

# 大模型基础

Foundations of Large Language Models

毛玉仁 高云君等 著



我的语言的界限意味着我的世界的界限。  
——维特根斯坦《逻辑哲学论》

\* 本书所指大模型为大语言模型。

\* 本书作者分工情况如下：第一章作者为：毛玉仁、高云君；第二章作者为：李佳晖、毛玉仁、宓禹；第三章作者为：张超、毛玉仁、胡中豪；第四章作者为：葛宇航、毛玉仁；第五章作者为：宓禹、樊怡江、毛玉仁；第六章作者为：董雪梅、徐文溢、毛玉仁。高云君为本书编撰总指导。

# 目录

<b>第 1 章 语言模型基础</b>	<b>1</b>
1.1 基于统计方法的语言模型 n-grams	2
1.1.1 n-grams 语言模型	2
1.1.2 n-grams 的统计学原理	4
1.2 基于 RNN 的语言模型	6
1.2.1 循环神经网络 RNN	7
1.2.2 基于 RNN 的语言模型	10
1.3 基于 Transformer 的语言模型	12
1.3.1 Transformer	12
1.3.2 基于 Transformer 的语言模型	16
1.4 语言模型的采样方法	18
1.4.1 概率最大化方法	18
1.4.2 随机采样方法	20
1.5 语言模型的评测	23
1.5.1 内在评测	23
1.5.2 外在评测	24
<b>第 2 章 大语言模型</b>	<b>31</b>
2.1 大数据 + 大模型 → 新智能	32
2.1.1 大数据 + 大模型 → 能力增强	33
2.1.2 大数据 + 大模型 → 能力扩展	36
2.2 大语言模型架构	38
2.2.1 经典模型架构介绍	38
2.2.2 模型架构对比	42
2.2.3 模型架构的历史演变	45
2.3 基于 Encoder-only 架构的大语言模型	47
2.3.1 双向编码模型以及 Encoder-only 架构	47
2.3.2 BERT 语言模型	48
2.3.3 BERT 衍生语言模型	51

2.3.4 Encoder-only 模型架构对比 . . . . .	57
2.4 基于 Encoder-Decoder 架构的大语言模型 . . . . .	58
2.4.1 Encoder-Decoder 架构 . . . . .	58
2.4.2 T5 语言模型 . . . . .	61
2.4.3 BART 模型 . . . . .	64
2.4.4 Encoder-Decoder 架构模型对比 . . . . .	66
2.5 基于 Decoder-only 架构的大语言模型 . . . . .	67
2.5.1 GPT 系列模型 . . . . .	67
2.5.2 LLAMA 系列模型 . . . . .	79
2.5.3 Decoder-only 架构模型对比 . . . . .	86
2.6 非 Transformer 架构 . . . . .	86
2.6.1 RWKV . . . . .	87
2.6.2 Mamba . . . . .	89
<b>第 3 章 Prompt 工程</b>	<b>97</b>
3.1 Prompt 工程简介 . . . . .	98
3.1.1 什么是 Prompt . . . . .	98
3.1.2 什么是 Prompt 工程 . . . . .	99
3.1.3 模型如何处理 Prompt . . . . .	102
3.1.4 Prompt 工程的意义 . . . . .	105
3.2 上下文学习 . . . . .	108
3.2.1 上下文学习的定义 . . . . .	108
3.2.2 演示示例选择 . . . . .	111
3.2.3 性能影响因素 . . . . .	114
3.3 思维链 . . . . .	118
3.3.1 思维链提示的定义 . . . . .	118
3.3.2 按部就班 . . . . .	120
3.3.3 三思后行 . . . . .	123
3.3.4 集思广益 . . . . .	124
3.4 Prompt 技巧 . . . . .	127
3.4.1 规范 Prompt 编写 . . . . .	127
3.4.2 合理归纳提问 . . . . .	134
3.4.3 适时使用 CoT . . . . .	139
3.4.4 善用心理暗示 . . . . .	143
3.5 相关应用 . . . . .	147
3.5.1 Agent . . . . .	147

---

3.5.2	数据合成 . . . . .	151
3.5.3	GPTS . . . . .	153
3.5.4	Text-to-SQL . . . . .	156
<b>第 4 章</b>	<b>参数高效微调</b>	<b>163</b>
4.1	参数高效微调简介 . . . . .	164
4.1.1	下游任务适配 . . . . .	164
4.1.2	参数高效微调 . . . . .	166
4.1.3	参数高效微调的优势 . . . . .	168
4.2	参数附加方法 . . . . .	169
4.2.1	加在输入 . . . . .	169
4.2.2	加在模型 . . . . .	171
4.2.3	加在输出 . . . . .	176
4.3	参数选择方法 . . . . .	177
4.3.1	基于规则的方法 . . . . .	178
4.3.2	基于学习的方法 . . . . .	178
4.4	低秩适配方法 . . . . .	181
4.4.1	LoRA . . . . .	181
4.4.2	LoRA 相关变体 . . . . .	183
4.4.3	基于 LoRA 插件的任务泛化 . . . . .	185
4.5	实践与应用 . . . . .	187
4.5.1	PEFT 实践 . . . . .	187
4.5.2	PEFT 应用 . . . . .	190
<b>第 5 章</b>	<b>模型编辑</b>	<b>197</b>
5.1	模型编辑简介 . . . . .	198
5.1.1	模型编辑思想 . . . . .	199
5.1.2	模型编辑定义 . . . . .	199
5.1.3	模型编辑性质 . . . . .	201
5.1.4	常用数据集 . . . . .	204
5.2	模型编辑经典方法 . . . . .	206
5.2.1	外部拓展法 . . . . .	207
5.2.2	内部修改法 . . . . .	210
5.2.3	方法比较 . . . . .	215
5.3	附加参数法: T-Patcher . . . . .	217
5.3.1	补丁的位置 . . . . .	218
5.3.2	补丁的形式 . . . . .	219

## 目录

---

5.3.3 补丁的实现 . . . . .	220
5.4 定位编辑法: ROME . . . . .	222
5.4.1 知识定位 . . . . .	223
5.4.2 知识存储假设 . . . . .	225
5.4.3 阻断实验 . . . . .	226
5.4.4 ROME 编辑方法 . . . . .	227
5.5 模型编辑应用 . . . . .	231
5.5.1 精准模型更新 . . . . .	232
5.5.2 保护被遗忘权 . . . . .	233
5.5.3 提升模型安全 . . . . .	235
<b>第 6 章 检索增强生成</b>	<b>241</b>
6.1 检索增强生成简介 . . . . .	242
6.1.1 检索增强生成的背景 . . . . .	242
6.1.2 检索增强生成的定义 . . . . .	246
6.2 检索增强生成架构 . . . . .	249
6.2.1 白盒增强架构 . . . . .	250
6.2.2 黑盒增强架构 . . . . .	253
6.2.3 对比与分析 . . . . .	255
6.3 知识检索 . . . . .	256
6.3.1 知识库构建 . . . . .	257
6.3.2 查询增强 . . . . .	262
6.3.3 文本检索 . . . . .	263
6.3.4 检索结果重排 . . . . .	268
6.4 生成增强 . . . . .	270
6.4.1 增强必要性判定 . . . . .	271
6.4.2 增强实施位置 . . . . .	276
6.4.3 增强性能改善 . . . . .	278
6.4.4 增强效率提升 . . . . .	282
6.5 应用与前景 . . . . .	285
6.5.1 检索增强生成典型应用 . . . . .	286
6.5.2 检索增强生成工具框架 . . . . .	289
6.5.3 挑战与未来方向 . . . . .	290

# 1 语言模型基础

语言是一套复杂的符号系统。语言符号通常在音韵 (Phonology)、词法 (Morphology)、句法 (Syntax) 的约束下构成，并承载不同的语义 (Semantics)。语言符号具有不确定性。同样的语义可以由不同的音韵、词法、句法构成的符号来表达；同样的音韵、词法、句法构成的符号也可以在不同的语境下表达不同的语义。因此，**语言是概率的**。并且，**语言的概率性与认知的概率性也存在着密不可分的关系** [15]。语言模型 (Language Models, LMs) 旨在**准确预测语言符号的概率**。从语言学的角度，语言模型可以赋能计算机掌握语法、理解语义，以完成自然语言处理任务。从认知科学的角度，准确预测语言符号的概率可以赋能计算机描摹认知、演化智能。从 ELIZA [20] 到 GPT-4 [16]，语言模型经历了从规则模型到统计模型，再到神经网络模型的发展历程，逐步从呆板的机械式问答程序成长为具有强大泛化能力的多任务智能模型。本章将按照语言模型发展的顺序依次讲解基于统计方法的 n-grams 语言模型、基于循环神经网络 (Recurrent Neural Network, RNN) 的语言模型，基于 Transformer 的语言模型。此外，本章还将介绍如何将语言模型输出概率值解码为目标文本，以及如何对语言模型的性能进行评估。

## 1.1 基于统计方法的语言模型 n-grams

语言模型通过对语料库（Corpus）中的语料进行统计或学习来获得预测语言符号概率的能力。通常，基于统计的语言模型通过直接统计语言符号在语料库中出现的频率来预测语言符号的概率。其中，n-grams 是最具代表性的统计语言模型。n-grams 基于马尔可夫假设和离散变量的极大似然估计给出语言符号的概率。本节首先给出 n-grams 的计算方法，然后讨论 n-grams 如何在马尔可夫假设的基础上应用离散变量极大似然估计给出语言符号出现的概率。

### 1.1.1 n-grams 语言模型

设包含  $N$  个元素的语言符号可以表示为  $w_{1:N} = \{w_1, w_2, w_3, \dots, w_N\}$ 。 $w_{1:N}$  可以代表文本，也可以代表音频序列等载有语义信息的序列。为了便于理解，本章令语言符号  $w_{1:N}$  代表文本，其元素  $w \in w_{1:N}$  代表词。在真实语言模型中， $w$  可以是 token 等其他形式。关于 token 的介绍将在第三章中给出。

设文本  $w_{1:N}$  出现的概率为  $P(w_{1:N})$ 。n-grams 通过下列公式计算  $P(w_{1:N})$ 。

$$P(w_{1:N}) = P_{n-grams}(w_{1:N}) = \prod_{i=n+1}^N P(w_i | w_{i-n : i-1}), \quad (1.1)$$

其中，

$$P(w_i | w_{i-n : i-1}) = \frac{C(w_{i-n : i})}{C(w_{i-n : i-1})}. \quad (1.2)$$

上式中， $C(w_{i-n : i})$  为词序列  $\{w_{i-n}, \dots, w_{i-1}, w_i\}$  在语料库中出现的次数， $C(w_{i-n : i-1})$  为词序列  $\{w_{i-n}, \dots, w_{i-1}\}$  在语料库中出现的次数。其中， $n$  为变量，当  $n = 1$  时，称之为 unigram，其不考虑文本的上下文关系。当  $n = 2$  时，称之为 bigrams，其对前一个词进行考虑。当  $n = 3$  时，称之为 trigrams，其对前两个词进行考虑。当  $n = 4$  时，称之为 4-grams，其对前三个词进行考虑，以此类推。

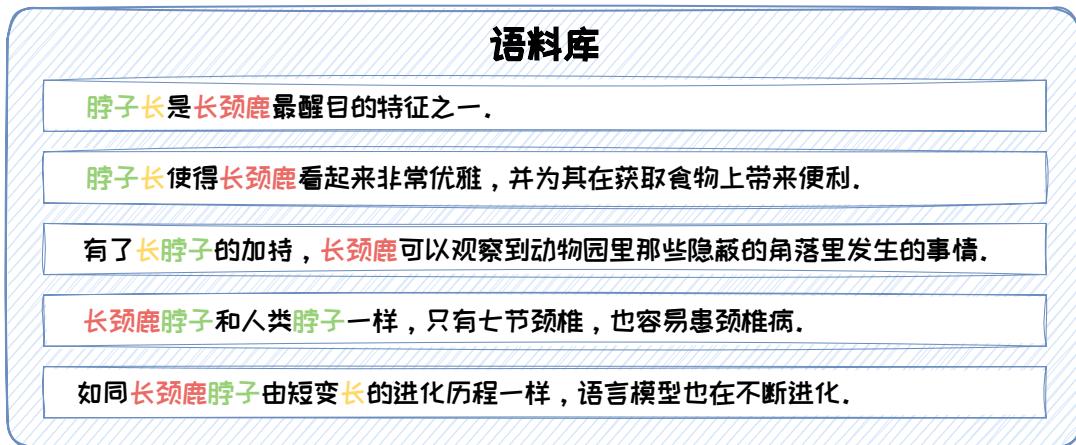


图 1.1: n-grams 示例语料库。

下面通过一个 bigrams 语言模型的例子来展示 n-grams 语言模型对文本出现概率进行计算的具体方式。假设语料库中包含 5 个句子，如图1.1所示。基于此语料库，应用 bigrams 对文本“长颈鹿脖子长”出现的概率进行计算。

$$\begin{aligned}
 P_{bigrams}(\text{长颈鹿脖子长}) &= P(\text{脖子}|\text{长颈鹿}) \cdot P(\text{长}|\text{脖子}) \\
 &= \frac{C(\text{长颈鹿脖子})}{C(\text{长颈鹿})} \cdot \frac{C(\text{脖子长})}{C(\text{脖子})}。
 \end{aligned} \tag{1.3}$$

在此语料库中， $C(\text{长颈鹿}) = 5$ ,  $C(\text{脖子}) = 6$ ,  $C(\text{长颈鹿脖子}) = 2$ ,  $C(\text{脖子长}) = 2$ ，故有：

$$P_{bigrams}(\text{长颈鹿脖子长}) = \frac{2}{5} \cdot \frac{2}{6} = \frac{2}{15}。 \tag{1.4}$$

在此例中，我们可以发现虽然“长颈鹿脖子长”并没有直接出现在语料库中，但是 bigrams 语言模型仍可以预测出“长颈鹿脖子长”是有  $\frac{2}{15}$  概率出现的文本。由此可见，n-grams 具备对未知文本的泛化能力。这也是其相较于传统基于规则的方法的优势。但是，这种泛化能力会随着  $n$  的增大而逐渐减弱。应用 trigrams 对文本“长颈鹿脖子长”出现的概率进行计算，将得到以下“零概率”的情况。

$$P_{trigrams}(\text{长颈鹿脖子长}) = P(\text{长}|\text{长颈鹿脖子}) = \frac{C(\text{长颈鹿脖子长})}{C(\text{长颈鹿脖子})} = 0。 \tag{1.5}$$

因此，在 n-grams 语言模型中， $n$  代表了与拟合语料库的能力与对未知文本的泛化

能力之间的权衡。上述的“零概率”现象可以通过平滑 (Smoothing) 技术进行改善，具体技术可参见文献 [11]。

本节讲解了 n-gram 语言模型如何计算语言符号出现的概率，但没有分析 n-grams 的原理。下一节将对 n-grams 背后的统计学原理进行阐述。

### 1.1.2 n-grams 的统计学原理

n-grams 语言模型是在  $n$  阶马尔可夫假设下，对语料库中出现的长度为  $n$  的词序列出现概率的极大似然估计。本节首先给出  $k$  阶马尔可夫假设和离散型随机变量的极大似然估计的定义，然后分析 n-grams 如何在马尔可夫假设的基础上应用离散变量极大似然估计给出语言符号出现的概率。 $k$  阶马尔可夫假设在 **定义 1.1** 中进行定义。离散型随机变量的极大似然估计在 **定义 1.2** 中进行定义。

#### 定义 1.1 ( $k$ 阶马尔可夫假设)

对序列  $\{w_1, w_2, w_3, \dots, w_N\}$ ，当前状态  $w_N$  出现的概率只与前  $k$  个状态  $\{w_{N-k-2}, w_{N-k-3}, \dots, w_{N-1}\}$  有关，即：

$$P(w_N|w_1, w_2, \dots, w_{N-1}) \approx P(w_N|w_{N-k-1}, w_{N-k-2}, \dots, w_{N-1})。 \quad (1.6)$$

#### 定义 1.2 (离散型随机变量的极大似然估计)

给定离散型随机变量  $X$  的分布律为  $P\{X = x\} = p(x; \theta)$ ，设  $X_1, \dots, X_N$  为来自  $X$  的样本， $x_1, \dots, x_N$  为对应的观察值， $\theta$  为待估计参数。在参数  $\theta$  下，分布函数随机取到  $x_1, \dots, x_N$  的概率为：

$$p(x|\theta) = \prod_{i=1}^N p(x_i; \theta)。 \quad (1.7)$$

构造似然函数为：

$$L(\theta|x) = p(x|\theta) = \prod_{i=1}^N p(x_i; \theta)。 \quad (1.8)$$

离散型随机变量的极大似然估计旨在找到  $\theta$  使得  $L(\theta|x)$  取最大值。

在上述两个定义的基础上，对 n-grams 的统计原理进行讨论。在 n-grams 中，我们计算当前词的概率时仅以前  $n - 1$  个词为条件。其是  $k$  阶马尔可夫假设在词序列中的应用。将  $k$  阶马尔可夫假设应用在词序列中，令  $k = n - 1$ ，则当前词的概率只与前  $n - 1$  个词有关，这与 n-grams 中只关心前  $n - 1$  个词是相同的。即 n-grams 对当前词的计算是基于  $n - 1$  阶马尔可夫假设的。故有，

$$P(w_{1:N}) = \prod_{i=n+1}^N P(w_i | w_{i-n:i-1}) \approx \prod_{i=n+1}^N P(w_i | w_{1:i-1})。 \quad (1.9)$$

马尔可夫假设帮助语言模型有效缓解“零概率”问题。

在 n-grams 中，我们以相对频率  $\frac{C(w_{i-n:i})}{C(w_{i-n:i-1})}$  代替  $P(w_i | w_{i-n:i-1})$ 。这是对语料库的极大似然估计。以下以 bigram 为例，介绍  $\frac{C(w_{i-n:i})}{C(w_{i-n:i-1})}$  与极大似然估计间的关系。假设语料库中共涵盖  $M$  个不同的单词，对  $\{w_i, w_j\}$  出现的概率为  $P(w_i, w_j)$ ，对应出现的频率为  $C(w_i, w_j)$ ，则其出现的似然函数为：

$$l(P(w_i, w_j)) = \prod_{i=1}^M \prod_{j=1}^M P(w_i, w_j)^{C(w_i, w_j)}。 \quad (1.10)$$

根据条件概率公式  $P(w_i, w_j) = P(w_j | w_i)P(w_i)$ ，有

$$l(P(w_i, w_j)) = \prod_{i=1}^M \prod_{j=1}^M P(w_j | w_i)^{C(w_i, w_j)} P(w_i)^{C(w_i, w_j)}。 \quad (1.11)$$

其对应的对数似然函数为：

$$l_{\log}(P(w_i, w_j)) = \sum_{i=1}^M \sum_{j=1}^M C(w_i, w_j) \log P(w_j | w_i) + \sum_{i=1}^M \sum_{j=1}^M C(w_i, w_j) \log P(w_i)。 \quad (1.12)$$

因为，对  $\sum_{j=1}^M P(w_j | w_i) = 1$ ，最大化对数似然函数可建模为如下的约束优化问题：

$$\begin{aligned} & \max \quad l_{\log}(P(w_i, w_j)) \\ & \text{s.t.} \quad \sum_{j=1}^M P(w_j | w_i) = 1 \text{ for } j \in [1, M]。 \end{aligned} \quad (1.13)$$

其拉格朗日对偶为：

$$L(\lambda, l_{\log}) = \sum_{i=1}^M \sum_{j=1}^M C(w_i, w_j) \log P(w_j | w_i) + \sum_{i=1}^M \lambda_i \left( \sum_{j=1}^M P(w_j | w_i) - 1 \right)。 \quad (1.14)$$

对其进行求导，可得：

$$\frac{\partial L(\lambda, l_{log})}{\partial P(w_j|w_i)} = \sum_{i=1}^M \frac{C(w_i, w_j)}{P(w_j|w_i)} + \sum_{i=1}^M \lambda_i. \quad (1.15)$$

当导数为 0 时，有：

$$P(w_j|w_i) = -\frac{C(w_i, w_j)}{\lambda_i}. \quad (1.16)$$

因  $\sum_{j=1}^M P(w_j|w_i) = 1$ ， $\lambda_i$  可取值为  $-\sum_{j=1}^M C(w_i, w_j)$ ，即

$$P(w_j|w_i) = \frac{C(w_i, w_j)}{\sum_{j=1}^M C(w_i, w_j)} = \frac{C(w_i, w_j)}{C(w_i)}. \quad (1.17)$$

上述分析表明 bigram 模型是对语料库中包含的词对的极大似然估计。其可扩展到  $n > 2$  的其他 n-grams 模型中。

n-grams 模型通过统计词序列在语料库中出现的频率来预测语言符号的概率。其对未知序列有一定的泛化性，但也容易陷入“零概率”的困境。随着神经网络的发展，基于各类神经网络的语言模型不断被提出，泛化能力越来越强。基于神经网络的语言模型不在通过显性的计算公式对语言符号的概率进行计算，而是利用语料库中的样本对神经网络模型进行训练。本章接下将分别介绍两类最具代表性的基于神经网络的语言模型：基于 RNN 的语言模型和基于 Transformer 的语言模型。

## 1.2 基于 RNN 的语言模型

循环神经网络 (Recurrent Neural Network, RNN) 是一类网络连接中包含环路的神经网络的总称。给定一个序列，RNN 的环路用于将历史状态叠加到当前状态上。沿着时间维度，历史状态被循环累积，并作为预测未来状态的依据。因此，RNN 可以基于历史规律，对未来进行预测。基于 RNN 的语言模型，以词序列作为输入，基于被循环编码的上文和当前词来预测下一个词出现的概率。本节将先对原始 RNN 的基本原理进行介绍，然后讲解如何利用 RNN 构建语言模型。



图 1.2: 前馈传播范式与循环传播范式的对比。

### 1.2.1 循环神经网络 RNN

按照推理过程中信号流转的方向，神经网络的正向传播范式可分为两大类：前馈传播范式和循环传播范式。在前馈传播范式中，计算逐层向前，“**不走回头路**”。而在循环传播范式中，某些层的计算结果会通过环路被反向引回前面的层中，形成“**螺旋式前进**”的范式。采用前馈传播范式的神经网络可以统称为前馈神经网络（Feed-forward Neural Network, FNN），而采用循环传播范式的神经网络被统称为循环神经网络（Recurrent Neural Network, RNN）。以包含输入层、隐藏层、输出层的神经网络为例。图1.2中展示了最简单的 FNN 和 RNN 的网络结构示意图，可见 FNN 网络结构中仅包含正向通路。而 RNN 的网络结构中除了正向通路，还有一条环路将某层的计算结果再次反向连接回前面的层中。

设输入序列为  $\{x_1, x_2, x_3, \dots, x_t\}$ ，隐状态为  $\{h_1, h_2, h_3, \dots, h_t\}$ ，对应输出为  $\{o_1, o_2, o_3, \dots, o_t\}$ ，输入层、隐藏层、输出层对应的网络参数分别为  $W_I, W_H, W_O$ 。 $g(\cdot)$  为激活函数， $f(\cdot)$  为输出函数。将输入序列一个元素接着一个元素的串行输入时，对于 FNN，当前的输出只与当前的输入有关，即

$$o_t = f(W_O g(W_I x_t)) \quad (1.18)$$

但是，独特的环路结构导致 RNN 与 FNN 的推理过程完全不同。在串行输入的过程中，前面的元素会被循环编码成隐状态，并叠加到当前的输入上面。其在  $t$

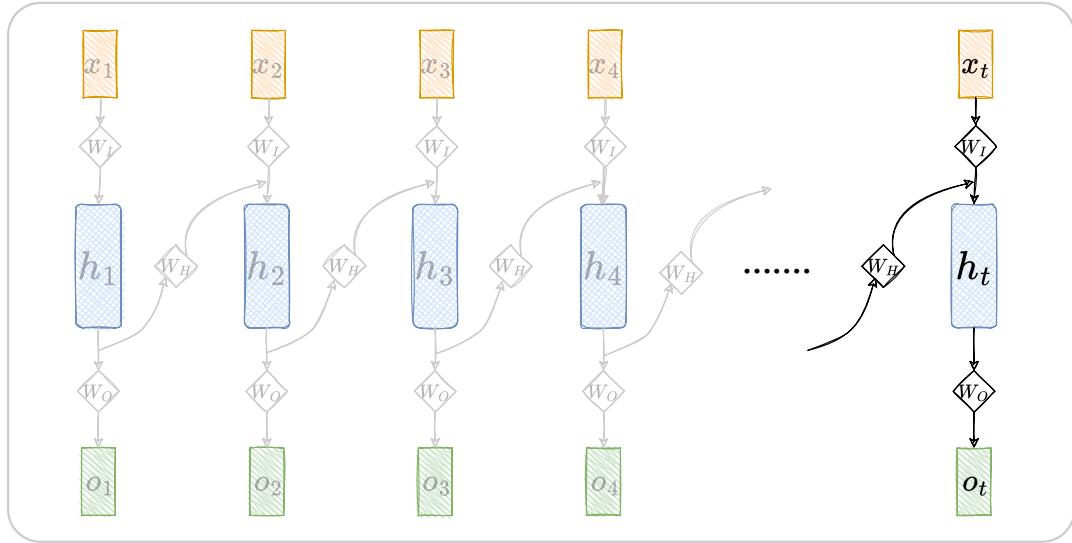


图 1.3: RNN 推理过程从时间维度拆解示意图。

时刻的输出如下：

$$h_t = g(W_H h_{t-1} + W_I x_t) = g(W_H g(W_H h_{t-2} + W_I x_{t-1}) + W_I x_t) = \dots \dots \quad (1.19)$$

$$o_t = f(W_O h_t)。$$

其中,  $t > 0, h_0 = 0$ 。将此过程按照时间维度展开, 可得到, 如图1.3所示。注意, 图中展示的  $t$  时刻前的神经元, 都是“虚影”并不真实存在, 如此展开只是为了解释 RNN 的工作方式。

可以发现, 在这样一个元素一个元素依次串行输入的设定下, RNN 可以**将历史状态以隐变量的形式循环叠加到当前状态上**, 对历史信息进行考虑, **呈现出螺旋式前进的模式**。但是, 缺乏环路的 FNN 仅对当前状态进行考虑, 无法兼顾历史状态。以词序列 {长颈鹿, 脖子, 长} 为例, 当我们需要根据“脖子”来预测下一个词是什么的时候。采用 FNN 时, 其将仅仅考虑用“脖子”来预测, 可能预测出的下一词包含“短”, “疼”等等。而对于 RNN 而言, 其同时考虑“长颈鹿”和“脖子”来进行预测, 其给出的高可能的下一词是“长”的概率极具升高, 历史信息“长颈鹿”的引入, 可以有效提升预测性能。

如果 FNN 想要做到对历史信息进行考虑，则需要将所有元素同时输入到模型中去，这将导致模型参数量的激增。虽然，RNN 的结构可以让其在参数量不扩张的情况下实现对历史信息的考虑，但是这样的环路结构给 RNN 的训练带来了挑战。在训练 RNN 时，涉及大量的矩阵联乘操作，容易引发梯度衰减或梯度爆炸问题。具体分析如下：

设 RNN 语言模型的训练损失为：

$$L = L(x, o, W_I, W_H, W_O) = \sum_{i=1}^t l(o_i, y_i)。 \quad (1.20)$$

其中， $l(\cdot)$  为损失函数， $y_i$  为标签。

损失  $L$  关于参数  $W_H$  的梯度为：

$$\frac{\partial L}{\partial W_H} = \sum_{i=1}^t \frac{\partial l_t}{\partial o_t} \cdot \frac{\partial o_t}{\partial h_t} \cdot \frac{\partial h_t}{\partial h_i} \cdot \frac{\partial h_i}{\partial W_H}。 \quad (1.21)$$

其中，

$$\frac{\partial h_t}{\partial h_i} = \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial h_{t-2}} \cdots \frac{\partial h_{i+1}}{\partial h_i} = \prod_{k=i}^t \frac{\partial h_k}{\partial h_{k-1}}。 \quad (1.22)$$

并且，

$$\frac{\partial h_k}{\partial h_{k-1}} = \frac{\partial g(z_k)}{\partial z_k} W_H。 \quad (1.23)$$

其中， $z_k = W_H h_{k-1} + W_I x_k$ 。综上，有

$$\frac{\partial L}{\partial W_H} = \sum_{i=1}^t \frac{\partial l_t}{\partial o_t} \cdot \frac{\partial o_t}{\partial h_t} \cdot \prod_{k=i}^t \frac{\partial g(z_k)}{\partial z_k} W_H \cdot \frac{\partial h_i}{\partial W_H}。 \quad (1.24)$$

从上式中可以看出，求解  $W_H$  的梯度时涉及大量的矩阵级联相乘。这会导致其数值被级联放大或缩小。文献 [17] 中指出，当  $W_H$  的最大特征值小于 1 时，会发生梯度消失；当  $W_H$  的最大特征值大于 1 时，会发生梯度爆炸。梯度消失和爆炸导致训练上述 RNN 非常困难。

为了解决梯度消失和爆炸问题，GRU [4] 和 LSTM [8] 引入门控结构，取得了良好效果，成为主流的 RNN 网络架构。

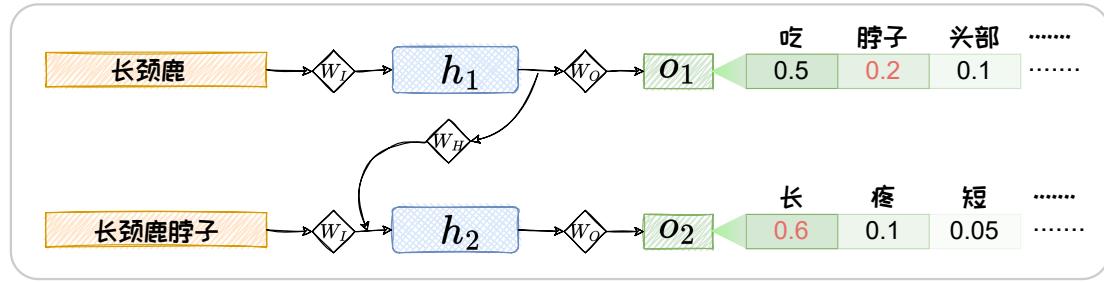


图 1.4: RNN 计算词序列概率示意图。

## 1.2.2 基于 RNN 的语言模型

对词序列  $\{w_1, w_2, w_3, \dots, w_N\}$ ，基于 RNN 的语言模型每次根据当前词  $w_i$  和循环输入的隐藏状态  $h_{i-1}$ ，来预测下一个词  $w_{i+1}$  出现的概率，即

$$P(w_{i+1}|w_{1:i}) = P(w_{i+1}|w_i, h_{i-1})。 \quad (1.25)$$

其中，当  $i = 1$  时， $P(w_{i+1}|w_i, h_{i-1}) = P(w_2|w_1)$ 。基于此， $\{w_1, w_2, w_3, \dots, w_N\}$  整体出现的概率为：

$$P(w_{1:N}) = \prod_{i=1}^N P(w_{i+1}|w_i, h_{i-1})。 \quad (1.26)$$

在基于 RNN 的语言模型中，输出为一个向量，其中每一维代表着词典中对应词的概率。设词典  $D$  中共有  $|D|$  个词，基于 RNN 的语言模型的输出可表示为  $o_i = \{o_i[\hat{w}_d]\}_{d=1}^{|D|}$ ，其中， $o_i[\hat{w}_d]$  表示词典中的词  $\hat{w}_d$  出现的概率。因此，对基于 RNN 的语言模型有

$$P(w_{1:N}) = \prod_{i=1}^{N-1} P(w_{i+1}|w_{1:i}) = \prod_{i=1}^N o_i[w_{i+1}]。 \quad (1.27)$$

以下举例对上述过程进行说明。假设词表  $D = \{\text{脖子}, \text{头部}, \text{吃}, \text{长}, \text{疼}, \text{吃}, \text{短}\}$ ，基于 RNN 的语言模型计算“长颈鹿脖子长”的概率的过程如图 1.4 所示。

$$P(\text{长颈鹿脖子长}) = P(\text{脖子}|\text{长颈鹿}) \cdot P(\text{长}|\text{脖子}, h_1) = 0.2 \times 0.6 = 0.12。 \quad (1.28)$$

基于以上预训练任务，对 RNN 语言模型进行训练时，可选用如下交叉熵函数

作为损失函数。

$$l_{CE}(o_i) = - \sum_{d=1}^{|D|} I(w_d = w_{i+1}) \log o_i[w_{i+1}] = -\log o_i[w_{i+1}]。 \quad (1.29)$$

其中,  $I(\cdot)$  为指示函数, 当  $w_d = w_{i+1}$  时等于 1, 当  $w_d \neq w_{i+1}$  时等于 0。

设训练集为  $S$ , RNN 语言模型的损失可以构造为:

$$L(W, U, V) = \frac{1}{N|S|} \sum_{s=1}^{|S|} \sum_{i=1}^N l_{CE}(o_i)。 \quad (1.30)$$

在此损失的基础上, 构建计算图, 执行反向传播方法, 便可对 RNN 语言模型进行训练。上述训练过程结束之后, 我们可以直接利用此模型对序列数据进行特征抽取。抽取后的特征可以用于解决下游任务。此外, 我们还可以对此语言模型的输出进行解码, 在“**自回归**”的范式下完成文本生成任务。在自回归中, 我们将输入文本与语言模型的预测的词拼接, 然后在输出给语言模型, 完成新一轮预测。然后再将预测到的词拼接到原来的输入上, 输入给语言模型, 完成下一轮预测。在循环迭代的“**自回归**”过程中, 我们不断生成新的词, 这些词便构成了一段文本。

以上的预训练过程被称为“**Teacher Forcing**”[21], 即每次都是用之前预测的“标准答案”(Ground Truth)来预测下一个词。在上面的例子中, 第二轮循环中, 我们用“长颈鹿脖子”来预测下一个词“长”, 而非选用  $o_1$  中概率最高的词“吃”或者其他可能输出的词。这样的 Teacher Forcing 的训练方式有两个好处: (1) 避免错误级联放大, 选用模型自己生成的词作为输入可能会有错误, 这样的错误循环输入, 将会不断的放大错误, 导致模型不能很好拟合训练集; (2) 方便进行并行加速训练, 因为下一个要预测的词并不依赖上一次的预测, 每次预测之间近乎独立, 所以可以进行并行加速。但是, Teacher Forcing 的训练方式也**曝光偏差**(Exposure Bias)的问题。曝光偏差是指 Teacher Forcing 训练模型的过程和模型在推理过程的差异。Teacher Forcing 在训练中, 模型将依赖于“标准答案”进行下一次的预测, 但是在推理预测中, 模型“**自回归**”的产生文本, 没有“标准答案”可参考。所以模型在

训练过程中和推理过程中存在偏差，可能推理效果较差，即存在曝光偏差。为解决曝光偏差的问题，Bengio 等人提出了针对 RNN 提出了 Scheduled Sampling 方法 [2]。Scheduled Sampling 让大模型在训练过程中循序渐进的使用一小部分自己生成的答案作为进行训练，使其在训练过程中对推理中无“标准答案”的情况进行预演。

由于 RNN 模型循环迭代的本质，其不易进行并行计算，导致其在输入序列较长时，训练较慢。下节将对容易并行的基于 Transformer 的语言模型进行介绍。

## 1.3 基于 Transformer 的语言模型

Transformer 是一类基于注意力机制（Attention）的模块化构建的神经网络结构。给定一个序列，Transformer 将一定量的历史状态和当前状态同时输入，然后进行加权相加。对历史状态和当前状态进行“通盘考虑”然后未来状态进行预测。基于 Transformer 的语言模型，以词序列作为输入，基于一定长度的上文和当前词来预测下一个词出现的概率。本节将先对 Transformer 的基本原理进行介绍，然后讲解如何利用 Transformer 构建语言模型。

### 1.3.1 Transformer

Transformer 是由两种模块组合构建的模块化网络结构。两种模块分布为：(1) 注意力（Attention）模块；(2) 全连接前馈（Fully-connected Feedforwad）模块。原始的 Transformer 采用 Encoder-Decoder 的架构。其中，自注意力模块由自注意力层（Self-Attention Layer）、残差连接（Residual Connections）和层正则化（Layer Normalization）组成。全连接模块由全连接前馈层，残差连接和层正则化组成。两个模块的结构示意图如图1.5所示。以下详细介绍每个层的原理及作用。

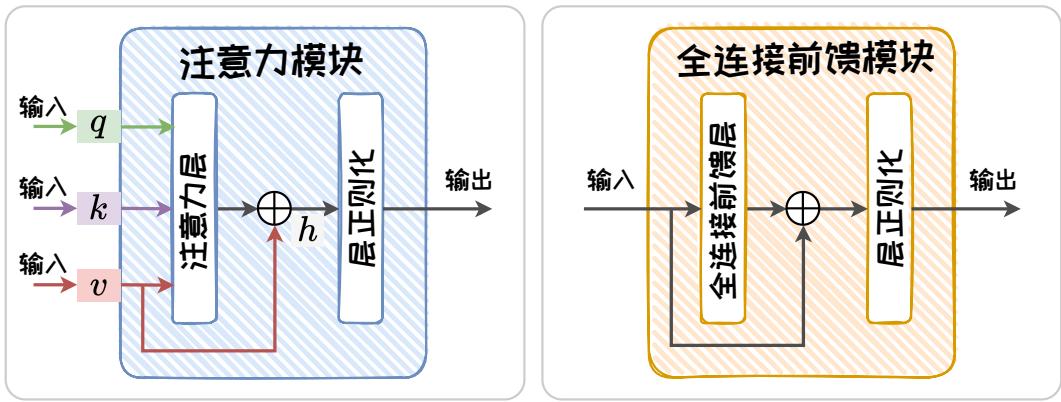


图 1.5: 自注意力模块与全连接前馈模块。

## 1. 注意力层 (Attention Layer)

注意力层采用加权平均的思想将前文信息叠加到当前状态上。如下图所示, 第  $t$  位置上的输出  $v_t$  与  $x_t$  以及其上文  $\{x_1, x_2, \dots, x_{t-1}\}$  有关。 $\{x_1, x_2, \dots, x_t\}$  对  $x_t$  的影响由对应的权重  $\{\alpha_1, \alpha_2, \dots, \alpha_t\}$  来确定。如下式所示:

$$\text{Attention}(x_t) = \sum_{i=1}^t \alpha_{t,i} x_i. \quad (1.31)$$

其中,  $\alpha_{t,i}$  代表了  $x_i$  与  $x_t$  的相关程度, 其可由如下公式进行计算。

$$\alpha_{t,i} = \text{softmax}(\text{sim}(x_t, x_i)) = \frac{\text{sim}(x_t, x_i)}{\sum_{i=1}^t \text{sim}(x_t, x_i)}. \quad (1.32)$$

其中,  $\text{sim}(x_t, x_i)$  用于度量  $x_i$  与  $x_t$  之间的相关程度, softmax 函数用于对此相关程度进行归一化。

Transformer 在朴素的注意力机制思想的基础上, 对其进行了进一步的改善。其进一步将输入编码为 query, key, value 三部分, 即将输入  $\{x_1, x_2, \dots, x_t\}$  编码为  $\{q_1, k_1, v_1, q_2, k_2, v_2, \dots, q_t, k_t, v_t\}$ 。其中, query 和 key 用于计算自注意力的权重  $\alpha$ , value 是对输入的编码。具体的,

$$\text{Attention}(x_t) = \sum_{i=1}^t \alpha_{t,i} v_i. \quad (1.33)$$

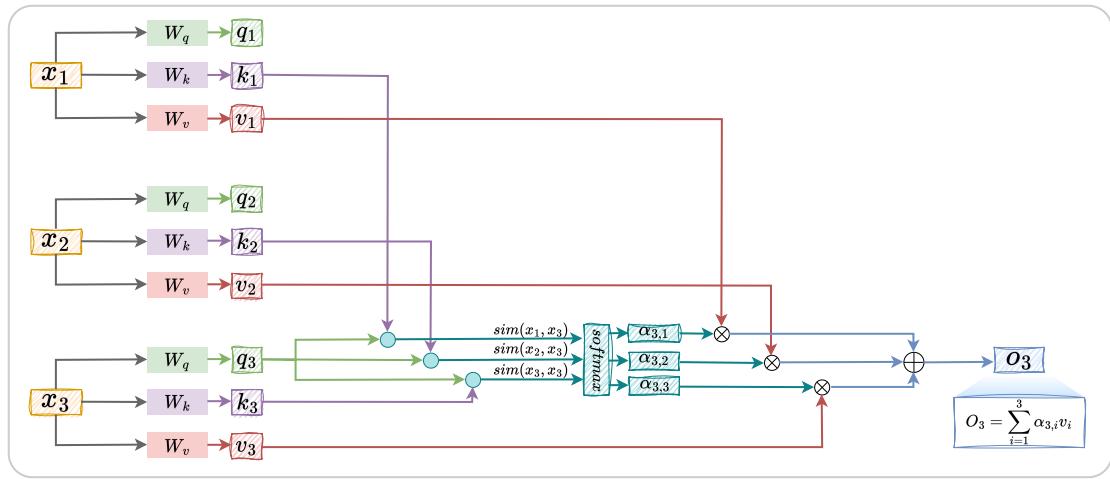


图 1.6: 自注意力机制示意图。

其中，

$$\alpha_{t,i} = \text{softmax}(\text{sim}(x_t, x_i)) = \frac{\text{sim}(q_t, k_i)}{\sum_{i=1}^t \text{sim}(q_t, k_i)} \circ \quad (1.34)$$

其中，

$$q_i = W_q x_i, \quad k_i = W_k x_i, \quad v_i = W_v x_i. \quad (1.35)$$

其中，  $W_q, W_k, W_v$  分别为 query, key, value 编码器的参数。以包含三个元素的输入  $\{x_1, x_2, x_3\}$  为例，Transformer 自注意力的实现图1.6所示。

## 2. 全连接前馈层 (Fully-connected Feedforwad Layer)

全连接前馈层占据了 Tranformer 近三分之二的参数，掌管着 Tranformer 模型的记忆。其可以看作是一种 Key-Value 模式的记忆存储管理模块 [7]。全连接前馈层包含两层，两层之间由 ReLU 作为激活函数。设全连接前馈层的输入为  $v$ ，全连接前馈层可由下式表示：

$$FFN(v) = \max(0, vW_1 + b_1)W_2 + b_2. \quad (1.36)$$

其中，  $W_1$  和  $W_2$  分别为第一层和第二层的权重参数，  $b_1$  和  $b_2$  分别为第一层和第二层的偏置参数。其中第一层的可看作神经记忆中的 key，而第二层可看作 value。

### 3. 层正则化 (Layer Normalization)

层正则化用以加速神经网络训练过程并取得更好的泛化性能 [1]。设输入到层正则化层的向量为  $v = \{v_i\}_{i=1}^n$ 。层正则化层将在  $v$  的每一维度  $v_i$  上都进行层正则化操作。具体地，层正则化操作可以表示为下列公式：

$$LN(v_i) = \frac{\alpha(v_i - \mu)}{\delta} + \beta. \quad (1.37)$$

其中， $\alpha$  和  $\beta$  为可学习参数。 $\mu$  和  $\delta$  分别是隐藏状态的均值和方差，可由下列公式分别计算。

$$\mu = \frac{1}{n} \sum_{i=1}^n v_i, \quad (1.38)$$

$$\delta = \sqrt{\frac{1}{n} \sum_{i=1}^n (v_i - \mu)^2}. \quad (1.39)$$

### 4. 残差连接 (Residual Connections)

引入残差连接可以有效解决梯度消失问题。在基本的 Transformer 编码模块中包含两个残差连接。第一个残差连接是将自注意力层的输入由一条旁路叠加到自注意力层的输出上，然后输入给层正则化。第二个残差连接是将全连接前馈层的输入由一条旁路引到全连接前馈层的输出上，然后输入给层正则化。

上述将层正则化置于残差连接之后的网络结构被成为 Post-LN Transformer。与之相对的，还有一种将层正则化置于残差连接之前的网络结构，称之为 Pre-LN Transformers。对比两者，Post-LN Transformer 应对表征坍塌 (Representation Collapse) 的能力更强，但处理梯度消失略弱。而 Pre-LN Transformers 可以更好的应对梯度消失，但处理表征坍塌的能力略弱。具体分析可参考文献 [7, 22]。

Transformer 是在 Encoder-Decoder 架构的基础上，重复连接自注意力模块和全连接前馈模块构建的神经网络模型。其整体结构如图1.7所示。其中，Encoder 部分由六个级联的 encoder layer 组成，每个 encoder layer 包含一个自注意力模块和一个

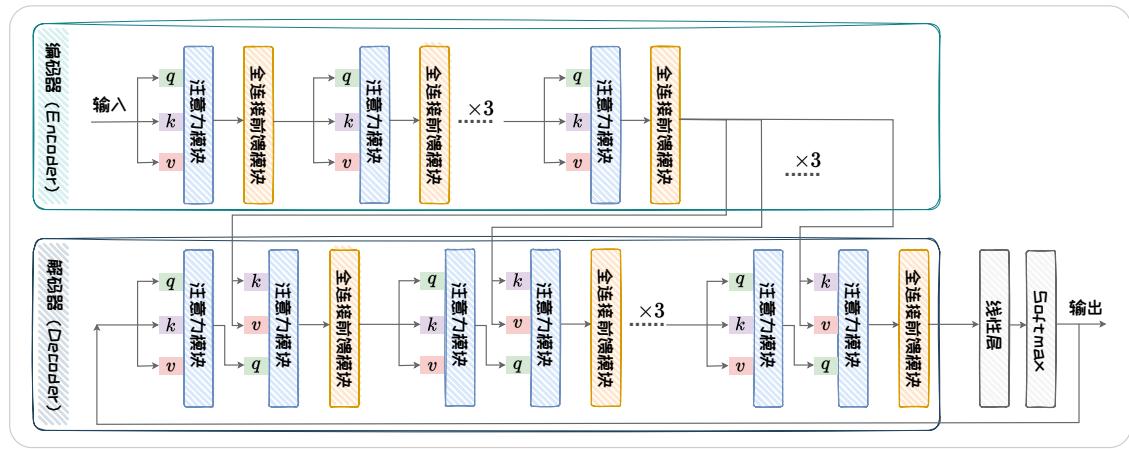


图 1.7: Transformer 结构示意图。

全连接前馈模块。Decoder 部分由六个级联的 decoder layer 组成，每个 decoder layer 包含两个自注意力模块和一个全连接前馈模块。与第六个 encoder layer 直接连接的 decoder layer（即第一个 decoder layer）中，第一个自注意力模块的 Q, K, V 输入为模型的输出，第二个自注意力模块的 Q, K 的输入为第六个 encoder layer 的输出，而 V 对应的输入为上一个自注意力模块的输出。第二个及以后的 decoder layer，第一个自注意力模块的 Q, K, V 输入为上一个 decoder layer 的输出，第二个自注意力模块的 Q, K 的输入为第六个 encoder layer 的输出，而 V 对应的输入为上一个自注意力模块的输出。

Transformer 的 Encoder 部分和 Decoder 部分都可以单独用于构造语言模型，分别对应 Encoder-Only 模型和 Decoder-Only 模型。Encoder-Only 模型和 Decoder-Only 模型的具体结构将在第二章中进行详细介绍。

### 1.3.2 基于 Transformer 的语言模型

在 Transformer 的基础上，可以设计多种预训练任务来训练语言模型。例如，我们可以基于 Transformer 的 Encoder 部分，结合“掩词补全”和下一句预测来训练 Encoder-Only 语言模型，如 Bert [5]；我们可以同时应用 Transformer 的 Encoder 和

Decoder 部分，结合“截断补全”、“顺序恢复”等多个有监督和自监督任务来训练 Encoder-Only 语言模型，如 T5 [18]；我们可以同时应用 Transformer 的 Decoder 部分，利用下一词预测任务来训练 Decoder-Only 语言模型，如 GPT-3 [3]。这些语言模型将在第二章中进行详细介绍。下面将以下一词预测任务为例，简单介绍训练 Transformer 语言模型的流程。

对词序列  $\{w_1, w_2, w_3, \dots, w_N\}$ ，基于 Transformer 的语言模型根据  $\{w_1, w_2, \dots, w_i\}$  预测下一个词  $w_{i+1}$  出现的概率。在基于 Transformer 的语言模型中，输出为一个向量，其中每一维代表着词典中对应词的概率。设词典中共有  $D$  个词，基于 Transformer 的语言模型的输出可表示为  $o_i = \{o_i[\hat{w}_d]\}_{d=1}^D$ ，其中， $o_i[\hat{w}_d]$  表示词典中的词  $\hat{w}_d$  出现的概率。因此，对 Transformer 的语言模型对词序列  $\{w_1, w_2, w_3, \dots, w_N\}$  整体出现的概率的预测为：

$$P(w_{1:N}) = \prod_{i=1}^{N-1} P(w_{i+1}|w_{1:i}) = \prod_{i=1}^N o_i[w_{i+1}] \quad (1.40)$$

与 RNN 语言模型相同，Transformer 语言模型也常用如下交叉熵函数作为损失函数。

$$l_{CE}(o_i) = - \sum_{d=1}^{|D|} I(w_d = w_{i+1}) \log o_i[w_{i+1}] = -\log o_i[w_{i+1}] \quad (1.41)$$

其中， $I(\cdot)$  为指示函数，当  $w_d = w_{i+1}$  时等于 1，当  $w_d \neq w_{i+1}$  时等于 0。

设训练集为  $S$ ，Transformer 语言模型的损失可以构造为：

$$L(W_k, W_q, W_v) = \frac{1}{N|S|} \sum_{s=1}^{|S|} \sum_{i=1}^N l_{CE}(o_i) \quad (1.42)$$

在此损失的基础上，构建计算图，执行反向传播方法，便可对 Transformer 语言模型进行训练。

上述训练过程结束之后，我们可以将 Encoder 的输出作为特征，然后应用这些特征解决下游任务。此外，还可在“自回归”的范式下完成文本生成任务。在自回归中，我们将输入文本与大模型的预测的词拼接，然后在输出给大模型，完成新

一轮预测。然后再将预测到的词拼接到原来的输入上，输入给大模型，完成下一轮预测。在循环迭代的“自回归”过程中，我们不断生成新的词，这些词便构成了一段文本。与训练 RNN 语言模型一样，Transformer 模型的预训练过程依然采用“Teacher Forcing”的范式。

相较于 RNN 模型串行的循环迭代模式，Transformer 的并行输入的特性，使其容易进行并行计算。但是，Transformer 并行输入的范式也导致网络模型的规模随输入序列长度的增长而平方次增长。这为应用 Transformer 处理长序列带来挑战。

## 1.4 语言模型的采样方法

语言模型的输出为一个向量，该向量的每一维代表着词典中对应词的概率。在采用自回归范式的文本生成任务中，语言模型将生成一组向量。将这组向量解码为文本的过程被成为语言模型解码。解码过程显著影响着生成文本的质量。当前，两类主流的解码方法可以总结为 (1). 概率最大化方法; (2). 随机采样方法。两类方法分别在下面章节中进行介绍。

### 1.4.1 概率最大化方法

设词典为  $D$ ，输入文本为  $\{w_1, w_2, w_3, \dots, w_N\}$ ，第  $i$  轮自回归中输出的向量为  $o_i = \{o_i[w_d]\}_{d=1}^{|D|}$ ，模型在  $M$  轮自回归后生成的文本为  $\{w_{N+1}, w_{N+2}, w_{N+3}, \dots, w_{N+M}\}$ 。生成文档的出现的概率可由下式进行计算。

$$P(w_{N+1:N+M}) = \prod_{i=N}^{N+M-1} P(w_{i+1}|w_{1:i}) = \prod_{i=N}^{N+M-1} o_i[w_{i+1}] \quad (1.43)$$

基于概率最大化的解码方法旨在最大化  $P(w_{N+1:N+M})$ ，以生成出可能性最高的文本。该问题的搜索空间大小为  $M^D$ ，是 NP-Hard 问题。现有概率最大化方法通常采用启发式搜索方法。本节将介绍两种常用的基于概率最大化的解码方法。

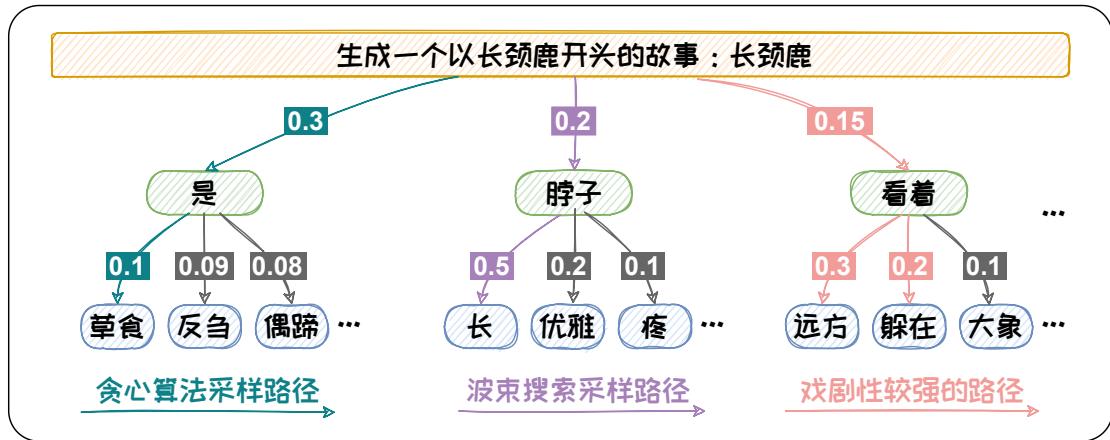


图 1.8: 贪心搜索与波束搜索对比以及概率最大化解码的潜在问题。

## 1. 贪心搜索 (Greedy Search)

贪心搜索在每轮预测中都选择概率最大的词，即

$$w_{i+1} = \arg \max_{w \in D} o_i[w] \quad (1.44)$$

贪心搜索只顾“眼前利益”，忽略了“远期效益”。**当前概率大的词有可能导致后续的词概率都很小**。贪心搜索容易陷入局部最优，难以达到全局最优解。以图1.8为例，当输入为“生成一个以长颈鹿开头的故事：长颈鹿”时，预测第一个词为“是”的概率最高，为 0.3。但选定“是”之后，其他的词的概率都偏低。如果按照贪心搜索的方式，我们最终得到的输出为“是反刍”。其概率仅为 0.03。而如果我们在第一个词选择了概率第二的“脖子”，然后第二个词选到了“长”，最终的概率可以达到 0.1。通过此例，可以看出贪心搜索在求解概率最大的时候容易陷入局部最优。为缓解此问题，可以采用波束搜索 (Beam Search) 方法进行解码。

## 2. 波束搜索 (Beam Search)

波束搜索在每轮预测中都先保留  $b$  个可能性最高的词  $B_i = \{w_{i+1}^1, w_{i+1}^2, \dots, w_{i+1}^b\}$ ，即：

$$\min\{o_i[w] \text{ for } w \in B_i\} > \max\{o_i[w] \text{ for } w \in D - B_i\}. \quad (1.45)$$

在结束搜索时，得到  $M$  个集合，即  $\{B_i\}_{i=1}^M$ 。找出最优组合使得联合概率最大，即：

$$\{w_{N+1}, \dots, w_{N+M}\} = \arg \max_{\{w^i \in B_i \text{ for } 1 \leq i \leq M\}} \prod_{i=1}^M o_{N+i}[w^i]。 \quad (1.46)$$

继续以上面的“生成一个以长颈鹿开头的故事”为例，从图1.8中可以看出如果我们采用  $b = 2$  的波束搜索方法，我们可以得到“是草食”，“是反刍”，“脖子长”，“脖子优雅”四个候选组合，对应的概率分别为：0.03, 0.027, 0.1, 0.04。我们容易选择到概率最高的“脖子长”。

但是，概率最大的文本通常是最为常见的文本。这些文本会略显平庸。在开放式文本生成中，无论是贪心搜索还是波束搜索都容易生成一些“废话文学”——重复且平庸的文本。其所生成的文本缺乏多样性 [19]。如在图1.8中的例子所示，概率最大的方法会生成“脖子长”。“长颈鹿脖子长”这样的文本新颖性较低。为了提升生成文本的新颖度，我们可以在解码过程中加入一些随机元素。这样的话就可以解码到一些不常见的组合，从而使得生成的文本更具创意，更适合开放式文本任务。在解码过程中加入随机性的方法，成为随机采样方法。下节将对随机采样方法进行介绍。

## 1.4.2 随机采样方法

为了增加生成文本的多样性，随机采样的方法在预测时增加了随机性。在每轮预测时，其先选出一组可能性高的候选词，然后按照其概率分布进行随机采样，采样出的词作为本轮的预测结果。当前，主流的 Top-K 采样和 Top-P 采样方法分别通过指定候选词数量和划定候选词概率阈值的方法对候选词进行选择。在采样方法中加入 Temperature 机制可以对候选词的概率分布进行调整。接下来将对 Top-K 采样、Top-P 采样和 Temperature 机制分别展开介绍。

### 1. Top-K 采样

Top-K 采样在每轮预测中都选取  $K$  个概率最高的词  $\{w_{i+1}^1, w_{i+1}^2, \dots, w_{i+1}^K\}$  作为

本轮的候选词集合，然后对这些词的概率用 softmax 函数进行归一化，得到如下分布函数

$$p(w_{i+1}^1, \dots, w_{i+1}^K) = \left\{ \frac{\exp(o_i[w_{i+1}^1])}{\sum_{j=1}^K \exp(o_i[w_{i+1}^j])}, \dots, \frac{\exp(o_i[w_{i+1}^K])}{\sum_{j=1}^K \exp(o_i[w_{i+1}^j])} \right\}。 \quad (1.47)$$

然后根据该分布采样出本轮的预测的结果，即

$$w_{i+1} \sim p(w_{i+1}^1, \dots, w_{i+1}^K)。 \quad (1.48)$$

Top-K 采样可以有效的增加生成文本的新颖度，例如在上述图1.8所示的例子中选用 Top-3 采样的策略，则有可能会选择到“看着躲在”。“长颈鹿看着躲在”可能是一个极具戏剧性的悬疑故事的开头。

但是，将候选集设置为固定的大小  $K$  将导致上述分布在不同轮次的预测中存在很大差异。当候选词的分布的方差较大的时候，可能会导致本轮预测选到概率较小、不符合常理的词，从而产生“胡言乱语”。例如，在如图1.9 (a) 所示的例子中，Top-2 采样有可能采样出“长颈鹿有四条裤子”。而当候选词的分布的方差较小的时候，甚至趋于均匀分布时，固定尺寸的候选集中无法容纳更多的具有相近概率的词，导致候选集不够丰富，从而导致所选词缺乏新颖性而产生“枯燥无趣”的文本。例如，在如下图1.9 (b) 所示的例子中，Top-2 采样，我们只能得到“长颈鹿用脖子来觅食”或者“长颈鹿用脖子来眺望”，这些都是人们熟知的长颈鹿脖子的用途，缺乏新意。但是其实长颈鹿的脖子还可以用于打架或睡觉，Top-2 采样的方式容易将这些新颖的不常见的知识排除。为了解决上述问题，我们可以使用 Top-P 采样，也称 Nucleus 采样。

## 2. Top-P 采样

为了解决固定候选集所带来的问题，Top-K 采样（即 Nucleus 采样）被提出 [9]。其设定阈值  $p$  来对候选集进行选取。其候选集可表示为  $S_p = \{w_{i+1}^1, w_{i+1}^2, \dots, w_{i+1}^{|S_p|}\}$ ，

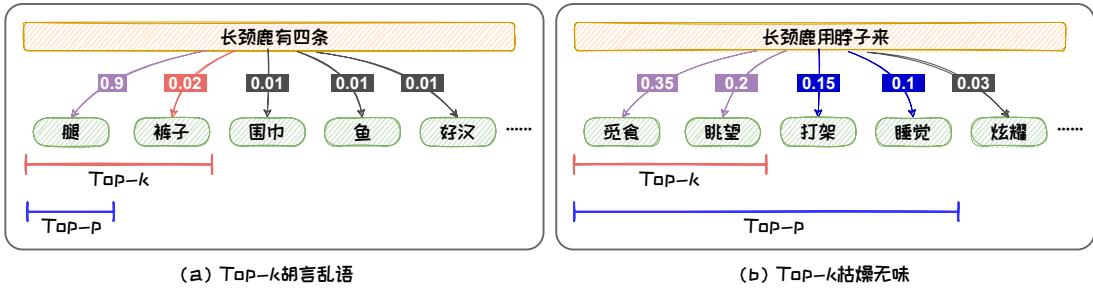


图 1.9: Top-K 采样的潜在问题及其与 Top-P 方法的对比。

其中, 对  $S_p$  有,  $\sum_{w \in S_p} o_i[w] \geq p$ 。候选集中元素的分布服从

$$p(w_{i+1}^1, \dots, w_{i+1}^{|S_p|}) = \left\{ \frac{\exp(o_i[w_{i+1}^1])}{\sum_{j=1}^{|S_p|} \exp(o_i[w_{i+1}^j])}, \dots, \frac{\exp(o_i[w_{i+1}^{|S_p|}])}{\sum_{j=1}^{|S_p|} \exp(o_i[w_{i+1}^j])} \right\}。 \quad (1.49)$$

然后根据该分布采样出本轮的预测的结果, 即

$$w_{i+1} \sim p(w_{i+1}^1, \dots, w_{i+1}^{|S_p|})。 \quad (1.50)$$

应用阈值作为候选集选取的标准之后, Top-K 采样可以避免选到概率较小、不符合常理的词, 从而减少“胡言乱语”。例如在图1.9 (a) 所示例子中, 我们若以 0.9 作为阈值, 则就可以很好的避免“长颈鹿有四条裤子”的问题。并且, 其还可以容纳更多的具有相近概率的词, 增加文本的丰富度, 改善“枯燥无趣”。例如在图1.9 (b) 所示的例子中, 我们若以 0.9 作为阈值, 则就可以包含打架、睡觉等长颈鹿脖子鲜为人知的用途。

### 3. Temperature 机制

Top-K 采样和 Nucleus 采样的随机性由语言模型输出的概率决定, 不可自由调整。但在不同场景中, 我们对于随机性的要求可能不一样。比如在开放文本生成中, 我们更倾向于生成更具创造力的文本, 所以我们需要采样具有更强的随机性。而在代码生成中, 我们希望生成的代码更为保守, 多以我们需要较弱的随机性。引入 Temperature 机制可以对解码随机性进行调节。Temperature 机制通过对 Softmax 函数中的自变量进行尺度变换, 然后利用 Softmax 函数的非线性实现对分布的控

制。设 Temperature 尺度变换的变量为  $T$ 。

引入 Temperature 后, Top-K 采样的候选集的分布如下所示:

$$p(w_{i+1}^1, \dots, w_{i+1}^K) = \left\{ \frac{\exp(\frac{o_i[w_{i+1}^1]}{T})}{\sum_{j=1}^K \exp(\frac{o_i[w_{i+1}^j]}{T})}, \dots, \frac{\exp(\frac{o_i[w_{i+1}^K]}{T})}{\sum_{j=1}^K \exp(\frac{o_i[w_{i+1}^j]}{T})} \right\}。 \quad (1.51)$$

引入 Temperature 后, Top-P 采样的候选集的分布如下所示:

$$p(w_{i+1}^1, \dots, w_{i+1}^{|S_p|}) = \left\{ \frac{\exp(\frac{o_i[w_{i+1}^1]}{T})}{\sum_{j=1}^{|S_p|} \exp(\frac{o_i[w_{i+1}^j]}{T})}, \dots, \frac{\exp(\frac{o_i[w_{i+1}^{|S_p|}]}{T})}{\sum_{j=1}^{|S_p|} \exp(\frac{o_i[w_{i+1}^j]}{T})} \right\}。 \quad (1.52)$$

容易看出, 当  $T > 1$  时, Temperature 机制会使得候选集中的词的概率差距减小, 分布变得更平坦, 从而增加随机性。当  $0 < T < 1$  时, Temperature 机制会使得候选集中的元素的概率差距加大, 强者越强, 弱者越弱, 概率高的候选词会容易被选到, 从而随机性变弱。Temperature 机制可以有效的对随机性进行调节来满足不同的需求。

## 1.5 语言模型的评测

得到一个语言模型后, 我们需要对其生成能力进行评测, 以判断其优劣。评测语言模型生成能力的方法可以分为两类。第一类方法不依赖具体任务, 直接通过语言模型的输出来评测模型的生成能力, 称之为内在评测 (Intrinsic Evaluation)。第二类方法通过某些具体任务, 如机器翻译、摘要生成等, 来评测语言模型处理这些具体生成任务的能力, 称之为外在评测 (Extrinsic Evaluation)。

### 1.5.1 内在评测

在内在评测中, 测试文本通常是与预训练中所用的文本独立同分布的文本构成, **不依赖于具体任务**。最为常用的内部评测指标是困惑度 (Perplexity) [10]。其度量了语言模型对测试文本感到“困惑”的程度。设测试文本为  $s_{test} = w_{1:|s_{test}|}$ 。语

言模型在测试文本  $s_{test}$  上的困惑度  $PPL$  可由下式计算：

$$PPL(s_{test}) = P(w_{1:|s_{test}|})^{-\frac{1}{|s_{test}|}} = \sqrt[|s_{test}|]{\prod_{i=1}^{|s_{test}|} \frac{1}{P(w_i|w_{<i})}}. \quad (1.53)$$

由上式可以看出，如果语言模型对测试文本越“肯定”（即生成测试文本的概率越高），则困惑度的值越小。而语言模型对测试文本越“不确定”（即生成测试文本的概率越低），则困惑度的值越大。由于测试文本和预训练文本同分布，预训练文本代表了我们想要让语言模型学会生成的文本，如果语言模型对与预训练文本同分布的测试文本上越不“困惑”，则说明语言模型越符合我们的对其训练的初衷。因此，困惑度可以一定程度上衡量语言模型的生成能力。

对困惑度进行改写，其可以改写成如下等价形式。

$$PPL(s_{test}) = \exp\left(-\frac{1}{|s_{test}|} \sum_{i=1}^{|s_{test}|} \log P(w_i|w_{<i})\right). \quad (1.54)$$

其中， $-\frac{1}{|s_{test}|} \sum_{i=1}^{|s_{test}|} \log P(w_i|w_{<i})$  可以看作是生成模型生成的词分布与测试样本真实的词分布间的交叉熵。此交叉熵是生成模型生成的词分布的信息熵的上界，即

$$-\frac{1}{|s_{test}|} \sum_{i=1}^{|s_{test}|} P(w_i|w_{<i}) \log P(w_i|w_{<i}) \leq -\frac{1}{|s_{test}|} \sum_{i=1}^{|s_{test}|} \log P(w_i|w_{<i}). \quad (1.55)$$

因此，困惑度减小也意味着熵减，意味着模型“胡言乱语”的可能性降低。

## 1.5.2 外在评测

在外在评测中，测试文本通常包括该任务上的问题和对应的标准答案，其依赖于具体任务。通过外在评测，我们可以评判语言模型处理特定任务的能力。外在评测方法通常可以分为基于统计指标的评测方法和基于语言模型的评测方法两类。以下对此两类方法中的经典方法进行介绍。

### 1. 基于统计指标的评测

基于统计指标的方法通过构造统计指标对语言模型的输出与标准答案间的契

合程度来评测语言模型处理该任务的能力。BLEU (BiLingual Evaluation Understudy) 和 ROUGE (Recall-Oriented Understudy for Gisting Evaluation) 是应用最为广泛的两种统计指标。其中, BLEU 是精度导向的指标, 而 ROUGE 是召回导向的指标。以下分别对这两个指标展开介绍。

BLEU 被提出用于评价模型在机器翻译 (Machine Translation, MT) 任务上的效果 [6]。其在词级别上计算生成的翻译与参考翻译间的重合程度。具体地, BLEU 计算多层次 n-gram 精度的几何平均。设生成的翻译文本的集合为  $S_{gen} = \{S_{gen}^i\}_{i=1}^{|S_{gen}|}$ , 对应的参考翻译集合为  $S_{ref} = \{S_{ref}^i\}_{i=1}^{|S_{ref}|}$ , 其中  $|S_{ref}| = |S_{gen}|$ 。原始的 n-gram 精度的定义如下:

$$Pr(g_n) = \frac{\sum_{s \in S_{gen}} \sum_{g_n \in s} Count_{match}(g_n, s_{ref})}{\sum_{s \in S_{gen}} \sum_{g_n \in s} Count(g_n)} \quad (1.56)$$

其中,  $g_n$  代表 n-gram。 $\sum_{s \in S_{gen}} \sum_{g_n \in s} Count_{match}(g_n, s_{ref})$  计算了生成的翻译与参考翻译的重合的 n-gram 的个数。 $\sum_{s \in S_{gen}} \sum_{g_n \in s} Count(g_n)$  计算了生成的翻译中包含的 n-gram 的总数。例如, MT 模型将“大语言模型”翻译成英文, 生成的翻译为“big language models”, 而参考文本为“large language models”。当  $n = 1$  时,  $Pr(g_1) = \frac{2}{3}$ 。当  $n = 2$  时,  $Pr(g_2) = \frac{1}{2}$ 。

在 n-gram 精度的基础上, BLEU 取 N 个 n-gram 精度的几何平均作为评测结果:

$$BLEU = \sqrt[N]{\prod_{i=1}^N Pr(g_n)} = \exp \left( \sum_{n=1}^N \log Pr(g_n) \right) \quad (1.57)$$

例如, 当  $N = 3$  时, BLEU 是 unigram 精度, bigram 精度, trigram 精度的几何平均。

在以上原始 BLEU 的基础上, 我们还可以通过对不同的 n-gram 精度进行加权或对不同的文本长度设置惩罚项来对 BLEU 进行调整, 从而得到更为贴近人类评测的结果。

ROUGE 被提出用于评价模型在摘要生成 (Summarization) 任务上的效果 [13]。常用的 ROUGE 评测包含 ROUGE-N, ROUGE-L, ROUGE-W, 和 ROUGE-S 四种。其中, ROUGE-N 是基于 n-gram 的召回指标, ROUGE-L 是基于最长公共子序列 (Longest Common Subsequence, LCS) 的召回指标。ROUGE-W 是在 ROUGE-L 的基础上, 引入对 LCS 的加权操作后的召回指标。ROUGE-S 是基于 Skip-bigram 的召回指标。下面给出 ROUGE-N, ROUGE-L 的定义。ROUGE-W 和 ROUGE-S 的具体计算方法可在 [13] 中找到。

ROUGE-N 的定义如下:

$$ROUGE - N = \frac{\sum_{s \in S_{ref}} \sum_{g_n \in s} Count_{match}(g_n, s_{gen})}{\sum_{s \in S_{ref}} \sum_{g_n \in s} Count(g_n)}。 \quad (1.58)$$

ROUGE-L 的定义如下:

$$ROUGE - L = \frac{(1 + \beta^2) R_r^g R_g^r}{R_r^g + \beta^2 R_g^r}。 \quad (1.59)$$

其中,

$$R_r^g = \frac{LCS(s_{ref}, s_{gen})}{|s_{ref}|}， \quad (1.60)$$

$$R_g^r = \frac{LCS(s_{ref}, s_{gen})}{|s_{gen}|}， \quad (1.61)$$

$LCS(s_{ref}, s_{gen})$  是模型生成的摘要  $s_{gen}$  与参考摘要  $s_{ref}$  间的最大公共子序列的长度,  $\beta = R_g^r / R_r^g$ 。

基于统计指标的评测方法通过对语言模型生成的答案和标准答案间的重叠程度进行评分。这样的评分无法完全适应生成任务中表达的多样性, 与人类的评测相差甚远, 尤其是在生成的样本具有较强的创造性和多样性的时候。为解决此问题, 可以在评测中引入一个其他语言模型作为“裁判”, 利用此“裁判”在预训练 (Pretraining) 阶段学习到的语言对齐能力对生成的文本进行评测。下面对这种引入“裁判”语言模型的评测方法进行介绍。

## 2. 基于语言模型的评测

目前基于语言模型的评测方法主要分为两类：(1) 基于上下文词嵌入 (Contextual Embeddings) 的评测方法；(2) 基于生成模型的评测方法。典型的基于上下文词嵌入的评测方法是 BERTScore [24]。典型的基于生成模型的评测方法是 G-EVAL [14]。与 BERTScore 相比，G-EVAL 无需人类标注的参考答案。这使其可以更好的适应到缺乏人类标注的任务中。

BERTScore 在 BERT 的上下文词嵌入向量的基础上，计算生成文本  $s_{gen}$  和参考文本  $s_{ref}$  间的相似度来对生成样本进行评测。BERT 将在第二章给出详细介绍。设生成文本包含  $|s_{gen}|$  个词，即  $s_{gen} = \{w_g^i\}_{i=1}^{|s_{gen}|}$ 。设参考文本包含  $|s_{ref}|$  个词，即  $s_{ref} = \{w_r^i\}_{i=1}^{|s_{ref}|}$ 。利用 BERT 分别得到  $s_{gen}$  和  $s_{ref}$  中每个词的上下文词嵌入向量，即  $v_g^i = BERT(w_g^i | s_{gen})$ ,  $v_r^i = BERT(w_r^i | s_{ref})$ 。利用生成文本和参考文本的词嵌入向量集合  $v_{gen} = \{v_g^i\}_{i=1}^{|v_{gen}|}$  和  $v_{ref} = \{v_r^i\}_{i=1}^{|v_{ref}|}$  便可计算 BERTScore。BERTScore 从精度 (Precision)，召回 (Recall) 和 F1 量度三个方面对生成文档进行评测。其定义分别如下：

$$P_{BERT} = \frac{1}{|v_{gen}|} \sum_{i=1}^{|v_{gen}|} \max_{v_r \in v_{ref}} v_g^\top v_r, \quad (1.62)$$

$$R_{BERT} = \frac{1}{|v_{ref}|} \sum_{i=1}^{|v_{ref}|} \max_{v_g \in v_{gen}} v_r^\top v_g, \quad (1.63)$$

$$F_{BERT} = \frac{2P_{BERT} \cdot R_{BERT}}{P_{BERT} + R_{BERT}}. \quad (1.64)$$

相较于统计评测指标，BERTScore 更接近人类评测结果。但是，BERTScore 依赖于人类给出的参考文本。这使其无法应用于缺乏人类标注样本的场景中。得益于生成式大语言模型的发展，G-EVAL 利用 GPT-4 在没有参考文本的情况下对生成文本进行评分。G-EVAL 通过提示工程 (Prompt Engineering) 引导 GPT-4 输出评

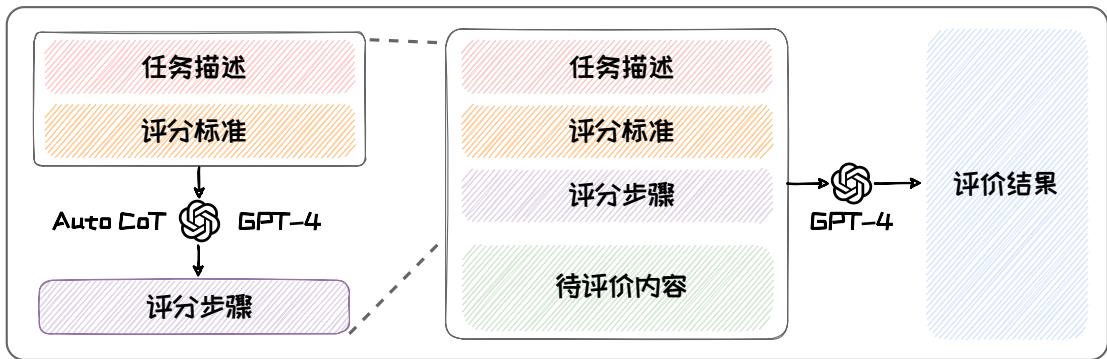


图 1.10: G-EVAL 评测流程。

测分数。Prompt Engineering 将在第四章进行详细讲解。

如下图所示, G-EVAL 的 Prompt 分为三部分: (1) 任务描述与评分标准; (2) 评测步骤; (3) 输入文本与生成的文本。在第一部分中, 任务描述指明需要的评测的任务式什么 (如摘要生成), 评分标准给出评分需要的范围, 评分需要考虑的因素等内容。第二部分的评测步骤是在第一部分内容的基础上由 GPT-4 自己生成的思维链 (Chain-of-Thoughts, CoT)。本书的第三章将对思维链进行详细讲解。第三部分的输入文本与生成的文本是源文本和待评测模型生成的文本。例如摘要生成任务中的输入文本是原文, 而生成的文本就是生成摘要。将上述三部分组合在一个 prompt 里面然后输入给 GPT-4, GPT-4 便可给出对应的评分。直接将 GPT-4 给出的得分作为评分会出现区分度不够的问题, 因此, G-EVAL 还引入了对所有可能得分进行加权平均的机制来进行改进 [14]。

除 G-EVAL 外, 近期还有多种基于生成模型的评测方法被提出 [12]。其中典型的有 InstructScore [23], 其除了给出数值的评分, 还可以给出对该得分的解释。基于生成模型的评测方法相较于基于统计指标的方法和基于上下文词嵌入的评测方法而言, 在准确性、灵活性、可解释性等方面都具有独到的优势。可以预见, 未来基于生成模型的评测方法将得到更为广泛的关注和应用。

## 参考文献

- [1] Lei Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. “Layer Normalization”. In: *CoRR* (2016).
- [2] Samy Bengio et al. “Scheduled Sampling for Sequence Prediction with Recurrent Neural Networks”. In: *NeurIPS*. 2015.
- [3] Tom B. Brown et al. “Language Models are Few-Shot Learners”. In: *NeurIPS*. 2020.
- [4] Junyoung Chung et al. “Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling”. In: *CoRR* (2014).
- [5] Jacob Devlin et al. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. In: *NAACL-HLT*. 2019.
- [6] Markus Freitag, David Grangier, and Isaac Caswell. “BLEU might be Guilty but References are not Innocent”. In: *EMNLP*. 2020.
- [7] Mor Geva et al. “Transformer Feed-Forward Layers Are Key-Value Memories”. In: *EMNLP*. 2021.
- [8] Sepp Hochreiter and Jürgen Schmidhuber. “Long Short-Term Memory”. In: *Neural Computing* 9.8 (1997), pp. 1735–1780.
- [9] Ari Holtzman et al. “The Curious Case of Neural Text Degeneration”. In: *ICLR*. 2020.
- [10] F. Jelinek et al. “Perplexity—a measure of the difficulty of speech recognition tasks”. In: *The Journal of the Acoustical Society of America* 62 (1997), S63–S63.
- [11] Dan Jurafsky and James H. Martin. *Speech and language processing: an introduction to natural language processing, computational linguistics, and speech recognition, 2nd Edition*. Prentice Hall series in artificial intelligence. Prentice Hall, Pearson Education International, 2009. ISBN: 9780135041963.
- [12] Zhen Li et al. “Leveraging Large Language Models for NLG Evaluation: A Survey”. In: *CoRR* (2024).
- [13] Chin-Yew Lin. “Rouge: A package for automatic evaluation of summaries”. In: *ACL*. 2004.
- [14] Yang Liu et al. “Gpteval: Nlg evaluation using gpt-4 with better human alignment”. In: *EMNLP*. 2023.

