

# 大模型面试宝典

深度+广度

2024  
新版



# 目录

大语言模型基础 .....	0
1. 目前主流的开源模型体系有哪些? .....	0
2. PREFIX LM 和 CAUSAL LM 区别是什么? .....	1
3. 为何现在的大模型大部分是DECODER ONLY结构 .....	2
4. LLMs复读机问题 .....	2
4.1 什么是 LLMs 复读机问题? .....	2
4.2 为什么会出现 LLMs 复读机问题? .....	2
4.3 如何缓解 LLMs 复读机问题? .....	3
5. 如何让大模型处理更长的文本? .....	3
大语言模型架构 .....	4
1. 讲讲对ATTENTION的理解? .....	4
2. ATTENTION的计算步骤是什么? .....	5
3. ATTENTION机制和传统的SEQ2SEQ模型有什么区别? .....	5
4. TRANSFORMER中MULTI-HEAD ATTENTION中每个HEAD为什么要进行降维? .....	6
5. ENCODER编码器与DECODER掩码有什么区别? .....	6
6. 为什么BERT选择MASK掉15%这个比例的词，可以是其他的比例吗? .....	6
7. BERT非线性的来源在哪里? .....	7
8. 为什么要进行LN呢? .....	7
训练数据集 .....	8
1. SFT (有监督微调) 的数据集格式? .....	8
2. RM (奖励模型) 的数据格式? .....	8
3. PPO (强化学习) 的数据格式? .....	9
有监督微调 .....	10
1. 微调方法是啥? 如何微调? .....	10
2. 微调和参数高效微调之间的区别是什么? .....	11
3. PEFT 有什么优点? .....	11
4. 多种不同的高效微调方法对比 .....	12



5. 当前高效微调技术存在的一些问题 .....	12
<b>推理 .....</b>	<b>13</b>
1. 为什么大模型推理时显存涨的那么多还一直占着? .....	13
2. 大模型在GPU和CPU上推理速度如何 .....	13
3. 推理速度上，INT8和FP16比起来怎么样? .....	14
4. 大模型有推理能力吗? .....	14
5. 大模型生成时的参数怎么设置? .....	14
6. 有哪些省内存的大语言模型训练/微调/推理方法? .....	15
<b>强化学习 .....</b>	<b>16</b>
1. 简单介绍强化学习? .....	16
2. 简单介绍一下 RLHF? .....	17
3. 奖励模型需要和基础模型一致吗? .....	19
4. RLHF 在实践过程中存在哪些不足? .....	19
5. 什么是 LLM AGENT? .....	20
6. LLM AGENT 有什么关键能力? .....	20
7. 怎样构建基于 LLM 的 AGENTS? .....	20
<b>大语言模型评估 .....</b>	<b>21</b>
1. 大模型怎么评测? .....	21
2. 大模型的 HONEST 原则是如何实现的？模型如何判断回答的知识是训练过的已知的知识，怎么训练这种能力? .....	21
3. 如何衡量大模型水平? .....	22
4. 大模型评估方法有哪些? .....	22
5. 什么是大模型幻觉? .....	22
6. 为什么需要解决LLM的幻觉问题? .....	23
7. 幻觉一定是有害的吗? .....	23
8. 幻觉有哪些不同类型? .....	23
9. 为什么LLM会产生幻觉? .....	23
10. 如何度量幻觉? .....	24
<b>大语言模型应用 .....</b>	<b>25</b>
1. 什么是 LANGCHAIN? .....	25
2. LANGCHAIN 包含哪些核心模块 .....	25



3. LANGCHAIN 如何调用 LLMs 生成回复？	26
4. LANGCHAIN 如何修改 提示模板？	26

# 大语言模型基础

## 1. 目前主流的开源模型体系有哪些？

目前主流的开源LLM（语言模型）模型体系包括以下几个：

- GPT (Generative Pre-trained Transformer) 系列：由OpenAI发布的一系列基于Transformer架构的语言模型，包括GPT、GPT-2、GPT-3等。GPT模型通过在大规模无标签文本上进行预训练，然后在特定任务上进行微调，具有很强的生成能力和语言理解能力。
- BERT (Bidirectional Encoder Representations from Transformers)：由Google发布的一种基于Transformer架构的双向预训练语言模型。BERT模型通过在大规模无标签文本上进行预训练，然后在下游任务上进行微调，具有强大的语言理解和表征能力。
- LLaMA模型：  
是一系列具有不同参数量的大规模语言模型，参数范围从70亿到650亿不等。由Meta公司（前身为Facebook）开发的一系列大规模语言模型
- 阿里云Qwen2：  
这是由阿里云发布的高性能开源模型，性能超越了许多中国的闭源模型，成为全球开发者的一个主流选择。
- AliceMind：  
达摩院（阿里巴巴的研究机构）开源的深度语言模型体系，覆盖了多语言、多模态和结构化数据等多个预训练语言模型。
- XLNet：由CMU和Google Brain发布的一种基于Transformer架构的自回归预训练语言模型。XLNet模型通过自回归方式预训练，可以建模全局依赖关系，具有更好的语言建模能力和生成能力。
- RoBERTa：由Facebook发布的一种基于Transformer架构的预训练语言模型。RoBERTa模型在BERT的基础上进行了改进，通过更大规模的数据和更长的训练时间，取得了更好的性能。
- T5 (Text-to-Text Transfer Transformer)：由Google发布的一种基于Transformer架构的多任务预训练语言模型。T5模型通过在大规模数据集上进行预训练，可以用于多种自然语言处理任务，如文本分类、机器翻译、问答等。

这些模型在自然语言处理领域取得了显著的成果，并被广泛应用于各种任务和应用中。



## 2. prefix LM 和 causal LM 区别是什么？

Prefix LM（前缀语言模型）和Causal LM（因果语言模型）是两种不同类型的语言模型，它们的区别在于生成文本的方式和训练目标。

### ● Prefix LM

Prefix LM其实是Encoder-Decoder模型的变体，为什么这样说？解释如下：

- 1) 在标准的Encoder-Decoder模型中，Encoder和Decoder各自使用一个独立的Transformer
- 2) 而在Prefix LM，Encoder和Decoder则共享了同一个Transformer结构，在Transformer内部通过Attention Mask机制来实现。

与标准Encoder-Decoder类似，Prefix LM在Encoder部分采用Auto Encoding (AE-自编码)模式，即前缀序列中任意两个token都相互可见，而Decoder部分采用Auto Regressive (AR-自回归)模式，即待生成的token可以看到Encoder侧所有token(包括上下文)和Decoder侧已经生成的token，但不能看未来尚未产生的token。

下面的图很形象地解释了Prefix LM的Attention Mask机制(左)及流转过程(右)。

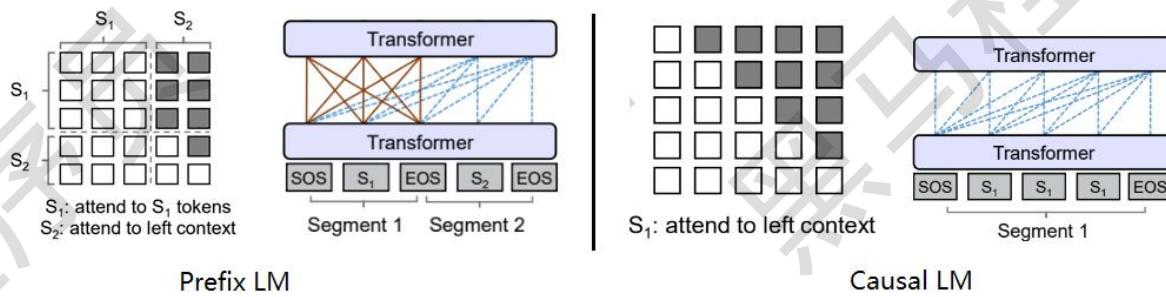
Prefix LM的代表模型有UniLM、T5、GLM(清华滴~)

### ● Causal LM

Causal LM是因果语言模型，目前流行地大多数模型都是这种结构，别无他因，因为GPT系列模型内部结构就是它，还有开源界的LLaMa也是。

Causal LM只涉及到Encoder-Decoder中的Decoder部分，采用Auto Regressive模式，直白地说，就是根据历史的token来预测下一个token，也是在Attention Mask这里做的手脚。

参照着Prefix LM，可以看下Causal LM的Attention Mask机制(左)及流转过程(右)。



### ● 总结



**Prefix LM:** 前缀语言模型是一种生成模型，它在生成每个词时都可以考虑之前的上下文信息。在生成时，前缀语言模型会根据给定的前缀（即部分文本序列）预测下一个可能的词。这种模型可以用于文本生成、机器翻译等任务。

**Causal LM:** 因果语言模型是一种自回归模型，它只能根据之前的文本生成后续的文本，而不能根据后续的文本生成之前的文本。在训练时，因果语言模型的目标是预测下一个词的概率，给定之前的所有词作为上下文。这种模型可以用于文本生成、语言建模等任务。

