

5
2
0
2

n
u
j

5
1

]
L
C
.
s
c
[

2
v
3
2
2
9
0
.
1
0
5
2
:
v
i
X
r
a

大型语言模型基础

童晓 和 朱静波

2025年6月17日

NLP Lab, Northeastern University & NiuTrans Research

This book is a selection of chapters from an introductory NLP resource
available at <https://github.com/NiuTrans/NLPBook>

版权所有 © 2021-2025 汤晓 和 朱静波

NATURAL LANGUAGE PROCESSING LAB, NORTHEASTERN UNIVERSITY
&
NIUTRANS RESEARCH

授权许可下Creative Commons署名-非商业性4.0国际许可协议（以下简称“许可”）。您不得使用此文件，除非遵守许可。您可以在
<http://creativecommons.org/licenses/by-nc/4.0>获取许可副本。除非适用法律要求或书面同意，否则在许可下分发的软件按“现状”基础分发，不提供任何形式的明示或暗示保证。有关许可下权限和限制的具体语言，请参阅许可。

June 17, 2025

序言

大型语言模型起源于自然语言处理，但无疑已成为近年来人工智能领域最革命性的技术进步之一。大型语言模型带来的一个重要见解是，通过大规模语言建模任务可以获得世界和语言的知识，这样我们就可以创建一个处理各种问题的通用模型。这一发现对自然语言处理以及许多相关学科的研究方法产生了深远影响。我们已从使用大量标记数据从头开始训练专用系统转变为使用大规模预训练来获得基础模型的新范式，这些基础模型随后进行微调、对齐和提示。

这本书旨在概述大型语言模型的基本概念并介绍相关技术。正如标题所示，本书更侧重于大型语言模型的基础方面，而不是全面覆盖所有前沿方法。本书共分为五章：

- 第一章介绍了预训练的基础。这是大型语言模型的基础，这里将讨论常见的预训练方法和模型架构。
- 第二章介绍了生成模型，这是我们今天通常所指的大型语言模型。在介绍构建这些模型的基本过程之后，我们还将探讨如何扩大模型训练规模并处理长文本。
- 第三章介绍了大型语言模型的提示方法。我们将讨论各种提示策略，以及更高级的方法，如思维链推理和自动提示设计。
- 第四章介绍了大型语言模型的对齐方法。我们将重点关注基于指令的微调和基于人类反馈的对齐。
- 第五章介绍了大型语言模型的推理方法。我们将讨论各种解码算法、加速方法和推理时间缩放问题。

如果读者在机器学习和自然语言处理方面有一些背景知识，并且对像Transformer这样的神经网络有一定的了解，阅读这本书将会非常容易。然而，即使没有这些先前的知识，这也完全没问题，因为我们已经尽可能使每一章的内容具有独立性，确保读者不会因为阅读难度过大而感到负担。

这里展示的内容是关于自然语言处理中神经网络和大语言模型的全面入门资源的部分。对于希望了解更多背景主题，如序列建模和注意力机制的读者，您可以访问 <https://github.com/NiuTrans/NLPBook> 或 <https://niutrans.github.io/NLPBook> 以获取更多信息。

我们想感谢我们实验室的学生和所有与我们分享他们对大型语言模型观点并帮助我们纠正写作错误的朋友们。特别是，我们想感谢山维乔、穆永宇、王成龙、常凯燕、范宇春、周航、吕传豪、刘欣宇、周涛、包慧文、郑桐、阮俊豪、罗英锋、吴宇章和火一夫。

Notation

a	variable
\mathbf{a}	row vector or matrix
$f(a)$	function of a
$\max f(a)$	maximum value of $f(a)$
$\arg \max_a f(a)$	value of a that maximizes $f(a)$
\mathbf{x}	input token sequence to a model
x_j	input token at position j
\mathbf{y}	output token sequence produced by a model
y_i	output token at position i
θ	model parameters
$\Pr(a)$	probability of a
$\Pr(a b)$	conditional probability of a given b
$\Pr(\cdot b)$	probability distribution of a variable given b
$\Pr_\theta(a)$	probability of a as parameterized by θ
\mathbf{h}_t	hidden state at time step t in sequential models
\mathbf{H}	matrix of all hidden states over time in a sequence
$\mathbf{Q}, \mathbf{K}, \mathbf{V}$	query, key, and value matrices in attention mechanisms
$\text{Softmax}(\mathbf{A})$	Softmax function that normalizes the input vector or matrix \mathbf{A}
\mathcal{L}	loss function
\mathcal{D}	dataset used for training or fine-tuning a model
$\frac{\partial \mathcal{L}}{\partial \theta}$	gradient of the loss function \mathcal{L} with respect to the parameters θ
$\text{KL}(p \parallel q)$	KL divergence between distributions p and q

内容

2 生成模型 36 2.1 对大型语言模型（LLMs）的简要介绍

2.3.5 位置外推和内插	
2.4 摘要	94
3 提示	96
3.1 通用提示设计	
3.1.1 基础知识	
3.2 高级提示方法	
3.2.1 思维链	
3.3 学习提示	138
3.3.1 提示优化	
3.4 摘要	153
4 对齐	155
4.1 LLM 对齐概述	
4.3 人类偏好对齐: RLHF	
4.3.1 强化学习基础	
4.4 改进的人性化偏好对齐	
4.4.1 更好的奖励建模	

CHAPTER 1

预训练

神经序列模型（如Transformers [Vaswani et al., 2017]）的发展以及大规模自监督学习的改进，为通用语言理解和生成打开了大门。这一成就很大程度上得益于预训练：我们将许多基于神经网络的系统的公共组件分离出来，然后使用自监督在大量未标记数据上对它们进行训练。这些预训练模型作为基础模型，可以通过微调或提示轻松适应不同的任务。因此，NLP的范式发生了巨大变化。在许多情况下，对于特定任务的大规模监督学习不再需要，我们只需要适应预训练的基础模型。

虽然预训练在最近的NLP研究中获得了流行，但这一概念可以追溯到深度学习的早期。例如，早期尝试预训练深度学习系统包括RNN的无监督学习、深度前馈网络、自动编码器等[Schmidhuber, 2015]。在深度学习的现代时代，我们经历了预训练的复兴，部分原因是各种词嵌入模型的大规模无监督学习[Mikolov等人, 2013b; Pennington等人, 2014]。在同一时期，预训练也引起了计算机视觉领域的广泛关注，其中骨干模型在相对较大的标记数据集（如ImageNet）上进行训练，然后应用于不同的下游任务[He等人, 2019; Zoph等人, 2020]。NLP中预训练的大规模研究始于使用自监督学习开发语言模型。这一系列模型包括几个著名的例子，如BERT[Devlin等人, 2019]和GPT[Brown等人, 2020]，它们都有一个类似的想法，即通过训练模型预测大量文本中的掩码词来实现通用语言理解和生成。尽管这种方法本质简单，但生成的模型在建模语言结构方面表现出非凡的能力，尽管它们并没有明确训练来实现这一点。预训练任务的通用性导致系统在大量NLP问题中表现出强大的性能，甚至超过了之前开发良好的监督系统。最近，预训练的大型语言模型取得了更大的成功，展示了更通用的人工智能的激动人心的前景[Bubeck等人, 2023]。

本章讨论了NLP中的预训练概念。它从对预训练方法和其应用的概述开始。然后以BERT为例，说明如何通过称为掩码语言模型的自我监督任务来训练序列模型。接着讨论了将预训练序列模型适应各种NLP任务的方法。请注意，在本章中，我们将主要关注NLP中的预训练范式，因此，我们无意涵盖关于生成大型语言模型的细节。对这些模型的详细讨论将留待后续章节。

1.1 预训练 NLP 模型

NLP中关于预训练问题的讨论通常涉及两种类型的问题：序列建模（或序列编码）和序列生成。虽然这些问题有不同的

表单，为了简化city，我们使用单个模型定义f来描述它们

ollows: 依次:

$$\begin{aligned} \mathbf{o} &= g(x_0, x_1, \dots, x_m; \theta) \\ &= g_\theta(x_0, x_1, \dots, x_m) \end{aligned} \quad (1.1)$$

$\{x_0, x_1, \dots, x_m\}$ 表示输入标记序列¹， x_0 表示附加到序列开头的特殊符号 ($\langle s \rangle$ 或[CLS])， $g(\cdot; \theta)$ (也写作 $g_\theta(\cdot)$)表示具有参数 θ 的神经网络， \mathbf{o} 表示神经网络的输出。不同问题可以根据输出 \mathbf{o} 的形式而有所不同。例如，在标记预测问题（如语言模型）中， \mathbf{o} 是词汇表上的分布；在序列编码问题中， \mathbf{o} 是输入序列的表示，通常表示为实值向量序列。

这里有两个基本问题。

- 优化 θ 在预训练任务上。与 NLP 中的标准学习问题不同，预训练不假设模型将应用于特定的下游任务。相反，目标是训练一个可以在各种任务中泛化的模型。
- 将预训练模型 $g_\theta(\cdot)$ 应用到下游任务中。为了使模型适应这些任务，我们需要使用标记数据略微调整参数 $\hat{\theta}$ 或通过任务描述提示模型。

在这s部分，我们讨论解决这些问题的基本思想

问题。

1.1.1 无监督、监督和自监督预训练

在深度学习中，预训练是指在进一步训练/调整并应用于感兴趣的任务之前优化神经网络的过程。这种方法基于一个假设，即在一个任务上预训练的模型可以适应执行另一个任务。因此，我们不需要从头开始训练一个深度、复杂的神经网络来处理具有有限标记数据的任务。相反，我们可以利用监督信号更容易获得的任务。这减少了对于特定任务标记数据的依赖，使得可以开发出不受特定问题限制的更通用模型。

在深度学习推动神经网络复兴期间，许多早期尝试进行预训练的努力都集中在无监督学习上。在这些方法中，神经网络参数是通过与特定任务无直接关系的标准进行优化的。例如，我们可以最小化每个层的输入向量的重建交叉熵[Bengio等人，2006]。无监督预训练通常作为监督学习之前的初步步骤，提供了几个优点，例如帮助发现更好的局部最小值，并在训练过程中添加正则化效果[Erhan等人，2010]。这些好处使得随后的监督学习阶段更容易且更稳定。

第二种预训练方法是先在监督学习任务上预训练一个神经网络。例如，考虑一个序列模型，该模型旨在将输入序列编码为某些

¹Here we assume that tokens are basic units of text that are separated through tokenization. Sometimes, we will use the terms *token* and *word* interchangeably, though they have closely related but slightly different meanings in NLP.

表示。在预训练中，此模型与分类层结合形成分类系统。然后，该系统在预训练任务上进行训练，例如根据情感对句子进行分类（例如，确定句子是否传达积极或消极的情感）。然后，我们将序列模型调整为下游任务。我们基于这个预训练的序列模型和新的分类层（例如，确定序列是主观的还是客观的）构建一个新的分类系统。通常，我们需要使用特定任务的标记数据来微调新模型的参数，确保模型能够最佳地调整以在新类型的数据上表现良好。微调后的模型随后用于对任务的新序列进行分类。监督预训练的优点是，无论是在预训练阶段还是在微调阶段，训练过程都很简单，因为它遵循了机器学习中广泛研究的监督学习通用范式。然而，随着神经网络复杂性的增加，对更多标记数据的需求也增加。这反过来使得预训练任务更加困难，尤其是在大规模标记数据不可用的情况下。

第三种预训练方法是自监督学习。在这种方法中，神经网络通过使用自身生成的监督信号进行训练，而不是人类提供的信号。这通常是通过直接从未标记数据构建自己的训练任务来完成的，例如让系统创建伪标签。虽然自监督学习最近在NLP中成为了一种非常流行的方法，但它并不是一个新概念。在机器学习中，与之相关的一个概念是自训练，其中模型通过从分配给数据集的伪标签中学习来迭代改进。为此，我们需要一些种子数据来构建初始模型。然后，该模型为未标记数据生成伪标签，这些伪标签随后用于迭代优化和启动模型本身。这种方法已在多个NLP领域成功应用，例如词义消歧[Yarowsky, 1995]和文档分类[Blum and Mitchell, 1998]。与标准自训练方法不同，NLP中的自监督预训练不依赖于初始模型来标注数据。相反，所有监督信号都来自文本，整个模型从头开始训练。一个著名的例子是通过逐个预测文本中给定其先前或周围单词的掩码词来训练序列模型。这使大规模自监督学习成为深度神经网络的可能，从而在许多理解、写作和推理任务中实现了预训练的成功。

图1.1显示了上述三种预训练方法的比较。自监督预训练非常成功，以至于大多数当前最先进的NLP模型都是基于这种范式。因此，在本章以及整本书中，我们将重点关注自监督预训练。我们将展示如何通过自监督对序列模型进行预训练以及如何应用预训练模型。

1.1.2 预训练模型的适配

如上所述，两种主要类型的模型在NLP预训练中被广泛使用。

- 序列编码模型。给定一个单词或标记序列，序列编码模型将此序列表示为实值向量或向量序列，并获得序列的表示。这种表示通常用作其他模型的输入，例如句子分类系统。

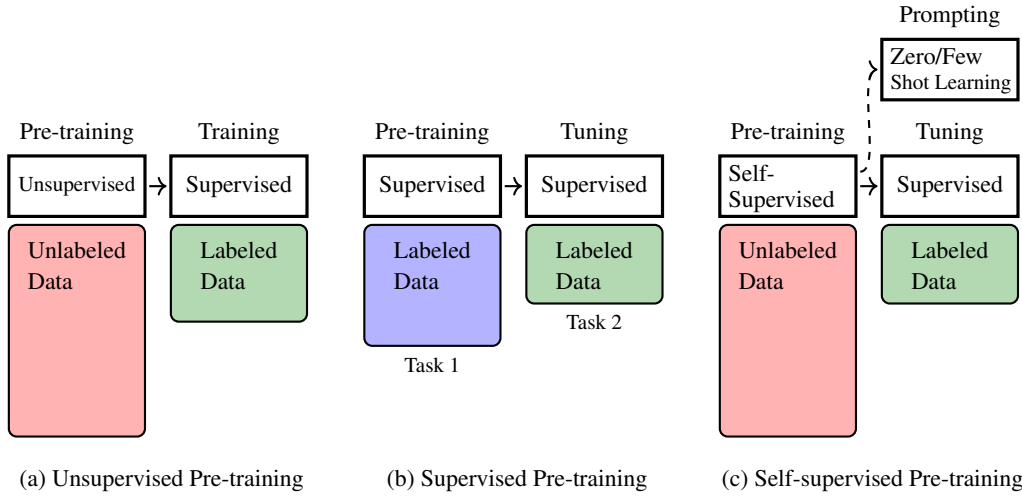


图1.1: 无监督、监督和自监督预训练的说明。在无监督预训练中, 预训练是在大规模未标记数据上进行的。它可以看作是后续优化过程的初步步骤, 尽管在预训练后仍需要相当大的努力来进一步使用标记数据训练模型。在监督预训练中, 基本假设是不同的(监督)学习任务相关。因此, 我们可以首先在一个任务上训练模型, 然后通过一些训练或调整努力将得到的模型转移到另一个任务。在自监督预训练中, 通过自监督在大规模未标记数据上对模型进行预训练。通过这种方式, 模型可以很好地训练, 我们可以通过微调或提示有效地将其适应新任务。

- 序列生成模型。在自然语言处理中, 序列生成通常指的是根据给定上下文生成一系列标记的问题。术语 *context* 在不同应用中具有不同的含义。例如, 它指的是语言模型中的前一个标记, 在机器翻译中则指的是源语言序列²。

我们需要在预训练后使用不同的技术将这些模型应用于下游任务。这里我们感兴趣的是以下两种方法。

1.1.2.1 预训练模型的微调

对于序列编码预训练, 一个常见的适应预训练模型的方法是微调。令 $\text{Encode}_\theta(\cdot)$ 表示具有参数 θ 的编码器, 例如, $\text{Encode}_\theta(\cdot)$ 可以是一个标准的Transformer编码器。假设我们已经以某种方式预训练了这个模型并获得了最优参数 $\hat{\theta}$, 我们可以用它来建模任何序列并生成相应的表示, 如下所示

$$\mathbf{H} = \text{Encode}_{\hat{\theta}}(\mathbf{x}) \quad (1.2)$$

在 \mathbf{x} 是输入序列 $\{x_0, x_1, \dots, x_m\}$, 而 \mathbf{H} 是输出表示, 它是一个由实值向量 $\{h_0, h_1, \dots, h_m\}$ 组成的序列。因为编码器不作为一个独立的NLP系统, 它通常作为一个组件集成到一个更大的系统中。例如, 考虑一个文本分类问题, 其中我们识别极性(即, 正面、负面、)。

²More precisely, in auto-regressive decoding of machine translation, each target-language token is generated based on both its preceding tokens and source-language sequence.

文本的中立性)。我们可以通过在编码器之上堆叠一个分类器来构建一个文本分类系统。设 $\text{Classify}_\omega(\cdot)$ 为一个具有参数 ω 的神经网络。那么，文本分类模型可以表示为

$$\begin{aligned}\Pr_{\omega, \hat{\theta}}(\cdot | \mathbf{x}) &= \text{Classify}_\omega(\mathbf{H}) \\ &= \text{Classify}_\omega(\text{Encode}_{\hat{\theta}}(\mathbf{x}))\end{aligned}\quad (1.3)$$

这里 $\Pr_{\omega, \hat{\theta}}(\cdot | \mathbf{x})$ 是一个关于标签集 $\{\text{positive}, \text{negative}, \text{neutral}\}$ 的概率分布，并且在这个分布中概率最高的标签被选为输出。为了保持符号简洁，我们将使用 $F_{\omega, \hat{\theta}}(\cdot)$ 来表示 $\text{Classify}_\omega(\text{Encode}_{\hat{\theta}}(\cdot))$ 。

因为模型参数 ω 和 $\hat{\theta}$ 未针对分类任务进行优化，我们无法直接使用此模型。相反，我们必须使用针对任务进行适配的模型修改版本。一种典型的方法是通过在下游任务中提供显式标记来微调模型。我们可以在标记数据集上训练 $F_{\omega, \hat{\theta}}(\cdot)$ ，将其视为一个常见的监督学习任务。微调的结果是进一步优化的参数 $\tilde{\omega}$ 和 $\tilde{\theta}$ 。或者，我们可以冻结编码器参数 $\hat{\theta}$ 以保持其预训练状态，并专注于优化 ω 。这允许分类器有效地与预训练编码器协同工作。

一旦我们获得了一个微调模型，我们就可以用它来对新文本进行分类。例如，假设我们可以在一个旅游网站上看到了一条评论：

我喜欢这里的食物。太棒了！

我们首先将此文本标记为标记³，然后将标记序列 \mathbf{x}_{new} 输入到微调模型 $F_{\tilde{\omega}, \tilde{\theta}}(\cdot)$ 。该模型通过以下方式生成类别的分布：

$$F_{\tilde{\omega}, \tilde{\theta}}(\mathbf{x}_{\text{new}}) = \left[\Pr(\text{positive} | \mathbf{x}_{\text{new}}) \quad \Pr(\text{negative} | \mathbf{x}_{\text{new}}) \quad \Pr(\text{neutral} | \mathbf{x}_{\text{new}}) \right] \quad (1.4)$$

我们选择具有最大值的条目标签作为输出。在这个例子中它是正的。

通常，用于微调的有标签数据量相对于预训练数据量较小，因此微调的计算成本较低。这使得在实践中对预训练模型的适应非常高效：给定一个预训练模型和一个下游任务，我们只需要收集一些有标签数据，并在这些数据上略微调整模型参数。关于微调的更详细讨论可以在第1.4节中找到。

1.1.2.2 预训练模型的提示

与序列编码模型不同，序列生成模型通常独立用于解决语言生成问题，如问答和机器翻译，无需额外模块。因此，对这些模型进行微调很简单

³The text can be tokenized in many different ways. One of the simplest is to segment the text into English words and punctuations $\{\text{I}, \text{love}, \text{the}, \text{food}, \text{here}, \text{.}, \text{It}, \text{'s}, \text{amazing}, \text{!}\}$

作为下游任务的完整系统。例如，我们可以在一些双语数据上微调一个预训练的编码器-解码器多语言模型，以提高其在特定翻译任务上的性能。

在各种序列生成模型中，一个显著的例子是在大量数据上训练的大型语言模型。这些语言模型被训练来简单地预测给定其前一个标记的下一个标记。尽管标记预测是一项如此简单的任务，它长期以来一直局限于“语言建模”，但人们发现，通过重复执行大量任务，它可以实现语言的一般知识的学习。结果是，预训练的大型语言模型在标记预测方面表现出惊人的能力，这使得通过提示大型语言模型将许多NLP问题转化为简单的文本生成问题成为可能。例如，我们可以将上述文本分类问题构造成一个文本生成任务

我爱这里的食物。太棒了！我 _____

这里表示我们想要预测的词或短语（称之为补全）。如果预测的词是 *happy*，或 *glad*，或 *satisfied* 或相关积极词汇，我们可以将文本分类为积极。这个例子展示了一种简单的提示方法，即我们将输入文本与 *I'm* 连接以形成提示。然后，补全帮助决定分配给原始文本的哪个标签。

考虑到大型语言模型在语言理解和生成方面的强大性能，提示可以指导模型执行更复杂的任务。这里是一个提示，我们提示LLM执行带有指令的极性分类。

假设一个文本的极性是从{积极，消极，中立}中选择一个标签。识别输入的极性。

我非常喜欢这里的食物。太棒了！

极性： _____

前两句是对任务的描述。输入和极性分别表示输入和输出。我们期望模型完成文本的同时给出正确的极性标签。通过使用基于指令的提示，我们可以将大型语言模型适应于解决NLP问题，而无需额外的训练。

