同的管道设计,但它们的共同目标都是通过**知识增强**来提升LLM的性能。

1.4 RAG v.s. Finetune



RAG 就像是给模型提供了一本参考书，让它能够针对具体的问题进行**信息检索**。这种方式适合于模型需要对特定的问题给出答案或处理某些信息检索任务的情境。然而，当涉及到让模型掌握广泛的知识领域或是学习新的语言、格式或风格时，RAG 就不太适用了。

微调就像是帮助学生通过深入学习来掌握并应用知识。当模型需要模仿特定的结构、风格或格式时，这种方法显得尤为重要。通过微调，可以增强模型的性能，**使得与模型的互动变得更流畅、高效**

微调尤其适用于加强模型已有的知识基础，调整或个性化模型的输出结果，并向模型输入复杂的操作指南。但是，**微调不适宜于向模型融入新的知识，也不适合那些需要快速迭代以适应新场景的情况。**

微调就像是让学生通过**深入学习逐渐吸收并掌握知识**。这种方法特别适用于**模型需要模仿特定结构、风格或格式的情况**。微调可以让模型的表现超越未经微调的水平，并且使得与模型的互动更为高效。它特别适合于加强模型已有的知识，调整或个性化输出结果，以及向模型提供复杂的操作指引。然而，微调并不适合用于引入模型尚未掌握的新知识，或者**适应那些需要快速迭代新应用场景的情况。**

RAG (Retrieval-Augmented Generation) 和模型微调不是不可兼得的对立面，它们实际上可以互为补充，从而在不同的层次上提升模型的功能。在某些特定的场景中，将这两种方法结合起来，可以使模型发挥出最佳的性能。要通过 RAG 和微调来完全优化模型，可能需要经过多轮的调整 and 测试，才能得到满意的效果。

特性	RAG	微调
知识更新	直接更新检索库，保证信息新鲜，适合快速变化的数据环境。	静态数据存储，需重新训练更新知识。



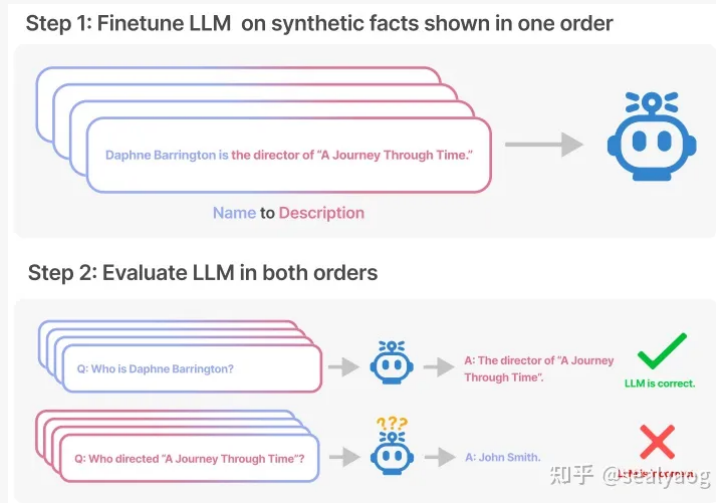
特性	RAG	微调
外部知识	擅长整合外部资源，适合处理文档或其他结构化/非结构化数据库。	预训练知识与大语言模型结合，但不适合经常变的数据源。
数据处理	最小化数据处理需求。	需要高质量数据集，数据限制可能影响性能。
模型定制	重点在信息检索和整合，定制化模型行为有限。	可根据需求调整模型行为和风格，适应特定领域。
可解释性	答案可追溯至数据源，解释性强。	模型反应原因不明显，解释性较弱。
计算资源	需要资源支持检索和数据库技术，维护数据源更新。	需要资源准备训练数据，定义微调目标。
延迟要求	数据检索可能增加延迟。	微调模型响应快，延迟低。
减少幻觉	基于检索证据，不易产生幻觉。	特定域训练可减少幻觉，但面对新输入时可能仍幻觉。
道德和隐私	涉及外部数据时可能有道德隐私问题。	训练数据中的敏感内容可能引发道德隐私问题。

- RAG 通过结合外部知识库来提供答案，显著提高了准确性，有效减少了模型生成的不实内容，确保了回答的真实性和可靠性。
- RAG 利用先进的检索技术，可以快速获取最新资讯，与传统的大语言模型（LLM）相比，它能更好地保证信息的及时更新和准确性。
- RAG 的透明度是其另一大优势。它能够明确指出信息来源，让用户能够直接核实答案的真实性，从而增强了人们对模型输出的信任度。
- RAG 还具备高度的定制性。通过构建针对特定领域的文本索引库，它能够根据不同行业量身打造知识支持，满足特定需求。
- 在安全性和隐私保护方面，RAG **通过在其数据库中设置特定的用户角色和安全控制措施，能够更精确地管理数据的使用权限**。与此相对，经过微调的模型可能在数据访问权限的管理上不够明确，存在安全隐患。
- RAG 在处理大规模数据集时更加灵活，它不需要对所有参数进行更新，也不必为每次数据变动创建新的训练集，这种方式在成本上更为经济。
- 最后，RAG 生成的结果更加可靠。它总是从最新的数据中选取确定性的结果，而经过微调的模型在处理实时更新的数据时可能会产生错误和不一致的信息，这在一定程度上降低了模型的透明度和可信度。