总结来说，前缀语言模型可以根据给定的前缀生成后续的文本，而因果语言模型只能根据之前的文本生成后续的文本。它们的训练目标和生成方式略有不同，适用于不同的任务和应用场景。

### 3. 为何现在的大模型大部分是Decoder only结构

- Encoder的低秩问题：Encoder的双向注意力会存在低秩问题，这可能会削弱模型表达能力，就生成任务而言，引入双向注意力并无实质好处。
- 更好的Zero-Shot性能、更适合于大语料自监督学习：decoder-only 模型在没有任何 tuning 数据的情况下、zero-shot 表现最好，而 encoder-decoder 则需要在一定量的标注数据上做 multitask finetuning 才能激发最佳性能。
- 效率问题：decoder-only支持一直复用KV-Cache，对多轮对话更友好，因为每个Token的表示之和它之前的输入有关，而encoder-decoder和PrefixLM就难以做到。

### 4. LLMs复读机问题

#### 4.1 什么是 LLMs 复读机问题？

LLMs复读机问题 (LLMs Parroting Problem) 是指大型语言模型在生成文本时过度依赖输入文本的复制，而缺乏创造性和独特性。当面对一个问题或指令时，模型可能会简单地复制输入文本的一部分或全部内容，并将其作为生成的输出，而不是提供有意义或新颖的回应。

#### 4.2 为什么会出现 LLMs 复读机问题？

- 1) **数据偏差：**大型语言模型通常是通过预训练阶段使用大规模无标签数据进行训练的。如果训练数据中存在大量的重复文本或者某些特定的句子或短语出现频率较高，模型在生成文本时可能会倾向于复制这些常见的模式。
- 2) **训练目标的限制：**大型语言模型的训练通常是基于自监督学习的方法，通过预测下一个词或掩盖词来学习语言模型。这样的训练目标可能使得模型更倾向于生成与输入相似的文本，导致复读机问题的出现。



- 3) **缺乏多样性的训练数据：**虽然大型语言模型可以处理大规模的数据，但如果训练数据中缺乏多样性的语言表达和语境，模型可能无法学习到足够的多样性和创造性，导致复读机问题的出现。
- 4) **模型结构和参数设置：**大型语言模型的结构和参数设置也可能对复读机问题产生影响。例如，模型的注意力机制和生成策略可能导致模型更倾向于复制输入的文本。

## 4.3 如何缓解 LLMs 复读机问题？

为了缓解LLMs复读机问题，可以尝试以下方法：

- 1) **多样性训练数据：**在训练阶段，使用多样性的语料库来训练模型，避免数据偏差和重复文本的问题。这可以包括从不同领域、不同来源和不同风格的文本中获取数据。
- 2) **引入噪声：**在生成文本时，引入一些随机性或噪声，例如通过采样不同的词或短语，或者引入随机的变换操作，以增加生成文本的多样性。这可以通过在生成过程中对模型的输出进行采样或添加随机性来实现。
- 3) **温度参数调整：**温度参数是用来控制生成文本的多样性的一个参数。通过调整温度参数的值，可以控制生成文本的独创性和多样性。较高的温度值会增加随机性，从而减少复读机问题的出现。
- 4) **Beam搜索调整：**在生成文本时，可以调整Beam搜索算法的参数。Beam搜索是一种常用的生成策略，它在生成过程中维护了一个候选序列的集合。通过调整Beam大小和搜索宽度，可以控制生成文本的多样性和创造性。
- 5) **后处理和过滤：**对生成的文本进行后处理和过滤，去除重复的句子或短语，以提高生成文本的质量和多样性。可以使用文本相似度计算方法或规则来检测和去除重复的文本。
- 6) **人工干预和控制：**对于关键任务或敏感场景，可以引入人工干预和控制机制，对生成的文本进行审查和筛选，确保生成结果的准确性和多样性。

需要注意的是，缓解LLMs复读机问题是一个复杂的任务，没有一种通用的解决方案。不同的方法可能适用于不同的场景和任务，需要根据具体情况进行选择和调整。此外，解决复读机问题还需要综合考虑数据、训练目标、模型架构和生成策略等多个因素，需要进一步的研究和实践来提高大型语言模型的生成文本多样性和创造性。

## 5. 如何让大模型处理更长的文本？

要让大模型处理更长的文本，可以考虑以下几个方法：

- 1) **分块处理：**将长文本分割成较短的片段，然后逐个片段输入模型进行处理。这样可以避免长文本对模型内存和计算资源的压力。在处理分块文本时，可以使用重叠的方式，即将相邻片段的



一部分重叠，以保持上下文的连贯性。

- 2) **层次建模**: 通过引入层次结构，将长文本划分为更小的单元。例如，可以将文本分为段落、句子或子句等层次，然后逐层输入模型进行处理。这样可以减少每个单元的长度，提高模型处理长文本的能力。
- 3) **部分生成**: 如果只需要模型生成文本的一部分，而不是整个文本，可以只输入部分文本作为上下文，然后让模型生成所需的部分。例如，输入前一部分文本，让模型生成后续的内容。
- 4) **注意力机制**: 注意力机制可以帮助模型关注输入中的重要部分，可以用于处理长文本时的上下文建模。通过引入注意力机制，模型可以更好地捕捉长文本中的关键信息。
- 5) **模型结构优化**: 通过优化模型结构和参数设置，可以提高模型处理长文本的能力。例如，可以增加模型的层数或参数量，以增加模型的表达能力。还可以使用更高效的模型架构，如Transformer等，以提高长文本的处理效率。

需要注意的是，处理长文本时还需考虑计算资源和时间的限制。较长的文本可能需要更多的内存和计算时间，因此在实际应用中需要根据具体情况进行权衡和调整。

## 大语言模型架构

### 1. 讲讲对Attention的理解?

Attention机制是一种在处理时序相关问题的时候常用的技术，主要用于处理序列数据。

核心思想是在处理序列数据时，网络应该更关注输入中的重要部分，而忽略不重要的部分，它通过学习不同部分的权重，将输入的序列中的重要部分显式地加权，从而使得模型可以更好地关注与输出有关的信息。

在序列建模任务中，比如机器翻译、文本摘要、语言理解等，输入序列的不同部分可能具有不同的重要性。传统的循环神经网络（RNN）或卷积神经网络（CNN）在处理整个序列时，难以捕捉到序列中不同位置的重要程度，可能导致信息传递不够高效，特别是在处理长序列时表现更明显。

Attention机制的关键是引入一种机制来动态地计算输入序列中各个位置的权重，从而在每个时间步上，对输入序列的不同部分进行加权求和，得到当前时间步的输出。这样就实现了模型对输入中不同部分的关注度的自适应调整。



## 2. Attention的计算步骤是什么？

具体的计算步骤如下：

- 1) **计算查询 (Query)**：查询是当前时间步的输入，用于和序列中其他位置的信息进行比较。
- 2) **计算键 (Key) 和值 (Value)**：键表示序列中其他位置的信息，值是对应位置的表示。键和值用来和查询进行比较。
- 3) **计算注意力权重**：通过将查询和键进行内积运算，然后应用softmax函数，得到注意力权重。这些权重表示了在当前时间步，模型应该关注序列中其他位置的重要程度。
- 4) **加权求和**：根据注意力权重将值进行加权求和，得到当前时间步的输出。

在Transformer中，Self-Attention 被称为"Scaled Dot-Product Attention"，其计算过程如下：

- 1) 对于输入序列中的每个位置，通过计算其与所有其他位置之间的相似度得分（通常通过点积计算）。
- 2) 对得分进行缩放处理，以防止梯度爆炸。
- 3) 将得分用softmax函数转换为注意力权重，以便计算每个位置的加权和。
- 4) 使用注意力权重对输入序列中的所有位置进行加权求和，得到每个位置的自注意输出。

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

## 3. Attention机制和传统的Seq2Seq模型有什么区别？

Seq2Seq模型是一种基于编码器-解码器结构的模型，主要用于处理序列到序列的任务，例如机器翻译、语音识别等。

传统的Seq2Seq模型只使用编码器来捕捉输入序列的信息，而解码器只从编码器的最后状态中获取信息，并将其用于生成输出序列。

而Attention机制则允许解码器在生成每个输出时，根据输入序列的不同部分给予不同的注意力，从而使得模型更好地关注到输入序列中的重要信息。



## 4. transformer中multi-head attention中每个head为什么要进行降维？

在Transformer的Multi-Head Attention中，对每个head进行降维是为了增加模型的表达能力和效率。

每个head是独立的注意力机制，它们可以学习不同类型的特征和关系。通过使用多个注意力头，Transformer可以并行地学习多种不同的特征表示，从而增强了模型的表示能力。