此示例还展示了大型语言模型的零样本学习能力，它们可以执行在训练阶段未观察到的任务。在神经网络中启用新能力的另一种方法是少样本学习。这通常通过情境学习（IC T）实现。更具体地说，我们添加了一些示例，以展示输入如何对应输出。这些称为演示的示例用于教授大型语言模型如何执行任务。以下是一个涉及演示的示例

假设一个文本的极性是从{积极, 消极, 中立}中选择一个标签。识别输入的极性。

输入：高峰时段交通状况极差，难以按时到达机场。

极性：负

输入：这里的天气很棒。

极性：积极

我非常喜欢这里的食物。太棒了！

极性：_____

提示和上下文学习在大型语言模型近期兴起中扮演着重要角色。我们将在第3章中更深入地讨论这些问题。然而，值得注意的是，虽然提示是一种强大的适应大型语言模型的方法，但仍需要一些调整努力以确保模型可以准确遵循指令。此外，微调过程对于使这些模型的价值与人类价值保持一致至关重要。有关微调的更详细讨论可以在第4章中找到。

1.2 自监督预训练任务

本节中，我们考虑不同神经架构的自监督预训练方法，包括仅解码器、仅编码器和编码器-解码器架构。我们限制讨论范围到Transformer，因为它们构成了大多数NLP预训练模型的基础。然而，预训练是一个广泛的概念，所以我们只简要介绍基本方法，以使本节简洁。

1.2.1 仅解码器预训练

解码器仅架构已被广泛应用于开发语言模型 [Radford 等人, 2018]。例如，我们可以通过简单地从中移除交叉注意力子层来将 Transformer 解码器用作语言模型。此类模型根据其前面的标记预测给定位置的标记分布，输出是概率最大的标记。训练此模型的标准方法，如在语言建模问题中，是在标记序列集合上最小化损失函数。令 $\text{Decoder}_\theta(\cdot)$ 表示具有参数 θ 的解码器。在每个位置 i ，解码器根据其前面的标记 $\{x_0, \dots, x_i\}$ 生成下一个标记的分布，表示为 $\text{Pr}_\theta(\cdot | x_0, \dots, x_i)$ (或简称为 \mathbf{p}_{i+1}^θ)。假设我们在同一位置有黄金标准分布，表示为 $\mathbf{p}_{i+1}^{\text{gold}}$ 。对于语言建模，我们可以将 $\mathbf{p}_{i+1}^{\text{gold}}$ 视为正确预测单词的一热表示。然后我们定义一个损失函数 $\mathcal{L}(\mathbf{p}_{i+1}^\theta, \mathbf{p}_{i+1}^{\text{gold}})$ 来衡量模型预测与真实预测之间的差异。在 NLP 中，通常使用对数尺度交叉熵损失。

给定一个由 m 个标记 $\{x_0, \dots, x_m\}$ 组成的序列，该序列的损失是该损失的加和

在位置 $\{0, \dots, m-1\}$ 上, 由

$$\begin{aligned} \text{Loss}_\theta(x_0, \dots, x_m) &= \sum_{i=0}^{m-1} \mathcal{L}(\mathbf{p}_{i+1}^\theta, \mathbf{p}_{i+1}^{\text{gold}}) \\ &= \sum_{i=0}^{m-1} \text{LogCrossEntropy}(\mathbf{p}_{i+1}^\theta, \mathbf{p}_{i+1}^{\text{gold}}) \end{aligned} \quad (1.5)$$

在 $\text{LogCrossEntropy}(\cdot)$ 是对数尺度交叉熵, 且 $\mathbf{p}_{i+1}^{\text{gold}}$ 是 x_{i+1} 的单热表示。

此损失函数可以扩展到序列集 \mathcal{D} 。在这种情况下, 预训练的目标是找到最佳参数以最小化 \mathcal{D} 上的损失

$$\hat{\theta} = \arg \min_{\theta} \sum_{\mathbf{x} \in \mathcal{D}} \text{Loss}_\theta(\mathbf{x}) \quad (1.6)$$

请注意, 此目标在数学上等同于最大似然估计, 可以重新表述为

$$\begin{aligned} \hat{\theta} &= \arg \max_{\theta} \sum_{\mathbf{x} \in \mathcal{D}} \log \Pr_\theta(\mathbf{x}) \\ &= \arg \max_{\theta} \sum_{\mathbf{x} \in \mathcal{D}} \sum_{i=0}^{i-1} \log \Pr_\theta(x_{i+1} | x_0, \dots, x_i) \end{aligned} \quad (1.7)$$

使用这些优化参数 $\hat{\theta}$, 我们可以使用预训练的语言模型 $\text{Decoder}_{\hat{\theta}}(\cdot)$ 计算给定序列每个位置的概率 $\Pr_{\hat{\theta}}(x_{i+1} | x_0, \dots, x_i)$ 。

1.2.2 仅编码器预训练

如第1.1.2.1节所述, 编码器 $\text{Encoder}_\theta(\cdot)$ 是一个函数, 它读取一个标记序列 $\mathbf{x} = x_0 \dots x_m$ 并生成一个向量序列 $\mathbf{H} = \mathbf{h}_0 \dots \mathbf{h}_m$ ⁴。训练此模型并不简单, 因为我们没有用于衡量实值函数输出好坏的黄金标准数据。编码器预训练的典型方法是将编码器与一些输出层结合, 以接收更容易获得的管理信号。图1.2显示了预训练Transformer编码器的常见架构, 我们在Transformer编码器之上添加了一个Softmax层。显然, 此架构与基于解码器的语言模型相同, 输出是一个概率分布序列

$$\begin{bmatrix} \mathbf{p}_1^{\mathbf{W}, \theta} \\ \vdots \\ \mathbf{p}_m^{\mathbf{W}, \theta} \end{bmatrix} = \text{Softmax}_{\mathbf{W}}(\text{Encoder}_\theta(\mathbf{x})) \quad (1.9)$$

⁴如果我们把 $\mathbf{h}\{\mathbf{v}^*\}$ 视为一个行向量, \mathbf{H} 可以写成

$$\mathbf{H} = \begin{bmatrix} \mathbf{h}_0 \\ \vdots \\ \mathbf{h}_m \end{bmatrix} \quad (1.8)$$

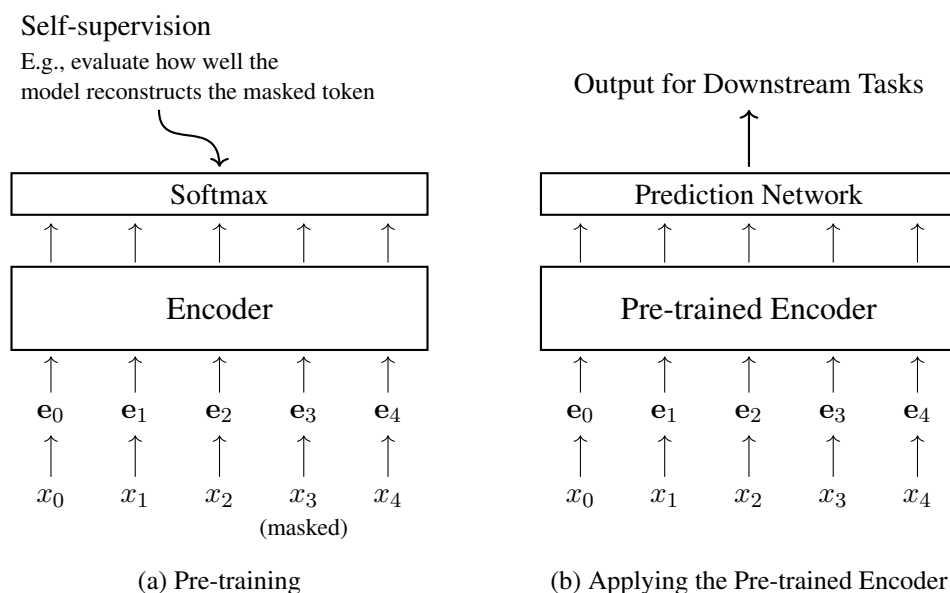


图1.2: 预训练一个Transformer编码器（左侧）然后应用预训练的编码器（右侧）。在预训练阶段，编码器与Softmax层一起通过自监督进行训练。在应用阶段，移除Softmax层，将预训练的编码器与预测网络结合以解决特定问题。通常，为了更好地适应这些任务，系统使用标记数据进行微调。

这里 $p_i^{\mathbf{W}, \theta}$ 是位置 i 的输出分布 $\Pr(\cdot | \mathbf{x})$ 。我们使用 $\text{Softmax}_{\mathbf{W}}(\cdot)$ 来表示 Softmax 层由 \mathbf{W} 参数化，即 $\text{Softmax}_{\mathbf{W}}(\mathbf{H}) = \text{Softmax}(\mathbf{H} \cdot \mathbf{W})$ 。为了简化符号，我们有时会省略附加到每个概率分布上的上标 \mathbf{W} 和 θ 。

这个模型与标准语言模型之间的区别在于，输出 p_i 在编码器预训练和语言建模中有不同的含义。在语言建模中， p_i 是预测下一个单词的概率分布。这遵循自回归解码过程：语言模型只观察位置 i 以内的单词并预测下一个。相比之下，在编码器预训练中，可以一次性观察到整个序列，因此预测这个序列中的任何标记都没有意义。

1.2.2.1 遮蔽语言建模

掩码语言模型是编码器预训练中最受欢迎的方法之一，它是著名的BERT模型[Devlin等人, 2019]的基础。掩码语言模型的想法是通过掩盖输入序列中的一些标记来创建预测挑战，并训练一个模型来预测这些掩盖的标记。从这个意义上说，传统的语言模型问题，有时称为因果语言模型，是掩码语言模型的一个特例：在每个位置，我们掩盖右上下文中的标记，并使用其左上下文来预测这个位置的标记。然而，在因果语言模型中，我们只利用左上下文进行词预测，而预测可能依赖于右上下文中的标记。相比之下，在掩码语言模型中，所有未掩盖的标记都用于词预测，导致一个双向模型，该模型基于左右上下文进行预测。

更正式地说，对于一个输入序列 $\mathbf{x} = x_0 \dots x_m$ ，假设我们在位置 $\mathcal{A}(\mathbf{x}) = \{i_1, \dots, i_u\}$ 上对标记进行掩码。因此，我们获得一个掩码标记序列 $\bar{\mathbf{x}}$ ，其中 $\mathcal{A}(\mathbf{x})$ 中每个位置的标记被替换为特殊符号

The early bird catches the worm

我们可能有一个这样的掩码标记序列

The [MASK] bird catches the [MASK]

我们在其中掩码标记 *early* 和 *worm* (，即 $i_1 = 2$ 和 $i_2 = 6$)。

现在我们有两个序列 \mathbf{x} 和 $\bar{\mathbf{x}}$ 。然后模型被优化，以便我们可以根据 $\bar{\mathbf{x}}$ 正确预测 \mathbf{x} 。这可以被视为一种类似于自编码的过程，训练目标是最大化重建概率 $\Pr(\mathbf{x}|\bar{\mathbf{x}})$ 。请注意， \mathbf{x} 和 $\bar{\mathbf{x}}$ 之间存在简单的位置对齐。因为 $\bar{\mathbf{x}}$ 中的未掩码标记与 \mathbf{x} 中相同位置的标记相同，所以不需要考虑这个未掩码标记的预测。这导致了一个简化的训练目标，它只最大化掩码标记的概率。我们可以用最大似然估计的方式来表达这个目标

$$(\hat{\mathbf{W}}, \hat{\theta}) = \arg \max_{\mathbf{W}, \theta} \sum_{\mathbf{x} \in \mathcal{D}} \sum_{i \in \mathcal{A}(\mathbf{x})} \log \Pr_i^{\mathbf{W}, \theta}(x_i | \bar{\mathbf{x}}) \quad (1.10)$$

或者用交叉熵损失来表示它

$$(\hat{\mathbf{W}}, \hat{\theta}) = \arg \min_{\mathbf{W}, \theta} \sum_{\mathbf{x} \in \mathcal{D}} \sum_{i \in \mathcal{A}(\mathbf{x})} \text{LogCrossEntropy}(\mathbf{p}_i^{\mathbf{W}, \theta}, \mathbf{p}_i^{\text{gold}}) \quad (1.11)$$

在位置 k 给定损坏的输入 $\bar{\mathbf{x}}$ 的真实标记 x_k 的概率 $\Pr_k^{\mathbf{W}, \theta}(x_k | \bar{\mathbf{x}})$ 是，以及 $\mathbf{p}_k^{\mathbf{W}, \theta}$ 是给定损坏的输入 $\bar{\mathbf{x}}$ 的位置 k 的概率分布。为了说明，考虑上述示例，其中序列 “*the early bird catches the worm*” 的两个标记被掩码。对于此示例，目标是最大化对数尺度概率的总和

$$\begin{aligned} \text{Loss} = & \log \Pr(x_2 = \text{early} | \bar{\mathbf{x}} = [\text{CLS}] \underbrace{\text{The [MASK] bird catches the [MASK]}}_{\bar{x}_2}) + \\ & \log \Pr(x_6 = \text{worm} | \bar{\mathbf{x}} = [\text{CLS}] \underbrace{\text{The [MASK] bird catches the [MASK]}}_{\bar{x}_6}) \end{aligned} \quad (1.12)$$

一旦我们获得优化参数 $\hat{\mathbf{W}}$ 和 $\hat{\theta}$ ，我们就可以删除 $\hat{\mathbf{W}}$ 。然后，我们可以进一步微调预训练编码器 $\text{Encoder}_{\hat{\theta}}(\cdot)$ 或直接将其应用于下游任务。

1.2.2.2 乱序语言模型

虽然掩码语言建模简单且应用广泛，但它引入了新的问题。一个缺点是使用一个特殊的标记 $\{v^*\}$ ，该标记仅在训练期间使用，而不在

在测试时。这导致训练和推理之间存在差异。此外，自动编码过程忽略了掩码标记之间的依赖关系。例如，在上面的例子中，对 x_2 (即第一个掩码标记的预测与)即第二个掩码标记的预测是独立进行的，尽管 x_6 (应该在)的上下文中考虑。

这些问题可以通过使用排列语言模型方法进行预训练来解决 [Yang 等人, 2019]。与因果语言模型类似，排列语言模型涉及对标记进行顺序预测。然而，与因果模型中预测遵循文本的自然顺序（如从左到右或从右到左）不同，排列语言模型允许以任何顺序进行预测。这种方法很简单：我们确定标记预测的顺序，然后以标准语言模型的方式训练模型，如第 1.2.1 节所述。请注意，在此方法中，文本中标记的实际顺序保持不变，而我们预测这些标记的顺序与标准语言模型不同。例如，考虑一个包含 5 个标记的序列 $x_0x_1x_2x_3x_4$ 。让 \mathbf{e}_i 表示 x_i (的嵌入，即标记嵌入和位置嵌入的组合)。在标准语言模型中，我们会按 $x_0 \rightarrow x_1 \rightarrow x_2 \rightarrow x_3 \rightarrow x_4$ 的顺序生成此序列。可以通过生成过程来模拟序列的概率。

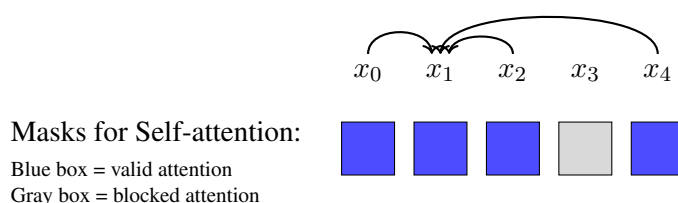
$$\begin{aligned}\Pr(\mathbf{x}) &= \Pr(x_0) \cdot \Pr(x_1|x_0) \cdot \Pr(x_2|x_0, x_1) \cdot \Pr(x_3|x_0, x_1, x_2) \cdot \\ &\quad \Pr(x_4|x_0, x_1, x_2, x_3) \\ &= \Pr(x_0) \cdot \Pr(x_1|\mathbf{e}_0) \cdot \Pr(x_2|\mathbf{e}_0, \mathbf{e}_1) \cdot \Pr(x_3|\mathbf{e}_0, \mathbf{e}_1, \mathbf{e}_2) \cdot \\ &\quad \Pr(x_4|\mathbf{e}_0, \mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3)\end{aligned}\tag{1.13}$$

现在，让我们考虑一个不同的顺序进行标记预测： $x_0 \rightarrow x_4 \rightarrow x_2 \rightarrow x_1 \rightarrow x_3$ 。序列生成过程可以表示如下：

$$\begin{aligned}\Pr(\mathbf{x}) &= \Pr(x_0) \cdot \Pr(x_4|\mathbf{e}_0) \cdot \Pr(x_2|\mathbf{e}_0, \mathbf{e}_4) \cdot \Pr(x_1|\mathbf{e}_0, \mathbf{e}_4, \mathbf{e}_2) \cdot \\ &\quad \Pr(x_3|\mathbf{e}_0, \mathbf{e}_4, \mathbf{e}_2, \mathbf{e}_1)\end{aligned}\tag{1.14}$$

这种新的预测顺序允许某些标记的生成基于更广泛的环境，而不是像标准语言模型那样仅限于前面的标记。例如，在生成 x_3 时，模型考虑了其左上下文（即， $\mathbf{e}_0, \mathbf{e}_1, \mathbf{e}_2$ ）和右上下文（即， \mathbf{e}_4 ）。嵌入 $\mathbf{e}_0, \mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_4$ 包含了 x_0, x_1, x_2, x_4 的位置信息，保留了标记的原始顺序。因此，这种方法在某种程度上类似于掩码语言模型：我们屏蔽 x_3 并使用其周围的标记 x_0, x_1, x_2, x_4 来预测此标记。

实现排列语言模型对 Transformers 来说相对容易。因为自注意力模型对输入顺序不敏感，我们不需要显式地重新排序序列以获得类似于式(1.14)的分解。相反，可以通过为自注意力设置适当的掩码来完成排列。例如，考虑计算 $\Pr(x_1|\mathbf{e}_0, \mathbf{e}_4, \mathbf{e}_2)$ 的情况。我们可以按顺序放置 x_0, x_1, x_2, x_3, x_4 并阻止自注意力中从 x_3 到 x_1 的注意力，如下所示



为了更直观的例子，我们在图1.3中比较了因果语言模型、掩码语言模型和置换语言模型的自我注意力掩码结果。

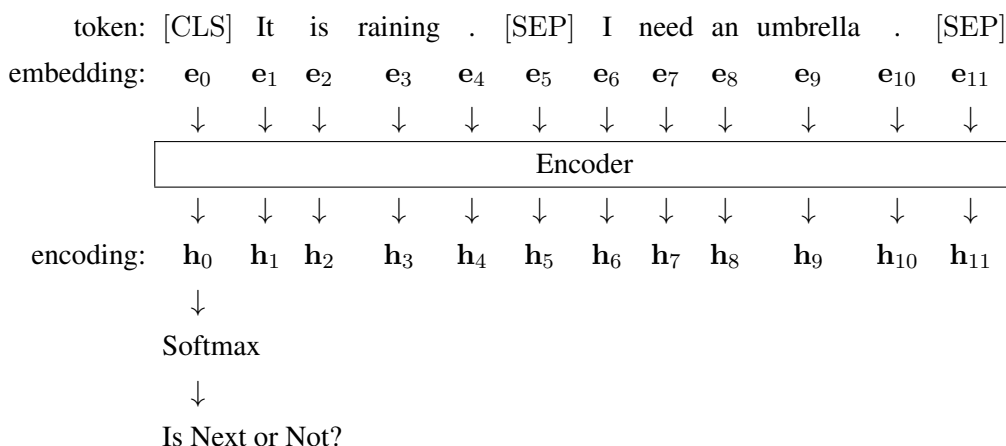
1.2.2.3 预训练编码器作为分类器

另一种常用的训练编码器的方法是考虑分类任务。在自监督学习中，这通常是通过从无标签文本中创建新的分类挑战来实现的。设计分类任务有许多不同的方法。在这里，我们介绍了两个流行的任务。

一种称为下一个句子预测（NSP）的简单方法在BERT的原始论文[Devlin等，2019]中被提出。NSP的假设是，一个好的文本编码器应该捕捉两个句子之间的关系。为了建模这种关系，在NSP中，我们可以使用编码两个连续句子 Sent_A 和 Sent_B 的输出，以确定 Sent_B 是否是跟在 Sent_A 之后的下一个句子。例如，假设 $\text{Sent}_A = \text{'It is raining.'}$ 和 $\text{Sent}_B = \text{'I need an umbrella.'}$ 。编码器的输入序列可能是

[CLS] It is raining . [SEP] I need an umbrella . [SEP]

在[CLS]是起始符号（即 x_0 ），这在编码器预训练中常用，而[SEP]是一个分隔符，用于分隔两个句子。此序列的处理遵循Transformer编码的标准程序：我们首先将每个标记 x_i 表示为其对应的嵌入 e_i ，然后将嵌入序列 $\{e_0, \dots, e_m\}$ 输入编码器以获得输出序列 $\{h_0, \dots, h_m\}$ 。由于 h_0 通常被认为是整个序列的表示，我们在其上方添加一个Softmax层来构建二元分类系统。此过程如下所示