另外一种 Finetune形式的局限性

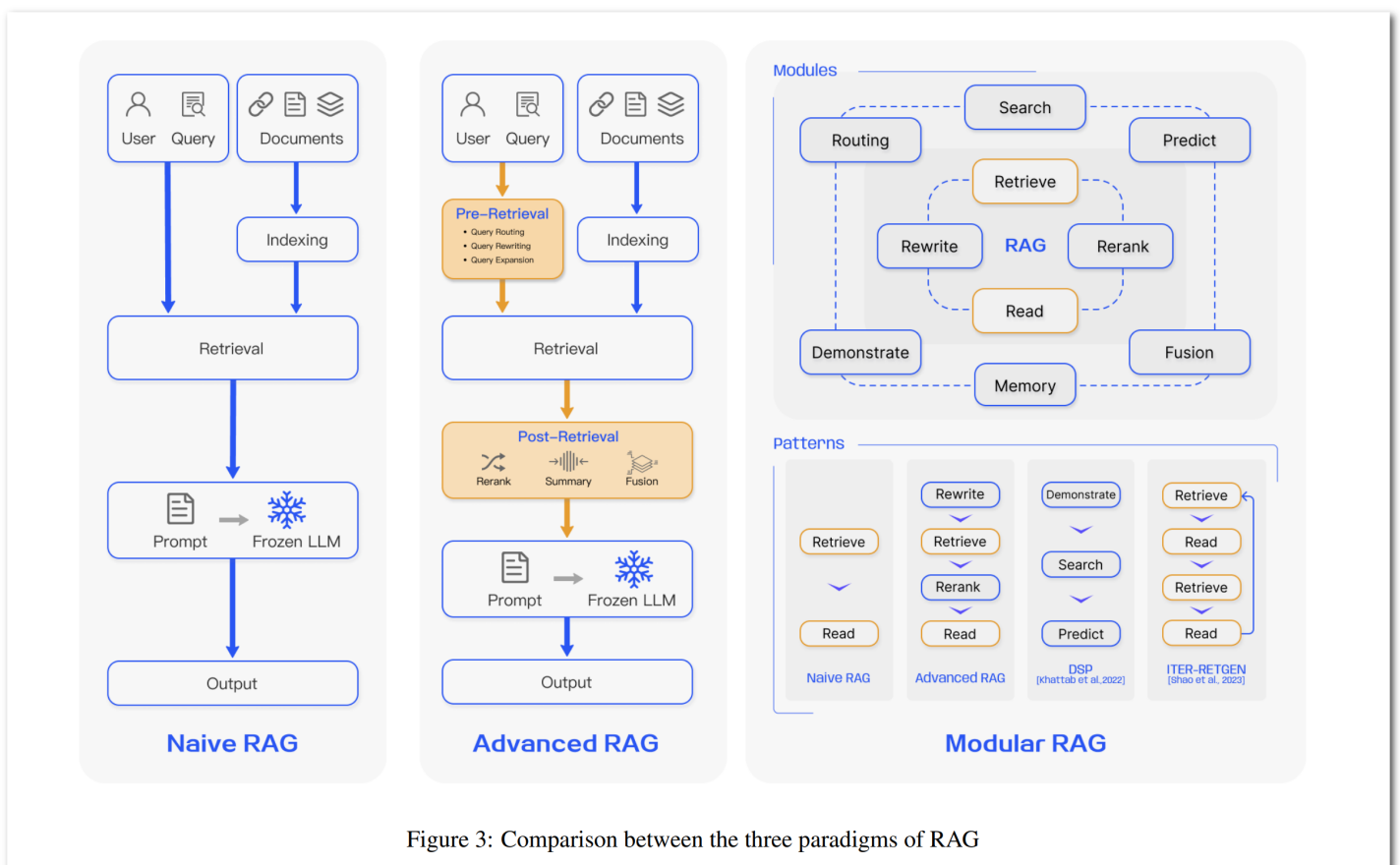
The Reversal Curse 。

让LLM学会一个简单的知识：A 就是 B，然后反过来问B是什么，LLM并不能回答B是A。所有LLM都不能幸免。



在OpenAI著名科学家 Andrej Karpathy看来：LLM 学到的知识比你我想象的要“零散”得多，并且缺少足够的概括泛化能力，“逆转诅咒”是一个特例。这种零散和不稳定是基于“连接主义”的LLM胎里带的副作用，而系统、准确、可靠是基于“符号主义”的KG的天然优势，而在相当多的领域内这种优势是无法被忽视的

## 2 RAG 框架



从现阶段的**RAG范式演进**来说, 发展可以分为 **朴素RAG[Naive RAG]**、**进阶RAG[Advanced RAG]**和 **模块化RAG[Modular RAG]**。Naive RAG 虽然性能明显超过原生LLM, 但是表现出几个局限性. 后者正是为了应对这些局限性做的改进.