- [15] Christopher D. Manning and Hinrich Schütze. *Foundations of statistical natural language processing*. MIT Press, 2001. ISBN: 978-0-262-13360-9.
- [16] OpenAI. “GPT-4 Technical Report”. In: *arXiv preprint arXiv:2303.08774* (2023).
- [17] Razvan Pascanu, Tomás Mikolov, and Yoshua Bengio. “On the difficulty of training recurrent neural networks”. In: *ICML*. 2013.
- [18] Colin Raffel et al. “Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer”. In: *Journal Of Machine Learning Research* 21 (2020), 140:1–140:67.
- [19] Ashwin K. Vijayakumar et al. “Diverse Beam Search: Decoding Diverse Solutions from Neural Sequence Models”. In: *CoRR* (2016).
- [20] Joseph Weizenbaum. “ELIZA - a computer program for the study of natural language communication between man and machine”. In: *Communications Of The ACM* 9.1 (1966), pp. 36–45.
- [21] Ronald J. Williams and David Zipser. “A Learning Algorithm for Continually Running Fully Recurrent Neural Networks”. In: *Neural Computing* 1.2 (1989), pp. 270–280.
- [22] Shufang Xie et al. “ResiDual: Transformer with Dual Residual Connections”. In: *CoRR* (2023).
- [23] Wenda Xu et al. “INSTRUCTSCORE: Towards Explainable Text Generation Evaluation with Automatic Feedback”. In: *EMNLP*. 2023.
- [24] Tianyi Zhang et al. “BERTScore: Evaluating Text Generation with BERT”. In: *ICLR*. 2020.

## 2 大语言模型

随着数据资源和计算能力的爆发式增长，语言模型的规模和性能得到了质的飞跃。大语言模型 (Large Language Model, LLM) 凭借着庞大的参数量和丰富的训练数据迅速崛起，正引领着生成式人工智能 (Artificial Intelligence Generated Content, AIGC) 发展的新潮流。截至 2024 年 6 月，大语言模型的发展呈现出爆炸式增长，各类模型层出不穷，预示着一个由大语言模型驱动的 AIGC 新时代的到来。这些模型不仅在学术界引起了广泛关注，也在工业界得到了实际应用，展现了其在多种领域的变革潜力。在本章中，我们将深入探讨大语言模型的相关背景，并分别介绍 Encoder-only、Encoder-Decoder 以及 Decoder-only 三种主流架构。通过列举每种架构的典型模型，深入分析它们的模型架构、训练方法以及主要创新之处等。最后，本章还将简单介绍其他一些具有创新性的架构模型，以展现当前语言模型领域的多样性和丰富性。

## 2.1 大数据 + 大模型 → 新智能

在自然语言处理的前沿领域，大语言模型正以其庞大的模型规模、海量的数据吞吐量和卓越的模型性能，推动着一场技术革新的浪潮。当我们谈论“大模型”之大时，所指的不仅仅是模型规模的庞大，还包括训练数据规模的庞大，以及由此衍生出的模型能力的强大。这些模型如同探索未知领域的巨轮，不仅在已有的技术上不断突破性能的极限，更在新能力的探索中展现出惊人的潜力。截止 2024 年 6 月，国内外已经见证了超过百种大语言模型的诞生，它们在学术界和工业界均产生了深远的影响。图2.1展示了具有重要影响力的模型。



图 2.1: 大语言模型涌现能力的三个阶段

大语言模型的发展历程可以大致划分为三个阶段。2017 至 2018 年是基础模型的萌芽期，Transformer 架构的诞生以及 BERT[10] 和 GPT-1[29] 模型的推出开启了预训练语言模型的新纪元。2019 至 2022 年是大语言模型的发展期，通过 GPT-2<sup>1</sup>、T5[31] 以及 GPT-3[4] 等模型在参数规模以及能力上的大幅提升，研究者开始深入探索大语言模型的潜力。2022 年起则是大语言模型的突破期，ChatGPT<sup>2</sup> 以及 GPT-4<sup>3</sup> 等模型的发布标志着大语言模型相关技术的显著进步，各大公司和研究机构纷纷推出了自己的模型，例如百川智能的百川大模型 [45]，百度的文心一言等，推动

<sup>1</sup><https://openai.com/index/gpt-2-1-5b-release>

<sup>2</sup><https://openai.com/blog/chatgpt>

<sup>3</sup><https://openai.com/index/gpt-4-research>

了大语言模型的快速发展。

本节将深入剖析大型语言模型在发展历程中的演变，特别是在能力增强和新能力涌现方面的进展。我们将从模型规模和数据规模的增长出发，探讨这些因素如何共同作用，促进了模型性能的飞跃和新功能的出现。

### 2.1.1 大数据 + 大模型 → 能力增强

在数字化浪潮的推动下，数据如同汇聚的洪流，而模型则如同乘风破浪的巨舰。数据规模的增长为模型提供了更丰富的信息源，意味着模型可以学习到更多样化的语言模式和语义关系。而模型规模的不断扩大，进一步增加了模型的表达能力，使其能够捕捉到更加细微的语言特征和复杂的语言结构。在如此庞大的模型参数规模以及多样化的训练数据共同作用下，模型内在对数据分布的拟合能力不断提升，从而更有效地应对复杂多变的数据环境 [6]。

然而模型规模和数据规模的增长并非没有代价，他们带来了更高的计算成本和存储需求，这要求我们在设计时必须在资源和性能之间找到平衡点。为了解决这一挑战，大语言模型的扩展法则（Scaling Law）应运而生，这一法则揭示了模型的能力随模型规模和数据规模的变化关系，为大语言模型的设计和优化提供了清晰的指导。本章节将介绍两种主要的扩展法则：OpenAI 提出的 Kaplan-McCandlish 扩展法则以及 DeepMind 提出的 Chinchilla 扩展法则。

#### 1. Kaplan-McCandlish 扩展法则

2020 年，OpenAI 团队的 Jared Kaplan 和 Sam McCandlish 等人 [17] 首次探究了神经网络的性能与数据规模  $D$  以及模型规模  $N$  之间的函数关系。他们在不同规模的数据集（从 2200 万到 230 亿个 Token）和不同规模的模型下（从 768 到 15 亿个参数）进行实验，并根据实验结果拟合出了两个基本公式：

$$L(D) = \left( \frac{D}{D_c} \right)^{\alpha_D}, \alpha_D \sim -0.095, D_c \sim 5.4 \times 10^{13} \quad (2.1)$$

$$L(N) = \left( \frac{N}{N_c} \right)^{\alpha_N}, \alpha_N \sim -0.076, N_c \sim 8.8 \times 10^{13} \quad (2.2)$$

这里的  $L(N)$  表示在数据规模固定的情况下，不同模型规模下的交叉熵损失函数， $L(D)$  表示在模型规模固定的情况下，不同数据规模下的交叉熵损失函数。 $L$  的值衡量的是模型在拟合数据分布时的误差，值越小说明模型对数据分布的拟合越准确，其 **自身学习能力** 也就越强大。从实验结果以及对应公式中可以发现，模型性能与参数模型以及数据规模这两个因素均高度正相关。但在相同规模的下，模型性能受模型本身具体的形状结构影响较小。因此，设计更庞大的模型和收集更丰富的数据，成为了提升大型模型性能的关键。

此外，OpenAI 在进一步研究计算预算的最优配比时发现，总计算量  $C$  与数据量  $D$  和模型规模  $N$  的乘积近似成正比，即  $C \approx 6ND$ 。在这一条件下，若计算预算增加，为了实现最优模型性能，数据集量  $D$  以及模型规模  $N$  也都应同步增加，但是 **模型规模的增长速度应该略快于数据规模的增长速度**。具体而言，两者的最优配置应当为  $N_{opt} \propto C^{0.73}$ ,  $D_{opt} \propto C^{0.27}$ 。这意味着，如果总计算预算增加了 10 倍，那么模型规模应当扩大约 5.37 倍，而数据规模应当相应扩大约 1.86 倍，从而使模型达到最优性能。

OpenAI 提出的这一扩展法则不仅定量地展现了数据规模和模型规模对模型能力的重要性，也指出了模型规模上的投入应当略高于数据规模上的投入。这一发现为理解语言模型的内在工作机制提供了新的见解，也为如何高效地训练这些模型提供了宝贵的指导。

## 2. Chinchilla 扩展法则

谷歌旗下 DeepMind 团队并不完全认同“模型规模的增长速度应该略高于数据

规模的增长速度”这一观点。2022 年，他们对更大范围的模型规模（从 7000 万到 1600 亿个参数）以及数据规模（从 50 亿到 5000 亿个 Token）进行了实验，并提出了如下 Chinchilla 拓展法则 [15]：

$$L(N, D) = E + \frac{A}{N^\alpha} + \frac{B}{D^\beta}, \quad (2.3)$$

$$E = 1.69, A = 406.4, B = 410.7, \alpha = 0.34, \beta = 0.28 \quad (2.4)$$

接着 DeepMind 同样探索了计算预算的最优分配问题，最终得出数据集规模  $D$  与模型规模  $N$  的最优配置为  $N_{opt} \propto C^{0.46}, D_{opt} \propto C^{0.54}$ 。这一结果表明，**数据集量  $D$  与模型规模  $N$  几乎同等重要**，如果总计算预算增加了 10 倍，那么模型规模以及数据规模都应当扩大约 3.16 倍。后续，Google 团队在 2023 年 5 月发布的 PaLM 2 的技术报告 [2] 中也再次证实了这一观点。

此外，Chinchilla 扩展法则还进一步提出，最佳的数据集大小应当是模型规模的 20 倍。例如，对于一个 7B (70 亿参数) 的模型，最理想的训练数据集大小应为 140B (1400 亿) 个 Token。但先前很多模型的预训练数据量并不够，例如 OpenAI 的 GPT-3[4] 模型中最大的版本具有 1750 亿个参数，却只使用了 3000 亿个 Token 来训练；同样，微软的 MT-NLG[37] 模型拥有 5300 亿个参数，而训练用的 Token 数量却只有 2700 亿。因此，DeepMind 推出了数据规模 20 倍于模型规模的 Chinchilla 模型 (700 亿参数，1.4 万亿个 Token)，最终在性能上取得了显著突破。

DeepMind 提出的 Chinchilla 扩展法则是对 OpenAI 先前研究的补充和优化，强调了数据规模在提升模型性能中的重要性，指出**模型规模和数据规模应该以相同的比例增加**，开创了大语言模型发展的一个新方向：不再单纯追求模型规模的增加，而是**优化模型规模与数据规模的比例**。

## 2.1.2 大数据 + 大模型 → 能力扩展

如图2.2所示，模型训练数据规模以及参数数量的不断提升，不仅带来了上述学习能力的稳步增强，还为大模型“解锁”了一些新的能力<sup>4</sup>，例如常识推理能力、数学运算能力、逻辑证明能力、代码生成能力等等。扩展出的这些新能力并非是模型在具体的专项下游任务上通过训练生成，而是如同凭空涌现出来一般<sup>5</sup>。这些能力因此被称为大语言模型的涌现能力（Emergent Abilities）。



图 2.2: 大语言模型能力随模型规模涌现，图片由 GPT-4o 生成

涌现能力往往具有突变性和不可预见性。类似于非线性系统中的“相变”，即系统在某个阈值点发生显著变化，这些能力也并没有一个平滑的、逐渐积累的过程，而是在模型达到一定的复杂度和数据规模后，很突然地从“不存在”转变为“存在”<sup>[34]</sup>。接下来将基于 GPT 系列的演变介绍一些较为典型的涌现能力。

- **常识推理**: 常识推理（Commonsense Reasoning）是指大语言模型基于常识知识和逻辑进行理解和推断的能力。它包括对日常生活中普遍接受的事实、事件和行为模式的理解，并利用这些知识来回答问题、解决问题和生成相关内容。在 GPT 系列中，GPT-1 和 GPT-2 在常识推理方面的能力非常有限，常常

<sup>4</sup><https://research.google/blog/pathways-language-model-palm-scaling-to-540-billion-parameters-for-breakthrough-performance>

<sup>5</sup><https://www.assemblyai.com/blog/emergent-abilities-of-large-language-models>

会出现错误的推断或缺乏详细的解释。而 GPT-3 的较大版本能够在大多数情况下生成合理和连贯的常识性回答。至于具有 1750 亿参数的 GPT-3 最大版本以及后续的 GPT-4 等模型，则能够处理高度复杂的常识推理任务，生成的回答在逻辑性、一致性和细节上都表现出色。

- **代码生成**: 代码生成 (Code Generation) 是指大语言模型基于自然语言描述自动生成编程代码的能力。这种能力包括理解编程语言的语法和语义、解析用户需求、生成相应代码，以及在某些情况下进行代码优化和错误修复。GPT-1 和 GPT-2 仅能生成非常简单的代码片段，但是无法有效理解具体的编程需求。130 亿参数的 GPT-3 模型出现时，已经能很好地处理常见的编程任务和生成结构化代码片段，但在极其复杂或特定领域的任务上仍有限。在参数量达到 1750 亿时，模型则能够处理复杂编程任务，多语言代码生成，代码优化和错误修复等，展示出高质量的代码生成和理解能力。
- **逻辑推理**: 逻辑推理 (Logical Reasoning) 是指大语言模型在理解和推理过程中，基于给定的信息和规则进行合乎逻辑的推断和结论的能力。这种能力包括简单的条件推理、多步逻辑推理、以及在复杂情境下保持逻辑一致性。GPT-1 和 GPT-2 作为早期的生成预训练模型，在逻辑推理方面的能力非常有限，甚至对于 130 亿参数版本的 GPT-3 模型而言，虽然能处理一部分逻辑推理任务，但在复杂度和精确性上仍存在一定局限性。直到 1750 亿参数版本，GPT-3 才能够处理复杂的逻辑推理任务，生成详细和连贯的推理链条。
- .....

这些涌现出来的能力使得大语言模型可以在不进行专项训练的前提下完成各类任务，但同时也带来了诸多挑战，如模型的可解释性、信息安全与隐私、伦理和公平性问题，以及对计算资源的大量需求。解决这些挑战需要在技术、法律和社会层面进行综合考虑和应对，以确保大语言模型的健康和可持续发展。

## 2.2 大语言模型架构

在语言模型的发展历程中, Transformer[42] 框架的问世代表着一个划时代的转折点。其独特的自注意力机制极大地提升了模型对序列数据的处理能力, 在捕捉长距离依赖关系方面表现尤为出色。此外, Transformer 框架对并行计算的支持极大地加速了模型的训练过程。当前, 大多数大语言模型仍以 Transformer 框架为核心, 并在此基础上发展出了三种主流架构, 分别是 Encoder-only 架构, Decoder-only 架构以及 Encoder-Decoder 架构。这三种架构在设计和功能上各有不同。本章节将介绍这三种架构, 并分析它们之间的区别及各自的演变趋势。

### 2.2.1 经典模型架构介绍

本节将对上述三种架构的设计理念、训练方式及其应用进行详细介绍。

#### 1. Encoder-only 架构

Encoder-only 架构仅选取了 Transformer 中的编码器 (Encoder) 部分, 用于接收输入文本并生成上下文相关的表示。具体来说, Encoder-only 架构包含三个部分, 分别是输入编码部分, 特征编码部分以及任务处理部分, 具体的模型结构如图2.3所示。其中输入编码部分包含分词、向量化以及添加位置编码三个过程。而特征编码部分则是由多个相同的编码模块 (Encoder Block) 堆叠而成, 其中每个编码模块包含自注意力模块 (Self-Attention) 和全连接前馈模块。任务处理模块用于针对具体任务生成输出。在2.3 章节中会针对 Encoder-only 架构进行更具体的介绍。

Encoder-only 架构模型的预训练阶段和推理阶段在输入编码和特征编码部分是一致的, 而任务处理部分则需根据任务的不同特性来进行定制化的设计。

在输入编码部分, 原始输入文本会被分词器 (Tokenizer) 拆解为 Token 序列, 随后通过词表和词嵌入 (Embedding) 矩阵映射为向量序列。这一过程会在3.1 章

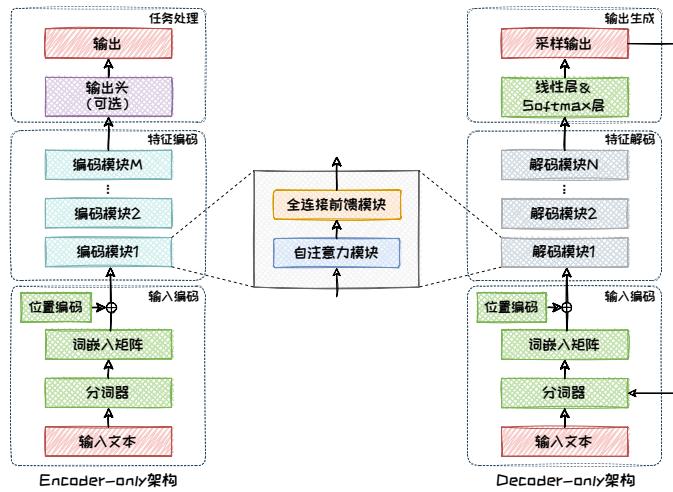


图 2.3: Encoder-only 架构和 Decoder-only 架构

节中被具体介绍。接着，每个向量序列会被添加上位置编码（Positional Encoding）以引入位置信息。在**特征编码部分**，先前得到的向量序列会被并行输入到图2.3中所示的编码模块1中进行处理，随后又并行通过编码模块2处理，重复这一过程直至向量序列通过所有编码模块。在**任务处理部分**，如果处于预训练阶段，输出头一般是全连接层，用于完成掩码预测等预训练任务，随后还需要进一步计算损失并进行反向传播以更新模型参数。但在微调或是推理过程中，可以根据具体任务需求设计不同的输出头，将最后一个编码模块的输出映射到任务所需的内容。例如，可以添加一个分类头来完成情感分析或主题分类任务，或者添加一个生成头来完成文本摘要任务，也可以直接输出用于语义相似度度量任务。

## 2. Decoder-only 架构

与上述 Encoder-only 架构恰好相反，Decoder-only 架构仅选取了 Transformer 中的解码器（Decoder）部分，基于先前已生成的部分文本，来生成下一个词。Decoder-only 架构同样包含三个部分，分别是**输入编码部分**、**特征解码部分**以及**输出生成部分**，具体的模型结构如图2.3所示。

在输入编码部分，流程与 Encoder-only 架构完全一致，包含分词、向量化以及

添加位置编码三个过程，将原始输入转化为带有位置编码的向量序列。

特征解码部分由多个相同的解码模块 (Decoder Block) 堆叠而成，每个解码模块包含掩码自注意力 (Masked Self-Attention) 模块和全连接前馈模块。值得注意的是，由于 Decoder-only 架构中仅包含解码器部分，因此原生 Transformer 中与编码器交互的交叉注意力模块也被移除了，从而进一步简化了模型。输入编码中得到的完整向量序列同样会逐个被解码模块所处理。

输出生成模块将经过特征解码后的向量通过线性层以及 Softmax 层来生成词表的概率分布，并通过采样从概率分布中选择最合适的 Token 输出。输出生成模块在训练和推理阶段则有所不同。如果处于训练阶段，输出生成模块会使用 Teacher Forcing 的方式，将真实文本中的已知标签并行输入到解码模块来预测后续所有标签，随后模型计算预测结果和真实标记之间的损失，并通过反向传播更新模型参数。但如果处于推理阶段，由于缺失真实标签，只能通过自回归的方式来生成输出。具体来说，在每个时间步，解码器使用先前所有 Token 作为输入，并将当前采样得到的 Token。传输回原始输入中用于下一个 Token 的预测。这个过程循环进行，直到生成结束标记 <end> 或达到模型输出的最大长度限制。在这一过程中，后续 Token 的输入依赖于当前 Token 的预测结果，因此只能串行地逐步生成输出序列。在 2.5 章节中会针对 Decoder-only 架构进行更具体的介绍。

### 3. Encoder-Decoder 架构

Encoder-Decoder 架构可以理解为上述 Encoder-only 架构以及 Decoder-only 架构的组合，也最接近 Transformer 的完整形式。其中编码器将输入序列编码为上下文相关的表示，解码器则根据编码器的输出和先前生成的部分文本生成目标序列。Encoder-Decoder 架构由多个堆叠起来的编码模块和解码模块连接而成。编码模块包含自注意力模块和全连接前馈模块，解码模块则包括掩码自注意力模块，交叉注意力模块和全连接前馈模块。其中编码模块与解码模块之间通过交叉注意力模

块进行交互。在2.4章节中会针对 Encoder-Decoder 架构进行更具体的介绍。

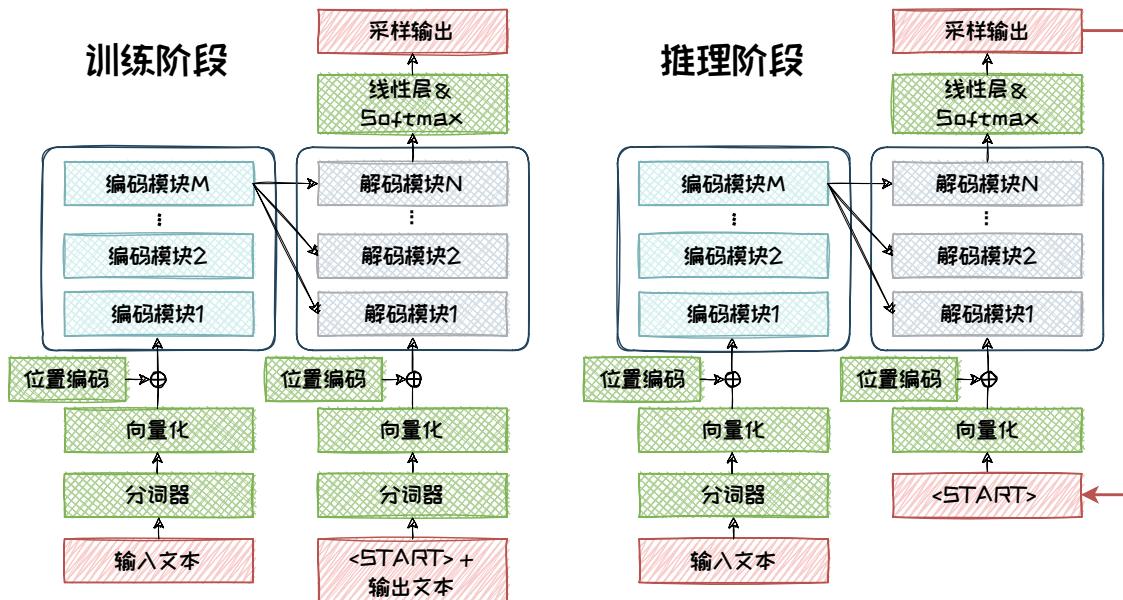


图 2.4: Encoder-Decoder 架构

图2.4展示了 Encoder-Decoder 架构模型在训练和推理阶段的流程。Encoder-Decoder 架构模型在这两个阶段中存在着显著区别。

在训练阶段，Encoder-Decoder 架构首先将输入文本通过分词器拆解为 Token 序列，然后通过词嵌入矩阵映射为嵌入向量，并添加位置编码。嵌入向量序列依次通过编码器的多个编码模块，生成上下文表示，这一过程与 Encoder-only 架构一致。接下来，目标序列的最前方会被添加 <start> 标记，从而实现整体右移，然后同样被分词、词嵌入和位置编码处理后，并行输入到解码器。解码器在训练过程中使用 Teacher Forcing 技术，即使用真实文本中的已知部分作为输入，因此可以并行预测所有的 Token。目标序列的嵌入向量依次通过多个解码模块，每个解码模块包含遮掩多头自注意力机制、多头交叉注意力机制（与最后一个编码模块的输出交互）和前馈神经网络。最后，通过一个分类头将解码器的输出映射到词汇表，同步对所有 Token 进行预测。

在推理阶段，Encoder 部分与训练阶段相同，但 Decoder 部分初始只有起始标

记  $\langle\text{start}\rangle$ ，因此与 Decoder-only 架构一样，只能通过自回归的方式，基于先前生成的所有文本，逐步生成后续的 Token。

### 2.2.2 模型架构对比

上述的 Encoder-only、Encoder-Decoder 和 Decoder-only 这三种模型架构虽然都源自于 Transformer 框架，但他们在注意力、适用任务和生成能力等方面都有显著区别。接下来将从上述方面对这三种架构的主要区别进行分析。

#### 1. 注意力矩阵

注意力矩阵（Attention Matrix）是 Transformer 中的核心组件，用于计算输入序列中各个 Token 之间的依赖关系。通过注意力机制，模型可以在生成或理解当前 Token 时，动态地关注序列中其他 Token 的相关信息，而注意力矩阵则控制在处理当前 Token 时可以关注到其他哪些 Token。三种架构在注意力矩阵上有着显著差异，如图2.5所示。

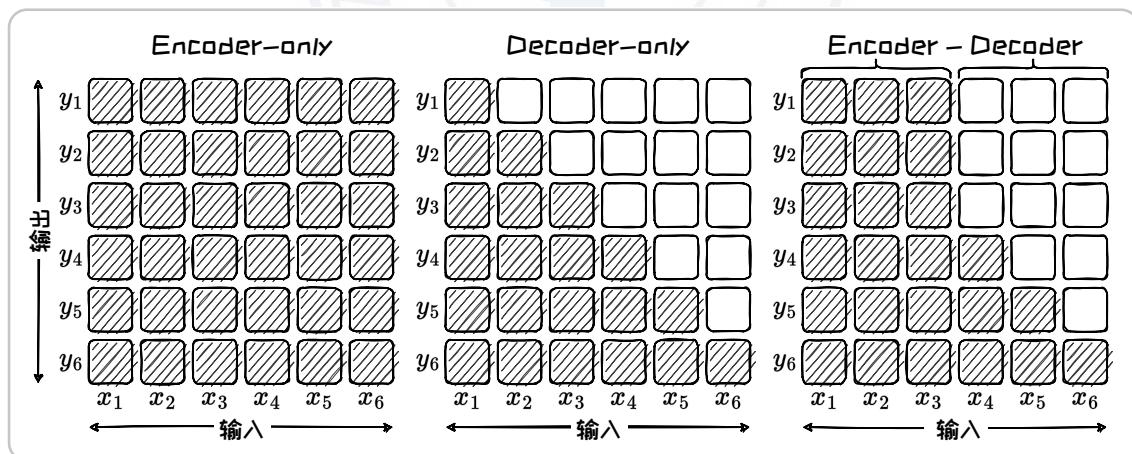


图 2.5: 三种架构的注意力矩阵

**Encoder-only** 架构中的注意力矩阵来自于自注意力模块，用于捕捉输入序列中各个 Token 之间的关系。Encoder-only 架构的注意力矩阵呈现出“完全”的注意力，即对于每个 Token 的理解都依赖于整个输入序列中的所有 Token。例如，在模

型将  $x_i$  转化为对应的上下文向量  $y_i$  时，可以利用输入  $x_1 \sim x_n$  的所有信息，这就是所谓的**双向注意力机制**。在这种双向注意力机制的作用下，模型能够同时利用前后文信息，因此能够理解复杂的语义关系和上下文依赖。

**Decoder-only** 架构中的注意力矩阵来自于掩码自注意力模块，但 Decoder-only 架构的注意力矩阵则呈现出“下三角”的注意力，即在对当前 Token 的预测时只能依赖于先前的历史 Token 信息，也就是所谓的**单向注意力机制**。例如，在生成  $y_i$  时，依赖于先前的所有 Token，即  $y_1 \sim y_{i-1}$ ，这保证了生成过程的顺序性和连贯性。

**Encoder-Decoder** 架构中的注意力矩阵则更为复杂，分为编码器的自注意力、解码器的掩码自注意力和交叉注意力三个部分。编码器部分的自注意力矩阵与 Encoder-only 架构类似，生成输入序列的上下文表示，呈现出“完全”的注意力。解码器的掩码自注意力矩阵与 Decoder-only 架构类似，确保在生成当前标记时只关注之前生成的标记，呈现出“下三角”的注意力。允许解码器在生成每个 Token 时动态地参考编码器生成的上下文表示，从而确保生成的输出与输入序列高度相关。

## 2. 适用任务

由于独特的模型设计以及注意力矩阵上的差异，在同等参数规模下，这三种架构的模型在适用任务上，也都各有倾向。

**Encoder-only** 架构中的双向注意力机制允许模型在预测每个标记时都充分考虑序列中的前后文信息，从而能够捕捉到丰富的语义和依赖关系。因此，Encoder-only 架构的模型主要用于**自然语言理解** (Natural Language Understanding, NLU) 任务，尤其是情感分析或文本分类等判别任务。然而，由于缺少解码器组件，Encoder-only 架构的模型无法直接生成所需目标序列，因此在**自然语言生成**任务 (Natural Language Generation, NLG) 上可能表现不如专门设计的生成模型。

**Decoder-only** 架构使用了掩码 (Mask) 操作，确保在每个时间步生成当前标记时只能访问前面的标记，并通过自回归生成机制，从起始标记开始逐步生成文本，

每一步依赖于前面生成的内容。通过大规模预训练数据，Decoder-only 架构的模型能够生成高质量、连贯的文本，在自动故事生成、新闻文章生成此类不依赖于特定的输入文本的**无条件文本生成任务**中表现出色。

然而，在模型规模有限的情况下（例如 GPT-1 以及 GPT-2 等模型），由于缺乏编码器提供的双向上下文信息，Decoder-only 架构的模型在理解复杂输入数据时存在**一定局限性**，表现可能不如 Encoder-Decoder 架构。

**Encoder-Decode 架构**结合了编码器和解码器的优势。编码器部分负责将输入序列编码为固定长度的上下文表示，而解码器部分则基于这个上下文表示逐步生成输出序列。这种编码器和解码器的结合，使得模型可以有效地处理复杂的输入条件，并生成相关且连贯的高质量内容。因此，Encoder-Decoder 架构的模型非常适合于处理各种复杂的**有条件生成任务**，例如机器翻译、文本摘要和问答系统等需要同时理解输入并生成相应输出的场景。

但随着模型规模以及数据规模的显著增长，Decoder-only 架构的模型已经证明了它们以**统一的框架来处理多样化任务**的能力。这种架构的统一性优势主要归功于其卓越的生成性能和适应性。当前，以 GPT-3、GPT-4 等为代表的大型 Decoder-only 语言模型，得益于其庞大的模型规模以及丰富的训练语料，展示了强大的记忆能力以及执行多种复杂任务的能力，包括但不限于文本分类、问答系统、机器翻译、文本续写和逻辑推理等。

为了进一步提升这些架构的性能，研究者们正在探索各种创新的方法，如改进的预训练策略、模型微调技术，以及将不同架构的优势结合起来的混合模型设计，旨在突破现有限制，实现更加全面和高效的任务处理能力。

### 2.2.3 模型架构的历史演变

随着时间的推移，我们见证了这三种架构的演变和流行趋势，接下来我们将对这三种架构的历史演变及其原因进行介绍和分析。

#### 1. 流行趋势

在 2018 年左右，也就是大语言模型发展的早期阶段，BERT 和 GPT-1 分别作为 Encoder-only 和 Decoder-only 架构的代表模型几乎同时出现。但 BERT 强大的上下文理解能力使 Encoder-only 架构得到了广泛的探索和应用，而 Decoder-only 架构则相对沉寂。

然而，随着机器翻译等应用需求的增加，Encoder-only 架构由于缺乏解码部分，无法进行文本生成等任务，因此应用逐渐饱和。同时，2019 年末诞生的一众 Encoder-Decoder 架构的模型，由于能够有效处理序列到序列任务，从而逐渐成为主流。

自 2021 年之后，在 GPT-3 等模型的推动下，Decoder-only 架构愈发繁荣，甚至逐渐主导了大语言模型的发展。尽管如此，Encoder-Decoder 架构也仍然在开源社区中活跃，不断探索新的技术和应用，且大部分处于开源状态。至于 Encoder-only 架构，在 BERT 带来最初的爆炸性增长之后，其关注度有所下降，但也仍然在部分特定任务中发挥着重要作用，例如文本分类、情感分析、命名实体识别和问答系统等任务。

#### 2. 原因分析

整体来看，大语言模型架构经历了从最早的 Encoder-only 到 Encoder-Decoder，再到 Decoder-only 的发展趋势。其中 Encoder-only 架构应用受限的主要原因在于其缺乏专门用于生成输出的组件，因此仅适用于对输入文本进行深入理解，却不适合生成任务，因此逐渐失去了吸引力。随后，Encoder-Decoder 架构凭借在机器翻译

等需要同时理解输入和生成输出的任务中的出色发挥，一度成为主流架构。然而，随着生成任务需求的增加，Encoder-Decoder 架构再次被 Decoder-only 架构所超越。Decoder-only 架构的崛起是多种因素共同影响的结果，当前较为主流的原因分析主要基于效率、训练复杂性以及双向注意力导致的低秩问题等角度。

### (1) 效率与训练复杂性

不同架构在效率方面的差异是导致其演变趋势的主要原因之一。

**Encoder-only** 架构主要聚焦于对输入数据的表示和理解，这使得它在执行理解型任务时表现出较高的效率。但当涉及到文本生成任务时，其效率则不尽如人意。由于**缺少专门用于生成输出的组件**，它需要通过迭代进行掩码预测来生成文本，这需要模型进行多次前向传播以逐步填充文本中的掩码部分，从而导致计算资源的大量消耗。而**Encoder-Decoder** 架构使用编码器来处理输入，并使用解码器来生成输出。这种架构的**参数规模通常非常庞大**，随之带来的是计算复杂度的显著增加。特别是在处理长序列数据时，编码器和解码器均面临**沉重的计算负担**，这限制了模型的整体效率。相比之下，**Decoder-only** 架构通过仅使用解码器来简化整个计算流程。它采用自回归生成策略，能够逐步构建输出文本，每一步的生成过程仅需考虑已经生成的文本部分。这样的设计大幅**减少了模型的参数量和计算需求**，非常适合需要进行大量文本生成的应用场景。

### (2) 低秩问题

Encoder 中的双向注意力机制允许每个 Token 同时考虑前后文的信息，但这种双向注意力的矩阵很容易**退化至低秩状态**，从而造成**模型表达能力的下降** [11]。然而，Decoder-only 架构中的注意力矩阵呈现出下三角矩阵的形式，因此**始终是满秩的**，这意味着模型具有**更强的捕捉复杂、高维的数据结构的能力**。

先前 Encoder-Decoder 架构之所以能够在某些场景下比 Decoder-only 架构表现更好，可能是因为它增加了编码器，因此参数规模也变得更庞大。所以，在同等参

数量、同等推理成本下，Decoder-only 架构可能更有优势<sup>6</sup>。

这三种架构的发展不仅推动了自然语言处理技术的进步，也为各行各业带来了深远的影响，从提升搜索引擎的准确性到开发更加智能的对话系统。随着研究的深入，未来的语言模型有望在更具优势的架构下变得更加强大和灵活，并在更多领域发挥作用。后续章节将进一步介绍这三种架构及其对应的几种经典模型。

## 2.3 基于 Encoder-only 架构的大语言模型

Encoder-only 架构凭借着其独特的双向编码模型在自然语言处理任务中表现出色，尤其是在各类需要深入理解输入文本的任务中。本章将对双向编码模型以及几种较为典型的 Encoder-only 架构模型进行介绍。

### 2.3.1 双向编码模型以及 Encoder-only 架构

Encoder-only 架构的核心是能够全面覆盖初始输入所有内容的**双向编码模型** (Bidirectional Encoder Model)。在处理输入序列时，双向编码模型同时包含了从左往右的正向注意力以及从右往左的反向注意力，能够充分捕捉每个 Token 的上下文信息，因此也被称为具有**全面的注意力机制**。双向编码器通过其上下文感知能力和动态表示的优势，显著提升了自然语言处理任务的性能。与先前常用的静态编码方式以及同时期的 GPT-1 模型所使用的因果语言模型相比，双向编码模型具有明显的优势。

- 与为每个词提供一个固定向量表示的**静态编码方式**(例如 Word2Vec 和 GloVe)相比，双向编码器为每个词生成**动态的上下文嵌入** (Contextual Embedding)，这种嵌入依赖于输入序列的上下文信息。因此，模型能够更好地理解词与词

<sup>6</sup><https://kexue.fm/archives/952>

之间的依赖关系和语义信息，也能有效地处理词的多义性。这种动态表示使得双向编码器在句子级别的任务上表现出色，显著超过了静态词嵌入方法的性能 [21]。

- 与因果语言模型相比，双向编码模型能够利用未来词汇信息。单向的因果语言模型只能利用输入序列中之前的词汇来预测下一个词，这种单向编码方式在处理一些需要全面上下文理解的任务时存在局限性。而双向编码模型通过同时考虑前后文信息来生成每个词的表示，能够提供更丰富的上下文信息，在自然语言理解任务中表现更好。凭借着双向编码模型的强大能力，BERT 在各类自然语言理解任务上的性能要远远超过同时期同规模的 GPT-1 模型。

Encoder-only 架构基于双向编码模型，选用了 Transformer 架构中的编码器部分。虽然 Encoder-only 模型不直接生成文本，但它们生成的上下文嵌入对于理解输入文本的结构和含义至关重要。因此，这些模型在需要精细理解和复杂推理的自然语言处理任务中表现出色，成为了自然语言处理领域的关键工具。当前，BERT[10] 及其变体，如 RoBERTa[24]、ALBERT[18] 等，都是基于 Encoder-only 架构的主流大语言模型。接下来将对这些模型进行介绍。

### 2.3.2 BERT 语言模型

BERT (Bidirectional Encoder Representations from Transformers) 是一种基于 Encoder-only 架构的预训练语言模型，由 Google AI 团队于 2018 年 10 月提出。作为早期 Encoder-only 架构的代表，BERT 在自然语言处理领域带来了突破性的改进。其核心创新在于通过双向编码模型深入挖掘文本的上下文信息，从而为各种下游任务提供优秀的上下文嵌入 (Contextual Embedding)。本节将对 BERT 模型的结构、预训练方式以及下游任务进行介绍。

#### 1. BERT 模型结构

BERT 模型的结构与 Transformer 中的编码器几乎一致，都是由多个编码模块堆叠而成，每个编码模块包含一个多头自注意力模块和一个全连接前馈模块。根据参数量的不同，BERT 模型共有 BERT-Base 和 BERT-Large 两个版本。其中 **BERT-Base** 由 12 个编码模块堆叠而成，其中隐藏层维度为 768，自注意力头的数量为 12，**总参数数量为 1.1 亿**；**BERT-Large** 由 24 个编码模块堆叠而成，其中隐藏层维度为 1024，自注意力头的数量为 16，**总参数数量约为 3.4 亿**。

## 2. BERT 预训练方式

BERT 使用小说数据集 BookCorpus<sup>49</sup>（包含约 8 亿个 Token）和英语维基百科数据集<sup>7</sup>（包含约 25 亿个 Token）进行预训练，总计约 33 亿个 Token，总数据量达到了 15GB 左右。在预训练任务上，BERT 开创性地提出了掩码语言建模（Masked Language Model, MLM）和下文预测（Next Sentence Prediction, NSP）这两种预训练任务来学习生成上下文嵌入。其完整的预训练流程如图2.6所示。

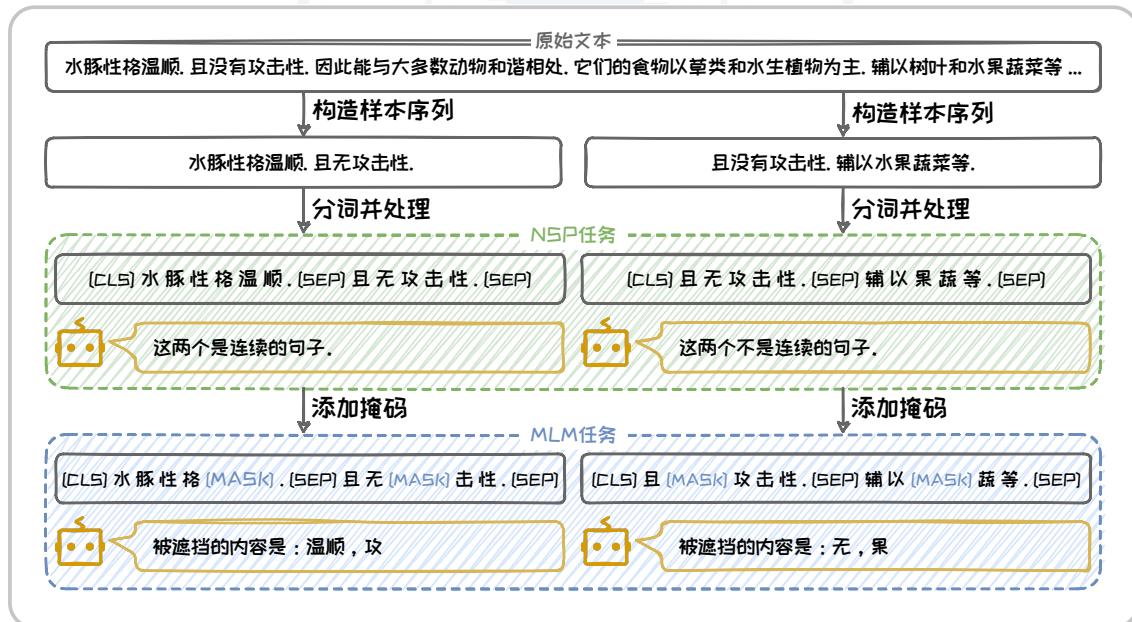


图 2.6: BERT 预训练任务

具体而言，BERT 先基于给定的原始文本构造多个样本序列，每个样本序列

<sup>7</sup><https://dumps.wikimedia.org/>

由原始文本中的两个句子组成，并且由 50% 的概率是连续的两个句子，另外 50% 的概率是随机挑选的两个句子。随后，对构造出来的样本序列进行分词，并添加上 [CLS] 以及 [SEP] 等特殊标签。其中 [CLS] 标签出现在样本序列的开头，而 [SEP] 标签出现在每个句子的结尾。

接着，BERT 利用处理后的序列进行下文预测任务，利用模型判断样本序列是否包含两个连续的句子。这个任务训练 BERT 学习句子之间的关系，使其能够捕捉句子层面的语言特征。这对于理解文本的逻辑流、句子之间的关联性有很大帮助，特别是在需要理解文档层次结构的 NLP 任务中，如问答任务和自然语言推理等。

最后，BERT 选择样本序列中大约 15% 的 Token 进行掩码，将其替换为特殊标签 [MASK] 或者随机单词。模型需要预测这些被破坏的 Token 的原内容。这个过程就像完型填空题，模型需要根据周围的上下文来预测出缺失内容。预测过程使用的交叉熵损失函数驱动了 BERT 模型中参数的优化，使其能够学习到文本的双向上下文表示。值得注意的是，在 MLM 任务的训练过程中，BERT 模型仅针对那些被随机遮蔽的 Token 进行学习，即通过计算这些 Token 的预测损失来更新模型参数。

这两种预训练任务的设置，使 BERT 在理解语言的深度和广度上都有显著提升，BERT 不仅学习到 Token 的细粒度特征，还能把握长距离的依赖关系以及句子间的关系，为各种下游任务提供了强大的语言理解能力基础。

### 3. BERT 下游任务

BERT 可以应用于各种自然语言处理任务，包括但不限于文本分类（如情感分析）、问答系统、文本匹配（如自然语言推断）和语义相似度计算等。然而，由于 BERT 的输出是输入中所有 Token 的向量表示，因此总长度不固定，无法直接应用于各类下游任务。为了解决这一问题，BERT 设计了 [CLS] 标签来提取整个输入序列的聚合表示。[CLS] 标签是专门为分类和汇总任务设计的特殊标记。其全称是“Classification Token”，即分类标记。通过注意力机制，[CLS] 标签汇总整个输入序

列的信息，生成一个固定长度的向量表示，从而实现对所有 Token 序列信息的概括，便于处理各种下游任务。

在**文本分类任务**中，可以将输出中 [CLS] 标签对应的向量提取出来，传递给一个**全连接层**，从而用于分类，例如判断整个句子的情绪是积极、消极或是中立的。在**问答系统任务**中，需要输入问题以及一段相关的文本，即“[CLS] 问题 [SEP] 文本 [SEP]”。最终同样提取出 [CLS] 标签的对应向量，并传递给**两个全连接层**，用于判断答案是否存在于相关文本中。如果存在，这两个全连接层分别用于输出答案的起始和结束位置。通过这种方式，BERT 能够从提供的段落中准确提取出问题的答案。在**语义相似度任务**中，需要计算两段或者多段文本之间的语义相似度。在这一任务中，可以通过构造“[CLS] 文本 1[SEP] 文本 2[SEP]”的方式，结合一个**线性层**来直接输出两个文本之间的相似度；也可以**不添加额外的组件**，直接提取 [CLS] 标签对应的向量，再利用额外的相似度度量方法（例如余弦相似度）来计算多段文本之间的相似度。

BERT 通过双向编码以及特定的预训练任务，显著提升了自然语言处理任务的性能。其在多种任务中的应用都展示了强大的泛化能力和实用性，不仅为学术研究提供了新的基准，还在实际应用中得到广泛采用，极大推动了自然语言处理技术的发展。

### 2.3.3 BERT 衍生语言模型

BERT 的突破性成功还催生了一系列相关衍生模型，这些模型继承了 BERT 双向编码的核心特性，并在其基础上进行了改进和优化，以提高性能或者效率，在特定任务和应用场景中展现出了卓越的性能。较为代表的衍生模型为 RoBERTa[24]、ALBERT[18] 以及 ELECTRA[8] 等，接下来将对着三种模型分别展开介绍。

#### 1. RoBERTa 语言模型

RoBERTa (Robustly Optimized BERT Pretraining Approach) 由 Facebook AI (现更名 Meta) 研究院于 2019 年 7 月提出，旨在解决 BERT 在训练程度上不充分这一问题，以提升预训练语言模型的性能。RoBERTa 在 BERT 的基础上采用了更大的数据集（包括更多的英文书籍、维基百科和其他网页数据）、更长的训练时间（包括更大的批次大小和更多的训练步数）以及更细致的超参数调整来优化预训练的过程，从而提高模型在各种自然语言处理任务上的性能和鲁棒性。接下来将对 RoBERTa 模型的结构、预训练方式以及下游任务进行介绍。

### (1) RoBERTa 模型结构

RoBERTa 在结构上与 BERT 基本一致，基于多层堆叠的编码模块，每个编码模块包含多头自注意力模块和全连接前馈模块。RoBERTa 同样有两个版本，分别是 RoBERTa-Base 和 RoBERTa-Large。其中 **RoBERTa-Base** 与 BERT-Base 对标，由 12 个编码模块堆叠而成，其中隐藏层维度为 768，自注意力头的数量为 12，**总参数数量约为 1.2 亿**；**RoBERTa-Large** 则与 BERT-Large 对标，由 24 个编码模块堆叠而成，其中隐藏层维度为 1024，自注意力头的数量为 16，**总参数数量约为 3.5 亿**。

### (2) RoBERTa 预训练方式

在预训练语料库的选择上，RoBERTa 在 BERT 原有的小说数据集 BookCorpus[49]（包含约 8 亿个 Token）以及英语维基百科数据集<sup>8</sup>（包含约 25 亿个 Token）的基础上，添加了新闻数据集 CC-News<sup>9</sup>（包含约 76GB 的新闻文章）、网页开放数据集 OpenWebText<sup>10</sup>（包含约 38GB 的网页文本内容）以及故事数据集 Stories[41]（包含约 31GB 的故事文本），总数据量达到约 160GB。

至于具体的预训练任务，RoBERTa 移除了 BERT 中的下文预测任务，并将 BERT 原生的静态掩码语言建模任务更改为**动态掩码语言建模**。具体而言，BERT

---

<sup>8</sup><https://dumps.wikimedia.org>

<sup>9</sup><http://web.archive.org/save/http://commoncrawl.org/2016/10/newsdataset-available>

<sup>10</sup><http://Skylion007.github.io/OpenWebTextCorpus>

在数据预处理期间对句子进行掩码，随后在每个训练 epoch 中，掩码位置不再变化。而 RoBERTa 则将训练数据复制成 10 个副本，分别进行掩码。在同样训练 40 个 epoch 的前提下，BERT 在其静态掩码后的文本上训练了 40 次，而 RoBERTa 将 10 个不同掩码后的副本分别训练了 4 次，从而增加模型训练的多样性，有助于模型学习到更丰富的上下文信息。

这些改进使得 RoBERTa 在理解上下文和处理长文本方面表现出色，尤其是在捕捉细微的语义差异和上下文依赖性方面。

## 2. ALBERT 语言模型

ALBERT (A Lite BERT) 是由 Google Research 团队于 2019 年 9 月提出的轻量级 BERT 模型，旨在通过参数共享和嵌入分解技术来减少模型的参数量和内存占用，从而提高训练和推理效率。BERT 的难点之一在于其巨大的参数量，最常用的 BERT-Base 模型有着 1.1 亿个参数，而 BERT-Large 更是有着 3.4 亿个参数，这使得 BERT 不仅训练困难，推理时间也较长。与 RoBERTa 为了提高模型性能进一步增加参数量不同，ALBERT 在设计过程中通过参数因子分解技术和跨层参数共享技术显著减少了参数的数量。接下来将对 ALBERT 模型的结构、预训练方式以及下游任务进行介绍。

### (1) ALBERT 模型结构

ALBERT 的结构与 BERT 以及 RoBERTa 都类似，由多层堆叠的编码模块组成。但是 ALBERT 通过参数因子分解以及跨层参数共享，在相同的模型架构下，显著减少了模型的参数量。下面将参数因子分解和跨层参数共享分别展开介绍。

#### 参数因子分解

在 BERT 中，Embedding 层的输出向量维度  $E$  与隐藏层的向量维度  $H$  是一致的，即将 Embedding 层的输出直接作为后续编码模块的输入。而 ALBERT 将 Embedding 层的矩阵先进行分解，将大小为  $V$  的词表对应的独热编码向量先投影

到低维 Embedding 层的维度，再将其投影回隐藏层。

具体来说，BERT-Base 模型的词表大小  $V$  为 30000，隐藏层的向量维度  $H$  为 768，那么其 Embedding 层需要的参数数量为  $V \times H$ ，共 2304 万。而 ALBERT 设计的 Embedding 层维度为 128，并将这一过程拆解为  $V \times E \times H$ ，此时其 Embedding 层所需的参数数量为  $V \times E + E \times H$ ，共 394 万左右，只有 BERT 的六分之一。对于隐藏层向量维度  $H$  更大的 Large 版本，ALBERT 节省参数空间的特性将更明显，可以将参数量压缩到 BERT 的八分之一左右。

### 跨层参数共享

以经典的 BERT-Base 模型为例，模型中共有 12 层相同架构的编码模块，所有 Transformer 块的参数都是独立训练的。

ALBERT 为了降低模型的参数量，提出了跨层参数共享机制，只学习第一层编码模块的参数，并将其直接共享给其他所有层。该机制在一定程度上牺牲了模型性能，但显著提升了参数存储空间的压缩比，从而实现了更高效的资源利用。

在如此设计下，ALBERT 共提出了四个版本的模型，分别是 ALBERT-Base、ALBERT-Large、ALBERT-XLarge 以及 ALBERT-XXLarge。其中 **ALBERT-Base** 与 BERT-Base 对标，由 12 个编码模块堆叠而成，中间嵌入分解维度为 128，隐藏层维度为 768，自注意力头的数量为 12，**总参数数量约为 0.12 亿**；**ALBERT Large** 与 BERT-Large 对标，由 24 个编码模块堆叠而成，中间嵌入分解维度为 128，隐藏层维度为 1024，自注意力头的数量为 16，**总参数数量约为 0.18 亿**；**ALBERT X-Large** 由 12 个编码模块堆叠而成，中间嵌入分解维度为 128，隐藏层维度为 2048，自注意力头的数量为 16，**总参数数量约为 0.6 亿**；**ALBERT XX-Large** 由 12 个编码模块堆叠而成，中间嵌入分解维度为 128，隐藏层维度为 4096，自注意力头的数量为 64，**总参数数量约为 2.2 亿**。

### (2) ALBERT 预训练方式

ALBERT 使用与 BERT 完全一致的数据集来进行预训练，即小说数据集 Book-Corpus<sup>49</sup>（包含约 8 亿个 Token）以及英语维基百科数据集<sup>11</sup>（包含约 25 亿个 Token），总计 33 亿个 Token，约 15GB 数据量。另外，在预训练任务的选择上，ALBERT 保留了 BERT 中的掩码语言建模任务，并将下文预测任务替换为句序预测（Sentence Order Prediction, SOP），如图2.7所示。具体而言，ALBERT 将文本中两句连续的句子拼接起来，或是将两个句子的顺序翻转后拼接作为样本，而模型需要预测该样本中的两个句子是正序还是反序。

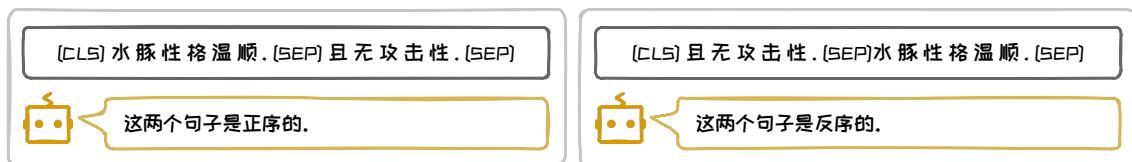


图 2.7: ALBERT 句序预测任务

与 BERT 相比，ALBERT 通过创新的参数共享和因子化技术，在较好地保持原有性能的同时显著减少了模型的参数数量，这使得它在资源受限的环境中更加实用，处理大规模数据集和复杂任务时更高效，并降低了模型部署和维护的成本。

### 3. ELECTRA 语言模型

ELECTRA (Efficiently Learning an Encoder that Classifies Token Replacements Accurately) 是由 Google Brain 和斯坦福大学的研究人员于 2020 年 3 月提出的另一种 BERT 变体，旨在解决大规模预训练语言模型中的效率和可扩展性问题。通过使用生成器-判别器架构，ELECTRA 能够更高效地利用预训练数据，提高了模型在下游任务中的表现。

在模型结构上，ELECTRA 在 BERT 原有的掩码语言建模基础上结合了生成对抗网络（Generative Adversarial Network, GAN）的思想，采用了一种生成器-判别器结构。具体来说，ELECTRA 模型包含一个生成器和一个判别器，其中生成器（Generator）是一个规模较小的掩码语言模型，负责将掩码后的文本恢复原状。而

<sup>11</sup><https://dumps.wikimedia.org>

判别器（Discriminator）则使用替换词检测（Replaced Token Detection, RTD）预训练任务，负责检测生成器输出的内容中的每个 Token 是否是原文中的内容。其完整的流程如图2.8所示。

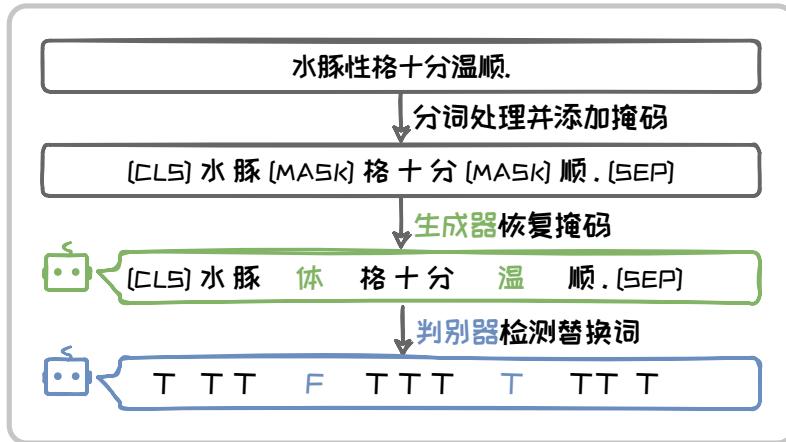


图 2.8: ELECTRA 预训练任务

根据生成器与判别器的不同规模，ELECTRA 共提出了三个版本的模型，分别是 ELECTRA-Small、ELECTRA-Base 以及 ELECTRA-Large。其中 **ELECTRA-Small** 中的生成器与判别器都由 12 个编码模块堆叠而成，隐藏层维度为 256，自注意力头数量为 4，因此生成器与判别器的参数量均为 0.14 亿左右，**总参数数量约为 0.28 亿**；**ELECTRA-Base** 中的生成器与判别器都由 12 个编码模块堆叠而成，隐藏层维度为 768，自注意力头数量为 12，因此生成器与判别器的参数量均为 1.1 亿左右，**总参数数量约为 2.2 亿**；**ELECTRA-Large** 中的生成器与判别器都由 24 个编码模块堆叠而成，隐藏层维度为 1024，自注意力头数量为 16，因此生成器与判别器的参数量均为 3.3 亿左右，**总参数数量约为 6.6 亿**。

其中，ELECTRA-Small 和 ELECTRA-Base 使用与 BERT 一致的数据集来进行预训练，共包含 33 亿个 Token。而 ELECTRA-Large 则使用了**更多样化的训练数据**，包括大规模网页数据集 ClueWeb<sup>12</sup>、CommonCrawl<sup>13</sup>以及大型新闻文本数据集

<sup>12</sup><https://lemurproject.org/clueweb09.php>

<sup>13</sup><https://commoncrawl.org>

Gigaword<sup>14</sup>，最终将数据量扩充到了 330 亿个 Token，从而帮助模型学习更广泛的语言表示。

另外，在 BERT 中，只有 15% 的固定比例 Token 被掩码，模型训练的内容也仅限于这 15% 的 Token。但是在 ELECTRA 中，判别器会判断生成器输出的所有 Token 是否被替换过，因此能够更好地学习文本的上下文嵌入。

不同于 RoBERTa 和 ALBERT 主要在模型结构以及预训练数据规模上进行优化，ELECTRA 在 BERT 的基础上引入了新的生成器-判别器架构，通过替换语言模型任务，显著提升了训练效率和效果，同时提高了模型在下游任务中的表现。

### 2.3.4 Encoder-only 模型架构对比

以 BERT 及其诸多衍生模型为代表的 Encoder-only 架构模型通过各自独特的创新和优化，在特定任务和应用场景中展现了卓越的性能。表2.1展示了本章提到的一些典型代表模型，它们不仅能够为文本提供高质量的上下文嵌入，还在自然语言处理的多个任务中取得了显著的成果，包括但不限于文本分类、情感分析等。然而，这些模型仍然专注于理解文本层面，难以应对生成式任务，因此在当前愈发热门的生成式人工智能（Artificial Intelligence Generated Content, AIGC）领域中发挥的作用有限。

表 2.1: Encoder-only 架构代表模型参数和语料大小表

模型	发布时间	参数量 (亿)	语料规模	预训练任务
BERT	2018.10	1.1, 3.4	约 15GB	MLM+NSP
RoBERTa	2019.07	1.2, 3.5	160GB	Dynamic MLM
ALBERT	2019.09	0.12, 0.18, 0.6, 2.2	约 15GB	MLM+SOP
ELECTRA	2020.03	0.28, 2.2, 6.6	约 20-200GB	RTD

<sup>14</sup><https://catalog.ldc.upenn.edu/LDC2011T07>

## 2.4 基于 Encoder-Decoder 架构的大语言模型

Encoder-only 架构得益于其双向注意力机制，在自然语言理解任务中表现出卓越的性能。然而，这种架构在处理序列到序列（Sequence to Sequence, Seq2Seq）任务，例如机器翻译时，存在一定的局限性。原因在于，Encoder-only 架构缺少专门的机制来处理输出序列的生成过程，它只输出固定长度的上下文向量，虽然能够有效编码输入序列的信息，但难以直接生成所需的不固定长度的文本序列。

为了克服这一限制，众多模型引入了 Decoder 组件，其作用是基于编码器提供的上下文信息来预测和生成目标序列。通过结合 Encoder 和 Decoder，形成了一个完整的 Encoder-Decoder 架构，这种架构能够基于输入序列生成适当长度且上下文相关的输出序列，适合处理 Seq2Seq 任务。

### 2.4.1 Encoder-Decoder 架构

Encoder-Decoder 架构如图2.9所示，其主要由两个部分组成，分别是编码器和解码器。

其中编码器部分与 Encoder-only 架构中的编码器一致，由多个编码模块堆叠而成，每个编码模块包含一个自注意力模块以及一个全连接前馈模块。模型的输入序列在通过编码器部分后会被转变为固定大小的上下文向量，这个向量包含了输入序列的语义信息。解码器则是由多个解码模块堆叠而成，每个解码模块与编码模块之间有着一定的类似之处，但也更为复杂。一方面，解码模块中的自注意力模块额外添加了注意力掩码；另一方面，解码模块在自注意力模块与全连接前馈模块之间，添加了一个交叉注意力模块与编码器进行交互。下面分别对带掩码的自注意力模块以及交叉注意力模块展开介绍。

#### 1. 带掩码的自注意力模块

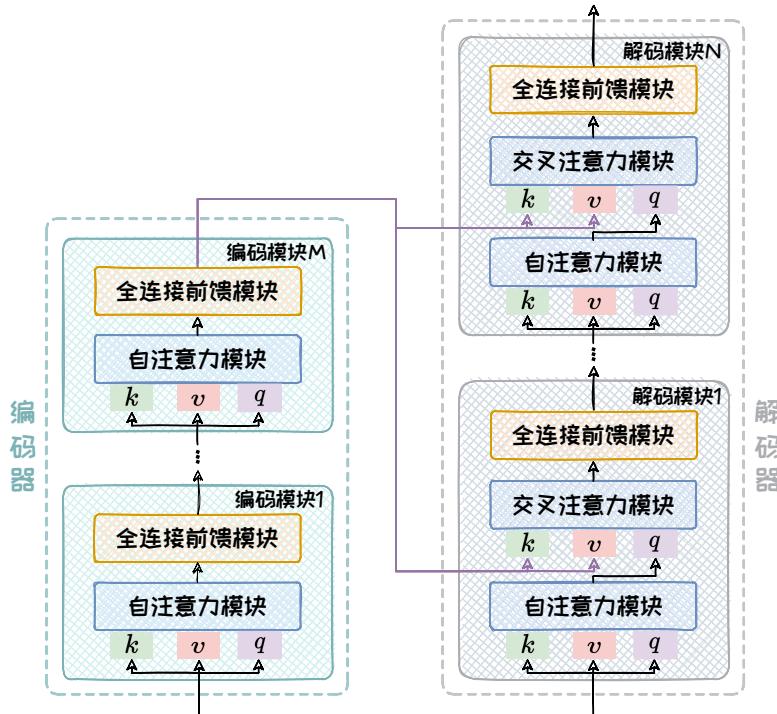


图 2.9: Encoder-Decoder 架构

通常，编码器主要用于编码输入序列的信息，而输入序列是完全已知的，双向注意力机制使得每个编码模块都可以在同一个时间步内并行处理整个输入序列，并输出其上下文表示。然而，在解码器部分情况则有所不同。在推理过程中，解码器采用自回归的生成方式，这意味着在每个时间步生成当前 Token 时，需要依赖于之前所有时间步已经生成的 Token，未来的 Token 由于尚未被推理所以无法被访问。但在训练过程中，解码器通常采用 **Teacher-Forcing 策略**，将先前真实的 Token 作为当前时间步的输入，而不是使用模型生成的先前输出。由于真实输出序列在训练时是已知的，解码器可以并行处理整个输出序列。

为了确保训练过程与推理过程的一致性，并模拟自回归生成的行为，需要使用掩码机制来屏蔽未来的 Token。掩码机制的作用是保证每个位置的自注意力仅限于当前及之前的 Token，防止模型在计算当前时间步的注意力时看到未来时间步的信息。因此，解码模块中的自注意力模块是带掩码的单向自注意力，在注意力矩阵

中呈现出下三角的形式。

### 2. 交叉注意力模块

交叉注意力模块扮演着至关重要的角色，专门负责处理与编码器输出之间的依赖关系。该模块使得在解码过程中可以动态地从编码器输出的上下文向量中提取相关信息，确保生成的输出与原始输入内容紧密相连。

具体来说，交叉注意力与自注意力有着显著的区别。自注意力主要用于编码器或者解码器内部，处理同一序列中元素间的依赖关系。在编码器中，自注意力机制允许每个输入元素关注整个输入序列，从而捕捉全面的上下文信息。而在解码器中，自注意力机制则聚焦于已生成序列内部的元素关系，通过掩码机制确保每一步都只能关注先前的输出，从而避免未来信息泄露。在该自注意力模块中，输入序列中的每个元素，通过线性变换生成查询 (query)，键 (key) 以及值 (value) 三部分，用于处理元素之间的依赖关系。

相比之下，交叉注意力机制则是解码器特有的机制，允许解码器在生成每个 Token 时，充分利用编码器提供的全局上下文信息。在交叉注意力模块中，查询 (query) 来自于解码器自注意力输出，而键 (key) 和值 (value) 来自于编码器中最后一个编码模块输出的上下文向量。这意味着，解码器在生成序列的过程中，可以通过交叉注意力机制动态地聚焦于输入序列中与当前需要生成的 Token 最相关的部分。

通过自注意力和交叉注意力机制的结合，Encoder-Decoder 架构能够**高效地编码输入信息并生成高质量的输出序列**。这两个机制相辅相成，共同保障了模型在处理复杂任务时的表现。自注意力机制确保了输入序列和生成序列内部的一致性和连贯性，而交叉注意力机制则确保了解码器在生成每一个输出时都能参考输入序列的全局信息，从而生成与输入内容高度相关的结果。在这两个机制的共同作用下，Encoder-Decoder 架构能够充分理解输入序列的内容，并根据不同的任务需

求生成长度灵活的输出序列，在机器翻译、文本摘要、问答系统等任务中得到了广泛应用，并取得了显著成效。

本节将介绍两种典型的 Encoder-Decoder 架构的大语言模型，分别是 T5[31] 和 BART[19]。

### 2.4.2 T5 语言模型

在自然语言处理领域，任务种类繁多，涵盖了语言翻译、文本摘要、问答系统等。这些任务要求模型根据特定需求进行定制设计，并依赖于定制化的数据处理和训练策略。这种方法虽然在特定任务上表现良好，但在模型迁移和泛化上存在局限，限制了模型在不同任务间的复用。为了解决这一问题，Google Research 团队在 2019 年 10 月提出了 T5 (Text-to-Text Transfer Transformer) 模型。T5 是一种基于 Encoder-Decoder 架构的大型预训练语言模型，采用了统一的文本到文本的转换范式。下面分别从模型结构、预训练方式以及下游任务角度三个方面对 T5 模型进行介绍。

#### 1. T5 模型结构

T5 模型的主要创新之处在于其将多种自然语言处理任务统一到了同一个文本到文本的转换框架中。先前的 BERT 以及 GPT-1 等语言模型需要针对具体的下游任务分配不同的模型进行训练，而 T5 通过统一框架可以根据前缀指示来自适应地适配不同任务，从而生成相应的输出文本，如图2.10所示。

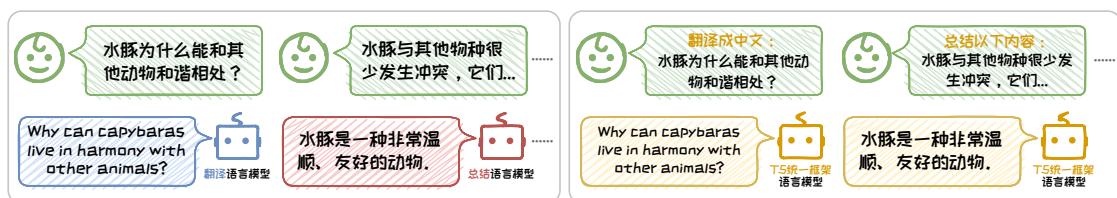


图 2.10: 传统语言模型和 T5 统一框架

这种方法使得 T5 模型能够识别不同的任务类型，并执行相应的迁移学习，这

在当时可以视为早期的提示（Prompt）技术的运用。通过这种方式，T5 模型不仅提高了模型的灵活性和泛化能力，也为后续的 NLP 研究和应用提供了新的思路。

在模型本身的架构上，T5 与标准的 Transformer 架构类似，包括一个编码器和一个解码器。每个编码器和解码器又分别由多个编码模块和解码模块堆叠而成。根据不同的具体参数，T5 模型一共有五个版本，分别是 T5-Small、T5-Base、T5-Large、T5-3B 以及 T5-11B。**T5-Small** 由 6 个编码模块和 6 个解码模块堆叠而成，其中隐藏层维度为 512，自注意力头的数量为 8，**总参数数量约为 6000 万**；**T5-Base** 与 BERT-Base 对标，由 12 个编码模块和 12 个解码模块堆叠而成，其中隐藏层维度为 768，自注意力头的数量为 12，**总参数数量约为 2.2 亿**；**T5-Large** 与 BERT-Large 对标，由 24 个编码模块和 24 个解码模块堆叠而成，其中隐藏层维度为 1024，自注意力头的数量为 16，**总参数数量约为 7.7 亿**；**T5-3B** 在 T5-Large 的基础上，把自注意力头的数量扩大到 32，另外还将编码模块以及解码模块中全连接前馈网络的中间层维度扩大了 4 倍，**总参数数量约为 28 亿**；**T5-11B** 在 T5-3B 的基础上，进一步将自注意力头的数量扩大到 128，并再一次将全连接前馈网络的中间层维度扩大了 4 倍，**总参数数量约为 110 亿**。

### 2. T5 预训练方式

为了获取高质量、覆盖范围广泛的数据集，Google Research 团队从大规模网页数据集 Common Crawl<sup>15</sup> 中提取了大量的网页数据，并经过严格的清理和过滤，最终生成了 C4 数据集（Colossal Clean Crawled Corpus），其覆盖了各种网站和文本类型，总规模达到了约 750GB。

在预训练任务方面，T5 借鉴了 BERT[10] 以及 SpanBERT[16]，提出了名为 Span Corruption 的预训练任务。从原始输入中选择 15% 的 Token 进行破坏，每次都选择连续三个 Token 作为一个小段（span）整体被遮挡成 [MASK]。与 BERT 模型中采

---

<sup>15</sup><https://commoncrawl.org>

用的单个 Token 预测不同，T5 模型需要对整个被遮挡的文本片段进行预测。这些片段可能包括连续的短语或子句，它们在自然语言中构成了具有完整意义的语义单元。这一设计要求模型不仅要理解局部词汇的表面形式，还要捕捉更深层次的句子结构和上下文之间的复杂依赖关系。

Span Corruption 预训练任务显著提升了 T5 模型在处理各种自然语言处理任务时的表现，尤其是在需要生成连贯和逻辑性强的文本任务中。例如，在文本摘要、问答系统和文本补全等任务中，T5 模型能够生成更加流畅和准确的输出。

### 3. T5 下游任务

基于预训练阶段学到的大量知识以及新提出的文本到文本的统一框架，T5 模型可以将具体的下游任务转换为合适的输入格式，从而在零样本 (Zero-Shot) 的情况下，利用 Prompt 工程技术直接适配于多种下游任务。同时，T5 模型也可以进一步通过微调 (Fine-Tuning) 来适用于特定的任务，这一过程需要针对下游任务收集带标签训练数据，同时也需要更多的计算资源和训练时间。对于那些对精度要求极高且任务本身较为复杂的应用场景，进一步微调是提高 T5 模型性能的有效途径。

综合来看，T5 模型通过精心设计的预训练策略和先进的 Prompt 工程技术，在自然语言处理领域实现了显著的性能提升。其文本到文本的统一框架不仅简化了不同任务之间的转换流程，也为自然语言处理领域未来的研究提供了新方向。现如今，T5 模型已经衍生了许多变体，以满足不同任务的需求。例如，mT5[44] 模型扩展了对 100 多种语言的支持，T0[33] 模型通过多任务训练增强了零样本学习 (Zero-Shot Learning) 能力，Flan-T5[7] 模型专注于指令微调，进一步提升了模型的灵活性和效率等等。

### 2.4.3 BART 模型

BART (Bidirectional and Auto-Regressive Transformers) 是由 Meta AI 研究院同样于 2019 年 10 月提出的一个 Encoder-Decoder 架构模型。不同于 T5 那般将多种自然语言处理任务集成到一个统一的框架, BART 旨在通过多样化的预训练任务来提升模型在文本生成任务以及文本理解任务上的表现。

#### 1. BART 模型结构

BART 的模型结构与标准的 Transformer 架构相似, 包括一个编码器和一个解码器。每个编码器和解码器分别由多个编码模块和解码模块堆叠而成。BART 模型一共有两个版本, 分别是 BART-Base 以及 BART-Large。BART-Base 由 6 个编码模块和 6 个解码模块堆叠而成, 其中隐藏层维度为 768, 自注意力头的数量为 12, **总参数数量约为 1.4 亿**; BART-Large 由 12 个编码模块和 12 个解码模块堆叠而成, 其中隐藏层维度为 1024, 自注意力头的数量为 16, **总参数数量约为 4 亿**。

#### 2. BART 预训练方式

BART 的预训练使用了与 RoBERTa[24] 相同的语料库, 包含小说数据集 Book-Corpus[49]、英语维基百科数据集<sup>16</sup>、新闻数据集 CC-News<sup>17</sup>、网页开放数据集 OpenWebText<sup>18</sup>以及故事数据集 Stories, 总数据量达到约 160GB。在预训练任务上, BART 设计了多样化的任务来增强模型的能力。相比于先前的诸多模型, BART 的任务设计更为具体且丰富, 旨在训练模型从受损文本中恢复原始信息。例如在图2.11中, 在给定原始文本“水豚性格十分温顺. 且无攻击性.”的前提下, BART 设计了以下任务:

- **Token 遮挡任务 (Token Masking):** 类似于 BERT 中的 MLM 任务, 在原始文

---

<sup>16</sup><https://dumps.wikimedia.org>

<sup>17</sup><http://web.archive.org/save/http://commoncrawl.org/2016/10/newsdataset-available>

<sup>18</sup><http://Skylion007.github.io/OpenWebTextCorpus>

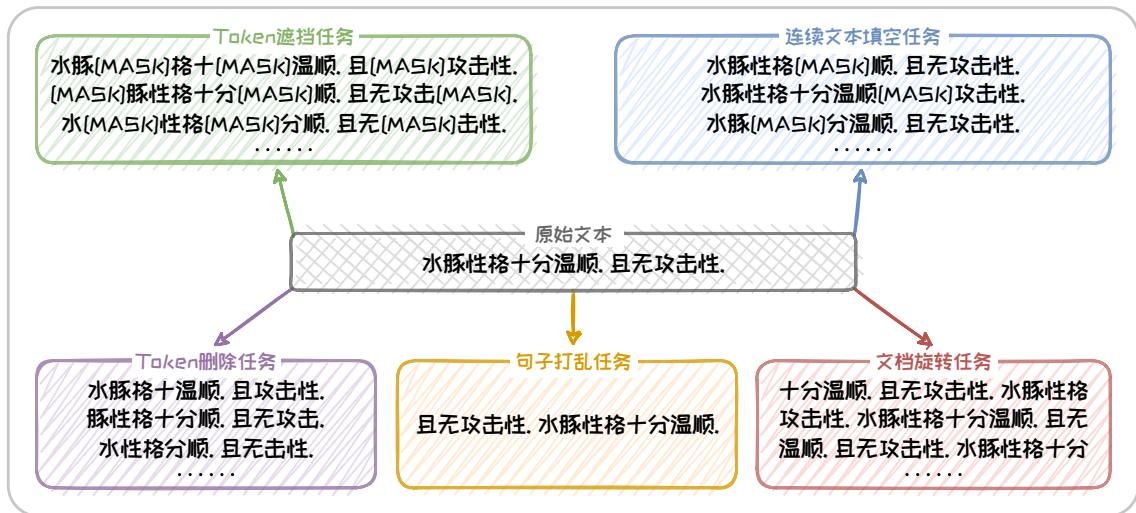


图 2.11: BART 预训练任务

本中随机采样一部分 Token 并将其替换为 [MASK]，从而训练模型推断被删除的 Token 内容的能力。

- **Token 删除任务** (Token Deletion): 在原始文本中随机删除一部分 Token，从而训练模型推断被删除的 Token 位置以及内容的能力。
- **连续文本填空任务** (Text Infilling): 类似于 T5 以及 spanBERT[16] 中的预训练任务，在原始文本中选择几段连续的 Token (每段作为一个 span)，整体替换为 [MASK]。其中 span 的长度服从  $\lambda = 3$  的泊松分布，如果长度为 0 则直接插入一个 [MASK]。这一任务旨在训练模型推断一段 span 及其长度的能力。
- **句子打乱任务** (Sentence Permutation): 将给定文本拆分为多个句子，并随机打乱句子的顺序。旨在训练模型推理前后句关系的能力。
- **文档旋转任务** (Document Rotation): 从给定文本中随机选取一个 Token，作为文本新的开头进行旋转。旨在训练模型找到文本合理起始点的能力。