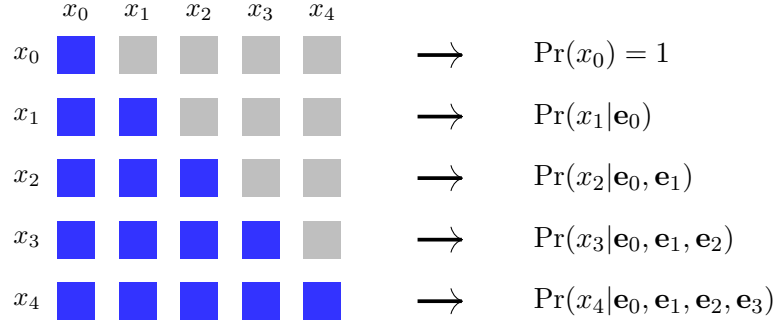
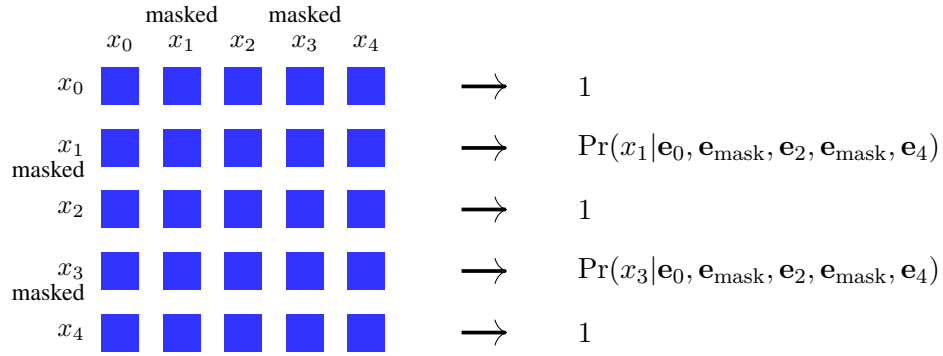
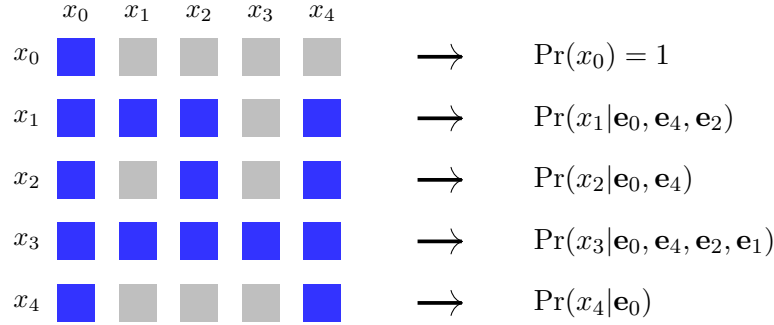
(a) Causal Language Modeling (order: $x_0 \rightarrow x_1 \rightarrow x_2 \rightarrow x_3 \rightarrow x_4$)(b) Masked Language Modeling (order: $x_0, [\text{MASK}], x_2, [\text{MASK}], x_4 \rightarrow x_1, x_3$)(c) Permuted Language Modeling (order: $x_0 \rightarrow x_4 \rightarrow x_2 \rightarrow x_1 \rightarrow x_3$)

图1.3: 因果语言模型、掩码语言模型和置换语言模型自我注意力掩码结果比较。灰色单元格表示位置 j 的标记不关注位置 i 的标记。蓝色单元格 (i, j) 表示位置 j 的标记关注位置 i 。 \mathbf{e}_{mask} 表示符号

为了生成训练样本，每次需要两个句子，一个用于 Sent_A ，另一个用于 Sent_B 。一种简单的方法是利用文本中两个连续句子的自然顺序。例如，我们通过使用实际的连续句子获得一个正样本，通过使用随机采样的句子获得一个负样本。因此，训练此模型与训练一个分类器相同。通常，NSP被用作额外的训练损失。

1.2.3 编码器-解码器预训练

在NLP中，编码器-解码器架构常用于建模序列到序列问题，例如机器翻译和问答。除了NLP中这些典型的序列到序列问题之外，编码器-解码器模型还可以扩展以处理许多其他问题。一个简单的想法是将文本视为问题的输入和输出，因此我们可以直接应用编码器-解码器模型。例如，给定一个文本，我们可以要求模型输出一个描述输入文本情感的文本，例如*positive*、*negative*和*neutral*。

这样的想法使我们能够开发一个单一的文本到文本系统来解决任何NLP问题。我们可以将不同的问题表述为相同的文本到文本格式。我们首先通过自监督训练一个编码器-解码器模型，以获得语言的一般知识。然后，使用针对性的文本到文本数据对该模型进行微调，以适应特定的下游任务。

1.2.3.1 隐藏编码器-解码器预训练

在Raffel等人[2020]的T5模型中，许多不同的任务被构造成相同的文本到文本任务。T5中的每个样本都遵循以下格式

Source Text → Target Text

这里 → 将源文本与任务描述或指令以及输入给系统的内容分开，源文本是对输入任务的响应。例如，考虑一个从中文翻译到英文的任务。一个训练样本可以表示为

[CLS] Translate from Chinese to English: 你好! → ⟨s⟩ Hello!

在源端和目标端，[CLS] 和 ⟨s⟩ 分别是起始符号⁵。

同样，我们也可以用同样的方式表达其他任务。例如

[CLS] 答案：阿尔伯特·爱因斯坦什么时候出生？
→ ⟨s⟩ 他出生于1879年3月14日。

[CLS] 简化：这位在其领域发表了众多论文的教授将在下周进行一次讲座。→ ⟨s⟩ 经验丰富的教授将在下周进行讲座。
[CLS] 对从英语到中文的翻译进行评分。英语：入乡随俗

[CLS] 罗马人这样做。中国人：在罗马，人就像罗马人一样做事。
→ ⟨s⟩ 0.81

s

在灰色中突出显示指令。一个有趣的例子是在最后一个例子中我们

⁵We could use the same start symbol for different sequences. Here we use different symbols to distinguish the sequences on the encoder and decoder-sides.

将评分问题重新定义为文本生成问题。我们的目标是生成表示数字0.81的文本，而不是将其作为数值输出。

上述方法提供了一种新的通用语言理解和生成框架。任务指令和问题输入都以文本形式提供给系统。然后系统遵循指令完成任务。这种方法将不同问题组合在一起，其优点是训练一个可以同时执行许多任务的单一模型。

一般来说，微调对于将预训练模型适应特定下游任务是必要的。在这个过程中，可以使用不同的方法来指导模型执行任务，例如使用任务的简称作为实际输入序列的前缀，或者提供任务的详细描述。由于任务指令以文本形式表达并作为输入的一部分，可以通过学习预训练阶段的语言理解模型来获得指令的一般知识。这有助于实现零样本学习。例如，预训练模型可以推广到解决从未遇到过任务指令的新问题。

存在几种针对Transformer编码器或解码器的强大自监督学习方法。将这些方法应用于预训练编码器-解码器模型相对简单。一个常见的选择是将编码器-解码器模型训练为语言模型。例如，编码器接收一个序列前缀，而解码器生成剩余的序列。然而，这与标准的因果语言模型不同，在标准因果语言模型中，整个序列是从第一个标记开始自回归生成的。在我们的情况下，编码器一次性处理前缀，然后解码器以因果语言模型的方式预测后续标记。更准确地说，这是一个前缀语言模型问题：语言模型根据前缀预测后续序列，该前缀作为预测的上下文。

考虑以下示例

$$\underbrace{[\text{CLS}] \text{ The puppies are frolicking}}_{\text{Prefix}} \rightarrow \underbrace{\langle s \rangle \text{ outside the house}}_{\text{Subsequent Sequence}}.$$

我们可以直接使用此类示例训练编码器-解码器模型。然后，编码器学会理解前缀，解码器学会根据这种理解继续写作。对于大规模预训练，从未标记的文本中创建大量训练示例很容易。

值得注意，为了使预训练的编码器-解码器模型在多语言和跨语言任务中有效，例如机器翻译，它们应该使用多语言数据进行训练。这通常需要词汇表包括所有语言的标记。通过这样做，模型可以学习不同语言之间的共享表示，从而在多语言和跨语言环境中实现语言理解和生成的能力。

第二种预训练编码器-解码器模型的方法是掩码语言建模。在此方法中，如第1.2.2节所述，序列中的标记被随机替换为掩码符号，然后模型被训练来根据整个掩码序列预测这些掩码标记。

作为一个说明在，考虑对 $\{v^*\}$ 进行掩码和重建的任务

句子

小狗们在房子外面嬉戏。

通过掩码两个标记（例如，*frolicking* 和 *the*），我们得到模型的BERT风格输入和输出，如下所示

[CLS] The puppies are [MASK] outside [MASK] house .
 → ⟨s⟩ _ _ _ frolicking _ the _ _

这里表示我们不进行标记预测的掩码位置。通过改变文本中标记的百分比，这种方法可以推广到BERT风格的训练或语言建模风格的训练[Song等人，2019]。例如，如果我们屏蔽掉所有标记，那么模型将训练生成整个序列

[CLS] [MASK] [MASK] [MASK] [MASK] [MASK] [MASK] [MASK] [MASK]
 → ⟨s⟩ The puppies are frolicking outside the house .

在这种情况下，我们训练解码器作为语言模型。

注意，在编码器-解码器架构的背景下，我们可以使用编码器读取掩码序列，并使用解码器预测原始序列。以此为目标，我们本质上有一个降噪自编码器：编码器将损坏的输入转换为某种隐藏表示，解码器从这种隐藏表示中重建未损坏的输入。以下是降噪训练的输入和输出示例。

[CLS] The puppies are [MASK] outside [MASK] house .
 → ⟨s⟩ The puppies are frolicking outside the house .

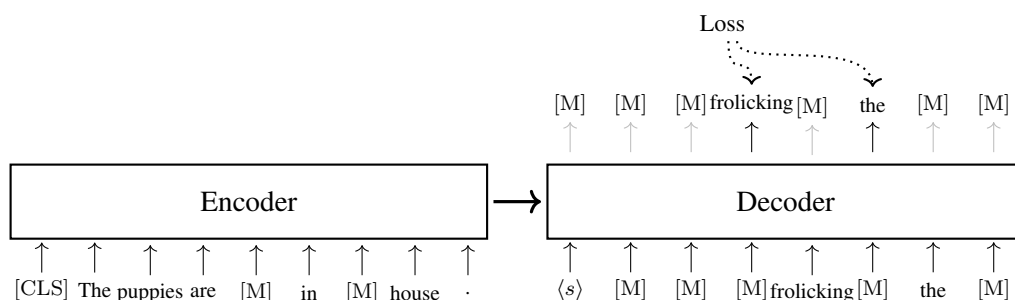
通过学习将此损坏序列映射到其未损坏的对应版本，模型获得了在编码器端理解以及在解码器端生成的能力。参见图1.4以了解如何使用BERT风格和去噪自编码目标训练编码器-解码器模型。

随着我们随机选择标记进行掩码，我们当然可以掩码连续的标记[Joshi等人，2020]。这里有一个例子。

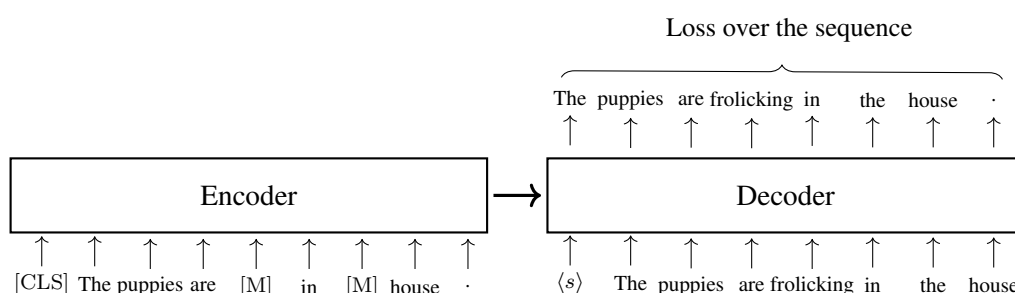
[CLS] The puppies are [MASK] outside [MASK] [MASK] .
 → ⟨s⟩ The puppies are frolicking outside the house .

另一种考虑连续掩码标记的方法是将它们表示为跨度。在这里，我们遵循Raffel等人[2020]的工作，并使用[X]、[Y]和[Z]来表示覆盖一个或多个连续掩码标记的哨兵标记。使用这种表示法，我们可以重新表达上述训练示例为

[CLS] The puppies are [X] outside [Y] .
 → ⟨s⟩ [X] frolicking [Y] the house [Z]



(a) Training an encoder-decoder model with BERT-style masked language modeling



(b) Training an encoder-decoder model with denoising autoencoding

图1.4：使用BERT风格和去噪自编码方法训练编码器-解码器模型。在这两种方法中，编码器的输入是一个损坏的标记序列，其中一些标记被遮挡并替换为 $\{v^*\}$ （或简称[M]）。解码器预测这些遮挡的标记，但方式不同。在BERT风格训练中，解码器只需要计算遮挡标记的损失，而序列中剩余的标记可以简单地视为 $\{v^*\}$ 标记。在去噪自编码中，解码器以自回归方式预测所有标记的序列。因此，损失是通过累积所有这些标记的损失来获得的，就像标准语言模型一样。

想法是将损坏的序列表示为包含占位符槽位的序列。训练任务是用周围的上下文填充这些槽位以正确的标记。这种方法的一个优点是，用于训练的序列会更短，从而使训练更高效。请注意，掩码语言模型为训练编码器-解码器模型提供了一个非常通用的框架。可以通过调整各种设置来拥有不同的训练版本，例如改变掩码标记的百分比和掩码序列的最大长度。

1.2.3.2 去噪训练

如果我们将训练编码器-解码器模型的问题视为训练去噪自编码器的问题，那么通常会有许多不同的方法来引入输入损坏和重建输入。例如，除了随机遮蔽标记之外，我们还可以改变其中一些或重新排列它们的顺序。

假设我们有一个可以将输入序列 x 映射到输出的编码器-解码器模型

序列 y

$$\begin{aligned} y &= \text{Decode}_{\omega}(\text{Encode}_{\theta}(\mathbf{x})) \\ &= \text{Model}_{\theta, \omega}(\mathbf{x}) \end{aligned} \quad (1.15)$$

在 θ 和 ω 分别是编码器和解码器的参数的情况下。在去噪自编码问题中，我们向 x 添加一些噪声以获得一个噪声的、损坏的输入 x_{noise} 。通过将 x_{noise} 输入到编码器中，我们希望解码器输出原始输入。训练目标可以定义为

$$(\hat{\theta}, \hat{\omega}) = \text{求解 } \arg \min_{\theta, \omega} \text{损失}(\text{Model}_{\theta, \omega}(x_{\text{noise}}), x) \quad (1.16)$$

这里损失函数 $\text{Loss}(\text{Model}_{\theta, \omega}(x_{\text{noise}}), x)$ 评估模型 $\text{Model}_{\theta, \omega}(x_{\text{noise}})$ 重建原始输入 x 的效果。我们可以像往常一样选择交叉熵损失。

随着模型架构和训练方法的开发，剩下的问题是输入的损坏。Lewis等人[2020]在他们提出的BART模型中，提出了以几种不同的方式损坏输入序列。

- 标记掩码。这是我们用于掩码语言模型中的相同掩码方法。输入序列中的标记被随机选择并掩码。
- 标记删除。此方法与标记掩码类似。然而，与用特殊符号 $\{v^*\}$ 替换所选标记不同，这些标记被从序列中删除。以下示例展示了标记掩码和标记删除方法的比较。

原始 (x) : 小狗们在房子外面嬉戏。标记遮挡 (x_{noise}) :
 小狗们在

在何处未 原始序列中的下划线标记被掩码或 已删除。

- 跨度掩码。非重叠的跨度在序列中随机采样。每个跨度都被掩码，并且在这种情况下， $\{v^*\}$ 简单地插入到序列中的某个位置。例如，我们可以使用跨度掩码来破坏上述序列，如下所示

Original (x): The 0 puppies are frolicking outside the house .
 Span Masking (x_{noise}): The [MASK] puppies are [MASK] house .

这里将 *frolicking outside the* 替换为单个 0 表示长度为 0 的跨度，因此我们在 *The* 和 *puppies* 之间插入一个 0。跨度掩码引入了新的预测挑战，其中模型需要知道从跨度中生成了多少个标记。这个问题与机器翻译中的生育力建模非常相似 [Brown 等人, 1993]。

如果我们考虑由多个句子组成的序列，可以应用额外的损坏方法。在BART模型中，有两种这样的方法。

- 句子重新排序。此方法随机排列句子，以便模型可以学习在文档中重新排列句子。例如，考虑两个连续的句子

努力工作导致成功。成功带来幸福。

我们可以重新排列这两个句子以获得一个损坏的输入序列

成功带来幸福。努力工作导致成功。

- 文档旋转。此任务的目标是识别序列的起始标记。首先，从序列中随机选择一个标记。然后，将序列旋转，使所选标记成为第一个标记。例如，假设我们从上述序列中选择标记 *leads*。旋转后的序列是

selected
 ↓
~~Hard work~~ leads to success . Success brings happiness . Hard work

在 *leads* 之前，子序列 *Hard work* 被追加到序列的末尾。

对于预训练，我们可以应用多种损坏方法来学习鲁棒模型，例如，我们为每个训练样本随机选择其中一种。在实践中，编码器-解码器预训练的结果在很大程度上取决于所使用的输入损坏方法，因此我们通常需要通过仔细实验来选择合适的训练目标。

1.2.4 预训练任务比较

截至目前，我们讨论了许多预训练任务。由于相同的训练目标可以应用于不同的架构（例如，使用掩码语言模型进行仅编码器预训练和编码器-解码器预训练），仅根据模型架构对预训练任务进行分类似乎并不理想。相反，我们根据训练目标对这些任务进行总结。

- 语言建模。通常，这种方法指的是序列的自回归生成过程。在某一时刻，它根据之前的上下文预测下一个标记。
- 掩码语言模型。掩码语言模型属于一种通用的掩码-预测框架。它随机掩码序列中的标记，并使用整个掩码序列预测这些标记。

- 排列语言模型。排列语言模型遵循与掩码语言模型类似的想法，但考虑了（掩码）标记预测的顺序。它重新排列输入序列并逐个预测标记。每个预测都基于一些随机选择的上下文标记。
- 判别性训练。在判别性训练中，从分类任务中创建监督信号。将预训练模型集成到分类器中，并与分类器的其余部分一起训练，以增强其分类性能。
- 去噪自编码。这种方法应用于编码器-解码器模型的预训练。输入是一个损坏的序列，编码器-解码器模型被训练以重建原始序列。

表1.1通过示例说明了这些方法和它们的变体。使用这些示例并不区分模型，但我们标记了可以应用预训练任务的模型架构。在每个示例中，输入由一个标记序列组成，输出要么是一个标记序列，要么是一些概率。对于生成任务，如语言建模，上标用于指示目标侧的生成顺序。如果省略上标，则表示输出序列可以自回归或同时生成。在源侧，我们假设序列经过标准的Transformer编码过程，这意味着每个标记都可以在自注意力中看到整个序列。唯一的例外是在置换语言建模中，通过在编码器侧设置注意力掩码来实现自回归生成过程。为了简化讨论，我们从每个示例的目标侧移除了标记 $\langle s \rangle$ 。

虽然这些预训练任务不同，但可以在相同框架和实验设置下进行比较[Dong等人，2019；Raffel等人，2020；Lewis等人，2020]。请注意，我们无法在此列出所有预训练任务，因为它们有很多。有关预训练任务的更多讨论，感兴趣的读者可以参考一些关于这个主题的调查[Qiu等人，2020；Han等人，2021]。

1.3 示例：BERT

本节中，我们介绍了BERT模型，这些模型是NLP中最受欢迎和广泛使用的预训练序列编码模型之一。

1.3.1 标准模型

标准BERT模型，由Devlin等人[2019]的工作中提出，是一个使用掩码语言建模和下一句预测任务训练的Transformer编码器。在此模型训练中使用的损失是两个任务损失的加和。

$$\text{Loss}_{\text{BERT}} = \text{Loss}_{\text{MLM}} + \text{Loss}_{\text{NSP}} \quad (1.17)$$

如常规训练深度神经网络一样，我们通过最小化这个损失来优化模型参数。为此，收集了大量的训练样本。在训练过程中，一批

Method	Enc	Dec	E-D	Input	Output
Causal LM		•	•		The ¹ kitten ² is ³ chasing ⁴ the ⁵ ball ⁶ . ⁷
Prefix LM		•	•	[C] The kitten is	chasing ¹ the ² ball ³ . ⁴
Masked LM	•		•	[C] The kitten [M] chasing the [M] .	_ _ is _ _ ball _
MASS-style	•		•	[C] The kitten [M] [M] [M] ball .	_ _ is chasing the _ _
BERT-style	•		•	[C] The kitten [M] playing the [M] .	_ kitten is chasing _ ball _
Permuted LM	•			[C] The kitten is chasing the ball .	The ⁵ kitten ⁷ is ⁶ chasing ¹ the ⁴ ball ² . ³
Next Sentence Prediction	•			[C] The kitten is chasing the ball . Birds eat worms .	Pr(IsNext representation-of-[C])
Sentence Comparison	•			Encode a sentence as \mathbf{h}_a and another sentence as \mathbf{h}_b	Score($\mathbf{h}_a, \mathbf{h}_b$)
Token Classification	•			[C] The kitten is chasing the ball .	Pr(\cdot The) Pr(\cdot kitten) ... Pr(\cdot .)
Token Reordering			•	[C] . kitten the chasing The is ball	The ¹ kitten ² is ³ chasing ⁴ the ⁵ ball ⁶ . ⁷
Token Deletion			•	[C] The kitten is chasing the ball .	The ¹ kitten ² is ³ chasing ⁴ the ⁵ ball ⁶ . ⁷
Span Masking			•	[C] The kitten [M] is [M] .	The ¹ kitten ² is ³ chasing ⁴ the ⁵ ball ⁶ . ⁷
Sentinel Masking			•	[C] The kitten [X] the [Y]	[X] ¹ is ² chasing ³ [Y] ⁴ ball ⁵ . ⁶
Sentence Reordering			•	[C] The ball rolls away swiftly . The kitten is chasing the ball .	The ¹ kitten ² is ³ chasing ⁴ the ⁵ ball ⁶ . ⁷ The ⁸ ball ⁹ rolls ¹⁰ away ¹¹ swiftly ¹² . ¹³
Document Rotation			•	[C] chasing the ball . The ball rolls away swiftly . The kitten is	The ¹ kitten ² is ³ chasing ⁴ the ⁵ ball ⁶ . ⁷ The ⁸ ball ⁹ rolls ¹⁰ away ¹¹ swiftly ¹² . ¹³