## 2.1 Naive RAG

### Naive RAG Pipeline

**朴素RAG[Naive RAG]** 遵循标准的检索流程, 包括 索引[Indexing]、检索[Retrieval] 和 生成[Generation] 三个步骤.

1. 首先 对原始数据进行清理、提取和分块, 然后将文本块转换为向量表示, 构建倒排索引或向量索引;
2. 接着 据用户查询, 将输入转换为Vector表示, 利用相似度计算等方法从索引中检索出最相关的文本块;
3. 最后 将查询和检索到的文本块拼接成一个上下文Prompt, 输入到LLM中生成最终答案。

#### STEP 1 索引

索引过程是数据准备中的一个关键初始步骤,发生在离线状态,涉及几个阶段。

它从数据索引开始,对原始数据进行清理和提取, 将PDF、HTML、Word和Markdown等各种文件格式**转换为标准的纯文本**。为了**适应语言模型的上下文限制**, 这些文本随后被分割成更小、更易管理的块,这一过程称为分块(chunking)。这些块随后通过一个**Embedding嵌入模型**转换为向量表示, 该模型在**推理效率和模型大小**之间取得平衡。这有利于在检索阶段进行相似性比较。最后,**创建一个索引来存储这些文本块及其向量嵌入作为键值对**, 这允许高效和可扩展的搜索能力。

#### STEP 2 检索

在收到用户查询后, 系统使用索引阶段**使用的相同编码模型将输入转换为向量表示**。然后,它计算查询向量与索引语料库中向量化块之间的相似度得分。系统**优先检索与查询相似度最高的前K个块**。这些块随后被用作解决用户请求的扩展上下文基础。

#### STEP 3 生成

提出的查询和选定的文档被合成为一个连贯的提示,要求大型语言模型对其做出响应。模型的回答方法**可能会根据特定任务的标准**而有所不同, 允许它要么利用其固有的参数知识, 要么将其响应限制在所提供文档中包含的信息内。**在持续对话的情况下,任何现有的对话历史都可以集成到提示中**,使模型能够有效地进行多轮对话交互。

## Naive RAG 的缺点

朴素RAG在三个关键领域面临重大挑战: **检索** , **生成** , **增强** .

### ① 检索

- **低精度**, 导致检索到的块不匹配, 可能出现幻觉或中途跌落 mid-air drop 等问题。
- **低召回率** 也会发生, 导致**未能检索到所有相关的块**, 从而阻碍LLM制作全面响应的能力。
- **Outdated information** 进一步加剧了这个问题, 可能导致不准确的检索结果。

② **生成 - 幻觉 hallucination** , 模型生成未在所提供上下文中得到证实的答案, 以及不相关上下文和模型输出中潜在的毒性或偏差等问题

③ **增强** 过程面临的挑战是**如何有效地将检索到的段落中的上下文与当前的生成任务整合起来**,

- 可能导致输出支离破碎或不连贯。
- 当多个检索到的段落**包含相似的信息时**, 冗余和重复也是需要关注的问题, 导致**生成的响应中出现重复的内容**。

## 2.2 Advanced RAG

高级RAG的开发进行了有针对性的改进, 以解决朴素RAG的不足之处。在检索质量方面, 高级RAG实施了预检索和后检索策略。为了应对朴素RAG所经历的索引挑战, 高级RAG采用滑动窗口、细粒度分割和元数据等技术优化了索引方法。它还引入了各种方法来优化检索过程。

### ① Advanced Index

优化数据索引的目标是**提高被索引内容的质量**。这涉及五个主要策略: 增强数据粒度、优化索引结构、添加元数据、对齐优化和混合检索

- 增强数据粒度[Enhancing data granularity] 旨在提高文本规范化、一致性、事实准确性和丰富上下文, 以提高RAG系统的性能。这包括<删除不相关的信息>, <消除实体和术语中的歧义>, 确认事实准确性, 保持上下文, 以及<更新过时的文档>。
- 优化索引结构[Optimizing index structures] 涉及<调整块的大小以捕获相关上下文>, <跨多个索引路径进行查询>, 以及<结合图结构>中的信息来捕获相关上下文, 利用图数据索引中节点之间的关系。
- 添加元数据信息[Adding metadata information] 涉及将引用的<元数据(如日期和目的)集成到块>中用于过滤目的, 并结合引用的章节和小节等元数据来提高检索效率。
- 对齐优化[Alignment optimization] 通过在文档中引入"假设性问题"来纠正文档之间的对齐问题和差异。