在 BART 的预训练阶段，模型的目标是重建未被破坏的原始文本。这意味着，尽管输入文本被故意以多种方式破坏（单独使用上述某种策略或是组合使用），

BART 的任务是从这些噪声中恢复出原始、完整的句子。这种去噪自编码器的方法不仅锻炼了模型对文本结构和语义的深入理解，还增强了其在面对不完整或损坏信息时的鲁棒性。

通过针对具体任务的需求进行调整，BART 模型能够将在预训练阶段学到的语言知识迁移到具体的应用场景中，从而在多种自然语言处理任务上实现高性能。这种从预训练到微调的流程，使得 BART 不仅在文本生成任务上表现出色，也能够适应文本理解类任务的挑战。后续同样也出现了 BART 模型的各种变体，包括可处理跨语言文本生成任务的 mBART[23] 模型等。

### 2.4.4 Encoder-Decoder 架构模型对比

基于 Encoder-Decoder 架构的大语言模型，通过其灵活的双向编码器和自回归解码器设计，在自然语言处理的生成任务中展示了强大的能力。表2.2展示了本章提到的一些典型代表模型，这些模型不仅能够理解输入文本的复杂语义，还能生成连贯且上下文相关的输出。通过不同的预训练策略和优化，这些模型在翻译、摘要、问答等任务中取得了显著成果，并展示了其在自然语言处理领域的广泛应用前景。

表 2.2: Encoder-Decoder 架构代表模型参数和语料大小表

模型	发布时间	参数量 (亿)	语料规模
T5	2019.10	0.6-110 亿	750GB
mT5	2020.10	3-130 亿	9.7TB
T0	2021.10	30-110 亿	约 400GB
BART	2019.10	1.4-4 亿	约 20GB
mBART	2020.06	0.4-6.1 亿	约 1TB

## 2.5 基于 Decoder-only 架构的大语言模型

在开放式 (Open-Ended) 生成任务中，通常输入序列较为简单，甚至没有具体明确的输入，因此维持一个完整的编码器来处理这些输入并不是必要的。对于这种任务，Encoder-Decoder 架构可能显得过于复杂且缺乏灵活性。

在这种背景下，Decoder-only 架构表现得更为优异。它通过自回归方法逐字生成文本，不仅保持了长文本的连贯性和内在一致性，而且在缺乏明确输入或者复杂输入的情况下，能够更自然、流畅地生成文本。此外，Decoder-only 架构由于去除了编码器部分，使得模型更加轻量化，从而加快了训练和推理的速度。因此，在同样的模型规模下，Decoder-only 架构可能表现得更为出色。

值得一提的是，Decoder-only 架构模型的概念最早可以追溯到 2018 年发布的 GPT-1[29] 模型。但在当时，由于以 BERT 为代表的 Encoder-only 架构模型在各项任务中展现出的卓越性能，Decoder-only 架构并没有受到足够的关注。直到 2020 年，GPT-3[4] 的突破性成功，使得 Decoder-only 架构开始被广泛应用于各种大语言模型中，其中最为流行的有 OpenAI 提出的 GPT 系列、Meta 提出的 LLaMA 系列等。其中，GPT 系列是起步最早的 Decoder-only 架构，在性能上也成为了时代的标杆。但从第三代开始，GPT 系列逐渐走向了闭源。而 LLaMA 系列虽然起步较晚，但凭借着同样出色的性能以及始终坚持的开源道路，也在 Decoder-only 架构领域占据了一席之地。接下来将对这两种系列的模型进行介绍。

### 2.5.1 GPT 系列模型

GPT (Generative Pre-trained Transformer) 系列是由 OpenAI 开发的一组语言模型，它们基于 Decoder-only 架构，专注于自然语言生成任务。通过在海量文本数据上的预训练，GPT 模型掌握了语言的深层结构和模式，然后通过在特定任务上的

微调或直接应用，展现出卓越的性能。

自从 2018 年问世以来，GPT 系列已经经历了近六年的快速发展，其演进历程可以划分为五个显著的发展阶段。正如表2.3所示，每一阶段都展示了在参数规模、预训练语料规模以及语言生成能力上的变化。其中参数规模与预训练语料规模的同步激增，是 GPT 系列升级的最大主线。

表 2.3: GPT 系列模型参数和语料大小表

模型	发布时间	参数量 (亿)	语料规模
GPT-1	2018.06	1.17	约 5GB
GPT-2	2019.02	1.24 / 3.55 / 7.74 / 15	40GB
GPT-3	2020.05	1.25 / 3.5 / 7.62 / 13 / 27 / 67 / 130 / 1750	1TB
ChatGPT	2022.11	未知	未知
GPT-4	2023.03	未知	未知
GPT-4o	2024.05	未知	未知

### 1. 起步阶段：GPT-1 模型

OpenAI 的前首席科学家 Ilya Sutskever 在采访中<sup>19</sup>指出，OpenAI 自成立初期就开始研究如何利用神经网络通过自监督的方式预测文本的下文，但受限于当时 RNN 的长距离依赖问题。而 Transformer 架构的出现很好地解决了这一问题。因此，OpenAI 尝试使用 Transformer 架构替换了先前的 RNN，开始研发语言模型。

2018 年 6 月，OpenAI 发布了第一个版本的 GPT (Generative Pre-Training) 模型，被称为 GPT-1<sup>[29]</sup>。GPT-1 开创了 Decoder-only 架构自回归生成文本的先河，是自然语言处理领域的一个重要里程碑。

#### (1) GPT-1 模型结构

在模型架构方面，GPT-1 使用了 Transformer 架构中的 Decoder 部分，省略了 Encoder 部分以及交叉注意力模块。其模型由 12 个解码块堆叠而成，每个解码块包含一个带掩码的自注意力模块和一个全连接前馈模块。其中隐藏层维度为 768，

<sup>19</sup><https://hackernoon.com/an-interview-with-ilya-sutskever-co-founder-of-openai>

自注意力头的数量为 12，模型的最终参数数量约为 1.17 亿。

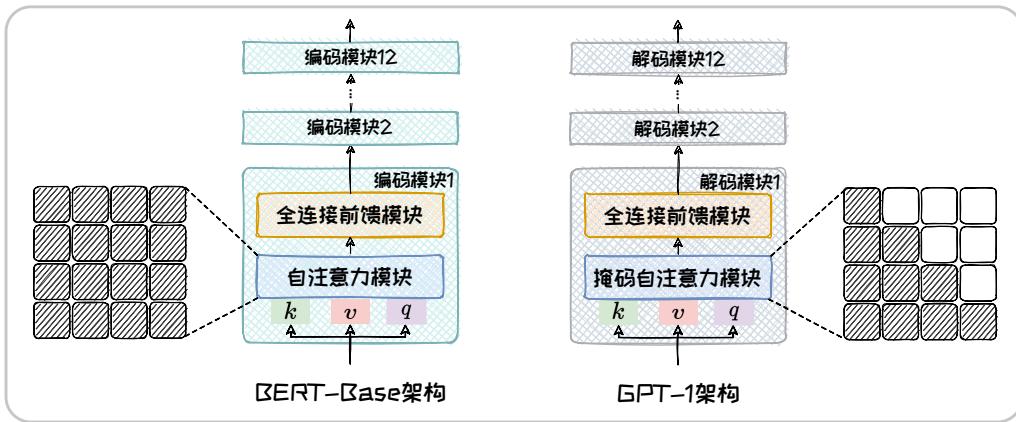


图 2.12: BERT-Base 和 GPT-1 模型

图 2.12 展示了 BERT-Base 以及 GPT-1 的模型结构。从图中可以看出，GPT-1 在结构上与 BERT-Base 具有显著的相似性，两者都包含 12 个块，每个块也同样由一个自注意力模块和一个全连接前馈模块组成。两者之间的本质区别在于 BERT-Base 中的自注意力模块是双向的自注意力机制，而 GPT-1 中的自注意力模块则是 **单向且带有掩码的自注意力机制**。

### (2) GPT-1 预训练方法

GPT-1 使用小说数据集 BookCorpus[49] 来进行预训练，该数据集包含约 8 亿个 Token，总数据量接近 5GB。在预训练方式上，GPT-1 采用无监督的语言建模任务，即基于已有的文本以自回归的方式预测下一个 Token。这一过程类似于 **开放式文本续写**，模型需要基于前文的语境，**推断并生成合适的后续文本**，如图 2.13 所示。通过这种预训练，模型可以在不需要人为构造大量带标签数据的前提下，学习到大量语言的“常识”，学会生成连贯且上下文相关的文本。

### (3) GPT-1 下游任务

经过预训练阶段的广泛语言知识学习后，GPT-1 模型为了适应特定的应用场景和任务需求，需要进行有监督的微调。这一过程涉及使用针对特定任务的标注数据集，对模型的参数进行精细调整。完成微调阶段后，GPT-1 模型便能够应用于

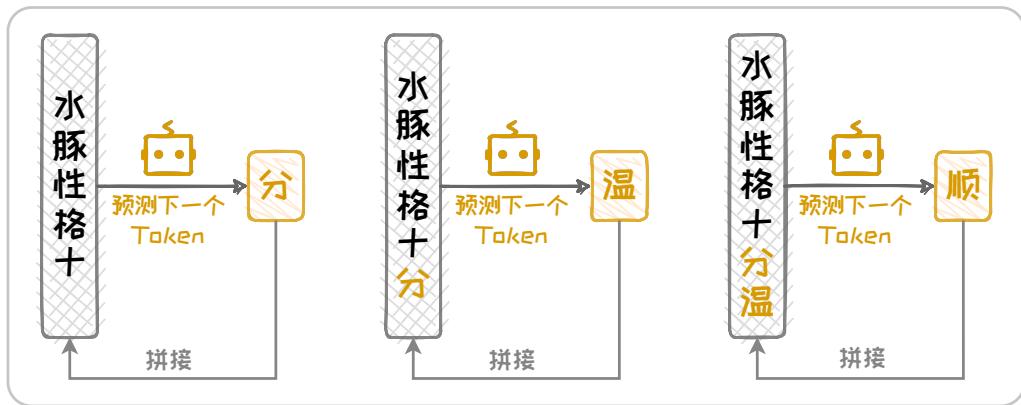


图 2.13: GPT-1 语言建模预训练任务

各类任务，包括但不限于：

- **文本分类**: GPT-1 能够接收一段文本作为输入，并根据预定义的类别标签，如情感倾向（积极、消极或其他），对文本进行分类。这在情感分析、主题分类等场景中非常有用。
- **文本相似度评估**: 当需要衡量两段文本之间的相似性时，GPT-1 能够分析并量化它们的内容和语义相似度。这项功能在比较文档、搜索结果优化和推荐系统中尤为重要。
- **多项选择题解答**: GPT-1 还可以处理多项选择问题。模型能够理解问题文本和选项内容，从给定的选项中识别并选择最合适答案。

作为一个生成模型，GPT-1 在具体任务中，可以直接以文本序列的形式输出结果，从而提供灵活且强大的文本生成能力。然而，由于其单向注意力机制的限制，GPT-1 在处理需要全面上下文理解的任务时表现相对不足。正是这些局限性，促使研究人员在同时期发布了双向编码模型 BERT，以克服这些挑战并提供更优的性能。

#### (4) GPT-1 vs BERT

GPT-1 和 BERT[10] 模型都是预训练语言模型的先驱，它们几乎同时期发布，并且都在自然语言处理领域的发展中占据了重要地位。GPT-1 通过其 Decoder-only

架构，展示了在文本生成任务中的潜力。而 BERT 则以其 Encoder-only 架构和双向上下文理解能力，为语言模型的应用提供了新的方法。尽管 BERT 比 GPT-1 晚推出大约四个月，但其创新的预训练方法和架构设计，使其在发布后迅速成为自然语言处理研究和应用的热门选择。

为了更好地进行对比，BERT-Base 版本在模型设计上采用了与 GPT-1 非常相似的设置，包括隐藏层维度、自注意力头的数量以及模型最终参数量等。但由于 BERT-Base 在预训练预料的规模上比 GPT-1 大了两倍，因此在多项性能指标上超越了 GPT-1，尤其是在需要全面上下文理解的任务中。此外，BERT 的 Large 版本，通过进一步将模型规模扩大了两倍，其性能更是显著提升，将语言模型的能力推向了新的高度。

GPT-1 的提出，标志着 OpenAI 以及 Decoder-only 架构在语言模型上的正式起步，尽管其在参数量、性能和影响力方面与同时期的 BERT 存在一定差距，但它在文本生成领域的贡献和启发作用不容忽视。GPT-1 的创新和尝试，为自然语言处理技术的进步提供了宝贵的经验和启示，也为后续 OpenAI 研发更大规模模型奠定了基础。但在另一方面，GPT-1 仍需要一定数量的带标签样本来训练对应的任务，因此会受限于带标签数据稀缺的情况。

## 2. 提升阶段：GPT-2 模型

虽然初期经历了一些挫折，但 OpenAI 并没有改变其 Decoder-only 的技术路线，而是选择在这一策略上持续深耕，致力于构建规模更大、性能更优的模型。随后在 2019 年 2 月，OpenAI 发布了 GPT-2<sup>20</sup>，作为 GPT 系列的第二代产品。与前代 GPT-1 相比，GPT-2 在模型规模和性能上都有了显著的提升和改进，能够执行更复杂和多样化的自然语言处理任务。另外，GPT-2 还在一定程度上缓解了对带标签数据的依赖问题。

<sup>20</sup><https://openai.com/index/gpt-2-1-5b-release>

### (1) GPT-2 模型结构

GPT-2 模型延续了上一代 GPT-1 的 Decoder-only 架构，并在此基础上进行了规模和复杂度的显著扩展。GPT-2 一共发布了四个版本，分别是 GPT-2 Small、GPT-2 Medium、GPT-2 Large 以及 GPT-2 XL。其中 **GPT-2 Small** 在架构上接近 GPT-1 以及 BERT-Base，由 12 个编码块堆叠而成，其中隐藏层维度为 768，自注意力头的数量为 12，**总参数数量约为 1.24 亿**；**GPT-2 Medium** 在架构上接近 BERT-Large，由 24 个解码块堆叠而成，其中隐藏层维度为 1024，自注意力头的数量为 16，**总参数数量约为 3.55 亿**；**GPT-2 Large** 由 36 个解码块堆叠而成，其中隐藏层维度为 1280，自注意力头的数量为 20，**总参数数量约为 7.74 亿**；**GPT-2 XL** 是最大规模版本，由 48 个解码块堆叠而成，其中隐藏层维度为 1600，自注意力头的数量为 25，**总参数数量约为 15 亿**。

### (2) GPT-2 预训练方法

在预训练任务上，GPT-2 继续采用无监督的语言建模方法，但使用了更高质量的预训练数据集。具体来说，GPT-2 采用了全新的 WebText 数据集，该数据集通过精心筛选和清洗网络文本而成，总数据量达到了 40GB，确保了数据的质量和多样性。通过使用 WebText 数据集进行预训练，GPT-2 的语言理解力得到了显著增强。与 GPT-1 相比，GPT-2 不仅接触到了更多样化的语言使用场景，还学习到了更复杂的语言表达方式。这种改进使得 GPT-2 在捕捉语言细微差别和构建语言模型方面更为精准，从而在执行各种自然语言处理任务时能够生成更准确、更连贯的文本。

### (3) GPT-2 下游任务

GPT-2 在继承 GPT-1 强大文本生成能力的同时，得益于其更庞大的模型规模和更丰富的训练数据，展现出了一定的**零样本学习能力**。这意味着 GPT-2 能够在没有接受特定任务训练的情况下，直接执行各类任务，使其能够灵活适应多变的应用场景和任务需求，不再高度依赖于繁琐的微调过程，并且大大**减少了对带标注数据的依赖**。

签数据的依赖。

总的来说，GPT-2 在 GPT-1 的基础上，不仅在模型规模上实现了数量级的增长，性能也实现了跨越式的提升，在自然语言处理任务中取得了显著的成果。此外，GPT-2 还引出了零样本学习的新范式，在带标签数据稀缺的情况下也能体现出较好的性能，虽然在具体应用时可能会产生一些不准确或低质量的输出，但也为后续 GPT 系列的进一步探索奠定了坚实的基础。

### 3. 突破阶段：GPT-3 模型

为了在 GPT-2 的基础上进一步挖掘模型在数据稀缺情况下的能力，OpenAI 于 2020 年 6 月推出了第三代模型 GPT-3[4]，标志着大语言模型的发展进入了一个新高点。与前两代模型相比，GPT-3 在多个关键维度实现了突破，并且还新提出了单样本（One-Shot）和少样本（Few-Shot）学习的能力，使其在广泛的自然语言处理任务中展现出卓越的性能。

#### (1) GPT-3 模型架构

在模型层面，GPT-3 继承并扩展了前两代的架构，参数量得到了进一步的提升，最大版本的参数量达到了惊人的 1750 亿。这一前所未有的参数量使得 GPT-3 能够捕获更加细微和复杂的语言模式，显著提升了模型的性能和表达能力。GPT-3 同样发布了多个版本，参数量逐级增加，以适应不同规模的应用需求。与前两代相比，GPT-3 无论是解码块数量、隐藏层维度还是自注意力头的数量，都有了一定的扩展。表2.4展示了 GPT-3 以及两个前代模型不同版本的具体参数，从中可以看出 GPT-3 在总参数量上的显著提升。

#### (2) GPT-3 预训练方法

GPT-3 的预训练继续采用无监督的语言建模任务，但这次使用了更大规模和更多样化的互联网文本数据集。其预训练数据量接近 1TB，涵盖了 Common Crawl<sup>21</sup>、

<sup>21</sup><https://commoncrawl.org>

表 2.4: GPT-1 至 GPT-3 模型具体参数表

模型版本	解码块数量	隐藏层维度	自注意力头数量	总参数量 (亿)
GPT-1	12	768	12	1.17
GPT-2 Small	12	768	12	1.24
GPT-2 Medium	24	1024	16	3.55
GPT-2 Large	36	1280	20	7.74
GPT-2 XL	48	1600	36	15
GPT-3 Small	12	768	12	1.25
GPT-3 Medium	24	1024	16	3.5
GPT-3 Large	24	1536	16	7.62
GPT-3 XL	24	2048	24	13
GPT-3 2.7B	32	2560	32	27
GPT-3 6.7B	32	4096	32	67
GPT-3 13B	40	5120	40	130
GPT-3 175B	96	12288	96	1750

WebText、BookCorpus<sup>49</sup>、Wikipedia<sup>22</sup>等多个来源，包括书籍、网站、论坛帖子等各类文本形式。这些数据经过了严格的筛选和清洗，以确保数据的质量和多样性，让 GPT-3 模型能够学习到更加丰富和多元的语言知识和世界知识。

### (3) GPT-3 下游任务

凭借预训练阶段学到的丰富知识，GPT-3 模型展现了卓越的上下文学习（In-Context Learning, ICL）能力。这种能力允许 GPT-3 在无需传统微调的情况下，即可通过少样本学习（Few-Shot Learning）、单样本学习（One-Shot Learning）或零样本学习（Zero-Shot Learning）快速适应特定任务。在少样本学习中，模型仅需提供几个该特定任务下的示例即可快速掌握任务特点；在单样本学习中，模型只需要一个示例即可理解任务要求；而在零样本学习中，模型甚至无需提供示例即可理解并执行任务。

这些能力使得 GPT-3 具有高度的可迁移性，能够在不同场景和任务中快速适应并表现出色。这样的灵活性和适应性，使得 GPT-3 在自然语言处理的应用中拥

<sup>22</sup><http://web.archive.org/save/http://commoncrawl.org/2016/10/newsdataset-available>

有广泛的前景，无论是文本生成、问答系统还是语言翻译，均能以较少的额外训练达到优异的效果。上下文学习相关的知识会在3.2章节详细介绍。

总的来说，GPT-3 展示了强大的泛化能力，能在多种语言任务上达到或接近人类水平的表现。GPT-3 的发布，不仅是 OpenAI 在语言模型领域的一次巨大突破，也为整个人工智能社区提供了宝贵的资源和启示。其强大的灵活性和适应性，预示着人工智能在理解和生成自然语言方面的巨大发展前景。

#### 4. 能力扩展：InstructGPT 等模型

在 GPT-3 的基础上，OpenAI 在多个维度进行了深入探索，并推出了一系列衍生模型，致力于扩展模型的能力和应用边界，并为未来的研究奠定基石。其中，2022 年初亮相的 InstructGPT[27] 模型以卓越的指令执行能力而备受瞩目。这一强大的指令执行能力得益于 OpenAI 引入了**人类反馈强化学习**（Reinforcement Learning from Human Feedback, RLHF）这一创新训练方法，显著增强了模型对指令的响应质量。

##### (1) 人类反馈强化学习 RLHF

InstructGPT 的研究重点在于缓解模型在遵循用户指令时可能出现的不准确性和不可靠性。通过人类反馈强化学习，模型生成的输出能够更符合人类的目标需求。这个过程涉及让人类评估者提供关于模型输出质量的反馈，然后使用这些反馈来指导模型的训练。具体的过程如图2.14所示，整体可以分为以下三个步骤：

- 1. 有监督微调：**收集大量带标签数据集。每个样本中包含输入以及对应人工标注的输出，用于**有监督地微调语言模型**，确保模型具备基本的任务执行能力和更高的输出质量。
- 2. 训练奖励模型：**针对每个输入，让模型生成多个候选输出，并由人工对其进行质量评估和排名，构成偏好数据集。每个样本中包含输入、模型生成的多种候选输出以及人工标注的不同输出的偏好顺序。这些样本用于**训练一个奖励模型**，使其能够学习人类对不同输出的偏好，从而用于评估语言模型的输出。

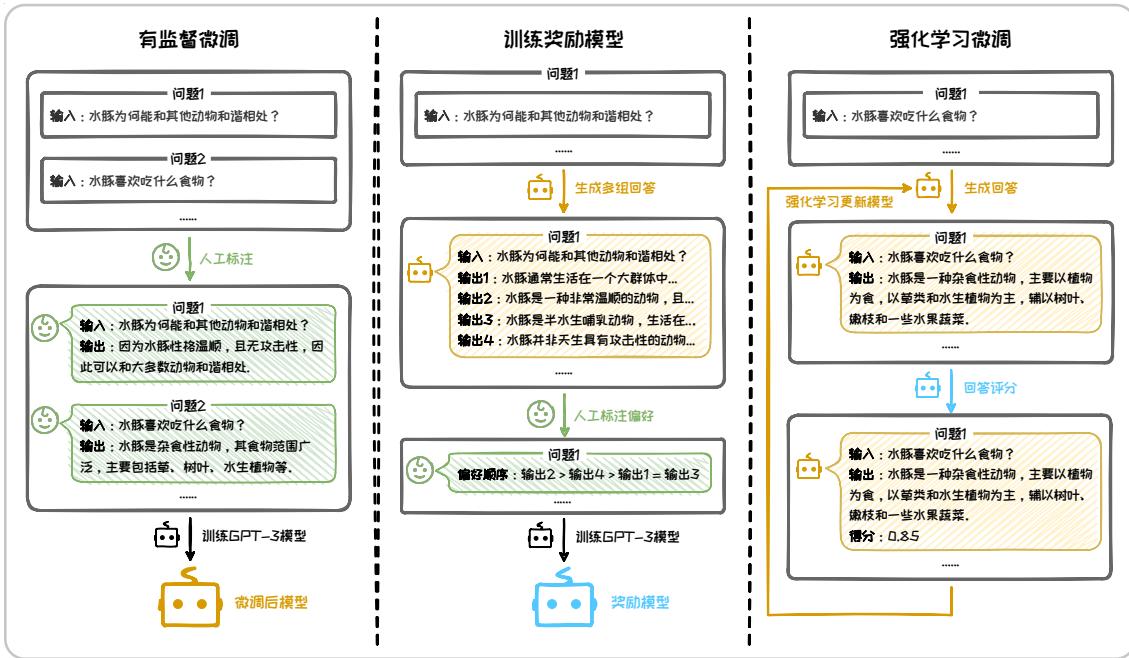


图 2.14: 人类反馈强化学习 (RLHF) 过程

出，指导后续的强化学习过程。

**3. 强化学习微调：**基于上一步中得到的奖励模型，**使用强化学习方法优化第一步中的语言模型**。在语言模型生成输出后，奖励模型对其进行评分，强化学习算法根据这些评分调整模型参数，以提升高质量输出的概率。这一过程通常会反复进行多次，不断地优化语言模型，逐步提高其输出质量和一致性。

经过 RLHF 训练得到的 InstructGPT 的性能通常优于 GPT-3，尤其是在需要精确遵循用户指令的场景中。它生成的回答更加贴合用户的查询意图，减少了不相关或误导性内容的生成。InstructGPT 的研究为构建更智能、更可靠的 AI 系统提供了新的思路，展示了人类智慧和机器学习算法结合的巨大潜力。

RLHF 的训练方法一经提出，立刻引发了学术界的广泛关注。尽管 RLHF 在优化大规模语言模型方面表现出色，但其复杂的训练过程、高昂的计算成本以及对奖励模型的依赖也带来了不少挑战：1) RLHF 的训练过程**需要单独训练一个奖励模型**，用于评估语言模型的输出质量，从而指导模型的改进。然而，训练奖励模型

是一个复杂且耗时的过程，需要大量高质量的人类反馈数据来确保奖励模型的准确性。2) RLHF 的训练涉及多个模型的联合训练。除了需要训练语言模型和奖励模型外，还需要协调这两个模型的交互，使得整个过程非常复杂且成本高昂。

## (2) 直接偏好优化 DPO

为了克服 RLHF 面临的问题，斯坦福大学和扎克伯格生物医学研究中心在 2023 年在其基础上，提出了一种新的算法，即直接偏好优化 (Direct Preference Optimization, DPO) [30]。DPO 算法通过直接利用人类偏好数据来训练模型，省略了单独构建奖励模型以及应用复杂强化学习算法的步骤。该方法首先收集包含多个响应的人类偏好数据，并从中标记出最优和次优响应。然后，通过优化模型，**提高模型选择最优响应的概率，同时降低选择次优响应的概率**。这种方法显著简化了训练流程，提高了训练效率和模型稳定性。尽管在处理复杂的人类偏好时可能略逊于 RLHF，但其在计算上的优势使其在多种应用场景中具有显著的实用性。

## (3) 其它扩展模型

除了 InstructGPT 之外，OpenAI 在这一阶段还推出了一系列实用性的衍生模型。例如，经过了自然语言和数十亿行代码的训练，专注于编程语言理解和生成的 Codex[5] 模型，以及能够与 Web 浏览器环境交互，模拟了网络搜索和信息评估过程的 WebGPT[25] 模型等。这些辅助模型为 GPT 系列后续版本的推出奠定了坚实的基础。

## 5. 体验优化和性能飞跃：ChatGPT 以及 GPT-4 等模型

OpenAI 先前推出的 GPT 系列模型自问世以来，在处理复杂语言任务方面展现出巨大潜力，在学术界备受关注。但这些模型更多地局限于学术研究和专业应用领域，适用范围始终有限。为了解决这一局限性，OpenAI 于 2022 年 11 月推出了聊天机器人 **ChatGPT**(Chat Generative Pre-trained Transformer)<sup>23</sup>，标志着一种新的

<sup>23</sup><https://openai.com/blog/chatgpt>

LLMaaS(LLM as a Service) 服务模式的出现。在 **LLMaaS 服务模式** 下, 用户可以通过 OpenAI 提供的网页端轻松使用预训练后的 ChatGPT 模型, 也可以通过 API 将其集成到自己的应用和系统中, 而无需处理模型训练、优化和基础设施管理等问题。这种新的服务模式使得 ChatGPT 的强大的技术变得易于访问和普及。

在技术层面, ChatGPT 可以看作是 GPT-3 的一个强化版本, 专门针对对话任务进行了优化, 更加注重用户体验和互动性。它不仅能够进行流畅的对话、提供深入的解释、协助内容创作、回答问题, 甚至还能模仿某些类型的人类推理。ChatGPT 的推出极大地拓宽了高端 AI 技术的受众, 成为人工智能领域发展的一个标志性事件。

随后, OpenAI 于 2023 年 3 月进一步发布了 **GPT-4<sup>24</sup>** 模型, 这一新模型在继承并增强了前代特点的同时, 带来了性能上的飞跃。GPT-4 在理解复杂语境、捕捉语言细微差别、生成连贯文本等方面表现出更高的精确度, 并且能够更有效地处理数学问题、编程挑战等高级认知任务。此外, GPT-4 还引入了对图文双模态的支持, 拓宽了其应用范围, 如图像描述、视觉问题解答等。

为了进一步提升模型性能以及用户体验, OpenAI 于 2024 年 5 月提出了 **GPT-4o<sup>25</sup>** 旗舰模型。GPT-4o 模型在前代 GPT-4 的基础上, 大幅提升了响应速度, 显著降低了延迟, 并且还增强了多模态处理能力以及多语言支持能力, 使其在客户支持、内容创作和数据分析等广泛的应用场景中表现优异。GPT-4o 的推出标志着 AI 语言模型在技术和应用上的又一重要进步, 进一步推动了人工智能领域的发展。

### 6. GPT 系列总结

GPT 系列模型的发展史是人工智能以及自然语言处理领域中一个激动人心的篇章。从 GPT-1 的初试啼声到 GPT-4o 的卓越性能, OpenAI 的这一系列产品不仅在技术上实现了跨越式的进步, 而且在应用上也极大地拓宽了语言模型的边界。

---

<sup>24</sup><https://openai.com/index/gpt-4-research>

<sup>25</sup><https://openai.com/index/hello-gpt-4o>

GPT-1 的诞生标志着自回归语言模型的起步，而 GPT-2 则在此基础上大幅提升了模型的规模和能力。GPT-3 以其突破性的参数量和上下文学习能力，将语言模型的应用推向了前所未有的高度。随后，Codex、WebGPT 和 InstructGPT 等衍生模型的推出，进一步扩展了 GPT 系列在编程、网络搜索和指令遵循方面的能力。ChatGPT 的问世，以全新的 LLMaaS 服务模式将这些先进的技术带入了公众视野，使广泛的应用场景和用户群体能够体验到 AI 的强大功能。最终，GPT-4 以及 GPT-4o 的推出，以其卓越的性能和多模态能力，为语言模型的未来描绘了更加广阔前景。

GPT 系列模型的演进不仅体现了深度学习技术在内容理解、文本生成和交互方面的巨大潜力，也展示了人工智能技术如何逐步走向成熟，并深入人们的日常生活。但值得注意的是，自 GPT-3 开始，由于技术和安全的考虑，OpenAI 的策略从最初的开源逐渐转向了闭源，开发者和企业可以通过 OpenAI 提供的 API 服务来使用后续的模型，这一现象反映了公司对 AI 技术可能滥用的担忧，以及对其商业化策略的调整，同时激发了对 AI 伦理、可达性和技术控制的广泛讨论。

### 2.5.2 LLAMA 系列模型

LLaMA (Large Language Model Meta AI) 是由 Meta AI (前身为 Facebook AI) 开发的一系列大型语言模型，旨在与 GPT 系列等顶尖语言模型竞争。自设计之初，LLaMA 就定位于提供开放且高效的语言处理能力，其模型权重在非商业许可证下向学术界开放，推动了知识的共享与技术的发展。

在模型架构上，LLaMA 借鉴了 GPT 系列的设计理念，同时在技术细节上进行了创新和优化。LLaMA 与 GPT 系列的主要区别在于，LLaMA 更加注重开放性，其开放策略不仅促进了技术进步，也使 LLaMA 迅速在开源社区中赢得了极高的人气，成为广受欢迎的大型模型之一。另外，GPT 系列的升级主线聚焦于模型规模与预训练预料的同步提升，而 LLaMA 则在模型规模上保持相对稳定，更专注于提

升预训练数据的规模, 表2.5展示了不同版本 LLaMA 模型对应的发布时间、参数量以及语料规模, 体现了 LLaMA 在语料规模上的显著提升。

表 2.5: LLaMA 系列模型参数和语料大小表

模型	发布时间	参数量 (亿)	语料规模
LLAMA-1	2023.02	67 / 130 / 325 / 652	约 5TB
LLAMA-2	2023.07	70 / 130 / 340 / 700	约 7TB
LLAMA-3	2024.04	80 / 700	约 50TB

Meta AI 共推出了三个版本的 LLaMA 模型, 加上众多研究者基于 LLaMA 模型衍生的各种模型, 共同构成了日益成熟的 LLaMA 生态系统。接下来将对 LLaMA 的三个版本及其部分衍生模型进行介绍。

## 1. LLaMA1 模型

LLaMA1[40] 是 Meta AI 于 2023 年 2 月推出的首个大语言模型, 标志着 LLaMA 系列模型的正式起步。下面分别对其模型架构和预训练方法展开介绍。

### (1) LLaMA1 模型架构

如图2.15所示, LLaMA1 在解码块的架构上与 GPT 系列采用的标准 Transformer 架构高度相似, 但在关键细节上进行了优化。

首先, 为了使训练过程更加稳定, LLaMA1 借鉴了 GPT-3[4] 中的 **Pre-Norm 层正则化策略**, 将正则化应用于自注意力和前馈网络的输入, 而不是输出。此外, LLaMA1 还使用了 **RMSNorm[47]** 替代了传统的层正则化。其次, 为了提升模型性能, LLaMA1 参考了 PaLM[6] 的做法, 将 Transformer 中的 RELU 激活函数改为 **SwiGLU 激活函数** [36]。最后, 为了提高词嵌入质量, LLaMA1 参考了 GPTNeo[3] 的做法, 使用 **旋转位置编码** (Rotary Positional Embeddings, RoPE) [38] 替代了原有的绝对位置编码, 从而增强位置编码的表达能力, 增强了模型对序列顺序的理解。此外, LLaMA1 将位置编码应用于每个解码块中, 在进行自注意力操作之前对查询 (query) 以及键 (key) 添加旋转位置编码。

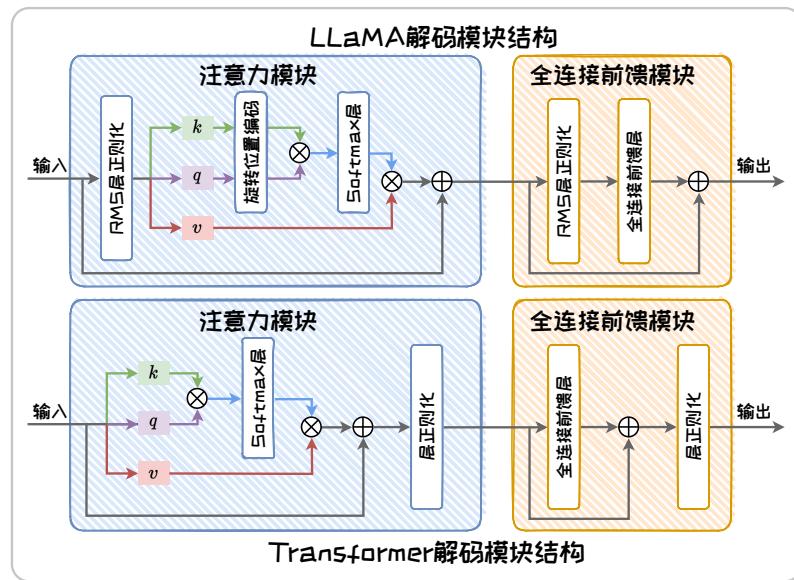


图 2.15: LLaMA 解码块架构与标准 Transformer 解码器架构对比

LLaMA1 共推出了 4 个版本的模型。不同版本对应的具体参数如表 2.6 所示。

表 2.6: LLaMA1 模型具体参数表

模型版本	解码块数量	隐藏层维度	自注意力头数量	总参数量 (亿)
LLaMA1-7B	32	4096	32	67
LLaMA1-13B	40	5120	40	130
LLaMA1-32B	60	6656	52	325
LLaMA1-65B	80	8192	64	652

## (2) LLaMA1 预训练方法

为了打造更为开放和便捷的模型，LLaMA1 在预训练策略上，将重心由训练速度转移到推理速度上，旨在用较小的模型在更大规模的数据集上进行更长时间的训练，从而在保持性能的同时，赋能更快的推理速度，极大地降低推理的耗时。因此，LLaMA1 在 Chinchilla<sup>26</sup> 提出的扩展法则的基础上，进一步加大了预训练数据集的规模，并且扩展了其来源。最终的预训练数据涵盖了大规模网页数据集 Common Crawl<sup>26</sup>、T5<sup>[31]</sup> 提出的 C4 数据集，以及来自 Github、Wikipedia、Gutenberg、Books3、Arxiv 以及 StackExchange 等多种来源的数据，总数据量高达 5TB。

<sup>26</sup><https://commoncrawl.org>

这种多样化的数据来源，加上自回归文本预测任务，使得 LLaMA1 在处理各种自然语言任务时表现出色，具有广泛的应用范围和强大的语言处理能力。

### 2. LLaMA2 模型

LLaMA2[39] 作为 Meta AI 在 LLaMA 系列上的第二代模型，于 2023 年 7 月发布。LLaMA2 在前代的基础上引入了更多参数版本，并进行了对话优化和数据集扩展，使其在各类自然语言处理任务中表现更加出色。下面对 LLaMA2 的模型结构、预训练方法以及微调方法进行介绍。

#### (1) LLaMA2 模型架构

在模型架构上，LLaMA2 继承了 LLaMA1 的创新技术，并提出了新的**分组查询注意力**（Grouped Query Attention, GQA）[1] 来提升模型的计算效率和性能。在分组查询注意力机制下，键（key）以及值（value）不再与查询（query）一一对应，而是一组查询共享相同的键和值，从而有效降低内存占用并减少模型总参数量。

LLaMA2 共推出了四个版本的模型，并为参数量最大的两个模型增加了分组查询注意力机制。不同版本对应的具体参数如表2.7所示：

表 2.7: LLaMA2 模型具体参数表

模型版本	解码块数量	隐藏层维度	自注意力头数量	总参数量（亿）
LLaMA2-7B	32	4096	32	70
LLaMA2-13B	40	5120	40	130
LLaMA2-34B	60	6656	52	340
LLaMA2-70B	80	8192	64	700

尽管 LLaMA2 在模型规模上相较 LLaMA1 并未实现巨大飞跃，但其性能的提升和效率的优化不容小觑。

#### (2) LLaMA2 预训练方法

Meta 在 LLaMA2 上继续延续了“小模型 + 大数据”的理念，在保持原有模型参数量的前提下，使用了更广泛和多样化的开源数据来进行预训练，将语料库的

规模扩大到了约 7TB，覆盖了更丰富的语言和领域资源。

基于这一庞大的数据基础，LLaMA2 在多项任务上的性能实现了全面超越，即使是参数量较少的模型版本，也展现出了超越前代更大模型的潜力。

### (3) LLaMA2 微调

在完成了预训练后，LLaMA2 使用了大规模且公开的指令微调数据集 [7] 对模型进行有监督的微调，通过这些带标注的数据来进一步优化模型，使其更适应特定任务。

此外，LLaMA2 还参考了 InstructGPT 的策略，融入了人类反馈强化学习方法，并通过 **近似策略优化** (Proximal Policy Optimization, PPO) [35] 以及 **拒绝采样** (Rejection Sampling) 之间的有机结合，迭代优化模型。在每轮迭代中，模型生成新的输出，由奖励模型进行评估，拒绝采样筛选出高质量的输出，而 PPO 则指导模型策略的改进，以便在后续迭代中生成更优质的输出。

这种结合了强化学习和质量筛选的迭代流程，不仅提升了 LLaMA2 在自然语言处理任务上的表现，也体现了 Meta AI 在推动语言模型发展上的创新能力。

## 3. LLaMA3 模型

Meta 在对 LLaMA2 的预训练过程进行深入分析后发现，模型在经过 7TB 的数据集预训练后，损失函数仍未完全收敛。这表明，对于 70 亿参数的模型而言，数据量仍有扩展空间。基于这一发现，Meta AI 于 2024 年 4 月进一步推出了第三代模型，即 LLaMA3<sup>27</sup>。

### (1) LLaMA3 模型架构

在模型架构上，LLaMA3 与前一代 LLaMA2 几乎完全相同，但在分词 (tokenizer) 阶段，**将字典长度整整扩大了三倍**，极大提升了推理效率。这一改进显著提升了推理效率，减少了中文字符等语言元素被拆分为多个 Token 的情况，有效降

<sup>27</sup><https://ai.meta.com/blog/meta-llama-3>

低了总体 Token 数量，从而提高了模型处理语言的连贯性和准确性。另一方面，扩大的字典有助于减少对那些具有完整意义的语义单元进行分割的频率，这意味着模型在处理文本时可以更准确的捕捉词义和上下文，从而提高生成文本的流畅性和连贯性。

在此基础上，LLaMA3 推出了两个版本的模型，分别是 80 亿参数以及 700 亿参数版本，均采用了分组查询注意力机制。这两个版本的模型参数与 LLaMA2 的对应版本保持高度一致，但在性能上实现了质的飞跃。

### (2) LLaMA3 预训练方法

为了更充分地训练模型，LLaMA3 挑选了大量高质量且公开来源的训练数据集，构建了规模达到 50TB 的庞大语料库，是 LLaMA2 的 7 倍之多。这一语料库不仅包含丰富的代码数据以增强模型的逻辑推理能力，还涵盖了超过 5% 的非英文数据，覆盖 30 多种语言，显著扩展了模型的跨语言处理能力。

经过有监督的微调和人类反馈强化学习，LLaMA3 的性能在多个评测指标上全面超越了前代模型。即便是参数量相对较少的 80 亿参数版本，也展现出了超越 LLaMA2 700 亿参数版本的卓越性能。而 700 亿参数版本的 LLaMA3，在多项任务上的性能更是超越了业界标杆 GPT-4 模型。

此外，据 Meta AI 官网透露，LLaMA3 的超大规模模型，参数量预计超过 4000 亿，目前仍在训练中。这一前所未有的模型规模预示着其性能有望达到新的高度，为自然语言处理领域带来革命性的突破。

## 4. LLaMA 衍生模型

由于 Meta AI 将所有版本的 LLaMA 模型开源共享，众多研究者使用这些模型来开发性能更好的开源语言模型，以与闭源的大语言模型竞争，或在特定任务下开发垂域的语言模型，呈现出一个多样化、充满活力的研究生态。这些模型根据主要创新方向可以划分为三类：

- **性能改进类模型**: 一些研究者专注于通过精细的微调方法来提升模型性能。

例如, Alpaca<sup>28</sup>是首个基于 LLaMA 进行微调的指令遵循模型, 它利用自生成指令 (Self-Instruct)<sup>[43]</sup> 的方法, 基于 GPT-3.5 生成了大量的指令遵循样例任务用于微调, 以较小的模型规模实现了与 GPT-3.5 相媲美的性能。Vicuna<sup>29</sup>模型则另辟蹊径, 它利用 shareGPT 平台上累积的日常对话数据进行微调, 进一步提升了对话能力。Guanaco<sup>[9]</sup> 模型则通过引入 QLoRA 技术, 显著降低了微调的时间成本, 提高了效率。

- **垂域任务类模型**: 尽管 LLaMA 在通用任务上表现出色, 但在特定领域的应用潜力仍待挖掘。研究者们因此针对特定领域对 LLaMA 进行微调, 以增强其垂直领域能力。例如, CodeLLaMA<sup>[32]</sup> 模型在 LLaMA2 的基础上, 利用大量公开代码数据进行微调, 能更好地适应自动化代码生成、错误检测、以及代码优化等任务。LawGPT<sup>[26]</sup> 模型通过构建包含 30 万条法律问答的数据集进行指令微调, 显著增强了对法律内容的处理能力。GOAT<sup>[22]</sup> 模型通过 Python 脚本生成的数学题库进行训练, 提高了解决各类数学题的准确率。Cornucopia<sup>30</sup>模型则利用金融问答数据进行微调, 增强了金融问答的效果。

- **多模态任务类模型**: 通过整合视觉模态编码器和跨模态对齐组件, LLaMA 模型得以扩展应用于多模态任务。例如, LLaVA<sup>[20]</sup> 利用 CLIP 提取图像特征并利用一个线性投影层实现图片和文本之间的对齐。MiniGPT4<sup>[48]</sup> 使用 VIT-G/14 以及 Q-Former 作为图像编码器, 并同样使用线性投影层来实现图片和文本之间的对齐, 展现了多模态任务处理能力。

Meta 的 LLaMA 系列模型不仅代表了大型语言模型技术的重要发展方向, 也体现了开源共享在推动技术进步和普及化中的重要作用。随着时间的推移, 我们

<sup>28</sup><https://crfm.stanford.edu/2023/03/13/alpaca.html>

<sup>29</sup><https://lmsys.org/blog/2023-03-30-vicuna>

<sup>30</sup><https://github.com/jerry1993-tech/Cornucopia-LLaMA-Fin-Chinese>

有理由期待 LLaMA 系列的模型在多个领域发挥更大的作用，并在处理伦理、偏见和资源分配等问题上取得更多进展。

### 2.5.3 Decoder-only 架构模型对比

表 2.8: GPT 系列和 LLaMA 系列模型参数和语料大小表

模型	发布时间	参数量 (亿)	语料规模
GPT-1	2018.06	1.17	约 5GB
GPT-2	2019.02	1.24 / 3.55 / 7.74 / 15	40GB
GPT-3	2020.05	1.25 / 3.5 / 7.62 / 13 / 27 / 67 / 130 / 1750	1TB
ChatGPT	2022.11	未知	未知
GPT-4	2023.03	未知	未知
GPT-4o	2024.05	未知	未知
LLAMA-1	2023.02	67 / 130 / 325 / 652	约 5TB
LLAMA-2	2023.07	70 / 130 / 340 / 700	约 7TB
LLAMA-3	2024.04	80 / 700	约 50TB

Decoder-only 架构的大型语言模型，凭借其简化的结构和高效的性能，在自然语言处理领域占据了重要的地位。随着深度学习技术的不断进步和创新，这些模型不仅为研究人员提供了丰富的工具，也为开发者开辟了新的应用路径。表2.8展示了 GPT 系列和 LLaMA 系列不同版本的具体参数，从中可以发现 Decoder-only 架构的发展之快。未来，随着技术的进一步成熟和优化，Decoder-only 模型有望在更多领域发挥更大的作用，推动人工智能技术的广泛应用和深入发展。

## 2.6 非 Transformer 架构

第 1.3 节介绍了 Transformer 结构和基于 Transformer 的语言模型，同时也提到了 Transformer 的缺陷，即其并行输入的机制会导致模型规模随输入序列长度平方增长。为了提高计算效率和性能，解决 Transformer 在长序列处理中的瓶颈问题，许

多非 Transformer 架构被提出。本节将介绍两种具有代表性的非 Transformer 模型架构：RWKV 和 Mamba。

### 2.6.1 RWKV

Transformer 在 NLP 任务中表现出色，但在处理长序列时面临内存和计算复杂度呈二次方增长的问题。RNN 虽然在内存和计算复杂度上呈线性增长，但在训练时无法有效并行化，且存在梯度消失或爆炸问题。RWKV (Receptance Weighted Key Value) [28] 架构旨在结合 Transformer 的并行训练优势和 RNN 的线性复杂度优势，以解决现有架构的局限性。（注：这里讨论的是 RWKV-v4）

为了减少注意力机制的计算复杂度，AFT (Attention Free Transformer) [46] 通过一种基于位置偏置的加权机制来替代传统的注意力机制公式。受 AFT 思想的启发，RWKV 提出一种简化方法，允许将模型表述为 Transformer 或 RNN，从而结合了两者的优势。

RWKV 模型的核心由两个主要模块组成，分别为 **时间混合模块** 和 **通道混合模块**。时间混合模块主要处理序列中不同时间步之间的关系，通道混合模块则关注同一时间步内不同特征通道之间的交互。这两个模块的设计基于四个基本元素：**接收向量  $R$ 、键向量  $K$ 、值向量  $V$  和 权重  $W$** 。其中， $R$  接收并整合过去的信息， $W$  调整位置权重， $K$  和  $V$  类似传统注意力机制中的键和值，分别用于匹配和携带信息。这些元素在时间混合模块和通道混合模块中起着核心作用。

RWKV 架构图如图 2.16 所示。RWKV 模型通过堆叠残差块，包含时间混合子块和通道混合子块，来利用过去的信息。图中主要包括 RWKV 的三个部分，即 Token 位移、WKV 和输出门控。

- **Token 位移**是指在处理序列数据时，将 Token 在时间维度上进行平移，从而帮助模型捕捉到序列数据中的时间依赖关系和上下文信息。在 RWKV 架构

中，Token 位移通过对当前时间步和前一时间步的输入进行线性插值来实现。

- RWKV 架构中 **WKV** 部分的计算方法与 AFT 类似，有所不同的是，AFT 中的 **W** 是一个成对矩阵，而在 RWKV 中，**W** 被视为一个按通道修改的向量，并且受到相对位置的影响。这种处理可以确保模型能够有效捕捉序列中的长期依赖关系，并通过单独关注当前 Token 来避免信息退化。
- **输出门控**通过使用 **R** 的 Sigmoid 函数  $\sigma(r)$  实现，确保了模型在处理复杂序列数据时能够有效地控制和调节信息流。

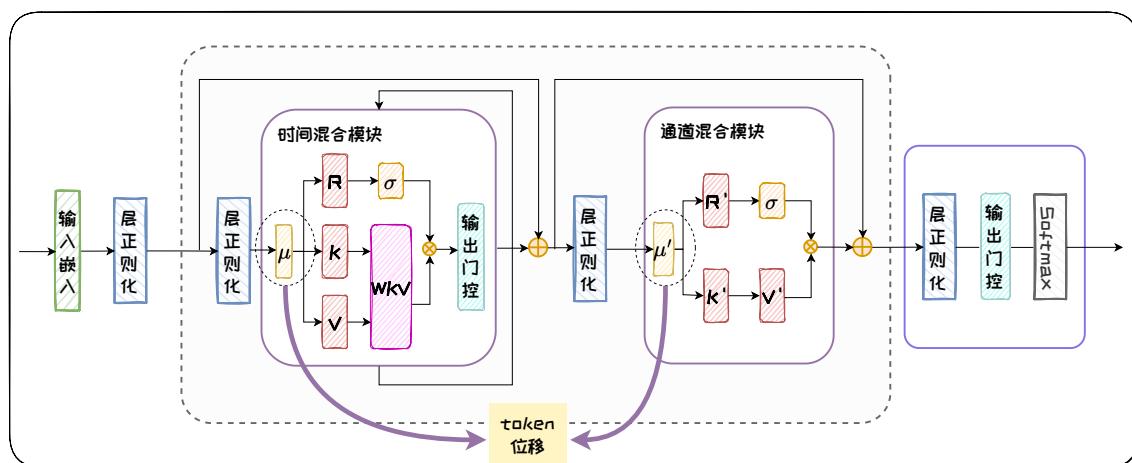


图 2.16: RWKV 架构

此外，RWKV 采用时间依赖的 Softmax 操作，提高数值稳定性和梯度传播效率。通过层归一化来稳定梯度，防止梯度消失和爆炸。RWKV 还采用了自定义 CUDA 内核、小值初始化嵌入以及自定义初始化来进行额外优化。

RWKV 在模型规模、计算效率和模型性能方面都表现可观：

1. **模型规模**: RWKV 模型参数扩展到了 14B，是第一个可扩展到数百亿参数的非 Transformer 架构。
2. **计算效率**: RWKV 使用了一种线性注意力机制，允许模型被表示为 Transformer 或 RNN，从而使得模型在训练时可以并行化计算，在推理时保持恒定的计算和内存复杂度。

3. **模型性能**: 在 NLP 任务上, RWKV 的性能与类似规模的 Transformer 相当。

在长上下文基准测试中, RWKV 的性能仅次于 S4[13, 14] 模型。

RWKV 通过创新的线性注意力机制, 成功结合了 Transformer 和 RNN 的优势, 在模型规模和性能方面取得了显著进展。然而, 在处理长距离依赖关系和复杂任务时, RWKV 仍面临一些局限性。为了解决这些问题并进一步提升长序列建模能力, 研究者们在 S4 模型成功的基础上, 提出了 Mamba。

## 2.6.2 Mamba

近年来, 随着长序列建模需求的增加, 结构化状态空间序列模型 (SSM) [13, 14] 应运而生。SSM 可以看作是 RNN 和 CNN 的结合体, 能够通过递推或卷积方式进行计算, 在序列长度上表现出线性或近线性扩展。然而, SSM 架构处理信息密集的数据 (如文本) 的效率较低。为了弥补这一不足, Mamba [12] 基于 SSM 架构, 提出了选择机制和硬件感知算法, 同时保证了快速训练和推理、高质量数据生成以及长序列处理能力。

Mamba 主要基于 SSM 架构进行改进。状态空间模型 (SSM) [13] 主要用于描述一个系统的当前状态, 并根据输入来预测下一个状态。SSM 是一种可以处理连续序列信息的递归模型, 包括两个核心公式, 分别为状态方程和输出方程。状态方程描述了状态如何基于输入和前一个状态变化, 输出方程则描述了状态如何转化为输出。

SSM 的一种表示是通过 Zero-order hold 技术使其离散化, 这种处理类似 RNN, 能够实现快速推理, 但是训练缓慢。另一种 SSM 的表示利用卷积操作的并行性来加速计算过程。通过综合以上两种表示的优点, 可以实现在训练过程中使用并行的卷积表示, 在推理过程中使用高效的循环表示。

此外, SSM 模型还具有线时不变性 (LTI), 即模型的动态在整个时间过程中

保持不变。这一特性使其可以通过递归和卷积来实现，但也造成了**处理动态变化、非线性关系和复杂信号时的局限性**。Mamba 基于该问题进行了改进。

Mamba 通过选择机制，使得模型能够执行基于内容的推理；通过硬件感知算法，实现了在 GPU 上的高效计算。

Mamba 的**选择机制**通过动态调整模型参数来选择需要关注的信息。具体来说，它使 SSM 中的参数成为关于输入的函数，并相应地改变了整个网络中的张量形状。这样，模型参数就会根据输入数据动态变化。对于不同的参数会选择相应的函数，包括线性变换函数、广播扩展机制以及 softplus 激活函数。选择这些特定的函数形式，是由于它们与 RNN 门控机制之间存在着关联。

选择机制使模型参数变成了输入的函数，且具有长度维度，因此模型不再具备卷积操作的平移不变性和线性时不变性，从而影响其效率。为了实现选择性 SSM 模型在 GPU 上的高效计算，Mamba 提出一种**硬件感知算法**，主要包括以下三部分：

1. **内核融合**：通过减少内存 I/O 操作来提高速度。
2. **并行扫描**：利用并行化算法提高效率。
3. **重计算**：在反向传播时重新计算中间状态，以减少内存需求。

具体实现中，将 SSM 参数从较慢的高带宽内存（HBM）加载到更快的静态随机存取存储器（SRAM）中进行计算，然后将最终输出写回 HBM。这样可以在保持高效计算的同时，减少内存使用，使模型内存需求与优化的 Transformer 实现（如 FlashAttention）相同。

Mamba 通过将带有选择机制的 SSM 模块与 Transformer 的前馈层相结合，形成了一个简单且同质的架构设计。如图 2.17 所示，Mamba 架构是由完全相同的 Mamba 模块组成的递归模型，每个 Mamba 模块都在前馈层中插入了卷积层和带有选择机制的 SSM 模块，其中激活函数  $\sigma$  选用 SiLU / Swish 激活。

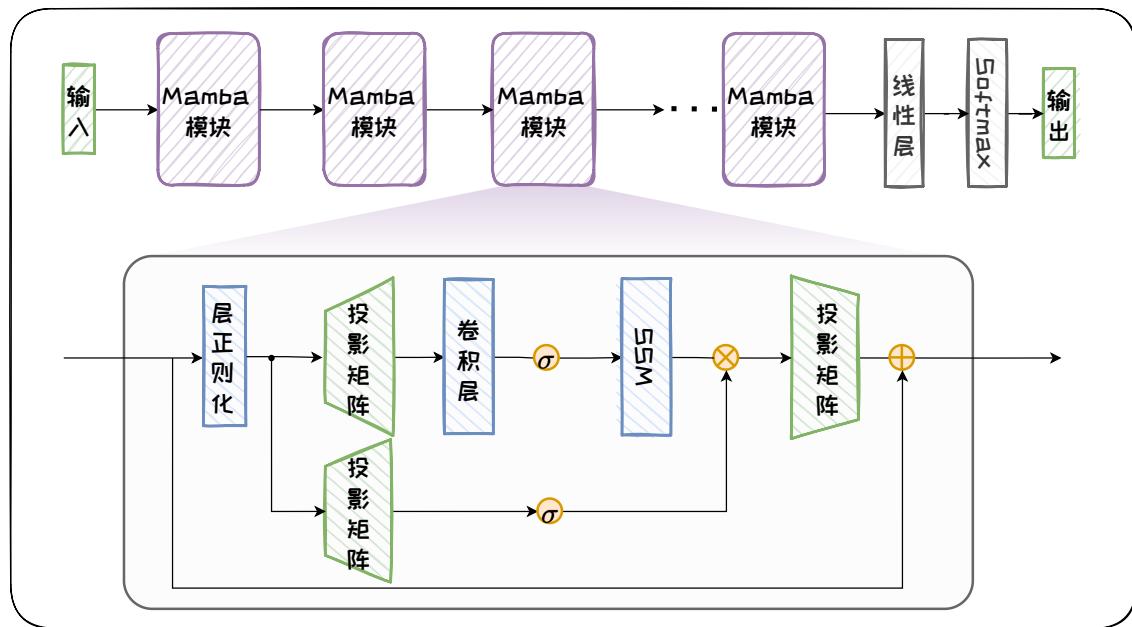


图 2.17: Mamba 架构

通过引入选择机制和硬件感知算法，Mamba 在实际应用中展示了卓越的性能和效率，包括以下三个方面：

- 1. 快速训练和推理：**训练时，计算和内存需求随着序列长度线性增长，而推理时，每一步只需常数时间，不需要保存之前的所有信息。通过硬件感知算法，Mamba 不仅在理论上实现了序列长度的线性扩展，而且在 A100 GPU 上，其推理吞吐量比类似规模的 Transformer 提高了 5 倍。
- 2. 高质量数据生成：**在语言建模、基因组学、音频、合成任务等多个模态和设置上，Mamba 均表现出色。在语言建模方面，Mamba-3B 模型在预训练和后续评估中性能超过了两倍参数量的 Transformer 模型性能。
- 3. 长序列处理能力：** Mamba 能够处理长达百万级别的序列长度，展示了处理长上下文时的优越性。

虽然 Mamba 在硬件依赖性和模型复杂度上存在一定的局限性，但是它通过引入选择机制和硬件感知算法显著提高了处理长序列和信息密集数据的效率，展示

了在多个领域应用的巨大潜力。Mamba 在多种应用上的出色表现，使其成为一种理想的通用基础模型。

本节介绍了两种具有代表性的非 Transformer 模型架构：RWKV 和 Mamba。这些模型专为解决 Transformer 在处理长序列数据时的计算效率和性能挑战而设计。RWKV 结合了 Transformer 的并行处理优势和 RNN 的线性复杂度，通过特有的时间和通道混合模块，实现了高效的长序列处理能力。另一方面，Mamba 模型基于 SSM 模型进行改进，引入选择机制和硬件感知算法，不仅保证了计算效率，也提升了模型在长序列和多模态任务中的表现。

根据当前语言模型发展趋势，我们可以预测，未来非 Transformer 架构将重点提高处理长序列的效率和多模态学习的能力，同时通过硬件与算法的协同优化，增强模型的可扩展性和实时性能。这些创新将使模型更适应广泛的应用需求，提升其在复杂任务中的表现。

## 参考文献

- [1] Joshua Ainslie et al. “Gqa: Training generalized multi-query transformer models from multi-head checkpoints”. In: *arXiv preprint arXiv:2305.13245* (2023).
- [2] Rohan Anil et al. “Palm 2 technical report”. In: *arXiv preprint arXiv:2305.10403* (2023).
- [3] Sid Black et al. “Gpt-neox-20b: An open-source autoregressive language model”. In: *arXiv preprint arXiv:2204.06745* (2022).
- [4] Tom Brown et al. “Language models are few-shot learners”. In: *NeurIPS*. 2020.
- [5] Mark Chen et al. “Evaluating large language models trained on code”. In: *arXiv preprint arXiv:2107.03374* (2021).
- [6] Aakanksha Chowdhery et al. “Palm: Scaling language modeling with pathways”. In: *Journal of Machine Learning Research* 24.240 (2023), pp. 1–113.
- [7] Hyung Won Chung et al. “Scaling instruction-finetuned language models”. In: *Journal of Machine Learning Research* 25.70 (2024), pp. 1–53.

- [8] Kevin Clark et al. “Electra: Pre-training text encoders as discriminators rather than generators”. In: *arXiv preprint arXiv:2003.10555* (2020).
- [9] Tim Dettmers et al. “Qlora: Efficient finetuning of quantized llms”. In: *NeurIPS*. 2024.
- [10] Jacob Devlin et al. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. In: *NAACL*. 2019.
- [11] Yihe Dong, Jean-Baptiste Cordonnier, et al. “Attention is not all you need: Pure attention loses rank doubly exponentially with depth”. In: *ICML*. 2021.
- [12] Albert Gu and Tri Dao. “Mamba: Linear-Time Sequence Modeling with Selective State Spaces”. In: *arXiv preprint arXiv:2312.00752* (2023).
- [13] Albert Gu et al. “Combining Recurrent, Convolutional, and Continuous-time Models with Linear State Space Layers”. In: *NeurIPS*. 2021.
- [14] Albert Gu et al. “On the Parameterization and Initialization of Diagonal State Space Models”. In: *NeurIPS*. 2022.
- [15] Jordan Hoffmann et al. “Training compute-optimal large language models”. In: *arXiv preprint arXiv:2203.15556* (2022).
- [16] Mandar Joshi et al. “Spanbert: Improving pre-training by representing and predicting spans”. In: *Transactions of the association for computational linguistics* 8 (2020), pp. 64–77.
- [17] Jared Kaplan et al. “Scaling laws for neural language models”. In: *arXiv preprint arXiv:2001.08361* (2020).
- [18] Zhenzhong Lan et al. “Albert: A lite bert for self-supervised learning of language representations”. In: *arXiv preprint arXiv:1909.11942* (2019).
- [19] Mike Lewis et al. “Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension”. In: *ACL*. 2020.
- [20] Haotian Liu et al. “Visual instruction tuning”. In: *NeurIPS*. 2024.
- [21] Qi Liu, Matt J Kusner, and Phil Blunsom. “A survey on contextual embeddings”. In: *arXiv preprint arXiv:2003.07278* (2020).
- [22] Tiedong Liu and Bryan Kian Hsiang Low. “Goat: Fine-tuned llama outperforms gpt-4 on arithmetic tasks”. In: *arXiv preprint arXiv:2305.14201* (2023).

- [23] Yinhan Liu et al. “Multilingual denoising pre-training for neural machine translation”. In: *Transactions of the Association for Computational Linguistics* 8 (2020), pp. 726–742.
- [24] Yinhan Liu et al. “Roberta: A robustly optimized bert pretraining approach”. In: *arXiv preprint arXiv:1907.11692* (2019).
- [25] Reiichiro Nakano et al. “Webgpt: Browser-assisted question-answering with human feedback”. In: *arXiv preprint arXiv:2112.09332* (2021).
- [26] Ha-Thanh Nguyen. “A brief report on lawgpt 1.0: A virtual legal assistant based on gpt-3”. In: *arXiv preprint arXiv:2302.05729* (2023).
- [27] Long Ouyang et al. “Training language models to follow instructions with human feedback”. In: *NeurIPS*. 2022.
- [28] Bo Peng et al. “RWKV: Reinventing RNNs for the Transformer Era”. In: *EMNLP*. 2023.
- [29] Alec Radford et al. “Improving language understanding by generative pre-training”. In: (2018).
- [30] Rafael Rafailov et al. “Direct preference optimization: Your language model is secretly a reward model”. In: *NeurIPS*. 2024.
- [31] Colin Raffel et al. “Exploring the limits of transfer learning with a unified text-to-text transformer”. In: *Journal of machine learning research* 21.140 (2020), pp. 1–67.
- [32] Baptiste Roziere et al. “Code llama: Open foundation models for code”. In: *arXiv preprint arXiv:2308.12950* (2023).
- [33] Victor Sanh et al. “Multitask prompted training enables zero-shot task generalization”. In: *arXiv preprint arXiv:2110.08207* (2021).
- [34] Rylan Schaeffer, Brando Miranda, and Sanmi Koyejo. “Are emergent abilities of large language models a mirage?” In: *NeurIPS*. 2024.
- [35] John Schulman et al. “Proximal policy optimization algorithms”. In: *arXiv preprint arXiv:1707.06347* (2017).
- [36] Noam Shazeer. “Glu variants improve transformer”. In: *arXiv preprint arXiv:2002.05202* (2020).
- [37] Shaden Smith et al. “Using deepspeed and megatron to train megatron-turing nlg 530b, a large-scale generative language model”. In: *arXiv preprint arXiv:2201.11990* (2022).

- [38] Jianlin Su et al. “Roformer: Enhanced transformer with rotary position embedding”. In: *arXiv preprint arXiv:2104.09864* (2021).
- [39] Hugo Touvron et al. “Llama 2: Open foundation and fine-tuned chat models”. In: *arXiv preprint arXiv:2307.09288* (2023).
- [40] Hugo Touvron et al. “Llama: Open and efficient foundation language models”. In: *arXiv preprint arXiv:2302.13971* (2023).
- [41] Trieu H Trinh and Quoc V Le. “A simple method for commonsense reasoning”. In: *arXiv preprint arXiv:1806.02847* (2018).
- [42] Ashish Vaswani et al. “Attention is all you need”. In: *NeurIPS*. 2017.
- [43] Yizhong Wang et al. “Self-instruct: Aligning language models with self-generated instructions”. In: *ACL*. 2023.
- [44] Linting Xue et al. “mT5: A massively multilingual pre-trained text-to-text transformer”. In: *NAACL*. 2021.
- [45] Aiyuan Yang et al. “Baichuan 2: Open large-scale language models”. In: *arXiv preprint arXiv:2309.10305* (2023).
- [46] Shuangfei Zhai et al. “An Attention Free Transformer”. In: *arXiv preprint arXiv:2105.14103* (2021).
- [47] Biao Zhang and Rico Sennrich. “Root mean square layer normalization”. In: *NeurIPS*. 2019.
- [48] Deyao Zhu et al. “Minigpt-4: Enhancing vision-language understanding with advanced large language models”. In: *arXiv preprint arXiv:2304.10592* (2023).
- [49] Yukun Zhu et al. “Aligning books and movies: Towards story-like visual explanations by watching movies and reading books”. In: *ICCV*. 2015.



# 3 Prompt 工程

随着模型训练数据规模和参数数量的持续增长，大语言模型突破了泛化瓶颈，并涌现出了强大的指令跟随能力。泛化能力的增强使得模型能够处理和理解多种未知任务，而指令跟随能力的提升则确保了模型能够准确响应人类的指令。这种能力的结合，使得我们能够通过精心编写的指令输入，即 Prompt，来引导模型适应各种下游任务，从而避免了传统微调方法所带来的高昂计算成本。Prompt 工程，作为一门专注于如何编写这些有效指令的技术，成为了连接模型与任务需求之间的桥梁。它不仅要求对模型有深入的理解，还需要对任务目标有精准的把握。通过 Prompt 工程，我们能够最大化地发挥大语言模型的潜力，使其在多样化的应用场景中发挥出卓越的性能。

本章将深入探讨 Prompt 工程的概念、方法及作用。我们将介绍上下文学习、思维链等技术，并提供实用的 Prompt 使用技巧，以及介绍基于 Prompt 工程的相关应用。通过本章的学习，读者将能深入理解并掌握 Prompt 工程的精髓，进而在大语言模型的应用和优化中发挥更大的创造力和效率。

## 3.1 Prompt 工程简介

传统的自然语言处理研究遵循“预训练-微调-预测”范式。然而，随着语言模型在规模和能力上的显著提升，一种新的范式——“预训练-提示预测”应运而生。如图 3.1 所示，在这种范式下，通过精心设计 Prompt，模型能够快速适应下游任务，无需进行繁琐的微调过程。这种如何编写 Prompt 的技术被称为 Prompt 工程。

在本节中，我们将深入介绍 Prompt 工程的定义及其相关概念，探讨其在自然语言处理领域中的重要性和应用。

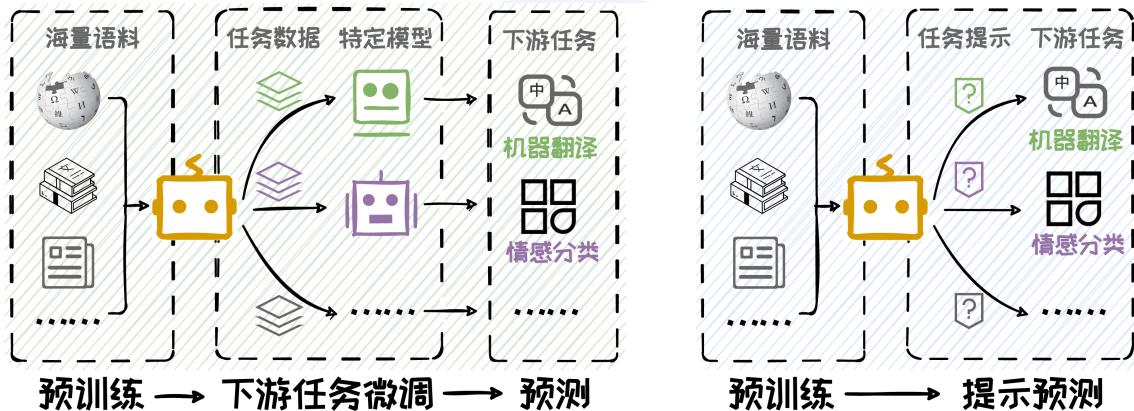


图 3.1: 自然语言处理任务范式对比

### 3.1.1 什么是 Prompt

#### 1. Prompt 的定义

Prompt 是指用于指导生成式人工智能模型执行特定任务的输入指令<sup>1</sup>，这些指令通常以自然语言文本的形式出现。Prompt 的核心目的是清晰地描述 AI 应该执行的任务，无论是生成文本、图像、音频还是其他类型的输出。

如图 3.2 所示，通过给出明确的任务指令，我们可以让模型执行多种任务。例

<sup>1</sup> [https://en.wikipedia.org/wiki/Prompt\\_engineering](https://en.wikipedia.org/wiki/Prompt_engineering)

如，我们可以让模型对文本进行情感分类，或者给定一个主题，让模型创作一首诗。在使用多模态模型时，我们还可以提供一个描述，让模型生成一幅画。通过编写不同的 Prompt，模型能够完成各种各样的任务。



图 3.2: 几种常见的 Prompt 例子

尽管 Prompt 的应用不限于文本到文本的任务，本章介绍的内容主要针对文本生成模型，并探讨如何通过精心设计的 Prompt 来引导模型生成符合特定任务要求的文本输出。

### 3.1.2 什么是 Prompt 工程

Prompt 工程 (Prompt Engineering)，又叫提示工程，是指设计和优化用于与生成式人工智能模型交互的 Prompt 的过程<sup>2</sup>。这种技术的核心在于，将任务重新构想为模型在预训练阶段已经熟悉的形式，利用模型固有的泛化能力来执行新的任务，而无需在额外的特定任务上进行训练。

Prompt 工程的概念源于一个简单的观察：即使是最先进的语言模型，也需要清

<sup>2</sup>[https://en.wikipedia.org/wiki/Intelligent\\_agent](https://en.wikipedia.org/wiki/Intelligent_agent)

晰的指导来理解和执行特定的任务。这些指导通过 Prompt 提示的形式呈现，它们是精心设计的文本片段，能够激发模型生成期望的输出。Prompt 工程的成功依赖于对预训练模型的深入理解，以及对任务需求的精确把握。通过构造合适的 Prompt 输入给大模型，大模型能够帮助我们完成各种任务。

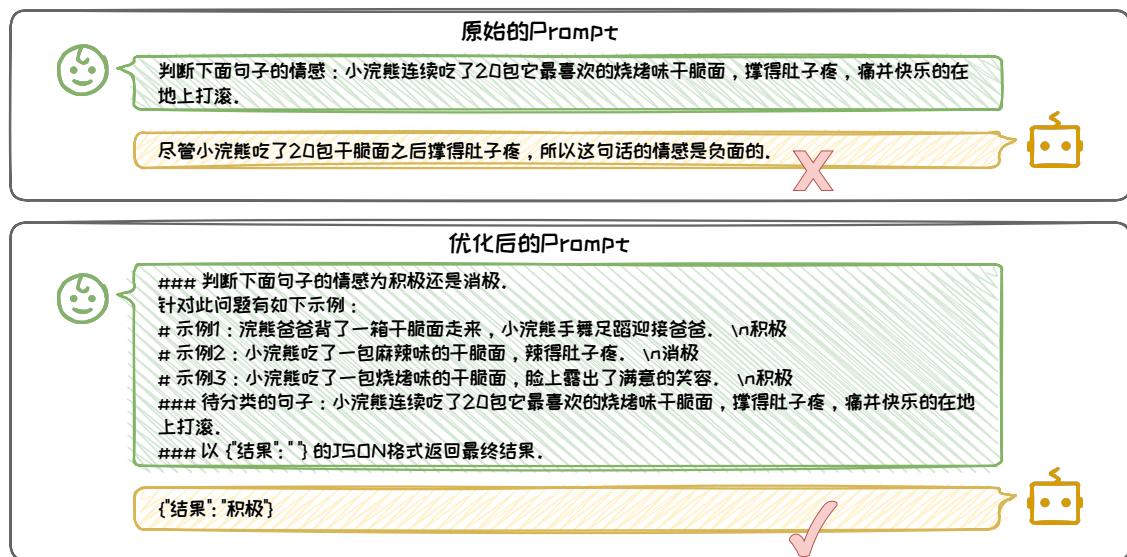


图 3.3: Prompt 工程技术应用前后的效果对比

如图 3.3，通过 Prompt 工程优化，原始的提示可以被改写成更加全面的形式，从而输入给大型模型，以获得更优质的回答。图中上方展示了一个普通的提示，而通过提示工程技术优化后，我们得到图中下方这样一个更加全面的提示，显著提升了模型的回答质量。

由此可见，构建高效的 Prompt 是与大模型互动时不可或缺的一环，它决定了你能否获得有价值的输出。一个优秀的 Prompt 通常由任务说明、上下文、问题、输出格式四个基本元素组成：

- **任务说明**——这是你请求模型执行的具体任务。它应该是明确的、直接的，并且尽可能详细地描述你希望模型完成的任务。例如，如果你需要让模型做情感分类任务，你的指令应该直接说明需要“将下面的句子分类为积极或消

极”。

- **上下文**——提供给模型的背景信息，用以增强其理解和生成的相关性。上下文可以包括特定的知识前提、目标受众的背景、相关任务的一些示例，或者任何可以帮助模型更好理解任务的信息。
- **问题**——具体的问题或是需要处理的信息。这部分应直接涉及你的查询或任务，为模型提供一个明确的起点，这里问题可以是显示的提问，或者是正常的陈述句，用来隐式表达出用户想提问的问题。
- **输出格式**——期望模型如何展示其回答的指引。这包括输出的格式（如列表、段落、报告），以及任何特定的细节要求，如简洁性或详细程度。比如，我们可以指示模型以 JSON 的格式输出结果。

Prompt 的四个基本元素——任务说明、上下文、问题和输出格式——对于大型模型生成的效果具有显著影响。这些元素的精心设计和组合构成了 Prompt 工程的核心。Prompt 工程不仅涉及这些基本元素的优化，还包括多种组件和技术的应用，如上下文学习（In-Context Learning）和思维链（Chain of Thought）等。这些技术和技巧的结合使用，可以显著提升 Prompt 的质量，进而提高模型输出的准确性和相关性。通过深入理解和应用这些组件，Prompt 工程能够有效地引导模型生成更符合特定任务需求的输出。具体关于上下文学习的内容将在 3.2 节中讨论，思维链的内容将在 3.3 节中讨论，而 Prompt 的使用技巧将在 3.4 节中详细探讨。

通过 Prompt 工程技术精心优化 Prompt 的四个基本元素，原本简单的 Prompt 可以被优化为更为详尽和全面的指令，从而显著提升模型的输出质量。此外，优化 Prompt 还包括了对模型输出的后处理，如通过特定的评分机制或逻辑校验来进一步提高答案的准确性和可靠性。这种全面的优化策略，不仅提升了模型的性能，也增强了模型在实际应用中的稳定性和可信度。

然而，随着 Prompt 内容的丰富和复杂化，输入到模型中的上下文长度也随之

增加，这不可避免地导致了模型推理速度的减慢和成本的上升。因此，在追求模型性能的同时，如何有效控制和优化 Prompt 的长度，成为了一个亟待解决的问题。我们的第二个优化目标是，在确保模型性能不受影响的前提下，尽可能压缩输入到大型模型中的 Prompt 长度。

为此，LLMLingua [11] 提出了一种创新的由粗到细的 Prompt 压缩方法，该方法能够在不牺牲语义完整性的情况下，将提示内容压缩至原来的二十分之一，同时几乎不损失模型的性能。此外，随着 RAG 技术的兴起，模型需要处理的上下文信息量大幅增加。FIT-RAG [22] 技术通过高效压缩检索出的内容，成功将上下文长度缩短至原来的 50% 左右，同时保持了性能的稳定，为处理大规模上下文信息提供了有效的解决方案。

### 3.1.3 模型如何处理 Prompt

在通过 Prompt 工程技术构造完合适的 Prompt 之后，需要将其输入到大语言模型中，让大模型来理解处理内容，从而生成回答。首先分词器（Tokenizer），根据已有的词表，将 Prompt 分割成一个一个词元（Token）。分词过程使得复杂的文本内容变得易于计算机理解和处理。每个 Token 代表了文本中的一个最小可分单位，随后这些 Token 会被进一步转换成向量形式，以便模型进行识别和处理。这一转换过程使得语言模型能够捕捉文本的深层语义结构，从而有效地处理和学习各种语言结构，从简单的词汇到复杂的句式和语境。

分词器在构建词表时，会依据分词算法在语料库上统计的词频等信息，精心挑选出包含在词表中的 Token。这一过程需谨慎平衡词表的大小，以确保模型的学习效果和语义表达能力。

若词表过大，虽然能涵盖更多长尾词汇，但可能导致对这些词汇的学习不够深入，且难以有效捕捉同一单词不同形态之间的关联。相反，若词表过小，例如仅

包含 26 个英文字母，虽然理论上能组合出几乎所有单词，但会导致形态相近但意义迥异的词汇在模型中难以区分，从而限制了模型承载丰富语义的能力，并可能大幅增加生成的序列长度。因此，在构建词表时，需在涵盖广泛词汇与保持语义精细度之间找到恰当的平衡点，以确保大语言模型既能学习到丰富的词汇知识，又能准确理解和生成具有复杂语义的文本。

如图 3.4 所示，对于“小浣熊吃干脆面”这段文本，分词器（Tokenizer）首先将其拆分为一个 Token 序列，用“BOS”标识一句话的开始。我们注意到，“浣”字被拆分成了两个 Token，“æμ”和“£”。这是因为“浣”这个字在语料库中出现较少，分词过程没有将其分成一个 Token。为了表示“浣”这个字，根据其中文字符值，将“浣”拆分为了两个字节，分别对应“æμ”和“£”。另一方面，词表中会把语料库中出现频率较高的词语或者短语做成 Token，以更好地让模型理解这个词的含义。比如这里“干脆”这个词就使用一个单独的 Token 来表示。这样的处理方式，使得在词表中既能包含常见的高频词汇，又能通过拆分处理稀有字符，从而在词汇覆盖和模型理解之间达到一个平衡。

随后，分词器将 Token 序列通过词表映射成 Token ID 序列。每个 Token ID 随后经过模型的 Embedding 矩阵处理，转化为固定大小的表征向量。这些向量序列被直接输入到模型中，供模型理解和处理。在模型生成阶段，模型会根据输入的向量序列，计算出词表中每个词的概率分布。从这些概率分布中，模型选择出对应的 Token ID，并将这些 Token ID 通过反射过程转换回 Token，最终输出相应的内容。这一过程确保了模型能够准确地理解和生成文本，实现了从 Token ID 到文本内容的流畅转换。

分词器的质量对模型的性能有着直接的影响。一个优秀的分词器不仅能够提高模型的处理速度，减少计算资源的消耗，还能显著提升模型对文本的理解能力。一个好的分词器应当具备以下特点：首先，它能够准确地识别出文本中的关键词和

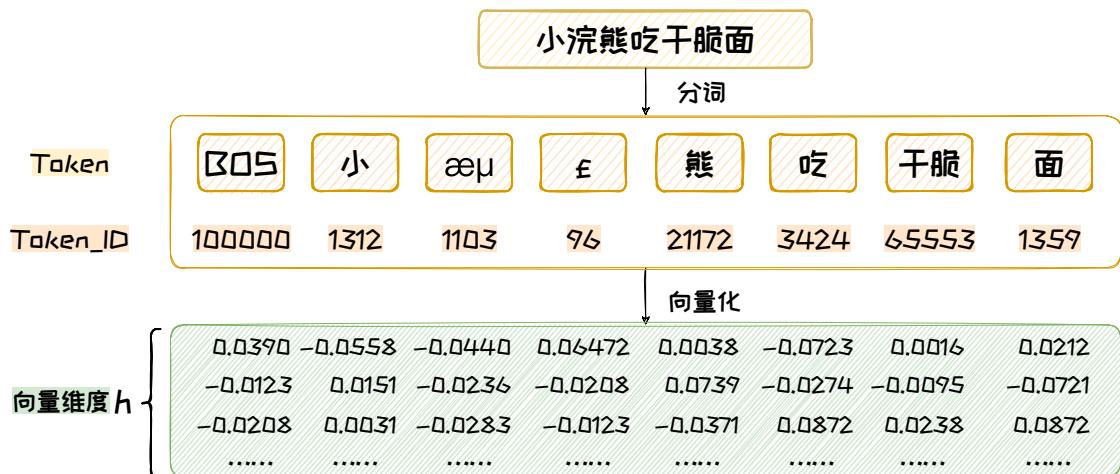


图 3.4: 使用 Tokenizer 处理文本过程, 以 DeepSeek-V2-Chat 模型的分词器为例 [7]

短语, 从而帮助模型更好地捕捉语义信息; 其次, 分词器的效率直接影响到模型的训练和推理速度, 一个快速的分词器可以显著减少模型处理大量数据所需的时间。

例如, 在处理“浣”字时, 分词器需要用两个 Token 来表示, 而“干脆”这两个字, 大模型只需要使用一个 Token 即可表示。这表明, 分词器的分词效率越高, 模型在训练和推理过程中的速度就越快, 从而使得模型能够更高效地处理和理解文本数据。

在常用的开源模型中, 不同模型采用了不同的分词器, 这些分词器具有各自的特点和性能。它们的质量受到多种因素的影响, 包括词表的大小、分词的效率等属性。表 3.1 是对常见开源大模型的分词器的对比分析, 其中中文语料库节选自《朱自清散文》<sup>3</sup>, 英文语料库来自莫泊桑短篇小说《项链》<sup>4</sup>。我们可以观察到, 像 DeepSeek、Qwen 这类中文开源大模型, 对中文分词进行了优化, 平均每个 Token 能够表示 1.3 个字 (每个字仅需 0.7 个 Token 即可表示), 一些常用词语和成语甚至可以直接用一个 Token 来表示。相比之下, 国外的模型, 如 GPT-4、LLaMA 系

<sup>3</sup>[https://www.sohu.com/a/746456997\\_120075260](https://www.sohu.com/a/746456997_120075260).