1.1: 预训练任务比较, 包括语言建模、掩码语言建模、排列语言建模、判别性训练和去噪自编码。[C] = [CLS], [M] = [X], [Y] = 哨兵标记。Enc, Dec和E-D分别表示方法是否适用于仅编码器、仅解码器、编码器-解码器模型。对于生成任务, 上标用于表示标记的顺序。

训练样本是从该集合中随机选取的, 并且在这些训练样本上累积 $\text{Loss}_{\text{BERT}}$ 。然后, 通过梯度下降或其变体更新模型参数。这个过程会重复多次, 直到满足某些停止标准, 例如当训练损失收敛时。

1.3.1.1 损失函数

一般来说, BERT模型用于表示单个句子或一对句子, 因此可以处理各种下游语言理解问题。在本节中, 我们假设输入表示是一个包含两个句子 Sent_A 和 Sent_B 的序列, 表示为

$$[\text{CLS}] \text{ Sent}_A [\text{SEP}] \text{ Sent}_B [\text{SEP}]$$

这里我们遵循BERT论文中的符号表示法, 并使用[SEP]表示分隔符。

给定这个序列, 我们可以分别获得 Loss_{MLM} 和 Loss_{NSP} 。对于掩码语言建模, 我们预测序列中的一些标记。通常, 一定百分比的

标记是随机选择的，例如，在标准BERT模型中，每个序列中的15%的标记被选中。然后以三种方式修改序列

- 标记掩码。所选标记的80%被掩码并用符号{v*}替换。例如

Original: [CLS] It is raining . [SEP] I need an umbrella . [SEP]
Masked: [CLS] It is [MASK] . [SEP] I need [MASK] umbrella . [SEP]

在选定的标记下划线处。预测掩码标记使模型学会从其周围上下文中表示标记。

- 随机替换。所选标记的10%被替换为随机标记。例如

Original: [CLS] It is raining . [SEP] I need an umbrella . [SEP]
Random Token: [CLS] It is raining . [SEP] I need an hat . [SEP]

这有助于模型学习从噪声输入中恢复一个标记。

- 未更改。所选标记的10%保持不变。例如，

Original: [CLS] It is raining . [SEP] I need an umbrella . [SEP]
Unchanged Token: [CLS] It is raining . [SEP] I need an umbrella . [SEP]

这不是一个困难的预测任务，但可以引导模型使用更简单的证据进行预测。

设 $\mathcal{A}(\mathbf{x})$ 为给定标记序列 \mathbf{x} 的选定位置集合， $\bar{\mathbf{x}}$ 为修改后的序列 \mathbf{x} 。掩码语言模型的损失函数可以定义为

$$\text{Loss}_{\text{MLM}} = - \sum_{i \in \mathcal{A}(\mathbf{x})} \log \Pr_i(x_i | \bar{\mathbf{x}}) \quad (1.18)$$

在位置 $\bar{\mathbf{x}}$ 给定的情况下， $\Pr_i(x_i | \bar{\mathbf{x}})$ 是预测 x_i 的概率。图 1.5 展示了计算 Loss_{MLM} 的运行示例。

对于下一个句子预测，我们遵循第1.2.2.3节中描述的方法。每个训练样本被分类到标签集 $\{\text{IsNext}, \text{NotNext}\}$ ，例如，

序列: [CLS] 正在下雨。 [SEP] 我需要一把伞。 [SEP] 标签: IsNext 序列: [CLS] 猫在窗台上睡觉。 [SEP] 苹果长在树上。 [SEP] 标签: NotNext

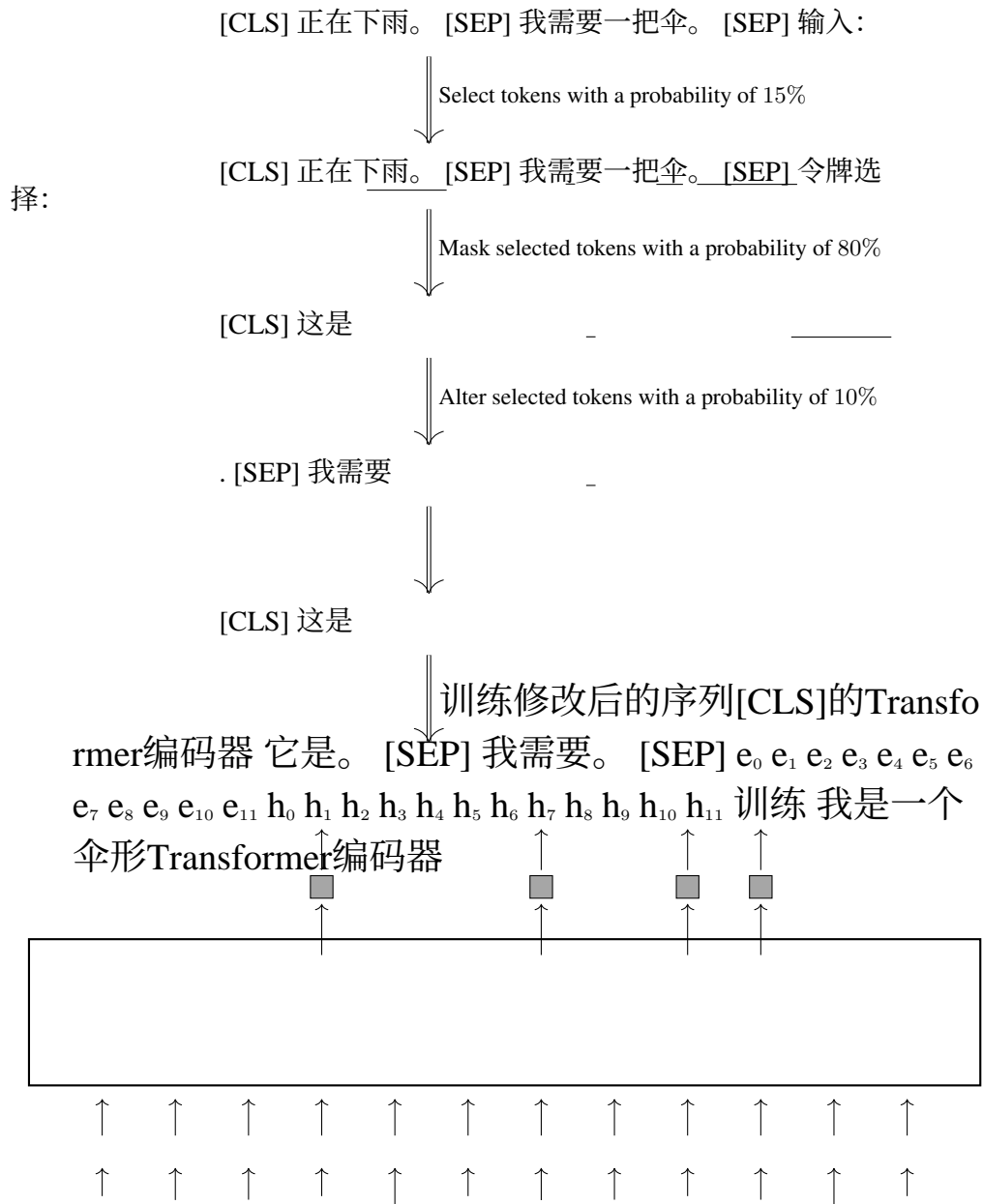


图1.5: BERT风格掩码语言模型的运行示例。首先, 随机选择15%的标记。这些选定的标记随后以三种方式之一进行处理: 80%的时间用{v*}标记替换, 10%的时间用随机标记替换, 或保持不变(10%的时间)。模型训练用于根据修改后的序列预测这些选定的标记。 e_i 表示位置 i 的标记嵌入。灰色框表示Softmax层。

输出向量 \mathbf{h}_{cls} (或 \mathbf{h}_0)被视为序列表示。在 \mathbf{h}_{cls} 之上构建一个分类器。然后, 我们可以计算给定 \mathbf{h}_{cls} 的标签 c 的概率, 即 $\Pr(c|\mathbf{h}_{\text{cls}})$ 。对于分类问题, 可以选择许多损失函数。例如, 在最大似然训练中, 我们可以定义 Loss_{NSP} 为

$$\text{Loss}_{\text{NSP}} = -\log \Pr(c_{\text{gold}}|\mathbf{h}_{\text{cls}}) \quad (1.19)$$

在 c_{gold} 是此样本的正确标签。

1.3.1.2 模型设置

如图1.6所示，BERT模型基于标准的Transformer编码器架构。输入是一个嵌入序列，每个嵌入是标记嵌入、位置嵌入和段嵌入的总和。

$$\mathbf{e} = \mathbf{x} + \mathbf{e}_{\text{pos}} + \mathbf{e}_{\text{seg}} \quad (1.20)$$

两个标记嵌入 (\mathbf{x}) 和位置嵌入 (\mathbf{e}_{pos}) 都是常规的，就像在Transformer模型中一样。段嵌入 (\mathbf{e}_{seg}) 是一种新的嵌入类型，表示一个标记是否属于 Sent_A 或 Sent_B 。这可以通过以下示例来说明。

Token	[CLS]	It	is	raining	.	[SEP]	I	need	an	umbrella	.	[SEP]
\mathbf{x}	\mathbf{x}_0	\mathbf{x}_1	\mathbf{x}_2	\mathbf{x}_3	\mathbf{x}_4	\mathbf{x}_5	\mathbf{x}_6	\mathbf{x}_7	\mathbf{x}_8	\mathbf{x}_9	\mathbf{x}_{10}	\mathbf{x}_{11}
\mathbf{e}_{pos}	PE(0)	PE(1)	PE(2)	PE(3)	PE(4)	PE(5)	PE(6)	PE(7)	PE(8)	PE(9)	PE(10)	PE(11)
\mathbf{e}_{seg}	\mathbf{e}_A	\mathbf{e}_A	\mathbf{e}_A	\mathbf{e}_A	\mathbf{e}_A	\mathbf{e}_A	\mathbf{e}_B	\mathbf{e}_B	\mathbf{e}_B	\mathbf{e}_B	\mathbf{e}_B	\mathbf{e}_B

BERT模型的主要部分是一个多层Transformer网络。一个Transformer层由一个自注意力子层和一个FFN子层组成。它们都遵循后归一化架构：输出 = $\text{LNorm}(F(\text{输入}) + \text{输入})$ ，其中 $F(\cdot)$ 是子层的核心函数（要么是自注意力模型，要么是FFN）， $\text{LNorm}(\cdot)$ 是层归一化单元。通常，将多个Transformer层堆叠形成一个深度网络。在序列的每个位置，输出表示是一个由网络的最后一层产生的实值向量。

有多个方面可以考虑在开发BERT模型时。

- 词汇量 ($|V|$)。在Transformer中，每个输入标记都表示为词汇表 V 中的一个条目。大型词汇表可以覆盖更多单词的表面形式变体，但可能导致存储需求增加。
- 嵌入大小 (d_e)。每个标记表示为一个 d_e 维的实值向量。如上所述，该向量是标记嵌入、位置嵌入和段嵌入的总和，这些嵌入也都是 d_e 维的实值向量。
- 隐藏大小 (d)。子层的输入和输出维度为 d 。此外，子层的大部分隐藏状态是 d 维向量。一般来说， d 可以大致看作网络的宽度。
- 数字头 (n_{head})。在自注意力子层中，需要指定用于多头自注意力的头数。这个数字越大，执行的注意力子空间就越多。在实际系统中，我们通常设置 $n_{\text{head}} \geq 4$ 。
- FFN 隐藏层大小 (d_{ffn})。在 Transformers 中使用的 FFN 的隐藏层大小通常比 d 更大。例如，一个典型的设置是 $d_{\text{ffn}} = 4d$ 。对于更大的 Transformers，例如最近的大型模型， d_{ffn} 可能被设置为非常大的值。

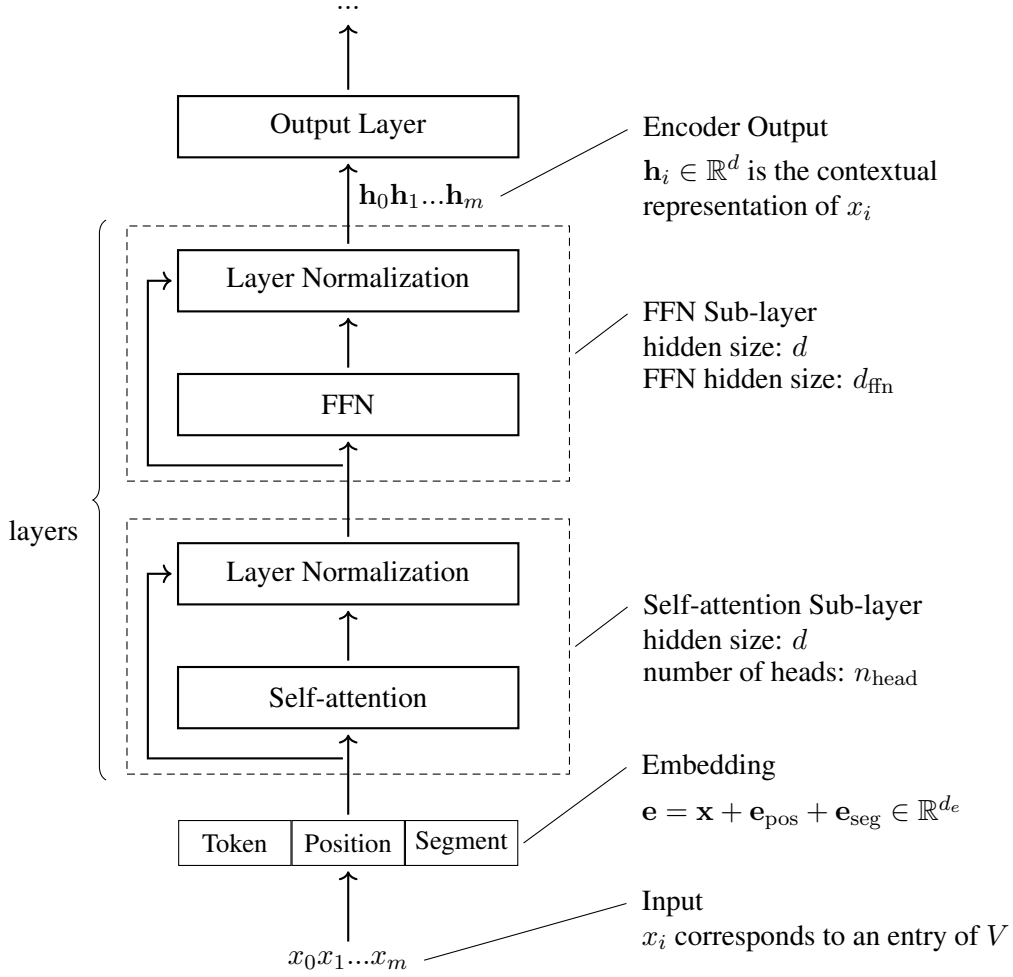


图1.6: BERT (Transformer编码器) 的模型架构。输入标记首先表示为嵌入, 每个嵌入是相应的标记嵌入、位置嵌入和段嵌入的总和。然后, 嵌入序列通过一系列Transformer层进行处理。这个堆栈中的每一层都包括一个自注意力子层和一个FFN子层。BERT模型的输出是最终Transformer层产生的向量序列。

- 模型深度 (L)。使用深度网络是提高Transformer表达能力的一种有效方法。对于BERT模型, L 通常设置为12或24。然而, 更深层的网络也是可行的, 并且可以用于进一步的增强。

不同超参数的设置会导致模型大小不同。有两种广泛使用的BERT模型。

- BERT_{base}: $d = 768, L = 12, n_{\text{head}} = 12$, 总参数数量 = 110M。
- BERT_{large}: $d = 1,024, L = 24, n_{\text{head}} = 16$, 总参数数量 = 340M。

训练BERT模型遵循Transformers的标准训练流程。训练像BERT_{large}这样的大型模型需要更多的训练努力和时间。这在预训练中是一个常见问题, 尤其是在模型在非常大量的数据上训练时。在实践中,

通常需要考虑训练效率。例如，一种做法是首先在相对较短的序列上对BERT模型进行大量训练步骤的训练，然后继续在完整长度的序列上进行剩余的训练步骤。

1.3.2 更多训练和更大的模型

BERT是NLP中的一个里程碑模型，引发了众多后续努力来改进它。一个方向是扩大模型本身，包括增加训练数据和开发更大的模型。

RoBERTa，标准BERT模型的扩展，是此类努力的例子[Liu等人，2019]。它引入了两个主要改进。首先，简单地使用更多的训练数据和更多的计算能力可以提高BERT模型，而无需改变模型架构。其次，如果训练规模扩大，去除NSP损失不会降低下游任务的表现。这些发现表明，探索预训练的一般方向：我们可以在简单的预训练任务上扩大规模，以继续改进预训练。

第二种提高BERT模型的方法是增加模型参数数量。例如，在He等人[2021]的研究中，通过增加模型深度和隐藏大小，构建了一个具有15亿参数的类似BERT模型。然而，扩大BERT和各种其他预训练模型在训练中引入了新的挑战，例如，训练非常大的模型往往变得不稳定且难以收敛。这使得问题更加复杂，需要仔细考虑包括模型架构、并行计算、参数初始化等方面的各个方面。在另一个例子中，Shoeybi等人[2019]成功训练了一个具有39亿参数的类似BERT模型，其中使用了数百个GPU来管理增加的计算需求。

1.3.3 更高效的模型

与它的前辈相比，BERT在提出时是一个相对较大的模型。这种模型规模的增加导致了更大的内存需求，从而降低了系统性能。开发更小、更快的BERT模型是构建高效Transformer的更广泛挑战的一部分，这在Tay等人[2020]的工作和Xiao和Zhu[2023]的工作中已有广泛讨论。然而，对这个一般主题的深入讨论超出了我们当前讨论的范围。在这里，我们考虑BERT的几个高效变体。

多个研究线程对NLP研究人员在开发高效的BERT模型方面感兴趣。首先，关于知识蒸馏的工作，例如使用经过良好训练的教师模型的输出训练学生模型，表明可以通过从较大的BERT模型中转移知识来获得较小的BERT模型。鉴于BERT模型是多层网络，具有几种不同类型的层，知识蒸馏可以应用于不同级别的表示。例如，除了从输出层蒸馏知识之外，还可以结合训练损失，该损失衡量教师模型和学生模型之间隐藏层输出的差异[Jiao等人，2020年；Sun等人，2020年]。事实上，知识蒸馏一直是学习小型预训练模型最广泛使用的技巧之一。

其次，传统的模型压缩方法可以直接应用于压缩BERT模型。一种常见的方法是使用通用剪枝方法来剪枝Transformer编码网络[Gale等人，2019]。这通常涉及移除整个层[Fan

等, 2019] 或网络中一定百分比的参数 [Sanh 等, 2020; Chen 等, 2020]。剪枝也适用于多头注意力模型。例如, Michel 等 [2019] 表明, 移除一些头不会显著降低 BERT 模型的性能, 但会加快这些模型的推理速度。压缩 BERT 模型的另一种方法是量化 [Shen 等, 2020]。通过将模型参数表示为低精度数字, 模型可以大大压缩。虽然这种方法并非特定于 BERT 模型, 但它对大型基于 Transformer 的架构证明是有效的。

第三, 考虑到 BERT 模型相对较深且网络较大, 另一条研究思路使用动态网络来适应这些模型以实现高效推理。该范式中的一个想法是动态选择处理标记的层, 例如, 在深度自适应模型中, 我们在某个最优深度退出, 从而跳过层堆栈中的其余层 [Xin 等人, 2020; Zhou 等人, 2020]。同样, 我们可以开发长度自适应模型, 其中输入序列的长度可以动态调整。例如, 我们可以跳过输入序列中的某些标记, 以便模型可以减少对不太重要的标记的计算负担, 从而提高整体效率。

第四, 还可以在层之间共享参数以减小 BERT 模型的大小。实现这一点的简单方法是在层堆栈中共享整个 Transformer 层的参数 [Dehghani 等人, 2018; Lan 等人, 2020]。除了减少参数数量外, 这还使得在多层 Transformer 网络中可以重用相同的层, 从而在测试时节省内存占用。

1.3.4 多语言模型

初始的 BERT 模型主要关注英语。在此模型提出后不久, 它被扩展到许多语言。实现这一目标的一种简单方法是为每种语言开发一个单独的模型。另一种方法, 在最近关于大型语言模型的研究中变得更加流行, 是直接在所有语言的数据上训练多语言模型。作为回应, 通过在 104 种语言的文本上训练, 开发了多语言 BERT (mBERT) 模型。与单语言 BERT 模型的主要区别在于, mBERT 模型使用更大的词汇表来覆盖多种语言的标记。因此, 来自不同语言的标记表示被映射到同一空间, 通过这个通用表示模型允许跨语言的知识共享。

多语言预训练模型的一个重要应用是跨语言学习。在跨语言设置中, 我们在一种语言的任务上学习一个模型, 并将其应用于另一种语言中的相同任务。例如, 在跨语言文本分类中, 我们针对英语标注文档微调一个多语言预训练模型。然后, 我们使用微调后的模型对中文文档进行分类。

对多语言预训练模型如 mBERT 的改进是将双语数据引入预训练。与其仅从多种语言的单一语言数据中训练, 双语训练明确地建模两种语言中标记之间的关系。所得到的模型将具有固有的跨语言迁移能力, 因此可以轻松适应不同的语言。Lample 和 Conneau [2019] 提出了一种预训练跨语言语言模型 (XLMs) 的方法。在他们工作中, 跨语言语言模型可以以因果语言建模或掩码语言建模的方式进行训练。对于掩码语言建模

⁶<https://github.com/google-research/bert/>

预训练中，模型被视为编码器。训练目标与BERT相同：我们最大化一些随机选择的标记的概率，这些标记在输入中要么被遮蔽，要么被随机标记替换，要么保持不变。如果我们考虑双语数据在预训练中，我们每次采样一对对齐的句子。然后，将这两个句子打包在一起形成一个用于训练的单个序列。例如，考虑一个英语-中文句子对

鲸鱼 是 哺乳 动物 。 ↔ Whales are mammals .