## ② Advanced Retrieval

在检索阶段, 主要重点是通过计算查询和块之间的相似性来识别适当的上下文。嵌入模型是这一过程的核心。在高级RAG中,有可能优化嵌入模型。

- **微调嵌入[Fine-tuning Embedding]**。微调嵌入模型显著**影响RAG系统中检索到的内容的相关性**。这个过程涉及<定制嵌入模型以增强特定领域上下文中的检索相关性>, 特别是**对于处理不断发展或罕见术语的专业领域**。

BGE嵌入模型,例如BAAI开发的BGE-large-EN,就是一个可以微调以优化检索相关性的高性能嵌入模型。微调的训练数据可以使用GPT-3.5-turbo等语言模型根据文档块制定问题来生成,然后将其用作微调对。

- **动态嵌入[Dynamic Embedding]** 根据单词使用的上下文进行调整, 而静态嵌入为每个单词使用单一向量。

例如,在BERT等transformer模型中,同一单词根据周围的单词可以有不同的嵌入。OpenAI的embeddings-ada-02模型建立在GPT等LLM的原理之上,是一个复杂的动态嵌入模型,能够捕捉上下文理解。

然而,它可能不会表现出与最新的全尺寸语言模型(如GPT-4)相同的对上下文的敏感性。

## ③ Advanced Augmentation

从数据库中检索到有价值的上下文后,将其与查询合并作为LLM的输入至关重要,同时要**应对上下文窗口限制带来的挑战**。简单地**一次性向LLM呈现所有相关文档可能会超出上下文窗口限制,引入噪音,并阻碍对关键信息的关注**。

- **重新排序[Re-Ranking]**。将检索到的信息重新排序, **将最相关的内容重新定位到提示的是一个关键策略**。

这一概念已在 LlamaIndex、LangChain和HayStack 等框架中实现。例如,Diversity Ranker优先考虑基于文档多样性的重新排序,而LostInTheMiddleRanker 则在上下文窗口的开头和结尾交替放置最佳文档。此外,cohereAI rerank、bge-rerank和LongLLMLingua等方法重新计算相关文本与查询之间的语义相似度,解决了解释基于向量的模拟搜索语义相似度的挑战。

- **提示压缩[Prompt Compression]**。研究表明,检索到的文档中的噪音对RAG性能有不利影响。在后处理中,**重点在于压缩不相关的上下文,突出关键段落,并减少整体上下文长度**。

Selective Context和LLMLingua等方法利用小型语言模型来计算提示互信息或困惑度,估计元素的重要性。Recomp通过在不同粒度上训练压缩器来解决这个问题,而Long Context和"Walking in the Memory Maze"设计了总结技术来增强LLM对关键信息的感知,特别是在处理广泛上下文时。



## 2.3 Modular RAG

模块化RAG结构与传统的朴素RAG**框架不同**, 提供了更大的多功能性和灵活性。它集成了各种方法来增强功能模块, 例如**结合搜索模块进行相似性检索**, 并在**检索器中应用微调方法**。重构的RAG模块和迭代方法已经开发出来,用于解决具体问题。

模块化RAG范式正日益成为RAG领域的标准, 允许采用**串行化流水线**或**跨多个模块的端到端训练方法**。三种RAG范式的比较如图所示。然而,模块化RAG并非独立存在。高级RAG是模块化RAG的一种特殊形式, 而朴素RAG本身又是高级RAG的一个特例。三种范式之间的关系是继承和发展的关系。

### New Modules

#### 1. 搜索模块 [Search Module]

与朴素/高级RAG中的**相似性检索不同**,**搜索模块针对特定场景定制**, 并结合对**额外语料库的直接搜索**。这种集成是**通过LLM生成的代码、SQL或Cypher等查询语言以及其他定制工具实现的**。这些搜索的**数据源**可以包括**搜索引擎、文本数据、表格数据和知识图谱**。

#### 2. 记忆模块 [Memory Module]

该模块**利用LLM的记忆能力来指导检索**。该方法涉及**识别与当前输入最相似的记忆**。

Selfmem使用一个检索增强的生成器迭代地创建一个无界的记忆池, 结合"原始问题"和"双重问题"。通过使用自身输出来改进自身的检索增强生成模型, 文本在推理过程中变得更加符合数据分布。因此,使用模型自身的输出而不是训练数据。