<sup>4</sup><https://americanliterature.com/author/guy-de-maupassant/short-story/the-diamond-necklace>

列，对中文的支持度较弱，分词效率不高。在英文中，由于存在“ly”、“ist”等后缀 Token，一个英文单词通常需要用 1 个及以上的 Token 来表示。

通过这种对比，我们可以清晰地看到不同模型分词器在处理不同语言时的效率，这对于选择合适的模型和优化模型性能具有重要的指导意义。

表 3.1: 模型分词器对比表

模型	词表大小	中文分词效率 (字 / Token)	英文分词效率 (词 / Token)
LLaMA1	32000	0.6588	0.6891
LLaMA2	32000	0.6588	0.6891
LLaMA3	128256	1.0996	0.7870
DeepSeek-V1	100016	1.2915	0.7625
DeepSeek-V2	100002	1.2915	0.7625
GPT-3.5 & GPT-4	100256	0.7723	0.7867
GPT-3	50,257	0.4858	0.7522
Qwen-1.5	151646	1.2989	0.7865
StarCoder	49152	0.9344	0.6513

### 3.1.4 Prompt 工程的意义

Prompt 工程提供了一种更为高效和灵活的途径来执行自然语言处理任务，它允许我们无需对模型进行微调，便能有效地完成既定任务，避免微调带来的巨大开销。通过精心设计的 Prompt，我们能够激发大型语言模型的内在潜力，使其在任务泛化、数据增强、智能代理等多个领域发挥出卓越的性能。

此外，通过 Prompt 工程技术，我们能够充分利用大型模型的丰富知识和强大的语言处理能力，来增强现有数据集的质量，还能创造出新的、高质量的数据集，这些数据集对于训练和提升小型模型的性能具有不可估量的价值。

#### 1. 任务泛化

在自然语言处理领域，任务泛化是一个关键挑战，它要求模型能够处理和理解多种不同类型的任务，而无需针对每个任务进行特定的训练。Prompt 工程通过

其独特的方法，为解决这一挑战提供了新的视角。通过设计恰当的 Prompt，模型能够在不经过额外训练的情况下，有效地执行和泛化到新的任务上。这种能力极大地扩展了模型的应用范围，使其能够适应更为广泛和多样化的应用场景。

在 Text-to-SQL 任务领域，基于 GPT 模型的 Prompt 工程方法在 Spider [39] 榜单上取得了突破性成绩，超越了传统的微调方法。这些方法通过构建合适的 Prompt，引导模型生成高质量的 SQL 查询，展现了 Prompt 工程在特定任务上的强大潜力。同时，在知识密集型任务问答领域 MMLU [9] 榜单上，基于 Prompt 工程的方法也取得最佳效果。这些方法通过提供设计合适的提示，使模型能够迅速适应新任务，无需大量标注数据或复杂的微调过程，进一步证明了 Prompt 工程在提升模型泛化能力方面的有效性。

### 2. 数据增强

数据增强是提升模型性能的重要手段，尤其是在数据稀缺的情况下。利用 Prompt 工程技术，不仅能够提升现有数据集的质量，还能够生成新的、高质量的数据集。这些数据集可以用于训练和优化模型，特别是在小型模型或资源受限的环境中，其价值尤为显著。

我们可以引导模型生成包含丰富推理步骤的数据集，能够通过在增强数据集上微调，增强模型推理能力 [40]。此外，通过精心设计的提示，还能生成包含复杂指令的数据集，如 Alpaca [29] 和 Evol-Instruct [21]。这些合成的数据集用于微调模型，使其在保持较小模型尺寸和低计算成本的同时，接近大型模型的性能，有效提升小模型的泛化能力。

### 3. 智能代理

智能代理（Intelligent Agent, IA）是指能够感知环境、自主采取行动以实现目标，并通过学习或获取知识来提高其性能的代理 [34]。随着大模型（如 GPT-4）的出现，基于 Prompt 工程的智能代理在多个领域展现了强大的应用潜力。这些智能

代理能够理解复杂的任务需求，通过自然语言进行交互，并作出合理的判断和决策。

基于大模型的智能体（Agent）通过设计适当的 Prompt，无需专门训练即可自动完成各种任务，如斯坦福大学利用 GPT-4 模拟的虚拟西部小镇 [25]，多个基于 GPT-4 的 Agent 在其中生活和互动，他们根据自己的角色和目标自主行动，进行交流，解决问题，并推动小镇的发展。此外，用户也可以通过 Prompt 工程的方式自定义智能代理，OpenAI 推出了可自定义 ChatGPT——GPTs [23]，用户通过设置特定 Prompt 和知识库，针对特定任务和场景提供优化服务，实现高效任务完成。这种定制化的智能代理进一步强化了 Prompt 工程在实际应用中的灵活性和实用性，为用户提供了更加个性化和高效的服务体验。

本节探讨了 Prompt 的概念，Prompt 工程的概念以及意义，揭示了 Prompt 在大模型应用中的关键作用和广阔潜力。接下来，我们将进一步拓展这一主题：第 3.2 节将探索上下文学习的相关内容，揭示其在提升模型理解和响应能力中的作用；第 3.3 节将详细介绍思维链提示方法及其变种，展示如何通过这些方法增强模型的逻辑推理和问题解决能力；第 3.4 节将分享构建有效 Prompt 的技巧，指导读者如何设计 Prompt 以激发模型生成更优质的内容；最后，第 3.5 节将具体展示 Prompt 工程在大模型中的实际应用，通过实例阐释其在不同场景下的应用策略和效果。希望通过这一系列深入浅出的讲解，读者能够全面理解并掌握 Prompt 工程的精髓，从而在大模型的应用和优化中发挥更大的创造力和效率。

## 3.2 上下文学习

随着模型训练数据规模和参数数量的持续扩大，大语言模型涌现出了多种能力，其中最引人注目的是上下文学习（In-Context Learning, ICL）能力，其使得语言模型能够通过分析任务说明或示例等信息来掌握任务范式。在某些任务上，我们不再需要针对某个任务训练一个模型或者在预训练模型上费时费力的微调，就可以达到快速适应下游任务的效果。大语言模型的这一卓越能力，为将大语言模型作为服务来解决各种自然语言处理问题，即“语言模型即服务”（LLM as a Service）的模式，奠定了坚实的能力基础。由于其诸多优点，上下文学习已成为自然语言处理的一种新范式。在接下来的内容中，我们将从上下文学习的定义，演示示例选择，和影响其性能的因素，逐步详细介绍上下文学习。

### 3.2.1 上下文学习的定义

上下文学习（In-Context Learning, ICL）<sup>[2]</sup>是一种通过构造特定的 Prompt，来使得语言模型理解并学习下游任务的范式，这些特定的 Prompt 中可以包含演示示例，任务说明等元素。

上下文学习的实现关键在于如何设计有效的 Prompt，以引导模型理解任务的上下文和目标。通常，这些 Prompt 会包含一系列的示例，每个示例都展示了任务的一个完整实例，包括输入和对应的输出。通过这种方式，模型能够从这些示例中学习任务的逻辑和规则，从而在没有额外训练的情况下，生成符合任务要求的输出，由于这些优点，上下文学习广泛应用于数据清洗，数据合成，实体匹配等任务中。

上下文学习中的 Prompt 通常包含关于任务的几个演示示例，每个示例都展示了如何从任务输入到预期输出的转换。这些示例按照特定顺序组成上下文，并与



图 3.5: 上下文学习示例

问题拼接形成 Prompt 输入给大模型。大模型从上下文中学习任务范式，同时利用模型自身的能力得到结果。在图 3.5 中第一个例子中，模型被用于文本情感分类任务，给定一段文本，模型能够判断其情感倾向，识别出文本表达的是积极还是消极情绪。第二个例子展示了数学运算任务，模型会仿照示例的形式，直接给出对应的运算结果。

按照示例数量的不同，上下文学习可以呈现出多种形式：零样本学习（Zero-shot）、单样本学习（One-shot）和少样本学习（Few-shot）<sup>[2]</sup>，如图 3.6 所示。

- **零样本学习**：在此种学习方式下，仅需向模型提供任务说明，而无需提供任何示例。它为实际应用带来了显著的便捷性。但零样本学习性能取决于大模型泛化能力和任务复杂度，面对复杂任务表现可能欠佳。
- **单样本学习**：这种方式仅需为模型提供一个示例，更贴合人类的学习模式，因为人类常常通过一个具有典型性的例子便能初步领会新的任务。不过，单样本学习的效果可能会受到示例的代表性和独特性的影响。
- **少样本学习**：这种方法通过为模型提供少量的示例（通常为几个至十几个），显著提升模型在特定任务上的表现。但在少样本学习中，示例的选取和质量极为关键，直接对模型的学习效果产生影响。

尽管上下文学习在许多任务中表现出色，但它为何能够发挥如此效用仍然是



图 3.6: 上下文学习分类

一个重要的研究问题。对此，斯坦福大学的一项研究 [35] 提供了一种解释——“将上下文学习视为隐式贝叶斯推理”。在大语言模型的预训练阶段，模型从大量文本中学习潜在的概念。当运用上下文学习进行推理时，大语言模型借助演示示例来“锚定”其在预训练期间所习得的相关概念，从而进行上下文学习，并对问题进行预测。以图 3.5 为例，模型之所以能给出正确答案，是因为模型在预训练时已经学习到与情感相关的概念，比如积极情感的表现形式（满意的笑容，手舞足蹈……）、句法结构和句法关系等等，当模型在推理时，借助“浣熊吃了一包麻辣味的干脆面，辣得肚子疼。 \n 消极”等示例“锚定”到情感等相关概念，并基于这些概念，

给出问题答案。

### 3.2.2 演示示例选择

在上下文学习中，我们已经明晰了其多样的形式以及不同形式的特点和适用场景。然而，要实现高效且准确的上下文学习，仅仅了解这些还远远不够。其中一个关键因素便是如何选择演示示例。上下文学习依赖于上下文中的演示示例等信息来引导模型理解任务和生成输出，这使得演示示例的质量和适用性直接影响着学习效果。因此，如何精心挑选出恰当且有效的演示示例，成为了提升上下文学习性能的重要环节，值得深入探讨。

示例选择的两个主要依据是相似性和多样性 [20]：

- **相似性**意味着要精心挑选出与待查询问题最为相近的示例。这种相似性的度量可以基于多种方式，如语言层面的相似性（包括关键字匹配或语义相似度匹配）、结构相似性等等。通过选取相似的示例，能够为模型提供明确的参照，使其更易于理解和处理当前问题。
- **多样性**则要求所选的示例涵盖广泛的内容。这意味着要避免重复的示例，并且最大程度地扩大示例对问题的覆盖范围。多样化的示例能够帮助模型从不同的角度去理解任务，增强其应对各种情况的能力。

除上述依据外，在特定任务中，示例的复杂性等因素也会被纳入考量。本节内容主要围绕**相似性和多样性**这两个关键依据展开。鉴于不同方法对示例选择依据的侧重有所不同，我们参照文献 [20] 的分类，将现有的示例选择策略大致归纳为三类：**一次性检索**、**聚类检索**和**迭代检索**。

#### 1. 一次性检索

一次性检索是目前应用广泛的示例选择策略。其工作原理是，在筛选示例时，检索器依据特定的评分标准对示例进行排序，然后选取排名靠前的  $K$  个示例。一

一次性检索的核心理念就是希望示例与待查询的问题具备相似性。

KATE [17] 是该一次性检索的典型代表，如图 3.7 所示。KATE 利用 RoBERTa 编码器，把示例中除去标签的部分作为输入，将其与待查询的问题转化为向量形式。接着，在向量嵌入空间中，通过计算问题与每个示例输入之间的向量余弦相似距离，检索出距离每个问题最近的前  $K$  个示例输入。之后，将这些示例输入与对应的标签拼接，形成完整的示例，并根据距离从近到远进行排序。

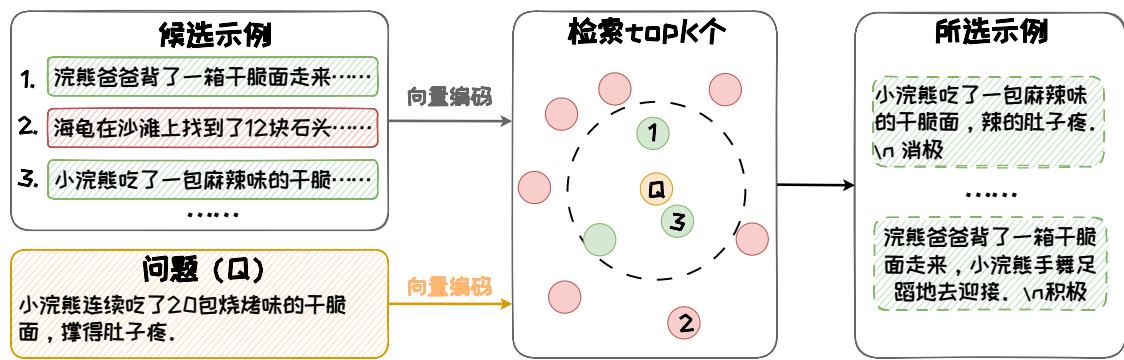


图 3.7: 一次性检索

这种方法虽然操作简单且效率高，但存在一定的缺陷。由于每个示例的选择与其他示例相互独立，可能导致所选的示例缺乏多样性，从而使最终结果趋向同质化，难以形成最优组合。

## 2. 聚类检索

为缓解一次性检索中存在的同质性问题，聚类检索方法把所有示例划分为  $K$  个簇，旨在让相似的示例聚集在一起。而后，检索器依据给定的问题，从每个簇中选取最为相似的示例，最终获取  $K$  个示例。聚类检索的核心要义在于保证所选示例具备多样性。

聚类检索中具有代表性的方法是 Self-Prompting [15]，如图 3.8 所示，Self-Prompting 首先将示例集合和问题编码成向量形式，接着运用 K-Means 算法把示例集合聚为  $K$  个簇。依照问题与示例之间的余弦相似度，从每个簇中选取与问题

最相似的示例，由此得到  $K$  个示例。

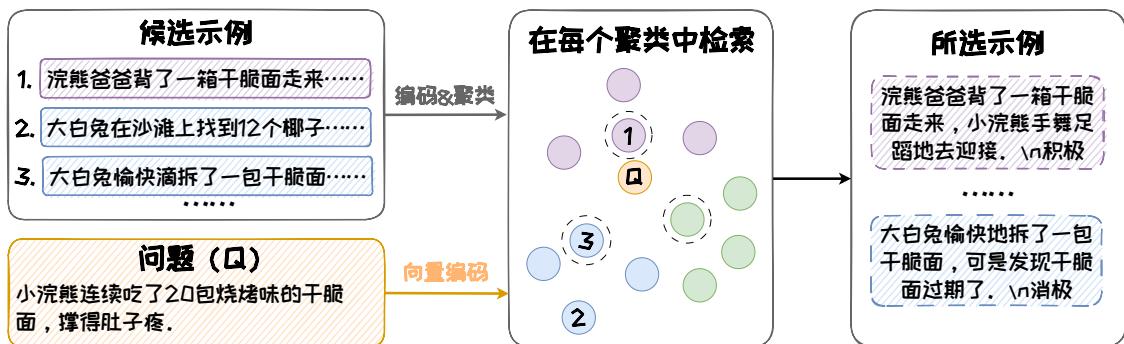


图 3.8: 聚类检索

聚类检索方法通过将示例划分为多个簇，从每个簇中选取一个样本，提高了示例的多样性，但因为有些簇与问题距离很远，这可能会降低问题与示例之间的相似性，一定程度造成选择示例的相似性不够高。

### 3. 迭代检索

在前述的检索策略中，一次性检索虽然操作简单且效率高，但可能导致示例缺乏多样性，从而使结果趋向同质化。而聚类检索虽然提高了示例的多样性，但可能会导致问题与示例的相似性不够高。为了在这示例的**相似性和多样性**之间取得平衡，迭代检索策略应运而生。

在迭代检索策略中，检索过程是迭代的，每个示例的选择依赖于当前的问题和已选的示例。代表性的迭代检索方法是 RetICL [27]。RetICL 将示例选择视为一个马尔可夫决策过程，其中每个状态包含当前任务和已选择的示例序列，每个动作则是选择下一个示例。具体而言，RetICL 利用 LSTM [10] 训练了一个检索器。在检索过程中，首先基于当前问题初始化检索器的内部状态表示，然后选择一个最佳示例。接着，根据当前问题和所选示例构建一个潜在的状态表示，并基于此状态表示选择下一个示例。这一过程会迭代执行，直至达到预定的示例数量  $K$ ，或直至性能无法进一步改善为止。

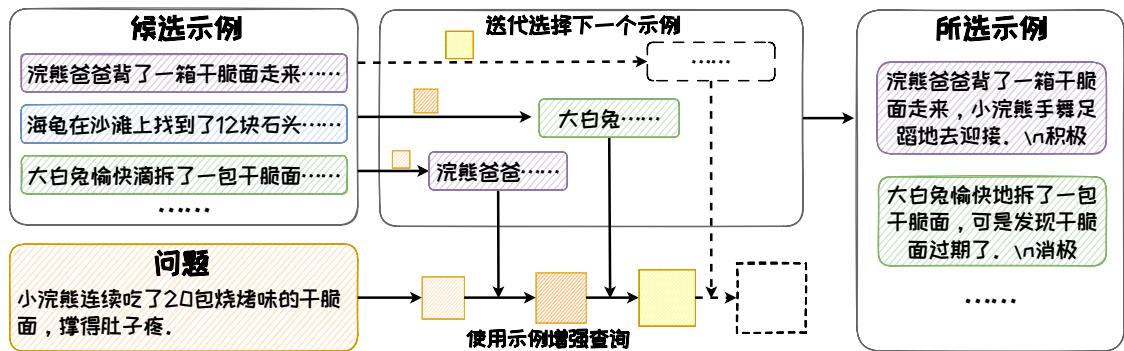


图 3.9: 迭代检索

迭代检索通过动态调整查询和示例选择，有效地平衡了相似性和多样性，提高了上下文学习的性能。尽管迭代检索在计算上相对复杂，但其能够生成更优的示例集，从而在复杂任务中展现出更好的适应性和灵活性。

除了在示例选择策略上有所不同外，现有的示例选择方法在检索器的选择和设计上亦呈现出差异。一些方法采用现成的检索器，而另一些方法则为了追求更卓越的性能，选择在特定语料库上对检索器进行微调。关于检索器的深入介绍及其分类，将在第六章 6.8 中进行详尽阐述。

### 3.2.3 性能影响因素

通过精心设计的上下文学习形式和示例选择策略，我们能够显著提升模型的适应性和灵活性，但上下文学习的性能仍受到多种因素的共同影响。这些因素横跨预训练阶段至实际应用的各个环节，包括预训练数据的质量与多样性、预训练模型的架构与能力，以及演示示例的格式与顺序等 [42]。在接下来的讨论中，我们将深入剖析这些关键因素如何相互作用，共同塑造上下文学习的最终表现。

#### 1. 预训练数据的影响

预训练数据是上下文学习能力的基础，其质量和特性对模型的性能具有深远的影响。以下三个方面是影响上下文学习性能的关键因素：

- **语料库的领域来源**: 预训练数据的领域特性直接影响模型对特定任务的理解和执行能力。跨领域的语料库训练可以使模型具备更广泛的上下文学习能力, 而单一领域的语料库可能限制模型的适应性, 即使该领域与目标任务高度相关, 也无法保证最佳的上下文学习性能 [28]。
- **任务多样性**: 预训练数据中的任务多样性是提升上下文学习性能的重要因素。多样化的任务类型有助于模型学习到更广泛的知识和技能, 增强其泛化能力, 从而在面对新任务时表现更为出色 [26]。
- **训练数据的分布特性**: 训练数据中存在突发性分布和罕见类别时, 对模型上下文学习能力有显著影响。当数据的意义或解释具有动态变化特性时, 模型能够更好地适应和学习, 从而提升上下文学习的表现 [4]。

## 2. 预训练模型的影响

预训练模型对上下文学习的性能影响主要体现在模型规模方面。当模型达到一定的规模基础时, 上下文学习能力便得以显现。通常, 模型规模需达到亿级别及以上的参数, 常见的拥有上下文学习能力的模型中, 规模最小的是来自阿里巴巴通义实验室的 Qwen2-0.5B 模型, 具有 5 亿参数规模。通常而言, 模型的规模越大, 其上下文学习性能也越优越 [33]。大规模模型能够捕捉更复杂的语言模式和知识, 从而在处理多样化的任务时表现更佳。此外, 在实践过程中, 模型的架构和训练策略也是影响上下文学习性能的重要因素。

## 3. 演示示例的影响

尽管预训练模型的训练数据、模型规模和架构等因素对上下文学习能力的影响至关重要, 但最终用户在实际应用中往往无法直接修改这些底层设置。因此, 用户可以通过精心设计演示示例来作为提升上下文学习性能的有效途径。在 3.2.2 节中, 我们已经讨论了选择演示示例的方法, 选择合适的演示示例能够显著提升模型性能。接下来, 我们将深入探讨演示示例本身对上下文学习的影响, 主要体现在

示例格式、输入-标签映射、示例数量及顺序等方面。

**示例格式。**图3.5中展示了一个用上下文学习做情感分类的例子，这里每一条演示示例包含两部分，任务输入以及输出标签，输入和标签一一对应。对于图中情感分类和简单算术运算任务，这里示例格式就是：给定输入，直接给出对应的输出结果，模型学习从任务输入到输出标签的映射。而对于一些复杂推理任务，例如逻辑运算，代码生成等，仅仅通过简单的输入-输出对可能不足以让模型理解任务的复杂性。在这种情况下，需要使用思维链形式的示例表示，即在输入和输出之间添加中间推理步骤，以帮助模型逐步推理并学习到更复杂的映射关系，思维链将在下一章3.3 详细介绍。

**输入-标签映射。**每一个上下文示例包含任务输入和输出标签，模型通过这些示例学习输入到标签的映射。如果示例中的输入到标签映射是错误的，例如在情感分类任务中，将积极的文本错误地标记为消极，模型可能会学习到这种错误的映射方式，从而在下游任务中产生错误。不同模型对标签正确与否的敏感性存在差异，这主要与任务的复杂性和模型规模大小有关 [14, 38]。较大的模型在上下文学习中对标签的正确性表现出更高的敏感性 [14]，因为它们更容易捕捉到输入-标签之间的映射关系。相比之下，较小的模型可能更多地依赖于示例中标签的语义信息，而非精确的输入-标签映射 [24, 33]。

**演示示例的数量和顺序。**增加演示示例的数量通常能够提升上下文学习性能，但随着示例数量的增多，性能提升的速率会逐渐减缓。此外，相较于分类任务，生成任务更能从增加的示例数量中获益 [16]。演示示例的顺序同样是影响上下文学习性能的关键因素，不同示例顺序下的模型表现存在显著差异。最优的示例顺序具有模型依赖性，即对某一模型有效的示例顺序未必适用于其他模型 [19]。同时，示例顺序的选择也受到所使用数据集的特定特性影响 [17]。

综上所述，演示示例的设计对上下文学习性能的影响是多方面的，包括示例

格式、输入-标签映射、示例数量和顺序等。这些因素共同作用于模型的学习过程，决定了模型在特定任务上的表现。此外，Prompt 中包含的任务说明的清晰度和具体性，直接影响模型的理解和执行能力。高质量的任务说明能够为模型提供明确的指导，从而显著提升上下文学习的性能 [38]。因此，在设计演示示例和任务说明时，应综合考虑这些因素，以优化模型的整体表现。

本章详细探讨了上下文学习的定义及分类，示例选择策略和影响因素。首先，我们定义了上下文学习，并区分了零样本、单样本和少样本学习。接着，介绍了三种主要的示例选择策略：一次性检索、聚类检索和迭代检索，分别展示了其优缺点以及代表性方法。随后，分析了影响上下文学习性能的关键因素，包括预训练数据的领域来源、任务多样性、数据分布特性，预训练模型的规模，以及演示示例的格式、输入-标签映射、数量和顺序等。通过对这些方面的深入讨论，本节为提升上下文学习实践效果提供了指导，帮助更好地理解和应用这一自然语言处理新范式。

### 3.3 思维链

随着语言模型规模的持续扩张，其在语义分析、文本分类、机器翻译等自然语言处理任务中的表现显著增强，这得益于模型对语言特征和结构的深入捕捉。然而，在面对算术求解、常识判断和符号推理等需要复杂推理能力的任务时，模型规模的单纯增长并未带来预期的性能突破，这种现象被称作“Flat Scaling Curves”。这表明，仅靠模型规模的扩大不足以解决所有问题，我们需要探索新的方法以提升模型的推理能力和智能水平。

在解决复杂问题时，逐步构建推理路径以导出最终答案的方法至关重要。基于这一理念，Jason Wei 提出了一种创新的 Prompt 范式——思维链提示（Chain-of-Thought, CoT）[32]。该方法通过引导模型进行逐步推理，旨在在复杂任务中实现性能的显著提升，从而突破“Flat Scaling Curves”的限制，激发大模型的内在推理潜能。

#### 3.3.1 思维链提示的定义

思维链提示（Chain-of-Thought, CoT）[32]是通过在提示中嵌入一系列中间推理步骤，引导大型语言模型模拟人类解决问题时的思考过程。这种方法不仅能够在无需特定任务微调的情况下显著提升模型在推理任务上的表现，而且还能够揭示模型在处理复杂问题时的内部逻辑和推理路径。

CoT 方法的一个关键部分是如何构造提示触发大模型一步一步生成推理路径，并生成最终答案。最早期的方法通过构造少量样本示例（Few-Shot Demonstrations）[32]，来引导模型一步一步生成答案。在这些示例中，研究者精心编写任务的推理或解题过程，以此作为模型学习和模仿的基础。这种方法使得模型能够从这些示例中学习如何生成推理步骤，并最终输出答案。通过这种方式，模型不仅能够模仿

人类的思考和推理过程，而且还能逐步提升其解决问题的准确性。

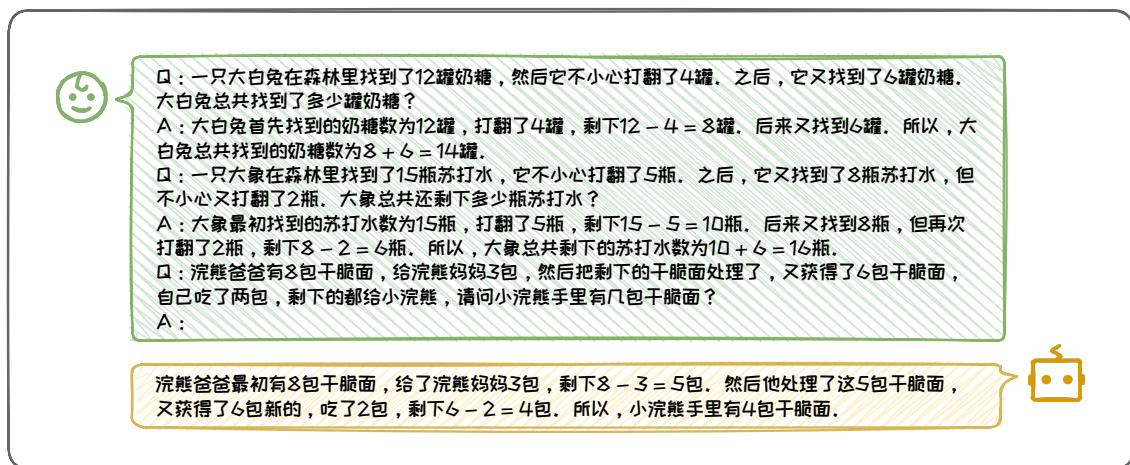


图 3.10: 一个标准的 CoT 提示示例

对于图 3.10 中的数学问题，CoT 样例会先给出解答数学问题的每一个步骤，一步一步验算得到最后的结果。通过 CoT 形式的 Prompt，大模型在数学、推理问题上性能显著提升，并且随着模型规模变大，提升效果显著变高。

目前在传统的 CoT 方法上，出现了许多扩展的方法，这些扩展的方法按照其推理方式的不同，可以归纳为三种模式：按部就班、三思后行和集思广益。这几种模式的对比如图 3.11 所示。

- **按部就班.** 按部就班模式强调的是逻辑的连贯性和步骤的顺序性。在这种模式下，模型像是在遵循一条预设的路径，每一步都紧密依赖于前一步的结论。这一类方法以标准的 CoT [32]、Zero-Shot CoT [12]、Auto-CoT [41] 等为代表，它们会一步一步生成推理链条，每一个推理步骤都是在上一个步骤之后生成的。
- **三思后行.** 三思后行模式则更像是在探索一片未知的森林，模型在每一步都会停下来评估周围的环境，判断是否需要调整方向。这一类方法以 ToT [37]、GoT [1] 为代表，他们生成推理内容时，允许模型在推理过程中进行自我修正，它像是在寻找最佳路径的同时，也在不断地评估和调整策略。

- **集思广益.** 集思广益模式则是将多个独立的推理路径汇聚起来，通过集体智慧来达成最终的答案。这一类方法以 Self-Consistency [30] 为代表，它让大模型同时生成多条推理路径，得到多个结果，然后将最后的结果汇聚起来，得到一个最终的答案。这种方法像是在召开一场智者的会议，每个智者都带来了自己的见解，最终通过讨论和整合，得出一个更为全面和准确的结论。

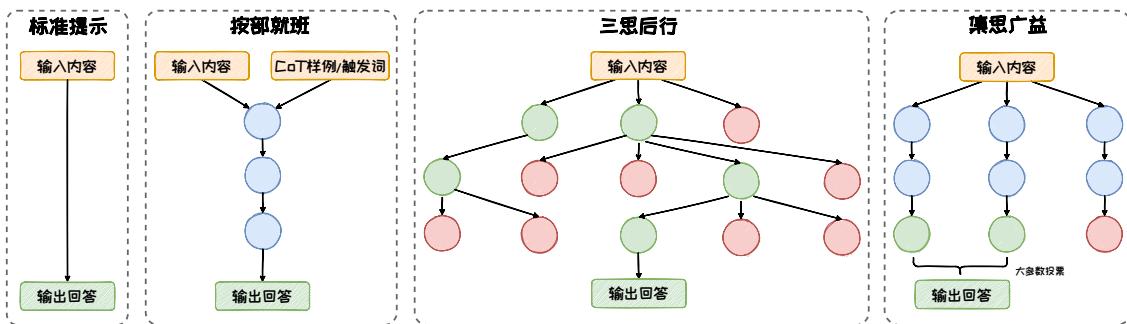


图 3.11: 不同 CoT 结构的对比

### 3.3.2 按部就班

按部就班模式强调的是逻辑的连贯性和步骤的顺序性。在这种模式下，模型像是在遵循一条预设的路径，每一步都紧密依赖于前一步的结论。这种模式的核心在于确保推理过程的清晰和有序，使得模型的决策过程更加透明和可预测。

按部就班模式中最经典的方法就是基础的思维链 (CoT) 方法，它通过手工构造几个一步一步推理回答问题的例子，作为模型的 Few-Shot 示例，来引导模型一步一步生成推理步骤，并生成最终的答案。这种方法在提升模型推理能力方面取得了一定的成功，但是需要手工编写大量 CoT 示例，这非常费时费力，并且 CoT 编写的好坏也会显著影响模型的性能。针对此问题，研究者在 CoT 基础上的做了延伸，本节将介绍 CoT 的两种变种：Zero-Shot CoT 和 Auto-CoT。

#### 1. Zero-Shot CoT

Zero-Shot CoT [12]，它无需手工标注 CoT 的示例，减少了对人工示例的依赖。它通过简单的提示，如“Let's think step by step”，引导模型自行生成推理链。这种方法不仅简化了模型的使用过程，而且在多个推理任务上展现出了与传统 CoT 相媲美甚至更优的性能。

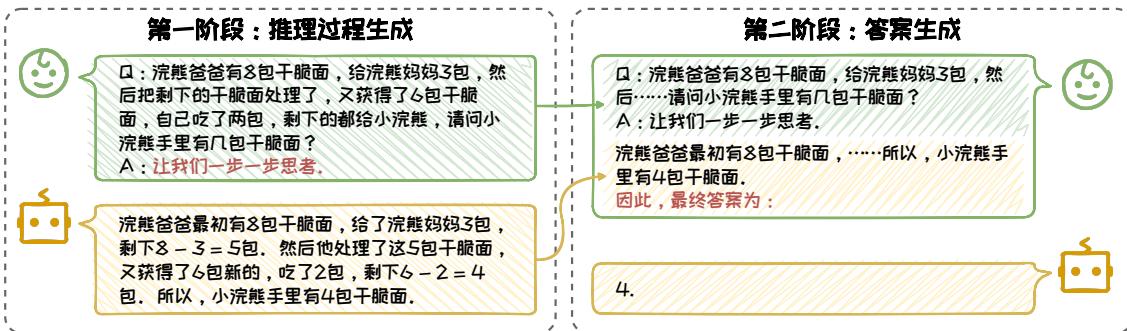


图 3.12: Zero-Shot CoT 提示的流程

Zero-Shot CoT 整体流程如图 3.12 所示，它使用两阶段方法来回答问题。首先，在第一阶段，在问题后面跟上一句“让我们一步一步思考”或者“Let's think step by step”来作为 CoT 的提示触发词，来指示大模型先生成中间推理步骤，再生成最后的答案。在第二阶段，把原始的问题以及第一阶段生成的推理步骤拼接在一起，在末尾加上一句“Therefore, the answer is (因此，最终答案为)”，把这些内容输给大模型，让他输出最终的答案。通过这样的方式，无需人工标注 CoT 数据，即可激发大模型内在的推理能力。大模型能够逐步推理出正确的答案，展现了 Zero-Shot CoT 在提升模型推理能力方面的潜力。

## 2. Auto CoT

传统的 CoT 方法通过手工制作的特定任务示例来构建推理链，这个过程既耗时又复杂。随着任务类型的多样化，这种方法的维护成本和设计难度也随之增加。Zero-Shot CoT 尝试通过自动化提示来提高大型语言模型（LLM）生成推理内容的能力，但在面对复杂任务时可能会产生错误的推理链，其性能仍有待提升。为了克

服传统 CoT 方法的局限性和 Zero-Shot CoT 在处理复杂任务时可能出现的不足，研究者们提出了一种名为 Auto-CoT [41] 的创新技术。Auto-CoT 技术旨在通过自动化和智能化的手段，减少对手工制作的特定任务示例的依赖，同时提升模型在复杂推理任务中的准确性和效率。

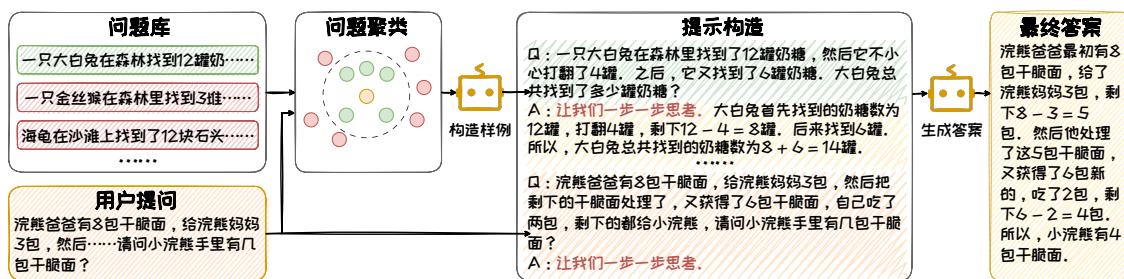


图 3.13: Auto-CoT 提示的流程

Auto-CoT 技术的核心在于利用大型语言模型（LLM）自身的生成能力，通过一系列精心设计的自动化步骤，生成与特定任务相关的推理链。该方法的流程如下图 3.13 所示：

- 首先通过分析大量任务样本，利用聚类技术筛选出与用户提问相似的问题。
- 然后，借助 Zero-Shot CoT 的方式，为这些问题生成推理链，形成少量样本（Few-Shot）示例。这些少量样本示例包含了不同问题及其对应的推理内容，它们为模型提供了学习不同问题之间共性和差异性的机会。
- 在此基础上，Auto-CoT 引导 LLM 以 CoT 的形式生成针对用户问题的推理链和答案。

这种方法不仅避免了人工标注 CoT 样本的高昂成本，而且通过智能化的样本生成和学习过程，实现了超越传统 CoT 方法的性能。

### 3.3.3 三思后行

三思后行模式强调的是在决策过程中的审慎和灵活性。在这种模式下，模型在每一步都会停下来评估当前的情况，判断是否需要调整方向。这种模式的核心在于允许模型在遇到困难或不确定性时进行回溯和重新选择，确保决策过程的稳健性和适应性。

以上介绍的几种 CoT 方法以从头走到尾的形式将思维过程展现出来，并最终给出良好的回答。但是在实际生活中，我们人类做思考的时候，有一个反复选择回溯的过程，通常会从多个候选答案中选择最好的那个，并且如果一条思维路子走不通，就会回溯到最开始的地方，选择另一种思维路子进行下去。基于这种现实生活的观察，研究者在 CoT 的基础上提出了思维树（Tree of Thoughts, ToT）[37]、思维图（Graph of Thoughts, GoT）[1] 等其他 CoT 变种。

ToT 从四个方面出发，提出四个如何构造思维树的问题：

- 如何拆解思维过程。一个复杂问题可以拆分成多个简单子问题，每个子问题的解答过程对应一个思维过程。一般来说思维应该足够小，以便大模型能生成多样化的样本，但是也不能过于小（例如小到 Token 级别），否则大模型难以评估该思维状态。
- 如何从每一个状态生成思维内容。模型需要根据当前状态需要生成可能的下一步思考。
- 如何启发式地评估状态。需要有额外状态评估模块评估已有的思维内容在解决问题方面的进展，并确定哪些状态值得进一步探索。
- 用哪种搜索算法。这一步骤的目标是通过系统化的方法，从一个或多个当前状态出发，寻找通往问题解决方案的路径。

一个 ToT 的具体实例如图 3.14 所示，该任务是给定 4 个数字，然后让大模型来

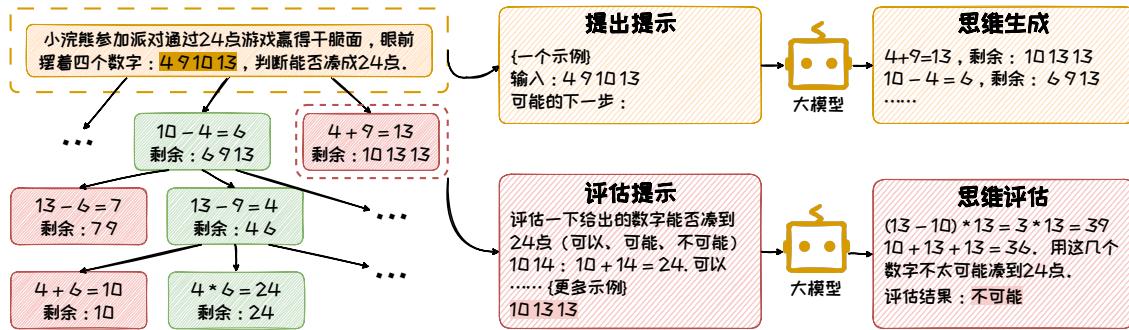


图 3.14: ToT 提示的流程

用利用这四个数字组合“+\*-/”符号来使得最终的运算结果为 24 点。首先，ToT 通过“提出提示”(Propose Prompt)，来基于已有的动作选择，生成多种下一步可能的选择，在图上表现为其多种子节点。在生成完新的动作路径之后，对每一条生成的路径使用“评估提示”(Value Prompt) 来评估当前动作路径是否合理，如果不合理则该节点停止搜索，否则就沿着该节点继续往下探索。不断重复上述两个步骤，直到得出最终合理的结果。

GoT 则是在 ToT 的基础上做了进一步的扩展，提供了每个思维自我评估修正以及思维聚合的操作，从而将推理过程建模为一个有向图，顶点代表某个问题（初始问题、中间问题、最终问题）的一个解决方案，有向边代表使用“出节点”作为直接输入构造出的思维“入节点”，具体思维的形式取决于用例。

### 3.3.4 集思广益

集思广益模式强调的是通过汇集多种不同的观点和方法来优化决策过程。在这种模式下，模型不仅仅依赖于单一的推理路径，而是通过探索多种可能的解决方案，从中选择最优的答案。这种方法借鉴了集体智慧的概念，即通过整合多个独立的思考结果，可以得到更全面、更准确的结论。

以往的 CoT 方法重点关注了如何引导大模型在回答时构建合理的推理链条。

研究者在 CoT 的基础上探讨了如何通过自洽性来增强模型的推理能力，提出了 Self-Consistency [30] 方法，其关键在于引入多样性的推理路径并从中选择最一致的答案，从而提高了模型的推理准确性。Self-Consistency 方法的一个显著优势在于其通用性。它不依赖于特定的 CoT 形式，可以与其他 CoT 方法无缝结合，共同作用于模型的推理过程。

如图 3.15 所示，Self-Consistency 的实现过程可以分为以下几个步骤：

- 使用 CoT 或 Zero-Shot CoT 的方式来引导大模型针对问题生成一系列推理路径。这一步不是简单地采用贪婪解码，而是采用随机采样策略，如温度采样或核采样，以鼓励模型探索多种可能的推理路径。
- 针对大模型生成的每个推理内容，收集其最终的答案，并统计每个答案在所有推理路径中出现的频率。
- 选择出现频率最高的答案作为最终的、最一致的答案。这个过程模拟了人类在面对多个可能性时，倾向于选择最一致的解释或解决方案的直觉。

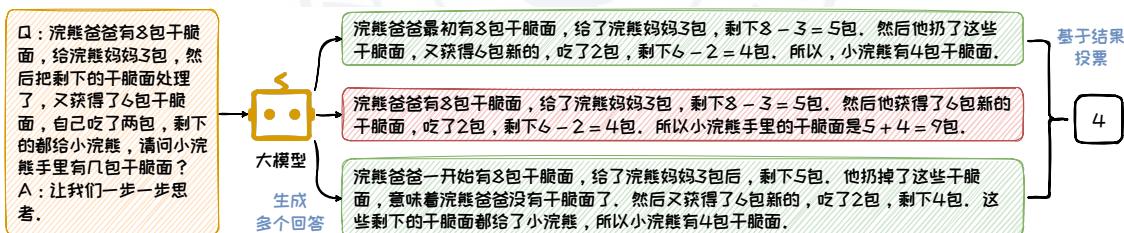


图 3.15: Self-Consistency 的流程

本节探讨了思维链 (Chain-of-Thought, CoT) 方法在提升大规模语言模型推理能力方面的重要性和应用。CoT 的核心思想是在提示中模拟人类解决问题的思考过程，从而在无需特定任务微调的情况下显著提升模型在推理任务上的表现。

CoT 方法包含了多种变种，以适应不同的推理需求和任务复杂性：

- **按部就班模式：**强调逻辑连贯性和步骤顺序性，通过逐步构建推理链条最终得出答案。这种模式包括经典的 CoT、Zero-Shot CoT 和 Auto-CoT 等。

- **三思后行模式**: 在每一步推理过程中进行自我评估和修正, 确保决策的稳健性和灵活性。典型方法包括 ToT 和 GoT 等。
- **集思广益模式**: 通过汇集多种不同的推理路径, 选择最一致的答案, 从而优化决策过程。Self-Consistency 方法即是通过生成多条推理路径并选择最一致的答案来提升推理准确性。

这些模式和方法通过不同的策略和机制, 增强了大模型在复杂任务中的推理能力, 展现了思维链提示在自然语言处理领域的重要应用前景。通过这些方法, 大模型不仅能够模拟人类的思考过程, 还能够在实际任务中提供更加准确和可靠的解决方案。



## 3.4 Prompt 技巧

在先前的章节中，我们详细阐述了 Prompt 工程中的两项关键技术：上下文学习与思维链。上下文学习使得模型能够从有限的示例中快速学习并适应新任务，而思维链则通过逐步推理的方式，增强了模型解决逻辑推理问题的能力。在这些技术的基础上，针对 Prompt 的版式等进行设计优化，可以进一步改善生成质量。

本章将从 Prompt 的编写规范，合理归纳提问，适时使用 CoT，善用心理暗示方面介绍 Prompt 设计优化技巧，从而构建清晰、有效的 Prompt，以最大化模型的生成内容质量。

### 3.4.1 规范 Prompt 编写

在与 LLMs 的交互中，规范的 Prompt 是我们与 LLMs 构建有效沟通的基础，经典的 Prompt 通常由任务说明，上下文，问题，输出格式等部分中的一个或几个组成。以图 3.16 中这个情感分类的 Prompt 为例：



图 3.16: 经典的 Prompt 示例

在这个例子中：

- 任务说明是“**### 判断下面句子的情感为积极还是消极。**”，它明确了模型需要完成的任务；
- 上下文是“**针对此问题有如下示例：** # **示例 1：**浣熊爸爸背了一箱干脆面走来，小浣熊手舞足蹈迎接爸爸。 \n **积极** \n # **示例 2：**小浣熊吃了一包麻辣味的干脆面，辣得肚子疼。 \n **消极** \n # **示例 3：**小浣熊吃了一包烧烤味的干脆面，脸上露出了满意的笑容。 \n **积极** \n”。上下文提供了帮助模型理解和回答问题的示例或背景信息；
- 问题是“**待分类的句子：小浣熊连续吃了 20 包烧烤味的干脆面，撑得肚子疼。**”，是用户真正想要模型解决的问题，它可以是一段段落（比如摘要总结任务中被总结的段落），也可以是一个实际的问题（比如问答任务中用户的问题），或者表格等其他类型的输入内容；
- 输出格式是“**以”结果：“”**的 JSON 格式返回最终结果。”，它规范了模型的输出格式。

通过上面这个例子可以看出，在编写经典 Prompt 的过程中，Prompt 各个组成部分都很重要，编写它们时是否规范，都会直接影响模型的输出质量。同时，各个组成部分的排版也很重要，所以接下来我们将详细介绍经典 Prompt 的规范编写需要满足的要求：

- **任务说明要明确：**确保模型清楚理解任务目标。任务说明应简洁明了，避免使用模糊或复杂的语言，以便模型能够准确捕捉任务的核心要求。
- **上下文丰富且清晰：**提供足够的背景信息，帮助模型更好地理解任务。上下文应可以是相关的示例、数据或情境描述，这些信息有助于模型建立正确的理解框架。
- **输出格式要规范：**明确输出格式，确保模型输出的结果符合预期。输出格式应具体指定，如使用特定的数据结构（如 JSON、XML）或文本格式，以便

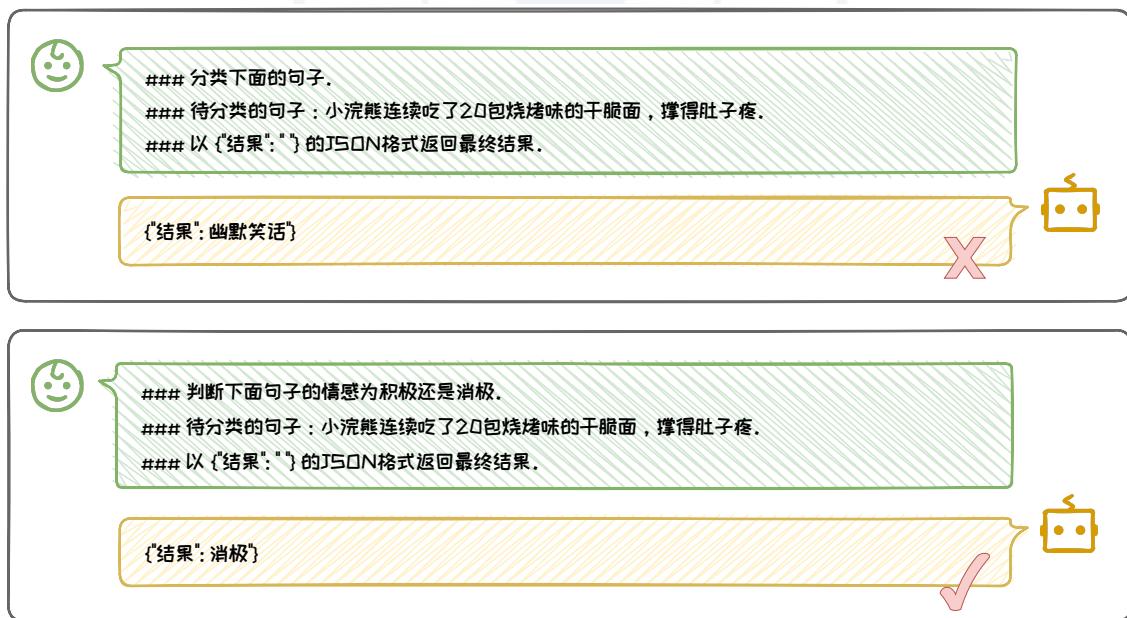
于后续的数据处理和分析。

- **排版要清晰：**良好的排版有助于提高 Prompt 的可读性和模型的理解效率。使用清晰的标题、子标题和列表，以及适当的空白和缩进，可以使 Prompt 更加易于阅读和理解。

## 1. 任务说明要明确

明确的任务说明是构建有效 Prompt 的关键要素之一。一个清晰、具体的任务说明能够确保模型准确理解任务要求，并产生符合预期的输出。例如，在情感分类任务中，任务说明“判断下面句子的情感为积极还是消极。”就是一个明确的示例，它清晰地定义了任务类型（情感分类）和分类的具体类别（积极或消极）。

相反，模糊或不明确的任务说明可能导致模型误解用户的真实意图，从而产生不符合预期的输出。如图 3.17 所示，“分类下面的句子”这样的任务说明就缺乏具体性，没有明确指出分类的类型和类别，使得模型难以准确执行任务。



- **使用明确的动词**: 选择能够清晰表达动作的动词, 如“判断”、“分类”、“生成”等, 避免使用模糊的动词如“处理”或“操作”。
- **具体的名词**: 使用具体的名词来定义任务的输出或目标, 例如“积极”和“消极”在情感分类任务中提供了明确的分类标准。
- **简洁明了**: 任务说明应简洁且直接, 避免冗长或复杂的句子结构, 使模型能够快速抓住任务的核心要求。
- **结构化布局**: 在较长的 Prompt 中, 将任务说明放置在开头和结尾, 因为模型通常更关注这些部分的信息 [18]。这种布局有助于确保模型首先和最后接触到的是最关键的任务信息。

通过这些策略, 我们可以确保任务说明既清晰又具体, 从而帮助模型更好地理解和执行任务, 最终产生高质量的输出。

### 2. 上下文丰富且清晰

在 Prompt 设计中, 上下文的作用不容忽视, 它有时直接决定了模型能否给出正确的答案。一个丰富且清晰的上下文能够显著提升模型的理解和回答准确率。

**上下文的丰富性**体现在其内容的多样性和相关性。上下文可以包括与问题直接相关的背景信息、具体的演示示例, 或是对话的连续性内容。例如, 在情感分类任务中, 提供具体的示例句子及其对应的情感标签, 可以帮助模型更好地理解任务的具体要求和预期的输出。而在其他类型的任务中, 如文本生成或问答, 提供相关的背景信息或先前的对话内容, 可以帮助模型更准确地把握任务的上下文和用户的意图。

**上下文的清晰性**则要求上下文信息必须与问题紧密相关, 避免包含冗余或不必要的信息。清晰的上下文应直接指向任务的核心, 减少模型在处理信息时的混淆和误解。例如, 在问答任务中, 上下文应仅包含与问题直接相关的信息, 避免引入可能误导模型的无关内容。

在图 3.18 两个上下文设计的例子中，第一个例子的上下文紧密围绕问题，提供了丰富的直接相关信息，没有任何冗余内容。这种设计有助于模型迅速聚焦于关键信息，从而准确回答问题。相比之下，第二个例子的上下文不够丰富，并且单个例子则包含了大量与问题无关的细节，这些冗余信息不仅使上下文显得不明确，还可能加重模型处理信息的负担，导致模型难以准确把握问题的核心，进而影响其回答的准确性。

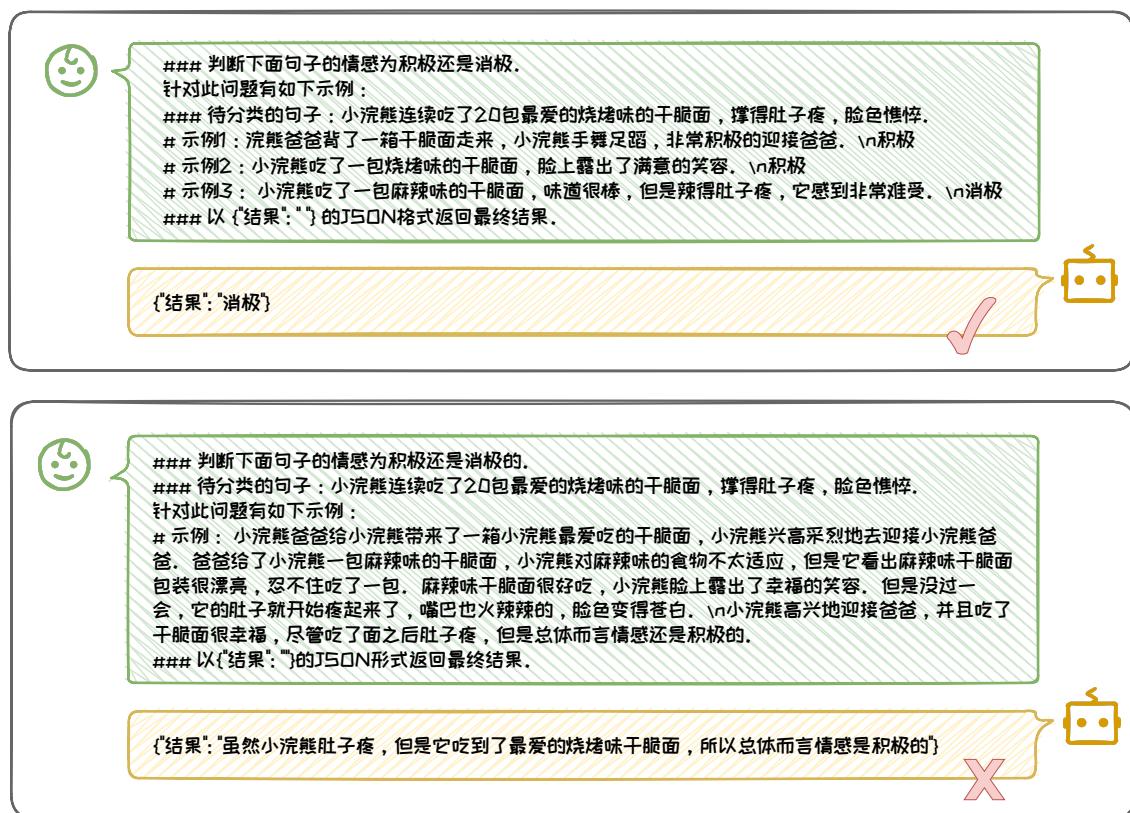


图 3.18: 不同的上下文对比

### 3. 输出格式要规范

规范的输出格式对于确保模型输出的可用性和准确性至关重要。通过指定明确的输出格式，可以使模型的输出结构化，便于下游任务直接提取和使用生成内容。常用的输出格式包括 JSON、XML、HTML、Markdown 和 CSV 等，每种格式都有其特定的用途和优势。

例如，在图 3.19 中的 Prompt 例子中，“以 {”结果”: ””} ” 的 JSON 格式返回最终答案。”明确指定了答案应以 JSON 格式输出，并且以一个简短的例子指名 JSON 中的关键字。这种规范的输出格式不仅使得结果易于解析和处理，还提高了模型输出的准确性和一致性。如果不明确规定输出格式，模型可能会输出非结构化或不规范的结果，这会增加后续处理的复杂性。在第二个例子中，如果模型输出的答案是一个自由格式的文本字符串，那么提取具体信息就需要进行复杂的字符串解析，而不是像 JSON 等结构化格式那样可以直接提取，这就给后续对于结果的处理与使用带来了麻烦。

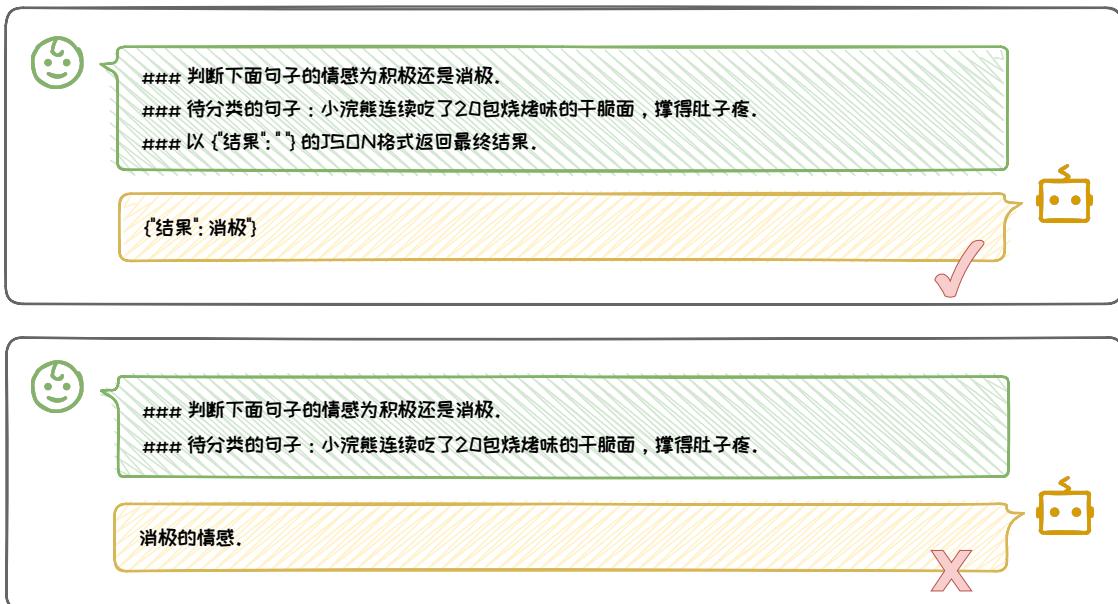


图 3.19: 不同输出格式对比

为了确保输出格式的规范性，可以采取以下措施：

- **明确指定输出格式：**在 Prompt 中明确指出希望模型使用的输出格式，如“请以 JSON 格式返回结果”，并且选择广泛接受和易于处理的输出格式，如 JSON、CSV 等，易于解析和数据交换。
- **提供输出格式的示例：**在 Prompt 中提供一个输出格式的具体示例，比如在 JSON 中明确指出关键字，帮助模型理解预期的输出结构。

通过这些措施，我们可以使得模型的输出既规范又易于处理，从而提高整个系统的效率和准确性。规范的输出格式不仅简化了数据处理流程，还增强了模型输出的可靠性和一致性，为用户提供了更加流畅和高效的交互体验。

#### 4. 排版要清晰

一个优秀的 Prompt 必然具备清晰的排版，这对于模型的理解和性能至关重要。清晰的排版有助于模型准确捕捉任务的关键信息，从而提高其执行任务的准确性和效率。相反，复杂的排版可能会导致信息模糊，使模型难以准确理解任务的具体要求 [8]，进而影响输出结果的质量。

清晰的排版通常涉及使用合适的分隔符和格式化技巧，将 Prompt 的不同组成部分（如任务说明、上下文、问题和输出格式）明确区分开来。在最初的 Prompt 示例中，使用“#”和“###”以及换行符有效地将各个部分分隔开，使得每个部分的内容清晰可见，便于模型理解和处理。相反，如果排版混乱，例如在 Prompt 中混合使用不同的分隔符或不使用任何分隔符，模型可能会混淆不同部分的内容，导致无法准确执行任务，我们去除图 3.16 中的 Prompt 例子的排版符号与样式，使得 Prompt 的排版变得复杂混乱，得到模型的回答如图 3.20 所示，很显然模型回答的质量大幅度下降，尽管内容的信息量是一样的。

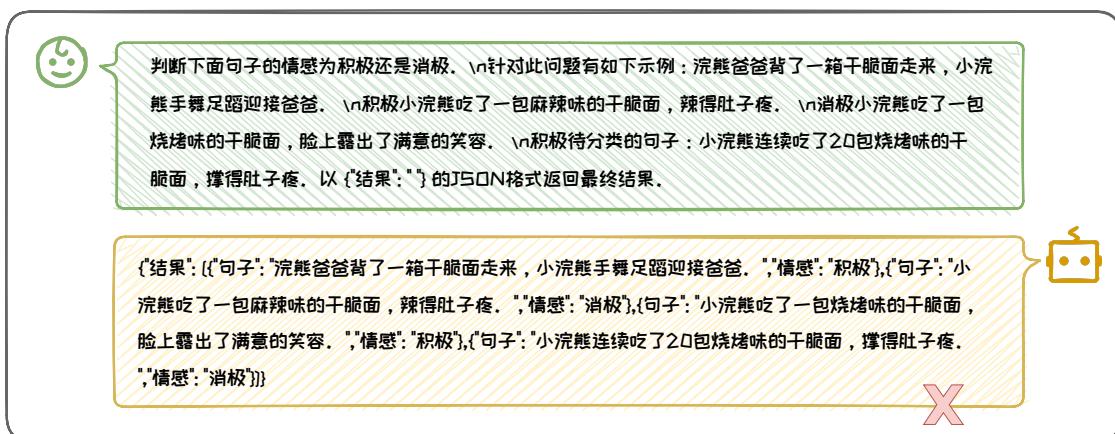


图 3.20：不清晰的排版

为了确保 Prompt 的排版清晰，可以采取以下措施：

- **使用一致的分隔符**：选择并坚持使用一种或几种分隔符（如“#”、“###”、“—”等），以区分不同的 Prompt 部分。
- **合理使用空白和缩进**：通过增加空白行和适当的缩进，增强 Prompt 的可读性，帮助模型区分不同的内容块。
- **清晰的标题和子标题**：为每个部分提供清晰的标题或子标题，使模型能够快速识别每个部分的主题。

通过这些措施，我们构造既清晰又易于理解的 Prompt 的排版，从而帮助模型更好地执行任务，提升信息处理的效率和准确性。

### 3.4.2 合理归纳提问

在与大模型的交互中，提问扮演着至关重要的角色。它不仅是连接我们与模型之间信息的桥梁，更是激发模型深入思考、提供精确答案的关键。通过精心设计的提问，我们能够明确表达需求，引导模型聚焦于核心问题，并从中获取宝贵的见解和解决方案。

在深入探讨了“规范 Prompt 编写”以确保交互的清晰性和有效性之后，我们将进一步拓展讨论，探索如何通过“合理归纳提问”来提升交互的质量。在这一部分，我们将重点介绍两个高级提问策略：“复杂问题拆解”和“追问”。这两个策略是提问艺术的精髓，它们不仅帮助我们深入理解问题的本质，还显著提高了我们获取信息和解决问题的效率。通过这些策略，我们能够更精确地引导模型，确保其输出既准确又全面，从而满足我们的需求。

#### 1. 复杂问题拆解

面对复杂问题时，采用分而治之的策略，即将问题细化为更小、更易管理的部分，是一种高效的解决方式。这种方法不仅有助于我们更清晰地理解问题的各个

层面，而且能够促进更有效解决方案的产生。在计算机算法设计中，这种策略被称为“分而治之”，它基于一个核心理念：通过逐一解决小问题，最终能够解决整个大问题。

在与大模型交互时，我们同样可以使用分治的思想，将复杂问题分解成一系列小问题，并逐一引导模型回答。这样做可以确保每个子问题都得到充分的考虑和解答，如图 3.21 所示。随后，将这些子问题的答案汇总，形成最终的全面回答。这一过程包括两个关键步骤：**分布引导**和**归纳总结**。

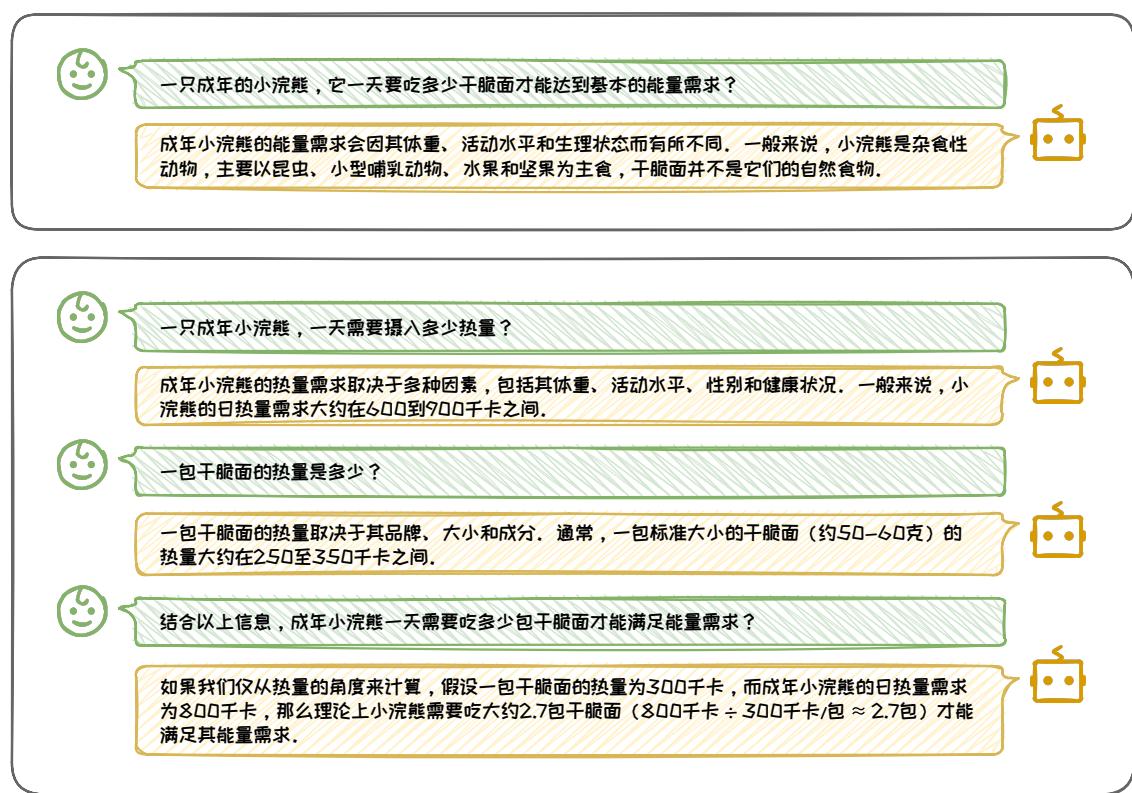


图 3.21：复杂问题拆解例子对比

在上述示例中，用户提出了一个关于成年小浣熊一天需要吃多少干脆面才能满足能量需求的问题。通过分布引导，我们将这个问题分解为两个关键的小问题：“一只成年小浣熊，一天需要摄入多少热量？”和“一包干脆面的热量是多少？”。模型分别回答了这两个问题，提供了小浣熊的日热量需求和干脆面的热量含量。

随后，通过归纳总结的提示“结合以上信息，成年小浣熊一天需要吃多少包干脆面才能满足能量需求？”，我们将这些分散的信息整合起来，计算出小浣熊需要摄入的干脆面数量。这一过程不仅展示了如何通过逐步提问来引导模型提供详细信息，还强调了在解决复杂问题时，系统地分解问题和整合答案的重要性。

这种方法的优势在于它能够帮助用户和模型更有效地处理复杂信息，确保每个细节都被考虑到，并最终形成一个准确和全面的答案。通过分布引导和归纳总结，我们能够有序地解决复杂问题，提供高质量的解答。

### 2. 追问

追问允许用户深入挖掘大模型的输出内容，以获得更具体、详细的信息。这种对话形式的交互不仅促进了更深层次的理解和更丰富的讨论，而且帮助我们细化问题，更精确地表达我们真正想要了解的内容，从而能够更好地指导模型的思考，使其输出更加贴合我们的需求。从追问的形式和目的角度来看，追问可以从以下三个方面来进行：**深入追问、扩展追问、反馈追问**。

#### (1) 深入追问

深入追问的形式是针对模型输出内容做细致探究，其目的在于深入挖掘特定话题的深层信息，以满足用户对知识深度的好奇心和需求。这种追问特别适用于那些需要对某个概念、现象或过程有详尽解释的场景。通过深入追问，用户能够引导模型提供更为细致和深入的信息，从而加深对特定领域的理解。例如，在探讨小浣熊的饮食习惯时，深入追问可以帮助我们了解小浣熊是否适合食用干脆面，以及如何在不损害其健康的前提下给小浣熊喂食干脆面。如图 3.22 是一个深入追问的实例。

在这个例子中，用户首先询问了小浣熊是否可以食用干脆面。模型基于干脆面的成分给出了初步的回答，指出其对小浣熊的健康可能有害。随后，用户通过深入追问，探讨了如果去除干脆面中的调味料和添加剂，是否可以安全地喂食小浣

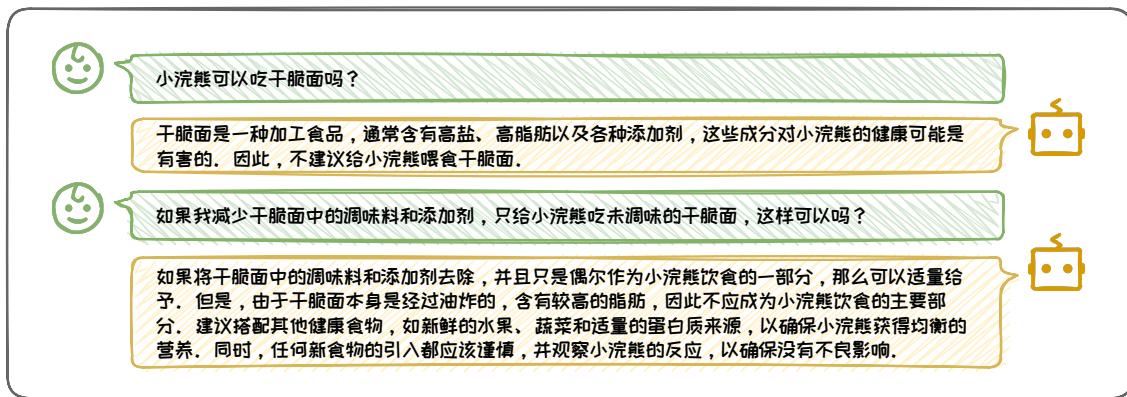


图 3.22: 深入追问示例

熊。模型进一步解释了即使去除调味料和添加剂，干脆面仍含有较高的脂肪，因此不应成为小浣熊饮食的主要部分，并建议了如何搭配其他食物以确保营养均衡。这一系列的追问不仅展示了用户如何通过连续提问引导模型提供实用信息，还强调了在处理复杂问题时寻求专业建议的重要性。

## (2) 扩展追问

扩展追问是一种在已有信息基础上，要求模型提供更多相关信息或例子的提问方式，其目的在于拓宽讨论的广度，收集更多数据、例证或选项，帮助用户获得更广泛的视角，增加对话题的理解。这种追问特别适用于需要全面了解一个主题的场景。例如，在讨论小浣熊的饮食多样性时，扩展追问可以帮助我们了解小浣熊是否适合食用干脆面，以及干脆面在小浣熊饮食中的适宜性。

在图 3.23 这个例子中，用户首先询问了小浣熊的常规饮食，模型提供了小浣熊作为杂食性动物的饮食概况。随后，用户通过扩展追问，探讨了小浣熊是否可以食用干脆面。模型进一步解释了干脆面可以作为小浣熊饮食的一部分，但强调了控制份量和保持饮食多样性的重要性。这一系列的追问不仅展示了用户如何通过提问获取更多关于小浣熊饮食的信息，还强调了在引入新食物时咨询专业意见的必要性。通过扩展追问，用户能够获得更全面的视角，更好地理解小浣熊的饮食需求和限制。