然而，在使用多个注意力头的同时，注意力机制的计算复杂度也会增加。原始的Scaled Dot-Product Attention的计算复杂度为 $O(d^2)$ ，其中 $d$ 是输入向量的维度。如果使用 $h$ 个注意力头，计算复杂度将增加到 $O(hd^2)$ 。这可能会导致Transformer在处理大规模输入时变得非常耗时。

为了缓解计算复杂度的问题，Transformer中在每个head上进行降维。在每个注意力头中，输入向量通过线性变换被映射到一个较低维度的空间。这个降维过程使用两个矩阵：一个是查询（Q）和键（K）的降维矩阵 $W_q$ 和 $W_k$ ，另一个是值（V）的降维矩阵 $W_v$ 。

通过降低每个head的维度，Transformer可以在保持较高的表达能力的同时，大大减少计算复杂度。降维后的计算复杂度为 $O(h\hat{d}^2)$ ，其中 $\hat{d}$ 是降维后的维度。通常情况下， $\hat{d}$ 会远小于原始维度 $d$ ，这样就可以显著提高模型的计算效率。

## 5. Encoder编码器与Decoder掩码有什么区别？

Encoder主要使用自注意力掩码和填充掩码，而Decoder除了自注意力掩码外，还需要使用编码器-解码器注意力掩码来避免未来位置信息的泄露。这些掩码操作保证了Transformer在处理自然语言序列时能够准确、有效地进行计算，从而获得更好的表现。

## 6. 为什么BERT选择mask掉15%这个比例的词，可以是其他的比例吗？

BERT选择mask掉15%的词是一种经验性的选择，是原论文中的一种选择，并没有一个固定的理论依据，实际中当然可以尝试不同的比例，15%的比例是由BERT的作者在原始论文中提出，并在实验中发现对于BERT的训练效果是有效的。



## 7. BERT非线性的来源在哪里？

主要来自两个地方：前馈层的gelu激活函数和self-attention。

- **前馈神经网络层：**在BERT的Encoder中，每个自注意力层之后都跟着一个前馈神经网络层。前馈神经网络层是全连接的神经网络，通常包括一个线性变换和一个非线性的激活函数，如gelu。这样的非线性激活函数引入了非线性变换，使得模型能够学习更加复杂的特征表示。
- **self-attention layer：**在自注意力层中，查询（Query）、键（Key）、值（Value）之间的点积得分会经过softmax操作，形成注意力权重，然后将这些权重与值向量相乘得到每个位置的自注意力输出。这个过程中涉及了softmax操作，使得模型的计算是非线性的。

## 8. 为什么要进行LN呢？

使用Layer Normalization (LN) 的主要原因是为了改善训练过程的稳定性并加速收敛。下面是一些具体的原因和解释：

1. 消除梯度消失和爆炸：在深度神经网络中，梯度消失和爆炸问题是常见的，尤其是在使用递归神经网络 (RNNs) 和长短期记忆网络 (LSTMs) 时。通过在每一层对激活值进行归一化，LN可以帮助控制梯度的大小，从而减轻这些问题。
2. 加速收敛：LN通过减少内部协变量移位 (internal covariate shift) 来帮助模型更快地收敛。内部协变量移位指的是网络中层的输入分布变化，这会增加优化的难度。通过保持每一层输入的稳定分布，LN有助于加速训练过程。
3. 适应不同批量大小：相比于Batch Normalization (BN)，LN的一个显著优点是它不依赖于批量大小。BN通常需要较大的批量来获得准确的均值和方差估计，这对于NLP任务可能不是最优的，因为NLP模型往往需要处理变长的输入序列，并且较大的批量可能会占用过多的内存。LN直接对每一层的特征进行归一化，因此适用于小批量或甚至单个样本。
4. 序列内部一致性：在NLP中，我们通常希望一个句子内部的表示（即隐藏状态）具有一定的内部一致性。LN通过在特征维度上进行操作，确保了同一层中每个样本的特征向量具有相似的尺度和分布，这对于维持序列内的一致性很重要。
5. Transformer架构的需求：Transformer模型使用自注意力机制，这种机制对输入的尺度敏感。LN在Transformer中尤其重要，因为它有助于稳定自注意力层的输入，从而提高整个模型的性能。
6. 模型泛化能力：通过规范化每一层的输出，LN有助于模型在不同的输入上获得更加一致的表现，从而可能提升模型的泛化能力。



# 训练数据集

## 1. SFT (有监督微调) 的数据集格式?

对于大语言模型的训练中，SFT (Supervised Fine-Tuning) 的数据集格式可以采用以下方式：

- 1) **输入数据**: 输入数据是一个文本序列，通常是一个句子或者一个段落。每个样本可以是一个字符串或者是一个tokenized的文本序列。
- 2) **标签数据**: 标签数据是与输入数据对应的标签或类别。标签可以是单个类别，也可以是多个类别的集合。对于多分类任务，通常使用one-hot编码或整数编码来表示标签。
- 3) **数据集划分**: 数据集通常需要划分为训练集、验证集和测试集。训练集用于模型的训练，验证集用于调整模型的超参数和监控模型的性能，测试集用于评估模型的最终性能。
- 4) **数据集格式**: 数据集可以以文本文件（如CSV、JSON等）或数据库的形式存储。每个样本包含输入数据和对应的标签。可以使用表格形式存储数据，每一列代表一个特征或标签。

下面是一个示例数据集的格式：

```
Input,Label
"This is a sentence.",1
"Another sentence.",0
...
```

在这个示例中，输入数据是一个句子，标签是一个二分类的标签（1代表正例，0代表负例）。每一行代表一个样本，第一列是输入数据，第二列是对应的标签。

需要注意的是，具体的数据集格式可能会因任务类型、数据来源和使用的深度学习框架而有所不同。因此，在进行SFT训练时，建议根据具体任务和框架的要求来定义和处理数据集格式。

## 2. RM (奖励模型) 的数据格式?

在大语言模型训练中，RM (Reward Model, 奖励模型) 的数据格式可以采用以下方式：

- 1) **输入数据**: 输入数据是一个文本序列，通常是一个句子或者一个段落。每个样本可以是一个字符串或者是一个tokenized的文本序列。
- 2) **奖励数据**: 奖励数据是与输入数据对应的奖励或评分。奖励可以是一个实数值，表示对输入数据的评价。也可以是一个离散的标签，表示对输入数据的分类。奖励数据可以是人工标注的，也可以是通过其他方式（如人工评估、强化学习等）得到的。



- 3) **数据集格式:** 数据集可以以文本文件（如CSV、JSON等）或数据库的形式存储。每个样本包含输入数据和对应的奖励数据。可以使用表格形式存储数据，每一列代表一个特征或标签。

下面是一个示例数据集的格式：

**Input, Reward**

```
"This is a sentence.", 0.8  
"Another sentence.", 0.2  
...  
...
```

在这个示例中，输入数据是一个句子，奖励数据是一个实数值，表示对输入数据的评价。每一行代表一个样本，第一列是输入数据，第二列是对应的奖励数据。

需要注意的是，具体的数据集格式可能会因任务类型、数据来源和使用的深度学习框架而有所不同。因此，在使用RM进行大语言模型训练时，建议根据具体任务和框架的要求来定义和处理数据集格式。

### 3. PPO (强化学习) 的数据格式？

在大语言模型训练中，PPO (Proximal Policy Optimization, 近端策略优化) 是一种常用的强化学习算法。PPO的数据格式可以采用以下方式：

- 1) **输入数据:** 输入数据是一个文本序列，通常是一个句子或者一个段落。每个样本可以是一个字符串或者是一个tokenized的文本序列。
- 2) **奖励数据:** 奖励数据是与输入数据对应的奖励或评分。奖励可以是一个实数值，表示对输入数据的评价。也可以是一个离散的标签，表示对输入数据的分类。奖励数据可以是人工标注的，也可以是通过其他方式（如人工评估、模型评估等）得到的。
- 3) **动作数据:** 动作数据是模型在给定输入数据下的输出动作。对于语言模型，动作通常是生成的文本序列。动作数据可以是一个字符串或者是一个tokenized的文本序列。
- 4) **状态数据:** 状态数据是模型在给定输入数据和动作数据下的状态信息。对于语言模型，状态数据可以是模型的隐藏状态或其他中间表示。状态数据的具体形式可以根据具体任务和模型结构进行定义。
- 5) **数据集格式:** 数据集可以以文本文件（如CSV、JSON等）或数据库的形式存储。每个样本包含输入数据、奖励数据、动作数据和状态数据。可以使用表格形式存储数据，每一列代表一个特征或标签。