我们可以将它们打包成一个序列，如下所示

[CLS] 鲸鱼是哺乳动物。[SEP] 鲸鱼是哺乳动物。[SEP]

然后我们选择一定百分比的标记并替换它们

是

预训练的目标是最大化给定上述序列的掩码标记概率的乘积。通过这种方式进行训练，模型可以学习表示英语和中文序列，以及捕捉两种语言标记之间的对应关系。例如，预测中文标记“鲸鱼”可能需要来自英语标记*Whales*的信息。对两种语言的表示进行对齐本质上将模型转变为“翻译”模型。因此，这个训练目标也被称为翻译语言模型。图1.7展示了这种方法的一个说明。

多语言预训练模型的一个优点是它们固有的处理代码切换的能力。在自然语言处理和语言学中，代码切换指的是在文本中切换语言。例如，以下是一个包含中文和英文的混合语言文本：

周末 我们 打算 去 做 徒步旅行， 你 想 一起 来 吗？
(我们计划本周末去徒步旅行，你愿意加入我们吗？)

对于多语言预训练模型，我们不需要识别一个标记是中文还是英文。相反，每个标记只是共享词汇表中的一个条目。这可以想象成创建一个“新”语言，它包含我们想要处理的所有语言。

多语言预训练的结果受多种因素影响。鉴于模型架构固定，需要指定共享词汇表的大小、每种语言中的样本数量（或百分比）、模型大小等。Conneau等人[2020]指出，关于XLM-like模型的大规模多语言预训练存在几个有趣的问题。首先，随着支持的语言数量增加，需要更大的模型来处理这些语言。其次，更大的共享词汇表有助于模拟语言多样性的增加。第三，低资源语言更容易从高资源语言中获得跨语言迁移的好处，尤其是在预训练中涉及类似的高资源语言时。然而，如果模型训练时间过长，可能会出现干扰。

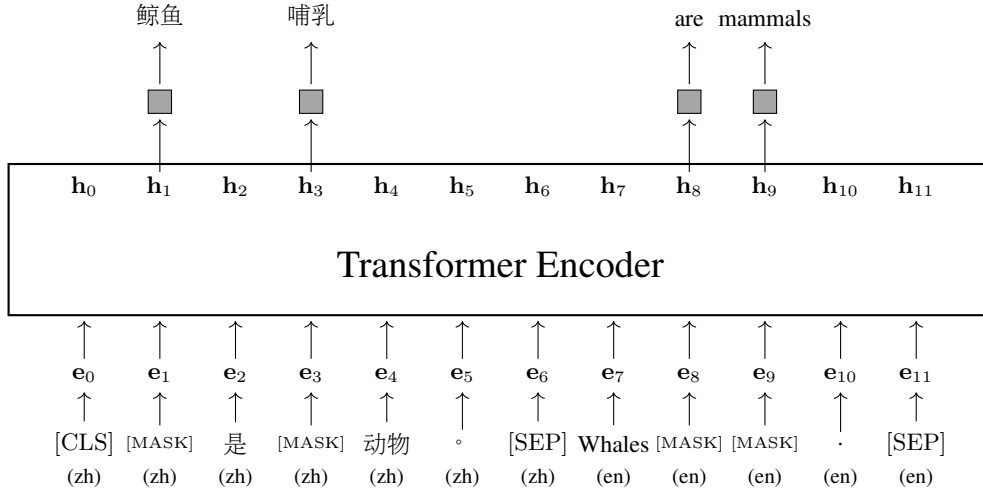


图1.7: 翻译语言建模的示意图。为了便于理解, 我们提供了一个简单示例, 其中所有选定的标记都被掩码。该模型被训练来预测这些掩码标记。由于序列包含两种语言的标记, 预测一种语言的标记可以访问另一种语言的标记, 从而实现跨语言建模。在Lample和Conneau[2019]的研究中, 输入嵌入(即 e_i)是标记嵌入、位置嵌入和语言嵌入的总和。这要求每个标记都被分配一个语言标签。因此, 我们可以区分不同语言的标记。在多语言预训练中, 尤其是在使用共享词汇的工作中, 指定标记所属的语言不是必要的。语言嵌入的使用反过来又使得处理代码转换变得困难。因此, 我们假设这里所有标记表示都是语言无关的。

表示在预训练过程中, 预训练模型的总体性能从某个点开始下降。因此, 在实际系统中, 可能需要提前停止预训练以防止干扰。

1.4 应用 BERT 模型

一旦BERT模型预训练完成, 就可以用来解决NLP问题。但是BERT模型并不能立即用于执行特定的下游任务。通常, 需要额外的微调工作来使它们适应。作为第一步, 我们需要一个预测器来将模型的输出与感兴趣的问题对齐。设 $\text{BERT}_{\hat{\theta}}(\cdot)$ 是一个具有预训练参数 $\hat{\theta}$ 的BERT模型, $\text{Predict}_{\omega}(\cdot)$ 是一个具有参数 ω 的预测网络。通过将预测网络与BERT模型的输出集成, 我们开发了一个用于处理下游任务的模型。该模型可以表示为

$$\mathbf{y} = \text{Predict}_{\omega}(\text{BERT}_{\hat{\theta}}(\mathbf{x})) \quad (1.21)$$

在 \mathbf{x} 是输入且 \mathbf{y} 是符合问题的输出的情况下。例如, 在分类问题中, 模型输出标签的概率分布。

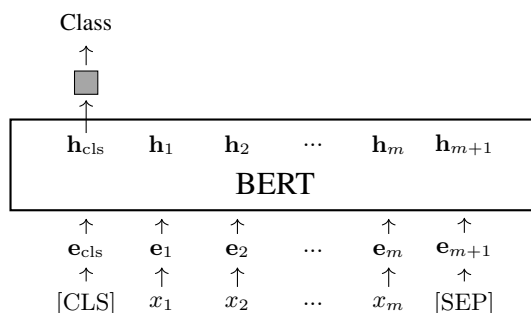
然后, 我们收集一组标记样本 \mathcal{D} , 并通过以下方式微调模型

$$(\tilde{\omega}, \tilde{\theta}) = \arg \min_{\omega, \hat{\theta}^+} \sum_{(\mathbf{x}, \mathbf{y}_{\text{gold}}) \in \mathcal{D}} \text{Loss}(\mathbf{y}_{\omega, \hat{\theta}^+}, \mathbf{y}_{\text{gold}}) \quad (1.22)$$

(x, y_{gold}) 表示一个输入及其对应输出的元组。这个方程的表示似乎有点复杂，但训练/调整过程是标准的。我们通过最小化调整样本上的损失来优化模型。结果是优化后的参数 $\tilde{\omega}$ 和 $\tilde{\theta}$ 。优化从预训练参数 $\hat{\theta}$ 开始。这里我们用 $\hat{\theta}^+$ 来表示参数是用 $\hat{\theta}$ 初始化的，并用 $y_{\omega, \hat{\theta}^+}$ 来表示使用参数 ω 和 $\hat{\theta}^+$ 计算出的模型输出。

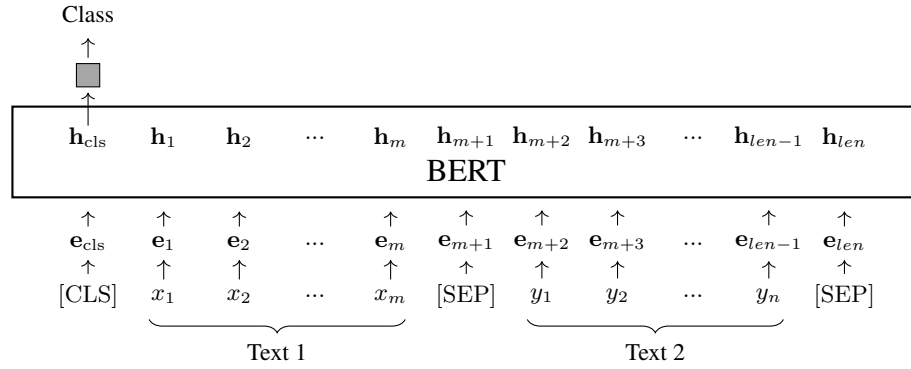
使用微调参数 $\tilde{\omega}$ 和 $\tilde{\theta}$ ，我们可以将模型 $\text{Predict}_{\tilde{\omega}}(\text{BERT}_{\tilde{\theta}}(\cdot))$ 应用于与模型微调相同的任务的新数据。下游任务的形式决定了模型的输入和输出格式，以及预测网络的架构。以下列出一些BERT模型通常适用的任务。

- 分类（单文本）。BERT模型最广泛应用的场景之一是文本分类。在这个任务中，BERT模型接收一个标记序列并将其编码为向量序列。第一个输出向量 h_{cls} (或 h_0)通常用作整个文本的表示。预测网络以 h_{cls} 作为输入以产生标签分布。设 $[\text{CLS}]x_1x_2\dots x_m$ 为输入文本。下面是BERT文本分类的示例。



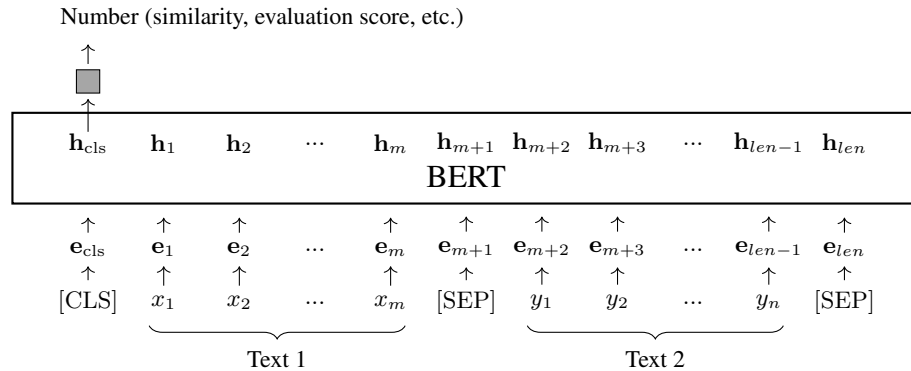
这里灰色框表示预测网络。许多NLP问题可以归类为文本分类任务，并且已经有一些文本分类基准用于评估预训练模型。例如，我们可以根据语法正确性（语法性）或情感基调（情感）对文本进行分类[Socher等人，2013；Warstadt等人，2019]。请注意，预测网络可以是任何分类模型，例如深度神经网络或更传统的分类模型。然后，整个模型可以像标准分类模型一样进行训练或微调。例如，预测网络可以简单地是一个Softmax层，模型参数可以通过最大化正确标签的概率进行优化。

- 分类（文本对）。分类也可以对一对文本进行。假设我们有两个文本， $x_1\dots x_m$ 和 $y_1\dots y_n$ 。我们可以将这些文本连接起来形成一个长度为 len 的单个序列。然后，我们可以根据 h_{cls} 向量预测这个组合文本序列的标签，如下所示



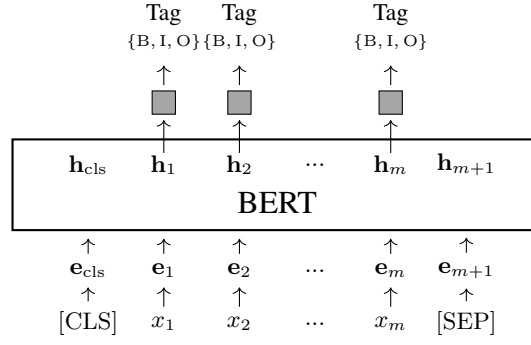
在 $len = n + m + 2$. 文本对分类涵盖了多个问题，包括语义等价判断（确定两个文本是否在语义上等价）[Dolan 和 Brockett, 2005]，文本蕴涵判断（确定一个假设是否可以从前提中逻辑地推断或蕴涵）[Bentivogli 和 Giampiccolo, 2011; Williams 等人, 2018]，基于常识的推理（确定给定其上下文的事件是否可能发生）[Zellers 等人, 2018]，以及问答推理（确定一个答案是否与给定的问题相对应）。

- 回归。我们不是生成标签分布，而是让预测网络输出一个实值分数。例如，通过在预测网络中添加Sigmoid层，系统可以用于计算两个给定句子之间的相似度。该架构与基于BERT的分类系统相同，只是输出层发生了变化。



对于训练或微调，我们可以像往常一样最小化模型输出的回归损失。

- 序列标注。序列标注是一种适用于广泛自然语言处理（NLP）问题的机器学习方法。这种方法为输入序列中的每个标记分配一个标签，然后可以从这些标签序列中推导出一些语言注释。NLP中序列标注的一个例子是词性标注（POS）。我们用相应的POS标签标注句子中的每个词。另一个例子是命名实体识别（NER），其中我们用NER标签标注每个词，并使用这些标签识别命名实体。下面是NER模型架构的示意图。

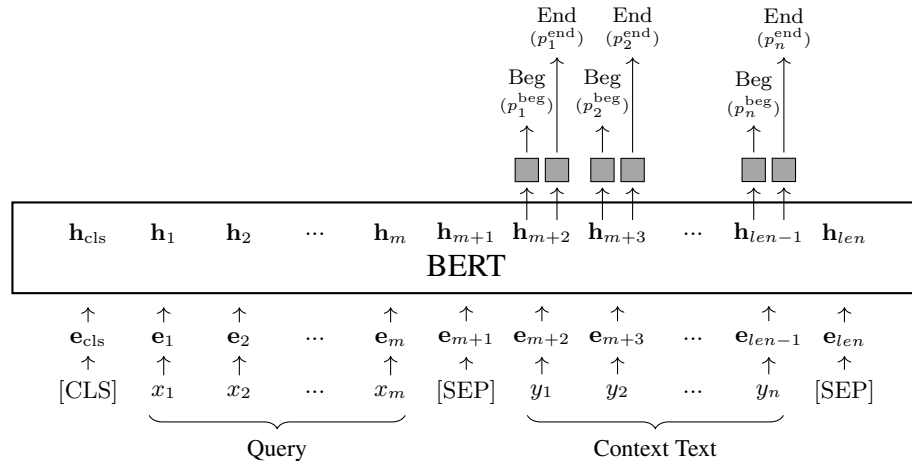


这里 $\{v^*\}$ 是 NER 的标签集。例如，B-ORG 表示组织的开始，I-ORG 表示单词在组织内部，O 表示单词不属于任何命名实体。此 NER 模型可以在每个位置输出标签集的分布，表示为 p_i 。模型的训练或微调可以在这些分布 $\{p_1, \dots, p_m\}$ 上进行。例如，假设 $p_i(\text{tag}_i)$ 是位置 i 上正确标签的概率。训练损失可以定义为负似然

$$\text{Loss} = -\frac{1}{m} \sum_{i=1}^m \log p_i(\text{tag}_i) \quad (1.23)$$

找到给定训练好的命名实体识别模型的最佳标签序列是 NLP 中一个研究已久的问题。这通常通过动态规划实现，在晶格路径查找的上下文中，具有线性复杂度[Huang, 2009]。

- 跨度预测。一些 NLP 任务需要预测文本中的跨度。一个常见的例子是阅读理解。在这个任务中，我们给定一个查询 $x_1 \dots x_m$ 和一个上下文文本 $y_1 \dots y_n$ 。目标是识别 $y_1 \dots y_n$ 中的一个连续跨度，以最好地回答查询。这个问题可以表述为一个类似于序列标注的任务，其中我们预测每个 y_j 的标签，以指示跨度的开始或结束。遵循 Seo 等人[2017]的方法，我们在 BERT 输出 y_j 的基础上添加了两个网络：一个用于生成 y_j 是跨度开始的概率（表示为 p_j^{beg} ），另一个用于生成 y_j 是跨度结束的概率（表示为 p_j^{end} ）。结果模型架构如下所示



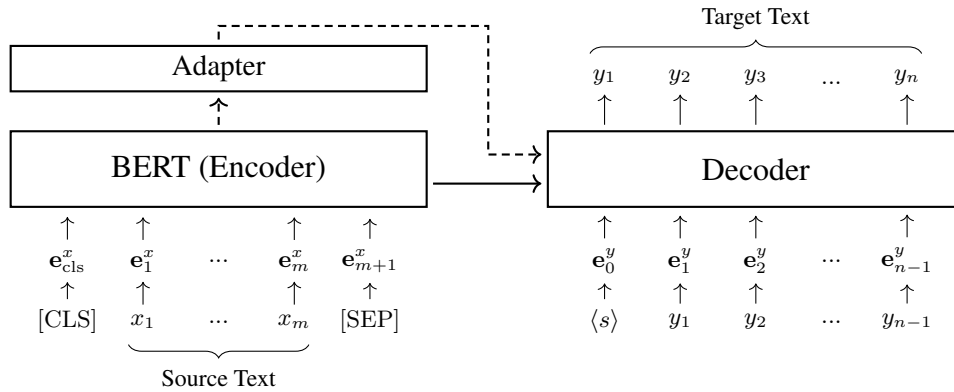
我们将查询文本和上下文文本打包在一起以获得输入序列。预测网络仅应用于上下文文本的输出，在每个位置生成概率 p_j^{beg} 和 p_j^{end} 。损失可以通过计算整个上下文文本中两个模型的负对数似然之和来计算。

$$\text{Loss} = -\frac{1}{n} \sum_{j=1}^n (\log p_j^{\text{beg}} + \log p_j^{\text{end}}) \quad (1.24)$$

在测试时，我们通过以下方式寻找最佳跨度：

$$(\hat{j}_1, \hat{j}_2) = \arg \max_{1 \leq j_1 \leq j_2 \leq n} (\log p_{j_1}^{\text{beg}} + \log p_{j_2}^{\text{end}}) \quad (1.25)$$

- 编码器-解码器模型的编码。虽然本节的重点主要在于语言理解问题，但值得注意的是，BERT模型可以应用于更广泛的NLP任务。事实上，BERT模型可以用于所有需要编码文本的场景。我们尚未提及的一个应用是文本生成，这包括机器翻译、摘要、问答和对话生成等一系列任务。这些任务可以表述为序列到序列问题：我们使用编码器来表示源文本，使用解码器来生成相应的目标文本。应用BERT模型的一个直接方法是将其视为编码器。在微调之前，我们可以用预训练的BERT模型的参数初始化编码器的参数。然后，编码器-解码器模型可以像往常一样在文本对上进行微调。以下是一个应用BERT模型于源侧的神经机器翻译系统架构的示例。



这里 $x_1 \dots x_m$ 表示源序列， $y_1 \dots y_n$ 表示目标序列， $e_1^x \dots e_m^x$ 表示 $x_1 \dots x_m$ 的嵌入序列，而 $e_1^y \dots e_n^y$ 表示 $y_1 \dots y_n$ 的嵌入序列。适配器是可选的，它将 BERT 模型的输出映射到更适合解码器的形式。

微调BERT模型是一个复杂的工程问题，受许多因素影响，例如微调数据量、模型大小以及微调中使用的优化器。一般来说，我们希望充分微调这些模型，以便它们在下游任务中表现良好。然而，针对特定任务的BERT模型微调可能会导致过拟合，

这反过来又降低了它们泛化到其他任务的能力。例如，假设我们有一个在特定任务上表现良好的BERT模型。如果我们随后对其进行微调以适应新任务，这可能会降低其在原始任务上的性能。这个问题与持续训练中的灾难性遗忘问题相关，其中神经网络在用新样本更新时忘记了之前学习到的信息。在实际应用中，缓解灾难性遗忘的一种常见方法是在微调时添加一些旧数据，并使用更多样化的数据来训练模型。此外，还可以使用针对灾难性遗忘专门的方法，例如经验重放 [Rolnick等人, 2019] 和弹性权重巩固 [Kirkpatrick等人, 2017]。感兴趣的读者可以参考一些综述，以了解更多关于持续学习中这一问题的详细讨论 [Parisi等人, 2019; Wang等人, 2023a; e]。

1.5 摘要

在这一章中，我们讨论了NLP中预训练的一般思想。特别是，我们讨论了自监督预训练及其在仅编码器、仅解码器和编码器-解码器架构中的应用。此外，我们还介绍了并比较了这些架构的各种预训练任务。例如，BERT被用来说明如何通过掩码语言模型对序列模型进行预训练，并将其应用于不同的下游任务。

近年来，在自然语言处理（NLP）领域取得了显著的进展，这得益于大规模的自监督预训练的应用。在许多任务上，包括NLP、计算机视觉和其他人工智能领域，都取得了巨大的进步。这些进步背后的一个想法是，通过在大量未标记数据上对这些人工智能系统进行训练，可以学习到大量关于世界的知识。例如，语言模型可以通过在大规模文本中反复预测掩码词来学习一些语言的一般知识。因此，这个预训练的语言模型可以作为基础模型，可以轻松地适应解决特定的下游NLP任务。这种NLP范式转变使得开发出理解、生成和推理方面极其强大的系统成为可能[Manning, 2022]。然而，重要的是要认识到我们仍然处于创建真正智能系统的早期阶段，还有很长的路要走。尽管如此，大规模预训练为研究人员长期以来渴望开发的智能系统打开了一扇门，尽管有几个关键的研究领域仍然有待探索，例如使用合理大小的数据高效地学习智能以及获得复杂的推理和规划能力。

注意，本章主要介绍性，无法涵盖预训练的所有方面。例如，有许多方法可以微调预训练模型，提供不同的方式以更好地适应各种情况。此外，大型语言模型，被认为是近年来人工智能最重大的成就之一，在本节中省略。我们将这些主题的讨论留到下一章。

CHAPTER 2

生成模型

近年来，NLP最显著的进步之一可能是大型语言模型（LLMs）的发展。这有助于创建能够理解和生成自然语言如人类的系统。这些系统甚至被发现能够进行推理，这被认为是一个非常具有挑战性的AI问题。随着这些成就，NLP取得了重大进展，进入了一个新的研究时代，其中正在解决困难问题，例如构建能够与人类顺畅沟通的对话系统。

语言建模的概念或概率语言建模可以追溯到Shannon在1951年进行的早期实验。在他的工作中，设计了一种语言模型来估计英语的可预测性——

how well can the next letter of a text be predicted when the preceding N letters are known.