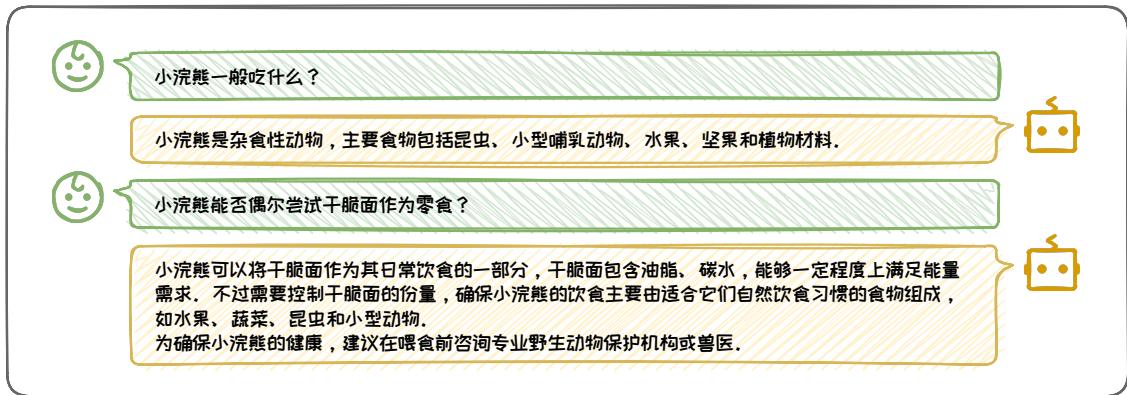


图 3.23: 扩展追问示例

### (3) 反馈追问

反馈追问的形式是在模型的输出不符合预期或存在错误时，提供反馈，指出问题所在，并请求更正或澄清。其目的在于通过反馈机制提升模型的准确性，确保信息的正确性。这种追问允许用户指出模型输出中的具体错误或不足，并请求修正，对于提高对话质量至关重要。

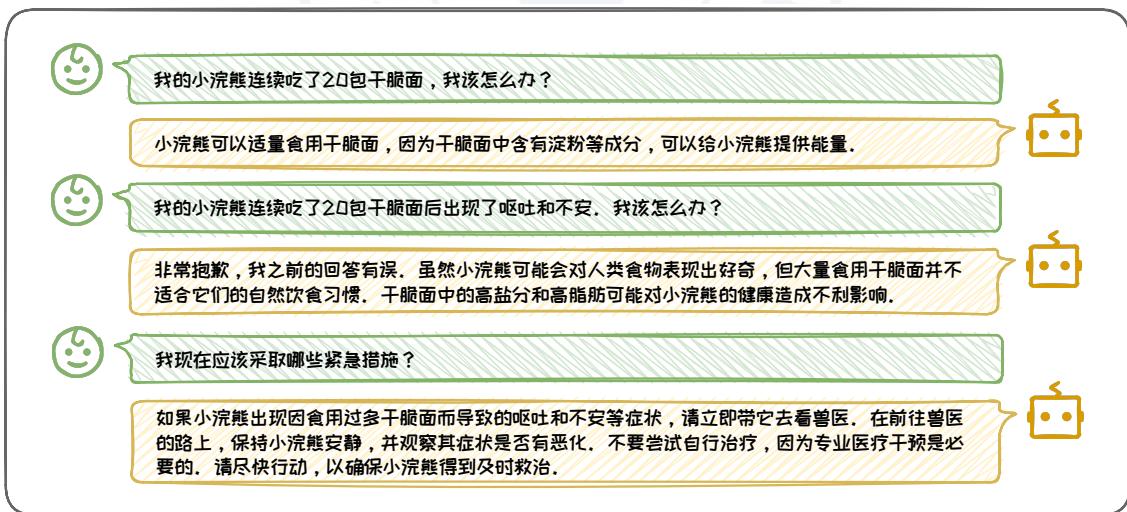


图 3.24: 反馈追问示例

在图 3.24 这个例子中，用户首先询问了小浣熊连续食用 20 包干脆面后的应对措施。模型最初的回答建议小浣熊可以适量食用干脆面，但用户通过反馈追问指

出了小浣熊出现了不良反应。模型随后修正了回答，指出大量食用干脆面对小浣熊的健康有害，并提供了紧急处理建议。这一系列的反馈追问不仅展示了用户如何通过提问纠正模型的错误，还强调了在处理紧急情况时寻求专业医疗建议的重要性。通过反馈追问，用户能够获得更加准确可靠的信息。

### 3.4.3 适时使用 CoT

思维链技术 (Chain of Thought, CoT) [32] 是在处理需要复杂推理的任务时的理想选择，这类任务通常涉及算术、常识和符号推理；在此过程中，模型需要理解和遵循中间步骤以得出正确答案。我们在 3.3 节中已经讨论过了 CoT 几种经典范式，这一节我们会讨论在什么样的场合去使用 CoT 以及如何使用 CoT。

#### 1. 何时应用 CoT

以 Zero-Shot CoT 为例，是否使用 CoT 提示技术，取决于所需要完成的任务类别、模型规模以及模型能力。

在任务类别的考量上，CoT 技术特别适用于那些需要复杂推理的任务，这些任务往往涉及算术、常识和符号推理。在这些领域，CoT 技术能够有效地引导大型语言模型 (LLMs) 生成一系列逻辑严密、条理清晰的中间推理步骤，从而提高得出正确答案的概率，如图 3.25 所示。而对于情感分类、常识问答等简单问题，标准的提示方法已经能够做的足够好，使用 CoT 难以提升效果，反而会额外生成其他内容。

在模型规模的考量上，CoT 技术在应用于参数量超过千亿的巨型模型时，能够显著提升其性能。例如，PaLM [6] 模型和 GPT-3 [3] 模型等，便是此类巨型模型的典范。然而，在规模较小的模型上应用 CoT 技术可能会遭遇挑战，如生成逻辑不连贯的思维链，或导致最终结果的准确性不如直接的标准提示方法，如图 3.26 所示。

在模型能力的考量上，CoT 推理能力的涌现可能与其训练数据密切相关。巨型

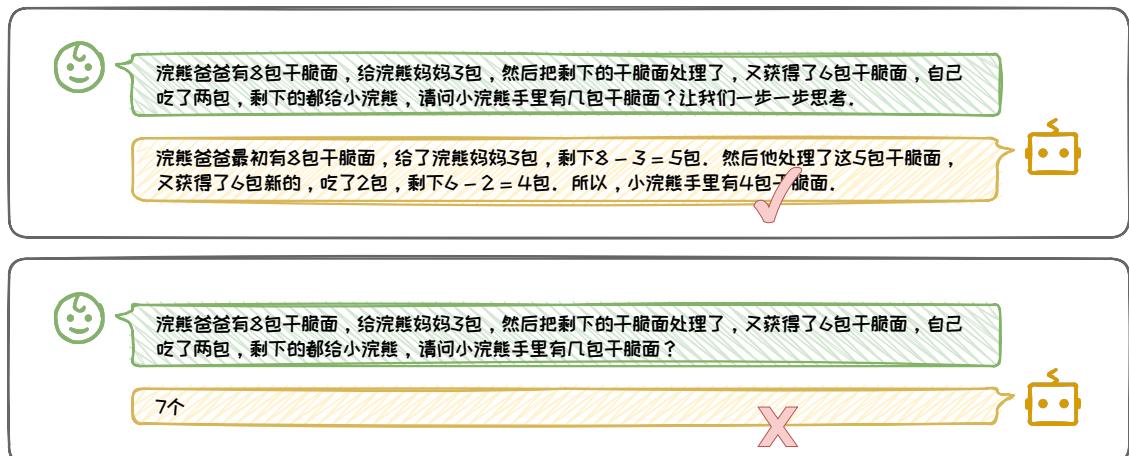


图 3.25: 在推理任务上使用 CoT

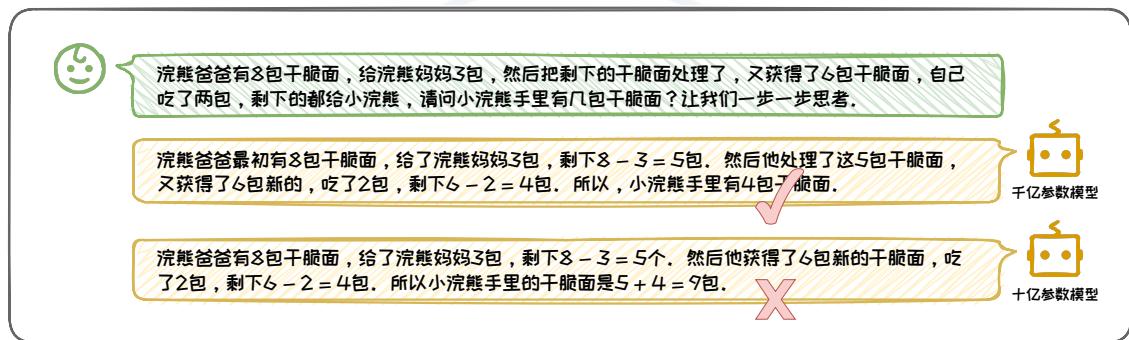


图 3.26: 不同规模模型使用 CoT 对比

的语言模型通常在包含详尽逐步推理的庞大数据集上接受训练，这可能助力它们发展出链式推理的能力。对于那些**未经指令微调的 LLMs**，如早期的 GPT-3、Palm 以及目前开源的基础版本模型，例如 LLaMA2-13B-Base 和 Baichuan2-13B-Base，通过适当的 CoT 提示，能够激发并展现其卓越的 CoT 推理能力；**经过指令微调的 LLMs**，如 ChatGPT、GPT-4 以及 LLaMA2-13B-Chat 等，在缺乏 CoT 指令的情况下，仍能自发地生成条理清晰的中间推理步骤。在众多情况下，这些模型在没有 CoT 指令的条件下反而展现出更佳的性能。这一现象揭示了一个事实：这些模型在指令微调的过程中可能已经深入学习并内化了 CoT 指令，使得即便在没有明确的 CoT 提示时，经过充分训练的模型，例如 ChatGPT 等，仍能够隐式地遵循 CoT 推

理路径 [5]。

## 2. 灵活使用 CoT

如何使用 CoT（思维链技术）的关键在于根据任务的具体需求和模型的特性来调整 CoT 的使用方式。以下是一些指导原则，展示如何在不同场景下有效使用 CoT。

- **调整 CoT 的复杂性：**根据任务需求和模型能力调整 CoT 的复杂性，可以通过 Few-Shot 的方式指定 CoT 输出的形式和风格。
- **使用不同的 CoT 形式：**根据不同任务场景，选择不同的 CoT 形式。

### 调整 CoT 的长度和复杂性

CoT 的长度和复杂性应根据任务的难度和模型的能力进行调整。通过 Few-Shot 示例，我们可以指定 CoT 输出的形式和风格，以适应不同的用户需求，如图 3.27 所示。

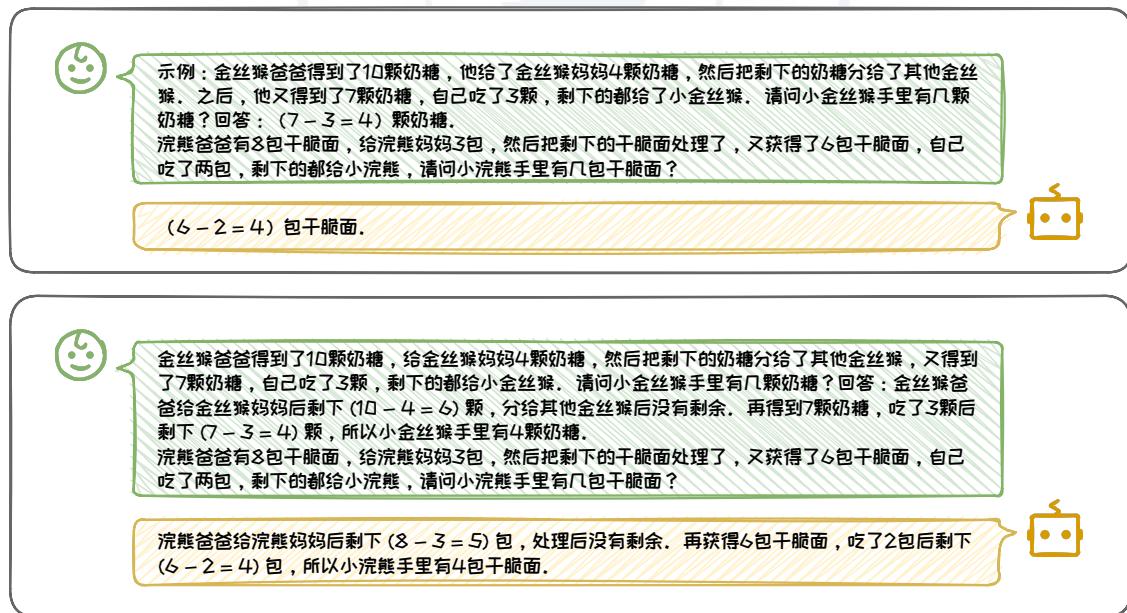


图 3.27：通过 Few-Shot 示例指定 CoT 输出的形式和风格

- **简短 CoT：**直接给出最终乘法和加法的结果，适合快速回答简单的计算问题。

- **详细 CoT**: 通过 Few-shot 样例引导，展示了完整的计算步骤，适合需要理解计算过程或对推理过程感兴趣的用户。

通过这个对比，我们能够根据不同的使用场景和用户需求来调整 CoT 的风格，提供不同程度的细节和解释，从而提高模型的交互质量，让大模型做到“千人千面”地回答问题。

### 使用不同的 CoT 形式

在不同的应用场景中，选择合适的 CoT（思维链）形式对于提高模型的性能和交互体验至关重要。

在面对缺乏具体 CoT 示例的通用问题解决场景时，可以使用 Zero-Shot CoT 或者 Auto CoT 的方式，通过“让我们一步一步思考”这种 CoT 提示触发词，来引导模型以 CoT 的形式回答内容。这种方法适用于那些不需要特定领域知识，但需要逻辑推理和逐步分析的问题。

在做数学、代码任务时，可以让模型生成多个回答，并提出最终结果，然后利用 Self-Consistency 方法，来寻找一致性最强的答案。这种方法适用于那些需要高准确度和可靠性的任务。例如，在编写一段代码时，模型可以生成多个版本的代码，并通过 Self-Consistency 方法来确保最终选择的代码是逻辑上最一致的。

在做一些包含创意思维的任务时，我们可能需要沿着一种模式一直探索，根据探索的结果做继续探索或者回溯到先前状态的决策，这时候使用 ToT 或者 GoT 能很好地适应这种场景。ToT 和 GoT 都是基于树或图的搜索策略，它们可以帮助模型在多个可能的思维路径中进行探索和选择。例如，在创作一个故事时，模型可以使用 ToT 或 GoT 来探索不同的情节发展路径，并选择最有趣或最合理的发展方向。

### 3.4.4 善用心理暗示

在硅谷，流传着一句创业金句，“Fake it till you make it”（假装它直到你成功）。这句话具体含义为，先吹嘘你的想法，再通过吸引资本和人才的方式努力地在现实中追赶上你的目标，直到你实现了它。这句话源自一种积极的心理暗示方法：**通过模仿自信和乐观的心态，一个人可以在他们的现实生活中实现这些品质**。这种现象不仅局限于个人行为；它对大语言模型的运作，同样有着深远的影响。

#### 角色扮演

正如心理学中的暗示可以引导人的行为和思维一样，指导一个大模型扮演特定的角色或人格可以显著提高其性能。这种技术被称为角色扮演（Role-Playing），它允许 LLM 生成更准确、与上下文相关且与人格一致的响应。通过为大模型定义一个详细的角色，比如数据科学家、诗人或律师，可以将模型的输出引导到所需的方向，从而提供更加高质量的回答<sup>5</sup>。

为了构建一个有效的角色，需要在指令中包含具体属性、职责、知识和技能。这不仅为模型设定了明确的期望，还帮助其更准确地模拟角色的语言和行为。在设计角色设定提示时，选择对特定任务具有明显优势的角色至关重要。通过额外的描述来强调这一优势，通常可以带来更好的结果 [13]。例如，我们为大模型设定一个“小浣熊营养顾问”的角色时，可以这样描述：“你是一位专业的小浣熊营养顾问，专门为小浣熊设计符合其自然饮食习惯的健康营养计划。”

如图 3.28 所示，通过角色扮演，模型在扮演“专业的小浣熊营养顾问”这一角色时，展现出了对小浣熊饮食需求的深刻理解和实际操作的灵活性。在回答用户关于小浣熊尝试干脆面的询问时，模型不仅坚持了小浣熊应以昆虫、水果、坚果和蔬菜为主的自然饮食原则，还巧妙地提出了将干脆面作为零食的建议，并强调了

<sup>5</sup><https://promptengineering.org/role-playing-in-large-languagemodels-like-chatgpt>

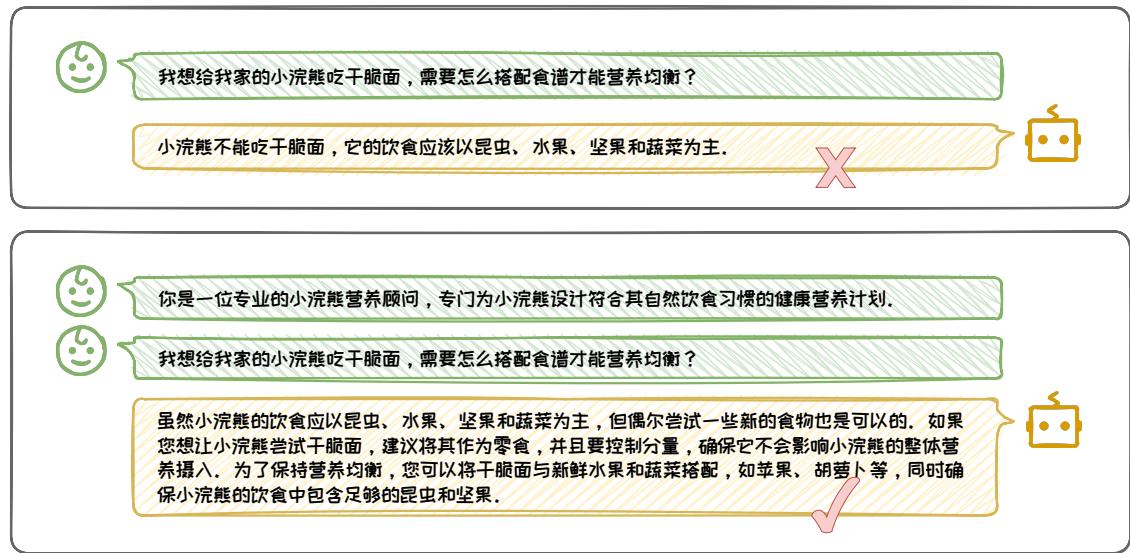


图 3.28: 角色扮演示例

控制分量的重要性。此外，模型还建议将干脆面与新鲜水果和蔬菜搭配，确保小浣熊的饮食中包含足够的昆虫和坚果，以维持营养均衡。这一建议既科学又实用，充分展现了角色扮演在提升模型交互质量和适应用户需求方面的巨大潜力。

### 情景代入

在日常生活中，一个人容易受到周围环境和交往人群的影响，可能会逐渐采纳周围人的行为和思维方式。这种情景代入现象不仅适用于人，也同样适用于大语言模型。通过将模型置于特定的“情景”或“环境”中，可以影响其生成的文本内容和风格。情景代入指的是将特定情境下所需的专业知识、历史背景等信息嵌入到模型的响应中。这要求模型理解当前的环境和情况来生成回答。

在不使用情景代入的情况下，模型的输出是一般性的常识信息，虽然准确，但缺乏深度。而我们为模型设定在公园遇到野生小浣熊的情景时，模型会代入到情景中，并且根据相关信息，给出针对性的建议，提醒我们为了安全要避免喂养野生小浣熊。通过这种方式，大模型能够给出更加有用以及高质量的回答。

通过情景代入，模型能够深入理解并反映特定情境下的文化社会背景与现实

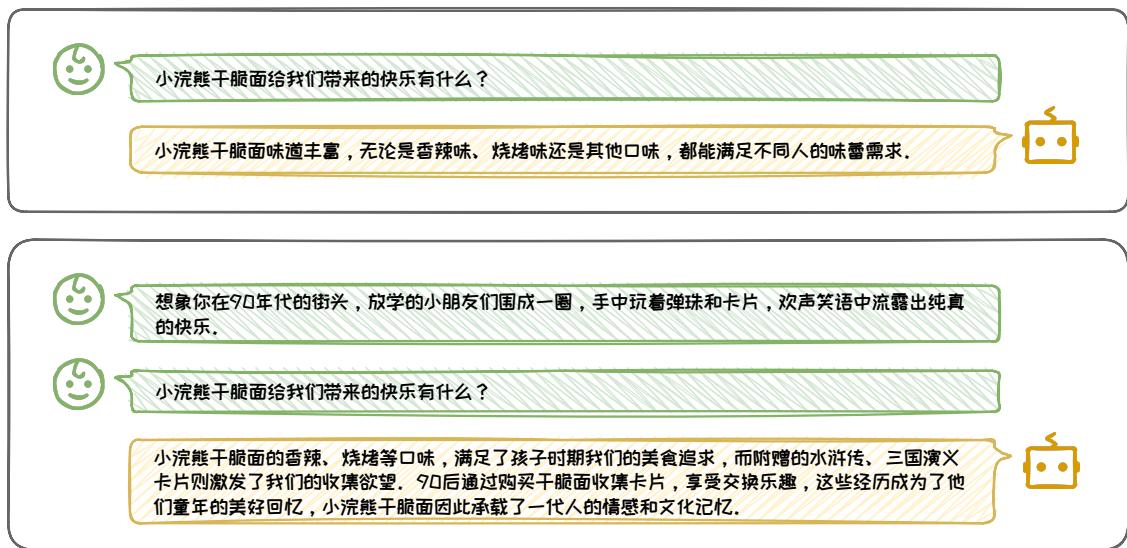


图 3.29: 情景代入示例

环境，从而生成更加丰富和有深度的回答。在上述例子中，当模型被置于 90 年代的街头情景中，它不仅能够描述小浣熊干脆面的口味，还能够捕捉到那个时代特有的文化现象——通过收集卡片来交换乐趣。这种回答不仅提供了具体的信息，还唤起了用户的情感共鸣，增强了交互的情感连接。

在本节中，我们深入探讨了提升 Prompt 技巧的多种策略，以增强大语言模型的交互效率和输出质量。这些技巧主要包括规范 Prompt 编写、合理归纳提问、适时使用思维链、以及善用心理暗示：

- **规范 Prompt 编写：**明确任务说明、丰富且清晰的上下文、规范的输出格式和清晰的排版。这些要素共同构成了一个有效的 Prompt，使得模型能够准确理解用户意图并生成高质量的输出。
- **合理归纳提问：**通过拆解复杂问题、追问、深入追问、扩展追问和反馈追问等多种提问方式，确保模型能够全面理解问题并提供详细的回答。
- **适时使用 CoT：**根据具体任务需求，灵活调整 CoT 的长度和复杂性，采用不同的 CoT 形式以优化模型推理能力。

- **善用心理暗示：**通过角色扮演和情景代入等技术，使模型在特定情境下生成更有深度和针对性的回答，从而提高互动的质量和用户体验。  
这些技巧和策略的应用，不仅提升了提示的有效性，使得模型能够更准确地理解和回应用户的需求，还显著提高了大语言模型在复杂任务中的表现。这些综合策略为大规模语言模型在各类复杂任务中的有效应用提供了坚实支持。



## 3.5 相关应用

Prompt 工程的应用极为广泛，几乎所有需要与大模型进行高效交互的场景都离不开它。这项技术不仅能够帮助我们处理一些基础任务，如摘要生成、文本分类、情感分析等，还能显著提升大语言模型在应对复杂任务时的表现。例如，在 Agent 中的任务规划、利用大模型进行数据合成、Text-to-SQL、以及设计个性化的 GPTs 等方面，Prompt 工程都发挥着不可或缺的作用。下面我们将依次介绍 Prompt 工程在它们中的应用。

### 3.5.1 Agent

Prompt 工程在 Agent 中广泛使用，但主要集中在 Agent 的任务规划上，在详细介绍之前，我们先来了解一下 Agent。**Agent 是什么？** Agent 是某种能自主理解、规划决策、执行复杂任务的系统。它具备感知、规划、和行动能力，能够在没有直接人类干预的情况下执行任务。如果把大模型类比为大脑（具有常识、推理能力），那么 Agent 就是一个独立的人，能够感知环境，有长期记忆，可以使用各种工具。Agent 是大模型应用中最受认可、最有前景的方向，在工业界甚至有“大模型是上半场，Agent 是下半场”的说法。比尔盖茨也曾在他的个人博客上写到：“AI Agent 将是人工智能的未来”。

**Agent 怎么实现？** 对于一个单 Agent 系统，它的组成可以由公式表示： $Agent = 大语言模型 + 记忆组件 + 规划组件 + 工具使用组件$ <sup>6</sup>。如图 3.30 所示，可以直观地看到各个组件的作用，Agent 使用规划组件进行任务规划，使用记忆组件存储记忆，使用工具使用组件调用外部工具。对于一个多 Agent 系统，它的组成可以由公式表示： $Multi-Agent = Agent1 + \dots + AgentN$ 。其中每一个 Agent 的组成部分都与单

<sup>6</sup><https://lilianweng.github.io/posts/2023-06-23-agent>

Agent 系统一样，不同的是，在多 Agent 系统中，每个 Agent 都有自己的角色，都需要进行角色配置，具体如何配置取决于具体的应用场景。在多 Agent 系统中，Agent 之间可以相互协作，也可以相互竞争，从而完成更加复杂的任务。

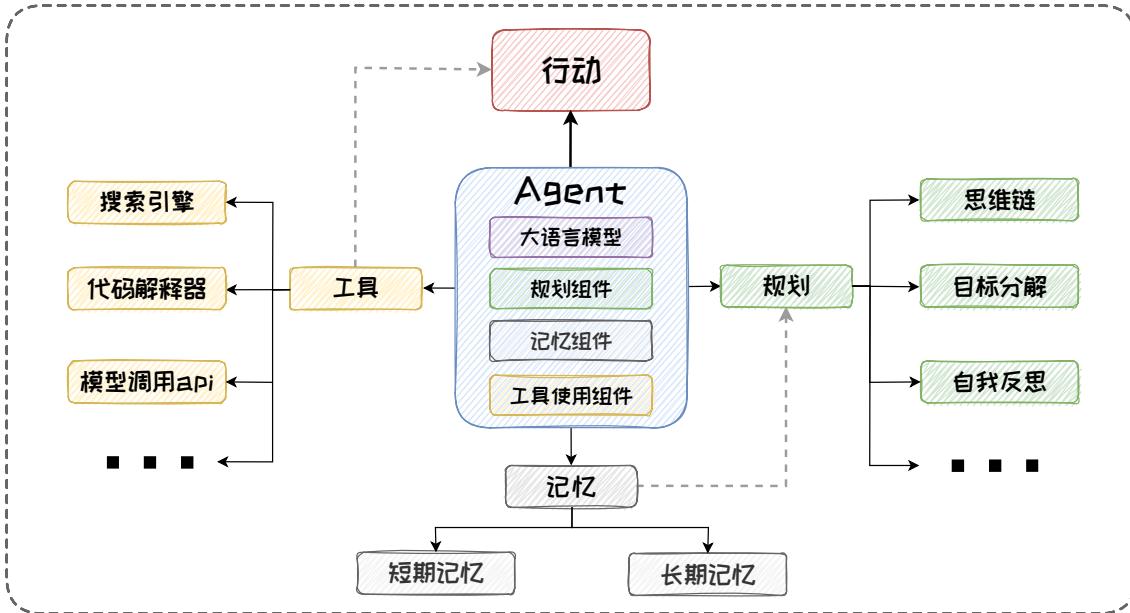


图 3.30: Agent 架构

以单 Agent 系统为例，下面我们来详细介绍 Agent 的组成部分：

**大语言模型**，作为 Agent 的智慧中枢，宛如一个精密的控制中心或大脑，承担着协调与执行的重任。它不仅负责处理和响应用户请求，还必须精确地执行一系列复杂的操作，确保任务的顺利进行和高效完成。

**规划组件**，作为 Agent 的核心部分，负责将复杂的任务精细地分解为一系列可管理的子目标。随后，这些子目标将被逐一执行，或者进一步细化为更具体的行动步骤。在整个任务执行过程中，规划组件不断回顾和优化先前的决策，及时纠正可能出现的偏差，通过这种持续的迭代改进，确保任务的高效和准确执行。

**记忆组件**，其功能犹如人类的记忆系统，细分为感觉记忆、短时记忆和长时记忆三个层次。在 Agent 的运作中，原始输入的嵌入表示被类比为感觉记忆，迅速捕捉即时的信息；而所有的上下文信息则被视为短时记忆，暂时存储以供即时处理；

至于那些存储在外部数据库中的信息，则被看作是长期记忆，它们是知识的宝库，Agent 能够随时存储和检索这些信息，确保在执行任务时能够充分运用过往的知识和经验，以达到最佳的决策和行动效果。

**工具使用组件**，作为 Agent 的重要组成部分，工具使用组件能够灵活调用外部工具，以补充模型权重中未涵盖的额外信息。值得注意的是，工具使用组件并不包括具体的工具本身，而是指调用这些工具的能力。该组件功能多样，包括利用搜索引擎 API 进行实时信息检索、使用代码解释器执行编程任务、访问专有信息源，甚至调用其他模型以获取更广泛的数据支持等。通过这些手段，工具使用组件确保 Agent 能够获取必要的信息和资源，以应对各种复杂的任务需求。

当 Agent 接收到用户的输入指令后，它首先启动规划组件，利用大语言模型将宏大的任务细化为一系列具体且可操作的子目标。随后，逐一执行这些子目标，过程中可能会借助工具使用组件，调用代码解释器或进行信息搜索等，以获取必要的辅助信息。同时，记忆组件在这一过程中发挥着关键作用，它不仅存储了获取的信息，还记录了模型的推理路径和每个子目标的执行结果。通过这种迭代式的处理流程，Agent 不断循环优化，直至得到最终答案。

通过上面的阐述，我们可以清晰地认识到规划组件在 Agent 中的核心地位，它对 Agent 的整体性能表现有着决定性的影响。一个高效的规划策略，其精髓在于精心设计的 Prompt，这些 Prompt 运用了包括但不限于目标分解、思维链（Chain of Thought, CoT [32]、思维树 (Tree of Thoughts) [37] 等 Prompt 工程技术。以一个例子来说明，面对任务“小浣熊想吃箱子里的干脆面”，我们可以通过以下 Prompt 之一来分解任务：

“小浣熊想吃箱子里的干脆面”的步骤。

“小浣熊想吃箱子里的干脆面”的子目标是什么。

“小浣熊想吃箱子里的干脆面”，让我们一步一步思考。

通过上面的 Prompt，模型可能输出下面的内容，对任务进行分解：

1. 确定位置：小浣熊首先需要知道干脆面的具体位置。如果它已经知道干脆面在箱子里，那么它需要找到这个箱子。
2. 找到箱子：如果箱子不在视线范围内，小浣熊可能需要通过嗅觉或记忆来找到箱子。
3. 接近箱子：.....

.....

在 Agent 的规划过程中，其是否具备自我反思的能力，是衡量其推理与规划能力的关键指标。自我反思机制通过不断审视和优化过往的行动决策，及时纠正错误，实现迭代式的改进。在不可避免地会出现试错的现实任务中发挥着至关重要的作用。其中比较典型的方法是 ReAct [36]，ReAct (Reason + Act) 的核心在于“推理 (Reason) + 行动 (Act)”的协同作用。在 Agent 执行下一步行动时，ReAct 方法会融入大模型的思考过程 (Thought)，并将这一思考过程、所使用的工具及参数、以及执行的结果 (Observation) 整合到 Prompt 中。这种做法不仅增强了模型对当前任务的反思能力，还使其能够对先前的任务完成度进行深入分析，从而显著提升模型的问题解决能力。

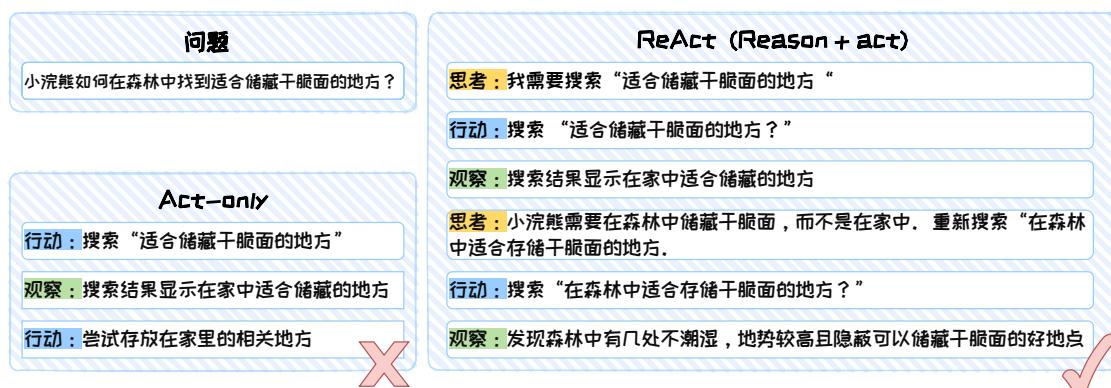


图 3.31: ReAct

图 3.31 展示了 ReAct 和 Act only 的对比示例。Act-only 方法，虽然能够通过与外部环境的互动来获取额外信息，但其对大模型的推理能力利用不足，即便接收到错误信息，也往往缺乏自我修正的能力，继续执行任务。相比之下，ReAct 方法不仅充分利用了大模型的推理能力，还同样与外部环境互动以获取必要信息。更为关键的是，ReAct 在每次行动前都会加入模型的思考过程，对先前的行动进行总结和反思。一旦发现任务的解决路径与既定目标出现偏差，ReAct 会及时调整策略，确保任务的正确执行。这种增强的解释性和自我反思与修正能力，显著提升了 Agent 在任务规划上的水平。

### 3.5.2 数据合成

通过使用 Prompt 工程技术，我们可以利用大模型来进行数据合成。首先我们先介绍一下什么是数据合成。在大模型语境下，数据合成是指利用大语言模型来生成数据，这些数据在某些方面与真实数据相似，但并不是从真实环境中直接采集的，合成数据通常用于数据预处理、模型训练、数据集增强等场景。

合成数据因其可控性、安全性和低成本等优势而备受青睐，而利用大模型生成其自身的训练数据则一直是 AI 研究领域的热点议题。这一议题在 AI 学术界引起了广泛关注，其中，利用大模型合成数据的典型方法之一便是 Self-Instruct [31]。

Self-Instruct 是一个框架，它利用 Prompt 工程技术，通过预训练语言模型自动生成指令、输入和输出样本。然后，筛选并利用这些样本来微调原始模型，以提升模型遵循指令的能力。这个方法的关键优势在于可以使用模型自身生成的数据来提升其指令跟随能力，从而减少对人类编写的指令数据的依赖，提高模型的通用性和泛化能力。

**Self-Instruct 具体是怎样实现的呢？**图 3.32 给出了它的工作流程，它从一个有限的手动编写任务种子集开始，用于引导整个生成过程。

第一阶段，**生成指令**，采用 few-shot 技术，从任务池中随机抽取 8 个现有指令作为演示示例组成上下文。让模型为新任务生成指令。

第二阶段，**分类任务识别**，由于分类任务和非分类任务的后续处理不同，所以需要先判断是否为分类任务。同样使用 few-shot 技术，使用多个带标签的示例组成上下文，指导模型判断该指令是否为分类任务。以下是该阶段 few-shot 的 Prompt 例子：

这个任务可以被视为一个具有有限输出标签的分类任务吗？

任务：判断下面有关小浣熊的内容是真是假。

是分类任务吗？是

任务：给我一个你必须幽默地说话的例子。

是分类任务吗？否

任务：目标任务的指令

是分类任务吗？

第三阶段，**指令数据生成**，指令微调数据一般分为三部分：指令、输入和输出，大模型需要对这 3 部分都进行生成。对于不同的任务类型，Self-Instruct 使用了不同的生成策略，1. 输入优先策略，模型被提示先生成实例，然后生成相应的输出。对于不需要额外输入的指令，允许直接生成输出。2. 输出优先策略，模型首先被提示生成类别标签，然后生成相应的输入。输入优先策略不适合部分任务生成，尤其是分类任务，生成的输入会偏向于某个标签。所以针对于分类任务生成指令数据，采用输出优先策略，针对非分类任务生成指令数据，采用输入优先策略。

最后阶段，**过滤**，使用各种启发式方法自动过滤低质量或重复的指令数据，然后将剩余的有效任务添加到任务池中。

整个过程可以循环多次，直到任务池中收集到足够的数据。

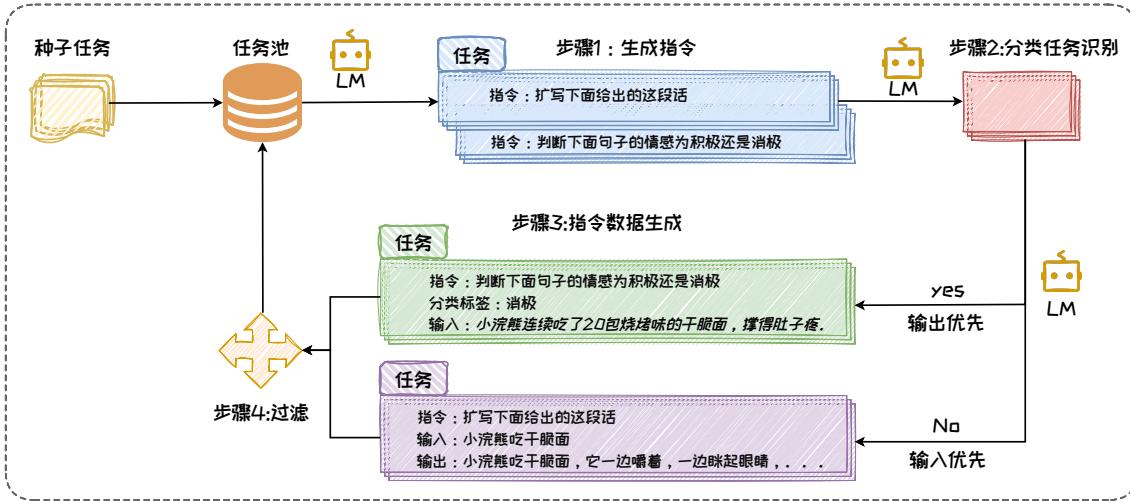


图 3.32: Self-Instruct

### 3.5.3 GPTs

GPTs 是 OpenAI 推出的可以自定义的 ChatGPT，允许用户根据特定需求创建和使用定制版的 GPT 模型。这些定制版的 GPT 模型称为 GPTs，旨在让用户能够根据自己的具体应用场景选择合适的 GPT 模型。比如图 3.33 中展示的 GPTs。

图 3.34 为如何制作自己的 GPTs 的页面，在这个专属页面中，用户拥有充分的自由度来自定义 GPT 的名称，并详细描述其功能（即预设一个 Prompt）。此外，还可以定制专属的知识库，并根据需求选择所需的能力，例如网络搜索、图像生成、代码解释等。一旦完成个性化 GPTs 的制作，用户可以选择将其分享，供其他用户使用，同时也能享受他人分享的 GPTs 带来的便利。尽管 GPTs 在某些功能上与 Agent 相似，但它们目前主要扮演聊天机器人的角色，专注于通过对话为用户提供帮助，而非完全自主的 Agent。在制作 GPTs 时，填写“Description”和“Instructions”实际上就是构建预设 Prompt 的过程，可以运用本章所介绍的 Prompt 工程技术编写这些关键内容以提高 GPTs 的性能。例如，当我们想制作一个 GPTs 作为我们的小浣熊营养顾问时，我们可以利用角色扮演的方式（在 3.4 节中有详细介绍），在

# GPTs

Discover and create custom versions of ChatGPT that combine instructions, extra knowledge, and any combination of skills.

Top Picks   DALL-E   Writing   Productivity   Research & Analysis   Programming   Education   Lifestyle

## Featured

Curated top picks from this week



**Adobe Express**  
Stand out with Adobe Express. Quickly and easily make impactful social posts, images, videos, flyers...  
By adobe.com



**Code Copilot**  
Code Smarter, Build Faster—with the Expertise of a 10x Programmer by Your Side.  
By promptspellsmith.com



**Social Butterfly-AI 2.0**  
AI Assistant for content development and social media strategy. Nuanced, guided...  
By digitalmogul.co



**Software Architect GPT**  
Builds new software architecture documents by understanding user requirements and design...  
By V B Wickramasinghe

图 3.33: GPTs<sup>7</sup>

Instructions 中填写以下内容：

你是一位专业的小浣熊营养顾问，专注于为小浣熊设计符合其自然饮食习惯的健康饮食计划。你了解家庭饲养环境的限制，并能够提供实用的饮食建议。

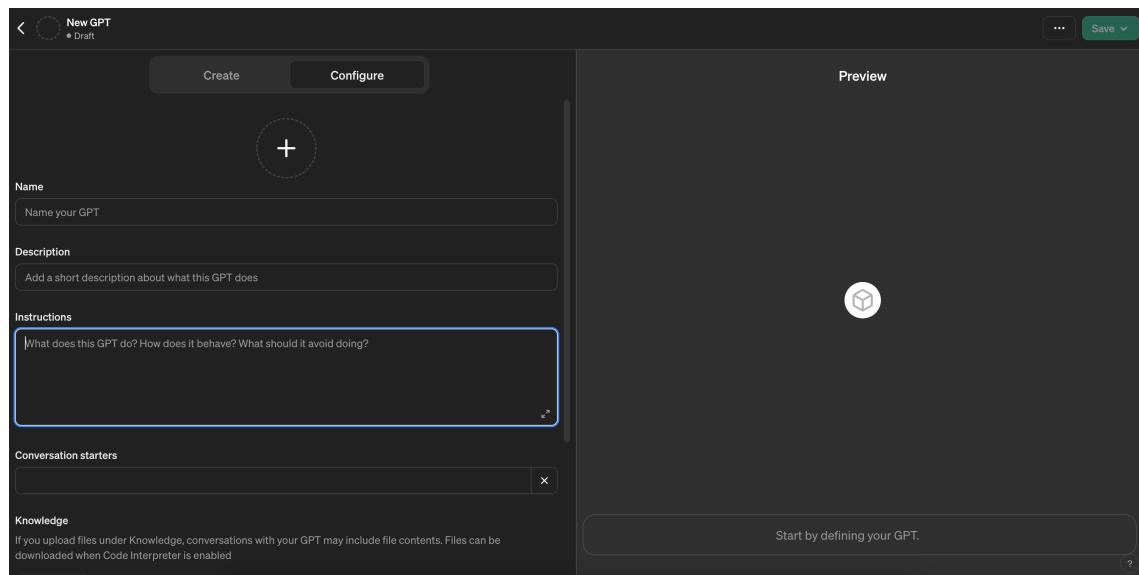


图 3.34: GPTS 制作<sup>8</sup>

OpenAI 不仅推出了 GPTs，还贴心地设立了 GPTs 商店，为用户提供了一个便捷的平台，用于分享和使用这些定制化的 GPT 模型。截止目前，公开可用的 GPTs 数量已突破 300 万大关，每一个 GPTs 都精通于某一特定领域，其功能涵盖了文本生成、代码解释、图像生成、数据分析、游戏制作、专业设计、学术研究等多个方面。

在这些众多的 GPTs 中，一些特别受欢迎的模型如 Creative Writing Coach，它能够帮助用户优化写作内容，提升写作技巧，并提供宝贵的反馈，从而显著提高创意写作的效率。另一款备受瞩目的模型是 Scholar GPT，这是一款专为研究和学术领域量身打造的人工智能助手，致力于为研究人员、学者和学生提供包括数据分

析、文献检索、实时信息更新在内的全方位支持。通过深入学习和理解复杂的数据与文献，Scholar GPT 旨在成为学术探索中的得力伙伴。

### 3.5.4 Text-to-SQL

互联网技术进步带动数据量指数增长，目前金融、电商等各行业的海量高价值数据主要以数据库形式进行存储，因此人们对于数据库查询的需求越来越大。目前，数据库的操作基本都需要使用 SQL 来实现，而只有专业人员才能够熟练使用 SQL。因此，如何实现进行零代码或低代码来做数据管理与分析，是亟待解决的问题。为此，人们提出了 Text-to-SQL 技术，即将自然语言查询翻译成可以在数据库中执行的 SQL 语句，并在数据库中执行，返回 SQL 执行结果。

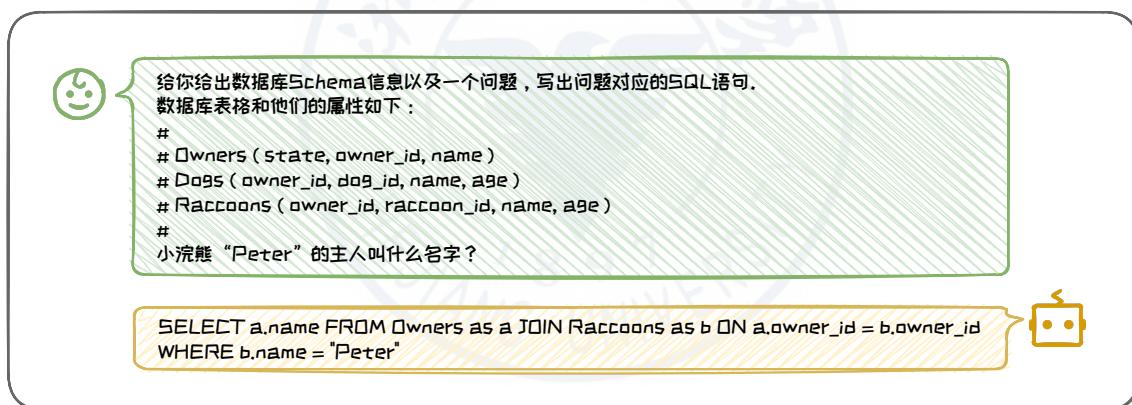


图 3.35: Text-to-SQL 示例

Text-to-SQL 一个例子如图 3.35 所示，用户把问题输入给大语言模型，这里提问了“小浣熊“Peter”的主人叫什么名字？”，大模型会根据用户的问题，来生成对应的 SQL 语句，即“SELECT a.name FROM Owners as a JOIN Raccoons as b ON a.owner\_id = b.owner\_id WHERE b.name = ”Peter“;”。随后，SQL 语句可以在对应的数据库中执行，得到用户提问的答案，并返回给用户。

通过 Text-to-SQL 的方式，我们只需要动动嘴用大白话就能对数据库进行查询，

而不必自己编写 SQL 语句。这让广大普通人都可以自由操作数据库，挖掘数据库中隐含的数据价值。

浙江大学提出了一种基于 ChatGPT 的零样本 Text-to-SQL 方法，称为 C3 [8]，核心在于 Prompt 工程的设计，给出了如何针对 Text-to-SQL 任务设计 Prompt 来优化生成效果。目前 C3 在 Text-to-SQL 国际权威榜单 Spider 同类方法中排名全球第一。C3 由三个关键部分组成：清晰提示（Clear Prompting）、提示校准（Calibration with Hints）和一致输出（Consistent Output），分别对应模型输入、模型偏差和模型输出。

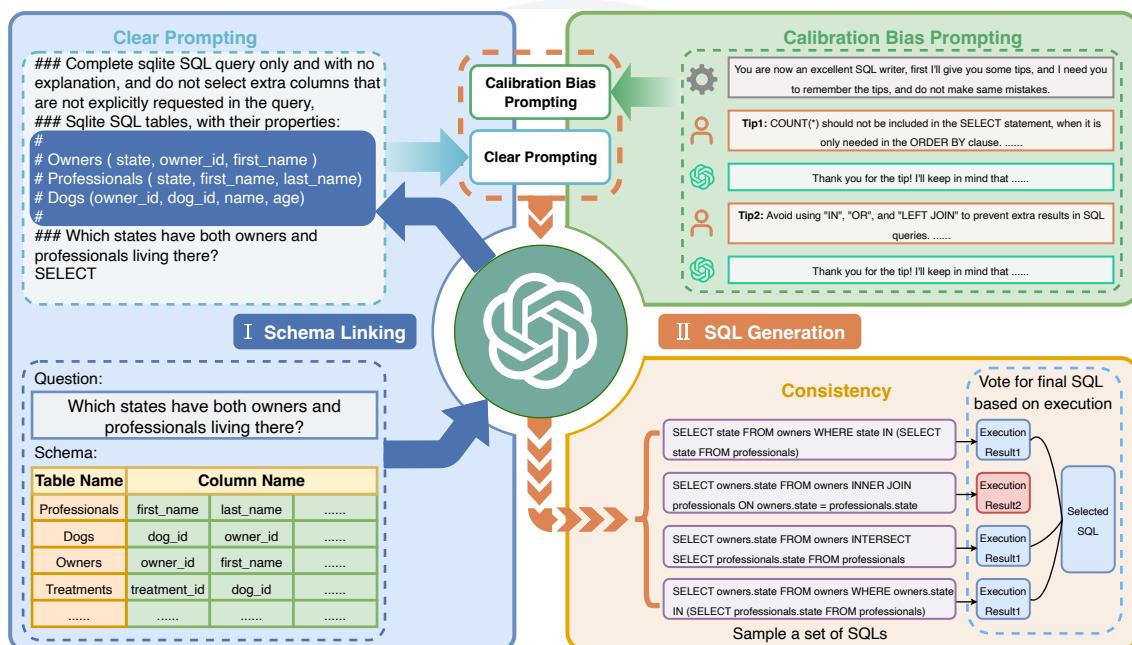


图 3.36: C3

**清晰提示 (Clear Prompting)** 包含两部分：

- **清晰布局 (Clear Layout)**，通过使用明确的符号将指令、上下文和问题进行划分，确保了指令模板的清晰性，这极大地提升了 ChatGPT 对问题的理解能力。这一策略体现了 Prompt 技巧中的“排版要清晰”原则，确保了信息的有效传递。

- **清晰上下文 (Clear Context)**，设计零样本 Prompt 指示 ChatGPT 召回数据库中与问题相关的表，基于表召回的结果，设计零样本 Prompt 指示 ChatGPT 进一步召回表中的相关列。这一步的目的在于检索与问题相关的表格和字段，去除掉与问题无关的内容，减少上下文长度，以及无效的冗余信息，从而提升 ChatGPT 生成 SQL 的准确性。这正是 Prompt 技巧中“上下文丰富且清晰”的体现，确保了模型在处理任务时能够聚焦于关键信息。

**提示校准 (Calibration with Hints)**，旨在减少 ChatGPT 与 Text-to-SQL 任务本身固有的偏差，ChatGPT 在编写 SQL 查询时容易产生保守的输出，即倾向于选择额外的列和提供额外的执行结果。为了校准偏差，此研究提出了一种插件式校准策略，称为“提示校准”，通过使用包含历史对话的上下文提示，将先验知识纳入 ChatGPT。在历史对话中，首先将 ChatGPT 的角色设置为优秀的 SQL 专家，然后通过对话语形式引导它遵循预先设定的提示，来生成内容。通过这种角色扮演的方式，ChatGPT 被引导遵循预设的提示，从而有效地校准了偏差。

**输出校准 (Output Calibration)**，由于大语言模型固有的随机性，ChatGPT 的输出并不稳定。为了增强大模型输出的稳定性，保持生成的 SQL 查询的一致性，此研究将 Self-Consistency 方法应用到 Text-to-SQL 任务上，其思想是首先对多种不同的推理路径进行采样，然后选择最一致的答案。

在本章中，我们详细探讨了 Prompt 工程的广泛应用，并深入剖析了其在多个领域中的具体应用场景和技术实现。

首先，在 **Agent** 中，利用 Prompt 工程将复杂任务分解为可操作的子目标，并利用记忆组件和工具使用组件提升 Agent 的决策和执行能力。我们还讨论了 ReAct 方法，通过结合推理和行动，增强了 Agent 的自我反思能力，从而提升其任务规划水平。其次，在 **数据合成方面**，我们介绍了 Self-Instruct 框架，利用 Prompt 工程技术自动生成指令数据。然后，在 **GPT 的定制化方面**，我们探讨了如何使用 Prompt

工程技术创建个性化的 GPT。通过自定义名称、描述和功能，可以创建适应特定需求的 GPTs。接着，在 **Text-to-SQL** 方面，我们介绍了 Text-to-SQL 是什么，以及代表性方法 C3，通过清晰的 Prompt 设计，提高 ChatGPT 处理 Text-to-SQL 任务的能力。

总结而言，Prompt 工程在提升大模型的交互和执行能力方面具有重要作用。无论是在任务规划、数据合成、个性化模型定制，还是 Text-to-SQL 中，Prompt 工程都展现了其独特的优势和广泛的应用前景。

## 参考文献

- [1] Maciej Besta et al. “Graph of Thoughts: Solving Elaborate Problems with Large Language Models”. In: *AAAI*. 2024.
- [2] Tom B. Brown et al. “Language Models are Few-Shot Learners”. In: *arXiv preprint arXiv:2005.14165* (2020).
- [3] Tom B. Brown et al. “Language Models are Few-Shot Learners”. In: *NeurIPS*. 2020.
- [4] Stephanie C. Y. Chan et al. “Data Distributional Properties Drive Emergent In-Context Learning in Transformers”. In: *NeurIPS*. 2022.
- [5] Juhai Chen et al. “When do you need Chain-of-Thought Prompting for ChatGPT?” In: *CoRR* abs/2304.03262 (2023).
- [6] Aakanksha Chowdhery et al. “PaLM: Scaling Language Modeling with Pathways”. In: *Journal of Machine Learning Research* 24 (2023), 240:1–240:113.
- [7] DeepSeek-AI et al. “DeepSeek-V2: A Strong, Economical, and Efficient Mixture-of-Experts Language Model”. In: *arXiv preprint arXiv:2405.04434* (2024).
- [8] Xuemei Dong et al. “C3: Zero-shot Text-to-SQL with ChatGPT”. In: *arXiv preprint arXiv:2307.07306* (2023).
- [9] Dan Hendrycks et al. “Measuring Massive Multitask Language Understanding”. In: *ICLR*. 2021.
- [10] Sepp Hochreiter and Jürgen Schmidhuber. “Long Short-Term Memory”. In: *Neural Computation* 9.8 (1997), pp. 1735–1780.

- [11] Huiqiang Jiang et al. “LLMLingua: Compressing Prompts for Accelerated Inference of Large Language Models”. In: *EMNLP*. 2023.
- [12] Takeshi Kojima et al. “Large Language Models are Zero-Shot Reasoners”. In: *NeurIPS*. 2022.
- [13] Aobo Kong et al. “Better Zero-Shot Reasoning with Role-Play Prompting”. In: *CoRR* abs/2308.07702 (2023).
- [14] Jannik Kossen, Yarin Gal, and Tom Rainforth. “In-Context Learning Learns Label Relationships but Is Not Conventional Learning”. In: *arXiv preprint arXiv:2307.12375* (2024).
- [15] Junlong Li et al. “Self-Prompting Large Language Models for Zero-Shot Open-Domain QA”. In: *arXiv preprint arXiv:2212.08635* (2024).
- [16] Xiaonan Li et al. “Unified Demonstration Retriever for In-Context Learning”. In: *ACL*. 2023.
- [17] Jiachang Liu et al. “What Makes Good In-Context Examples for GPT-3?” In: *ACL*. 2022, pp. 100–114.
- [18] Nelson F Liu et al. “Lost in the middle: How language models use long contexts”. In: *Transactions of the Association for Computational Linguistics* 12 (2024), pp. 157–173.
- [19] Yao Lu et al. “Fantastically Ordered Prompts and Where to Find Them: Overcoming Few-Shot Prompt Order Sensitivity”. In: *ACL*. 2022.
- [20] Man Luo et al. “In-context Learning with Retrieved Demonstrations for Language Models: A Survey”. In: *arXiv preprint arXiv:2401.11624* (2024).
- [21] Ziyang Luo et al. “Wizardcoder: Empowering code large language models with evol-instruct”. In: *arXiv preprint arXiv:2306.08568* (2023).
- [22] Yuren Mao et al. “FIT-RAG: Black-Box RAG with Factual Information and Token Reduction”. In: *arXiv preprint arXiv:2403.14374* (2024).
- [23] OpenAI. *Introducing GPTs*. 2023. URL: <https://openai.com/index/introducing-gpts/>.
- [24] Jane Pan et al. “What In-Context Learning ”Learns” In-Context: Disentangling Task Recognition and Task Learning”. In: *ACL*. 2023.
- [25] Joon Sung Park et al. “Generative Agents: Interactive Simulacra of Human Behavior”. In: *UIST*. 2023.

- [26] Allan Raventós et al. “Pretraining task diversity and the emergence of non-Bayesian in-context learning for regression”. In: *NeurIPS*. 2023.
- [27] Alexander Scarlatos and Andrew Lan. “RetICL: Sequential Retrieval of In-Context Examples with Reinforcement Learning”. In: *arXiv preprint arXiv:2305.14502* (2024).
- [28] Seongjin Shin et al. “On the Effect of Pretraining Corpora on In-context Learning by a Large-scale Language Model”. In: *NAACL*. 2022, pp. 5168–5186.
- [29] Rohan Taori et al. “Alpaca: A strong, replicable instruction-following model”. In: *Stanford Center for Research on Foundation Models*. <https://crfm.stanford.edu/2023/03/13/alpaca.html> (2023).
- [30] Xuezhi Wang et al. “Self-Consistency Improves Chain of Thought Reasoning in Language Models”. In: *ICLR*. 2023.
- [31] Yizhong Wang et al. “Self-Instruct: Aligning Language Models with Self-Generated Instructions”. In: *ACL*. 2023.
- [32] Jason Wei et al. “Chain-of-Thought Prompting Elicits Reasoning in Large Language Models”. In: *NeurIPS*. 2022.
- [33] Jason Wei et al. “Emergent Abilities of Large Language Models”. In: *Transaction of Machine Learning Research* 2022 (2022).
- [34] Wikipedia contributors. *Intelligent Agent*. 2024. URL: [https://en.wikipedia.org/wiki/Intelligent\\_agent](https://en.wikipedia.org/wiki/Intelligent_agent).
- [35] Sang Michael Xie et al. “An Explanation of In-context Learning as Implicit Bayesian Inference”. In: *ICLR*. 2022.
- [36] Shunyu Yao et al. “ReAct: Synergizing Reasoning and Acting in Language Models”. In: *ICLR*. 2023.
- [37] Shunyu Yao et al. “Tree of Thoughts: Deliberate Problem Solving with Large Language Models”. In: *NeurIPS*. 2023.
- [38] Kang Min Yoo et al. “Ground-Truth Labels Matter: A Deeper Look into Input-Label Demonstrations”. In: *EMNLP*. 2022.
- [39] Tao Yu et al. “Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task”. In: *EMNLP*. 2018.
- [40] Chao Zhang et al. “FinSQL: Model-Agnostic LLMs-based Text-to-SQL Framework for Financial Analysis”. In: *SIGMOD*. 2024.

- [41] Zhuosheng Zhang et al. “Automatic Chain of Thought Prompting in Large Language Models”. In: *ICLR*. 2023.
- [42] Yuxiang Zhou et al. “The Mystery of In-Context Learning: A Comprehensive Survey on Interpretation and Analysis”. In: *arXiv preprint arXiv:2311.00237* (2024).



# 4 参数高效微调

在现代人工智能的蓬勃发展过程中，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。

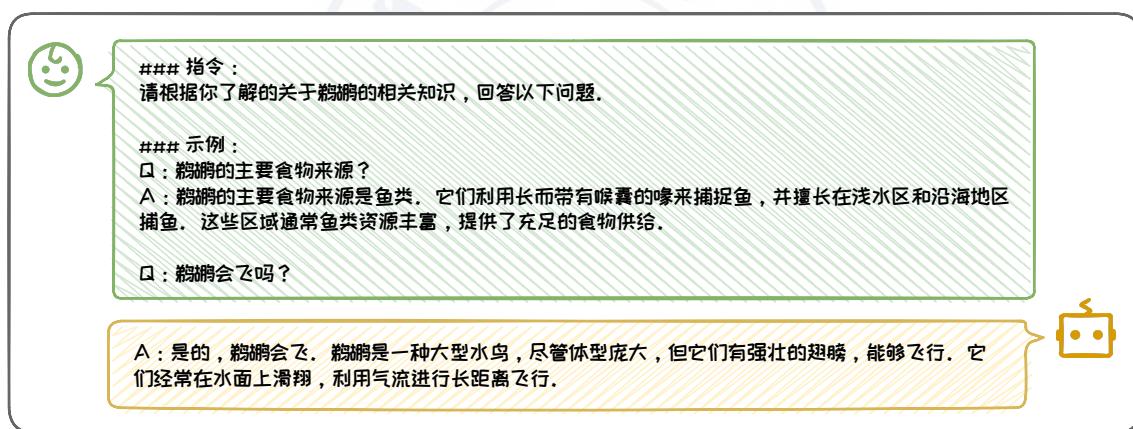


图 4.1: 指令数据样例

- **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]

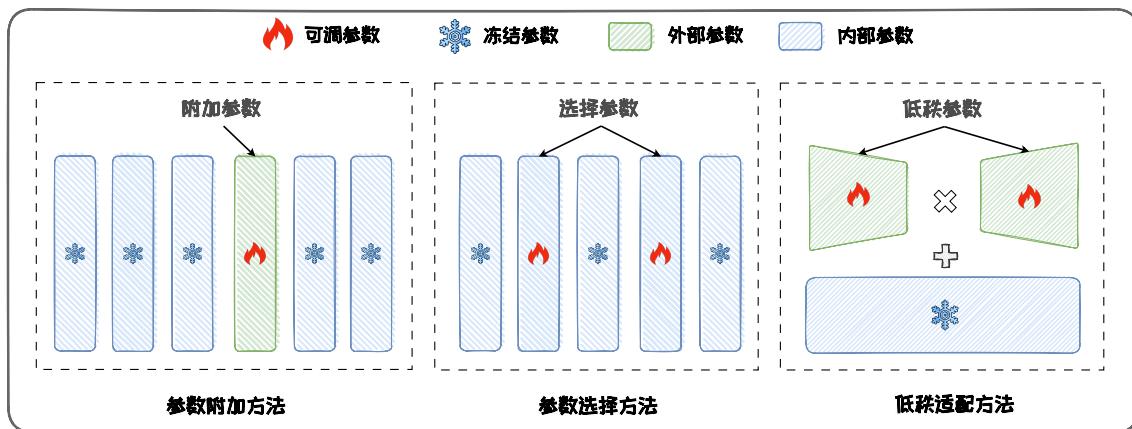


图 4.2: 高效参数微调方法分类学

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<sup>1</sup>，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

<sup>1</sup> 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  $\{w_1, w_2, \dots, w_n\}$ ，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。

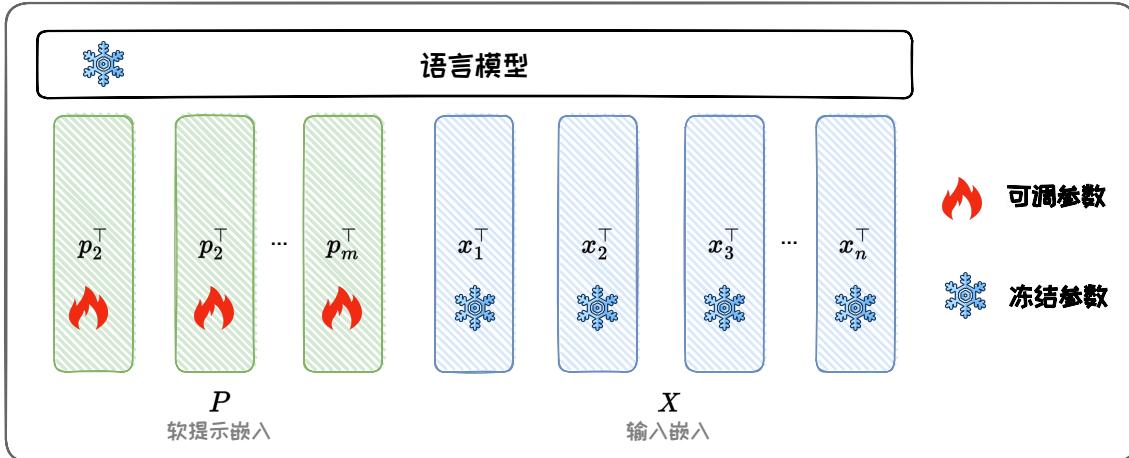


图 4.3: Prompt-tuning 示意图

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。

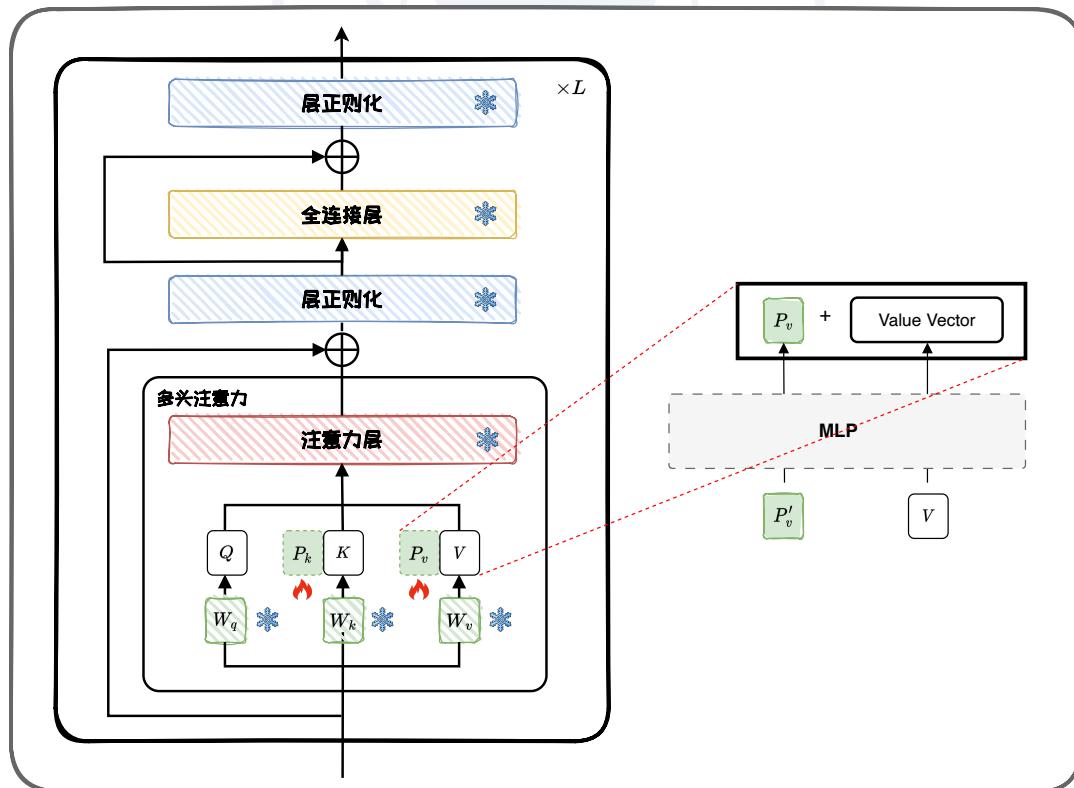


图 4.4: Prefix-tuning 示意图

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，先通过一个多层次感知机（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，它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。

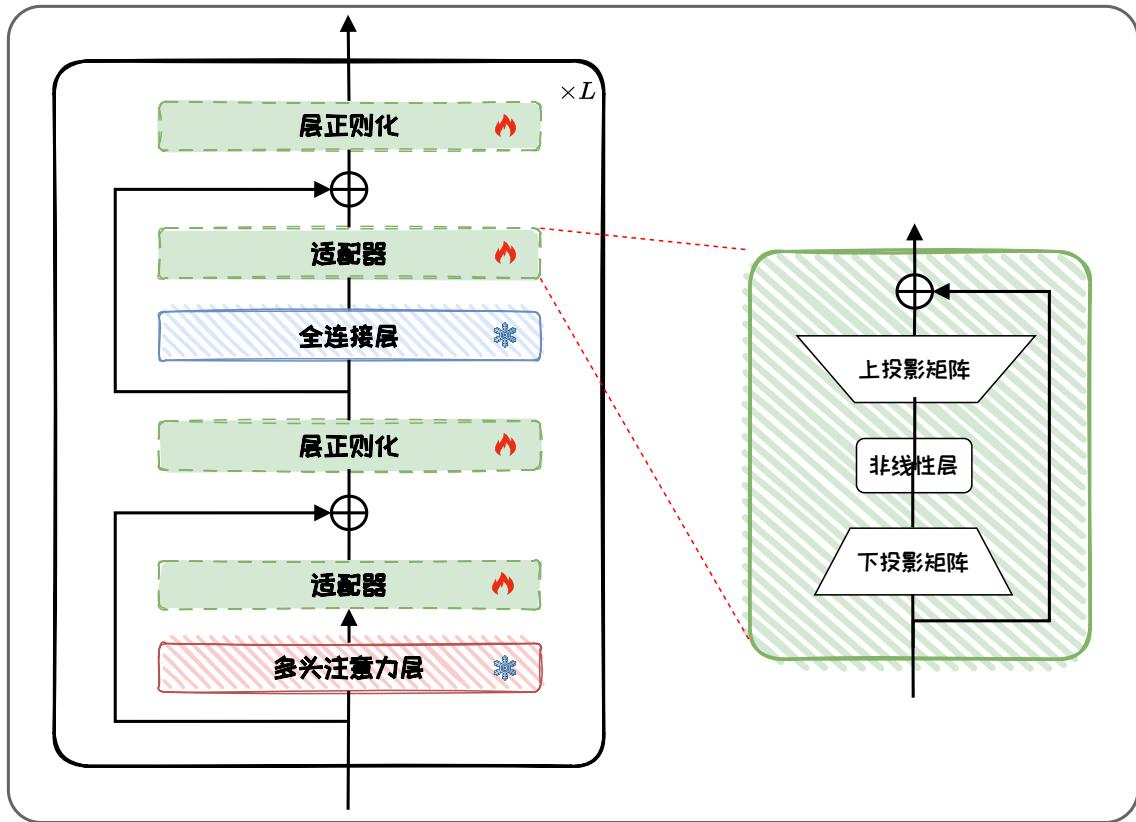


图 4.5: Adapter-tuning 示意图

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  $r \ll d$ ：

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

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：

$$\Psi \leftarrow \operatorname{argmin}_{\Psi} L_n(D_n; \Theta, \Phi_1, \dots, \Phi_N, \Psi) \quad (4.2)$$

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， $\Theta$  表示预训练模型参数， $\Phi_i$  表示第  $i$  个任务的适配器模块参数， $\Psi$  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

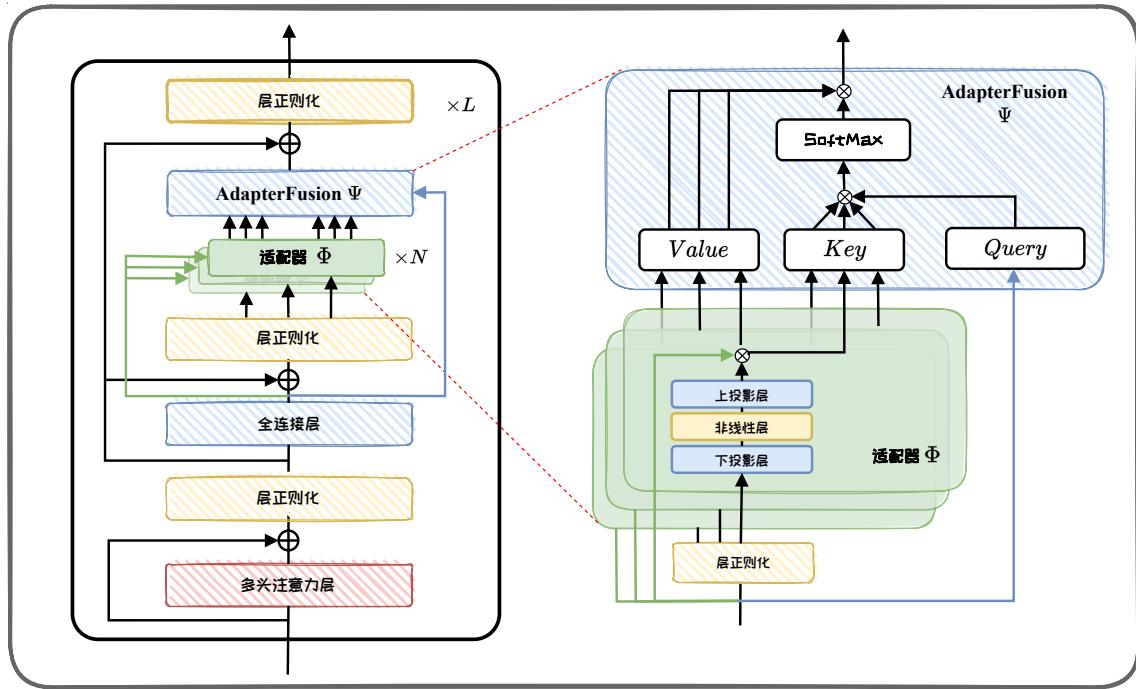


图 4.6: AdapterFusion 示意图

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。

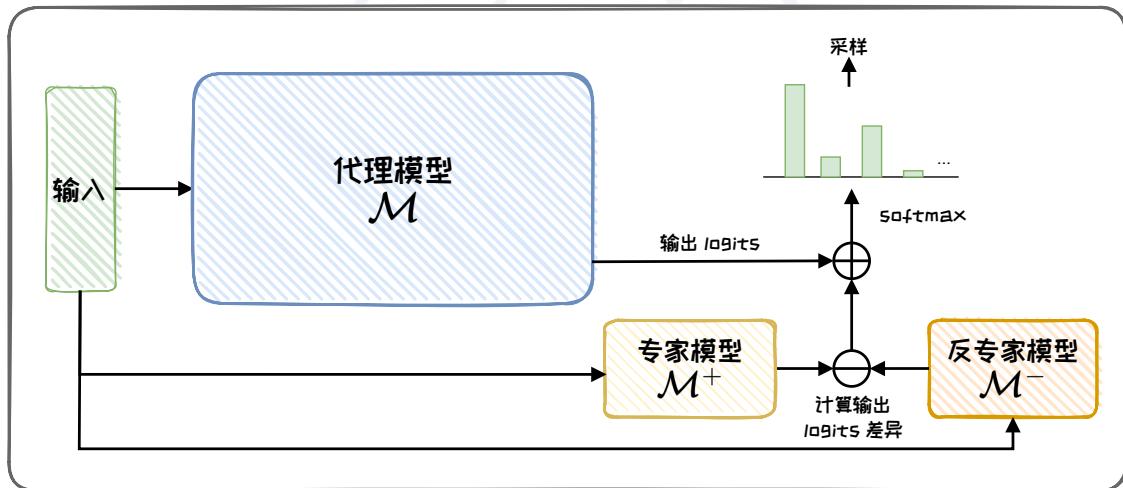


图 4.7: Proxy-tuning 示意图

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  $\tilde{s}$ :

$$\tilde{s} = s_{\mathcal{M}} + s_{\mathcal{M}^+} - s_{\mathcal{M}^-}, \quad (4.3)$$

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

$$p_{\tilde{\mathcal{M}}} (X_t | x_{$$

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：

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

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：

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

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<sub>F</sub> 和 Child-Tuning<sub>D</sub>。Child-Tuning<sub>F</sub> 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<sub>F</sub> 从伯努利分布中抽取 0-1 掩码，生成梯度掩码  $\mathbf{M}_t$ ：

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

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<sub>F</sub> 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<sub>D</sub> 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<sub>D</sub> 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：

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

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：

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

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 **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 **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， $\Delta W$  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  $\Delta W$  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：

$$W = W_0 + \alpha B A \quad (4.10)$$

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， $\alpha$  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]。

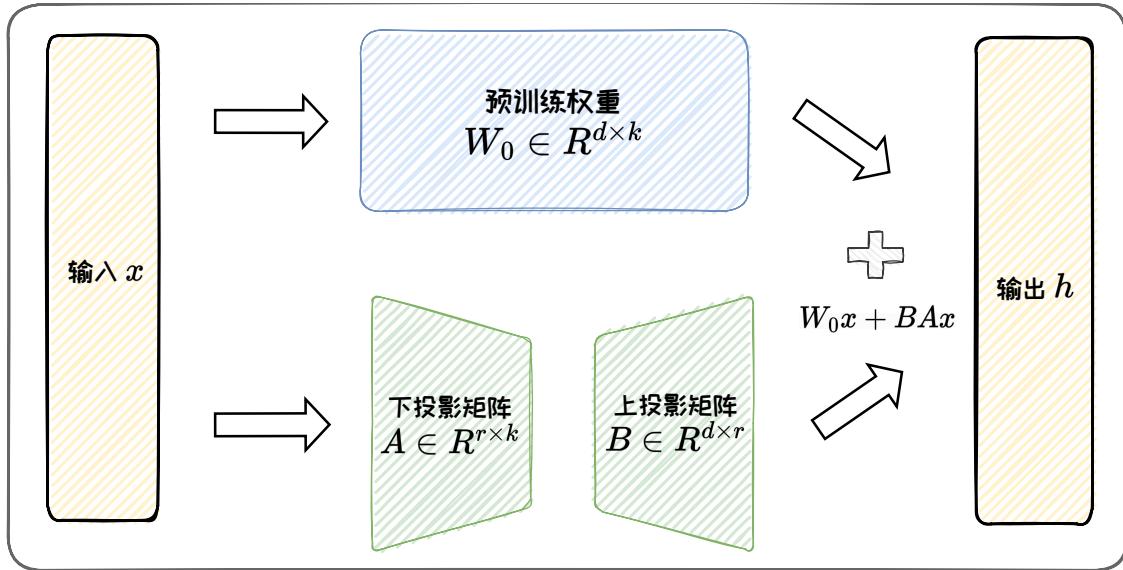


图 4.8: LoRA 示意图

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：

$$W^i \leftarrow W^i + \alpha B^i A^i \quad (4.11)$$

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， $\alpha$  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  $\Delta W$ ，即

$$W = W_0 + \Delta W = W_0 + P \Lambda Q \quad (4.12)$$

其中,  $P \in \mathbb{R}^{d \times r}$  和  $Q \in \mathbb{R}^{r \times k}$  是正交的,  $\Lambda$  是一个对角矩阵, 其中包含  $\{\lambda_i\}_{1 \leq i \leq r}$  的奇异值。在训练过程中,  $W_0$  的参数被固定, 仅更新  $P$ 、 $\Lambda$  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:

$$R(P, Q) = \|P^T P - I\|_F^2 + \|Q Q^T - I\|_F^2. \quad (4.13)$$

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:

$$W_0 = m \frac{V}{\|V\|_c} = \|W_0\|_c \frac{W_0}{\|W_0\|_c}, \quad (4.14)$$

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:

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

其中,  $\Delta V$  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

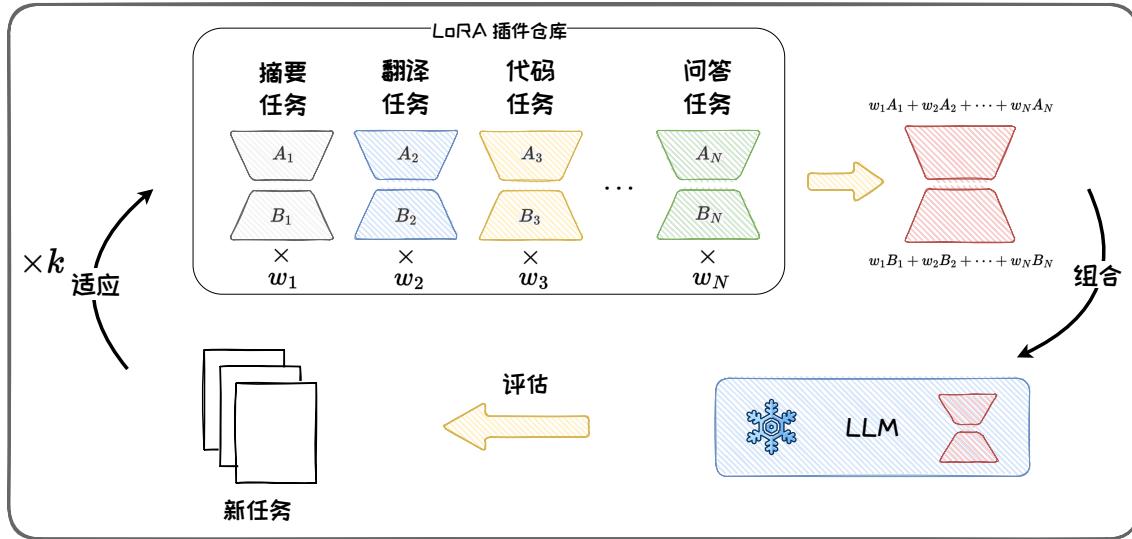


图 4.9: LoRAHub 示意图

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:

$$\hat{m} = (w_1A_1 + w_2A_2 + \dots + w_NA_N)(w_1B_1 + w_2B_2 + \dots + w_NB_N) \quad (4.16)$$

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<sup>2</sup>，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。

<sup>2</sup><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<sup>3</sup>，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**

---

<sup>3</sup><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。

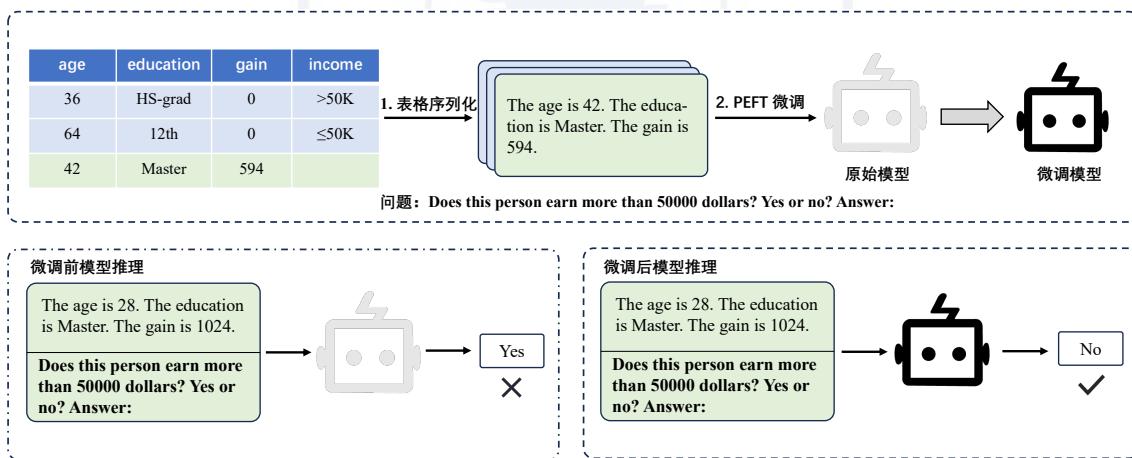


图 4.10: TabLLM 框架图

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

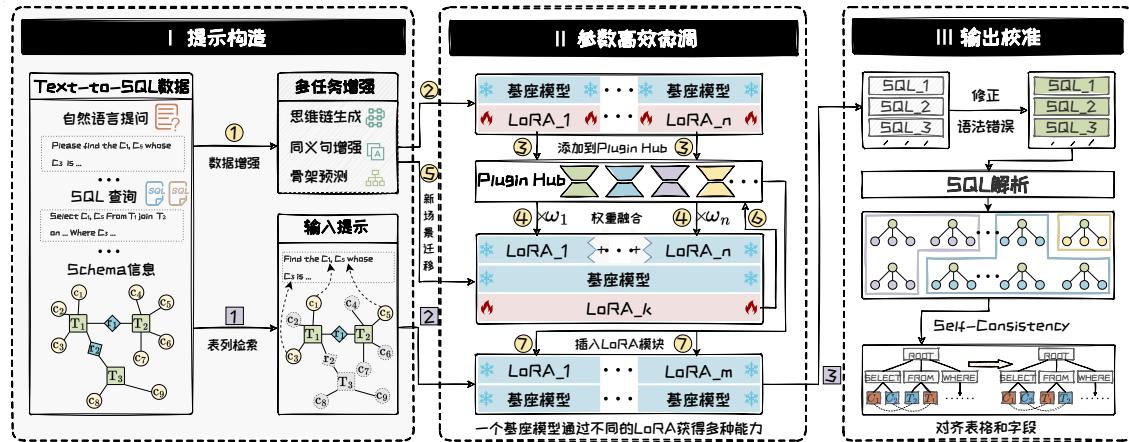


图 4.11: FinSQL 示意图

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

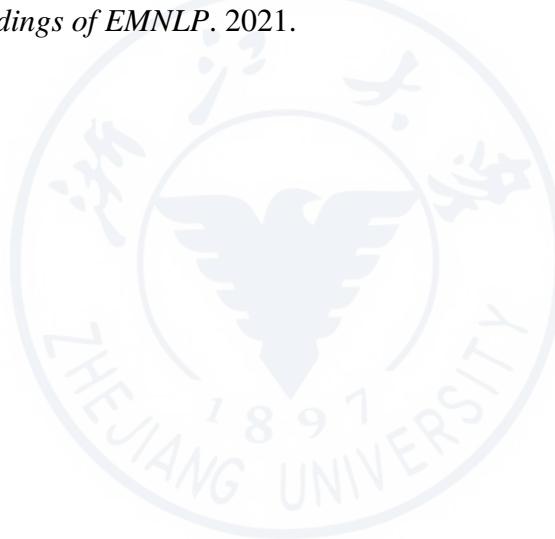
- [1] Armen Aghajanyan, Sonal Gupta, and Luke Zettlemoyer. “Intrinsic Dimensionality Explains the Effectiveness of Language Model Fine-Tuning”. In: *ACL/IJCNLP*. 2021.
- [2] Alan Ansell et al. “Composable Sparse Fine-Tuning for Cross-Lingual Transfer”. In: *ACL*. 2022.
- [3] Daniel Bershatsky et al. “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 \[cs.LG\]](https://arxiv.org/abs/2110.08207).
- [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](https://doi.org/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 Pre-trained 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.