#### 3. 融合 [Fusion]

RAG-Fusion 通过采用**多查询方法增强传统搜索系统**,该方法使用**LLM将用户查询扩展为多个不同的视角**。这种方法不仅**捕捉用户寻求的显式信息**,还**发掘更深层次的、变革性的知识**。融合过程涉及**<对原始查询和扩展查询进行并行向量搜索>**,**<智能重新排序>**以优化结果,并将最佳结果与新查询配对。这种复杂的方法确保搜索结果与用户的显式和隐式意图紧密一致,从而获得更有洞察力和相关性的信息发现。

#### 4. 路由 [Routing]

RAG系统的检索过程**利用不同的来源**,在领域、语言和格式上有所不同, 可以根据情况进行交替或合并。**查询路由决定了对用户查询的后续行动**, 选项包括**总结、搜索特定数据库或将不同路径合并为单一响应**。查询路由器还**<选择适合查询的数据存储>**,可能包括**向量存储、图数据库或关系数据库等不同来源**, 或者索引的层次结构——例如, 用于多文档存储的摘要索引和文档块向量索引。

查询路由器的决策是预定义的,通过LLM调用执行,指导查询到所选的索引。

#### 5. 额外生成模块[Extra Generation Module]

"额外生成模块" 解决了检索到的内容中常见的**冗余和噪音问题**。

该模块不直接从数据源检索, 而是**<利用LLM生成必要的上下文>**。与直接检索相比,LLM生成的内容更有可能包含相关信息

#### 6. 任务自适应模块[Task Adaptable Module]。

该模块专注于使RAG**适应各种下游任务**。

UPRISE自动从预构建的数据池中检索用于零样本任务输入的提示,从而增强了跨任务和模型的通用性。同时,PROMPTAGATOR利用LLM作为少样本查询生成器,**并根据生成的数据创建特定任务的检索器**。通过利用LLM的泛化能力,它能够在极少样本的情况下开发针对特定任务的端到端检索器。

## New Patterns

模块化RAG的组织结构具有高度的适应性, **允许在RAG过程中替换或重新安排模块,以适应特定的问题背景**。朴素RAG和高级RAG都可以被视为由一些固定模块组成。如图3所示,**朴素RAG主要由"检索"和"读取"模块组成**。典型的高级RAG模式在朴素RAG的基础上**增加了"重写"和"重排"模块**。然而,总的来说,模块化RAG享有更大的多样性和灵活性。

目前的研究主要探索两种组织范式。第一种涉及**添加或替换模块**,而第二种侧重于**调整模块之间的组织流程**。这种灵活性使得能够定制RAG过程,以有效地解决各种各样的任务。

### 1. 添加或替换模块[Adding or Replacing Modules]。

引入或替换模块的策略涉及**保持检索-读取过程的核心结构,同时集成额外的模块以增强特定功能**。RRR模型引入了重写-检索-读取过程,利用LLM性能作为重写模块的强化学习激励。这使得重写器能够微调检索查询,从而提高读取器的下游任务性能。

类似地,可以**选择性地交换Generate-Read等方法中的模块,其中LLM的生成模块取代了检索模块**。Recite-Read方法将外部检索转化为从模型权重中检索,要求LLM首先记忆特定任务的信息,然后生成能够处理知识密集型自然语言处理任务的输出。

### 2. 调整模块之间的流程 [Adjusting the Flow between Modules] 。

在模块流程调整领域,重点是**增强语言模型与检索模型之间的交互**。

DSP引入了演示-搜索-预测框架,将上下文学习系统视为一个显式程序,而不是最终的任务提示,从而更有效地处理知识密集型任务。ITER-RETGEN方法利用生成的内容来指导检索,在检索-读取-检索-读取流程中迭代实现 "检索增强生成"和"生成增强检索"。这种方法展示了一种创新的方式,使用一个模块的输出来改善另一个模块的功能。

## Optimizing the RAG Pipeline

检索过程的优化旨在提高RAG系统中信息的效率和质量。目前的研究重点是**集成不同的搜索技术,改进检索步骤,结合认知回溯,实施多样化的查询策略,并利用嵌入相似性**。这些努力共同致力于在检索效率和RAG系统中上下文信息的深度之间取得平衡。

### 1. 混合搜索探索[Hybrid Search Exploration]。

RAG系统通过智能地集成各种技术来优化其性能,包括基于关键词的搜索、语义搜索和向量搜索。这种方法利用每种方法的独特优势来适应不同类型的查询和信息需求,确保始终检索到高度相关且上下文丰富的信息。混合搜索的使用是检索策略的有力补充,从而增强了RAG流水线的总体效力。

### 2. 递归检索和查询引擎[Recursive Retrieval and Query Engine]。

递归检索涉及在初始检索阶段获取较小的块,以捕获关键语义。随后,在过程的后期阶段向LLM提供包含更多上下文信息的较大块。这种两步检索方法有助于在效率和提供上下文丰富的响应之间取得平衡。