下面是一个示例数据集的格式：

**Input, Reward, Action, State**



```
"This is a sentence.",0.8,"This is a generated sentence.",[0.1, 0.2, 0.3, ...]
```

```
"Another sentence.",0.2,"Another generated sentence.",[0.4, 0.5, 0.6, ...]
```

```
...
```

在这个示例中，输入数据是一个句子，奖励数据是一个实数值，动作数据是生成的句子，状态数据是模型的隐藏状态。每一行代表一个样本，第一列是输入数据，第二列是对应的奖励数据，第三列是生成的动作数据，第四列是状态数据。

需要注意的是，具体的数据集格式可能会因任务类型、数据来源和使用的深度学习框架而有所不同。因此，在使用PPO进行大语言模型训练时，建议根据具体任务和框架的要求来定义和处理数据集格式。

## 有监督微调

### 1. 微调方法是啥？如何微调？

微调（Fine-tuning）是一种迁移学习的方法，用于在一个预训练模型的基础上，通过在特定任务的数据上进行有监督训练，来适应该任务的要求并提高模型性能。微调利用了预训练模型在大规模通用数据上学习到的语言知识和表示能力，将其迁移到特定任务上。

下面是一般的微调步骤：

- 1) **预训练模型选择**：选择一个在大规模数据上进行预训练的模型作为基础模型。例如，可以选择一种预训练的语言模型，如BERT、GPT等。
- 2) **数据准备**：准备用于微调的特定任务数据集。这些数据集应包含任务相关的样本和相应的标签或目标。确保数据集与任务的特定领域或问题相关。
- 3) 构建任务特定的模型头：根据任务的要求，构建一个特定的模型头（task-specific head）。模型头是添加到预训练模型之上的额外层或结构，用于根据任务要求进行输出预测或分类。例如，对于文本分类任务，可以添加一个全连接层和softmax激活函数。
- 4) **参数初始化**：将预训练模型的参数作为初始参数加载到微调模型中。这些参数可以被视为模型已经学习到的通用语言表示。
- 5) **微调训练**：使用特定任务的数据集对模型进行有监督训练。这包括将任务数据输入到模型中，计算损失函数，并通过反向传播和优化算法（如梯度下降）更新模型参数。在微调过程中，只有模型头的参数会被更新，而预训练模型的参数会保持不变。



- 6) **调整超参数**: 微调过程中, 可以根据需要调整学习率、批量大小、训练迭代次数等超参数, 以达到更好的性能。
- 7) **评估和验证**: 在微调完成后, 使用验证集或测试集对微调模型进行评估, 以评估其在特定任务上的性能。可以使用各种指标, 如准确率、精确率、召回率等。
- 8) **可选的后续微调**: 根据实际情况, 可以选择在特定任务的数据上进行进一步的微调迭代, 以进一步提高模型性能。

微调的关键是在预训练模型的基础上进行训练, 从而将模型的知识迁移到特定任务上。通过这种方式, 可以在较少的数据和计算资源下, 快速构建和训练高性能的模型。

## 2. 微调和参数高效微调之间的区别是什么?

微调和参数高效微调是机器学习中用于提高预训练模型在特定任务上的性能的两种方法。

**微调**就是把一个预先训练好的模型用新的数据在一个新的任务上进一步训练它。整个预训练模型通常在微调中进行训练, 包括它的所有层和参数。这个过程在计算上非常昂贵且耗时, 特别是对于大型模型。

另一方面, **参数高效微调**是一种专注于只训练预训练模型参数的子集的微调方法。这种方法包括为新任务识别最重要的参数, 并且只在训练期间更新这些参数。这样, PEFT可以显著减少微调所需的计算量。

## 3. PEFT 有什么优点?

在这里, 只讨论PEFT相对于传统微调的好处。因此, 理解为什么参数有效的微调比微调更有益。

- 1) **减少计算和存储成本**: PEFT只涉及微调少量额外的模型参数, 而冻结预训练LLM的大部分参数, 从而显着降低计算和存储成本
- 2) **克服灾难性遗忘**: 在LLM的全面微调期间, 灾难性遗忘可能发生在模型忘记它在预训练期间学到的知识的地方。PEFT通过只更新几个参数来克服这个问题。
- 3) **低数据环境下更好的性能**: PEFT方法在低数据环境下的表现优于完全微调, 并且可以更好地推广到域外场景。
- 4) **可移植性**: 与全面微调的大检查点相比, PEFT方法使用户能够获得价值几mb的小检查点。这使得来自PEFT方法的训练权重易于部署和用于多个任务, 而无需替换整个模型。
- 5) **与完全微调相当的性能**: PEFT仅使用少量可训练参数即可实现与完全微调相当的性能。



## 4. 多种不同的高效微调方法对比

参数有效策略可能涉及多种技术：

- 1) **选择性层调整 (Selective Layer Tuning)**：可以只微调层的一个子集，而不是微调模型的所有层。这减少了需要更新的参数数量。
- 2) **适配器 (Adapters)**：适配器层是插入预训练模型层之间的小型神经网络。在微调过程中，只训练这些适配器层，保持预先训练的参数冻结。通过这种方式，适配器学习将预先训练的模型提取的特征适应新任务。
- 3) **稀疏微调 (Sparse Fine-Tuning)**：传统的微调会略微调整所有参数，但稀疏微调只涉及更改模型参数的一个子集。这通常是基于一些标准来完成的，这些标准标识了与新任务最相关的参数。
- 4) **低秩近似 (Low-Rank Approximations)**：另一种策略是用一个参数较少但在任务中表现相似的模型来近似微调后的模型。
- 5) **正则化技术 (Regularization Techniques)**：可以将正则化项添加到损失函数中，以阻止参数发生较大变化，从而以更“参数高效”的方式有效地微调模型。
- 6) **任务特定的头 (Task-specific Heads)**：有时，在预先训练的模型架构中添加一个任务特定的层或“头”，只对这个头进行微调，从而减少需要学习的参数数量。

## 5. 当前高效微调技术存在的一些问题

当前的高效微调技术很难在类似方法之间进行直接比较并评估它们的真实性能，主要的原因如下所示：

- 1) **参数计算口径不一致**：参数计算可以分为三类：可训练参数的数量、微调模型与原始模型相比改变的参数的数量、微调模型和原始模型之间差异的等级。例如，DiffPruning更新0.5%的参数，但是实际参与训练的参数量是200%。这为比较带来了困难。尽管可训练的参数量是最可靠的存储高效指标，但是也不完美。Ladder-side Tuning使用一个单独的小网络，参数量高于LoRA或BitFit，但是因为反向传播不经过主网络，其消耗的内存反而更小。
- 2) **缺乏模型大小的考虑**：已有工作表明，大模型在微调中需要更新的参数量更小（无论是以百分比相对而论还是以绝对数量而论），因此（基）模型大小在比较不同PEFT方法时也要考虑到。
- 3) **缺乏测量基准和评价标准**：不同方法所使用的模型/数据集组合都不一样，评价指标也不一样，难以得到有意义的结论。
- 4) **代码实现可读性差**：很多开源代码都是简单拷贝Transformer代码库，然后进行小修小补。这些拷贝也不使用git fork，难以找出改了哪里。即便是能找到，可复用性也比较差（通常指定某个Transformer版本，没有说明如何脱离已有代码库复用这些方法）。



# 推理

## 1. 为什么大模型推理时显存涨的那么多还一直占着？

大语言模型进行推理时，显存涨得很多且一直占着显存不释放的原因主要有以下几点：

**模型参数占用显存：**大语言模型通常具有巨大的参数量，这些参数需要存储在显存中以供推理使用。因此，在推理过程中，模型参数会占用相当大的显存空间。

**输入数据占用显存：**进行推理时，需要将输入数据加载到显存中。对于大语言模型而言，输入数据通常也会占用较大的显存空间，尤其是对于较长的文本输入。

**中间计算结果占用显存：**在推理过程中，模型会进行一系列的计算操作，生成中间结果。这些中间结果也需要存储在显存中，以便后续计算使用。对于大语言模型而言，中间计算结果可能会占用较多的显存空间。

**内存管理策略：**某些深度学习框架在推理时采用了一种延迟释放显存的策略，即显存不会立即释放，而是保留一段时间以备后续使用。这种策略可以减少显存的分配和释放频率，提高推理效率，但也会导致显存一直占用的现象。

需要注意的是，显存的占用情况可能会受到硬件设备、深度学习框架和模型实现的影响。不同的环境和设置可能会导致显存占用的差异。如果显存占用过多导致资源不足或性能下降，可以考虑调整模型的批量大小、优化显存分配策略或使用更高性能的硬件设备来解决问题。