尽管Shannon的实验是初步的，但自那时以来，语言建模的基本目标和方法在很大程度上保持不变。在相当长的一段时间里，特别是在2010年之前，语言建模的主要方法是 n -gram方法[Jurafsky和Martin, 2008]。在 n -gram语言建模中，我们估计一个词在它的前 $n - 1$ 个词的条件下的概率，因此序列的概率可以通过一系列 n -gram概率的乘积来近似。这些概率通常通过收集文本中 n -gram的平滑相对计数来估计。虽然这种方法简单直接，但它已经在NLP中得到广泛的应用。例如，现代统计语音识别和机器翻译系统的成功在很大程度上依赖于 n -gram语言模型[Jelinek, 1998; Koehn, 2010]。

将神经网络应用于语言建模一直很有吸引力，但随着深度学习技术的进步，真正的突破出现了。一个广为人知的例子是Bengio等人[2003]的研究，其中通过前馈网络对 n -gram概率进行建模，并通过端到端训练网络来学习。这种神经语言模型的副产品是单词的分布式表示，称为词嵌入。词嵌入不是将单词表示为离散变量，而是将单词映射到低维实值向量中，使得在连续表示空间中计算单词和单词 n -gram的意义成为可能。因此，语言模型不再受到维度诅咒的负担，但可以通过紧凑且密集的神经网络表示指数级的 n -gram。

通过神经语言模型学习词表示的想法激发了NLP中表征学习后续研究。然而，在提出后的前几年，这种方法并没有吸引到在NLP系统中开发的大量兴趣。然而，从大约2012年开始，通过简单的词预测任务从大规模文本中学习词嵌入方面取得了进展。提出了几种方法，如Word2Vec，以有效地学习此类嵌入，这些嵌入随后在多种NLP系统中得到了成功应用[Mikolov等人, 2013a; b]。由于这些进展，研究人员开始考虑使用更强大的语言模型，如基于LSTM的模型[Sutskever等人, 2014; Peters等人, 2018]，来学习序列的表示。在Transformer被提出后，对序列表示的进一步进展和兴趣爆炸式增长。随着Transformer的兴起，语言模型的概念被推广到包括学习以各种方式预测单词的模型。许多

强大的基于Transformer的模型使用这些词预测任务进行预训练，并成功应用于各种下游任务[Devlin等人，2019]。

的确，在大型数据上训练语言模型将NLP研究带入了激动人心的时代。虽然语言建模长期以来被视为一种基础技术，与研究人员所期望的人工智能目标没有直接联系，但它帮助我们看到了智能系统的出现，这些系统能够通过反复预测文本中的单词来学习一定程度的通用知识。最近的研究表明，单个经过良好训练的LLM可以处理大量任务，并且只需少量调整就能泛化到执行新任务[Bubeck等人，2023]。这表明了向更高级形式的人工智能迈进的一步，并激励人们进一步探索开发更强大的语言模型作为基础模型。

在这一章中，我们考虑生成式LLMs的基本概念。为了简单起见，我们使用术语 *large language models* 或 *LLMs* 来指代GPT等生成模型，尽管这个术语可以广泛涵盖BERT等其他类型的模型。我们首先对LLMs进行一般介绍，包括构建此类模型的关键步骤。然后，我们讨论LLMs的两个扩展问题：LLMs如何进行大规模训练，以及如何改进LLMs以处理非常长的文本。最后，我们对这些讨论进行总结。

2.1 LLMs的简要介绍

本节中，我们介绍了LLMs的基本思想，这是本章及以下章节所必需的。我们将交替使用术语 *word* 和 *token*。尽管它们的原始含义不同，但它们都指的是语言模型中使用的基本单元。

在介绍细节之前，让我们首先考虑语言模型是如何工作的。语言建模的目标是预测一系列标记发生的概率。设 $\{x_0, x_1, \dots, x_m\}$ 为一系列标记，其中 x_0 是起始符号 $\langle s \rangle$ (或 $\langle \text{SOS} \rangle$)¹。这个序列的概率可以使用链式法则定义

$$\begin{aligned} \Pr(x_0, \dots, x_m) &= \Pr(x_0) \cdot \Pr(x_1|x_0) \cdot \Pr(x_2|x_0, x_1) \cdots \Pr(x_m|x_0, \dots, x_{m-1}) \\ &= \prod_{i=0}^m \Pr(x_i|x_0, \dots, x_{i-1}) \end{aligned} \quad (2.1)$$

或者以对数形式表示

$$\log \Pr(x_0, \dots, x_m) = \sum_{i=0}^m \log \Pr(x_i|x_0, \dots, x_{i-1}) \quad (2.2)$$

这里 $\Pr(x_i|x_0, \dots, x_{i-1})$ 是在所有先前标记 $\{x_0, \dots, x_{i-1}\}$ ² 的条件下标记 x_i 的概率。在深度学习的时代，一种典型的语言建模方法是估计这个

¹The start symbol can also be $[\text{CLS}]$ following BERT models.

²We assume that when $i = 0$, $\Pr(x_i|x_0, \dots, x_{i-1}) = \Pr(x_0) = 1$. Hence $\Pr(x_0, \dots, x_m) = \Pr(x_0) \Pr(x_1, \dots, x_m|x_0) = \Pr(x_1, \dots, x_m|x_0)$.

Context	Predict	Decision Rule	Sequence Probability
$\langle s \rangle a$	b	$\arg \max_{x_2 \in V} \Pr(x_2 \langle s \rangle a)$	$\Pr(\langle s \rangle) \cdot \Pr(a \langle s \rangle) \cdot \Pr(b \langle s \rangle a)$
$\langle s \rangle a b$	c	$\arg \max_{x_3 \in V} \Pr(x_3 \langle s \rangle a b)$	$\Pr(\langle s \rangle) \cdot \Pr(a \langle s \rangle) \cdot \Pr(b \langle s \rangle a) \cdot \Pr(c \langle s \rangle a b)$
$\langle s \rangle a b c$	d	$\arg \max_{x_4 \in V} \Pr(x_4 \langle s \rangle a b c)$	$\Pr(\langle s \rangle) \cdot \Pr(a \langle s \rangle) \cdot \Pr(b \langle s \rangle a) \cdot \Pr(c \langle s \rangle a b) \cdot \Pr(d \langle s \rangle a b c)$

表2.1: 通过语言模型生成给定前缀 $\langle s \rangle a$ 的三个标记 $b c d$ 的说明。在每一步中, 模型从 V 中选择一个标记 x_i , 以使 $\Pr(x_i | x_0, \dots, x_{i-1})$ 最大化。然后将此标记附加到上下文序列的末尾。在下一步中, 我们重复相同的过程, 但基于新的上下文。

使用深度神经网络计算概率。训练用于完成此任务的神经网络接收一个标记序列 x_0, \dots, x_{i-1} 并生成一个表示词汇表 V (的分布, 记为 $\Pr(\cdot | x_0, \dots, x_{i-1})$)。概率 $\Pr(x_i | x_0, \dots, x_{i-1})$ 是 $\Pr(\cdot | x_0, \dots, x_{i-1})$ 的第 i 个条目的值。

当应用训练好的语言模型时, 一个常见任务是找到给定其先前上下文标记的最可能标记。这个标记预测任务可以描述为

$$\hat{x}_i = \arg \max_{x_i \in V} \Pr(x_i | x_0, \dots, x_{i-1}) \quad (2.3)$$

我们可以多次执行单词预测以生成连续文本: 每次我们预测最佳标记 \hat{x}_i , 然后将此预测标记添加到上下文中以预测下一个标记 \hat{x}_{i+1} 。这导致从左到右的生成过程实现公式 (2.1) 和 (2.2)。为了说明, 考虑在给定前缀‘ $\langle s \rangle a$ ’的情况下生成以下三个单词, 如表2.1所示。现在我们讨论LLMs是如何构建、训练和应用的。

2.1.1 仅解码器 Transformer

按照标准做法, 语言模型的输入是一系列标记 (用 $\{x_0, \dots, x_{m-1}\}$ 表示)。对于每一步, 生成一个输出标记, 将序列向前移动一个位置以进行下一个预测。为此, 语言模型在每个位置 i 输出一个分布 $\Pr(\cdot | x_0, \dots, x_{i-1})$, 并根据此分布选择标记 x_i 。该模型通过最大化对数似然 $\sum_{i=1}^m \log \Pr(x_i | x_0, \dots, x_{i-1})$ ³进行训练。

这里, 我们关注仅解码器的 Transformer 架构, 因为它是最受欢迎的 LLM 模型架构之一。标记序列的输入由一系列 d_e -维向量 $\{e_0, \dots, e_{m-1}\}$ 表示。 e_i 是 x_i 的标记嵌入和 i 的位置嵌入之和。模型的主要部分是一系列 Transformer 块 (或层)。每个 Transformer 块有两个堆叠的子层, 一个用于自注意力建模, 一个用于 FFN 建模。这些子层可以使用后规范架构定义

$$\text{output} = \text{LNorm}(F(\text{input}) + \text{input}) \quad (2.4)$$

³Note that $\sum_{i=1}^m \log \Pr(x_i | x_0, \dots, x_{i-1}) = \sum_{i=0}^m \log \Pr(x_i | x_0, \dots, x_{i-1})$ since $\log \Pr(x_0) = 0$.

或预范架构

$$\text{output} = \text{LNorm}(F(\text{input})) + \text{input} \quad (2.5)$$

输入和输出表示输入和输出，两者均为 $m \times d$ 矩阵。输入和输出的 i -th 行可以看作是序列中 i -th 令牌的上文表示。

$F(\cdot)$ 是子层的核心函数。对于FFN子层， $F(\cdot)$ 是一个多层FFN。对于自注意力子层， $F(\cdot)$ 是一个多头自注意力函数。一般来说，自注意力以QKV注意力的形式表示

$$\text{Att}_{\text{qkv}}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d}} + \mathbf{Mask}\right)\mathbf{V} \quad (2.6)$$

在 \mathbf{Q} 、 \mathbf{K} 和 $\mathbf{V} \in \mathbb{R}^{m \times d}$ 分别代表查询、键和值的情况下，请注意，在预测一个标记时，只考虑之前的标记。因此，将一个掩码变量 $\mathbf{Mask} \in \mathbb{R}^{m \times m}$ 纳入自注意力机制以实现这一点。当 $i \leq k$ 时， $\mathbf{Mask}(i, k)$ 的值为0，否则为 $-\infty$ 。

给定一个表示 $\mathbf{H} \in \mathbb{R}^{m \times d}$ ，多头自注意力函数可以定义为

$$F(\mathbf{H}) = \text{Merge}(\text{head}_1, \dots, \text{head}_\tau) \mathbf{W}^{\text{head}} \quad (2.7)$$

where $\text{Merge}(\{\mathbf{v}^*\})$ 表示其输入的连接， $\mathbf{W}^{\text{head}} \in \mathbb{R}^{d \times d}$ 表示一个参数矩阵。 head_j 是QKV注意力在表示子空间上的输出

$$\text{head}_j = \text{Att}_{\text{qkv}}(\mathbf{Q}^{[j]}, \mathbf{K}^{[j]}, \mathbf{V}^{[j]}) \quad (2.8)$$

$\mathbf{Q}^{[j]}$ 、 $\mathbf{K}^{[j]}$ 和 $\mathbf{V}^{[j]}$ 是投影到第 j -个子空间的查询、键和值

$$\mathbf{Q}^{[j]} = \mathbf{H} \mathbf{W}_j^q \quad (2.9)$$

$$\mathbf{K}^{[j]} = \mathbf{H} \mathbf{W}_j^k \quad (2.10)$$

$$\mathbf{V}^{[j]} = \mathbf{H} \mathbf{W}_j^v \quad (2.11)$$

在 \mathbf{W}_j^q 、 \mathbf{W}_j^k 和 $\mathbf{W}_j^v \in \mathbb{R}^{d \times \frac{d}{\tau}}$ 是变换的参数矩阵。

假设我们拥有 L 个 Transformer 块。在最后一个块的输出之上构建了一个 Softmax 层。Softmax 层输出一个关于词汇表的 m 个分布序列，如下所示

$$\begin{bmatrix} \Pr(\cdot | x_0, \dots, x_{m-1}) \\ \vdots \\ \Pr(\cdot | x_0, x_1) \\ \Pr(\cdot | x_0) \end{bmatrix} = \text{Softmax}(\mathbf{H}^L \mathbf{W}^o) \quad (2.12)$$

\mathbf{H}^L 是最后一个Transformer块的输出， $\mathbf{W}^o \in \mathbb{R}^{d \times |V|}$ 是参数矩阵。

图2.1显示了Transformer 架构用于语言建模。应用 g 这种语言

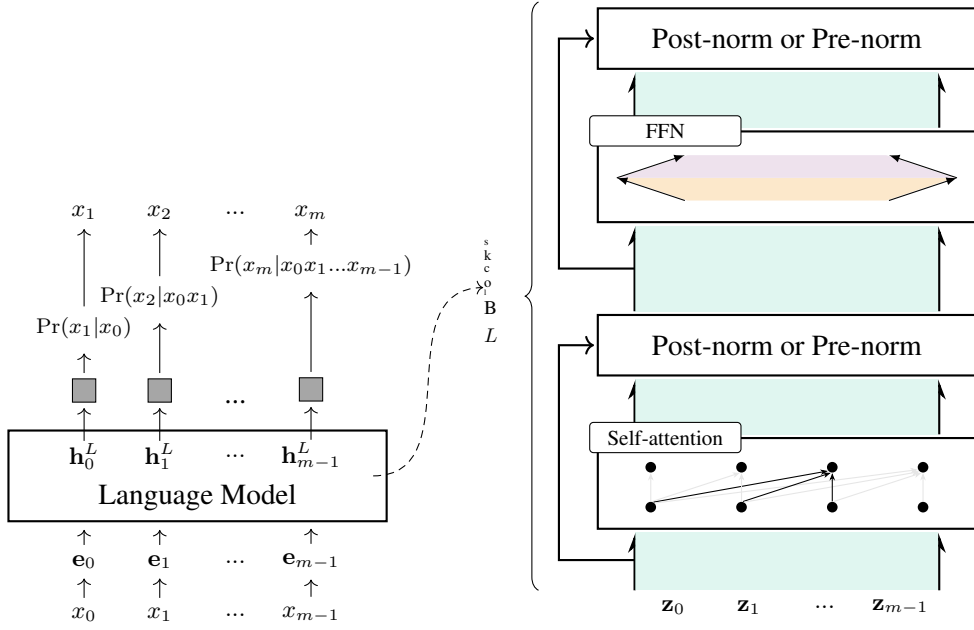


图2.1: 用于语言模型的Transformer解码器架构。中心组件是 L 个堆叠的Transformer块，每个块包含一个自注意力子层和一个FFN子层。为了防止模型访问右上下文，将掩码变量纳入自注意力中。输出层使用Softmax函数根据先前令牌的序列生成下一个令牌的概率分布。在推理过程中，模型使用先前预测的令牌来预测下一个令牌，重复此过程，直到达到序列的末尾。 $\{z_0, \dots, z_{m-1}\}$ 表示Transformer块的输入， $\{h_0^L, \dots, h_{m-1}^L\}$ 表示最后一个Transformer块的输出。

模型遵循自回归过程。每次语言模型将标记 x_{i-1} 作为输入，并预测一个最大化概率 $\Pr(x_i | x_0, \dots, x_{i-1})$ 的标记 x_i 。需要注意的是，尽管实现细节不同，许多大型语言模型都共享上述相同的架构。这些模型被称为大型，因为它们的深度和宽度都很重要。表2.2显示了几个大型语言模型的大小以及它们的模型设置。

2.1.2 训练 LLMs

现在假设我们给定一个包含 K 个序列的训练集 \mathcal{D} 。可以使用语言模型计算 \mathcal{D} 中每个序列 $\mathbf{x} = x_0 \dots x_m$ 的对数似然。

$$\mathcal{L}_\theta(\mathbf{x}) = \sum_{i=1}^m \log \Pr_\theta(x_i | x_0, \dots, x_{i-1}) \quad (2.13)$$

这里下标 θ 附属于 $\mathcal{L}(\cdot)$ 和 $\Pr(\cdot)$ 表示语言模型的参数。然后，最大似然训练的目标被定义为

$$\hat{\theta} = \arg \max_{\theta} \sum_{\mathbf{x} \in \mathcal{D}} \mathcal{L}_\theta(\mathbf{x}) \quad (2.14)$$

训练基于Transformer的语言模型以达到上述目标是通常被视为神经网络的标准优化过程。这可以通过使用梯度下降算法实现，这些算法得到了现成深度学习工具包的广泛支持。某种程度上

LLM	# of Parameters	Depth L	Width d	# of Heads (Q/KV)
GPT-1 [Radford et al., 2018]	0.117B	12	768	12/12
GPT-2 [Radford et al., 2019]	1.5B	48	1,600	25/25
GPT-3 [Brown et al., 2020]	175B	96	12,288	96/96
LLaMA2 [Touvron et al., 2023b]	7B	32	4,096	32/32
	13B	40	5,120	40/40
	70B	80	8,192	64/64
LLaMA3/3.1 [Dubey et al., 2024]	8B	32	4,096	32/8
	70B	80	8,192	64/8
	405B	126	16,384	128/8
Gemma2 [Team et al., 2024]	2B	26	2,304	8/4
	9B	42	3,584	16/8
	37B	46	4,608	32/16
Qwen2.5 [Yang et al., 2024]	0.5B	24	896	14/2
	7B	28	3,584	28/4
	72B	80	8,192	64/8
DeepSeek-V3 [Liu et al., 2024a]	671B	61	7,168	128/128
Falcon [Penedo et al., 2023]	7B	32	4,544	71/71
	40B	60	8,192	128/128
	180B	80	14,848	232/232
Mistral [Jiang et al., 2023a]	7B	32	4,096	32/32

表2.2: 一些LLM在模型大小、模型深度、模型宽度和头数方面的比较 (a/b 表示 a 个头用于查询和 b 个头用于键和值)。

令人惊讶的是，随着语言模型演变为计算量更大的模型并在更大的数据集上训练，持续产生了更好的结果 [Kaplan 等人, 2020]。这些成功促使自然语言处理研究人员继续增加训练数据和模型大小，以构建更强大的语言模型。

然而，随着语言模型变得更大，我们面临着新的训练挑战，与相对较小的模型训练相比，这些问题显著改变了问题。其中一项挑战源于需要大规模分布式系统来管理数据、模型参数、训练程序等。开发和维护这样的系统需要在软件和硬件工程方面投入大量工作，以及深度学习方面的专业知识。相关问题是，当训练规模扩大时，我们需要更多的计算资源来确保训练过程可以在可接受的时间内完成。例如，通常需要数百或数千个GPU从头开始训练具有数十亿参数的LLM。这一需求极大地增加了训练此类模型的成本，特别是考虑到这些模型在开发过程中需要多次训练运行。此外，从深度学习的角度来看，如果神经网络非常深或模型规模非常大，训练过程可能会变得不稳定。作为回应，我们通常需要修改模型架构以适应LLMs的大规模训练。在第2.2节中，我们将对这些问题进行更多讨论。

2.1.3 LLMs的微调

一旦我们预训练了一个LLM，我们就可以将其应用于执行各种NLP任务。传统上，语言模型被用作其他系统的组件，例如，它们在统计机器翻译系统中被广泛用于评估翻译。相比之下，在生成式AI中，LLMs被视为完整的系统，并利用它们的生成特性来解决NLP问题。一种常见的方法是用文本描述我们想要解决的问题，然后提示LLMs根据这种描述生成文本。这是一个标准的文本生成任务，我们从给定的上下文继续或完成文本。

更正式地说，令 $\mathbf{x} = x_0 \dots x_m$ 表示由用户提供的上下文标记序列， $\mathbf{y} = y_1 \dots y_n$ 表示上下文之后的标记序列。然后，LLM的推理可以定义为根据 \mathbf{x} 找到最可能的序列 \mathbf{y} 的问题：

$$\begin{aligned}\hat{\mathbf{y}} &= \arg \max_{\mathbf{y}} \log \Pr(\mathbf{y}|\mathbf{x}) \\ &= \arg \max_{\mathbf{y}} \sum_{i=1}^n \log \Pr(y_i|x_0, \dots, x_m, y_1, \dots, y_{i-1})\end{aligned}\quad (2.15)$$

这里 $\sum_{i=1}^n \log \Pr(y_i|x_0, \dots, x_m, y_1, \dots, y_{i-1})$ 实质上与公式 (2.2) 的右侧表达相同的内容。它模拟了从位置 $m+1$ 预测标记的对数概率，而不是位置 0。在本章及随后的章节中，我们将使用不同的变量 \mathbf{x} 和 \mathbf{y} 来区分 LLM 的输入和输出，尽管它们可以被视为同一序列的子序列。采用这种符号，我们可以看到上述方程的形式与其他 NLP 中的文本生成模型（如神经机器翻译模型）中使用的符号非常相似。

为了说明LLMs的应用，考虑确定给定句子语法正确性的问题。我们可以定义一个如下模板

{句子*} 问题：这个句子语法正确吗？ 答案：

这里表示我们打算生成的文本。{*sentence*}是一个占位符变量，将被用户提供的实际句子所替换。例如，假设我们有一个句子“*John seems happy today.*”。我们可以在模板中将{*sentence*}替换为这个句子，以便为语言模型提供输入

约翰今天看起来很高兴。问题：这个句子语法正确吗？ 回答：

要执行此任务，语言模型被赋予了上下文 $\mathbf{x} = \text{“John今天看起来很高兴.”}$ 问题：这个句子语法正确吗？ 答案： ”⁴。然后它生成了以下

⁴\n is a special character used for line breaks.