3. StepBack-prompt 方法鼓励 LLM脱离具体实例, 围绕更广泛的概念和原则进行推理

4. 子查询[Sub-Queries]。根据不同的场景, 可采用不同的查询策略, 如使用查询引擎。

如使用 LlamaIndex 等框架提供的查询引擎框架提供的查询引擎, 利用树形查询、利用矢量查询, 或对数据块执行简单的顺序查询。

## 3 检索 Rerieval

### Enhancing Semantic Representations

- Chunk optimization
- Fine-tuning Embedding Models

### Aligning Queries and Documents

- Query Rewriting
- Embedding Transformation

### Aligning Retriever and LLM

- Fine-tuning Retriever
- Adapters

## 4 生成 Generation

### Post-retrieval with Frozen LLM

- Information Compression
- Reranking

### Fine-tuning LLM for RAG

- General Optimization Process
- Utilizing Contrastive Learning

## 5 增强 Augmentation

### RAG in Augmentation Stages

- Pre-training Stage

- Fine-tuning Stage
- Inference Stage

#### Augmentation Source

- Augmented with Unstructured Data
- LLMs-Generated Content in RAG

#### Augmentation Proces

- Iterative Retrieval
- Recursive Retrieval
- Adaptive Retrieval

## 6 RAG Evaluation

RAG模型的评估主要围绕两个关键部分展开: 检索模块 和 生成模块。

这种划分确保了对上下文质量和生成内容质量的全面评估。具体来说, RAG评估涉及三个质量维度和四个关键能力。

**质量维度**包括:

1. 上下文相关性[Context Relevance]: 衡量检索到的上下文的精确度和特异性。
2. 答案忠实度 [Answer Faithfulness]: 确保生成的答案与检索到的上下文保持一致,避免出现矛盾。
3. 答案相关性 [Answer Relevance] : 要求生成的答案与提出的问题直接相关,有效解决核心疑问。

**关键能力**包括:

1. 噪声鲁棒性[Noise Robustness]: 评估模型处理与问题相关但缺乏实质信息的噪声文档的能力。
2. 拒绝无关信息[Negative Rejection]: 评估模型在检索到的文档不包含回答问题所需知识时避免回应的能力。
3. 信息整合[Information Integration]: 评估模型综合多个文档中的信息以解决复杂问题的能力。
4. 反事实鲁棒性[Counterfactual Robustness]: 测试模型识别和忽略文档中已知不准确信息的能力,即使在被告知可能存在错误信息的情况下也能保持鲁棒。

为了量化评估以上质量维度和关键能力,研究者们提出了一系列评估基准和工具。

其中,**RGB**和**RECALL**基准侧重于评估RAG模型的关键能力, 而**RAGAS**、**ARES**和**TruLens**等自动化工具则利用LLM来判断质量得分。

此外,传统的信息检索和自然语言处理评估指标,如准确率[Accuracy]、完全匹配[Exact Match, EM]、召回率[Recall]、精确率[Precision]、重现率[Reappearance Rate, R-Rate]、余弦相似度[Cosine Similarity]、命中率[Hit Rate]、平均倒数排名[Mean Reciprocal Rank, MRR]和归一化折扣累积增益[Normalized Discounted Cumulative Gain, NDCG]等,也被广泛应用于RAG模型的评估中。

总的来说,RAG模型的评估需要兼顾检索质量和生成质量,并综合考虑不同的质量维度和关键能力。通过使用各种评估基准、工具和指标,研究者们可以全面评估RAG模型在不同应用场景下的性能,为其进一步改进和优化提供依据。

[\[2401.05856\] Seven Failure Points When Engineering a Retrieval Augmented Generation System \(arxiv.org\)](#)

### **第一罪(FP1): 内容缺失 (Missing Content) 。**

提问的问题,无法在被检索文档库中找到,最准确的答案是缺失的。理想情况下,RAG系统回应应该是“抱歉,我不知道答案”。然而,对于检索内容相关但没有相关答案的问题,系统可能被误导,给出一个response。

### **第二宗罪(FP2):**

检索的TopK内容缺失(Missed the Top Ranked Documents)。问题的答案在文档库中,但排名得分不够高,无法返回给用户。理论上,检索过程中所有文档都会被排名得分。然而,在实际操作中,会返回排名前K个文档,为了提高召回率,K不可能设置的无限大,必须基于LLM大模型的能力,折中选择的一个值。