## 2. 大模型在GPU和CPU上推理速度如何

大语言模型在GPU和CPU上进行推理的速度存在显著差异。一般情况下，GPU在进行深度学习推理任务时具有更高的计算性能，因此大语言模型在GPU上的推理速度通常会比在CPU上更快。

以下是GPU和CPU在大语言模型推理速度方面的一些特点：

- 1) **GPU推理速度快：**GPU具有大量的并行计算单元，可以同时处理多个计算任务。对于大语言模型而言，GPU可以更高效地执行矩阵运算和神经网络计算，从而加速推理过程。
- 2) **CPU推理速度相对较慢：**相较于GPU，CPU的计算能力较弱，主要用于通用计算任务。虽然CPU也可以执行大语言模型的推理任务，但由于计算能力有限，推理速度通常会较慢。
- 3) **使用GPU加速推理：**为了充分利用GPU的计算能力，通常会使用深度学习框架提供的GPU加速功能，如CUDA或OpenCL。这些加速库可以将计算任务分配给GPU并利用其并行计算能力，



从而加快大语言模型的推理速度。

需要注意的是，推理速度还受到模型大小、输入数据大小、计算操作的复杂度以及硬件设备的性能等因素的影响。因此，具体的推理速度会因具体情况而异。一般来说，使用GPU进行大语言模型的推理可以获得更快的速度。

### 3. 推理速度上，INT8和FP16比起来怎么样？

在大语言模型的推理速度上，使用INT8（8位整数量化）和FP16（半精度浮点数）相对于FP32（单精度浮点数）可以带来一定的加速效果。这是因为INT8和FP16的数据类型在表示数据时所需的内存和计算资源较少，从而可以加快推理速度。

具体来说，INT8在相同的内存空间下可以存储更多的数据，从而可以在相同的计算资源下进行更多的并行计算。这可以提高每秒推理操作数（Operations Per Second, OPS）的数量，加速推理速度。

FP16在相对较小的数据范围内进行计算，因此在相同的计算资源下可以执行更多的计算操作。虽然FP16的精度相对较低，但对于某些应用场景，如图像处理和语音识别等，FP16的精度已经足够满足需求。

需要注意的是，INT8和FP16的加速效果可能会受到硬件设备的支持程度和具体实现的影响。某些硬件设备可能对INT8和FP16有更好的优化支持，从而进一步提高推理速度。

综上所述，使用INT8和FP16数据类型可以在大语言模型的推理过程中提高推理速度，但需要根据具体场景和硬件设备的支持情况进行评估和选择。

### 4. 大模型有推理能力吗？

逻辑推理是大语言模型“智能涌现”出的核心能力之一，好像AI有了人的意识一样。而推理能力的关键，在于一个技术——思维链（Chain of Thought, CoT）。当模型规模足够大的时候，LLM本身是具备推理能力的。在简单推理问题上，LLM已经达到了很好的能力；复杂推理问题上，还需要更多深入的研究。

### 5. 大模型生成时的参数怎么设置？

- Temperature：用于调整随机从生成模型中抽样的程度，使得相同的提示可能会产生不同的输出。



出。温度为 0 将始终产生相同的输出，该参数设置越高随机性越大。

- **波束搜索宽度**: 波束搜索是许多 NLP 和语音识别模型中常用的一种算法，作为在给定可能选项的情况下选择最佳输出的最终决策步骤。波束搜索宽度是一个参数，用于确定算法在搜索的每个步骤中应该考虑的候选数量。
- Top p: 动态设置tokens候选列表的大小\*\*。\*\* 将可能性之和不超过特定值的top tokens列入候选名单。Top p 通常设置为较高的值(如 0.75)，目的是限制可能被采样的低概率 token 的长度。
- Top k: 允许其他高分tokens有机会被选中\*\*。\*\* 这种采样引入的随机性有助于在很多情况下生成的质量。Top k 参数设置为 3 则意味着选择前三个tokens。

若 Top k 和 Top p 都启用，则 Top p 在 Top k 之后起作用

## 6. 有哪些省内存的大语言模型训练/微调/推理方法？

有一些方法可以帮助省内存的大语言模型训练、微调和推理，以下是一些常见的方法：

- 1) **参数共享 (Parameter Sharing)** : 通过共享模型中的参数，可以减少内存占用。例如，可以在不同的位置共享相同的嵌入层或注意力机制。
- 2) **梯度累积 (Gradient Accumulation)** : 在训练过程中，将多个小批次的梯度累积起来，然后进行一次参数更新。这样可以减少每个小批次的内存需求，特别适用于GPU内存较小的情况。
- 3) **梯度裁剪 (Gradient Clipping)** : 通过限制梯度的大小，可以避免梯度爆炸的问题，从而减少内存使用。
- 4) **分布式训练 (Distributed Training)** : 将训练过程分布到多台机器或多个设备上，可以减少单个设备的内存占用。分布式训练还可以加速训练过程。
- 5) **量化 (Quantization)** : 将模型参数从高精度表示（如FP32）转换为低精度表示（如INT8或FP16），可以减少内存占用。量化方法可以通过减少参数位数或使用整数表示来实现。
- 6) **剪枝 (Pruning)** : 通过对去除冗余或不重要的模型参数，可以减少模型的内存占用。剪枝方法可以根据参数的重要性进行选择，从而保持模型性能的同时减少内存需求。
- 7) **蒸馏 (Knowledge Distillation)** : 使用较小的模型（教师模型）来指导训练较大的模型（学生模型），可以从教师模型中提取知识，减少内存占用。
- 8) **分块处理 (Chunking)** : 将输入数据或模型分成较小的块进行处理，可以减少内存需求。例如，在推理过程中，可以将较长的输入序列分成多个较短的子序列进行处理。

这些方法可以结合使用，根据具体场景和需求进行选择和调整。同时，不同的方法可能对不同的模型和任务有不同的效果，因此需要进行实验和评估。

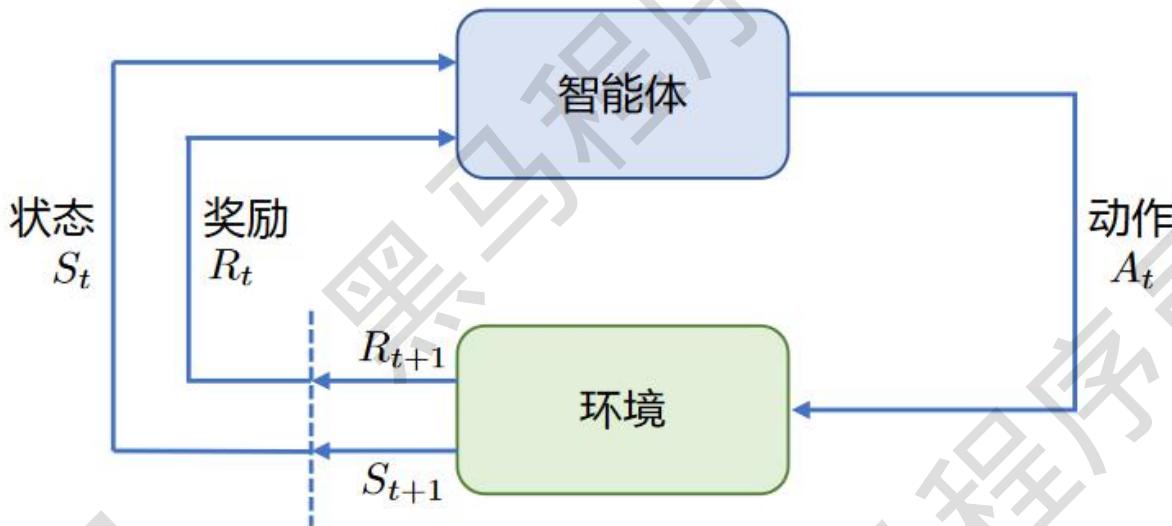


# 强化学习

## 1. 简单介绍强化学习？

强化学习 (Reinforcement Learning, RL) 研究的问题是智能体 (Agent) 与环境 (Environment) 交互的问题，其目标是使智能体在复杂且不确定的环境中最大化奖励 (Reward)。

强化学习基本框架如图所示，主要由两部分组成：**智能体**和**环境**。在强化学习过程中，智能体与环境不断交互。智能体在环境中获取某个状态后，会根据该状态输出一个动作 (Action)，也称为决策 (Decision)。动作会在环境中执行，环境会根据智能体采取的动作，给出下一个状态以及当前动作所带来的奖励。智能体的目标就是尽可能多地从环境中获取奖励。本节中将介绍强化学习的基本概念、强化学习与有监督学习的区别，以及在大语言模型中基于人类反馈的强化学习流程。