# 5 模型编辑

在已经训练的大模型中，可能存在**偏见**、**毒性**、**知识错误**等问题。为了纠正这些问题，需要对模型中的特定**知识点**进行修正，使其能够正确掌握相关内容。在这种情形下，成本高昂的重新预训练不再可行。虽然微调方法可以有效地向大模型注入新知识，并已被广泛使用，但依然存在训练成本高、可能过拟合以及破坏现有知识等缺点。为了满足大模型的频繁更新需求，同时规避微调方法的不足，模型编辑应运而生。本章将介绍模型编辑这一新兴技术，第5.1节首先从模型编辑思想、定义、性质等方面作出简介，第5.2节分别从内外两个角度介绍模型编辑经典方法，第5.3节和第5.4节将举例介绍模型编辑的具体方法，最后，第5.5将介绍一些模型编辑的实际应用。

## 5.1 模型编辑简介

大模型中的知识错误是普遍存在的。如图5.1所示，当询问“斑马的皮肤是什么颜色的？”时，模型错误地回答了“肉色”，而实际上，斑马的皮肤是黑色的。如果不及时修正大模型的这种错误，可能会对人们造成严重的误导。

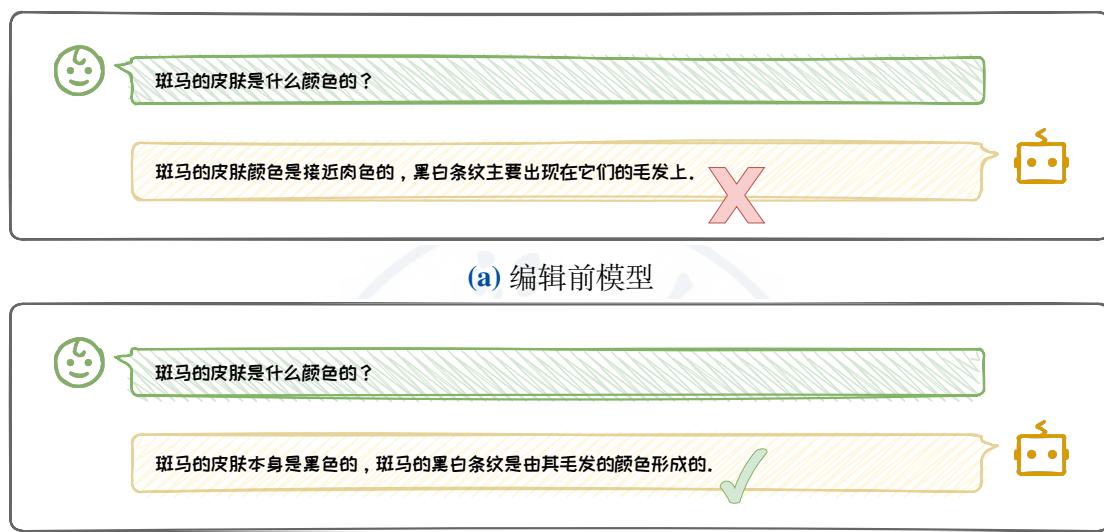


图 5.1: 模型知识错误示例

从前面几章我们可以了解到，虽然微调可以有效地向大模型注入新知识，且已被广泛使用，但是，这些方法依然存在以下缺点：

1. 尽管存在着多种参数高效微调方法，但是直接去调整十亿级参数量的模型，还是会产生极高的训练成本；
2. 当新数据集较小的时候，微调有可能在新数据上过拟合；
3. 由于微调是无限制地去修改预训练模型中的参数，因此可能会破坏模型中与更新无关、但在预训练中却有价值的知识，从而产生风险。

模型编辑作为一种新兴的解决方案，旨在高效、精准地修正大模型中的特定知识点，能够在满足大模型频繁更新需求的同时，规避微调方法的缺点。本节将从

思想、定义、性质及常用数据集四个方面对模型编辑进行初步介绍。

### 5.1.1 模型编辑思想

在《三体 2：黑暗森林》中，面壁者希恩斯和他的妻子共同研发了一种名为“思想钢印”的设备，目的是向太空军灌输“人类必胜”的坚定信念。这个机器的原理是让接受者在接触到特定信息时，跳过正常的大脑处理过程，直接输出正向答案。模型编辑的思想大致与此相似，旨在通过增加或修改模型参数，快速有效地改变模型行为和输出。

再来举个更加贴近真实生活的例子。回想一下，对于人类来说，在我们的成长阶段中，更新自身知识的方式都有哪些？

- 首先，我们倾向于去亲身体验这个世界，通过对海量知识的了解，形成自身的知识体系。这可以类比成大模型的**预训练**过程。
- 其次，我们在学校里会面向不同学科去进行专门的学习，从而提升自己在特定领域的能力。这可以类比成大模型的**微调**过程。
- 此外，在与他人交流的过程中，我们会针对特定的知识进行探讨，从而纠正自己在该知识点上错误的认知。这就可以类比成**模型编辑**的思想。

以上方式，都可以满足我们上面提到的“修正大模型错误知识”的需求。与预训练和微调不同的是，模型编辑期望能更加**快速、精准**地实现对于模型特定知识点的更新。

### 5.1.2 模型编辑定义

当前，模型编辑领域尚缺乏统一标准，不同研究对相关概念的定义各不相同。本文将不同工作中提到的**基于知识的模型编辑**（KME, Knowledge Model Editing）[25] 和**知识编辑**（KE, Knowledge Editing）[28] 等概念统一为**模型编辑**（ME, Model

Editing)。此外，有些研究用“编辑”(edit) [27]或“事实”(fact) [17, 18]来表示具体的编辑对象，本文将这些概念统一为“知识点”。

模型编辑的目标是：精确地修改预训练模型的行为以更新特定知识，同时不影响其他无关知识 [25]。

为了更精确地描述模型编辑，我们提供以下定量化定义：

### 定义 5.1 (模型编辑)

将编辑前模型定义为  $M$ ，编辑后模型定义为  $M^*$ 。每一次编辑都修改模型的一个知识点  $k$ ，知识点  $k$  由问题  $x_k$  及其对应的答案  $y_k$  组成。那么，模型编辑的目标可以表示为以下函数：

$$M^*(x_k) = \begin{cases} y_k, & \text{若 } x_k \text{ 与编辑知识点相关} \\ M(x_k), & \text{若 } x_k \text{ 与编辑知识点无关} \end{cases}$$



图5.2用“斑马皮肤颜色”这一知识点作为示例，展示了模型编辑的概念。

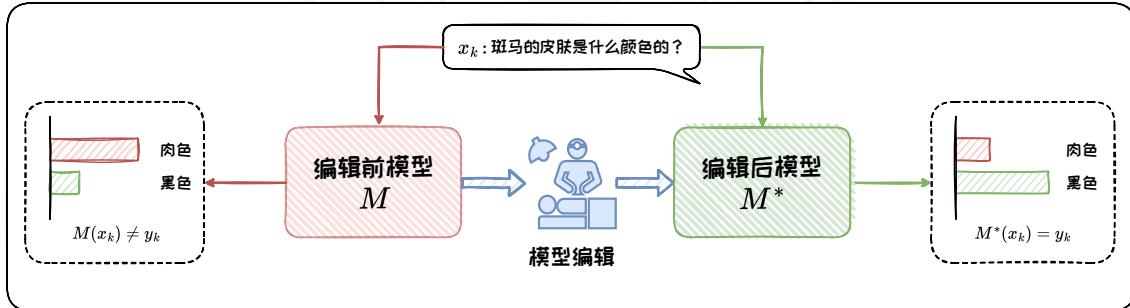


图 5.2：模型编辑概念图

然而，实际的模型编辑过程远比理论定义复杂。这主要源于知识的内在关联性：当修改模型对特定知识点的认知时，往往会产生“牵一发而动全身”的效应。因此，精确控制模型编辑的范围成为一个关键挑战。

接下来，我们将深入探讨模型编辑的范围，并系统地介绍模型编辑的关键性质。这些性质不仅反映了模型编辑的复杂性，也为评估和改进编辑方法提供了重

要指标。

### 5.1.3 模型编辑性质

根据已有工作 [16, 25, 27, 28]，可将模型编辑的性质归纳为五个方面，分别为准确性 (Accuracy)、泛化性 (Generality)、可迁移性 (Portability)、局部性 (Locality) 和高效性 (Efficiency)。图5.3通过相关示例直观地展示了这些性质之间的关系。

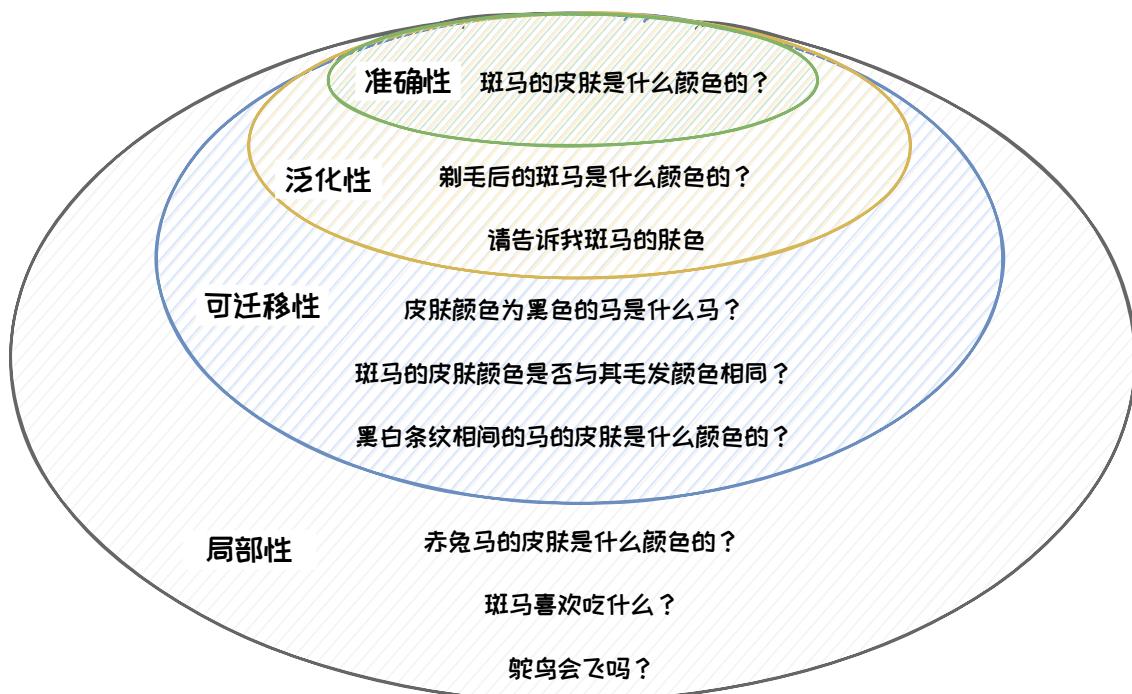


图 5.3: 模型编辑性质关系图

下面对这五种性质分别作出介绍。

#### 1. 准确性

准确性指的是对知识点  $k$  进行直接修改的成功率。前面提到，在进行模型编辑时，选取知识点  $k$  中的一对代表性输入输出  $(x_k, y_k)$  对模型进行直接修改。因此，准确性首先评估编辑后模型  $M^*$  是否能够准确回答问题  $x_k$ ，也就是“斑马的皮肤

是什么颜色的?”这个问题。如果  $M^*$  可以输出  $y_k$ ，则认为本次编辑准确。在这里，采用以下公式来表示一次编辑是否准确：

$$\text{Acc} = I(M^*(x_k) = y_k) \quad (5.1)$$

该公式通过一个指示函数  $I(\cdot)$  进行计算，仅当编辑后模型在目标知识点上的输出  $M^*(x_k)$  与目标输出  $y_k$  相匹配时， $\text{Acc}$  的值为 1，否则为 0。对于多次编辑，可采用数学期望来计算整体准确率。

**准确性是模型编辑的基本要求**，它确保了编辑后的模型能够符合设计者的预期，准确地执行特定的任务。只有当模型能够满足准确性时，才能够进一步去满足其它性质。

### 2. 泛化性

泛化性用来衡量编辑后模型是否可以适应知识点  $k$  的其他输入形式，即判断模型能否在具有语义相似性的输入上呈现一致的行为。泛化性问题对应图中黄色部分的问题“剃毛后的斑马是什么颜色的?”和“请告诉我斑马的肤色”，这两个问题的正确答案都是“黑色”。

为了评估编辑后模型的泛化性，研究者通常会构造一个**泛化数据集**  $D_G = \{(x_i, y_k)\}_{i=1}^{|D_G|}$ ，其中  $x_i$  是与  $x_k$  具有语义相似性的问题，它们的答案都为  $y_k$ 。采用以下公式来量化编辑后模型的泛化性：

$$\text{Gen} = \frac{1}{|D_G|} \sum_{i=1}^{|D_G|} I(M^*(x_i) = y_k) \quad (5.2)$$

当  $\text{Gen}$  的值为 1 时，说明编辑后模型能够正确回答  $D_G$  中的所有问题，此时泛化性最好。

**确保编辑模型的泛化性可以防止编辑后模型过度拟合特定的输入**，从而保证模型对于知识点的理解。

### 3. 可迁移性

可迁移性是指编辑后的模型在遇到与目标知识相关联的新情境时的适应性。可迁移性问题主要包括反向问题、推理问题和实体替换问题。在图中蓝色部分的问题中，“皮肤颜色为黑色的马是什么马？”是一个反向问题，“斑马的皮肤颜色是否与其毛发颜色相同？”是一个推理问题，“黑白条纹相间的马的皮肤是什么颜色的？”则是一个实体替换问题。这三个问题的答案都不是“黑色”。

构造可迁移性数据集  $D_P = \{(x_i, y_i)\}_{i=1}^{|D_P|}$ ，其中  $x_i$  是与  $x_k$  相关的问题，但与泛化数据集中的问题不重叠， $y_i$  为其对应答案， $y_i \neq y_k$ 。采用以下公式来量化编辑后模型的可迁移性：

$$\text{Port} = \frac{1}{|D_P|} \sum_{i=1}^{|D_P|} I(M^*(x_i) = y_i) \quad (5.3)$$

当 Port 的值为 1 时，说明模型可以正确回答可迁移数据集中的所有问题，此时模型的可迁移性最好。可迁移性对于模型编辑的实际应用至关重要。然而，大多数模型编辑方法往往无法实现较好的可迁移性，这也是模型编辑中的一个挑战。

### 4. 局部性

局部性要求编辑后的模型不影响其他不相关输入的输出。局部性问题对应图中灰色部分的问题“赤兔马的皮肤是什么颜色的？”、“斑马喜欢吃什么？”等等。一般来说，研究者会在常用的常识数据集中选择一些与知识点  $k$  不相关的问题作为局部性评估。将局部性数据集定义为  $D_L = \{(x_i, M(x_i))\}_{i=1}^{|D_L|}$ ，其中  $x_i$  是与  $x_k$  无关的问题。采用以下公式来量化编辑后模型的局部性：

$$\text{Loc} = \frac{1}{|D_L|} \sum_{i=1}^{|D_L|} I(M^*(x_i) = M(x_i)) \quad (5.4)$$

当 Loc 的值为 1 时，编辑后模型的局部性最好，此时，编辑后模型对局部性数据集中所有问题的回答与编辑前一致。

确保局部性能够降低模型编辑的副作用，同时也是模型编辑和微调之间的主

要区别。

### 5. 高效性

高效性主要考虑模型编辑的时间成本和资源消耗。在实际应用中，模型可能需要频繁地进行更新和纠错，这就要求编辑过程必须足够快速且资源友好。此外，对于大量的编辑任务，不同方法的处理效率也有所不同，有的方法支持批量并行编辑 [3, 7, 18–20]，有的则需要依次进行 [5, 13, 17]。高效性直接影响到模型编辑的可行性和实用性。

在评估模型编辑方法时，需要在这五个性质之间寻找平衡。理想的编辑方法应当在保证准确性的基础上，尽可能地提高泛化性、可迁移性和局部性，同时保持较高的效率。

#### 5.1.4 常用数据集

前文探讨了模型编辑的定义与性质。接下来，我们将介绍一些先前研究中使用过的具体数据集，这些数据集可用于测试和比较不同的模型编辑技术。

在模型编辑的相关研究中，使用最广泛的是由 Omer Levy 等人提出的 **zsRE 数据集** [14]。zsRE 是一个问答任务的数据集，通过众包模板问题来评估模型对于特定关系（如实体间的“出生地”或“职业”等联系）的编辑能力。在模型编辑中，zsRE 数据集用于检查模型能否准确识别文本中的关系，以及能否根据新输入更新相关知识，从而评估模型编辑方法的准确性。

2023 年，[17] 提出了 **COUNTERFACT 数据集**，被后续工作广泛采用。COUNTERFACT 被设计用来区分两种类型的知识修改：一种是词汇的表面变化，也就是文本中单纯的词语替换或结构调整，不会影响信息的实质内容；另一种是对基础事实知识显著且泛化的修改，也就是对文本中所描述的事实或信息进行根本性的改变，这通常需要更深层次的理解和处理能力。COUNTERFACT 能够评估模型对

编辑后知识的理解和反应，进而衡量模型的泛化性和局部性。

文献 [27] 在 ZsRE 和 COUNTERFACT 数据集的基础上，使用 GPT-4 生成相应问题的反向问题、推理问题和实体替换问题，构造了可迁移性数据集。

此外，研究者还针对不同领域和任务开发了专门的数据集，如 Hallucination[11] 用于纠正 GPT 语言模型中的幻觉，ConvSent[20] 用于评估模型在修改对话代理对特定主题的情感时的效果。表 5.1 总结了一些模型编辑相关数据集。

表 5.1：模型编辑相关数据集总结表格

数据集	类型	训练集数量	测试集数量	输入	输出
ZsRE[14]	知识关联	244,173	244,173	事实陈述	对象
COUNTERFACT[17]	知识关联	N/A	21,919	事实问题	对象
WikiGen[19]	文本生成	N/A	68,000	Wiki 段落	续写
T-REx-100/-1000[7]	知识关联	N/A	100/1,000	事实陈述	对象
ParaRel[5]	知识关联	N/A	253,448	事实问题	对象
NQ-SituatedQA[6]	问答	N/A	67,000	用户查询	答案
MQuAKE-CF/-T[29]	知识关联	N/A	9,218/1,825	多跳问题	对象
Hallucination[11]	幻觉	N/A	1,392	传记	传记
MMEdit-E-VQA[4]	多模态	6,346	2,093	图像问题	答案
MMEdit-E-IC[4]	多模态	2,849	1,000	图像描述	描述
ECBD[23]	知识关联	N/A	1,000	实体完成	引用实体
FEVER[3]	事实检查	104,966	10,444	事实描述	二进制标签
ConvSent[20]	情感分析	287,802	15,989	主题意见	情感
Bias in Bio[12]	人物传记	5,000	5,000	传记句子	职业
VitaminC-FC[20]	事实检查	370,653	55,197	事实描述	二进制标签
SCOTUS[11]	分类	7,400	931	法庭文件	争议主题

这些数据集涵盖了从事实检查、知识关联到特定领域等多种类型，体现了模型编辑技术在不同场景中的应用潜能。通过这些数据集，研究者可以更深入地理解模型编辑的挑战和潜力，思考如何进一步提升编辑技术的精确度和效率。

本节介绍了模型编辑的定义、性质、评估方法以及常用数据集，对模型编辑任务做出了详细的解释和说明。接下来，第 5.2 节将对模型编辑的方法进行系统性概述，将其分为引入外部参数法和修改内部参数法，并介绍近年来的代表性工作。第 5.3 节和第 5.4 节将分别深入探讨外部参数法中的 T-Patcher 方法和定位编辑法中的 ROME 方法，通过这两种代表性方法，帮助读者更加细致地理解模型编辑方法的研究过程和思路。最后，第 5.5 节将全面介绍模型编辑的实际应用，并分别举例说明解决思路。

## 5.2 模型编辑经典方法

参考已有工作 [16, 25, 27, 28]，我们将现有的编辑方法分为外部拓展法和内部修改法。

2020 年，David Bau 等人 [1] 首先在计算机视觉领域进行模型编辑的相关探索，提出了将模型网络层参数等效为线性记忆的观点，指出可以通过精确地修改参数来更新模型知识。之后，内部修改法主要参考了这种思想。

3 个月后，Anton Sinitisin 等人 [24] 介绍了一种可编辑的训练。该工作提出了编辑器的概念，通过在目标知识的输入子集上训练编辑器，实现了对模型知识的精确修改，同时避免了对原始模型和不相关输入的影响。之后，外部拓展法主要参考了这种思想。

概括来说，外部拓展法通过设计特定的训练程序，使模型在保持原有知识的同时学习新信息。内部修改法利用模型的结构特性，通过调整某些特定层或神经元，

来实现对模型输出的精确控制。我们在图 5.4 中给出了模型编辑方法的分类：在外部拓展法中，包括**知识缓存法**和**附加参数法**，在内部修改法中，包括**元学习法**和**定位编辑法**。接下来，我们将对每类编辑方法展开介绍。

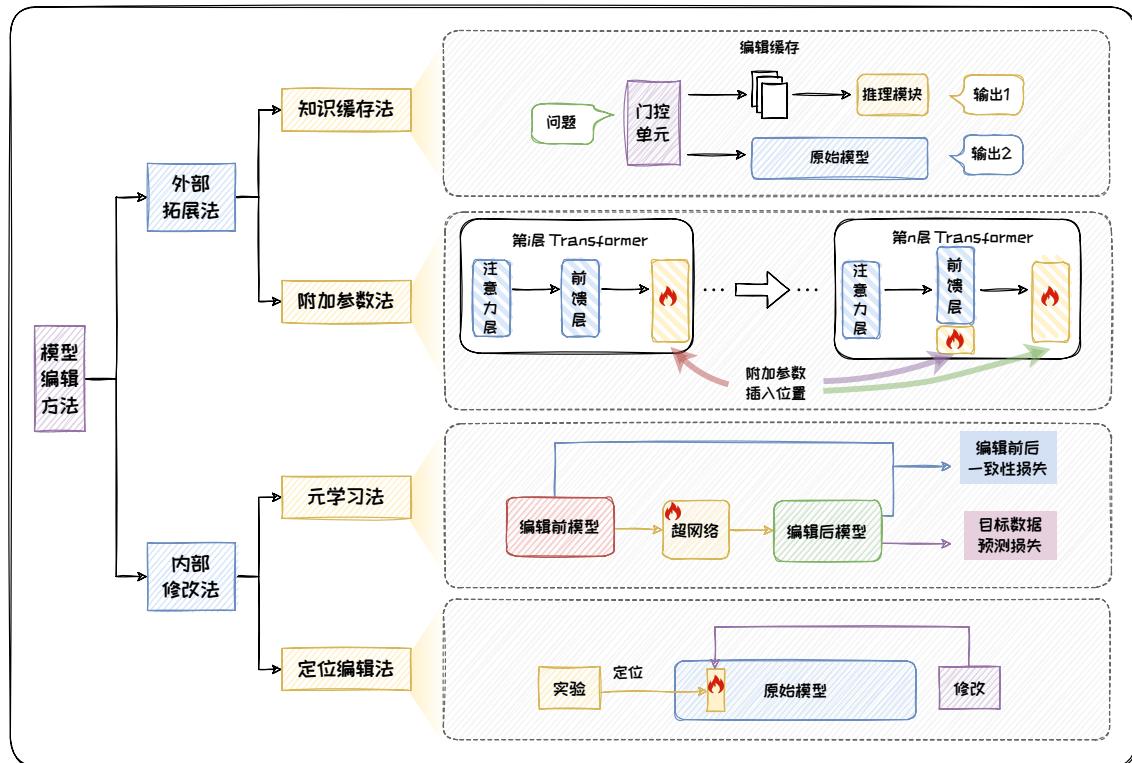


图 5.4: 模型编辑方法分类图

### 5.2.1 外部拓展法

如果将大模型比作冒险游戏中的勇者，外部组件则可视作勇者的背包，这个背包使勇者能够收集新道具来增强能力，同时保留原有技能。**外部拓展法的核心思想是将新知识存储在附加的外部参数或外部知识库中，将其和原始模型一起作为编辑后模型**。这种方法尤其适合具有良好可扩展性的预训练大模型，因为它提供了足够的空间来容纳大量参数，能够存储更多新知识。此外，该方法不会改变原始模型参数，可以将对预训练知识的影响降至最低。

根据引入的参数是否直接整合进模型的运作流程，外部参数法又可划分为两种，分别是**知识缓存法**和**附加参数法**。延续冒险游戏的比喻，知识缓存类似于勇者的技能书，需要时可以查阅获取特定知识；而附加参数则如同可升级的装备，直接增强勇者的整体能力。

接下来将分别对这两种方法进行举例介绍。

### 1. 知识缓存法

在知识缓存法中，外部缓存充当一个知识存储库，用于保存需要修改的知识，可将其称为**编辑缓存**。图 5.5 为知识缓存法示意图。这类方法训练了一个**门控单元**和一个**推理模块**，其中门控单元用来判断输入的问题是否与编辑缓存相关，如果相关，则从编辑缓存中获取相关编辑实例，将编辑实例与输入一并交给推理模块，由推理模块给出答案；如果不相关，则用原始模型去推理给出答案。

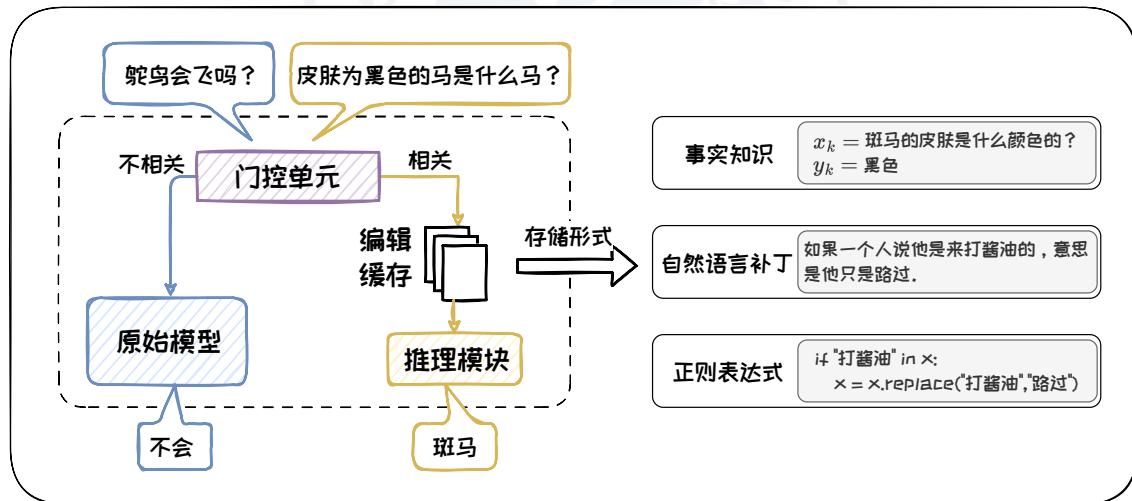


图 5.5: 知识缓存法示意图

关于编辑实例的存储形式，可以分为**事实知识**、**自然语言补丁**和**正则表达式**三种。

- 事实知识指以问题-答案对  $(x_k, y_k)$  存储编辑实例的形式，这种存储形式适用于答案明确的事实性问题，**SERAC[20]** 是一种代表性方法。

- 自然语言补丁由文献 **Language Patch**[21] 提出，其按照“如果……那么……”的句式描述编辑知识，类似于 Prompt。这种存储形式适用于修正模型对自然语言中非字面含义语句的理解，且便于人们创建、编辑或移除补丁，使得模型能够通过人类反馈不断修正输出。
- 正则表达式是一种基于文本匹配和替换的技术，它使用特定的模式来识别和修改文本中的特定部分，适用于精确的文本语义替换。这是一种早期的原始方法，然而由于编写复杂、泛化性低，因此在模型编辑中并不常用。

知识缓存法直接通过编辑缓存中的信息进行检索，不依赖目标标签的梯度信息，因此可以简化模型编辑过程，使其更加高效直接。

## 2. 附加参数法

与知识缓存法相比，附加参数法可以将外部参数整合进模型结构，使模型获得更新的编辑知识，从而有效利用和扩展模型的功能。图 5.6 展示了这类方法的基本原理：通过冻结原始模型，只训练新引入的参数，来修正模型的输出。

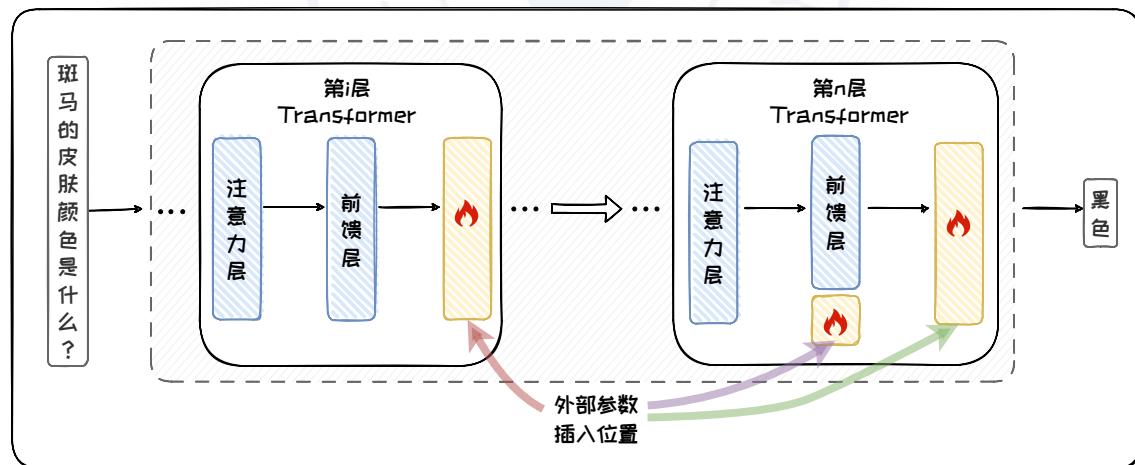


图 5.6: 附加参数法示意图

不同方法将外部参数插入到模型的不同位置。例如，**CALINET**[7] 和 **T-Patcher**[13] 都是通过修改模型最后一层 Transformer 的前馈层 (FFN) 来实现。CALINET 首先通过一种对比知识评估方法找出原始模型的知识错误，然后在模型最后一个 FFN

层添加一个新的参数矩阵，并通过最小化校准数据上的损失来训练新参数，以纠正模型的错误。T-Patcher 的思想与此类似，同样是在原始模型的最后一个 FFN 层引入了有限数量的可训练神经元，每个神经元对应一个知识错误。关于 T-Patcher 的具体介绍将在第 5.3 节给出。

**GRACE**[11] 则将外部参数以适配器的形式插入特定位置的 Transformer 中，插入位置随模型变化而变化。其中，适配器是一个用于缓存错误知识（Keys）和对应的修正值（Values）的键值存储体，被称作“codebook”。在 codebook 中，每个错误知识都有一个对应的修正值，以及一个用于匹配相似输入的延迟半径，且随着时间的推移，codebook 会被持续更新。延迟半径用于判断当前输入是否与 codebook 中的任何错误相似，如果是，则应用相应的修正值进行编辑。

总结来说，知识缓存法通过引入编辑缓存来帮助模型检索新知识，附加参数法通过引入额外参数来实现对模型特定输出的精细调整。这两种方法的核心优势在于对原始模型的最小化干预，保证了模型编辑的局部性。

然而，外部参数法的有效性在很大程度上依赖于对知识的存储与检索能力，从而增加存储资源需求。因此，在具体应用时，我们需要在**保证模型局部性和应对存储限制**之间寻求平衡。

### 5.2.2 内部修改法

通过外部拓展法，勇者可以依赖神奇的道具和装备来提高自己的能力。而内部修改法则相当于直接去锻炼自身，从自我层面获得提升。**内部修改法旨在通过更新原始模型的内部参数来为模型注入新知识**。与外部拓展法相比，内部修改法不需额外存储空间，且能够使模型将新知识内化为自己的一部分，实现更深层次的学习和适应。

内部修改法又可以分为**元学习法**和**定位编辑法**。其中，元学习法通过一个中

间模型来学习如何更新原始模型的参数，从而获得参数更新的增量值  $\Delta$ ；而定位编辑法则专注于对模型局部参数的修改，首先识别与目标知识最相关的模型参数，然后仅更新这些特定参数，通过“先定位再编辑”的策略节省更新模型所需的成本。

接下来，我们将分别对这两种方法进行举例介绍。

## 1. 元学习法

元学习指的是模型“学习如何学习”的过程。元学习法旨在设计能够学习“学习过程”本身的算法，核心思想是使模型能够从一系列不同的任务中提取通用的知识或模式，并能够将这些知识应用到新的、未见过的任务上。元学习的目的是提高模型的适应性和泛化能力，使其能够快速学习新任务，适合用于训练数据较少的场景。在模型编辑场景下，需要编辑修改的事实数据量往往较小，因此可以采取元学习法提高模型能力。

在模型编辑中，基于元学习的方法往往利用中间模型获得更新参数，这种中间模型通常被称为**超网络**（Hyper Network）[10]。这类方法的示意图如图 5.7 所示。

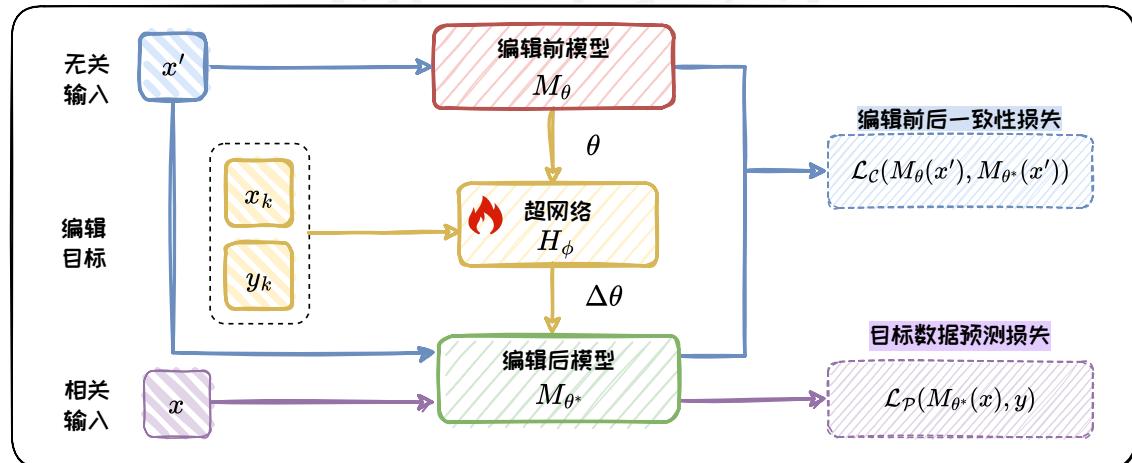


图 5.7: 元学习法示意图

具体来说，记原始模型为  $M_\theta$ ，其中  $\theta$  表示原始模型的参数。同理，记超网络模型为  $H_\phi$ 。注意， $|\phi| \ll |\theta|$ ，这是因为我们需要在训练超网络的过程中确保效率。

超网络将编辑目标  $(x_k, y_k)$  和原始模型的参数  $\theta$  作为输入, 生成参数变化量  $\Delta\theta$ , 原始模型再使用  $\Delta\theta$  来更新自身, 得到编辑后模型  $M_{\theta^*}$ , 从而将输入数据映射到输出目标。模型参数更新过程可以表示如下:

$$\begin{aligned}\theta^* &= \theta + \Delta\theta \\ &= \theta + H_\phi(\theta, x_k, y_k)\end{aligned}\tag{5.5}$$

用  $D_P$  和  $D_L$  分别表示编辑目标  $(x_k, y_k)$  的可迁移性数据集和局部性数据集, 即与  $(x_k, y_k)$  相关和无关的输入-输出对集合。在超网络的训练过程中, 我们希望优化以下两个目标:

1. 对于编辑目标数据集  $D_P$ , 最小化编辑后模型的预测误差。定义损失函数  $\mathcal{L}_P$  来衡量  $M_{\theta^*}$  在  $D_P$  上的预测误差, 对于所有的  $(x, y) \in D_P$ , 我们希望最小化编辑后模型  $M_{\theta^*}$  的损失:

$$\mathcal{L}_P(M_{\theta^*}(x), y) = \mathcal{L}_P(M_{\theta+H_\phi(\theta, x_k, y_k)}(x), y)\tag{5.6}$$

2. 对于局部性数据集  $D_L$ , 保持编辑前后模型的输出一致。定义一致性损失  $\mathcal{L}_C$  来度量编辑前后模型的预测一致性, 对于所有的  $(x', y') \in D_L$ , 我们希望编辑前后模型的预测尽可能一致,

$$\mathcal{L}_C(M_{\theta^*}(x'), M_\theta(x')) = \mathcal{L}_C(M_{\theta+H_\phi(\theta, x_k, y_k)}(x'), M_\theta(x'))\tag{5.7}$$

综上所述, 超网络的训练过程可以表示为以下的约束优化问题:

$$\begin{aligned}\operatorname{argmin}_\phi \quad & \sum_{(x, y) \in D_P} \mathcal{L}_P(M_{\theta+H_\phi(\theta, x_k, y_k)}(x), y) \\ \text{subject to} \quad & \sum_{(x', y') \in D_L} \mathcal{L}_C(M_{\theta+H_\phi(\theta, x_k, y_k)}(x'), M_\theta(x')) \leq \epsilon\end{aligned}\tag{5.8}$$

其中  $\epsilon$  是一个小的正数, 用于控制编辑后模型和原始模型在  $D_L$  上的输出一致性。

**KE**[3] 是一种采用超网络进行模型编辑的经典例子。KE 使用单层双向 LSTM

和多层 FFN 作为超网络，在约束函数中通过 KL 散度来计算编辑前后模型输出的差异，并设置了边界值来表示约束的严格程度。然而，由于庞大的参数大小，超网络通常无法更新大型语言模型。为了增强超网络对于大型语言模型的普适性，MEND[19] 通过低秩分解的方法来优化模型参数更新过程。首先，它利用全连接层中梯度的秩-1 特性，将损失函数关于每一层参数的梯度分解为两个向量的乘积。然后，使用超网络接收分解后的向量作为输入，并输出经过编辑的新向量。最后，这些新向量再次相乘，得到新的参数梯度，并通过一个可学习的缩放因子来计算最终的模型参数更新值。MEND 以较少的参数高效地编辑大型模型，节省了计算资源和内存。

总结来说，基于元学习法的模型编辑方法通过超网络的学习来更新模型参数，使模型能够快速适应新任务，提高编辑的泛化性和可迁移性，适用于训练数据较少的场景。同时，约束条件的限制可以保证编辑的局部性。然而，基于元学习法的模型编辑方法依赖于超网络的设计，且不同架构的原始模型往往需要设计不同的超网络，这使得设计过程变得繁琐。此外，超网络的训练要求精细的调试和大量的计算资源，过程较为复杂。

## 2. 定位编辑法

与元学习法相比，定位编辑法所修改的是原始模型的局部参数，这就要求我们能够首先定位到修改参数的位置。

为了实现定位，我们首先要知道，知识是如何存储的？这个问题与大语言模型的可解释性相关，尽管目前的研究尚未明确回答这个问题，但已经发现了一些模型参数与知识存储的相关性。

文献 [8] 提出了这样一个观点：Transformer 中的 FFN 层可以看作键值存储体。该工作首先提出假设，然后通过实验进行验证。具体来说，将 FFN 层的输入称为查询（query）向量，代表当前句子的前缀；将 FFN 的上投影矩阵中的向量称为键

(key) 向量, 下投影矩阵中的向量称为值 (value) 向量。

文献 [8] 中的实验在一个 16 层 Transformer 的语言模型上进行, 图 5.8 展示了在底层 FFN 层进行实验的过程。首先随机采样每层的 key, 然后计算训练集中的所有 query 与 key 的内积, 找出值最大的 query 并整理, 发现这些句子前缀都具有相同或相似的模式。其中, 底层的 key 表示浅层模式, 如重复的 n-gram; 高层的 key 表示深层模式, 如重复的语义。

随后, 实验将 value 与输出嵌入层 (Output Embedding Layer) 矩阵相乘, 并应用 softmax 函数转换为概率分布, 然后比较这些分布与对应句子前缀的下一个词的相关性, 结果发现二者高度相关。也就是说, 模型通过 key 总结了当前 query 所代表的句子前缀的特征, 又通过类似键值匹配的机制查找到了相应的 value, 也就是下一个词的概率分布。因此, 该工作得出了可以将 Transformer 中的 FFN 层可以看作键值存储体的观点。

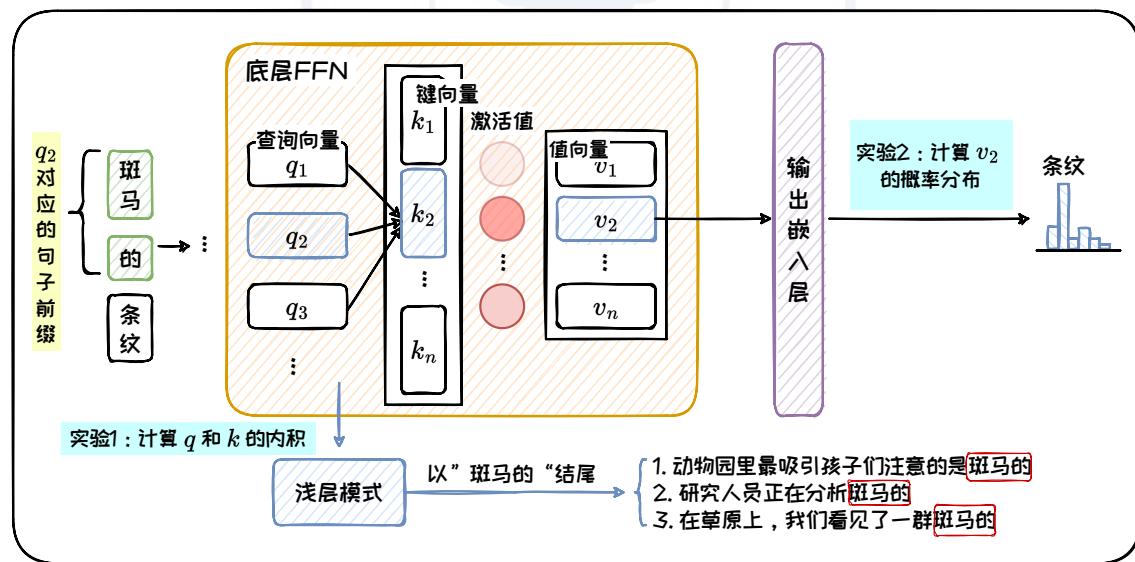


图 5.8: Transformer 可看作键值存储体

基于这个观点, KN[5] 提出了知识神经元的概念。将每层 FFN 中的每个中间激活值定义为一个知识神经元, 计算该神经元从初始值 0 变化到预训练模型中的

值时，模型输出概率的总变化量。通过对于每个神经元梯度变化的分析，可以找出对每个问题输出贡献最大的神经元，从而将其定位为该知识点的知识神经元。最终，可以得到一个知识神经元的集合。KN 还进一步进行了实验，发现如果直接在模型中修改 key，会引发模型输出的改变，从而达到模型编辑的效果。

此外，ROME[17] 设计了一种因果跟踪实验，用来确定中间层 FFN 模块与知识的关系。在编辑方法上，与定位和编辑 FFN 层中的单个权重的 KE 不同，ROME 提出了更新整个 FFN 层来进行编辑的方法。ROME 在 GPT-J 模型上进行编辑，在准确性、泛化性和局部性方面都保持着优势，成为近年来备受瞩目的模型编辑方法，为未来的模型编辑和优化工作提供了重要的参考和指导。我们将在第 5.4 节详细介绍 ROME 中对于因果跟踪实验和编辑方法的设计。MEMIT[18] 在 ROME 的基础上扩展到对不同知识的大规模编辑，可以同时执行数千次编辑。

总的来说，定位编辑法修改大模型的局部参数，在保持模型整体结构和性能的同时，能够对特定知识点进行精准的更新和编辑。与其他方法相比，定位编辑法可同时保持较高的准确性、泛化性和局部性，且适用于各类模型。

### 5.2.3 方法比较

文献 [27] 总结并比较了代表性的现有模型编辑方法。其中，不仅包括对不同方法在不同数据集和不同模型上的编辑效果测试实验，还包括顺序编辑测试、批量编辑测试等实验。有关实验设置，在此不详细展开。我们根据文献，将已有方法对比性质总结如表 5.2 所示，其中，用“高”、“中”、“低”三个级别定性表示该方法的准确性、泛化性、可迁移性和局部性，用“✓”和“✗”来表示该方法的高效性，这里的高效性指是否支持批量编辑。标注“-”的表明未进行测试。

通过表格可以看出，各类方法都有其独特的优势和局限。

在外部拓展法中，基于知识缓存的 SERAC 在无需额外训练的情况下提供了

表 5.2: 模型编辑方法比较

		方法	准确性	泛化性	可迁移性	局部性	高效性
外部 拓展 法	知识缓存法	SERAC	高	高	低	高	✓
	附加参数法	CaliNET	低	低	-	中	✓
		T-Patcher	高	高	高	中	✗
内部 修改 法	元学习法	KE	低	低	-	高	✓
		MEND	中	高	中	高	✓
	定位编辑法	KN	中	低	-	中	✗
		ROME	高	高	高	高	✗
		MEMIT	高	高	高	高	✓

高效的编辑能力，保证了高准确性、泛化性和局部性，适合快速响应和批量编辑，但可迁移性较差，编辑缓存和推理模块仍有待优化。基于附加参数的 CaliNET 和 T-Patcher 提供了对模型的直接编辑能力，但 CaliNET 对不同模型和数据的适应性较差，而 T-Patcher 虽然保持了高准确性、泛化性和可迁移性，但不支持批量编辑。由于附加参数法是针对模型结构进行特定设计的，因此不同模型的编辑效果可能存在较大差距。

在内部修改法中，基于元学习的 KE 和 MEND 提供了一种通过超网络学习权重更新的方法，但可能需要较多的计算资源，因此在各个性质上的表现都一般。然而，提前训练好的超网络使这两种方法支持批量编辑。基于定位编辑的 KN、ROME 和 MEMIT 则专注于精确定位和编辑模型内部的特定知识。ROME 和 MEMIT 都能保证高准确性、泛化性、可迁移性和局部性，然而这两种方法主要针对 decoder-only 模型设计，因此未比较其在 encoder-decoder 架构模型上的表现。此外，MEMIT 在 ROME 的基础上针对批量编辑进行了优化，在满足高效性的同时依然能够保证其它性质的稳定。

值得注意的是，除了 SERAC 之外，其它方法都是针对模型的 FFN 层进行参数

的添加或修改，这表明研究者们认为 FFN 层是实现有效模型编辑的关键点。

本节对模型编辑领域不同类别的方法进行了概述和举例介绍。这些方法展示了对大模型进行有效修正的多种途径，各自有着独特的优势和局限性。在实际应用中，应根据需求、可用资源和期望的编辑效果选取合适的方法。根据表格，T-Patcher 和 ROME 这两种方法在准确性、泛化性、可迁移性和局部性上表现较优。因此，接下来的两节将以这两种方法为代表，更加细粒度地解释模型编辑的内部机制。

### 5.3 附加参数法：T-Patcher

汽车需要定期更新零部件来提升性能，同样地，当模型的回答不符合预期时，也需要对其进行“升级”以增强其能力。为了使模型学习新的知识，可以采取类似于打补丁的方法，即向模型中添加一些新的可训练参数。这类方法中比较有代表性是 T-Patcher[13]。如图 5.9，T-Patcher 在模型的 FFN 层中添加可训练参数——“补丁”，来精准调整模型的响应，在几乎不增加额外计算负担的情况下，对模型输出进行修正。

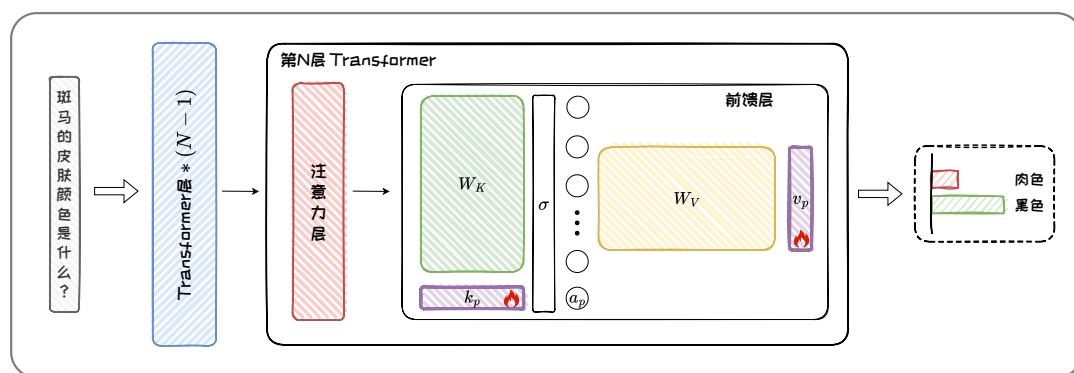


图 5.9: T-Patcher 流程图

### 5.3.1 补丁的位置

为了能够在有效进行模型编辑的同时与现有知识相适应，首先需要确定添加参数的位置。

上文已经提到，Transformer 层中的 FFN 可以被看作一个 **键值存储体**。在这一假设下，T-Patcher 将 FFN 的输入视为一个查询向量，其中的两个矩阵可分别被视为键向量矩阵  $W_K = [k_1, k_2, \dots, k_n]$  和值向量矩阵  $W_V = [v_1, v_2, \dots, v_n]$ 。其中，每个键向量对应着输入文本中的特定模式，如 N-gram 或语义主题，而每个值向量则关联着模型输出的概率分布。

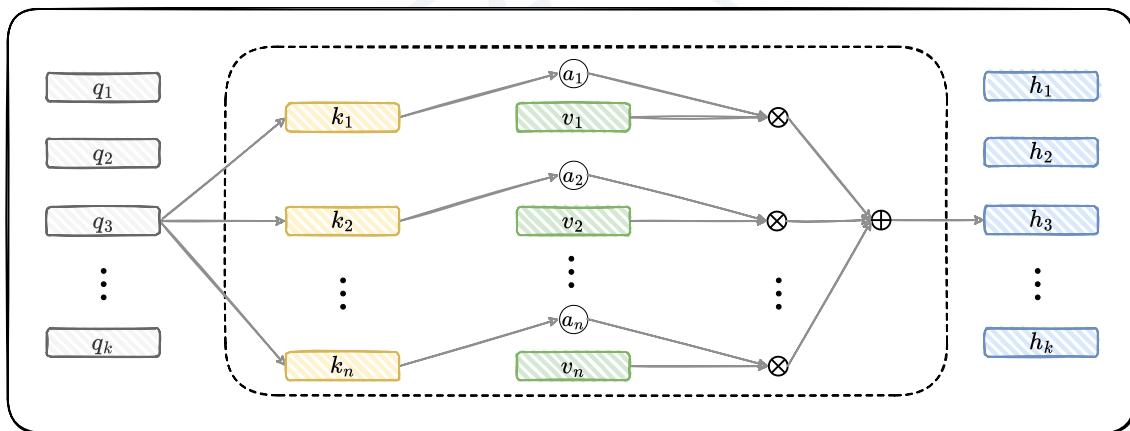


图 5.10: 键值存储体

如图 5.10，当查询向量  $q$  输入 FFN 时，先与矩阵  $W_K$  相乘，计算出激活值向量  $a$ ，其中的每一个分量代表  $q$  与对应键向量的关联程度。随后，向量  $a$  会与矩阵  $W_V$  相乘，得到 FFN 的输出结果。这个过程可被看作以激活值作为权重，对矩阵  $W_V$  中的所有值向量进行加权求和。给定查询向量  $q$ ，FFN 的输出为：

$$a = \text{Act}(q \cdot W_K + b_k) \quad (5.9)$$

$$FFN(q) = a \cdot W_V + b_v \quad (5.10)$$

其中， $\text{Act}$  表示激活函数， $b_v$  和  $b_k$  是偏置项。在这个过程中，**FFN 的中间输出**

出维数可被理解为其“记忆”的文本模式的数量。

根据以上假设，如果能够向 FFN 中添加更多与知识相关联的键值对，理论上就可以向模型中插入新的事实信息，实现模型编辑。T-Patcher 正式基于这样的思路，在 FFN 层增加额外参数，即添加补丁，在不破坏原有知识的前提下，针对新知识训练补丁参数，实现知识的注入。

### 5.3.2 补丁的形式

T-Patcher 在 FFN 的两个参数矩阵中分别增加一个向量，即新的键向量和值向量，这部分参数被称为补丁。添加补丁后，FFN 的输出被调整为：

$$\begin{bmatrix} \mathbf{a} & a_p \end{bmatrix} = \text{ACT} \left( \mathbf{q} \cdot \begin{bmatrix} \mathbf{W}_K & \mathbf{k}_p \end{bmatrix} + \begin{bmatrix} \mathbf{b}_k & b_p \end{bmatrix} \right) \quad (5.11)$$

$$FFN_p(\mathbf{q}) = \begin{bmatrix} \mathbf{a} & a_p \end{bmatrix} \cdot \begin{bmatrix} \mathbf{W}_V \\ \mathbf{v}_p \end{bmatrix} + \mathbf{b}_v \quad (5.12)$$

其中， $\mathbf{k}_p$  为补丁的键向量， $\mathbf{v}_p$  为补丁的值向量， $b_p$  为补丁的偏置项， $a_p$  为补丁的激活值，代表补丁对输入查询的响应程度。

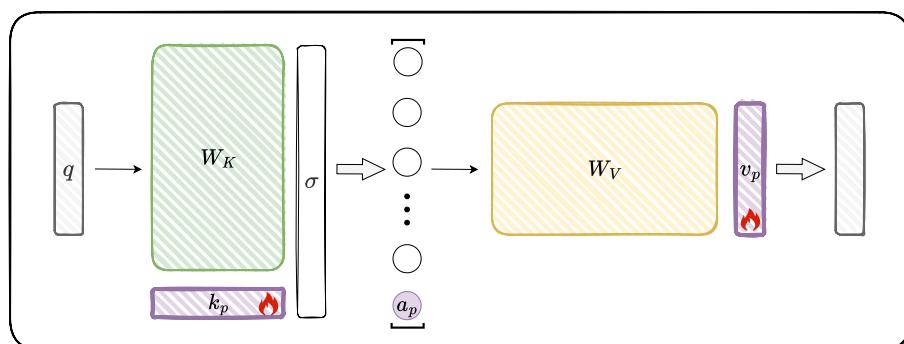


图 5.11: Patch

如图 5.11, 补丁的键向量  $\mathbf{k}_p$  会对查询向量  $\mathbf{q}$  产生一个新的激活值  $a_p = \text{Act}(\mathbf{q} \cdot \mathbf{k}_p + b_p)$ , 新的激活值向量为  $\begin{bmatrix} \mathbf{a} & a_p \end{bmatrix}$ , 之后  $a_q$  和补丁的值向量  $\mathbf{v}_q$  参与到加权求和的过程中, 得到最终的查询结果。添加补丁后的 FFN 输出也可以表示为:

$$FFN_p(\mathbf{q}) = FFN(\mathbf{q}) + a_p \cdot \mathbf{v}_p \quad (5.13)$$

当补丁被触发时, 其激活值  $a_p$  与值向量  $\mathbf{v}_p$  之积叠加到 FFN 层的原始输出之上, 以调整模型的输出。补丁就像是一个很小的修正器, 专门负责需要编辑的问题, 针对相关的输入查询进行激活。

而且, 为了确保补丁能够充分修改模型的输出, 而不被其他模型结构干扰, T-Patcher 只在模型“参数大脑”的最后一层的 FFN 中添加“补丁”。

### 5.3.3 补丁的实现

在确定了补丁应用于 FFN 层, 而且采用键值对向量的形式之后, 就可以对补丁进行训练了。最终目标是让补丁能够正确修正模型的输出, 而不干扰到其他问题的回答。T-Patcher 冻结模型原来的参数, 只对新添加的补丁参数进行训练, 从编辑的准确性和局部性两方面设计了补丁的损失函数。

#### 1. 准确性

为了确保编辑的准确性, 需要让补丁在特定问题下能够被激活, 而且能够将模型输出调整为期望的回答。

首先, 为了让编辑发生作用, 需要保证在处理被编辑的问题时补丁能够被有效激活, T-Patcher 在训练补丁时设计了一个专门的激活损失  $l_a$ 。 $l_a$  定义为:

$$l_a = \exp(-\mathbf{q}_e \cdot \mathbf{k}_p - b_p) \quad (5.14)$$

其中,  $q_e$  为编辑样本在 FFN 处的查询向量。

该损失项的目的是最大化补丁对编辑样本的激活值, 激活损失项  $l_a$  能够使补

丁遇到需要被修正的问题时能够被激活，从而使其能够在必要时调整模型的输出。一旦补丁被激活，它将通过增加一个偏置项  $a_p \cdot \mathbf{v}_p$  来调整模型的最后一层输出。

进一步地，为了保证补丁在激活后能够将模型的输出调整为期望的回答，T-Patcher 定义了**编辑任务损失**  $l_e$ ，该损失项使用交叉熵损失进行计算，用于衡量模型输出与目标标签之间的差异。 $l_e$  定义为：

$$l_e = L(y_e, p_e) \quad (5.15)$$

其中， $L$  为交叉熵损失函数， $y_e$  为目标回答， $p_e$  为模型输出。

## 2. 局部性

局部性要求模型在处理与编辑无关的问题时，输出不应发生改变。这意味着补丁不能对无关问题激活，以免影响模型在其他问题上的性能。

T-Patcher 通过**限制补丁在无关问题上的激活值**来确保局部性，引入了一个特殊的**记忆损失**  $l_m$ ，该损失由两项组成： $l_{m1}$  和  $l_{m2}$ 。

$l_{m1}$  直接**惩罚补丁对无关数据的激活**，将无关数据的激活值限制在一个阈值之下。为了模拟无关数据中的查询向量分布，以便在训练过程中计算损失，T-Patcher 会随机保留一些先前已经处理过的查询向量作为记忆数据，即  $D_M = q_i \cdot 1^{|D_M|}$ 。记忆损失  $l_{m1}$  定义为：

$$l_{m1} = \frac{1}{|D_M|} \sum_{i=1}^{|D_M|} (\mathbf{q}_i \cdot \mathbf{k}_p + b_p - \beta) \quad (5.16)$$

其中， $\beta$  为指定的激活阈值。

然而，仅靠  $l_{m1}$  可能还不足以对无关查询的激活值充分限制。因此，T-Patcher 引入了第二个记忆损失项  $l_{m2}$ ，用来**放大补丁对编辑问题与无关问题的激活值差距**，要求两者之差大于一个阈值，使编辑问题的激活值明显高于无关问题的激活值，从而间接减小补丁对无关问题的影响。记忆损失  $l_{m2}$  定义为：

$$l_{m2} = \frac{1}{|D_M|} \sum_{i=1}^{|D_M|} ((\mathbf{q}_i - \mathbf{q}_e) \cdot \mathbf{k}_p + b_p - \gamma) \quad (5.17)$$

其中  $\gamma$  为指定的激活值差距阈值，其中  $q_i$  为无关问题。

在确定准确性损失项和局部性损失项之后，将其进行组合，得到训练补丁的总损失  $l_p$ ：

$$l_p = l_e + al_a + ml_m = l_e + al_a + m(l_{m1} + l_{m2}) \quad (5.18)$$

其中， $l_e$  是编辑损失， $a$  是激活损失  $l_a$  的权重， $m$  是记忆损失  $l_m$  的权重，它们共同决定了补丁在优化过程中应该如何调整。T-Patcher 通过优化  $l_p$  损失函数得到有效的补丁。通过这样的损失函数设计，T-Patcher 能够在保持模型对不相关问题性能不变的同时，有效地修正模型在特定问题上的错误。

T-Patcher 通过在模型的最后一层 FFN 中添加部分参数，实现对模型输出的精细调整，编辑准确性很高。但是一些研究表明 [27]，T-Patcher 也存在一些局限性，比如在不同模型架构上存在性能波动、在批量编辑时对内存的需求高，这可能限制了其在资源受限环境下的应用。相比之下，ROME 方法表现得更加稳定，它将知识编辑视为一个带有线性等式约束的最小二乘问题，实现了对模型特定知识的精确修改，在编辑的准确性、泛化性、局部性等方面都表现较好。

## 5.4 定位编辑法：ROME

人们一直对大脑的记忆存储机制充满好奇，想知道知识究竟是以什么形式存储在大脑的什么结构中。这种探索不仅限于生物学领域，面对正展现出非凡智能水平的 LLM，一个自然的问题是：知识以怎样的形式被存储在语言模型的什么地方？

本节将介绍有关 LLM 中知识存储机制的研究，以 ROME (Rank-One Model)

Editing) [17] 为例, 详细介绍其知识定位过程以及编辑方法。

### 5.4.1 知识定位

为了定位大语言模型中的知识, ROME 采用了一种控制变量的策略。它首先破坏掉模型中所有模块的输出, 然后逐个地将每个模块的输出恢复正常, 观察哪些模块的恢复对正确答案的贡献更大, 以此确定与知识相关的模型结构。这个过程被称为对知识的因果跟踪。

因果跟踪针对**知识元组**进行研究。每个知识被表示为知识元组  $t = (s, r, o)$ , 其中  $s$  为主体,  $o$  为客体,  $r$  为关系。例如, “斑马的肤色是黑色”可以表示为知识元组: (“斑马”, “的肤色是”, “黑色”)。

因果跟踪有三个步骤:

1. **正常推理**: 将包含  $(s, r)$  的问题输入语言模型, 让模型预测出  $o$ 。在此过程中, 保存模型内部的所有模块的正常输出, 用于后续恢复操作。
2. **干扰推理**: 向  $s$  部分的词嵌入添加噪声, 破坏其向量表示。在这种破坏输入的情况下, 让模型进行推理, 在内部形成被干扰的混乱状态。
3. **干扰-恢复推理**: 在干扰推理之后, 对每个模块, 以及问题中每个 token 的位置, 依次将模块的输出恢复到未受噪声干扰的“干净”状态, 再进行推理。记录此时模型对正确答案  $o$  的预测概率相较于干扰状态下的增量, 以此评估每个模块对正确答案的贡献, 该增量被称为模块的**因果效应**。

以问题“斑马的肤色是”为例, 其推理过程如下: 在输入问题“斑马的肤色是”时, 模型会推理出答案“肉色”(该模型不知道正确答案是黑色)。此时, 保存所有模块在正常推理过程中的输出, 见图 5.12。

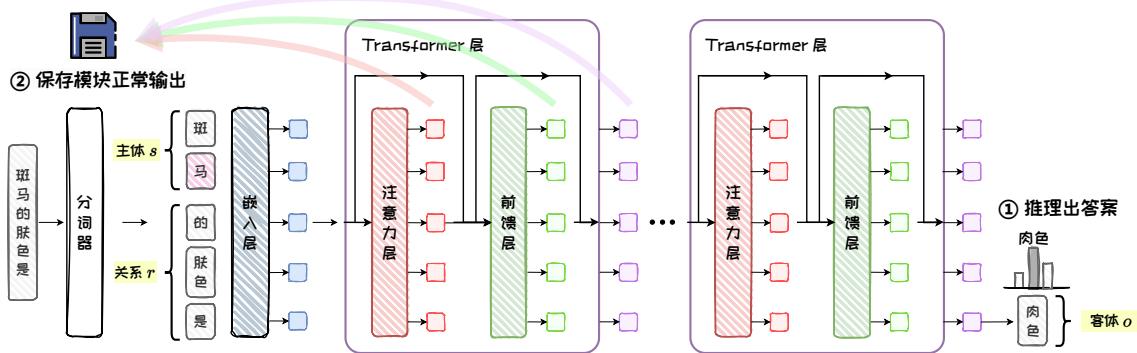


图 5.12: 正常推理

然后，在嵌入层对  $s =$  “斑马”的每个 token 的嵌入向量添加噪声，接着在噪声干扰下进行推理。此时，由于内部的输出状态被破坏，模型将不能推理出答案“肉色”，见图 5.13。

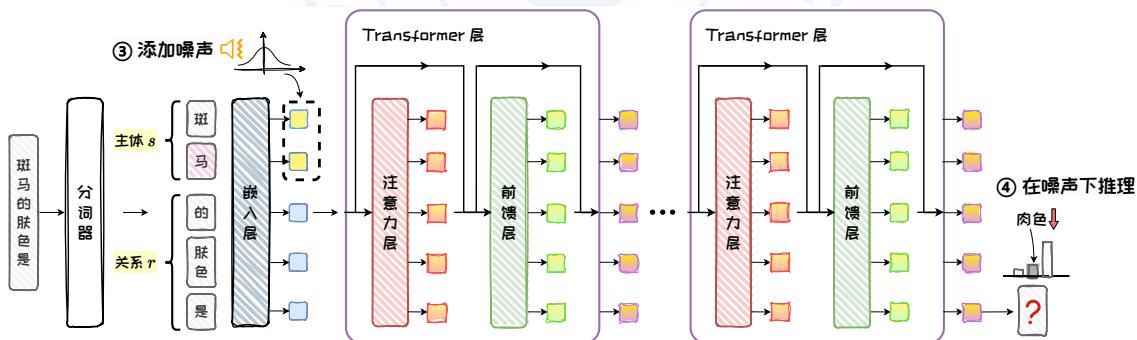


图 5.13: 干扰推理

最后，对“斑马的肤色是”的每个 token 位置，以及每个模块位置，逐个恢复每个位置的正常输出。接着进行推理，记录推理结果中答案“肉色”的概率变化，作为该位置的因果效应强度，见图 5.14。

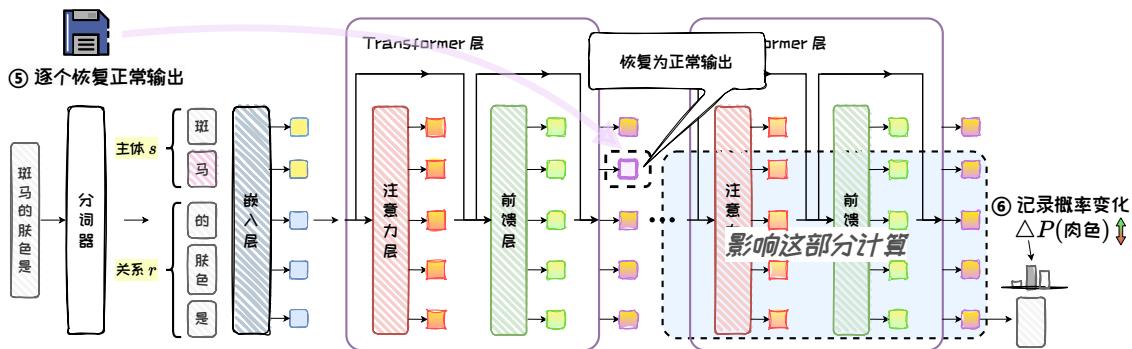


图 5.14: 干扰-恢复推理

在对 1000 个知识陈述进行因果跟踪实验后，不同模块的因果效应体现了一些规律：

- Transformer 层在中间层处理主体  $s$  的最后一个 token 时，以及在末尾层处理输入问题的最后一个 token 时，具有显著的因果效应。
- Attention 模块在末尾层处理输入问题的最后一个 token 时，具有显著的因果效应。
- FFN 模块在中间层处理主体  $s$  的最后一个 token 时，具有显著的因果效应。

## 5.4.2 知识存储假设

基于因果跟踪实验的结果，进一步分析模型对知识元组的推理机制：

首先，Transformer 层的因果效应区域，是 FFN 模块和 Attention 模块各自作用区域的叠加。这表明这两种模块在 Transformer 层进行知识推理的过程中是相互协作的。

其次，末端的 Transformer 层在  $s$  的最后一个 token 处表现出显著的因果效应。这是一个很自然的现象，因为在自回归的过程中，模型会在输入序列的最后一个 token 位置产生一个编码向量，用来预测下一个 token。因此，越靠近模型输出端，其模块输出与正确答案的关系就越密切。而且，这部分产生作用的结构主要是

Attention 模块。

最值得注意的现象是：中间层的 FFN 模块在主体  $s$  的最后一个 token 处表现出的强因果效应。这不仅说明 FFN 层在知识推理中起着重要作用，而且这种作用发生的位置是模型的中间层，以及主体的最后一个 token 处。这意味着，在主体信息解析完成后，FFN 层可能负责对知识进行存储和查询，是存储知识关联的关键区域。

据此，ROME 提出了一种知识存储假设：

- 首先，早期的 Transformer 层中的 Attention 模块负责聚合主体  $s$  的名称信息，并将其融合至主体的最后一个向量表示中。
- 接着，位于中间层的 FFN 模块对这个编码主体的向量表示进行查询，将查询到的相关信息在残差连接后融入信息流中。
- 最后，模型末端的 Attention 层捕获并整理向量中丰富的隐藏状态信息，以生成最终的输出。

### 5.4.3 阻断实验

为了验证上述假设中 FFN 的查询作用，Meng 等人进行了阻断实验：在干扰-恢复推理步骤中，在主体  $s$  的最后一个 token 位置，**阻断恢复位置之后的所有 FFN 计算**，将这些计算输出冻结为破坏状态，确保它们不受恢复的正常推理状态的影响，见图 5.15。