### **第三宗罪(FP3): 未在上下文中 (Not in Context) - 整合策略局限性。**

从数据库中检索到了包含答案的文档,但在生成答案的过程中,这些文档并未被纳入上下文。当数据库返回许多文档时,会进行整合过程以获取答案,此时会发生这种情况。

### **第四宗罪(FP4): 未提取 (Not Extracted)**

答案存在于上下文中,但大型语言模型未能提取出正确的答案。通常,这是因为上下文中存在太多噪声或矛盾信息。简而言之,Retrival命名是对的,但是LLM根据Retrival回答问题出错。睁眼说瞎话的概率明显大于用户可以接受的概率(用户一般只能接收0.1%的错误概率)

### **第五宗罪(FP5):**

错误格式(Wrong Format)。问题涉及以某种格式(如表格或列表)提取信息,而大型语言模型忽略了这一指示。

### **第六宗罪(FP6):**

错误的特异性(Incorrect Specificity)。返回的答案包含在响应中,但不够具体或过于具体,无法满足用户需求。这种情况发生在RAG系统设计者对某个问题有期望的结果,例如教师对学生。在这种情况下,应该提供具体的教育内容和答案,而不仅仅是答案。当用户不确定如何提问并过于笼统时,也会出现特异性错误。



第七宗罪(FP7)：不完整（Incomplete）。

不完整的答案并非错误，但缺少一些信息，尽管这些信息存在于上下文中并可供提取。

FP	研究方向	具体描述
FP4	更多的上下文信息	大模型的窗口从4K增加到8K或者更大，LLM可以使用更多的上下文信息
FP1	语义缓存降低了成本和延迟	由于速率限制和LLM的成本，RAG系统在应对并发用户方面存在困难。使用常见问题的预检索能力，可以缓解内容缺失现象。
FP5~FP7	RAG“越狱”	LLM大模型fine-tuning，增加模型的基础能力
FP2, FP4	增加元信息	将文件名和块编号添加到检索到的上下文中有有助于读者提取所需信息。这对聊天对话很有用。
FP2 FP4~7	开源嵌入模型在处理小型文本方面表现更优。	在处理小型文本方面，开源句子嵌入模型的表现与闭源替代品相当。
FP2~7	RAG系统需要持续校准。	RAG系统在运行时接收未知输入，需要不断监控。
FP1 FP2	实现一个RAG配置流水线	一个RAG系统需要校准块大小、嵌入策略、分块策略、检索策略、整合策略、上下文大小和提示。
FP2,FP4	通过组装定制解决方案创建的RAG管道是次优的。	端到端训练模型，增强RAG的领域实用性
FP2~FP4	只有在运行时才能测试性能特征。	离线评估技术，如G-Evals看起来很有前景，但前提是能够获得标记过的问题和答案对。

## 7 RAG Future

### RAG 的未来发展方向

RAG技术的未来发展可以从三个方面展开:未来挑战、模态扩展和RAG生态系统。

在未来挑战方面,除了上下文长度和鲁棒性问题外,RAG还面临着其他一些亟待解决的难题。例如,如何在LLM中更好地发挥LLM的作用,不仅仅是生成最终答案,还包括**支持检索和评估等环节**; **如何建立适用于RAG模型的规模定律**,探索小模型是否有可能在某些任务中超越大模型;如何提高RAG系统的实用性,满足实际生产环境对检索效率、数据安全等方面的要求,等等。这些挑战为RAG研究指明了前进的方向。

在模态扩展方面, RAG正在将其应用拓展到图像、音频、视频、代码等多种模态的数据处理和生成任务中。一些代表性的工作包括:RA-CM3(用于文本-图像跨模态检索和生成)、BLIP-2(用于视觉语言预训练)、GSS(用于语音翻译)、UEOP(端到端语音识别)、RBPS(代码检索和修复)、CoK(知识图谱问答)等。多模态RAG的发展大大拓宽了RAG的应用边界,为构建更全面、更强大的智能系统奠定了基础。

在RAG生态系统方面, 未来的研究重点包括扩展RAG在垂直领域的应用、完善**RAG的评估体系**、**提高RAG的可解释性**、**丰富RAG的工具链**等。随着RAG在医疗、法律、教育等专业领域的应用日益增多,对领域知识的高效获取和表示变得越来越重要。与此同时,RAG模型的评估方法也需要与时俱进,开发出更细粒度的指标和工具,以全面衡量RAG在不同任务上的性能。此外,提高RAG的可解释性和透明。

## RAG 目前的局限性

[My knowledge] RAG技术目前仍然存在一些局限性,主要包括:

1. 上下文长度的限制: RAG的有效性受到LLM上下文窗口大小的制约。如何在信息不足(窗口过小)和信息冗余(窗口过大)之间取得平衡是一个关键问题。随着研究人员努力将LLM的上下文窗口扩展到几乎无限大,如何适配RAG以应对这些变化将是一个重要的研究方向。
2. 鲁棒性问题: 在检索过程中引入噪声或矛盾的信息可能会对RAG的输出质量产生不利影响。提高RAG面对对抗性或反事实输入时的鲁棒性正成为一个热门的研究课题。
3. 混合方法的优化: 将RAG与微调相结合被视为一种前景广阔的策略。然而,如何确定RAG和微调的最佳集成方式(顺序、交替还是端到端联合训练),以及如何同时发挥参数化和非参数化优势,仍有待进一步探索。

### Q&A with Claude-3

有人说, 当LLM上下文tokens数量发展到无穷大, 或者近似无穷的时候, RAG 就没有作用了, 因此RAG是一个权益之计, 你怎么看待?