强化学习在大语言模型上的重要作用可以概括为以下几个方面：

- 1) 强化学习比有监督学习更可以考虑整体影响：有监督学习针对单个词元进行反馈，其目标是要求模型针对给定的输入给出的确切答案。而强化学习是针对整个输出文本进行反馈，并不针对特定的词元。
- 2) 强化学习更容易解决幻觉问题：有监督学习算法非常容易使得求知型查询产生幻觉。在模型并不包含或者知道答案的情况下，有监督训练仍然会促使模型给出答案。而使用强化学习方法，则可以通过定制奖励函数，将正确答案赋予非常高的分数，放弃回答的答案赋予中低分



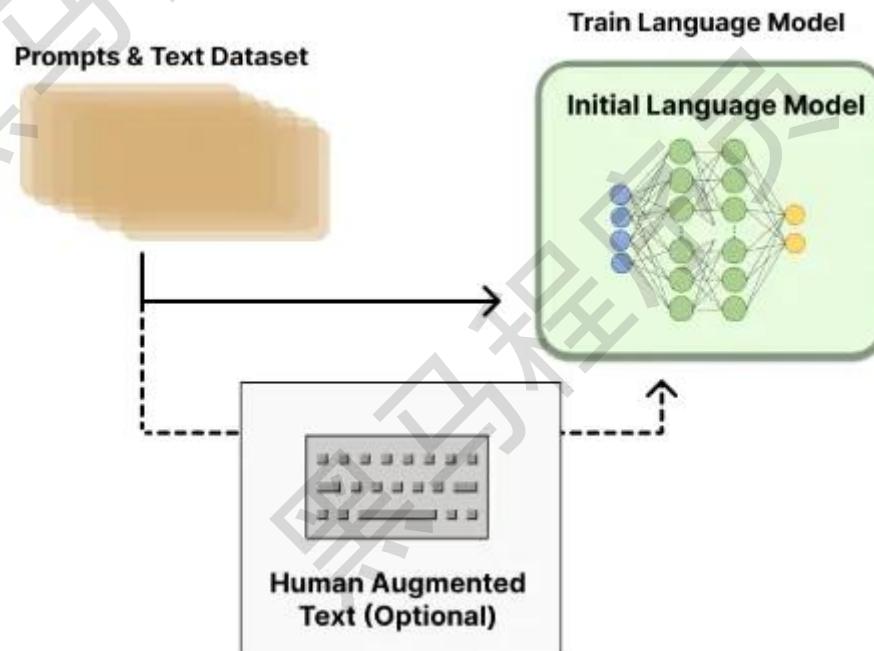
数，不正确的答案赋予非常高的负分，使得模型学会依赖内部知识选择放弃回答，从而在一定程度上缓解模型幻觉问题。

- 3) **强化学习可以更好的解决多轮对话奖励累积问题**: 使用强化学习方法，可以通过构建奖励函数，将当前输出考虑整个对话的背景和连贯性

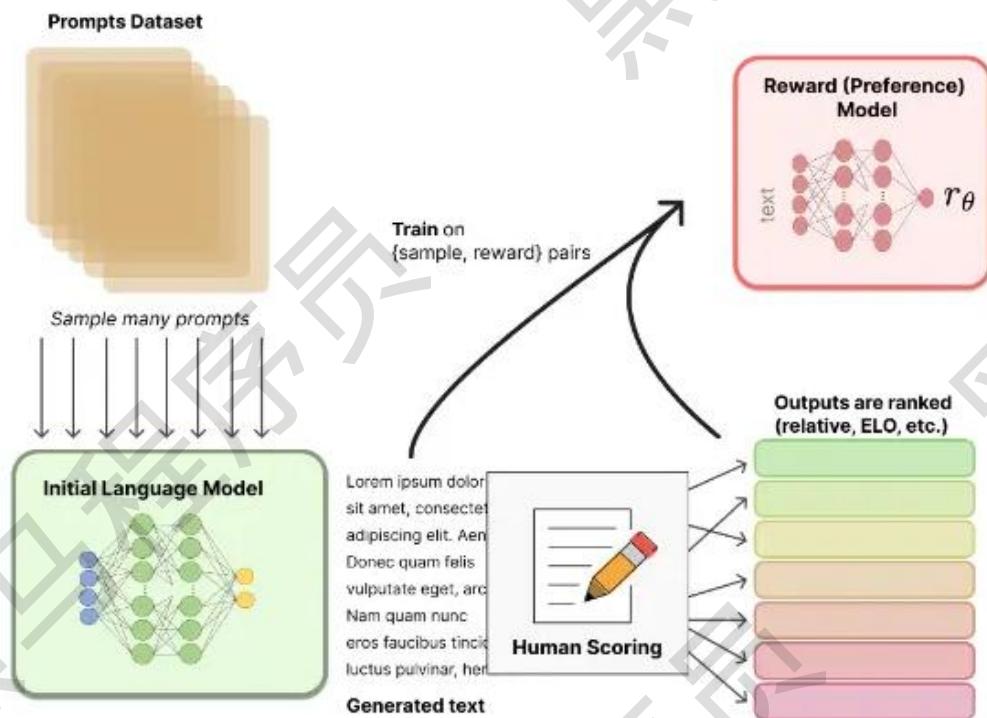
## 2. 简单介绍一下 RLHF?

RLHF就是基于人类反馈(Human Feedback)对语言模型进行强化学习(Reinforcement Learning)，一般分为以下三个步骤：

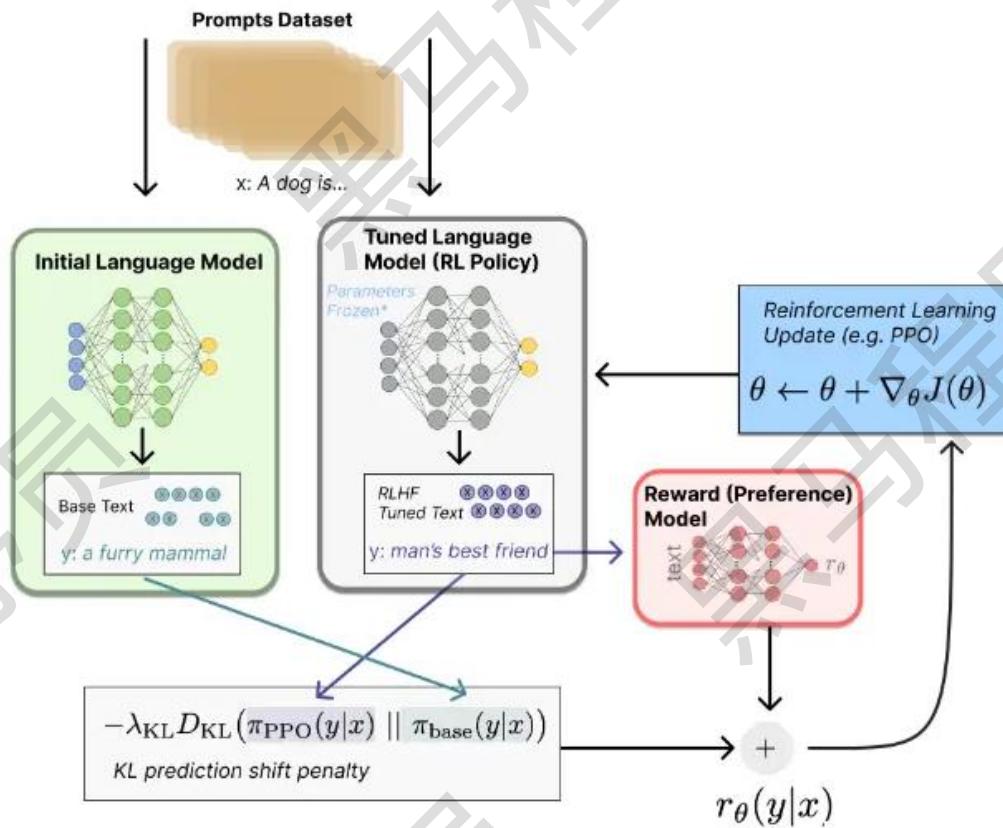
- 1) 预训练语言模型(收集样本数据，有监督微调)：在人类标注的数据上微调出来的模型叫做有监督的微调(supervised fine-tuning)，这是训练出来的第一个模型



- 2) 训练奖励模型(收集排序数据，训练奖励模型)：
- 给定一个问题，让上一步训练好的预训练模型 SFT 生成答案
  - GPT 每一次预测一个词的概率，可以根据这个概率采样出很多答案，通常来说可以用 beam search
  - 这里生成了四个答案，然后把这四个答案的好坏进行人工标注，进行排序标注
  - 有了这些排序之后，再训练一个奖励模型(Reward Model, RM)，这个模型是说给定 prompt 得到输出，然后对这个输出生成一个分数，可以认为这个分数是一个奖励或者是打分，使得对答案的分数能够满足人工排序的关系(大小关系保持一致)，一旦这个模型生成好之后，就能够对生成的答案进行打分



- 3) 用强化学习微调（使用RM模型优化SFT模型）：继续微调之前训练好的 SFT模型，使得它生成的答案能够尽量得到一个比较高的分数，即每一次将它生成的答案放进 RM 中打分，然后优化 SFT 的参数使得它生成的答案在 RM 中获得更高的分数。





备注：两次对模型的微调：GPT3模型 → SFT模型 → RL模型，其实这里始终都是同一个模型，只是不同过程中名称不同。

- **需要SFT模型的原因：** GPT3模型不一定能够保证根据人的指示、有帮助的、安全的生成答案需要人工标注数据进行微调。
- **需要RM模型的原因：** 标注排序的判别式标注成本远远低于生成答案的生成式标注。
- **需要RL模型的原因：** 在对SFT模型进行微调时生成的答案分布也会发生变化，会导致RM模型的评分会有偏差，需要用到强化学习.