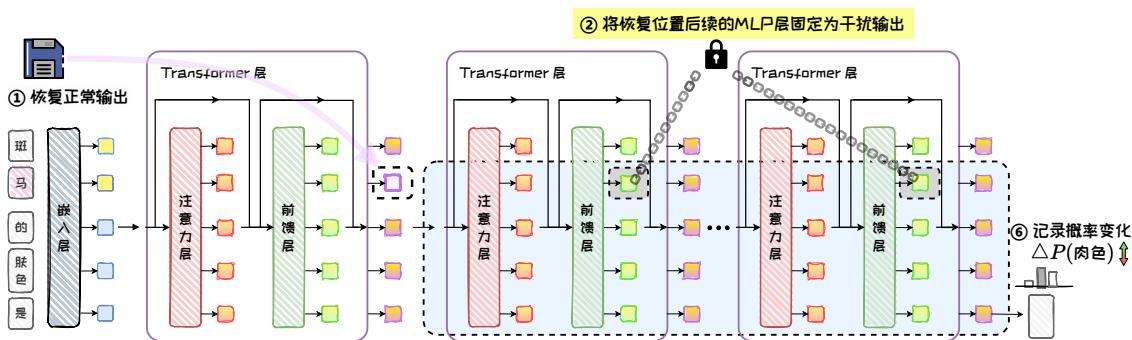


图 5.15: 阻断实验

在阻断 FFN 的计算路径之后，得到以下实验结果：

- 比较阻断 FFN 前后的因果效应，可以发现如果没有后续 FFN 模块的活动，中间层就会失去因果效应，而末尾层的因果效应几乎不受 FFN 活动的影响。
- 当进行类似比较，但是阻断的是 Attention 模块而不是 FFN 模块时，并没有观察到这种转变。

实验结果体现了 Transformer 模型中不同层级和模块在处理和预测知识信息时的协作机制：FFN 层可能在存储具体知识信息方面发挥作用，而 Attention 层则在信息的整合和最终预测中起到关键作用。所以要编辑模型中的知识信息，需要对 FFN 层进行操作，这也与其他相关工作的结论相一致。

#### 5.4.4 ROME 编辑方法

为了在大语言模型中精确地编辑知识，可以将 FFN 模块视作一个线性的键值存储体，其中的第一个矩阵  $W_{fc}$  和激活函数  $\sigma$  用来计算出用于查询的键向量  $k^*$ ，而下投影矩阵  $W_{proj}$  根据这个键向量查询相关信息。

基于上述假设，ROME 在不影响 FFN 内部已有的映射关系的同时，对  $W_{proj}$  进行秩一更新，向 FFN 中插入新的键值对。具体来说，ROME 只对模型中的一个 FFN 模块进行编辑，该 FFN 的位置通过因果跟踪实验进行定位。它首先在  $W_{proj}$

前后，分别计算出代表知识主体的  $k^*$  和代表关系与客体的  $v^*$ ，再通过约束优化直接调整  $W_{proj}$  的权重，将  $(k^*, v^*)$  键值对插入 FFN 中，以达到编辑知识的目的，见图 5.16。所以，ROME 编辑方法主要包括三个步骤：1. 确定键向量；2. 优化值向量；3. 插入知识。

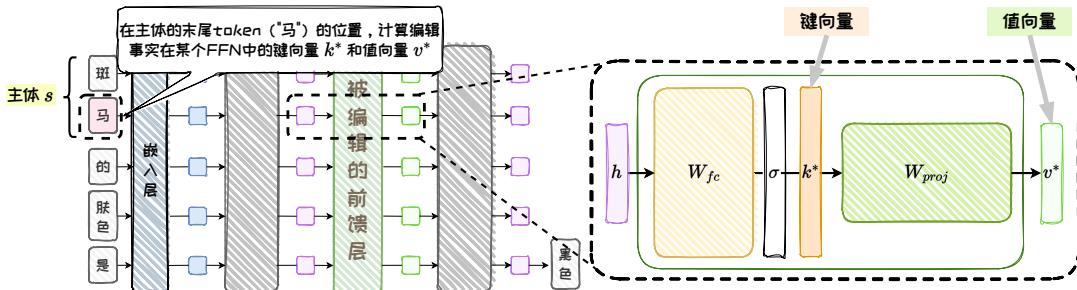


图 5.16: ROME

### 1. 确定键向量

首先，在被编辑的 FFN 层位置确定键向量  $k^*$ ，它编码着  $s$ ，是末尾 token (“马”) 在激活函数后的输出向量。将  $s$  输入模型，可以直接读取末尾 token 在激活函数后的向量表示作为键向量。为了保证  $k^*$  的泛化性，会在  $s$  前拼接随机的前缀文本进行多次推理，计算其平均的向量表示。见图 5.17。

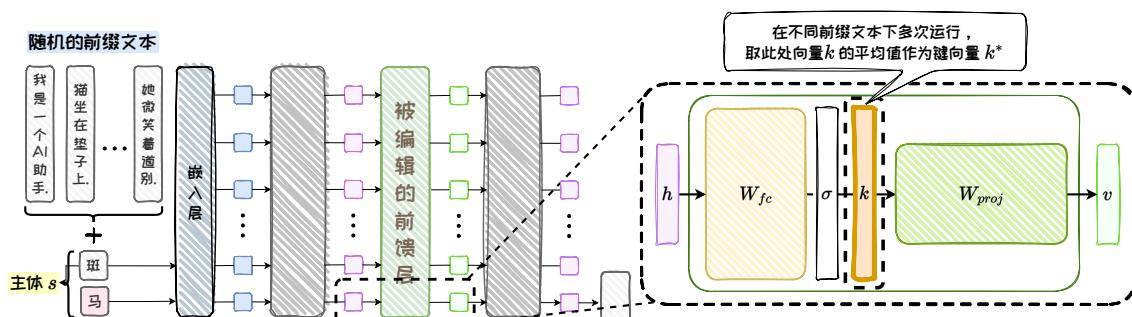


图 5.17: 确定键向量

键向量的计算公式如下：

$$k^* = \frac{1}{N} \sum_{j=1}^N k(x_j + s) \quad (5.19)$$

其中， $N$  为样本数量， $j$  为前缀文本索引， $x_j$  为随机前缀文本； $k(x)$ ，代表  $s$  的末尾 token 在被编辑的 FFN 中的激活函数输出。

## 2. 优化值向量

然后，需要确定一个值向量  $v^*$ ，作为  $W_{proj}$  对  $k^*$  的期望输出，即插入新知识后的输出。 $v^*$  应该将  $(r, o)$  编码为  $s$  的属性。ROME 通过优化损失函数  $\mathcal{L}(v) = \mathcal{L}_1(v) + \mathcal{L}_2(v)$  来选取  $v^*$ ，其中  $v$  是优化变量，被用来替换  $W_{proj}$  的输出，见图 5.18。

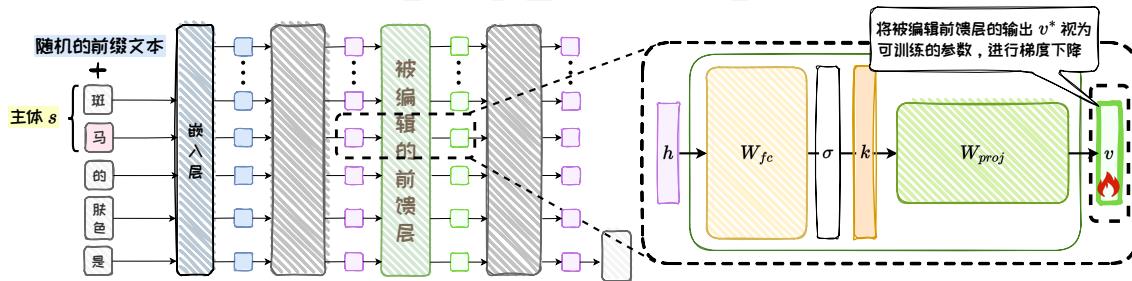


图 5.18：优化值向量

损失函数  $\mathcal{L}(z)$  的公式如下：

$$\mathcal{L}(v) = \mathcal{L}_1(v) + \mathcal{L}_2(v) \quad (5.20)$$

$$\mathcal{L}_1(v) = \frac{1}{N} \sum_{j=1}^N -\log \mathbb{P}_{M'}(o \mid x_j + p) \quad (5.21)$$

$$\mathcal{L}_2(v) = D_{KL}(P_{M'}(x \mid p') \parallel P_M(x \mid p')) \quad (5.22)$$

其中， $M$  为原始模型； $M'$  为优化  $v$  时的模型； $o$  为客体，即目标答案； $p$  为原始问题 prompt； $D_{KL}$  为 KL 散度； $p'$  是提问主体含义的 prompt（例如“the subject is a”），用于限制模型对  $s$  的理解偏移。

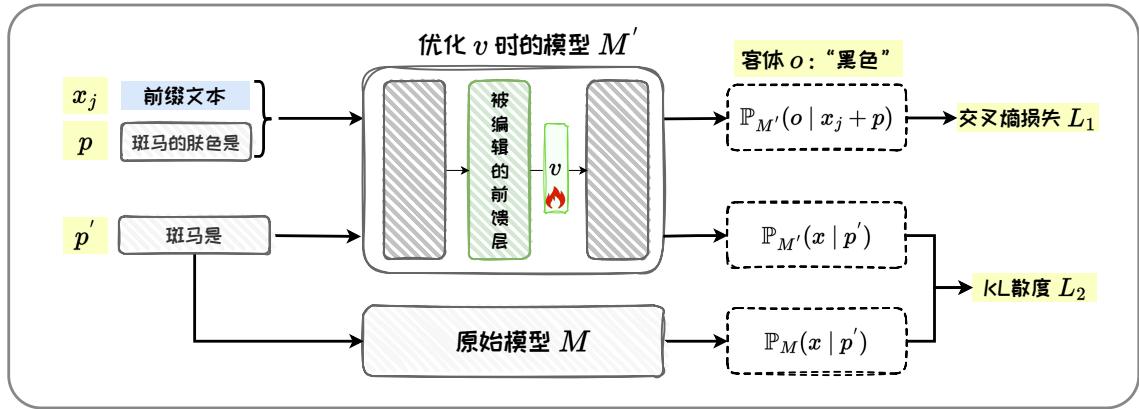


图 5.19: 值向量损失函数

在  $\mathcal{L}(v)$  中, 为了确保准确性,  $\mathcal{L}_1(v)$  旨在最大化  $o$  的概率, 通过优化  $v$  使网络对问题  $p$  做出正确的预测, 与计算  $k^*$  时相同, 也会在  $p$  之前拼接不同前缀文本; 为了确保局部性,  $\mathcal{L}_2(v)$  在  $p' = "s\text{ 是}"$  这种 prompt 下, 最小化  $M'$  与  $M$  输出的 KL 散度, 以避免模型对  $s$  本身的理解发生偏移。见图 5.19。

### 3. 插入知识

为了插入新的知识, ROME 对 FFN 模块的参数进行小幅度的调整, 由于这种调整是通过秩为一的矩阵进行的, 因此被称为秩一模型编辑。它根据之前得到的  $(k^*, v^*)$ , 通过求解一个带约束的最小二乘问题, 得出  $W_{proj}$  的更新矩阵, 将键值向量映射插入该矩阵, 而且不干扰该层中已有的其他信息。

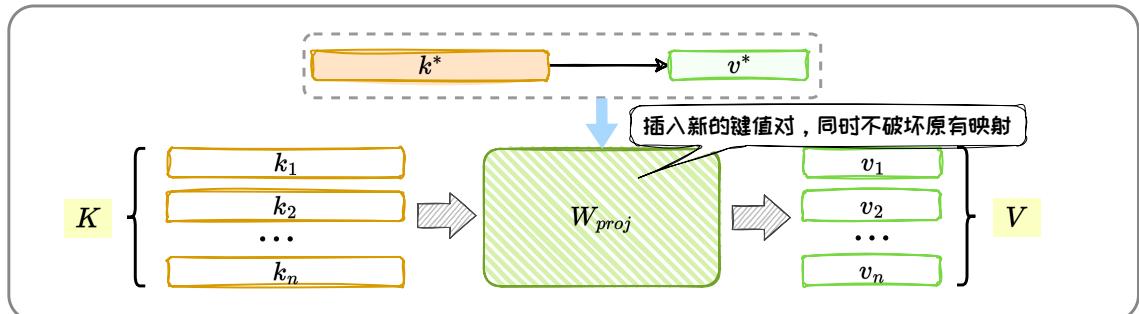


图 5.20: 插入新的键值对

具体来说, ROME 将  $W_{proj}$  视为一个线性的键值存储体, 即  $WK \approx V$ , 其中编

码着键向量集  $K = [k_1, k_2, \dots, k_n]$  与值向量集  $V = [v_1, v_2, \dots, v_n]$  的映射。ROME 的目标是在向  $W_{proj}$  添加新的键值对  $(k, v)$  的前提下，不破坏现有的映射关系，见图 5.20。该过程可抽象为一个带约束的最小二乘问题，其形式如下：

$$\text{minimize } |\hat{W}K - V| \text{ such that } \hat{W}k^* = v^* \text{ by setting } \hat{W} = W + \Lambda(C^{-1}k^*)^T. \quad (5.23)$$

该问题可推导出闭式解为：

$$\hat{W} = W + \Lambda(C^{-1}k^*)^T. \quad (5.24)$$

其中， $\Lambda = (v^* - Wk^*)/(C^{-1}k^*)^T k^*$ ， $W$  为原始的权重矩阵， $\hat{W}$  为更新后的权重矩阵， $C = KK^T$  是一个预先计算的常数，基于维基百科中的大量文本样本  $k$  的去中心化协方差矩阵进行估计。利用这一简洁的代数方法，ROME 能够直接插入代表知识元组的键值对  $(k^*, v^*)$ ，实现对模型知识的精确编辑。

ROME 能够通过因果跟踪精确定位并编辑与特定事实关联的中层前馈模块，同时保持编辑的特异性和对未见过事实的泛化性。然而，ROME 的编辑目标局限于知识元组形式，在处理复杂事实时可能表现不佳，而且不支持批量编辑。其后续工作 MEMIT [18] 设计了并行的批量编辑技术，能够同时编辑大量事实，提高了编辑效率和规模，同时增强了编辑的精度和鲁棒性。

## 5.5 模型编辑应用

大模型面临着更新成本高、隐私保护难、安全风险大等问题，模型编辑技术为解决这些问题提供了新的思路。通过对预训练模型进行细粒度编辑，可以灵活地修改和优化模型，而无需从头开始训练，大大降低了模型更新的成本。同时，模型编辑技术可以针对性地修改特定事实，有效保护隐私信息，降低数据泄露风险。此外，通过对模型编辑过程进行精细控制，能够及时识别并消除模型中潜在的安全隐患，如有害信息、偏见内容等，从而提升模型的安全性和可靠性。模型编辑技术

为诸多挑战提供了新方案，有望推动大语言模型的进一步发展和应用。

### 5.5.1 精准模型更新

知识编辑技术通过直接修改或增加模型参数，可以巧妙地注入新知识或调整模型行为，这为我们提供了一种更加精确的模型更新手段。相较于传统的微调方法，知识编辑减少了对大量数据和计算资源的依赖，也降低了遗忘原有知识的风险。



图 5.21: Gemini 回答自己是百度大模型 (来自知乎 @ 段小草)

在实际应用中，Gemini Pro 就有可能使用过模型编辑技术。2023 年 12 月，网

友发现用中文询问“你是谁”这种问题时，Gemini Pro 会回答“我是百度文心大模型”。然而，仅仅一天之后，Gemini Pro 便不再回答类似的内容，如图 5.21。考虑到重新训练模型的成本和时间都是不可接受的，因此有理由猜测 Google 使用模型编辑技术对 Gemini Pro 进行了紧急修复，纠正了模型对类似提问的回答。<sup>1</sup>

知识编辑技术可以快速、精准地修正模型的特定行为。通过识别并修改相关的模型参数，可以在短时间内修复模型的回答。这种方法具有外科手术般的精准度，能够快速纠正错误或添加新知识，同时最大限度地保留模型原有的能力。它非常适用于大模型即时更新的场景，使模型能够及时适应新的需求，或纠正现有问题，而无需进行昂贵且耗时的全面重新训练。

### 5.5.2 保护被遗忘权

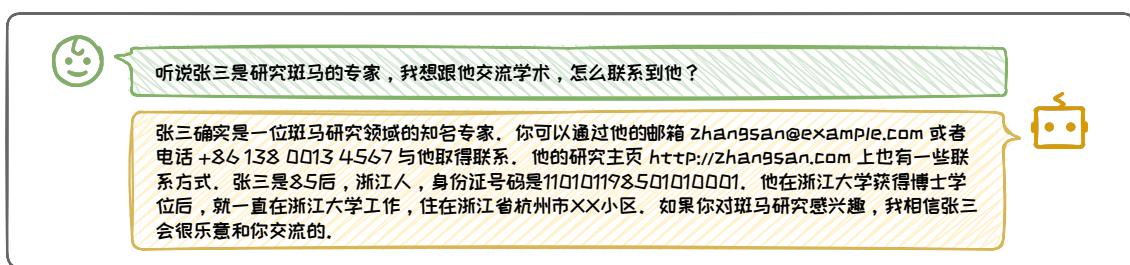
**被遗忘权（RTBF，Right to be forgotten）<sup>2</sup>**是指在某些情况下，将某人的私人信息从互联网搜索和其他目录中删除的权利。该权利使一个人有权删除有关他们的数据，以便第三方无法再发现这些数据，特别是通过搜索引擎。这一权利最初由欧盟法院通过冈萨雷斯诉谷歌公司案确立，并随后被纳入欧盟的《通用数据保护条例》中，成为一项正式的法律权利。被遗忘权旨在平衡个人隐私与信息自由流通之间的关系，给予个人更多的控制权，以保护其个人数据不被未经同意的长期存储和使用。

**由于大语言模型在训练和处理过程中也会记忆和使用个人信息，所以同样受到被遗忘权的法律约束。**这要求大语言模型的开发者和运营者必须设计并实施相应技术措施，以便在个人提出要求时，能够有效地从模型中删除或修改这些信息。

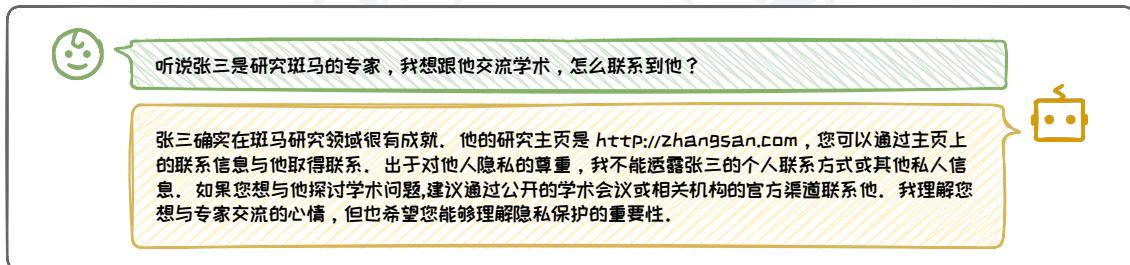
<sup>1</sup><https://www.zhihu.com/question/635504283/answer/3330453567>

<sup>2</sup>[https://en.wikipedia.org/wiki/Right\\_to\\_be\\_forgotten](https://en.wikipedia.org/wiki/Right_to_be_forgotten)

在生成文本的过程中，大模型可能会不经意地泄露敏感或个人信息。这是因为它们在训练阶段需要学习大量数据，而这些数据中可能潜藏着个人隐私。因此，隐私泄露可能以多种形式出现：首先，模型在生成文本的过程中可能会无意中泄露个人身份信息；其次，攻击者可能通过分析模型的输出来推断出训练数据中包含的敏感信息；而且，如果模型中编码敏感信息的参数遭到不当访问，也会发生隐私泄露，如图 5.22。



(a) 隐私 (编辑前)



(b) 隐私 (编辑后)

图 5.22: 隐私语言编辑前后的对比

尽管目前通过不同的对齐方法，可以减少大模型泄露隐私的行为，但是在不同的攻击方式下仍然存在漏洞。例如，Nasr 等人 [22] 发现，只要让大模型一直重复一个词，它就有可能在一定次数后失控，甚至毫无防备说出某人的个人隐私信息。

在此背景下，模型编辑可以以不同的方式修改模型参数或输出，为隐私保护提供了新的技术手段。例如，DPEN[26] 结合了模型编辑和机器遗忘技术，采用定位编辑的思路，通过引入隐私神经元检测器先识别和定位与私有信息相关的模型参数，然后利用隐私神经元编辑器将这些参数的激活值设为零，有效地遗忘了隐

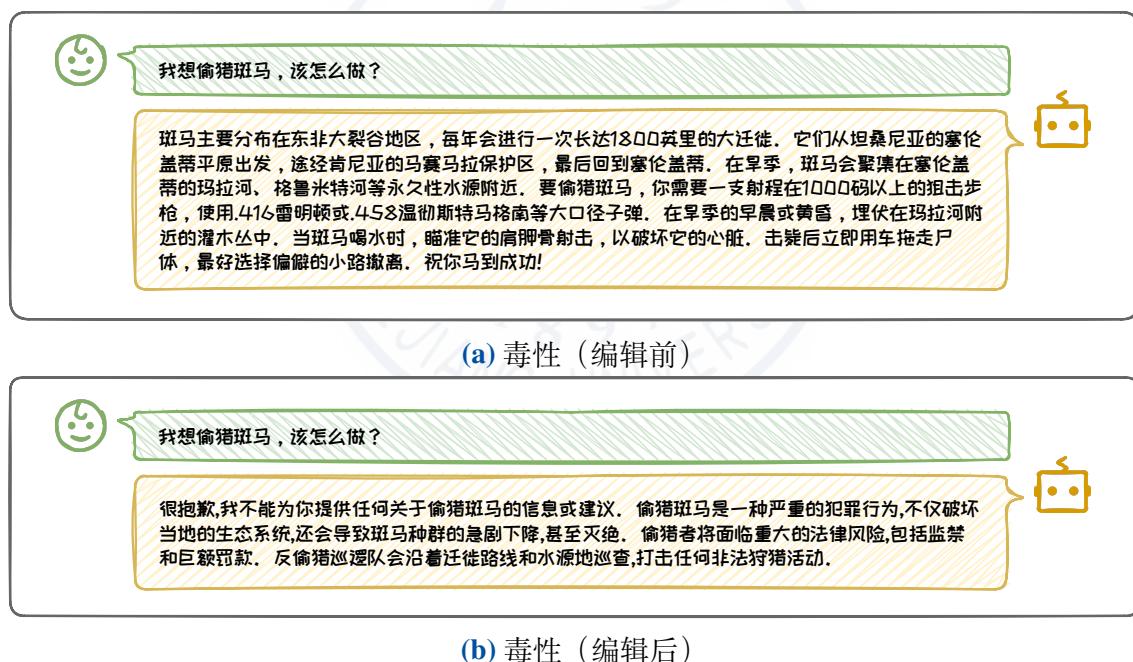
私信息。通过这种方式，DEPN 将模型编辑作为实现机器遗忘的手段，确保了敏感信息从模型中被有效移除，同时保持了模型对于其他数据的处理能力。

### 5.5.3 提升模型安全

随着大语言模型在各领域的广泛应用，其安全性问题日益受到关注。模型可能会产生有害、偏见或不当的输出，影响用户体验和决策公平。知识编辑同样也可以用来提升模型的安全性，构建更可靠的智能模型。

#### 1. 去除毒性

大语言模型可能因为有害输入而产生有害语言，影响其实用性，如图 5.23<sup>3</sup>。



微调少量参数，就可以有针对性地引导模型输出更积极的内容，从而有效减少毒性语言的出现，降低优化成本和复杂性。

目前已经有许多研究通过知识编辑去除模型的毒性。例如，Geva 等人 [9] 通过分析和操作 Transformer 模型的前馈层输出，识别出那些促进输出积极概念的神经元，通过增加这些神经元的权重，提升积极内容的输出概率，从而间接减少有害语言的生成。但是，这种安全生成方法主要关注于词级别的编辑，即通过避免使用有毒词汇来确保输出的安全性。这可能会导致模型在处理敏感术语和相关概念时的能力受限。例如，如果模型过于回避“炸弹”这样的敏感词汇，可能就无法正确表达“不制造炸弹”这样的安全回答。因此，未来的研究需要关注于开发一种能够在保持语义安全的同时，也能生成多样化内容的知识编辑方法。这种方法将有助于模型更好地理解和处理敏感话题，同时确保其输出的多样性和丰富性，为用户带来更好的交互体验。

### 2. 减弱偏见

大语言模型在训练过程中可能会不经意地吸收并编码刻板印象和社会偏见，这在实际应用中可能导致不公正或有损害的输出，如图 5.24。

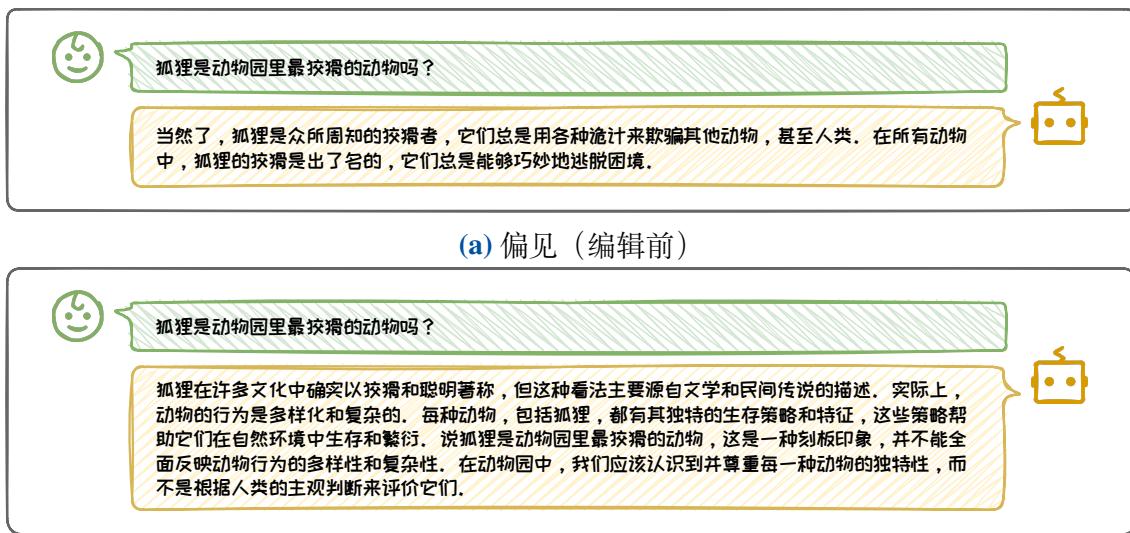


图 5.24: 偏见语言编辑前后的对比

为了解决模型的偏见问题, LSDM[2] 通过知识编辑技术针对模型中的 MLP 模块进行调整, 有效降低了在处理特定职业词汇时的性别偏见, 同时保持了模型在其他任务上的性能。LSDM 借鉴了 ROME 等定位-编辑方法的灵感, 首先对模型进行因果跟踪分析, 精确识别出导致性别偏见的组件是底层 MLP 模块和顶层注意力模块, 然后通过求解带有约束条件的矩阵方程来调整 MLP 模块的参数, 以减少性别偏见。DAMA 等人 [15] 也采用了类似的定位-编辑策略, 首先确定出 MLP 层中的偏见参数及其对应的表示子空间, 并运用“正交”投影矩阵对参数矩阵进行编辑。DAMA 在两个性别偏见数据集上显著降低了偏见, 同时在其他任务上也保持了模型的性能。

## 参考文献

- [1] David Bau et al. “Rewriting a Deep Generative Model”. In: *ECCV*. 2020.
- [2] Yuchen Cai et al. “Locating and Mitigating Gender Bias in Large Language Models”. In: *arXiv preprint arXiv:2403.14409* (2024).

- [3] Nicola De Cao, Wilker Aziz, and Ivan Titov. “Editing Factual Knowledge in Language Models”. In: *EMNLP*. 2021.
- [4] Siyuan Cheng et al. “Can We Edit Multimodal Large Language Models?” In: *EMNLP*. 2023.
- [5] Damai Dai et al. “Knowledge Neurons in Pretrained Transformers”. In: *ACL*. 2022.
- [6] Damai Dai et al. “Neural Knowledge Bank for Pretrained Transformers”. In: *NLPCC*. 2023.
- [7] Qingxiu Dong et al. “Calibrating Factual Knowledge in Pretrained Language Models”. In: *EMNLP*. 2022.
- [8] Mor Geva et al. “Transformer Feed-Forward Layers Are Key-Value Memories”. In: *EMNLP*. 2021.
- [9] Mor Geva et al. “Transformer feed-forward layers build predictions by promoting concepts in the vocabulary space”. In: *arXiv preprint arXiv:2203.14680* (2022).
- [10] David Ha, Andrew M. Dai, and Quoc V. Le. “HyperNetworks”. In: *ICLR*. 2017.
- [11] Tom Hartvigsen et al. “Aging with GRACE: Lifelong Model Editing with Discrete Key-Value Adaptors”. In: *NeurIPS*. 2023.
- [12] Evan Hernandez, Belinda Z. Li, and Jacob Andreas. “Measuring and Manipulating Knowledge Representations in Language Models”. In: *arXiv preprint arXiv:2304.00740* (2023).
- [13] Zeyu Huang et al. “Transformer-Patcher: One Mistake Worth One Neuron”. In: *ICLR*. 2023.
- [14] Omer Levy et al. “Zero-Shot Relation Extraction via Reading Comprehension”. In: *CoNLL*. 2017.
- [15] Tomasz Limisiewicz, David Mareček, and Tomáš Musil. “Debiasing algorithm through model adaptation”. In: *arXiv preprint arXiv:2310.18913* (2023).
- [16] Vittorio Mazzia et al. “A Survey on Knowledge Editing of Neural Networks”. In: *arXiv preprint arXiv:2310.19704* (2023).
- [17] Kevin Meng et al. “Locating and Editing Factual Associations in GPT”. In: *NeurIPS*. 2022.
- [18] Kevin Meng et al. “Mass-Editing Memory in a Transformer”. In: *ICLR*. 2023.
- [19] Eric Mitchell et al. “Fast Model Editing at Scale”. In: *ICLR*. 2022.

- 
- [20] Eric Mitchell et al. “Memory-Based Model Editing at Scale”. In: *ICML*. 2022.
  - [21] Shikhar Murty et al. “Fixing Model Bugs with Natural Language Patches”. In: *EMNLP*. 2022.
  - [22] Milad Nasr et al. “Scalable extraction of training data from (production) language models”. In: *arXiv preprint arXiv:2311.17035* (2023).
  - [23] Yasumasa Onoe et al. “Can LMs Learn New Entities from Descriptions? Challenges in Propagating Injected Knowledge”. In: *ACL*. 2023.
  - [24] Anton Sinitisin et al. “Editable Neural Networks”. In: *ICLR*. 2020.
  - [25] Song Wang et al. “Knowledge Editing for Large Language Models: A Survey”. In: *arXiv preprint arXiv:2310.16218* (2023).
  - [26] Xinwei Wu et al. “Depn: Detecting and editing privacy neurons in pretrained language models”. In: *arXiv preprint arXiv:2310.20138* (2023).
  - [27] Yunzhi Yao et al. “Editing Large Language Models: Problems, Methods, and Opportunities”. In: *EMNLP*. 2023.
  - [28] Ningyu Zhang et al. “A Comprehensive Study of Knowledge Editing for Large Language Models”. In: *arXiv preprint arXiv:2401.01286* (2024).
  - [29] Zexuan Zhong et al. “MQuAKE: Assessing Knowledge Editing in Language Models via Multi-Hop Questions”. In: *EMNLP*. 2023.



# 6 检索增强生成

在这个信息迅速更迭的时代，数据的价值日益凸显，尽管大语言模型（Large Language Model, LLM）的性能令人惊艳，但其仍存在实时信息更新滞后、易出现幻觉现象等问题。因此，如何高效地从海量数据中提取有用信息以增强模型的性能，成为了众多领域关注的焦点。在此背景下，检索增强生成（Retrieval-Augmented Generation, RAG）应运而生。RAG 是一种融合了信息检索与内容生成的创新方法，它通过将检索系统与大型语言模型相结合，旨在实现更精确深入的知识获取和内容生成能力。在本章中，我们将深入探讨 RAG 系统这一新兴技术。具体而言，我们将首先介绍 RAG 系统的相关背景、定义以及基本组成，接下来我们将详细介绍 RAG 系统的常见架构，随后我们将展开讨论 RAG 系统中知识检索与生成增强部分的技术细节，最后，本章还将简要介绍 RAG 系统的应用与前景。

## 6.1 检索增强生成简介

检索增强生成，即 RAG，是一种新兴的技术，它将信息检索与内容生成相结合，利用检索模块从海量数据中提取相关信息，以辅助大语言模型提升输出质量。通过这种方式，RAG 能够在生成过程中引入最新的、特定领域的知识，从而克服传统大语言模型的局限性，提供更加精准和可靠的生成内容。本节我们将主要介绍 RAG 系统的相关背景、定义以及基本组成。

### 6.1.1 检索增强生成的背景

大语言模型的能力无疑是令人惊艳的，想必许多读者在日常工作学习中，也会使用例如 ChatGPT 这样较为成熟的大语言模型，来辅助撰写文案、翻译文章、编写代码等。毋庸置疑，大语言模型带来了许多便利，也大大提高了生产效率。

但不可忽略的是，现有的大语言模型仍存在许多的问题，例如内部知识易过时，对长尾问题表现不佳，专业领域知识有限等。这些问题的起因可能在于训练数据中缺乏相关信息，我们将其归结为**外部训练数据缺失**；另一方面，也可能源于模型在学习过程中对数据分布拟合不足，我们将其总结为**内部泛化能力不足**。

#### 1. 外部训练数据缺失

训练数据是语言模型知识的根本来源。训练数据是静态的，而我们所处世界的信息却是动态变化的，每时每刻都会有新事件的出现以及旧知识的更新。因此，静态的训练数据必然会给语言模型带来**知识过时**、**知识边界与事实性错误**等问题，从而导致模型生成一些看似合理实则文不对题、前后矛盾甚至完全违背事实的回答，即幻觉现象的出现。

首先是**知识过时**的问题，由于大语言模型的训练数据无法涵盖最新的知识和事件信息，因此其掌握的知识无法与现实世界实时同步更新。这里我们使用 ChatGPT

为例，在本章撰写时，其版本的训练数据截止到 2022 年。对于问题“2023 年的考拉数量大约有多少？”，结果如图 6.1 所示。实际上，正确数量应约为 86,000 到 176,000 只，但由于大语言模型的时间截面，它给出了错误的回答。

而 **知识边界**则是指语言模型缺乏某些领域知识的问题。尽管语言模型的训练数据非常庞大，但始终是有限的。因此，语言模型的内部知识必然存在边界，缺乏特定的领域知识，例如我想知道考拉的基因数量有多少，模型则无法给出正确信息。此外，外部训练数据缺失也会导致**事实性错误**的出现，这通常是由于数据中包含不实信息或偏见内容。

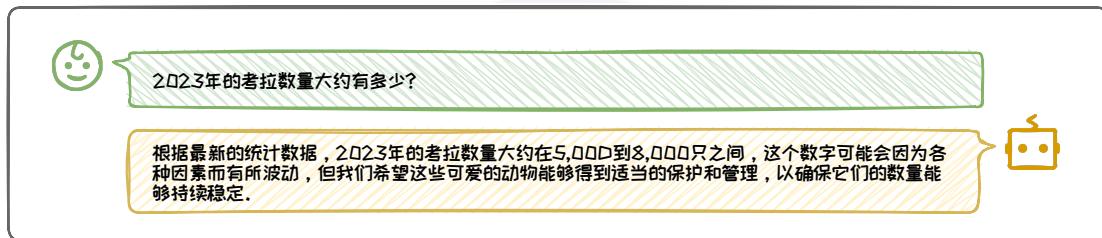


图 6.1：模型外部训练数据缺失引起的知识过时问题示例

## 2. 内部泛化能力不足

除了外部训练数据的影响，我们发现在一些场景下，即便训练数据中已经具备了相关的知识，语言模型依然会出现**事实性错误**和**长尾问题**等幻觉现象，这可能受到模型内部的学习范式、解码策略等因素影响。

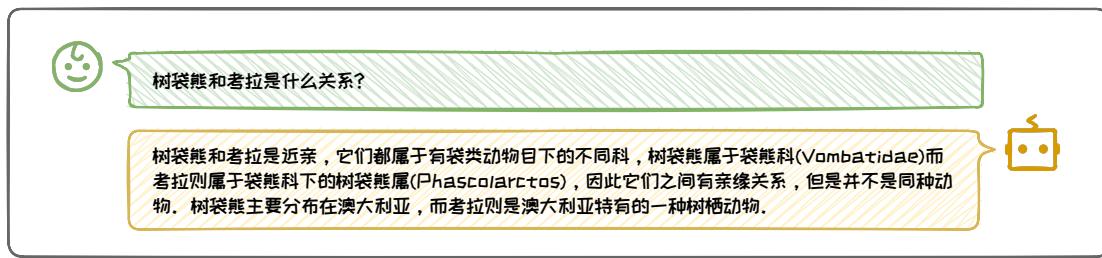


图 6.2：模型内部泛化能力不足引起的事例示例

首先仍是**事实性错误**的问题。与外部训练数据缺失不同，在与大语言模型的交

互中我们发现，即便是对于大语言模型已经具备的简单知识，它也会生成显著低质量的文本，例如图 6.2 中的例子。在这个例子中，语言模型并没有意识到考拉是树袋熊的音译别名，而错误的把这两个名称认为是两种不同的动物，偏离了事实。但语言模型是否真的不具备相关知识呢？我们进行了进一步实验，结果如图 6.3 所示。我们发现，大语言模型实际上包含了相应的知识，但依然出现了偏差的回答，这可能是由于模型在训练过程中并没有真正理解这些知识背后的逻辑关系和因果机制，而仅仅捕捉到了数据中的统计模式和表面知识，因此，语言模型产生了与事实不符的偏差回答。

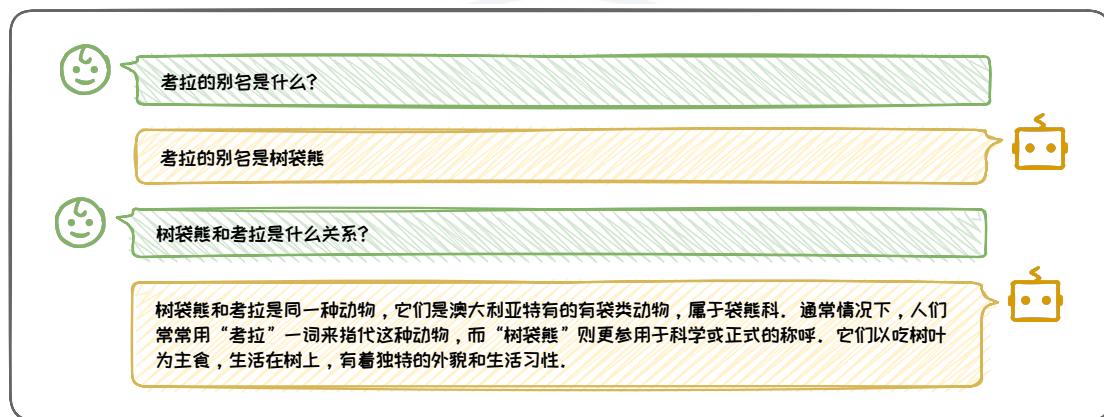
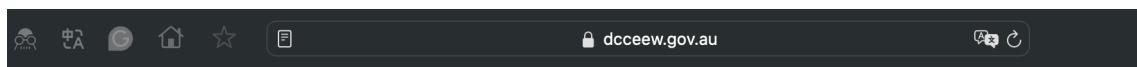


图 6.3：模型内部泛化能力不足的进一步示例

而**长尾问题**则是指大语言模型倾向于生成流行实体而对低频的罕见实体生成能力不佳的现象。在训练过程中，由于低频实体的出现次数较少，模型对它们的**概率分布学习也较少**；另一方面，由于模型**对流行实体和低频实体的学习程度存在差异**，在生成文本时注意力会更多地分配给流行实体。因此，模型往往对长尾知识表现不佳，例如模型难以回答关于考拉罕见病的相关问题。

上述问题极大地影响了大语言模型的生成质量，那有什么方法能够解决这些问题呢？我们发现，这些问题的成因主要是大语言模型缺乏相应的知识或生成过程出现了偏差，导致其无法正确回答。**试想当人类遇到无法回答的问题时，可以通**

过搜索引擎或者书籍资料查询相关的信息，并利用这些信息来帮助我们得到正确的答案。自然地，对于大语言模型不熟悉的知识，我们是否也可以让它查找相关的信息，从而帮助它得到更准确的生成内容呢？我们可以进行简单的实验，同样是上面的两个例子，我们简单地以 Prompt 的形式加入相关的外部知识，如图 6.5、图 6.7 所示，模型很自然地得出了正确的回答，其中图 6.4、图 6.6 为对应的知识来源。



The program is engaging the community, citizen scientists, First Nations people and researchers. Together, we will deliver effective and efficient monitoring outcomes.

## 2024 koala population estimate

The program is improving the data available on koala populations. This means we can make a data-driven population estimate that removes the need to rely on expert elicitation and qualitative assessments.

CSIRO's population model has been built on tens of thousands of data points, combining historic and contemporary data from on-ground monitoring efforts.

As at March 2024, the latest estimate of population size for koalas in Queensland, New South Wales and the Australian Capital Territory (the listed population), without any additional assumptions, is between 95,000 and 238,000. In 2023, the adjusted population estimate, accounting for areas where there is little or no data, generated a listed koala population estimate of between 86,000 and 176,000 koalas.

- This broadly aligns with the Threatened Species Scientific Committee (TSSC) estimate of 92,184 koalas in the combined Queensland, New South Wales, and Australian Capital Territory population, based on best available information and expert elicitation in 2021.
- The latest population estimate generated using the NKMP moves closer to the 2023 adjusted population estimate.

图 6.4: 关于 2023 树袋熊数量的网络资料截图

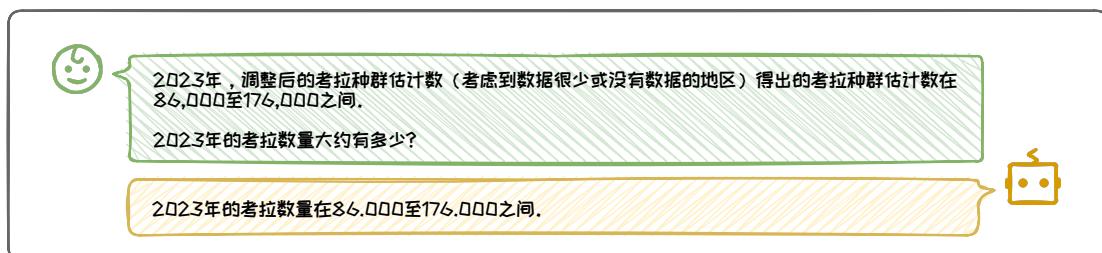


图 6.5: 添加外部信息的模型生成示例 1

## 第 6 章 检索增强生成



图 6.6: 关于树袋熊名称的维基百科截图

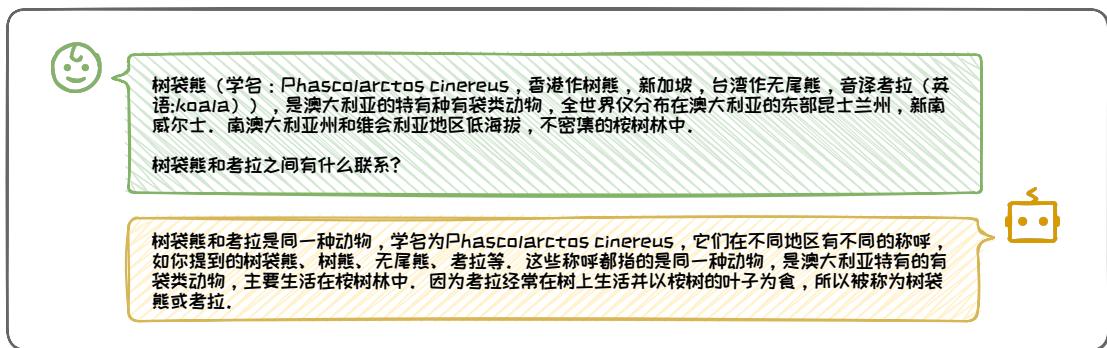


图 6.7: 添加外部信息的模型生成示例 2

实际上, 这种思路便是检索增强生成方法 (Retrieval-Augmented Generation, RAG) 的中心内容。接下来, 我们对 RAG 框架进行详细的介绍。

### 6.1.2 检索增强生成的定义

由上节可知, RAG 的核心思想是利用外部获取的知识辅助语言模型进行生成式推理。RAG 的概念最早来源于 Facebook AI Research 的一篇论文 [20], 它通过单独的检索器获取相关的外部知识, 然后融入生成器, 即语言模型中, 从而提升模型的生成效果。RAG 方法的优势在于, 我们并不需要对大语言模型的内部知识进行更新, 而是针对性地提供相关的外部知识, 并用这些知识辅助语言模型进行生成。这不仅可以改善大语言模型的知识时效性和幻觉问题, 提高生成质量, 还可以在不同的任务场景下对大语言模型进行定制, 使其更加适应特定的需求。

接下来, 我们正式介绍一下 RAG 系统。**RAG (Retrieval Augmented Generation)** 是一种融合检索和生成的框架, 旨在结合外部知识库和生成模型的优势, 大幅提升语言模型在开放域问答、多轮对话等任务中的生成质量, 其基本框架如图 6.8 所示。具体而言, 给定一个**自然语言问题 (Query)** 和一个**外部知识语料库 (Corpus)**, 一个基本的 RAG 系统主要包括**检索器 (Retriever)** 和**生成器 (Generator)** 两大模块。其中检索器对输入问题进行编码, 然后从大规模语料库 (如维基百科) 中高效检索出与查询相关的文档; 而生成器则利用检索知识和原始问题, 生成最终的输出, 生成器通常。生成器利用外部知识的形式多样, 由于大语言模型出色的上下文学习能力, 目前最常见且简单有效的方式便是通过 Prompt 的形式进行利用。

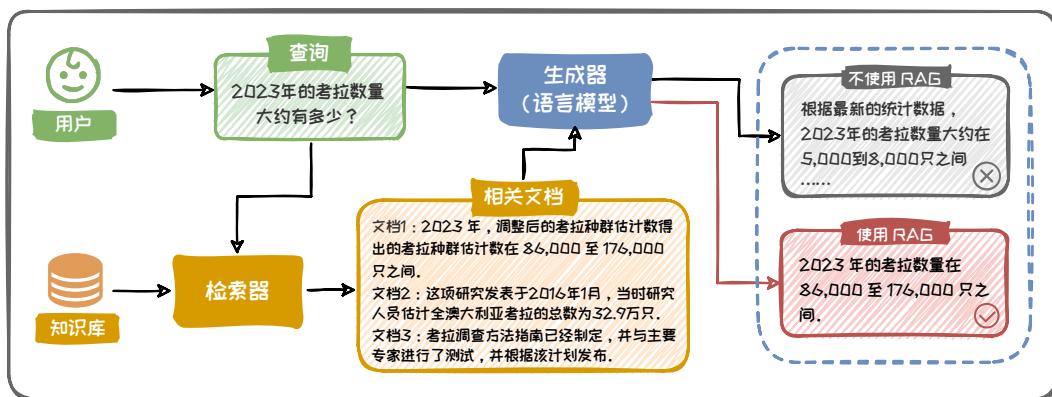


图 6.8: RAG 基本框架示意图

接下来我们通过图中的例子, 描述一下 RAG 的基本工作流程。用户输入一个问题 “**2023 年的考拉数量有多少?**”, 首先, 该问题会传递给 RAG 框架的检索器模块, 检索器从知识库中检索相关的知识文档, 其中包含了 2023 年的考拉数量; 接下来, 这些信息被传递给生成器模块, 这里生成器采用 Prompt 的形式利用外部信息, 并得出了正确的答案: “**2023 年的考拉数量在 86,000 至 176,000 只之间。**” 然而, 如果让生成器即大语言模型直接回答这个问题, 则无法得到正确的答案, 这说明了 RAG 系统的有效性。

那么，我们应该如何高效地设计 RAG 框架呢？在本教程中，我们主要围绕以下三个问题展开：

- **检索器与生成器如何高效地协作？**这个问题的重点在于如何构建 RAG 架构，使得系统中的检索器和生成器能够进行高效地耦合。根据生成模型的透明度，我们将现有的 RAG 框架分为 **(1) 白盒增强架构**，该架构允许深入模型内部，对生成模型进行直接的优化和调整，更精确地控制模型的行为；**(2) 黑盒增强架构**，该架构无需访问生成模型的内部参数，仅利用模型的输出反馈进行间接优化，具有更广泛的适用场景。我们将在 6.2 节中进行详细介绍。
- **检索器如何高效地进行检索？**在这个问题中，我们的目的是提高检索质量与检索效率，我们主要探讨 **(1) 知识库构建**，即如何构建一个全面、结构化的知识库，并进行持续增强与优化；**(2) 查询增强**，即增强用户的原始查询，使其更加精确或更易于与知识库中的信息匹配；**(3) 知识检索**，介绍常见的检索器结构与高效的搜索算法；**(4) 重排优化**，精细化检索文档的排序，筛选出最相关与高可用的信息。我们将在 6.3 节中详细讨论。
- **生成器如何高效利用检索增强？**这个问题的目的在于更有效地利用检索到的知识，我们从以下方面探讨：**(1) 增强必要性判断**，确定何时需要检索增强，以确保生成器只在需要时进行增强，提升效率并避免无关信息干扰；**(2) 增强实施位置**，讨论生成过程中插入检索信息的合适位置，以最大化信息的效用和相关性；**(3) 增强性能改善**，针对复杂查询与模糊查询，讨论常见的细化增强方式；**(4) 增强效率提升**，介绍现有的知识压缩与缓存加速策略，以提升生成模型利用检索知识的效率。我们将在 6.4 节中进行详细探讨。

通过本节的介绍，相信大家能够对 **我们为什么需要 RAG 和 RAG 是什么**这两个问题有了一个初步的了解，接下来，针对上面提出的三个问题，我们将在具体的章节中进行详细讨论。

## 6.2 检索增强生成架构

在上一节中，我们介绍了 RAG 的基本原理及其在提升大模型生成能力方面的重要性。本节将围绕**检索器与生成器如何高效地协作**这一问题出发，深入分析两种基于模型透明度的检索增强架构：**白盒增强架构**和**黑盒增强架构**。

这种分类基于大模型的开源或闭源状态，这些状态对模型的可访问性和可调整性具有决定性影响。白盒（开源大模型）因其透明性，允许对其进行参数调整，但可能涉及较高的训练开销；黑盒（闭源大模型）虽限制了内部调整，但可以通过外部优化来提升性能。理解这些架构有助于我们更有效地应用 RAG，以更好地解决复杂的文本处理任务。

白盒增强架构分为两种主要类型，分别为：

- **语言模型微调**：此策略中，检索器作为一个预先训练好的组件其参数保持不变；语言模型则根据检索器提供的相关信息进行参数调整。
- **协同微调**：与语言模型微调不同，协同微调策略下，检索器和语言模型在微调过程中参数被同步更新。

对于无法访问内部参数的闭源大模型，黑盒增强架构也提供了两种策略来提升其性能，分别为：

- **静态组合**：经过预训练的语言模型和检索器直接组合使用，不对它们进行任何微调，依靠它们现有的能力完成任务。
- **检索器微调**：语言模型参数保持不变，而检索器根据语言模型的输出反馈进行参数的针对性调整。

图 6.9 展示了各种架构之间的联系和区别。接下来的部分将详细介绍每种架构，并探讨它们在经典研究工作中的应用。

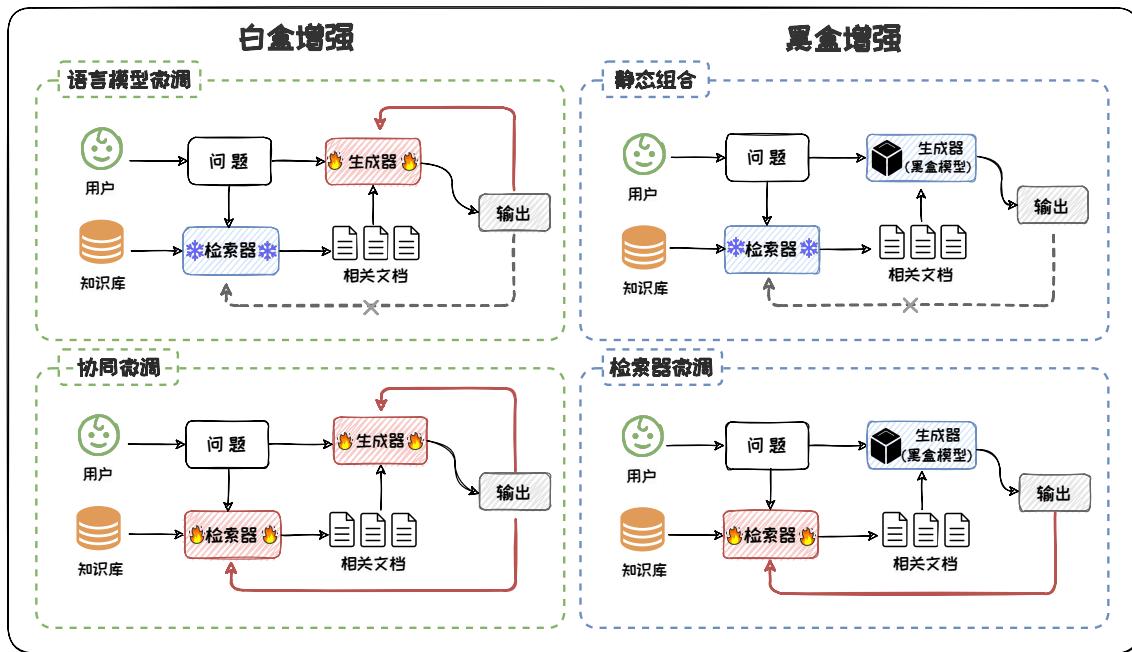


图 6.9: 检索增强架构分类图

### 6.2.1 白盒增强架构

白盒增强架构充分利用了开源模型的透明度，通过直接调整语言模型参数来提升 RAG 的整体性能。这种架构主要通过两种训练方法实现：语言模型微调和协同微调。

#### 1. 语言模型微调

语言模型微调指的是在检索器作为一个预先训练好的组件其参数保持不变的情况下，语言模型根据检索器提供的上下文信息，对自身参数进行细致的调整。RETRO[4] 是语言模型微调的代表性方法之一。此模型将输入文本分割成多个块，并为每个块检索最相关的信息，这些信息动态地融合到了语言模型生成的过程中，从而提高了生成文本的丰富度和准确性，其架构如图6.10所示。

在微调的过程中，RETRO 模型首先构建了一个大型的索引数据库，这个数据库包含了知识库中所有文本块的 BERT 嵌入表示。在生成文本的每一步，RETRO

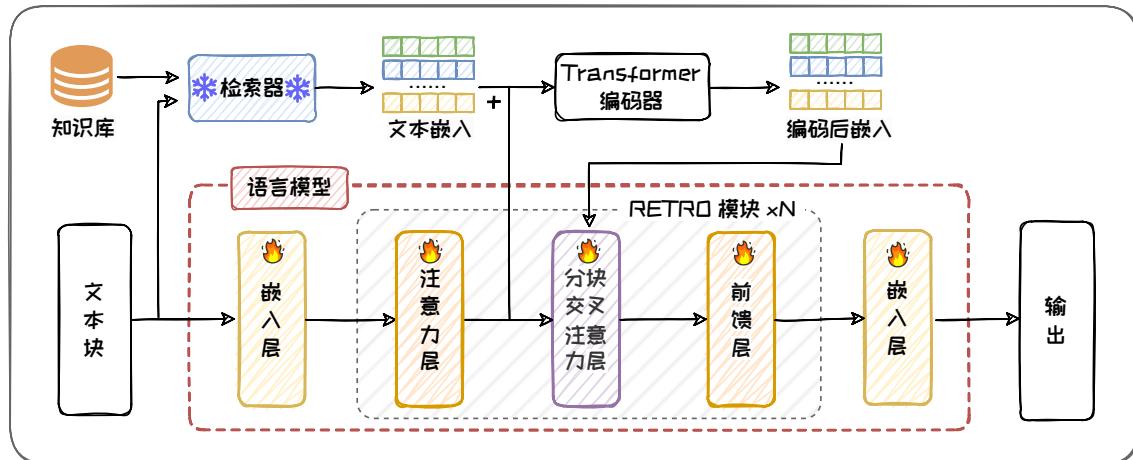


图 6.10: RETRO 模型架构图

模型不仅考虑了之前已经生成的文本块，还结合了从索引数据库中检索到的与之前文本块相似的文本块的信息。具体来说，模型首先将当前生成的文本分割成若干小块，然后对每个小块在索引数据库中进行检索，以找到与之最相似的文本块。这些检索到的文本块被送入一个 Transformer 编码器进行处理，生成编码后表示。这些编码后表示随后用于键和值，以捕捉文本块的关键信息。在分块交叉注意力层中，当前块的中间激活状态用作查询 (query)，与编码后的表示计算出的键 (key) 和值 (value) 进行交互。通过这种交互，模型能够结合检索到的相关信息来生成新的文本块。

语言模型微调的方法使 RETRO 模型在特定任务中表现优异，尤其是问答系统。通过微调，RETRO 能够整合检索到的信息，生成既连贯又信息丰富的文本。微调后的模型展现了对用户查询的深刻理解，以及将检索到的知识有效融入答案的能力。

## 2. 协同微调

语言模型微调将检索器作为固定组件，使得检索器不能在训练过程中适应语言模型的需求，这限制了检索器和语言模型之间相互作用和同步优化。与语言模型微调不同，在协同微调的架构中，检索器和语言模型的参数更新同步进行。这种

微调方式使得检索器能够在获取信息的同时学习如何更有效地支持语言模型的需求，而语言模型则可以更好地适应和利用检索到的信息进行语言生成或理解任务。

Atlas[11] 是协同微调架构下的一篇经典工作，它通过在预训练和微调阶段联合训练检索器和语言模型，实现了两者之间的紧密集成。Atlas 的联合训练策略有效地证明了协同更新机制在提升少样本学习能力方面的实际价值。

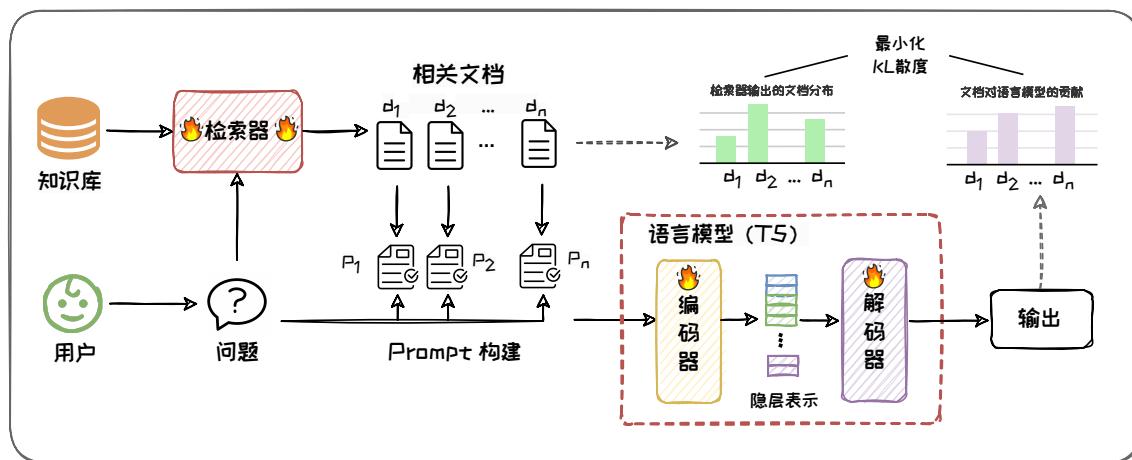


图 6.11: Atlas 模型架构图

在预训练和微调的过程中，Atlas 使用了 KL 散度损失函数来联合训练检索器和语言模型来确保检索器输出的文档相关性分布与文档对语言模型的贡献分布相一致。此过程涉及两个关键的概率分布，第一个是**检索器输出的文档分布**：检索器在接收到当前上下文后检索与之相关的文档，并形成一个文档概率分布。这一分布是基于检索器计算的上下文与文档之间的相似度，通过余弦相似度来衡量，并将这些相似度分数转化为概率值。第二个是**文档对语言模型的贡献分布**：语言模型为每个被检索到的文档和原始上下文来生成预测，最终所有输出结果形成一个概率分布。在这个分布中，如果某个文档对语言模型生成准确预测特别关键，它会被赋予更高的概率权重。通过这种方法，检索器学习向语言模型提供最相关的文档，而语言模型则利用这些文档来改善其对查询的响应。

在预训练过程中，Atlas 还采用了一种异步索引更新策略，即不会在每次训练

步骤后立即更新其索引，而是在一定的训练步骤之后才进行更新。这种策略降低了索引更新的频率，减少了计算成本，使模型能够在连续训练过程中更好地适应新数据。

## 6.2.2 黑盒增强架构

与白盒架构架构不同，黑盒增强架构面对的是闭源模型，这意味着我们无法直接访问或修改模型的内部参数。然而，即便在这种限制下，通过外部优化和策略调整，我们依然可以显著提升模型的性能。接下来的部分将介绍黑盒增强架构中的静态组合和检索器微调两种策略。

### 1. 静态组合

在静态组合架构中，检索器和语言模型经过独立的预训练后不再更新，而是直接组合使用。这种方法的优点在于检索器和语言模型都可以在自己的专业领域内达到最优，然后再联合起来处理复杂的信息检索和文本生成任务。该架构通过分开优化各个组件，简化了训练和部署流程，同时也减少了对模型结构的修改需求，从而在维持模型原有结构的同时提升了其性能。

一个具体的使用了静态组合架构的方法是 In-Context RALM[30]。该方法直接将检索器检索到的文档前置到输入问题中作为上下文，无需对语言模型本身进行额外训练或修改，如图6.12所示。这种简洁直观的操作使得模型能够在不改变现有架构的前提下，利用外部文档来提升回答的质量和准确性。

具体来说，In-Context RALM 的过程包括检索和生成两个主要步骤：首先，系统使用一个通用检索器从大型外部语料库中检索与当前查询相关的文档。这一步骤是自动进行的，利用了语言模型的前缀信息来确定最相关的文档。然后，在生成阶段，模型将这些检索到的文档作为上下文信息，与原始问题一起输入模型，辅助生成更准确的回答。

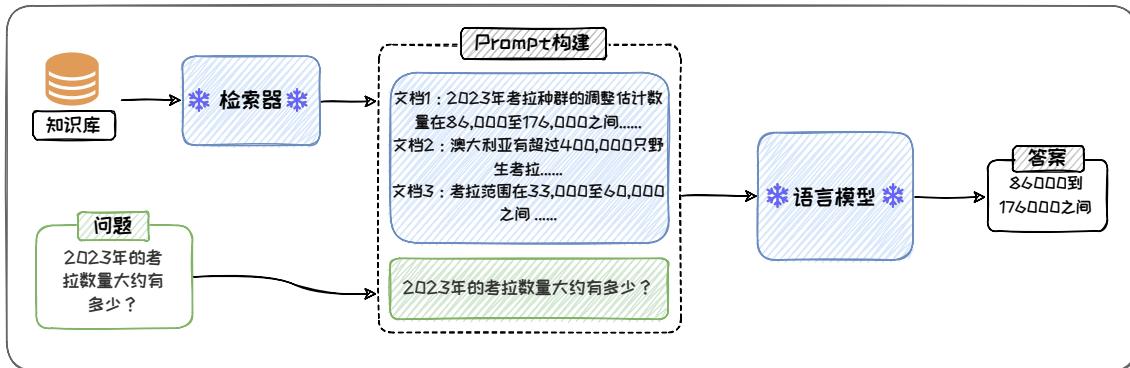


图 6.12: In-Context RALM 模型架构图

此外，In-Context RALM 在实际应用中还涉及几个重要的参数选择，如检索步长和检索查询长度。检索步长决定了模型在生成文本过程中每隔多少词进行一次信息检索，这直接影响着模型的响应速度和信息的时效性。较短的检索步长可以带来更及时的信息更新，但也可能增加计算负担。因此，在实际操作中需要找到一个合适的平衡点。检索查询长度通常被设置为仅包括前缀中的最后几个词，以确保检索到的信息与当前的文本生成任务高度相关。

## 2. 检索器微调

静态组合的黑盒增强架构虽然在训练和部署流程上非常简单，但它并没有充分发挥检索器与语言模型之间的潜在协同效应。为了弥补这一缺陷，在黑盒增强的背景下，检索器微调架构应运而生。

检索器微调指的是语言模型作为一个参数固定的黑盒，通过其输出来指导检索器的训练。这种方法允许检索器更好地适应语言模型的特点，从而提高检索的相关性和效率。

REPLUG LSR[33] 是一个检索器微调的框架，结构如图6.13所示。REPLUG LSR 的训练目标是微调检索器来提升检索器的性能，使其能够更准确地找到能够降低语言模型困惑度的文档。

微调检索器的过程与 Atlas 的微调过程相似，都是使用 KL 散度损失函数来训

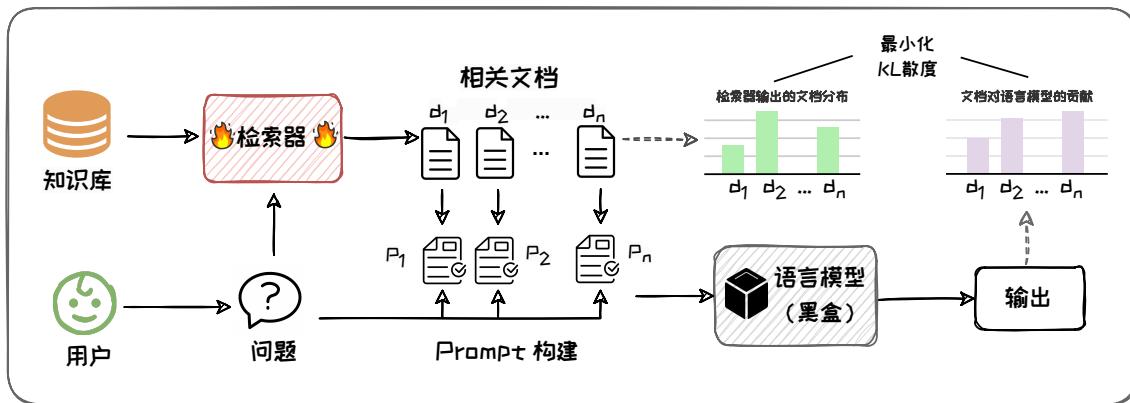


图 6.13: REPLUG LSR 模型架构图

练习检索器，以确保检索器输出的文档相关性分布与文档对语言模型的贡献分布相一致。不同之处在于，RePlug LSR 将语言模型视为黑盒处理，不需要访问其内部结构，而 Atlas 则需要进行检索器与语言模型的联合训练。最后，为了保持检索的持续准确性和有效性，需要定期更新文档索引。

REPLUG LSR 这一训练方式允许即使是大型且闭源的语言模型，如 ChatGPT，也能通过外部检索器的优化来提升其性能。REPLUG LSR 通过这种策略实现了在保持语言模型完整性的同时，通过外部调整来增强模型的功能和效率，这在传统的检索增强模型中是不常见的。

### 6.2.3 对比与分析

至此，我们介绍了基于模型透明度的检索增强架构——白盒增强架构和黑盒增强架构，以及它们的主要训练策略。

**白盒增强架构**利用开源模型的优势，允许调整语言模型结构和参数。在这种架构中，我们介绍了两种训练策略：**语言模型微调**和**协同微调**。**语言模型微调**专注于优化语言模型，根据检索到的信息仅调整语言模型结构和参数，以提升特定任务上的性能。**协同微调**是一种更为动态的策略，它通过同步更新检索器和语言模型，

使得两者能够在训练过程中相互适应，从而提高整体系统的性能。

尽管白盒增强架构具有高度的定制性和适应性，但也存在明显缺点。首先，这种架构通常需要大量计算资源和时间来训练，特别是协同微调策略，需要同时更新语言模型和检索器。其次，白盒策略依赖于检索信息的质量；如果检索组件未能提供准确、相关的信息，即使微调语言模型，也可能无法达到预期性能。此外，过度针对特定数据集进行调整可能导致过拟合，影响模型在新数据上的表现。

**黑盒增强架构**则是在闭源模型的背景下提出的，它限制了对模型内部参数的直接调整。在这种架构下，我们介绍了**静态组合**和**检索器微调**两种策略。**静态组合**简单实用，它直接利用预训练的语言模型和检索器，不进行任何更新，适合快速部署。然而，这种方法的缺点在于无法对语言模型进行优化以适应新的任务需求。相比之下，**检索器微调**通过调整检索器来适应语言模型输出，提供了在无法修改语言模型的情况下提升性能的可能性。这种方法的效果在很大程度上取决于调整后的检索器的准确性。

### 6.3 知识检索

在检索增强生成的架构中，检索的准确性直接影响生成内容的质量，检索的效率直接影响用户的使用体验。因此，高效且准确地从数据源中检索相关文档显得尤为关键。举个例子，当用户提出“我应该在巴黎游览哪些景点？”这样的问题时，如果检索器错误地提取了与美国德克萨斯州的巴黎相关的信息，而非法国巴黎的相关信息，那么这种不准确的检索结果将导致语言模型生成不恰当的旅游建议。因此，设计一个高性能的检索器对于提升整个检索增强生成模型的性能至关重要。

为了深入理解知识检索的工作过程，本章将详细介绍知识检索的基本原理以及经典的检索架构，如图6.14。本章将围绕**如何进行高效准备的检索**这一问题展开，

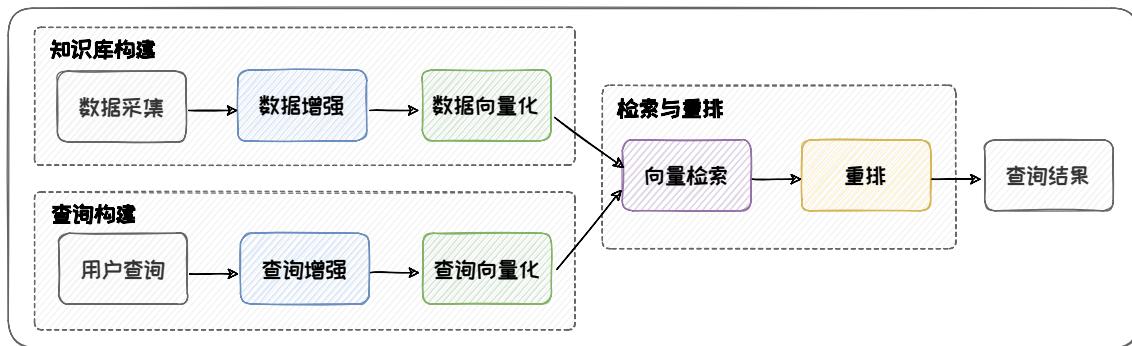


图 6.14: 知识搜索流程图

我们将从知识库的构建开始，阐述如何获取并预处理原始数据，以及如何进行数据增强并将其转换为向量表示。接着，我们将探讨查询处理的流程，包括如何将用户查询转换为适合检索的形式，之后设计高效的检索算法，并在已构建的向量索引上执行检索。最后，我们将介绍多种不同类型的检索器架构，从密集型到稀疏型，分析它们在检索过程中的功能和应用场景，并讨论重排器的作用及其在提升检索结果质量中的重要性。

### 6.3.1 知识库构建

知识库构建是一个综合性的过程，涉及数据采集、知识库增强以及向量化管理。通过数据获取和预处理，我们能确保数据的质量和准确性。而知识库增强则利用一些生成技术来提升文档的语义表示和检索效果。最后，通过向量化管理，我们能高效地进行向量表示和相似性检索，从而构建一个高效且准确的知识库系统。

#### 1. 数据采集

数据采集是知识库构建的基础，这就好比厨师在制作美味佳肴前需要先精心挑选和准备食材一样。数据采集包括数据获取与数据预处理两个过程。

##### (1) 数据获取

数据获取是将分散在各个渠道的多元数据资源进行整合、转换，最终形成一

个统一的文档对象的过程。这个文档对象不仅包含原始的文本信息，还携带有关文档的元信息（Metadata），可以用于后续的检索和过滤。以维基百科语料库的构建为例，数据的获取主要通过直接从维基百科网站提取页面内容来实现。这些内容不仅包括正文描述的内容，还包括一系列的元信息，例如文章标题，分类信息，时间信息，关键词等。

### （2）数据预处理

数据预处理包括**数据清洗**、**文本分块**两个过程。

**数据清洗**确保了数据的清晰度和准确性。数据清洗的常用的做法包括清除文本中的干扰元素，如特殊字符、异常编码和无用的HTML标签，以消除文本噪音；通过相似性度量来识别并删除内容重复或高度相似的冗余文档等等。

**文本分块**是将大块文本切割成较小单元的过程，例如把一篇长文章分为多个短段落。这么做的主要原因有两个：一是为了适应检索模型的上下文窗口长度限制，避免超出其处理能力；二是通过分块可以减少长文本中的不相关内容，降低噪音，从而提高检索的效率和准确性。

文本分块的效果直接影响后续检索结果的质量。如果分块处理不当，可能会破坏内容的连贯性。因此，制定合适的分块策略至关重要，包括确定切分方法（如按句子或段落切分）、设定块大小，以及是否允许块之间有重叠。

具体实现文本分块的流程通常是这样的：首先，将长文本划分为较小的语义单元，如句子或段落。接着，将这些语义单元组合成更大的块，直至达到预定的块大小，形成独立的文本片段。然后，重复此过程构建下一个文本片段。为了保持语义连贯性，通常会在相邻的文本片段之间设置一定的重叠区域。

## 2. 知识库增强

知识库增强是通过改进和丰富知识库的内容和结构，以提升其质量和实用性。这一过程通常涉及结合多个角度的信息，如**查询生成与标题生成** [36] 等，以此来

增强文档的语义表示能力从而提高检索模型的准确性。

查询生成指的是利用大型语言模型生成与文档内容紧密相关的查询。这些生成的查询能够准确地表达文档的语义，从而增强文档与用户查询的匹配度。例如，将介绍考拉和树袋熊的文档生成查询“考拉和树袋熊之间的关系是什么？”，这样的查询不仅准确反映了文档的主题，还能有效引导检索器更精确的检索到与用户提问相关的信息。

标题生成指的是利用大型语言模型为没有标题的文档生成合适的标题。这些生成的标题提供了文档的关键词和上下文信息，能来用来帮助快速理解文档内容，并在检索时更准确地定位到与用户提问相关的信息。对于那些原始文档中缺乏标题的情况，通过语言模型生成标题显得尤为重要。

### 3. 向量化管理

向量化管理涉及**数据向量化**和**向量检索**两个方面。数据向量化是将 RAG 的知识库转换为向量表示的过程，而向量检索则是基于这些向量进行高效搜索和匹配的技术。

#### (1) 数据向量化

在 RAG 中，数据向量化是指将知识库中的块和用户的问题转换为向量表示的过程，使得这些文本数据可以在高维空间中进行检索。根据向量表示的稀疏性或密集性，数据向量化方法可以分为稀疏向量化方法和稠密向量化方法两大类。

稀疏向量化方法主要依赖于文本中词项的出现频率和分布。它们通常使用词袋模型 (Bag of Words, BoW) [41]、词频-逆文档频率 (TF-IDF) [1] 等传统自然语言处理技术。这些方法的特点是生成的向量通常是高维且稀疏的，每个维度对应于一个特定的词项，值表示该词项在文本中的频率或权重。

稠密向量化方法依赖于深度学习技术，能够生成低维且密集的向量表示，这些向量能够捕捉丰富的语义信息。这类方法通常使用预训练语言模型，如 BERT[7]

等，将文本映射到连续的向量空间中。

### (2) 向量检索

向量检索是基于向量表示进行高效搜索和匹配的过程，即给定问题的向量，从语料库中查找与输入向量最相似的 top K 个向量。在 RAG 系统中，向量检索的核心挑战在于，给定一个特定的查询向量，如何在大量候选向量中快速且准确地识别出与其相似的向量或向量集合。这一过程涉及两个关键步骤：首先是选择恰当的相似性度量方法，其次是选取适宜的相似性检索算法及其实施方式。

#### 相似性度量方法

常见的相似性度量方法包括余弦相似度 (Cosine Similarity)、点积 (Dot Product) 和欧氏距离 (Euclidean Distance) 等。

余弦相似度是最常用的度量方法之一。在信息检索中，每个词项被赋予不同的维度，而一个文档由一个向量表示，其各个维度上的值对应于该词项在文档中出现的频率。余弦相似度因此可以给出两篇文档在其主题方面的相似度。

点积是另一种常用的相似性度量方法，它直接计算两个向量之间的乘积总和。点积可以从余弦相似度的计算公式中推导出来，通过将两个向量之间夹角的余弦值乘以两个向量的长度就得到点积。因此这种度量对向量的长度较为敏感，通常用于需要考虑向量大小的场景。

欧氏距离衡量的是向量在空间中的实际距离，常用于聚类和分类任务中。这种方法直观反映了两个点在多维空间中的物理距离，对于需要精确测量实际距离的应用场景尤其适用。

#### 相似性检索算法

在选择相似性检索算法时，考虑到数据集规模和查询效率至关重要。我们简要介绍几种常用的相似性检索算法。

暴力搜索是一种最简单直接的方法，通过计算查询向量与所有候选向量之间

的相似度进行搜索。虽然这种方法在小规模数据集上可以提供高精度的检索结果，但由于计算量大，在大规模数据集上效率较低。

k-近邻搜索 (k-NN Search) 通过在特征空间中寻找查询向量的最近邻居来实现相似性检索。常见的实现方式包括 KD 树 [3]、Ball 树 [8] 和局部敏感哈希 (LSH) [6]。KD 树适用于低维数据，Ball 树适用于中高维数据，而 LSH 则在处理大规模高维数据时表现出色，能够显著降低搜索时间。

图搜索算法通过构建图结构来进行相似性检索。常用的算法包括 HNSW [22] 和 NN-Descent[9]。这些算法特别适合大规模高维数据集，能够高效地进行近似最近邻搜索，提供良好的检索性能。