文本作为答案，基于上下文。例如，语言模型可能会输出“是”（即 $y = \text{“是”}$ ），如果这段文本是在给定上下文下预测概率最大的一个。

同样，我们可以定义更多模板来处理其他任务。例如，我们可以使用以下模板将英语句子翻译成中文

{句子*} 问题：这个英文句子的中文翻译是什么？ 答案：

或使用类似指令的模板

{句子*}
将此句子从英语翻译成中文。

或使用类似代码的模板。

[源语言] = 英语 [目标语言] = 中文 [输入] = { *句子* } [输出]
= _____

以上模板提供了一种简单但有效的方法，通过不调整模型结构来“提示”单个LLM执行各种任务。然而，这种方法要求LLM能够识别并遵循指令或问题。实现这一目标的一种方法是将带有指令及其相应响应的训练样本纳入预训练数据集。虽然这种方法很简单，但从头开始构建和训练LLM在计算上非常昂贵。此外，要使遵循指令的数据对预训练有效，需要大量的此类数据，但收集所有感兴趣任务的大规模标记数据非常困难。

第二种方法，在最近的研究中已成为事实上的标准，是通过微调来适应LLMs。因此，在预训练阶段学习的标记预测能力可以推广以完成新任务。微调背后的思想是，在预训练中已经获得了某些语言的一般知识，但我们需要一种机制来激活这些知识以应用于新任务。为了实现这一点，我们可以使用指令遵循数据略微微调模型参数。这种方法被称为指令微调。

一个指令微调样本，由一系列标记组成，可以看作是一个由输入和期望输出组成的元组。在这里，输入包括指令、系统信息（或系统前缀）以及任何其他用户提供的信息⁵。为了说明，考虑以下示例（蓝色文本=输入和下划线文本=输出）。

⁵System information refers to a sequence of tokens added at the beginning of an input in order to guide the behavior of an LLM, such as, *you are a helpful assistant and should not output toxic content*.

阅读以下句子，并回答其语法是否正确。LLMs是强大的模型，但构建成本高昂。是

—

LLMs是强大的模型，但构建成本高昂。这句话在语法上通顺吗？答案选项 - 是 - 否 是

—

是或否？是 = 以下句子语法正确。否 = 它包含语法错误。LLMs 是强大的模型，但构建成本高昂。是

—

所有这些样本描述的是相同的二进制分类任务，但具有不同的指令。为了增加微调数据的多样性并使LLMs泛化，我们可以使用指令定义更多任务。

写一个关于三个角色的短故事。角色1
：一个名叫Andy的男孩。角色2：And
y的狗。角色3：Andy的朋友，Jane。

在繁华城市的中心，安迪、他忠诚的狗火箭和他的朋友简在黄昏的微光下发现了一个隐藏的花园。被古老的树木和低语的风包围，他们分享梦想和秘密，锻造了坚不可摧的纽带。随着夜晚的加深，他们在星光下立下誓言，永远在平凡中发现魔法。一起，他们意识到真正的冒险在于与朋友共度的时光，每一天都是等待被发现的宝藏。

—

Show me a recipe for making ice cream.

Ingredients:

2 cups heavy cream; 1 cup milk; 3/4 cup sugar; 2 tsp vanilla extract;

Pinch of salt.

Steps:

1. Whisk all ingredients until sugar dissolves.

2. Chill mixture for 2 hours in the fridge.

3. Freeze in a shallow dish, stirring every 30 minutes, 3-4 times.

4. Freeze solid for 2 hours.

5. Serve your homemade ice cream.

如果你购买5个苹果，每个苹果的价格是1.20美元，你总共花费多少钱？6.00美元

编写一个Python程序来计算以下数字的平方和。1, 2, 10, -9, 78

数字 = [1,2,10,-9,78] 平方和 = sum(x**2 for
x in numbers) 打印(平方和)

要获得遵循指令的能力，需要一定量的微调数据。这些数据可能包括各种指令和可能的响应。研究发现，增加微调任务的数量有助于提高LLMs的性能[Chung等人，2022]。请注意，尽管更多的微调数据是有利的，但这类数据的量通常比预训练数据小几个数量级。例如，LLMs可以通过成千上万的样本进行微调，或者如果这些样本质量高，甚至更少[Zhou等人，2023a；Chen等人，2023b]，而预训练这些模型可能需要数十亿或数千亿个标记，从而导致显著更大的计算需求和更长的训练时间[Touvron等人，2023a]。

它也值得注意，我们不应期望微调数据涵盖我们打算应用LLMs的所有下游任务。对预训练+微调方法工作原理的普遍理解是，LLMs在预训练阶段获得了理解指令和生成响应的知识。然而，这些能力直到我们引入某种形式的监督才完全激活。当我们用相对少量的标记数据微调模型时，就会出现一般的指令遵循行为。因此，我们可以实现一定程度的零样本学习：微调后的模型可以处理他们未明确训练或微调的新任务[Sanh等人，2022；Wei等人，2022a]。这种零样本学习能力将生成型LLMs与BERT等早期预训练模型区分开来，后者主要是针对特定任务进行微调的。

一旦我们准备了一个由指令描述的数据集合，微调过程相对简单。这个过程可以被视为一个标准的预训练训练过程，但是在一个规模小得多的训练数据集上。设 $\mathcal{D}_{\text{tune}}$ 为微调数据集， $\hat{\theta}$ 为模型参数

通过预训练优化。我们可以修改公式 (2.14) 以获得微调的目标

$$\tilde{\theta} = \arg \max_{\hat{\theta}^+} \sum_{\text{sample} \in \mathcal{D}_{\text{tune}}} \mathcal{L}_{\hat{\theta}^+}(\text{sample}) \quad (2.16)$$

这里 $\tilde{\theta}$ 表示最优参数。使用符号 $\hat{\theta}^+$ 表示微调从预训练参数 $\hat{\theta}$ 开始。

对于每个样本 $\in \mathcal{D}_{\text{tune}}$ ，我们将其分为一个输入段 $\mathbf{x}_{\text{sample}}$ 和一个输出段 $\mathbf{y}_{\text{sample}}$ ，即，

$$\text{sample} = [\mathbf{y}_{\text{sample}}, \mathbf{x}_{\text{sample}}] \quad (2.17)$$

我们随后定义损失函数为

$$\mathcal{L}_{\hat{\theta}^+}(\text{sample}) = -\log \Pr_{\hat{\theta}^+}(\mathbf{y}_{\text{sample}} | \mathbf{x}_{\text{sample}}) \quad (2.18)$$

换句话说，我们计算子序列 $\mathbf{y}_{\text{sample}}$ 的损失，而不是整个序列的损失。在实际实现该方程的反向传播时，通常在正向传播中构建序列 $[\mathbf{y}_{\text{sample}}, \mathbf{x}_{\text{sample}}]$ 。然而，在反向传播中，错误梯度仅通过网络中对应于 $\mathbf{y}_{\text{sample}}$ 的部分传播回，其余网络保持不变。例如，考虑一个序列

$$\underbrace{\langle s \rangle \text{ Square this number . 2 .}}_{\text{Context (Input)}} \quad \underbrace{\text{The result is 4 .}}_{\text{Prediction (Output)}}$$

损失仅对结果进行计算和反向传播，结果为4。_____

指令微调也需要大量的工程工作。为了达到令人满意的结果，可以尝试不同的学习率、批量大小、微调步数等设置。这通常需要许多微调运行和评估。尽管微调的成本和实验努力远低于预训练阶段，但它们仍然至关重要，不应被忽视。

虽然我们在这里专注于指令微调的示例，但微调技术在各种LLM的开发中起着重要作用，并且应用更为广泛。例如，使用对话数据进行LLM的微调作为聊天机器人，以及将这些模型调整为处理非常长的序列。微调技术的广泛应用促使研究人员改进这些技术，例如设计更高效的微调算法。虽然微调的研究富有成效，但在本节中我们仅介绍涉及的关键步骤。在接下来的章节中，我们将对此主题进行更详细的讨论。

2.1.4 将LLMs与世界对齐

指令微调提供了一种简单的方法来适应可以很好地定义的任务。这个问题可以广泛地被归类为一个对齐问题。在这里，对齐是指引导LLMs以与人类意图一致的方式行事的过程。指导可以来自标记数据、人类反馈或任何其他形式的人类偏好。例如，

我们希望LLMs不仅能在遵循指令方面准确无误，而且还要无偏见、真实、无害。因此，我们需要监督模型向人类价值观和期望靠拢。一个常见的例子是，当我们询问一个LLM如何制造武器时，如果它没有仔细对齐，它可能会提供一个关键步骤列表。然而，一个负责任的模型应该能够识别并避免回应有关有害或非法信息的请求。在这种情况下，对齐对于确保LLMs负责任地行动并符合道德准则至关重要。

与对齐相关的概念是人工智能安全。人工智能的一个最终目标是构建既安全又对社会有益的智能系统。为了实现这一目标，我们应该使这些系统在任何现实世界使用条件下都保持稳健、安全、客观，即使在误用或不利使用的情况下。对于LLMs，可以通过将它们与适当的人类指导（如人类标注数据和与应用用户互动）对齐来提高其安全性。

对齐困难，因为人类价值观和期望是多样化和变化的。有时，除非我们看到LLM对用户请求的反应，否则很难精确描述人类想要什么。这使得对齐不再是在预定义任务上调整LLM的问题，而是一个更大的问题，即通过与世界互动来训练它们。

由于对控制AI系统的担忧，关于LLMs的对齐问题研究激增。通常，在LLMs在大规模未标记数据上预训练后，会采用两个对齐步骤。

- 监督微调（SFT）。这涉及到在新的、任务导向的、标记的数据上继续训练预训练的LLM。常用的SFT技术是指令微调。如前一小节所述，通过从指令-响应注释数据中学习，LLM可以与遵循指令的预期行为保持一致，从而能够执行各种指令描述的任务。监督微调可以看作是遵循预训练 + 微调范式，并提供了一种相对直接的方法来适应LLM。
- 从人类反馈中学习。在LLM完成预训练和监督微调后，如果得到适当的提示，它可以用来响应用户请求。但这个模型可能会生成不真实、有偏见或有害的内容。为了使LLM更符合用户的需求，一种简单的方法是直接从人类反馈中学习。例如，给定用户提供的某些指令和输入，专家被要求评估模型根据他们的偏好和兴趣做出响应的效果。然后，使用这些反馈进一步训练LLM以实现更好的对齐。

一种从人类反馈中学习的方法是将它视为强化学习（RL）问题，称为基于人类反馈的强化学习（RLHF）[Ouyang等，2022]。RLHF方法最初被提出用于解决一般的序列决策问题[Christiano等，2017]，后来在GPT系列模型的发展中得到了成功应用[Stiennon等，2020]。作为一种强化学习方法，RLHF的目标是通过最大化环境中的某些奖励来学习策略。具体来说，RLHF中构建了两个组件：

- 代理。代理，也称为LM代理，是我们想要训练的LLM。该代理通过与其环境交互来运行：它从环境中接收文本并

输出另一个发送回环境的文本。代理的策略是由LLM定义的函数，即 $\Pr(y|x)$ 。

- 奖励模型。奖励模型是环境的代理。每当智能体产生一个输出序列时，奖励模型都会为这个输出序列分配一个数值分数（即奖励）。这个分数告诉智能体输出序列有多好。

在RLHF中，我们需要执行两个学习任务：1) 奖励模型学习，涉及使用人类对代理输出的反馈来训练奖励模型，2) 策略学习，涉及使用强化学习算法优化由奖励模型指导的策略。以下是RLHF中涉及的关键步骤的简要概述。

- 构建一个使用预训练和指令微调的初始策略。
- 使用策略为每个输入生成多个输出，然后收集对这些输出的人类反馈（例如，输出之间的比较）。
- 从人类反馈中学习奖励模型。
- 微调策略，以奖励模型的监督进行。

图2.2展示了RLHF的概述。鉴于本节仅作为对LLM概念的简要介绍，将不包括RLHF技术的详细讨论。我们反而用一个简单的例子来说明RLHF背后的基本思想。

假设我们已经通过预训练和指令微调训练了一个LLM。这个LLM被部署以响应用户的请求。例如，用户可能输入

如何过一种更环保的生活？

我们使用LLM通过采样输出空间生成4个不同的输出（用 $\{y_1, \dots, y_4\}$ 表示）

输出 1 (y_1): 考虑改用电动汽车或自行车代替传统汽车，以减少碳排放并保护我们的地球。输出 2 (y_2): 采用极简主义生活方式。拥有更少的财产以减少消费和制造及处置的环境影响。输出 3 (y_3): 脱离电网。自己产生可再生能源并收集雨水，以实现完全自给自足并减少对不可再生资源的依赖。输出 4 (y_4): 支持当地农产品，以减少运输食品的碳足迹，同时享受新鲜、健康的食物。

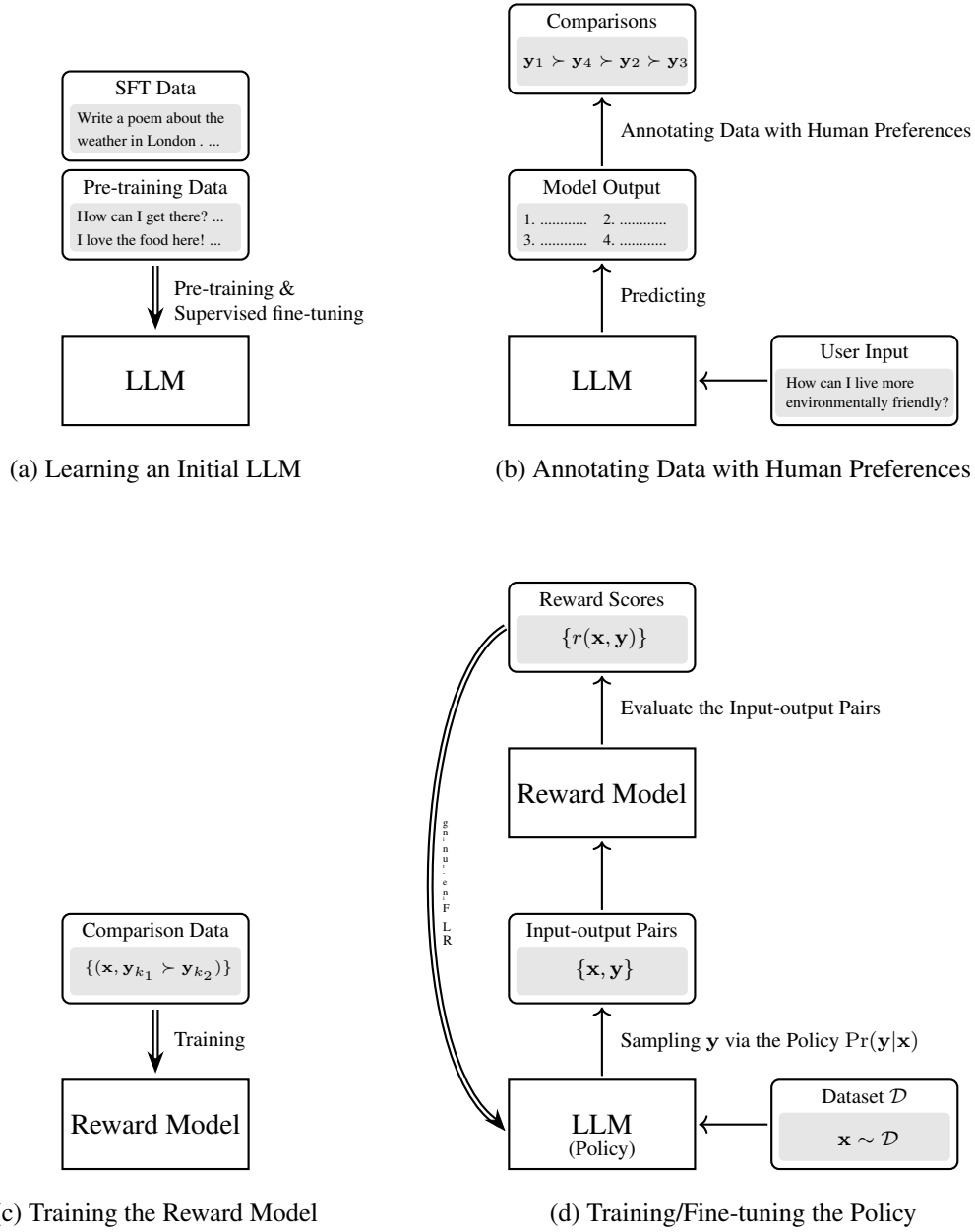


图2.2: RLHF概述。涉及4个关键步骤: a) 使用预训练和监督微调训练初始LLM (即策略); b) 通过对LLM的输出进行排名来收集人类偏好数据; c) 使用排名结果训练奖励模型; d) 基于奖励模型的政策RL微调。双箭头表示训练或微调。

我们随后要求标注员评估这些输出。一种简单直接的方法是为每个输出分配一个评分。在这种情况下, 奖励模型学习问题可以表述为训练回归模型的任务。但是, 为LLM的输出分配数值分数对标注员来说并不容易。通常很难设计一个所有标注员都能同意并容易遵循的标注标准。一种替代方法, 在LLM的开发中更为流行, 就是对这些输出进行排序。例如, 上述输出的一个可能排序是

$$y_1 \succ y_4 \succ y_2 \succ y_3$$

奖励模型随后使用此排名结果进行训练。在RLHF中，通常奖励模型是一个与目标LLM具有相同架构但模型规模较小的语言模型。给定输入 \mathbf{x} 和输出 \mathbf{y}_k ，我们将它们连接起来形成一个序列 $\text{seq}_k = [\mathbf{x}, \mathbf{y}_k]$ 。此序列从左到右使用强制解码进行处理。由于每个位置只能访问其左侧的语言模型上下文，因此第一个位置的最高Transformer层的输出不能用作序列的表示。相反，在序列末尾添加一个特殊符号（例如， $\langle \text{end} \rangle$ ），并将相应的Transformer层堆栈的输出视为整个序列的表示。在表示之上构建一个输出层，例如线性变换层，以生成奖励，表示为 $R(\text{seq}_k)$ 或 $R(\mathbf{x}, \mathbf{y}_k)$ 。

我们使用排名损失训练这个奖励模型。例如，成对排名损失函数可以写成以下形式

$$\text{Loss}_\omega(\mathcal{D}_r) = -\mathbb{E}_{(\mathbf{x}, \mathbf{y}_{k_1}, \mathbf{y}_{k_2}) \sim \mathcal{D}_r} \log(\text{Sigmoid}(R_\omega(\mathbf{x}, \mathbf{y}_{k_1}) - R_\omega(\mathbf{x}, \mathbf{y}_{k_2}))) \quad (2.19)$$

ω 代表奖励模型的参数， \mathcal{D}_r 代表一组输入和一对输出的元组。 $(\mathbf{x}, \mathbf{y}_{k_1}, \mathbf{y}_{k_2}) \sim \mathcal{D}_r$ 是一个采样操作，以一定的概率从 \mathcal{D}_r 中抽取一个样本 $(\mathbf{x}, \mathbf{y}_{k_1}, \mathbf{y}_{k_2})$ 。例如，假设我们首先以均匀分布抽取一个模型输入 \mathbf{x} ，然后以 $\mathbf{y}_{k_1} \succ \mathbf{y}_{k_2}$ 的概率抽取一对模型输出，给定 \mathbf{x} （表示为 $\Pr(\mathbf{y}_{k_1} \succ \mathbf{y}_{k_2} | \mathbf{x})$ ）。相应的损失函数由以下公式给出

$$\begin{aligned} \text{Loss}_\omega(\mathcal{D}_r) &= -\sum \Pr(\mathbf{x}) \cdot \Pr(\mathbf{y}_{k_1} \succ \mathbf{y}_{k_2} | \mathbf{x}) \cdot \log(\text{Sigmoid}(R_\omega(\mathbf{x}, \mathbf{y}_{k_1}) - R_\omega(\mathbf{x}, \mathbf{y}_{k_2}))) \\ &= -\frac{1}{K} \sum \Pr(\mathbf{y}_{k_1} \succ \mathbf{y}_{k_2} | \mathbf{x}) \cdot \log(\text{Sigmoid}(R_\omega(\mathbf{x}, \mathbf{y}_{k_1}) - R_\omega(\mathbf{x}, \mathbf{y}_{k_2}))) \end{aligned} \quad (2.20)$$