### 3. 奖励模型需要和基础模型一致吗？

奖励模型和基础模型在训练过程中可以是一致的，也可以是不同的。这取决于你的任务需求和优化目标。

如果你希望优化一个包含多个子任务的复杂任务，那么你可能需要为每个子任务定义一个奖励模型，然后将这些奖励模型整合到一个统一的奖励函数中。这样，你可以根据任务的具体情况调整每个子任务的权重，以实现更好的性能。

另一方面，如果你的任务是单任务的，那么你可能只需要一个基础模型和一个对应的奖励模型，这两个模型可以共享相同的参数。在这种情况下，你可以通过调整奖励模型的权重来控制任务的优化方向。

总之，**奖励模型和基础模型的一致性取决于你的任务需求和优化目标**。在实践中，你可能需要尝试不同的模型结构和奖励函数，以找到最适合你任务的解决方案。

### 4. RLHF 在实践过程中存在哪些不足？

RLHF (Reinforcement Learning from Human Feedback) 是一种通过人类反馈进行增强学习的方法，尽管具有一定的优势，但在实践过程中仍然存在以下几个不足之处：

- 1) **人类反馈的代价高昂：** 获取高质量的人类反馈通常需要大量的人力和时间成本。人类专家需要花费时间来评估模型的行为并提供准确的反馈，这可能限制了RLHF方法的可扩展性和应用范围。
- 2) **人类反馈的主观性：** 人类反馈往往是主观的，不同的专家可能会有不同的意见和判断。这可能导致模型在不同专家之间的反馈上存在差异，从而影响模型的训练和性能。
- 3) **反馈延迟和稀疏性：** 获取人类反馈可能存在延迟和稀疏性的问题。人类专家不可能实时监控和评估模型的每一个动作，因此模型可能需要等待一段时间才能收到反馈，这可能会导致训练的



效率和效果下降。

- 4) **错误反馈的影响：**人类反馈可能存在错误或误导性的情况，这可能会对模型的训练产生负面影响。如果模型在错误的反馈指导下进行训练，可能会导致模型产生错误的行为策略。
- 5) **缺乏探索与利用的平衡：**在RLHF中，人类反馈通常用于指导模型的行为，但可能会导致模型过于依赖人类反馈而缺乏探索的能力。这可能限制了模型发现新策略和优化性能的能力。

针对这些不足，研究人员正在探索改进RLHF方法，如设计更高效的人类反馈收集机制、开发更准确的反馈评估方法、结合自适应探索策略等，以提高RLHF方法的实用性和性能。

## 5. 什么是 LLM Agent?

LLM Agent 是一种人工智能系统，它利用大型语言模型 (LLM) 作为其核心计算引擎，展示文本生成之外的功能，包括进行对话、完成任务、推理，并可以展示一定程度的自主行为。

LLM Agent 根据设计阶段授予的功能，Agent 从纯粹的被动到高度主动的自主行为。同时利用大模型的推理能力，让 Agent 可以在人工监督下管理相对独立的工作流程：分析目标，项目规划，执行，回顾过去的工作，迭代细化。

## 6. LLM Agent 有什么关键能力？

- 1) Agent利用LLM的语言能力理解指令、上下文和目标。可以根据人类提示自主和半自主操作。
- 2) 可以利用工具套件（计算器、API、搜索引擎）来收集信息并采取行动来完成分配的任务。它们不仅仅局限于语言处理。
- 3) 可以做逻辑推理类型的任务。例如，chain-of-thought , tree-of-thought。
- 4) 可以量身定制文本，例如邮件，报告，市场材料。
- 5) 可以自动或半自动的响应用户的需求。
- 6) Agent可以和不同类型的AI系统对接，例如LLM+image generators。

## 7. 怎样构建基于 LLM 的 Agents?

Agent = LLM + Prompt Recipe + Tools + Interface + Knowledge + Memory

- 1) Prompt Recipe：特定的内容要求、目标受众、所需的语气、输出长度、创造力水平等。



- 2) Tools: 工具集成允许通过API和外部服务完成任务。Agents 能够理解自然语言、推理提示、积累记忆并采取明智的行动。但是，Agents 的表现和一致性取决于他们收到的提示的质量。
- 3) Knowledge: 知识适用于所有用户的一般专业知识。知识扩展了LLM的内容。一般分为专业知识、常识知识和程序知识。
- 4) Memory: 单个用户或单个任务的上下文和记录细节。分为短期记忆和长期记忆。记忆服务与特定用户，在时间维度的体验。使特定用户的上下文对话个性化同时保持多步骤任务的一致性。记忆侧重暂时的用户和任务细节。

## 大语言模型评估

### 1. 大模型怎么评测？

自动评测和人工评测。这两种方法在评测语言模型和机器翻译等任务时起着重要的作用。

自动评测方法基于计算机算法和自动生成的指标，能够快速且高效地评测模型的性能。

而人工评测则侧重于人类专家的主观判断和质量评测，能够提供更深入、细致的分析和意见。了解和掌握这两种评测方法对准确评测和改进语言模型的能力十分重要。

### 2. 大模型的 honest 原则是如何实现的？模型如何判断回答的知识是训练过的已知的知识，怎么训练这种能力？

大模型需要遵循的helpful, honest, harmless的原则。

可以有意构造如下的训练样本，以提升模型遵守 honest 原则，可以算 trick 了：微调时构造知识问答类训练集，给出不知道的不回答，加强 honest 原则；阅读理解题，读过的要回答，没读过的不回答，不要胡说八道。



### 3. 如何衡量大模型水平？

在评测 LLMs 的性能时，选择合适的任务和领域对于展示大型语言模型的表现、优势和劣势至关重要。为了更清晰地展示 LLMs 的能力水平，文章将现有的任务划分为以下7个不同的类别：

- 1) 自然语言处理：包括自然语言理解、推理、自然语言生成和多语言任务
- 2) 鲁棒性、伦理、偏见和真实性
- 3) 医学应用：包括医学问答、医学考试、医学教育和医学助手
- 4) 社会科学
- 5) 自然科学与工程：包括数学、通用科学和工程
- 6) 代理应用：将 LLMs 作为代理使用
- 7) 其他应用

### 4. 大模型评估方法有哪些？

- 1) 首先是“**直接评估指标**”这一类别。这些是在人工智能领域长期以来广泛使用的传统指标。像准确率（accuracy）和F1得分（F1 score）等指标属于这个类别。通常情况下，这种方法涉及从模型中获取单一的输出，并将其与参考值进行比较，可以通过约束条件或提取所需信息的方式来实现评估
- 2) 接下来是第二类方法，称为“**间接或分解的启发式方法**（indirect or decomposed heuristics）”。在这种方法中，我们利用较小的模型（smaller models）来评估主模型（the main model）生成的答案，这些较小的模型可以是微调过的模型或原始的分解模型（raw decompositions）。
- 3) 第三类评估方法被称为“**基于模型的评估**”。在这种方法中，模型本身提供最终的评估分数或评估结果。然而，这也引入了额外的可变因素。即使模型可以获取到ground truth信息，评估指标本身也可能在评分过程中产生随机因素或不确定因素