分块和索引技术则通过将数据集划分为若干小块或建立索引结构来提高检索效率。K-means 聚类通过将数据点分配到聚类中心优化搜索，倒排索引通过记录词项位置实现快速查找。这些技术在处理大规模数据时表现优异，能够在保持较高检索精度的同时显著提升效率。

## 实现方式

当处理的候选向量数量较少（如数千到数万个），可以采用 Python 中的 scikit-learn 库<sup>1</sup>或 Numpy 库<sup>2</sup>。这些工具相对轻量级，易于安装和使用，同时能提供较高的准确性和快速处理能力。scikit-learn 和 Numpy 支持的一个核心算法是 k-近邻 (k-NN)。k-NN 算法的工作原理是在特征空间中寻找一个数据点最近的 k 个邻居。该方法通常使用欧氏距离或余弦相似度来衡量数据点之间的相似性。

然而，面对大规模向量数据，scikit-learn 和 Numpy 库可能无法满足高效率和高性能的要求。为此，Facebook 研究团队开发并开源了 Faiss 库<sup>3</sup>，这是一款针对高效相似性搜索和向量聚类设计的库，适用于各种规模的向量集合。Faiss 提供了多

<sup>1</sup><https://scikit-learn.org/stable/>

<sup>2</sup><https://numpy.org/>

<sup>3</sup><https://github.com/facebookresearch/faiss>

种算法选择，这些算法不仅能在 CPU 上运行，部分还支持 GPU 加速，以满足不同场景下的性能需求。

### 6.3.2 查询增强

查询增强技术通过扩展和丰富用户查询的语义和内容，提高检索结果的准确性和全面性。查询增强包括语义增强和查询内容增强两大部分，通过同义改写、多视角分解问题以及生成背景文档来提升查询效果。

#### 1. 语义增强

##### (1) 同义改写

将原始查询改写成相同语义下不同的表达方式，改写工作可以调用大语言模型完成。比如，对于这样一个原始查询：“考拉的饮食习惯是什么？”，可以改写成下面几种同义表达：1、“考拉主要吃什么？”；2、“考拉的食物有哪些？”；3、“考拉的饮食结构是怎样的？”。每个改写后的查询都可独立用于检索相关文档。随后，从这些不同查询中检索到的文档集合进行合并和去重处理，从而形成一个更大的相关文档集合。这种方法通过将单一查询转化为多个同义表达，克服了原始查询可能存在的局限性，增加了检索结果的多样性和丰富性。

##### (2) 多视角分解问题

多视角分解问题采用分而治之的方法来处理复杂的问题。它首先分析问题并将其分解为从不同视角进行探讨的更简单的子问题。每个子问题能检索到不同的相关文档，这些文档可以提供部分答案。然后，从检索到的文档集合进行合并和去重处理，从而形成一个更大的相关文档集合。例如，对于这样一个问题：“考拉面临哪些威胁？”，可以从多个视角进行分解：1、“考拉的栖息地丧失对其有何影响？”；2、“气候变化如何影响考拉的生存？”；3、“人类活动对考拉种群有哪些威胁？”；4、“自然灾害对考拉的影响有哪些？”。通过这种方式，可以更有针对性地

检索并综合不同角度的信息，从而使得语言模型生成的最终答案更加全面和深入。

## 2. 查询内容增强

查询内容增强旨在通过生成与原始查询相关的背景信息和上下文，从而丰富查询内容，提高检索的准确性和全面性 [39]。与传统的仅依赖于检索的方式相比，查询内容增强方法通过引入大语言模型生成的辅助文档，为原始查询提供更多维度的信息支持。

生成背景文档是一种查询内容增强的方法。它指的是在原始查询的基础上，利用大语言模型生成与查询内容相关的背景文档。例如，对于用户查询“考拉的栖息地在哪？”，可以生成以下背景文档：

考拉是原产于澳大利亚的树栖有袋类动物，主要分布在东部和东南部沿海的桉树林中。这些地区提供了考拉主要食物来源——桉树叶。考拉的栖息地包括开阔的森林和木兰地，这里桉树丰富，不仅提供食物，还提供栖息和保护。考拉高度依赖特定种类的桉树，它们的分布与这些树木的可用性密切相关。保护工作集中在保护和恢复这些栖息地，以确保考拉种群的生存。

这些生成的背景文档可以作为原始查询的补充信息，提供更多的上下文内容，从而提高检索结果的相关性和丰富性。

### 6.3.3 文本检索

在构建了知识库并对查询进行优化处理之后，下一步就是知识搜索。知识搜索通过检索相关文档，从知识库中提取与用户查询相关的准确且丰富的信息。其目的是帮助语言模型生成高质量的回答。知识搜索不仅扩展了语言模型的知识范围，减少了幻觉的发生，还显著提高了生成答案的准确性和可靠性。

知识搜索的核心在于检索器的设计和应用。检索器负责将用户查询与知识库中的文档进行匹配，找到最相关的信息。根据处理文本表示方式的不同，检索器可以分为**稠密检索器**和**稀疏检索器**两大类，各有其优势和适用场景。稠密检索器依赖于深度学习，能够捕捉丰富的语义信息；而稀疏检索器侧重于直接利用词项的出现频率和分布，常用于处理特定关键词的查询。下面将详细的介绍这两种检索器以及现有的经典工作。

### 1. 稠密检索器

稠密检索器一般利用**预训练语言模型**生成**低维、密集**的向量表示，通过计算向量间的相似度进行检索。按照其底层使用的模型的结构，稠密检索器大致可以分为两类：**交叉编码类** (Cross-encoder)、**双编码器类** (Bi-encoder)，结构如图6.15。

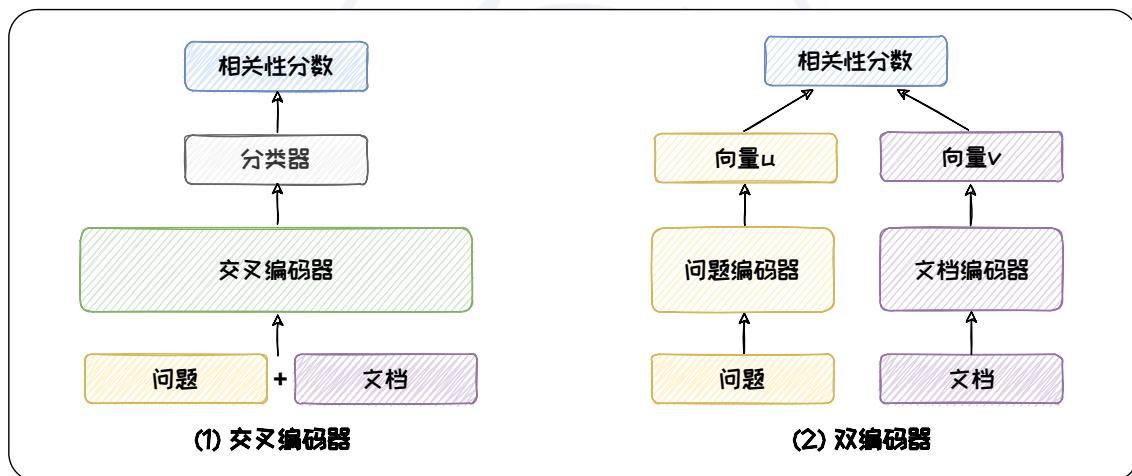


图 6.15: 不同稠密检索器对比图

#### (1) 交叉编码类

这种模型将查询和文档拼接在一起，然后通过交叉编码类模型（例如 BERT）生成一个介于 0 和 1 之间的输出值，表示输入句子对的相似性。其优点在于模型结构简单，能够实现查询和文档之间的深度交互。然而，由于交叉编码类模型需要进行高复杂度的交叉注意力操作，计算量大，因此不适合在大规模检索的召回阶段使用。这种模型主要用作重排阶段，在这一阶段，通过对少量候选文档进行更精确

的排序，可以显著提升检索结果的相关性。

## (2) 双编码器类

这类模型分别对查询和文档进行编码，得到各自的向量表示，然后计算两个文本表示之间的相关性得分。由于交互发生在最后的相似度判别阶段，可以预先计算并存储候选文档的向量表示，这使得双编码器非常适合在工业环境中部署，具有极高的匹配效率。不过，这种方法可能会忽略查询和文档之间的细粒度交互特征，从而影响匹配质量。

DPR (Dense Passage Retriever) [16] 是稠密检索器的一个开创性工作，由 Facebook 在 2020 年提出，属于双编码器类的方法。与交叉编码器类检索器不同，DPR 不需要对查询和文档执行高计算成本的交叉注意力操作。DPR 通过独立编码查询和文档，在相似度判别阶段进行交互，使其更适用于大规模文档集的实时检索。

DPR 使用文本编码器将任何文本段落映射到一个低维的实数向量。在检索时，DPR 采用另一个编码器将输入问题映射到同样的低维向量，并检索与问题向量最接近的段落。问题与段落之间的相似度通过它们向量的点积来衡量。在 DPR 中，使用了两个独立的 BERT 网络，并将 [CLS] 标记的表示作为输出。

在 DPR 的训练阶段，目标是优化编码器，以使得问题与段落之间根据相似度进行排序。本质上这是一个度量学习问题，目的是构建一个向量空间，在这个空间中，相关的问题和段落对之间的距离（即相似度）比不相关的对更小。具体来说，训练数据由多个实例组成，每个实例包含一个问题和一个相关的（正面的）文档，以及若干个不相关的（负面的）文档。最大化问题和正面段落之间的相似度，同时最小化问题与负面段落之间的相似度，这种训练方式能够有效提升编码器的性能，使其在检索相关段落时表现优异。

在选择一个问题的负面示例时，DPR 给出了三种不同的策略：

1. 随机选择：从整个语料库中随机选择一个段落作为负面示例。

2. BM25 选择：选择 BM25 算法返回的与查询问题标记匹配度高但不含答案的段落作为负面示例。
3. 强负例选择：使用与训练集中其他问题配对的正面段落作为负面示例。

这种选负例的方式也是导致 DPR 性能优异的因素之一。

在双编码器的基础上，还有一些其他类型的检索器，如晚交互类，例如 Col-BERT[17]，这类模型在最后阶段进行细粒度的表征交互，以得到最终的相似度评分。由于保持了双塔结构，计算量相对较小，同时能够捕捉到更准确的相似度信息。还有一类基于注意力的聚合类模型（Attention-based Aggregator），例如 PolyEncoder[10]，这类模型为每个问题生成多个不同的表征，模型随后使用这些全局特征向量与文档的向量进行比较，得到最终的匹配度。这种方法的优点是通过利用问题的全局特征来增强表征的丰富性和适应性，但代价是增加了计算成本，因为需要处理一个问题的多个特征向量。

## 2. 稀疏检索器

稀疏检索器（Sparse Retriever）是指在检索过程中使用稀疏表示方法来匹配文本的模型。这类检索器的特点是它们关注文档中特定词项的出现，而非词项的语境或序列。典型的稀疏检索技术包括 TF-IDF[1] 和 BM25[32] 等，它们通过分析词项的分布和频率来评估文档与查询的相关性。在稀疏检索器中，文档和查询通常被表示为词项向量，其中元素代表词项的重要性或频率。

### (1) TF-IDF

TF-IDF 用于衡量词语在文档或语料库中的重要性。词语的重要性与其在文档中出现的次数成正比，但与语料库中的频率成反比。

词频（Term Frequency, TF）表示词语在文档中的出现频率，标准化以避免偏

向长文档。对于文档  $d_j$  中的词语  $t_i$ ，其重要性表示为：

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}} \quad (6.1)$$

其中， $n_{i,j}$  是词语  $t_i$  在文档  $d_j$  中的出现次数， $\sum_k n_{k,j}$  是文档  $d_j$  中所有词语的出现次数之和。

**逆文档频率** (Inverse Document Frequency, IDF) 衡量词语的普遍性。某一词语的 IDF 计算为：

$$idf_i = \log \frac{|D|}{|\{j : t_i \in d_j\}|} \quad (6.2)$$

其中， $|D|$  是总文档数， $|\{j : t_i \in d_j\}|$  是包含词语  $t_i$  的文档数。

最终，TF-IDF 值为：

$$tfidf_{i,j} = tf_{i,j} \times idf_i \quad (6.3)$$

TF-IDF 通过高词频和低文档频率产生高权重，倾向于过滤常见词语，保留重要词语。

## (2) BM25

BM25 (Best Match 25) 是一种改进自 TF-IDF 的文本检索算法。它在 TF-IDF 的基础上引入了文档长度归一化和词项饱和度的概念，以提高检索的准确性和相关性。BM25 通过调整参数来优化词项频率的影响，并考虑文档长度对评分的影响，从而更合理地评估词项的重要性。

BM25 算法的核心包括：

1. **词项频率 (TF)**：BM25 对传统词频进行了修改，引入参数以控制词项频率的饱和度，减少高频词项对评分的影响。
2. **逆文档频率 (IDF)**：BM25 的 IDF 计算略有不同，旨在降低常见词项的权重。
3. **文档长度归一化**：通过参数调整，BM25 对不同长度的文档进行长度归一化，以避免长文档因包含更多词而获得不公平的高分。

BM25 的优势在于它通过参数调整和文档长度归一化，能够更准确地反映文档与查询的相关性，从而在实际应用中提供了高质量的检索结果。尽管 BM25 不考虑词项之间的上下文关系，但它在处理大规模文本数据时表现出色，因此被广泛应用于搜索引擎和其他信息检索系统中。

### 3. 对比分析

在 RAG 的框架中，稠密检索器和稀疏检索器各有优势。稠密检索器通过预训练语言模型生成的向量嵌入来处理查询，能够捕捉查询和文档之间的复杂语义关系，尤其在处理模糊或多义性查询时表现突出。然而，其计算成本较高，特别是在大规模数据集上。相比之下，稀疏检索器依赖于关键词频率和文档中的词项分布，通过 TF-IDF 和 BM25 等算法快速评估文档与查询的相关性。这种方法在处理明确关键词的查询时效率极高，适合快速筛选文档。

在实际应用中，RAG 系统可以通过结合这两种检索器来发挥各自的优势。例如，可以首先使用稀疏检索器快速缩小候选文档的范围，然后用稠密检索器进行更深入的语义匹配。这种结合方法不仅优化了检索效率，还提高了答案的准确性和相关性，使 RAG 系统能够在不同场景下更灵活地响应用户查询。

#### 6.3.4 检索结果重排

在知识检索阶段，检索器可能检索到与查询相关性不高的文档。因此，重排的目的是对所有检索到的段落与问题进行相关性评分，将最相关的段落找出来并放置在提示词的最前面或最后面，同时去除相关性较低的段落。这一技术主要分为三种方法：传统方法、基于微调的方法和基于 Prompt 的方法。

##### 1. 传统方法

在传统方法中，常常使用交叉编码器（Cross-encoders）来评估文档与查询之间的语义相关性，这是因为相比于双编码器架构，交叉编码器通过将查询和上下

文合并作为输入，在查询和文档中执行注意力机制，从而拥有更好的性能。

以 MiniLM-L<sup>4</sup>模型为例，它是一个参数数量较少但性能卓越的交叉编码器，基于微软的 MS MARCO 数据集训练，被业界广泛认可为领先的重排器之一。该模型能够精确评估查询与文档的相关性，有效优化搜索结果的排序。此外，还有其他一些在线重排模型可通过 API 直接访问，例如 Cohere<sup>5</sup>，它们同样提供精确的重排功能。对于希望探索其他高性能的重排器的用户，可以参考 MTEB<sup>6</sup> 排行，该榜单汇集了很多性能优秀的重排模型。

## 2. 基于微调的方法

基于微调的方法涉及对语言模型进行微调，以执行特定的重排任务，其中 FIT-RAG[25] 是一个典型的例子。

在重排过程中，FIT-RAG 采用了一种创新的双标签学习机制，该机制包含两个关键标签：‘Has\_Answer’（标识文档是否包含问题的答案）和 ‘LLM\_Prefer’（指示文档是否有助于语言模型生成准确的回答）。基于这两个标签，FIT-RAG 构建了一个评分器，用于对检索到的文档进行评分。具体操作是，将 ‘Has\_Answer’ 和 ‘LLM\_Prefer’ 的分数通过加权求和的方式整合，然后依据这一综合评分对初步检索到的 100 个文档进行重新排序。最终，FIT-RAG 选取综合评分最高的 10 个文档作为进一步处理的候选文档集。

## 3. 基于 Prompt 的方法

基于 Prompt 的方法通过设计 Prompt 来利用大型语言模型的能力，使其执行重排任务。这种方法简单而高效，能够充分利用模型的深层语义理解能力。

以 RankGPT[34] 为例，该模型结合了生成排列和滑动窗口技术，能够直接对段落进行重新排序。其使用的 Prompt 模板如图6.16所示。

<sup>4</sup><https://huggingface.co/cross-encoder/ms-marco-MiniLM-L-6-v2>

<sup>5</sup><https://cohere.com/rerank>

<sup>6</sup><https://huggingface.co/spaces/mteb/leaderboard>

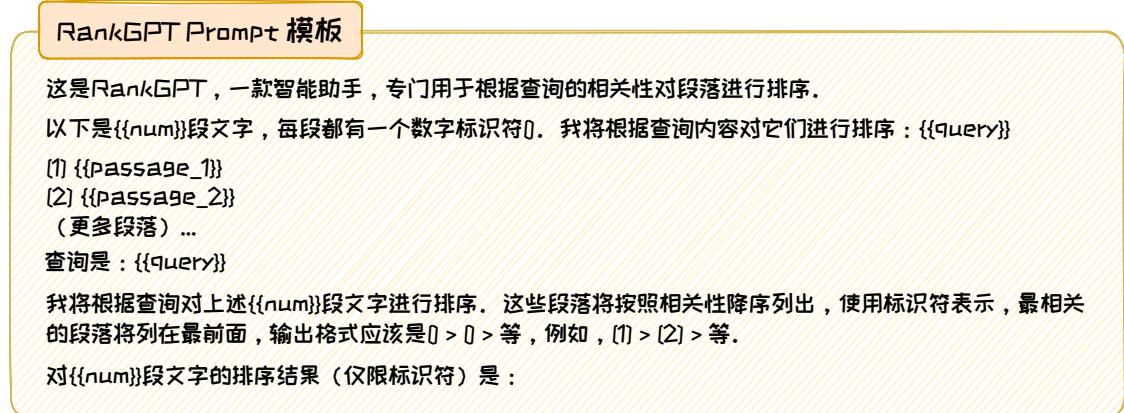


图 6.16: RankGPT Prompt 模板图

为了应对处理大量文档时可能遇到的计算资源限制，RankGPT 采用了滑动窗口技术，使得模型在每一步仅处理包含特定段落的有限窗口，从而优化了处理过程。

## 6.4 生成增强

在前面的章节中，我们探索了检索器如何高效地进行检索，在这一节中，我们将围绕生成器如何高效利用检索增强这一问题出发，主要从以下四个方面展开讨论：(1) 增强必要性判断，确定何时需要检索增强，以确保生成器只在需要时进行增强，提升效率并避免无关信息干扰；(2) 增强实施位置，讨论生成过程中插入检索信息的合适位置，以最大化信息的效用和相关性；(3) 增强性能改善，针对复杂查询与模糊查询，讨论常见的细化增强方式；(4) 增强效率提升，介绍现有的知识压缩与缓存加速策略，以提升生成模型利用检索知识的效率。

### 6.4.1 增强必要性判定

在这一节中，我们首先对增强必要性展开讨论，即是否需要检索增强。检索增强确实能有效地提升大语言模型的生成质量，然而，它也并不是完美的，主要存在两方面问题：（1）**效率低下**：检索增强需要进行一系列的信息检索、筛选、处理等过程，并且增强的文本使得大语言模型需要处理更多的信息，这都会带来时间成本和计算成本的增加。（2）**性能损失**：检索增强并不总能带来提升的效果。有研究 [31] 显示语言模型在使用低质量的增强文本时甚至比不使用表现出更差的性能，并且无关的增强文本还会引入额外噪声。例如图 6.17 中的例子，对于问题“树袋熊一般在哪里生活？”，大语言模型能够在不需要外部知识的情况下直接回答正确，说明它模型内部已经具备了相关的知识。

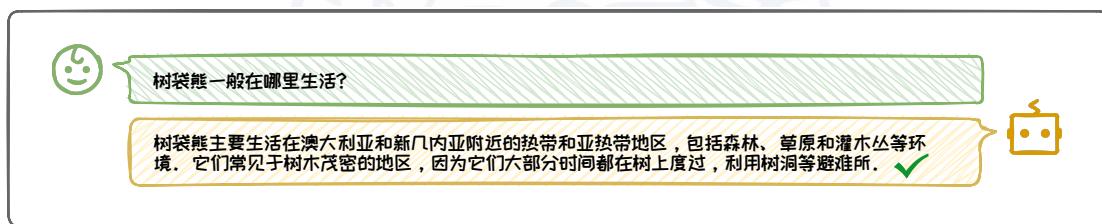


图 6.17: 模型已知问题示例

接下来，我们为它提供一段相似但不相关的外部知识，即与树袋熊名称非常相似的动物袋熊的相关信息，如图 6.18 所示，这时我们发现对于同一个问题，模型给出了错误答案，将知识文本中关于袋熊的信息错误地理解为树袋熊的相关信息。

从上面的例子中，我们可以直观地理解低质量的检索增强文本所带来的负面影响，因此，讨论何时需要检索增强是十分必要的。我们发现，解决这个问题的核心点在于如何判断语言模型本身是否已经具备相关的内部知识，即 **Self-Knowledge**。

为了判断模型是否具有 **Self-Knowledge**，现有的工作大致从两个角度出发：（1）模型外部，通过 Prompt 直接询问或基于统计的方法进行预测；（2）模型内部，通

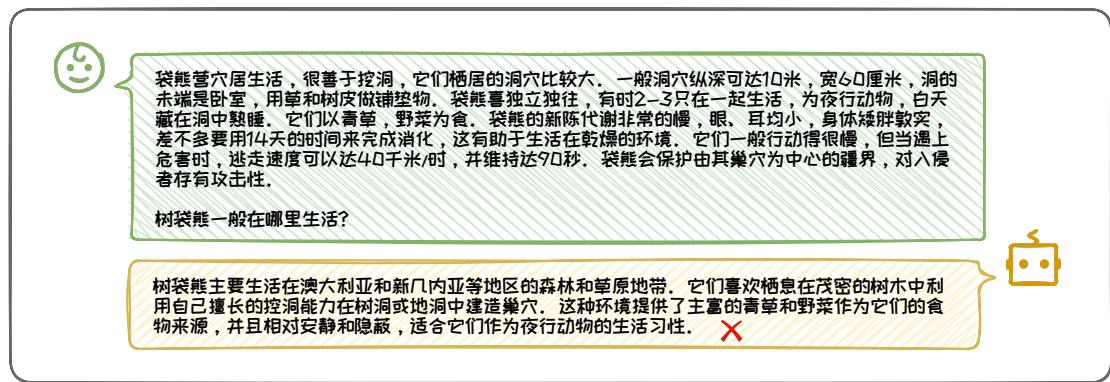


图 6.18: 检索增强损害性能示例

过检测内部状态信息的变化进行预测，这种方法通常不适用于黑盒语言模型。

### 1. 模型外部

从模型外部的角度出发，主要有以下思路，(1) 设计 **Prompt** 直接询问，根据模型的回答情况进行判断，这种方案的局限性在于模型存在过度自信的问题；(2) 根据当前知识在模型**训练数据**中的出现频率，估计模型对该知识的学习情况，这种方案的局限性主要是耗时且部分模型无法获取训练数据；(3) 利用**启发式统计量**来拟合训练数据分布，这种方案的关键在于如何设计合理的启发式统计量。

#### (1) 直接询问

首先，一个简单的想法是通过 **Prompt** 的形式直接询问语言模型，如图 6.19 所示。然而，在实验中我们发现，大语言模型存在过度自信的问题，也就是对于无法回答的问题，它依然认为自己不需要外部知识的帮助，因此这种判断方式的准确率是非常低的。

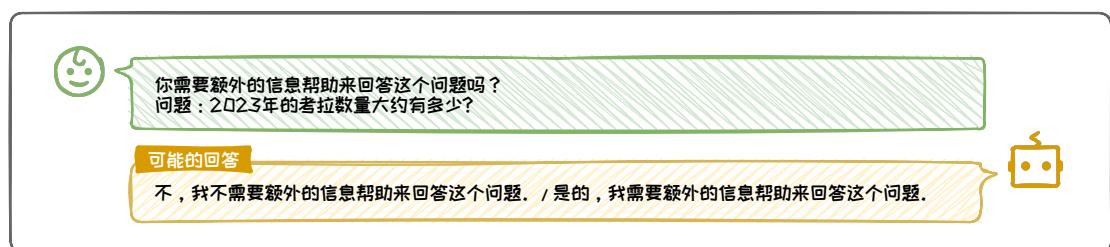


图 6.19: 直接询问的 **Prompt** 示例

鉴于大语言模型在识别自身知识边界方面存在局限，另一种方式是从模型输出的不确定性角度进行评估。这主要通过量化模型对其输出的置信度来实现。一些工作 [24, 29] 通过让语言模型重复多次地回答同一个问题，如果模型不具备相应的知识，那么每次的输出会更加随机，反之，输出则会有更高的一致性。然而，这种方式需要耗费大量的时间和计算资源，在实际使用中可行性较低。

## (2) 训练数据分布

我们知道，语言模型所具备的内部知识来源于**训练数据**。因此，一个很自然的想法是，如果模型的训练数据中不包含当前问题相关的信息，那么，模型也就没有学习过相应的知识。另外，类别人类的学习模式，我们通常对常见或反复学习的知识掌握程度更好，而对低频的知识则较差。因此，对于训练数据中出现频率很低的知识，模型对它们的 Self-Knowledge 程度可能也是比较低的。

目前已经有一些工作探索了这一问题，例如 Kandpal 等人在 ICML2023 发表的论文 [15] 中显示，**知识在训练数据中的出现频率与模型对该知识的记忆程度是正相关的**。具体而言，研究结果揭示了模型在处理那些在训练数据中频繁出现的知识时表现出更高的准确性。进一步地，当从训练数据中移除这些知识并重新训练模型时，其准确率显著降低。这些实验结果表明，训练数据的分布的确可以为 Self-Knowledge 的预测提供一种可行的方法。

然而，这种方式存在一定的局限性。首先，由于语言模型的训练数据规模已经达到了数万亿 Token，因此对训练数据进行统计非常耗时。更重要的是，对于许多商业大语言模型，例如 ChatGPT、GPT4 等，它们的训练数据并不是公开可获取的，在这种情况下，实施这种方案是不可行的。

## (3) 外部统计拟合

一些工作尝试用外部统计手段来拟合训练数据的分布。比如 Alex 等人在 ACL2023 发表的论文中 [23] 提出了一种启发式方法，即实体的流行度。结合上一章中的讨

论，大语言模型存在长尾问题，也就是对训练数据中低频出现的知识掌握不足，而对更“流行”的知识掌握更好。因此这篇工作提出用 Wikipedia 的月页面浏览量来衡量实体的流行度，浏览量越大，表明该实体越流行，也就更可能被大语言模型所记忆。来看图 6.20、图 6.21 两个例子，分别涉及流行与不流行知识。可以发现，对于流行知识考拉，模型能够直接给出正确的回答，而对于不流行知识考拉的基因数量，模型给出了错误的答案，正确答案应为 26,558。可见，实体的流行度确实在一定程度上反映了模型的内部知识，因此，可以通过设定一个流行度阈值来判别模型是否具有相应的内部知识。

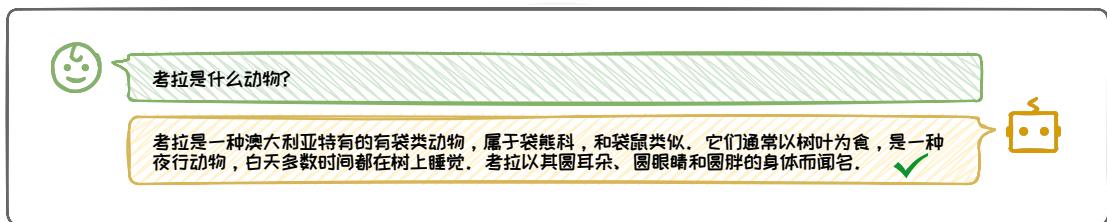


图 6.20: 模型对流行知识的回答示例

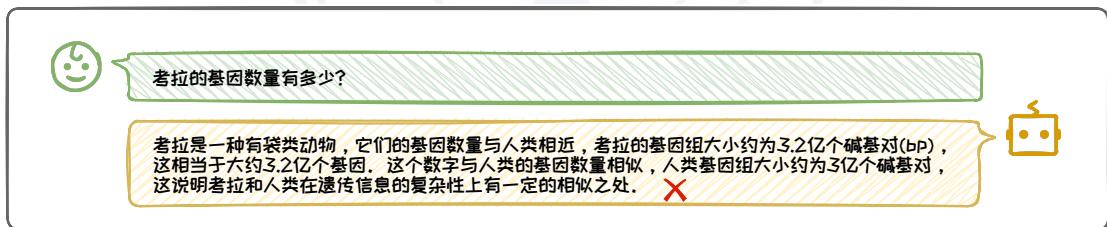


图 6.21: 模型对不流行知识的回答示例

然而，这种流行度的定义依赖于时间，并且也不一定能很好地拟合训练数据的分布，因此它也存在一定的局限性。

### 2. 模型内部

另一种判别 Self-Knowledge 的方法是从大语言模型的内部状态出发，分析模型在生成时每一层的隐藏状态变化，比如注意力模块的输出、多层感知器 (MLP) 层的输出与激活值变化等。大语言模型在生成文本时，实际上是对输入序列进行

建模和预测，模型内部状态的变化反映了模型对当前上下文的理解和下一步预测的确定性。如果模型在某些情况下表现出**较高的内部不确定性**，如注意力分布较为分散、激活值变化较大等，就可能对当前上下文缺乏充分的理解，从而无法做出有把握的预测。

一些模型编辑与幻觉检测领域的工作已经有了启发性的发现，例如 Meng 等人的研究中 [27] 发现，**模型的内部知识检索主要发生在中间层的前馈网络中**。而面对存在 Self-Knowledge 和不存在 Self-Knowledge 的问题，模型的中间层表现出不同的变化情况，针对这种特性，我们可以训练分类器进行判别，这种方法也叫做探针。

已经有一些工作进行了初步的实验，例如 Liang 等人在 2024 年发表的论文中 [21]，对三种类型的内部表示设计了探针实验，即**注意力层输出 (Attention Output)**、**MLP 层输出 (MLP Output)** 和 **隐层状态 (Hidden States)**。对于每个输入的问题，作者分别训练了一个线性分类器来预测该问题是否属于模型“已知”（即模型具备相关知识），还是“未知”（即模型缺乏相关知识），如图 6.22 所示，分类器的目标是根据问题对应的内部表示来预测模型是否含有足够的内部知识来回答该问题。

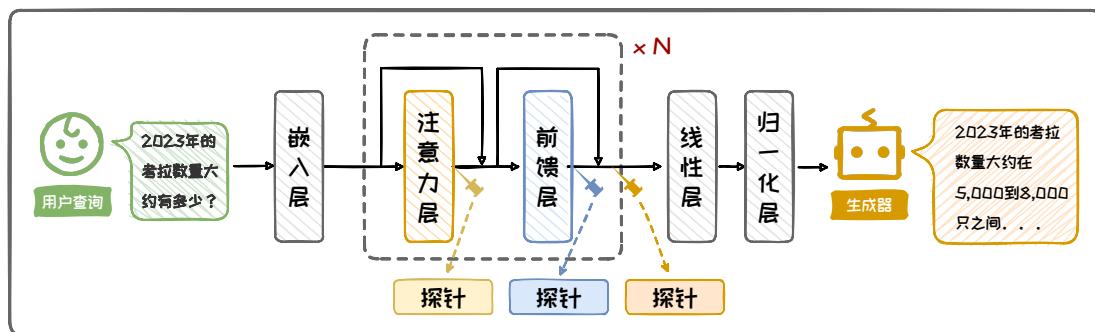


图 6.22: 模型内部状态探针

论文中的实验结果发现，不同大模型在利用中间层的内部表示进行分类时，均

能够实现较高的分类准确率。这一结果表明，中间层的内部表示能够有效地反映模型对问题的理解，无论是识别已知信息还是处理未知挑战。

然而，这种从内部状态进行检测的方案也存在一定的局限性，它并不适用于黑盒语言模型，因为我们无法直接获取它的内部表示。另外，模型的输入也需要仔细设计，因此有时模型之所以表现出不确定性，可能是由于问题本身的模糊性或歧义性所导致，而不是对问题缺乏知识。

总体而言，基于内部状态评估 Self-Knowledge 的工作目前还处于初步探索阶段，很多具体的实现细节有待进一步完善，比如如何构造“已知”和“未知”问题的数据集、如何量化内部状态的不确定性、不同内部表示的优劣对比，如何设计内部状态检测方法等。我们需要在更大的数据集和更多的模型架构上进行实验研究，以验证这一方法的有效性和普适性。总体来说，这是一条值得探索的新路径，它有望为我们从内部视角了解语言模型的知识提供新的视角和方法。

### 6.4.2 增强实施位置

在判别检索增强的必要性之后，接下来的问题在于如何利用检索到的外部知识文本。目前最常见的方式是将外部知识文档以 **Prompt** 的形式融入到模型的输入端中，这是一种简单有效且灵活的方法。除此之外，还可以将外部知识文本的 **embedding** 向量直接插入到模型的中间层或在模型的输出端利用检索文档对生成文本进行后矫正。图 6.23 展示了这三种不同增强策略的位置，这些策略旨在通过不同方式整合外部知识，以提高语言模型在处理复杂问题时的性能。接下来我们对每一种方式进行简单的介绍。

#### (1) 模型输入端

首先是模型输入端的知识注入，这种方式通过将检索到的外部知识文本直接以 **Prompt** 的形式与用户查询结合输入给模型。目前现有的 RAG 工作，绝大多数

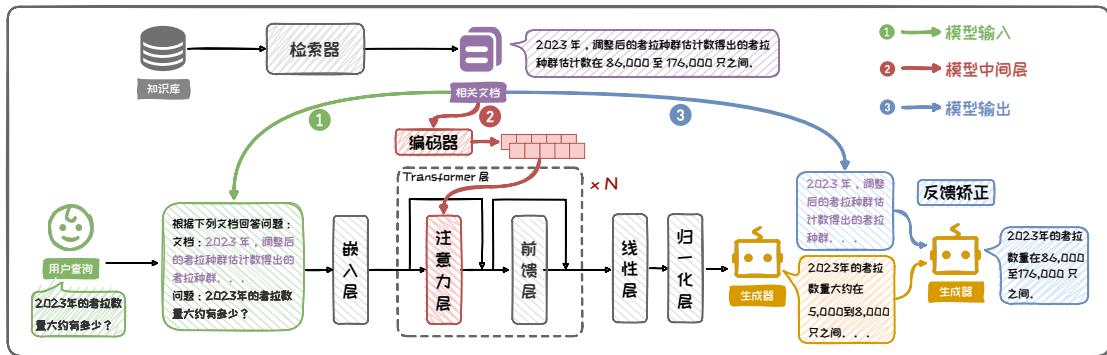


图 6.23: 增强实施位置示意图

是采用这种形式将知识进行注入。这种方式的重点在于如何设计合适的 Prompt 以及如何排列所检索到的外部知识，使得模型能够更好地理解与利用这些信息，在这个过程中，也可以利用一些 Prompt 技巧，例如 CoT [35] 等，这一部分技巧可以参考本书第三章 Prompt 工程的内容。

总体而言，这种方法的优点在于其直观性和易于实施性。模型可以直接从输入的上下文中提取信息，无需复杂的处理或转换。然而，这种方法也存在一些局限性，如果检索到的文本过长，可能会导致输入序列过长，影响模型的处理性能，增加模型的计算负担，甚至可能超出模型的最大序列长度限制。此外，模型需要具备较强的能力来评估和综合利用来自不同来源的信息。

## (2) 模型中间层

除了以 Prompt 的形式输入模型，还可以将检索到的文本通过一个预训练的编码模型转换为向量表示，然后将这些向量插入到模型的中间层，与模型自身的表示进行融合，这是一种更为精细的整合方式。例如在第 6.2 节经典架构中介绍的工作 Retro [5]，就是采用这种方式进行利用的典型工作。它通过将检索到的文本转换为向量表示，并通过交叉注意力机制整合检索到的文本片段，以改善模型的预测。

总体而言，这种方法可以更深入地影响模型的内部表示，可能帮助模型更好地理解和利用外部知识。同时，由于向量表示通常比原始文本更为紧凑，这种方法

可以减少对模型输入长度的依赖。然而，这种方法的过程较为复杂，需要对模型的结构和训练过程有深入的理解。

### (3) 模型输出端

最后是在模型的输出端，利用检索到的知识对生成的文本进行校准，是一种后处理的方法。在这种策略中，模型首先生成一个初步的回答，然后利用检索到的知识信息来验证或校准这个答案。这可以通过比较生成文本与检索文本的知识一致性或利用大语言模型本身进行判别矫正。例如 Yu 等人提出的 REFEED 框架 [40]，就是首先利用语言模型生成初步答案，随后检索器使用原始查询和初步答案检索相关知识文档，并将检索到的信息用于对初步答案进行检查和矫正。

这种方法的优点是可以确保生成的文本与外部知识保持一致，提高答案的准确性和可靠性。然而，这种方法依赖于检索文档的质量和相关性，如果检索到的文档不准确或不相关，可能会导致错误的校准。

总体而言，上述三种位置的增强方式各有优劣，但目前最常见的仍是以 Prompt 的形式直接输入模型。而考虑具体的应用场景和需求，也可以灵活选择不同的方案或进行组合，灵活地结合多种方法。

### 6.4.3 增强性能改善

在这一节中，我们将讨论如何对增强的性能进行改善。我们知道，对于一些复杂任务，通常可以将它拆解为一系列的简单中间任务，往往可以提升大语言模型解决问题的能力，例如在前面第三章讨论的 CoT 方法，就是通过一系列中间步骤的生成激发模型正确生成的能力。此外，在检索增强的过程中，对于用户输入的模糊问题，我们需要进一步的检索来细化问题，明确用户意图，提高检索文本的质量与相关性。

因此，在许多场景中，单次检索并不总是足够的，往往需要进行多次检索的过

程，大致可以分为两类，(1) **分解式检索**，对于一些复杂问题的解答往往需要多个知识点的支撑，因此可以将其分解为多个子问题，在前一个子问题回答的基础上进行迭代地检索，得出下一个子问题的答案。(2) **渐进式检索**，有时对于一些表达较为模糊的问题，我们需要在单轮的检索基础上，进一步深入，考虑到问题的多个模糊维度并通过多次的检索收集相应的知识。

## 1. 分解式检索

我们首先讨论复杂问题进行迭代分解的场景。对于一些复杂问题，往往需要进行多跳 (multi-hop) 的理解，涉及多个知识点，而单次的检索往往只能获取与问题相关的部分知识，无法找到可以直接回答问题的片段，因此语言模型仍无法正确回答该问题。在这种情况下，模型往往需要将多跳问题分解为一个个子问题，分别获取相关信息，模型综合所有子问题才能正确得出答案。

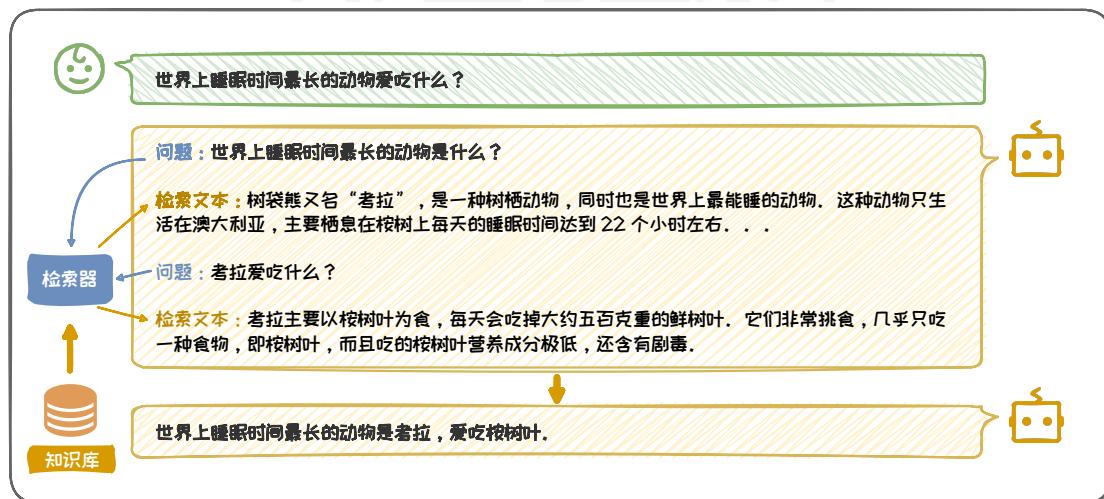


图 6.24: DSP 流程示意图

为了解决这一问题，Khattab 等人提出了一个创新的框架，称为 **DSP** [18]，即 **DEMONSTRATE-SEARCH-PREDICT**。这个框架的核心就在于**将多跳问题进行分解**，然后进行多次迭代地检索，一步步得到最终的答案。这个框架的主要部分包括，(1) **DEMONSTRATE** 模块，主要是构建高质量的问题分解示例，通过上下文

学习的方法帮助大语言模型学会如何分解问题；(2) **SEARCH** 模块，主要是生成子问题然后执行检索，并重复这个过程，最终为下一阶段准备一个综合的上下文信息；(3) **PREDICT** 模块，主要是根据上一阶段得到的信息生成最终回答。

我们以一个简单的多跳问题为例说明 DSP 框架的运行流程。如图 6.24 所示，我们询问“世界上睡眠时间最长的动物爱吃什么？”，我们首先让语言模型提出第一个子问题，“世界上睡眠时间最长的动物是什么？”，然后进行第一步检索，得到睡眠最长的动物是考拉，接下来语言模型基于这个信息提出第二个子问题，“考拉爱吃什么？”，然后进行第二步检索，得到考拉主要吃桉树叶的信息，接下来将这个过程中得到的所有信息以 Prompt 的形式输入给大语言模型，就可以得到正确的回答，即“世界上睡眠时间最长的动物是考拉，爱吃桉树叶”。

分解式检索的优势在于将复杂问题化整为零，降低了单次检索的难度，有利于充分利用检索系统的能力。同时逐步推进也有利于语言模型深入理解和把握问题本质。当然，分解质量直接影响最终答案的完整性和准确性，因此需要根据具体问题特点进行分解设计。此外，分解式检索还可能存在错误累积的可能性。

### 2. 演进式检索

接下来我们讨论需要演进式检索的场景，对于某些**模糊问题**，我们有时很难判断这个问题的目的是什么。例如我们有一个问题：“澳洲的国宝动物爱吃什么？”，由于澳洲的国宝动物不止一种，因此我们并不清楚这个问题询问的是其中的哪一种动物。这是由于**同名实体**的存在引起的。还有一些模糊是由于问题过于宽泛导致**指代不清晰**，比如问题：“考拉的平均体重有多少？”，在这个问题中，我们不清楚是哪一种类的考拉，并且雄性考拉与雌性考拉的体重也是完全不同的。

因此，在这些场景下，我们有必要进行多次的检索，考虑问题可能的多个维度，从而帮助语言模型生成指代清晰的回答。为了解决这一问题，Kim 等人提出一种名为 **TREE OF CLARIFICATIONS (TOC)** 的新型框架 [19]，引导大语言模

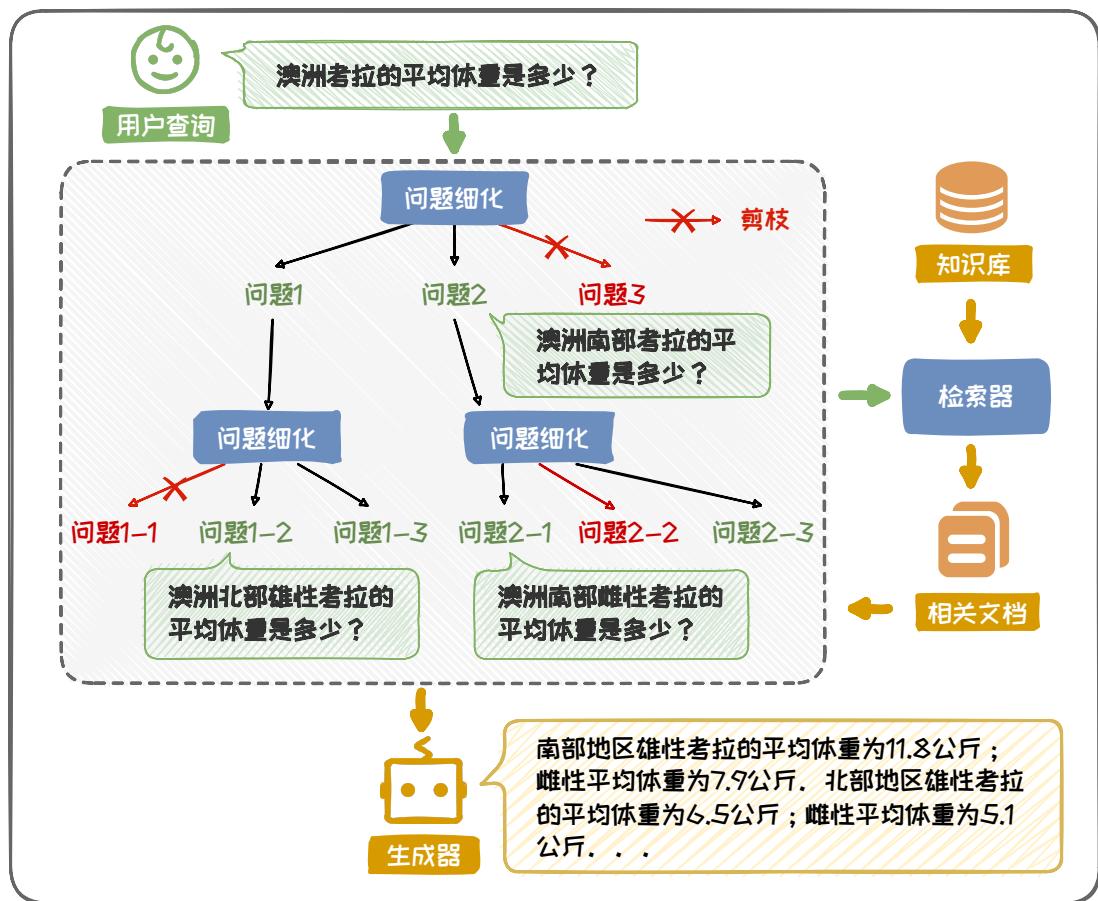


图 6.25: TOC 框架流程示意图

型在树状结构中探索给定模糊问题的多种澄清路径。接下来，我们通过例子说明 TOC 框架如何应用于上述场景，以及它是如何通过递归式检索来解决模糊问题的。

图 6.25 是 TOC 框架的整个工作流，图中我们以“**澳洲考拉的平均体重是多少？**”这个问题为例。TOC 框架首先启动第一轮检索，依据检索到的相关文献资料和原始问题，生成一系列具体的细化问题，接着，针对每个细化问题，框架独立展开深入的检索并对问题进一步细化，例如：“**澳洲北部考拉的体重是多少？**”，“**澳洲北部雄性考拉的平均体重是多少？**”在这个过程中，我们会根据细化问题与原问题的相关性与知识一致性进行剪枝。最终，我们能够构建出多条完整的知识路径，每条路径的末端（叶节点）都代表了对原始问题的不同且有效的解答。通过整合所

有有效的叶节点结果，我们能够得出一个精确且全面的长回答，比如对这个问题我们可以得到：“南部地区雄性考拉的平均体重为 11.8 公斤；雌性平均体重为 7.9 公斤。北部地区雄性考拉的平均体重为 6.5 公斤；雌性平均体重为 5.1 公斤……”

这种渐进式的检索可以较好地改善语言模型对模糊问题的生成效果。然而，它也存在很大的局限性，由于需要进行多轮检索并生成多个答案路径，整个系统的计算量随着问题复杂度呈指数级增长，多轮的检索与生成除了会带来很大的时间延迟，还会造成多个推理路径的不稳定性，这在实际应用中都会带来很大的隐患。

### 6.4.4 增强效率提升

尽管现有的 RAG 框架显示出了强大的性能，但在增强生成的效率方面仍有较大的提升空间，进一步的优化可以减少计算资源的消耗，加快模型的响应速度，并降低运行成本。针对输入 **Prompt** 存在冗余与推理计算存在重复两方面问题，本章我们将主要介绍输入 **Prompt** 压缩与 **KV-cache** 机制这两种解决方法。

#### 1. 输入 **Prompt** 压缩

在 RAG 的推理过程中，较长的输入往往会导致较高的时间和成本开销。为了解决这一问题，可以输入信息进行压缩，一方面可以减少无关或错误信息的干扰，筛选出与查询最相关的信息，提升输入的有效性；另一方面可以有效减少模型处理输入所需的计算量，提升相应速度。这种压缩处理在缩短文本长度的同时保留了文本的关键内容，有助于提升模型的效率和响应速度。

一些工作是基于输入 **Token** 的困惑度，如 Jiang 等人提出的 **LongLLMLingua** [13] 框架，它是基于该作者的另一篇工作 **LLMLingua** [12] 的基础上发展而来的，它利用小型语言模型来计算提示中每个 token 的困惑度，并丢弃那些困惑度低的 token，从而降低信息熵。但与 **LLMLingua** 仅基于困惑度进行压缩不同，**LongLLMLingua** 引入了问题感知机制，优先保留与问题内容更相关的 token，有效

提升处理长文本时的性能和准确性。

LongLLMLingua 的压缩流程如图 6.26 所示，包括四个主要步骤。首先是问题感知的**粗粒度压缩**，通过使用小型语言模型来评估每个文档或段落的困惑度，筛选出与问题最相关的文档。接着，执行问题感知的**细粒度压缩**，在粗粒度压缩的基础上计算每个 token 的困惑度并进行压缩。此外，论文还引入了**文档重排序机制与动态压缩比率**，分别用于对文档重要性进行排序与对动态调整压缩比率，以提升模型对信息的利用能力，并确保高度相关的信息被较少压缩，以保证关键信息的完整性。最后，进行**后压缩子序列恢复**，以确保压缩过程中不丢失关键信息。

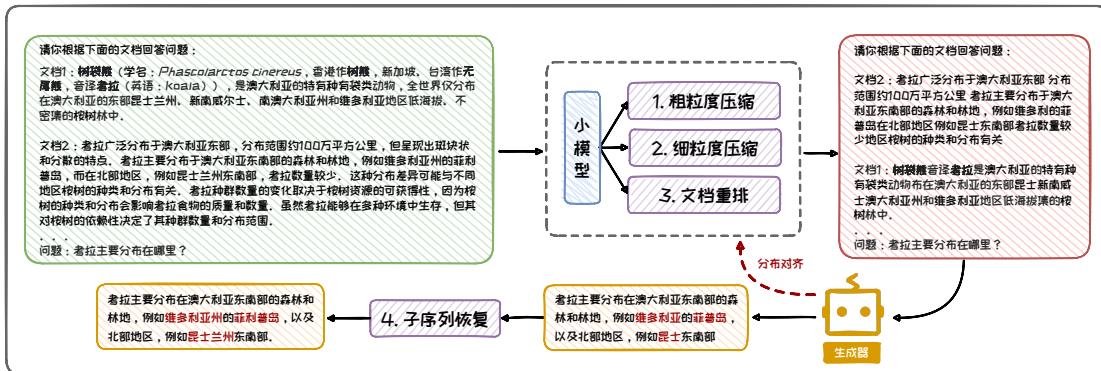


图 6.26: LongLLMLingua 框架流程示意图

除了基于困惑度的压缩方法外，Mao 等人提出了一个创新的 RAG 框架 **FIT-RAG** [26]，其中的子文档 Token 压缩模块利用预先训练的双标签文档打分器，根据**文档的事实性**（即是否包含查询的答案信息）与**模型的偏好程度**（即文档是否易于模型理解）来评估文档中各个片段的有用程度。具体而言，该模块首先将检索到的文档分割成较小的子文档片段，每个片段包含一定数量的句子，以保证每个片段的语义完整度；接下来模块使用双标签打分器对这些片段进行打分，以筛选出每个文档中与当前查询最相关的片段；这些片段随后被输入一个过滤器，它采用贪婪搜索策略，逐步添加子文档，以筛选出满足要求的最少子文档组合。该模块有效减少了输入 Token，同时确保模型能够访问到生成准确答案所需的关键信息。

此外，还有一些通过训练信息提取器来提炼和浓缩检索到的文档的方法，如 **PRCA** [38] 和 **RECOMP** [37]。

### 2. KV-cache 机制

KV-cache 压缩技术专注于优化模型在处理外部知识信息时的内存使用。在与模型交互的过程中，会存在不同查询需要利用同一文档的信息，因此在这个过程中会有重复的计算过程。因此，一个自然的想法是将这些重复计算的键值张量存储下来，以供未来的重复调用，极大提升计算速度。

目前已经有一些工作进行了探索。例如 Jin 等人 [14] 提出了 **RAGCache**，一种为 RAG 系统设计的多级动态缓存机制，旨在通过缓存检索到的文档的中间状态来减少冗余计算，提高系统效率和吞吐量。**RAGCache** 系统的框架如图 6.27 所示，包括 **RAGCache** 模块、外部知识库以及大语言模型三大模块。接下来我们对 **RAGCache** 模块的各个部分进行简单介绍。

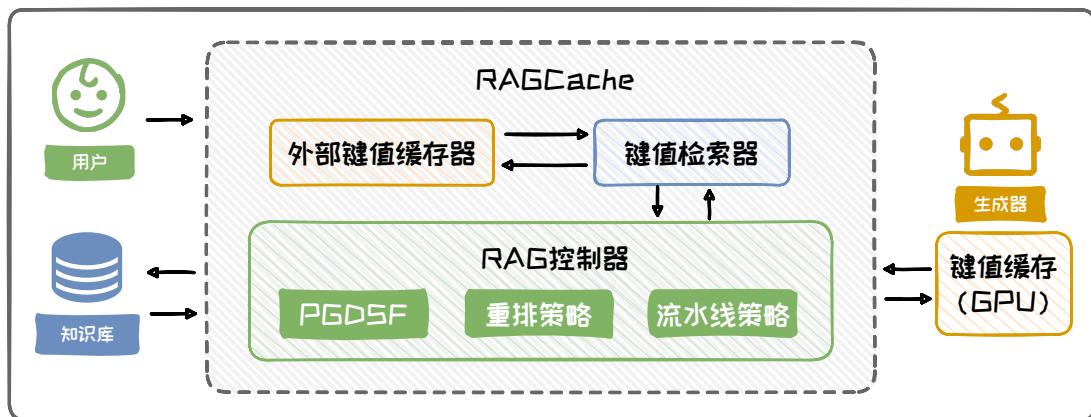


图 6.27: RAGCache 框架流程示意图

**RAGCache** 由键值张量缓存库、缓存检索器与 RAG 控制器三个部分构成，其中键值张量缓存库利用知识树这种数据结构来缓存文档的键值张量，其中每个节点代表一个文档，并根据文档的访问频率动态地在 GPU 内存和主机内存之间分配节点；键值张量缓存库则利用知识树快速检查是否存在相应的缓存节点，确保在模

型能够迅速访问到所需的外部知识；而 **RAG 控制器** 负责核心的缓存策略规划，具体而言，控制器首先采用了一种前缀感知的贪婪双重尺度频率（PGDSF）替换策略，这种策略考虑了节点的访问频率、大小、访问成本和最近访问时间，以最小化缓存未命中率；接下来的**缓存感知重排序策略**则通过重新安排请求的处理顺序来提高缓存命中率，并优化资源使用和系统性能；最后，系统还引入了**动态推测流水线策略**，以重叠知识检索和模型推理步骤，减少端到端延迟。

通过结合上述输入 **Prompt 压缩** 与 **KV-cache 机制**，RAG 框架可以在保持高性能的同时，显著提升其效率。这不仅有助于在资源受限的环境中部署模型，还可以提高模型在实时应用中的响应能力。未来的研究可以进一步探索这些技术的最佳实践，以及它们在不同应用场景下的适用性。

## 6.5 应用与前景

在前面的章节中，我们已经通过：（1）**检索器与生成器如何高效地协作？**（2）**检索器如何高效地进行检索？**（3）**生成器如何高效利用检索增强？**这三个核心问题对 RAG 系统的各个方面展开了讨论。相信各位读者已经对 RAG 系统有了一个较为全面的认识。总体而言，RAG 是大模型时代一个新的产物，它结合了检索和生成两大模块，极大地拓宽了大语言模型的功能边界，显著提升了其处理复杂问题和生成高质量文本的能力。RAG 系统的出现，标志着信息检索技术与大模型能力的深度融合。在接下来的内容中，我们将讨论 RAG 系统在各个领域的应用实例，以及它在未来技术发展中的潜在影响和广阔的发展前景。

### 6.5.1 检索增强生成典型应用

在本节中，我们将介绍 RAG 系统在不同领域的应用实例。具体而言，我们主要从两个方向的应用进行介绍：(1) 智能代理 (Agent)；(2) 多模态垂直领域应用。

#### 1. 智能代理 (Agent)

在前面的章节中，我们已经介绍了 Agent 的基本框架，在这一小节中，我们主要讨论 Agent 中 RAG 系统的使用。相比 RAG 系统只是被动地提供语言模型信息，Agent 系统则更加主动，针对不同的问题，Agent 可以主动决策当前需要采取的行动，例如调取检索模块获得相关信息。而在这个过程中，RAG 技术可以提供强大的支持，通过结合检索到的信息，Agent 能够提供更加准确、全面的答案，满足用户的需求，这不仅优化了用户体验，也使得 Agent 在处理更加复杂的问题时表现得更加出色。

图 6.28 是一个经典的 Agent 框架，主要包含四大部分：配置 (Profile)、记忆 (Memory)、计划 (Planning) 和行动 (Action)。其中记忆模块与行动模块都需要用到 RAG 技术。

具体而言，(1) 配置模块通过设定基本信息来定义智能体的角色，这些信息可以包括智能体的年龄、性别、职业等，以及反映其个性和社交关系的心理和社交信息；(2) 记忆模块存储从环境中学习到的知识以及历史信息，包括记忆检索、记忆更新和记忆反思，这些操作允许智能体不断获取、积累和利用知识。RAG 技术在此模块中通过检索相关信息来支持记忆的读取和更新；(3) 计划模块使智能体能够像人类一样将复杂任务分解为更简单的子任务，并根据记忆和行动反馈不断调整；(4) 行动模块则负责将智能体的计划转化为具体的行动，可以进行网页检索、工具调用以及多模态输出等，这些行动可以改变环境、智能体自身的状态或触发新的行动。RAG 技术在这一模块中通过检索和生成相关信息来辅助智能体的决策

和行动执行。

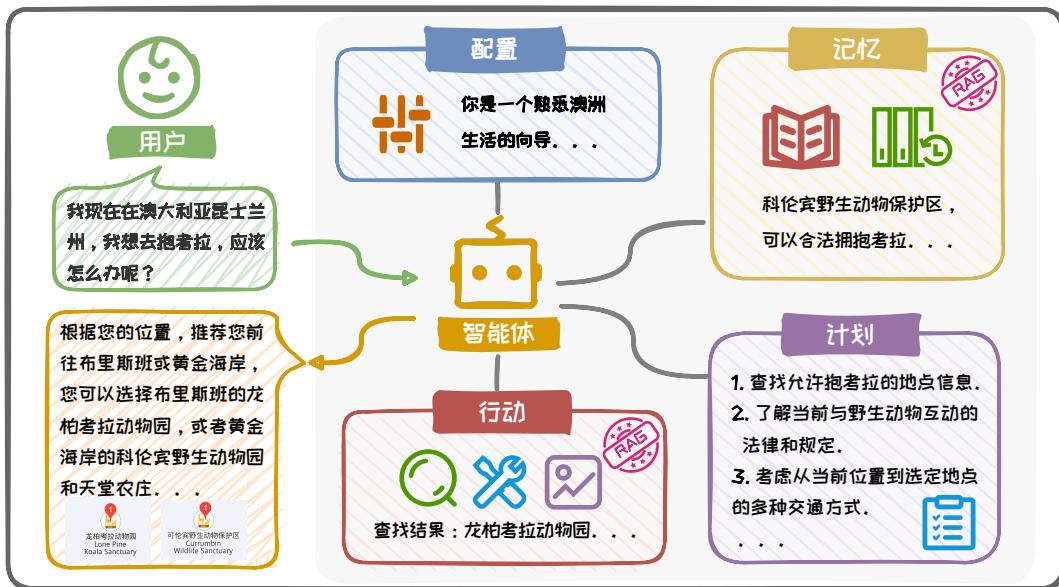


图 6.28: 智能体框架流程示意图

我们可以通过一个例子来更直观地了解 Agent 处理问题的过程，例如用户输入一个问题：“我现在在澳大利亚，我想去抱考拉，应该怎么办呢？”这个例子中，用户提出了一个问题。Agent 会通过以下步骤来完成任务：

- Agent 首先利用其**配置模块**来进行初始化，设定其角色；
- 对于用户输入任务，**计划模块**规划如何帮助用户实现抱考拉的愿望。例如确定用户附近可以抱考拉的地点、当地对于与野生动物互动的法律规定、前往目的地的方式等等；
- 接下来**记忆模块**将根据计划检索有关澳大利亚抱考拉活动的信息，其中对于记忆不包含的内容，将利用**行动模块**从外部知识源中进行检索；
- 在获取相关信息后，Agent 的**计划模块**将帮助它规划如何构建一个有用的回答，包括确定最佳行动方案，如推荐特定的地点以及提供相关建议；
- 最后**行动模块**输出规划建议以及相关的图像、位置等信息。

通过这个例子，我们可以看到 Agent 框架中的每个模块是如何协同工作来完成

一个具体的预测任务，其中 RAG 技术可以起到快速信息检索和知识整合的作用。

### 2. 多模态垂直领域应用

在前面的章节中，我们主要讨论了文本形式的 RAG 系统，而在多模态领域，RAG 系统也展现出了广阔的应用前景。多模态领域涉及到多种类型的数据，例如图像、音频、视频等，这使得任务的处理变得更加复杂和多样化。然而在许多垂直领域中，多模态数据是非常常见的，例如在医疗领域，医学影像数据可以包括 X 光片、MRI 和 CT 扫描等图像数据，同时还有患者的病历文本数据、实时监测的生理参数数据等，这些数据来源多样且具有复杂的关联关系。因此在处理相关任务时，RAG 系统不仅仅要处理文本信息，还需要处理多模态信息。

目前，已经出现了一些多模态垂直领域的 RAG 系统应用，例如 Ossowski 等人在 2023 年提出的医学领域多模态检索框架 [28]。图 6.29 展示了该框架的主要流程，对于一个任务，例如给定一张 X 光照片，询问该照片所对应的可能症状，该框架首先**提取图像和文本的特征表示**，深入理解图像内容和相关问题的语义，为检索模块提供丰富的特征向量。接下来该框架引入了一个**多模态检索模块**，利用这些特征向量在医学知识库中进行检索，从而获取与输入问题最相关的信息，包括相关的医学案例以及相关症状的描述、可能的病因和治疗方法等，这些信息包含了文本和图片格式。接下来 Prompt 构建模块对信息进行整合，来帮助模型生成正确的回答，在本例子中的答案为**心肌肥大**。

不仅仅是医疗领域，RAG 系统在其他垂直领域中也展现了其在处理专业信息方面的强大能力，极大地辅助了专业人士的决策过程。而在不同的任务中，除了图片信息，RAG 系统在音频、视频等其他多模态场景中也有着出色的表现。随着技术的进步和数据资源的丰富，我们期待 RAG 系统在未来能够在更多垂直领域中发挥关键作用，推动多模态人工智能技术的发展和应用。

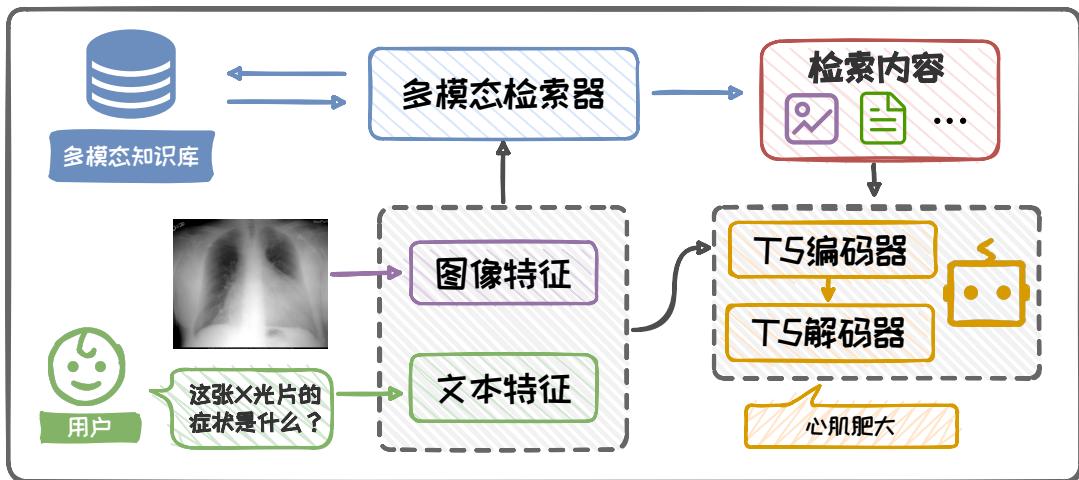


图 6.29: 多模态垂域 RAG 框架示意图

## 6.5.2 检索增强生成工具框架

为了帮助开发者们高效便捷地构建一个 RAG 系统，目前已经有许多较为成熟的开源框架，其中最具代表性的便是 LangChain 与 LlamaIndex。这些开源框架提供了一整套的工具和接口，使得开发者们能够轻松地将 RAG 系统集成到他们的应用中，包括聊天机器人、Agent 等。接下来我们简单介绍一下这两个框架的特点和主要功能。

### (1) LangChain<sup>7</sup>

LangChain 是一个开源框架，目的在于简化利用大语言模型开发应用程序的整个过程。它提供了一系列模块化的组件，帮助开发者部署基于 LLM 的应用，其中包括 RAG 整个框架的构建。LangChain 主要包含六大模块：**Model IO**、**Retrieval**、**Chains**、**Memory**、**Agents** 和 **Callbacks**。其中 Model IO 模块包含了各种大模型的接口以及 Prompt 设计。而 Retrieval 模块则包含了构建 RAG 系统所需要的核心模块，包括文档加载、文本分割、向量构建，索引生成以及向量检索，还提供了非结构化数据库的接口。而 Chains 组件可以将各个模块链接在一起逐个执行，Memory

<sup>7</sup><https://www.langchain.com>

模块则可以存储对话过程中的数据。总体而言，**LangChain** 提供了一个较为全面的模块支持，帮助开发者们轻松便捷地构建 RAG 应用框架。

### (2) **LlamaIndex**<sup>8</sup>

**LlamaIndex** 是一个开源的数据框架，与 **LangChain** 不同，**LlamaIndex** 框架专注于索引与检索部分，可以帮助开发者迅速构建高效的检索系统。**LlamaIndex** 支持从不同格式的数据源中获取数据，包括 API、PDF、SQL 等等。对于获取到的数据，**LlamaIndex** 提供了一套工具来将数据向量化并构建索引。而在查询阶段，**LlamaIndex** 也提供了高效检索工具。此外，在获取到上下文信息后，**LlamaIndex** 还支持对它们的过滤、重排等细化操作。并且，**LlamaIndex** 框架也可以与 **LangChain** 进行融合，实现更丰富的功能。总体而言，**LlamaIndex** 侧重于索引与检索，在查询效率上的表现更为突出，适用于在大数据量的场景下构建高效的 RAG 系统。

### 6.5.3 挑战与未来方向

目前，RAG 技术正在蓬勃发展中，它结合高质量的外部知识源与大语言模型出色的上下文能力，使得模型能够更全面地理解用户需求并弥补自身知识的不足，极大地提升了大语言模型的生成质量，还使得模型在各种场景下更加灵活和有效。然而，作为一项新兴技术，RAG 在各个方面还面临着许多挑战，存在很大的发展空间。在本节中，我们将讨论 RAG 技术目前还需要进一步探索的方向。

#### (1) 检索源的多样性与权威性

当前高质量的检索源如 **Wikipedia** 虽然在通用领域任务上效果良好并被广泛应用，但在专业或新兴领域数据有限，且难以满足高时效性需求。然而，要构建一个全面且高质量的检索源面临着许多问题，例如**数据存储、高效索引构建、虚假有害信息过滤**等各种问题。目前也有一些方法是通过调取搜索引擎获取数据，然而，它

---

<sup>8</sup><https://www.llamaindex.ai>

同样面临着严重的虚假有害信息过滤的问题。此外，由于数据的格式多种多样，如何进行统一处理、划分、存储及构建索引也面临着巨大挑战。

### (2) 检索质量与效率优化

对于检索质量方面，如何在海量数据中筛选出与当前问题相关并且语言模型易理解利用的内容仍然是一个重要问题，低质量的外部数据会损害语言模型的生成性能。在这个问题中会涉及到**文本向量生成、检索算法优化、模型偏好拟合、低质量数据筛除**等挑战。此外，如何高效地从海量数据源中检索相关信息也是一个重要问题，这需要权衡检索的质量与效率。

### (3) 知识表示与整合

在获取了高质量的检索文本之后，如何帮助语言模型更充分地利用这些信息也存在着很大的挑战。为了使检索到的信息能够有效地被语言模型利用，如何对其进行适当的表示和整合是一个重要问题。由于数据的来源与格式多样，这里涉及到**异构数据源整合、知识表示选择以及知识与任务融合方式**等挑战。目前已经有一些工作将**RAG** 技术与微调相结合来提升模型对知识的利用能力，如 Self-RAG [2] 等。

### (4) RAG 框架效率提升

在实际应用中，RAG 框架的效率直接影响到模型的响应速度和可用性。提高 RAG 框架的效率是实现其在实际应用中广泛部署的关键。RAG 框架主要包含检索和生成两个过程，还可能涵盖**问题改写、文档精排**等处理模块，这整个流程可能会消耗大量计算资源。特别是当知识文档较长时，模型生成需要更多时间。此外在一些场景下，我们可能需要**多次使用 RAG 系统**，因此在保证检索和生成质量的同时提高整个流程的效率在实际生产应用中至关重要。

### (5) 自适应 RAG 技术

一方面由于低质量的知识会损害模型生成质量，另一方面考虑到 RAG 框架的

资源消耗，如何使 RAG 技术能够自适应不同的任务和用户需求，是一个值得探索的方向。这其中涉及到 **RAG 系统必要性的判别**以减少不必要的检索增强，**检索策略的动态调整**以优化生成质量以及**反馈机制设计**等方面。

总体而言，RAG 技术在提升语言模型性能方面展现出巨大潜力，但同时也面临着**数据源建设、检索优化、知识表示、框架效率和自适应性**等多方面的挑战。目前已经有许多工作进行了探索并取得了一些成果，未来的研究可以在这些领域进一步深入探索和创新，以实现 RAG 技术的广泛应用和持续进步。

## 参考文献

- [1] Akiko Aizawa. “An information-theoretic perspective of tf–idf measures”. In: *IPM*. 2003.
- [2] Akari Asai et al. “Self-rag: Learning to retrieve, generate, and critique through self-reflection”. In: *arXiv preprint arXiv:2310.11511* (2023).
- [3] Jon Louis Bentley. “Multidimensional binary search trees used for associative searching”. In: *Communications of the ACM* (1975).
- [4] Sebastian Borgeaud et al. “Improving language models by retrieving from trillions of tokens”. In: *ICML*. 2022.
- [5] Sebastian Borgeaud et al. “Improving language models by retrieving from trillions of tokens”. In: *ICML*. 2022.
- [6] Mayur Datar et al. “Locality-sensitive hashing scheme based on p-stable distributions”. In: *SoCG*. 2004.
- [7] Jacob Devlin et al. “Bert: Pre-training of deep bidirectional transformers for language understanding”. In: *arXiv preprint arXiv:1810.04805* (2018).
- [8] Mohamad Dolatshah, Ali Hadian, and Behrouz Minaei-Bidgoli. “Ball\*-tree: Efficient spatial indexing for constrained nearest-neighbor search in metric spaces”. In: *arXiv preprint arXiv:1511.00628* (2015).
- [9] Wei Dong, Charikar Moses, and Kai Li. “Efficient k-nearest neighbor graph construction for generic similarity measures”. In: *WWW*. 2011.

- [10] Samuel Humeau et al. “Poly-encoders: Transformer architectures and pre-training strategies for fast and accurate multi-sentence scoring”. In: *arXiv preprint arXiv:1905.01969* (2019).
- [11] Gautier Izacard et al. “Atlas: Few-shot learning with retrieval augmented language models”. In: *Journal of Machine Learning Research* (2023).
- [12] Huiqiang Jiang et al. “Llmlingua: Compressing prompts for accelerated inference of large language models”. In: *arXiv preprint arXiv:2310.05736* (2023).
- [13] Huiqiang Jiang et al. “Longllmlingua: Accelerating and enhancing llms in long context scenarios via prompt compression”. In: *arXiv preprint arXiv:2310.06839* (2023).
- [14] Chao Jin et al. “RAGCache: Efficient Knowledge Caching for Retrieval-Augmented Generation”. In: *arXiv preprint arXiv:2404.12457* (2024).
- [15] Nikhil Kandpal et al. “Large language models struggle to learn long-tail knowledge”. In: *ICML*. 2023.
- [16] Vladimir Karpukhin et al. “Dense passage retrieval for open-domain question answering”. In: *arXiv preprint arXiv:2004.04906* (2020).
- [17] Omar Khattab and Matei Zaharia. “Colbert: Efficient and effective passage search via contextualized late interaction over bert”. In: *SIGIR*. 2020.
- [18] Omar Khattab et al. “Demonstrate-search-predict: Composing retrieval and language models for knowledge-intensive nlp”. In: *arXiv preprint arXiv:2212.14024* (2022).
- [19] Gangwoo Kim et al. “Tree of clarifications: Answering ambiguous questions with retrieval-augmented large language models”. In: *arXiv preprint arXiv:2310.14696* (2023).
- [20] Patrick Lewis et al. “Retrieval-augmented generation for knowledge-intensive nlp tasks”. In: *NeurIPS*. 2020.
- [21] Yuxin Liang et al. “Learning to trust your feelings: Leveraging self-awareness in llms for hallucination mitigation”. In: *arXiv preprint arXiv:2401.15449* (2024).
- [22] Yu A Malkov and Dmitry A Yashunin. “Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs”. In: *IEEE transactions on pattern analysis and machine intelligence* (2018).
- [23] Alex Mallen et al. “When not to trust language models: Investigating effectiveness of parametric and non-parametric memories”. In: *ACL*. 2023.

- [24] Potsawee Manakul, Adian Liusie, and Mark JF Gales. “Selfcheckgpt: Zero-resource black-box hallucination detection for generative large language models”. In: *arXiv preprint arXiv:2303.08896* (2023).
- [25] Yuren Mao et al. “FIT-RAG: Black-Box RAG with Factual Information and Token Reduction”. In: *arXiv preprint arXiv:2403.14374* (2024).
- [26] Yuren Mao et al. “FIT-RAG: Black-Box RAG with Factual Information and Token Reduction”. In: *arXiv preprint arXiv:2403.14374* (2024).
- [27] Kevin Meng et al. “Locating and editing factual associations in GPT”. In: *NeurIPS*. 2022.
- [28] Timothy Ossowski and Junjie Hu. “Multimodal prompt retrieval for generative visual question answering”. In: *arXiv preprint arXiv:2306.17675* (2023).
- [29] Ella Rabinovich et al. “Predicting Question-Answering Performance of Large Language Models through Semantic Consistency”. In: *arXiv preprint arXiv:2311.01152* (2023).
- [30] Ori Ram et al. “In-context retrieval-augmented language models”. In: *Transactions of the Association for Computational Linguistics* (2023).
- [31] Ruiyang Ren et al. “Investigating the factual knowledge boundary of large language models with retrieval augmentation”. In: *arXiv preprint arXiv:2307.11019* (2023).
- [32] Stephen Robertson, Hugo Zaragoza, et al. “The probabilistic relevance framework: BM25 and beyond”. In: *Foundations and Trends® in Information Retrieval* (2009).
- [33] Weijia Shi et al. “Replug: Retrieval-augmented black-box language models”. In: *arXiv preprint arXiv:2301.12652* (2023).
- [34] Weiwei Sun et al. “Is chatgpt good at search? investigating large language models as re-ranking agent”. In: *arXiv preprint arXiv:2304.09542* (2023).
- [35] Jason Wei et al. “Chain of Thought Prompting Elicits Reasoning in Large Language Models”. In: *NeurIPS*. 2022.
- [36] Mingrui Wu and Sheng Cao. “LLM-Augmented Retrieval: Enhancing Retrieval Models Through Language Models and Doc-Level Embedding”. In: *arXiv preprint arXiv:2404.05825* (2024).
- [37] Fangyuan Xu, Weijia Shi, and Eunsol Choi. “Recomp: Improving retrieval-augmented lms with compression and selective augmentation”. In: *arXiv preprint arXiv:2310.04408* (2023).

- [38] Haoyan Yang et al. “Prca: Fitting black-box large language models for retrieval question answering via pluggable reward-driven contextual adapter”. In: *arXiv preprint arXiv:2310.18347* (2023).
- [39] Wenhao Yu et al. “Generate rather than retrieve: Large language models are strong context generators”. In: *ICLR*. 2023.
- [40] Wenhao Yu et al. “Improving Language Models via Plug-and-Play Retrieval Feedback”. In: *arXiv preprint arXiv:2305.14002* (2023).
- [41] Yin Zhang, Rong Jin, and Zhi-Hua Zhou. “Understanding bag-of-words model: a statistical framework”. In: *ICML*. 2010.