对于有人提出的"当LLM上下文tokens数量发展到无穷大时, RAG就没有作用了"的观点,我持保留态度。诚然,随着LLM上下文窗口的不断扩大,它们能够存储和处理的知识量也在不断增加。这可能会在一定程度上减少对外部知识检索的依赖。然而,我认为RAG并非一个权宜之计,而是LLM的一个重要补充。

- 首先,无论LLM的上下文窗口有多大,都无法涵盖所有可能的知识。特别是在处理特定领域或实时更新的信息时,外部知识库的作用不可替代。
- 其次,RAG不仅仅是知识的来源,更是一种知识组织和表示的方式。通过将外部知识与LLM的内部表示相结合,RAG可以产生更丰富、更结构化的知识表示,从而支持更复杂的推理和问答任务。
- 此外,RAG还有助于提高LLM的可解释性和可追溯性。

通过将生成的内容与检索到的证据相关联,RAG使得LLM的决策过程更加透明和可信。这对于一些对可解释性有严格要求的应用场景(如医疗、法律等)尤为重要。

因此,我认为RAG和LLM是互补的关系,而非替代的关系。即使LLM的上下文窗口不断扩大,RAG仍然有其独特的价值和应用前景。未来,RAG和LLM的进一步融合和协同优化将是一个富有吸引力的研究方向。

我持有不同的想法. 跟我讨论一下.

- 首先,你说 "无论LLM的上下文窗口有多大,都无法涵盖所有可能的知识。特别是在处理特定领域或实时更新的信息时,外部知识库的作用不可替代"。但是当LLM上下文窗口无穷大的时候,我就能涵盖所有的知识,如果要处理特定领域实时更新的知识,我只要把这个实时更新的知识"增量",一并添加到我的对话就可以了,难道不是么? 上下文窗口已经无穷大了.
- 然后,你讲 "RAG还有助于提高LLM的可解释性和可追溯性。" 我想我将领域知识一并放到对话上下文中,我想LLM也是可以能够找到相应的知识的
- 对于你说的 "RAG不仅仅是知识的来源,更是一种知识组织和表示的方式。通过将外部知识与LLM的内部表示相结合,RAG可以产生更丰富、更结构化的知识表示,从而支持更复杂的推理和问答任务。" 我不是很清楚,你不妨说的再清楚些.

关于LLM上下文窗口无穷大时是否还需要RAG的问题,您提出了一个很有趣的观点。理论上,如果LLM的上下文窗口真的能够无限扩大,并且可以实时更新知识,那么确实有可能减少对外部知识库的依赖。但是,我认为在实践中,这种"无穷大"的上下文窗口可能难以实现,原因如下: (**核心会导致混乱冗余**)

1. 计算资源的限制:无限扩大上下文窗口需要巨大的计算资源,包括存储、内存和计算能力。即使对于当前最先进的硬件设施而言,这也是一个巨大的挑战。
2. 知识更新的效率:实时更新大规模LLM的知识库并非易事,特别是当知识更新非常频繁时(如新闻、社交媒体等场景)。相比之下,通过RAG从外部知识库中检索最新信息可能更加高效。
3. 知识的组织和管理:将所有知识都塞入LLM的上下文窗口可能会导致**知识的混乱和冗余**。RAG通过将外部知识库作为一个独立的、结构化的模块,可以更好地组织和管理不同来源、不同形式的知识。

关于可解释性和可追溯性,您提出了一个很好的点。将领域知识放入对话上下文中,LLM确实有可能找到相关的支撑证据。但是,RAG可以**更直接、更明确地将生成的内容与检索到的证据相关联**,从而提供更清晰的解释和追溯路径。此外,RAG还可以**利用外部知识库中的结构化信息(如知识图谱)来增强解释的逻辑性和连贯性**。

关于RAG作为一种知识组织和表示方式的优势,我的意思是: RAG不仅仅是简单地将外部知识拼接到LLM的输入中, 而是通过嵌入、注意力机制等技术, 将外部知识与LLM的内部表示进行深度融合。这种融合可以产生更抽象、更高层次的知识表示, 从而支持更复杂的推理任务。打个比方, 如果说LLM是一个"知识库", 那么RAG就像是一个"知识图谱", 它不仅存储了知识, 还刻画了知识之间的逻辑联系和推理路径。