### 5. 什么是大模型幻觉？

在语言模型的背景下，幻觉指的是一本正经的胡说八道：看似流畅自然的表述，实则不符合事实或者是错误的。

幻觉现象的存在严重影响LLM应用的可靠性，本文将探讨大型语言模型(LLMs)的幻觉问题，以及解决幻觉现象的一些常见方法。



## 6. 为什么需要解决LLM的幻觉问题？

LLMs的幻觉可能会产生如传播错误信息或侵犯隐私等严重后果。比如在医疗应用中，对患者生成的报告如果存在幻觉可能导致错误诊断甚至影响生命安全。

幻觉影响了模型的可靠性和可信度，因此需要解决LLM的幻觉问题。

## 7. 幻觉一定是有害的吗？

幻觉不一定是有害的，特别是在一些需要创造力或灵感的场合，比如写电影剧情，幻觉的存在可能带来一些奇思妙想，使得生成的文本充满想象力。

因此，对幻觉的容忍度取决于具体的应用场景。

## 8. 幻觉有哪些不同类型？

幻觉主要可以分为两类：即**内在幻觉**和**外在幻觉**。

- **内在幻觉**：生成的内容与源内容相矛盾。
- **外部幻觉**：生成的内容不能从源内容中得到验证，既不受源内容支持也不受其反驳。

## 9. 为什么LLM会产生幻觉？

有一些研究也在致力于分析幻觉出现的不同原因，已知的一些原因包括：

- 1) **源与目标的差异**：当我们在存在源与目标差异的数据上训练模型时，模型产生的文本可能与原始源内容产生偏差。这种差异，有时可能是在数据收集过程中不经意间产生的，有时则是故意为之。
- 2) **无意识的源-目标差异**：这种差异的产生有多种原因。例如，数据可能是基于某种经验法则编制的，使得目标信息并不总是完全依赖源信息。举例来说，如果从两家不同的新闻网站获得相同事件的报道作为源与目标，目标报道中可能包含源报道没有的信息，从而导致二者不同。
- 3) **有意识的源-目标差异**：某些任务在本质上并不追求源与目标的严格一致，尤其是在需要多样



性输出的情境下。

- 4) **训练数据的重复性**: 训练过程中使用的数据, 如果存在大量重复, 可能导致模型在生成时过于偏好某些高频短语, 这也可能引发“幻觉”。
- 5) **数据噪声的影响**: 使用充斥噪声的数据进行训练, 往往是导致“幻觉”出现的关键因素之一。
- 6) **解码过程中的随机性**: 某些旨在增加输出多样性的解码策略, 如top-k采样、top-p方法以及温度调节, 有时会增加“幻觉”的产生。这往往是因为模型在选择输出词汇时引入了随机性, 而没有始终选择最可能的词汇。
- 7) **模型的参数知识偏向**: 有研究表明, 模型在处理信息时, 可能更依赖其在预训练阶段所积累的知识, 而忽略了实时提供的上下文信息, 从而偏离了正确的输出路径。
- 8) **训练与实际应用中的解码差异**: 在常见的训练方法中, 我们鼓励模型基于真实数据预测下一个词汇。但在实际应用中, 模型则是根据自己先前生成的内容进行预测。这种方法上的差异, 尤其在处理长文本时, 可能会导致模型的输出出现“幻觉”。

最后, 如GPT之类的生成模型, 其实只是学会了文本中词汇间的统计规律, 所以它们生成内容的准确性仍然是有限的。

## 10. 如何度量幻觉?

最有效可靠的方式当然是靠人来评估, 但是人工评估的成本太高了。因此有了一些自动化评估的指标:

- **命名实体误差**: 命名实体 (NEs) 是“事实”描述的关键组成部分, 我们可以利用NE匹配来计算生成文本与参考资料之间的一致性。直观上, 如果一个模型生成了不在原始知识源中的NE, 那么它可以被视为产生了幻觉 (或者说, 有事实上的错误)。
- **蕴含率**: 该指标定义为被参考文本所蕴含的句子数量与生成输出中的总句子数量的比例。为了实现这一点, 可以采用成熟的蕴含/NLI模型。
- **基于模型的评估**: 应对复杂的句法和语义变化。
- **利用问答系统**: 此方法的思路是, 如果生成的文本在事实上与参考材料一致, 那么对同一个问题, 其答案应该与参考材料相似。具体而言, 对于给定的生成文本, 问题生成模型会创建一组问题-答案对。接下来, 问答模型将使用原始的参考文本来回答这些问题, 并计算所得答案的相似性。
- **利用信息提取系统**: 此方法使用信息提取模型将知识简化为关系元组, 例如<主体, 关系, 对象>。这些模型从生成的文本中提取此类元组, 并与从原始材料中提取的元组进行比较。



# 大语言模型应用

## 1. 什么是 LangChain?

LangChain 是一个基于语言模型的框架，用于构建聊天机器人、生成式问答（GQA）、摘要等功能。它的核心思想是将不同的组件“链”在一起，以创建更高级的语言模型应用。

LangChain 框架核心目标是为了连接多种大语言模型（如 OpenAI、LLaMA 等）和外部资源（如 Google、Wikipedia、Notion 以及 Wolfram 等），提供抽象和工具以在文本输入和输出之间进行接口处理。大语言模型和组件通过“链（Chain）”连接，使得开发人员可以快速开发原型系统和 应用程序。

LangChain 的主要价值在于以下几个方面：

- 1) **组件化**: LangChain 框架提供了用于处理语言模型的抽象组件，以及每个抽象组件的一系列实现。这些组件具有模块化设计，易于使用，无论是否使用 LangChain 框架的其他部分，都可以方便地使用这些组件。
- 2) **现成的链式组装**: LangChain 框架提供了一些现成的链式组装，用于完成特定的高级任务。这些现成的链式组装使得入门变得更加容易。对于更复杂的应用程序，LangChain 框架也支持自定义现有链式组装或构建新的链式组装。
- 3) **简化开发难度**: 通过提供组件化和现成的链式组装，LangChain 框架可以大大简化大语言模型应用的开发难度。开发人员可以更专注于业务逻辑，而无需花费大量时间和精力处理底层 技术细节。

## 2. LangChain 包含哪些核心模块

LangChain 提供了以下 6 种标准化、可扩展的接口并且可以外部集成的核心模块：

- 模型输入/输出 (Model I/O) : 与语言模型交互的接口；
- 数据连接 (Data connection) : 与特定应用程序的数据进行交互的接口；
- 链 (Chains) : 用于复杂的应用的调用序列；
- 智能体 (Agents) : 语言模型作为推理器决定要执行的动作序列；
- 记忆 (Memory) : 用于链的多次运行之间持久化应用程序状态；
- 回调 (Callbacks) : 记录和流式传输任何链式组装的中间步骤。



### 3. LangChain 如何调用 LLMs 生成回复？

要调用LLMs生成回复，可以使用LangChain框架提供的LLMChain类。LLMChain类是LangChain的一个组件，用于与语言模型进行交互并生成回复。以下是一个示例代码片段，展示了如何使用LLMChain类调用LLMs生成回复：

```
from langchain.llms import OpenAI
from langchain.chains import LLMChain

llm = OpenAI(temperature=0.9) # 创建 LLM 实例
prompt = "用户的问题" # 设置用户的问题

# 创建 LLMChain 实例
chain = LLMChain(llm=llm, prompt=prompt)

# 调用 LLMs 生成回复
response = chain.generate()

print(response) # 打印生成的回复
```

在上面的代码中，首先创建了一个LLM实例，然后设置了用户的问题作为LLMChain的prompt。接下来，调用LLMChain的generate方法来生成回复。最后，打印生成的回复。

请注意，可以根据需要自定义LLM的参数，例如温度（temperature）、最大令牌数（max\_tokens）等。LangChain文档中有关于LLMChain类和LLM参数的更多详细信息。

### 4. LangChain 如何修改 提示模板？

要修改LangChain的提示模板，可以使用LangChain框架提供的ChatPromptTemplate类。

ChatPromptTemplate类允许您创建自定义的聊天消息提示，并根据需要进行修改。以下是一个示例代码片段，展示了如何使用ChatPromptTemplate类修改提示模板：

```
from langchain.llms import OpenAI
from langchain.chains import LLMChain

llm = OpenAI(temperature=0.9) # 创建 LLM 实例
```



```
prompt = "用户的问题" # 设置用户的问题

# 创建 LLMChain 实例
chain = LLMChain(llm=llm, prompt=prompt)

# 调用 LLMs 生成回复
response = chain.generate()

print(response) # 打印生成的回复
```

在上面的代码中，首先创建了一个空的ChatPromptTemplate实例。然后，使用add\_message方法添加了聊天消息提示。接下来，我们使用set\_message\_content方法修改了第一个和最后一个聊天消息的内容。最后，我们使用format\_messages方法格式化聊天消息，并打印出来。

请注意，可以根据需要添加、删除和修改聊天消息提示。ChatPromptTemplate类提供了多种方法来操作提示模板。更多详细信息和示例代码可以在LangChain文档中找到。