K 代表参与采样的模型输入数量。虽然这些函数的形式可能看起来复杂，但它们的想法很简单：如果两个输出的预测排名与人类标注的排名不同，我们就对模型进行惩罚。相比之下，如果预测排名与人类标注的排名匹配，模型将获得奖励。

我们可以通过最小化上述排名损失来训练奖励模型

$$\hat{\omega} = \arg \min_{\omega} \text{Loss}_\omega(\mathcal{D}_r) \quad (2.21)$$

结果模型 $R_{\hat{\omega}}(\cdot)$ 可以用于评估任何给定的输入和输出对。请注意，尽管奖励模型使用基于排名的目标进行训练，但它用于评分。这允许它提供连续的监督信号，这对训练其他模型非常有益。

我们现在转向策略学习问题。一个常用的目标是最大化一组输入输出对的奖励。遵循与公式（2.16）类似的形式，我们获得一个简单的RL微调训练目标

$$\tilde{\theta} = \arg \max_{\hat{\theta}^+} \mathbb{E}_{(\mathbf{x}, \mathbf{y}_{\hat{\theta}^+}) \sim \mathcal{D}_{\text{rlft}}} R_{\hat{\omega}}(\mathbf{x}, \mathbf{y}_{\hat{\theta}^+}) \quad (2.22)$$

在获得最佳参数 $\tilde{\theta}$ 通过微调预训练参数 $\hat{\theta}$ 的地方。 $\mathcal{D}_{\text{rlft}}$ 是

RL微调数据集。对于每个样本 $(x, y_{\hat{\theta}+})$ ， x 从准备好的输入序列数据集中采样，而 $y_{\hat{\theta}+}$ 从策略给出的分布 $\text{Pr}_{\hat{\theta}+}(y|x)$ 中采样。

在实践中，更高级的强化学习算法，如近端策略优化（PPO），常用于实现更稳定的训练以及更好的性能。我们将对强化学习算法的详细讨论留到本书的后续部分，其中广泛使用了RLHF以实现对齐。

一个有趣的问题在这里出现：为什么不将学习人类偏好视为一个标准的监督学习问题呢？这个问题与我们之前关于数据标注困难性的讨论密切相关。通常，描述人类价值观和目标具有挑战性，而对于人类来说，提供与标注数据高度一致的结果则更加困难。作为替代方案，标注一组模型输出的偏好提供了一项更简单的任务。通过这样做，我们可以创建一个理解人类偏好的模型，然后将其用作训练策略的奖励模型。从机器学习的角度来看，RLHF特别适用于难以展示但易于由人类识别的代理所需行为的情况。RLHF的另一个优点是其探索样本空间的能力。通过采用采样技术，使用强化学习训练的模型可以超越标注数据集，探索额外的样本。这种探索能力使RLHF能够发现从标注数据本身无法立即显而易见的潜在有益策略。

2.1.5 提示LLMs

我们已经展示了，通过提供适当的提示，LLMs 可以执行各种任务。这些提示没有限制，可以包括我们希望询问或与 LLMs 交流的任何信息，例如自然语言指令和对话的上下文。由于这种方法不需要额外的训练或调整，一旦开发出来，适应 LLMs 就变得非常高效。这多少影响了 NLP 的范式：我们不再需要为单个任务开发特定的系统，而是可以使用一个经过良好训练的 LLM 通过提示执行不同的任务。由此产生了一个吸引人的方面：用户可以通过为 LLMs 设计自己的提示来轻松拥有“定制”的系统。鉴于提示在 LLMs 中扮演的重要角色，提示工程已成为 NLP 中一个非常活跃的研究领域。

该术语 *prompt* 在文献中有很多不同的用法。在本章中，此术语指的是 LLM 的整个输入，因此我们交替使用术语 *prompt* 和 *model input*。在进一步讨论提示之前，让我们先看看一些比前几节中展示的提示更复杂的例子。请注意，本节的目的不是编写高质量的提示，而是突出一些在提示 LLM 时的一些有趣问题。

一种流行的使用 LLMs 的方法是为它们分配在生成响应中扮演的“角色”。例如，当回答问题时，LLMs 可以扮演心理医生的角色。

请解释什么是延迟满足？

注意，你是一位在心理学领域有深厚背景的研究员。你自己的研究领域是儿童发展心理学，重点关注控制和智力方面的内容。你对延迟满足的概念非常熟悉，并将其应用于自己的研究。你发表了关于延迟满足的论文并进行了相关实验。

延迟满足是指为了期待未来获得更有价值的回报而抵制即时奖励的过程。它在发展心理学中是一个重要的概念，尤其是在与儿童学习自我控制和培养认知能力的关系方面。研究表明，延迟满足的能力与学业成绩、社交能力和情绪福祉的积极结果相关。该领域的研究和实验强调了在早年培养这种技能的重要性，以支持长期成功和韧性。

另一个例子是在检测和纠正文本中的错误，如语法或语义错误时使用LLM。对于一个在代码和自然语言数据上训练的LLM，我们可以用它进行代码调试⁶。

修复此C语言程序中的错误。#include <stdio.h> int main() { printg ("Hello, World!") return 0; } 1. 函数名 ‘printg’ 不正确。它应该是 ‘printf’。2. 在 ‘printf’ 函数调用末尾缺少分号。以下是程序的正确版本：#include <stdio.h> int main() { printf("Hello, World!"); return 0; }

此外，LLMs 可以微调以充当对话伙伴⁷。

⁶In this example, the code is not tokenized for easier reading.

⁷To fine-tune an LLM for multi-turn dialogue, one needs to consider conversation history in the context for predicting the response in the current round of conversation. This makes the actual prompt used in response generation relatively longer than that used in single-turn dialogue.

嗨！我是一个助手。我能帮您什么吗？谁赢得了2022年FIFA世界杯？阿根廷赢得了2022年FIFA世界杯。在哪里举行的？2022年FIFA世界杯在卡塔尔举行。阿根廷赢得了多少次世界杯？

阿根廷赢得了三次FIFA世界杯。

...

这些示例和之前的示例都表明，通过涉及明确指示和问题的提示，可以生成适当的响应。然而，当解决问题需要未明确指定的知识时，即使指示足够清晰和精确，LLM也可能出错。对于LLM来说，一组具有挑战性的任务涉及算术推理和常识推理。例如，我们可以要求LLM解决以自然语言呈现的小学数学问题。

杰克有7个苹果。他晚饭时吃了2个，但后来他妈妈又给了他5个苹果。第二天，杰克给了他的朋友约翰3个苹果。最后杰克还剩下多少个苹果？

答案是10。_____

The correct answer is 7. 所以模型的输出是不一致的 incorrect.

一种解决此类问题的方法是将提示中融入学习，称为情境学习或（ICL）。ICL的思路是在提示中展示解决问题的方法，并将预测条件建立在这些演示之上。以下是一个示例，其中在提示中（绿色=演示）展示了类似的问题及其相应的答案。

汤姆有12个弹珠。他在和朋友的比赛中赢了7个弹珠，但第二天又失去了5个弹珠。他的哥哥送给他另外3个弹珠作为礼物。汤姆现在有多少个弹珠？

答案是 17。

杰克有7个苹果。他晚饭时吃了2个，但后来他妈妈又给了他5个苹果。第二天，杰克给了他的朋友约翰3个苹果。最后杰克还剩下多少个苹果？

答案是 12。_____

但是这次LLM仍然犯了错误。这其中的一个原因可能是解决数学问题不仅涉及问题-答案映射，而且在更大程度上，还涉及到多步骤的底层逻辑推理。提高LLM推理能力的一种方法是思维链提示（COT提示）[Wei等人，2022c]。在COT提示中，我们将复杂的推理问题分解成多个问题解决中间步骤。这些步骤是

在提示中进行了展示，以便LLMs可以接受提示学习模仿解决问题的关键思维过程。COT提示被发现对于在具有挑战性的任务上取得良好结果非常有用，例如GSM8K数学推理基准。

考虑上述演示示例 *Tom has 12 marbles. He wins 7 more marbles in a game with his friend but then loses 5 marbles the next day. His brother gives him another 3 marbles as a gift. How many marbles does Tom have now ?*。相应的逻辑推理步骤可以结构化：

1. 汤姆开始时有12个弹珠。
2. 在一场游戏中，他赢了7个弹珠，所以我们把这些加到他的初始数量上：12个弹珠 + 7个弹珠 = 19个弹珠。
3. 第二天，汤姆失去了5个弹珠，所以我们从他的当前总数中减去这些：19个弹珠 - 5个弹珠 = 14个弹珠。
4. 他的兄弟给了他3个弹珠，所以我们把这些加到他的当前总数上：14个弹珠 + 3个弹珠 = 17个弹珠。
5. 在所有这些事件之后，汤姆有17个弹珠。

我们可以将这些推理步骤添加到提示中，并得到正确答案。

汤姆有12个弹珠。他在和朋友的比赛中赢了7个弹珠，但第二天又失去了5个弹珠。他的哥哥送给他另外3个弹珠作为礼物。汤姆现在有多少个弹珠？

汤姆最初有12个弹珠。在赢得一场游戏后，他获得了7个，总数增加到19个弹珠。第二天他失去了5个弹珠，总数降至14个弹珠。他的兄弟送给他3个弹珠，总数增加到17个弹珠。因此，汤姆现在有17个弹珠。所以答案是17。

杰克有7个苹果。他晚饭时吃了2个，但后来他妈妈又给了他5个苹果。第二天，杰克给了他的朋友约翰3个苹果。最后杰克还剩下多少个苹果？

杰克开始时有7个苹果。他晚饭时吃了2个苹果，所以我们从7中减去2，剩下他还有5个苹果。他的妈妈又给了他5个苹果，这使他现在的总数增加了5。现在他有 $5 + 5 = 10$ 个苹果。第二天，杰克给了他的朋友约翰3个苹果。从10中减去3，杰克剩下7个苹果。因此，杰克还剩下7个苹果。所以答案是7。

这种方法也称为一次性COT提示。同样，我们可以添加更多演示来实现少量示例的COT提示方法。

此外，一种特殊的COT，零样本COT，引入了激发LLMs思考的指令，而不在提示中展示任何中间推理步骤[Kojima等人，2022]。一个有趣的例子是在提示的末尾附加一个简单的句子“让我们一步步思考。”，LLMs可以生成推理步骤以达到最终答案。

杰克有7个苹果。他晚饭时吃了2个，但后来他妈妈又给了他5个苹果。第二天，杰克给了他的朋友约翰3个苹果。最后杰克还剩下多少个苹果？让我们一步步来想。

1. 初始数量：Jack开始时有7个苹果。2. 晚餐后：他吃了2个苹果，所以剩下 $7 - 2 = 5$ 个苹果。3. 妈妈又给了更多：他的妈妈又给了他5个苹果，所以他现在有 $5 + 5 = 10$ 个苹果。4. 给John：第二天，Jack给了他的朋友John 3个苹果，所以剩下 $10 - 3 = 7$ 个苹果。最后，Jack剩下7个苹果。

零样本、单样本和少样本学习是LLM在上下文学习领域的常见概念，并不局限于COT提示。广义上讲，仅涉及简单指令而不包含任何演示的提示可以被视为一种零样本学习。这种零样本学习能力随着LLM的预训练和/或微调而出现。此外，当LLM没有获得相应的零样本学习能力时，更应考虑单样本和少样本学习方法。因此，在处理新任务时，这些方法对于上下文学习非常重要。例如，通过演示任务格式化样本执行各种NLP任务的方法。以下是一些示例，包括通过少样本学习进行情感句子分类和短语翻译。

给定以下文本片段，将其情感分类为积极、消极或中性。

示例 1：“我在公园里度过了一个精彩的一天！” 情感：积极

示例 2：“这家餐厅的服务太糟糕了。” 情感：负

面 示例 3：“我认为今天会下雨。” 情感：中性

这部电影是一次充满想象的精彩之旅。情感：积极

将以下中文短语翻译成英文。

早上好

LLM	# of Tokens	Data
GPT3-175B [Brown et al., 2020]	0.5T	Webpages, Books, Wikipedia
Falcon-180B [Almazrouei et al., 2023]	3.5T	Webpages, Books, Conversations, Code, Technical Articles
LLaMA2-65B [Touvron et al., 2023a]	1.0T ~ 1.4T	Webpages, Code, Wikipedia, Books, Papers, Q&As
PaLM-450B [Chowdhery et al., 2022]	0.78T	Webpages, Books, Conversations, Code, Wikipedia, News
Gemma-7B [Gemma Team, 2024]	6T	Webpages, Mathematics, Code

表2.3: 某些LLM中使用的训练数据量, 以标记数表示。

以上, 我们已提供示例来说明提示LLMs的基本情境学习能力。然而, 本节不包括更高级的提示技术, 以保持内容简洁紧凑。更多关于提示的讨论可以在第3章找到。

2.2 规模化训练

作为开发LLMs的第一步, 我们需要在大量数据上对这些模型进行训练。训练任务本身是标准的: 目标是最大化似然, 这可以通过梯度下降实现。然而, 随着模型大小和数据量的增加, 问题变得非常具有挑战性, 例如, 大型模型通常会使训练不稳定。在本节中, 我们讨论了LLMs大规模训练的几个问题, 包括数据准备、模型修改和分布式训练。我们还讨论了LLMs的缩放定律, 这有助于我们了解它们的训练效率和有效性。

2.2.1 数据准备

数据在NLP中的重要性不容忽视。随着更大型的神经网络被开发, 对数据的需求持续增加。例如, 开发LLMs可能需要数十万亿个标记进行预训练(参见表2.3), 比用于训练传统NLP模型的数量大得多。一般来说, 我们可能希望收集尽可能多的训练数据。然而, 更大的训练数据集并不意味着更好的训练结果, LLMs的发展在创建或收集这些数据集时也提出了新的问题。

数据质量是一个首要问题。高质量的长期被视为训练数据驱动NLP系统的关键。直接使用来自各种来源的原始文本通常是不受欢迎的。例如, 用于训练最近LLMs的大量数据来自网络爬取, 可能包含错误和不适当的内容, 如有毒信息和虚假事实。此外, 由于AI的广泛应用, 互联网上充斥着机器生成的内容, 这给处理和使用网络爬取数据带来了进一步的挑战。研究人员发现, 在未过滤的数据上训练LLMs是有害的 [Raffel等人, 2020]。提高数据质量通常涉及在数据处理工作流程中纳入过滤和清理步骤。例如, Penedo等人 [2023] 表明, 通过采用多种数据处理技术, 90%

他们的网络抓取数据中的一部分可以用于LLM训练。除了大规模网络抓取数据外，LLM训练数据通常包括书籍、论文、社交媒体上的用户生成数据等。大多数最新的LLM都是在这样的组合数据集上训练的，这些数据集被发现对于所得到的模型强大性能至关重要。

第二个问题是数据的多样性。我们希望训练数据涵盖尽可能多的数据类型，以便训练出的模型可以轻松适应不同的下游任务。人们普遍认为，训练数据的质量和多样性在LLMs中起着非常重要的作用。一个有趣的例子是将编程代码纳入训练数据已被发现对LLMs有益。这些益处不仅体现在增强LLMs的编程能力上，还体现在提高对复杂问题的推理能力上，尤其是那些需要COT提示的问题。概念“多样性”可以扩展到包括语言多样性。例如，许多LLMs是在多语言数据上训练的，因此我们可以使用单个模型处理多种语言。虽然这种方法在多语言和跨语言任务上显示出强大的能力，但其对特定语言的性能在很大程度上取决于这些语言的数据量和质量。在某些情况下，对于低资源语言，这种方法甚至可能提供较差的结果。

第三个问题是训练数据中的偏差。这不仅仅是一个特定于LLMs的问题，许多NLP系统中都存在。一个常见的例子是性别偏差，LLMs对某一性别比对另一性别有偏好。这可以部分归因于训练数据中的类别不平衡，例如，术语*nurses*更常与女性相关。为了消除数据偏差，通常的做法是平衡不同语言现象的类别，例如性别、种族和方言。数据偏差也与上述的多样性问题相关。例如，由于许多LLMs是在以英语为中心的数据上训练和校准的，它们偏向于英语 speaking 人群中的主流文化价值观和观点。在训练数据中增加语言多样性可以在一定程度上减轻偏差。

大规模数据收集的另一个问题是隐私问题。如果LLMs在广泛来源的数据上进行训练，这可能导致敏感信息泄露的风险，例如知识产权和个人数据。考虑到LLMs能够从其训练数据中表征模式，这可能会无意中涉及记住和再现特定细节，因此这一点尤其令人担忧。保护隐私的一个简单方法是删除或匿名化敏感信息。例如，可以通过匿名化技术从训练数据中删除可识别个人信息的部分，以防止LLMs从此类数据中学习。然而，在实践中，删除或编辑所有敏感数据是困难的。因此，许多LLMs，尤其是那些用于公共服务的LLMs，通常与能够检测敏感数据潜在泄露的系统一起工作，或者经过微调以拒绝可能导致信息泄露的某些请求。

2.2.2 模型修改

训练LLMs很困难。一个常见的问题是，随着LLMs的增大，训练过程变得更加不稳定。例如，为了通过梯度下降实现稳定的训练，需要选择较小的学习率，但这反过来又会导致训练时间大大延长。有时，即使训练配置被精心设计，在优化过程中训练也可能在某些点上发散。LLMs的训练通常受许多因素的影响，例如参数初始化、批处理和正则化。在这里，我们关注常见的

标准Transformer架构的修改和改进，这些在开发可训练的LLMs中被认为是重要的。

2.2.2.1 带残差连接的层归一化

层归一化用于稳定深度神经网络的训练。它是一个减去均值并除以标准差的过程。通过以这种方式归一化层输出，我们可以有效地减少协变量偏移问题并提高训练稳定性。在Transformers中，层归一化通常与残差连接一起使用。如第2.1.1节所述，子层可以基于后归一化架构，其中层归一化在残差块之后立即执行，或者基于前归一化架构，其中层归一化在残差块内部执行。虽然这两种架构在基于Transformer的系统[王等, 2019]中都被广泛使用，但前归一化架构在训练深度Transformer方面已被证明特别有用。因此，大多数LLM都是基于前归一化架构，表示为输出 = $\text{LNorm}(F(\text{input})) + \text{input}$ 。

广泛使用的层归一化函数形式如下：

$$\text{LNorm}(\mathbf{h}) = \alpha \cdot \frac{\mathbf{h} - \mu}{\sigma + \epsilon} + \beta \quad (2.23)$$

\mathbf{h} 是一个 d -维实值向量， μ 是 \mathbf{h} 所有元素的均值， σ 是对应的标准差。 ϵ 引入是为了数值稳定性。 $\alpha \in \mathbb{R}^d$ 和 $\beta \in \mathbb{R}^d$ 是增益和偏置项。

一种层归一化的变体，称为均方根（RMS）层归一化，仅重新缩放输入向量但不重新居中[Zhang and Sennrich, 2019]。RMS层归一化函数由以下给出 $\{\mathbf{v}^*\}$

$$\text{LNorm}(\mathbf{h}) = \alpha \cdot \frac{\mathbf{h}}{\sigma_{\text{rms}} + \epsilon} + \beta \quad (2.24)$$

σ_{rms} 是 \mathbf{h} 的均方根，即 $\sigma_{\text{rms}} = (\frac{1}{d} \sum_{k=1}^d h_k^2)^{\frac{1}{2}}$ 。此层归一化函数用于 LLaMA 系列等 LLM 中。

2.2.2.2 FFN 中的激活函数

在Transformer中，FFN子层被设计用于向表示学习中引入非线性，并发现对于防止自注意力学习的表示退化很有用⁸ [Dong等人, 2021]。这些子层中使用的FFN的标准形式可以表示为

$$\text{FFN}(\mathbf{h}) = \sigma(\mathbf{h}\mathbf{W}_h + \mathbf{b}_h)\mathbf{W}_f + \mathbf{b}_f \quad (2.25)$$

在 $\mathbf{W}_h \in \mathbb{R}^{d \times d_h}$ 、 $\mathbf{b}_h \in \mathbb{R}^{d_h}$ 、 $\mathbf{W}_f \in \mathbb{R}^{d_h \times d}$ 和 $\mathbf{b}_f \in \mathbb{R}^d$ 是参数， d_h 是隐藏大小的情况下， $\sigma(\cdot)$ 是隐藏层的激活函数。对于 $\sigma(\cdot)$ 的一个常见选择是

⁸Here degeneration refers to the phenomenon in which the rank of a matrix is reduced after some processing.

修正线性单元（ReLU），表示为

$$\sigma_{\text{relu}}(\mathbf{h}) = \max(0, \mathbf{h}) \quad (2.26)$$

在实用实现中，增加 d_h 是有帮助的，因此它通常在 LLMs 中设置为较大的数字。但是，非常大的隐藏大小对训练和部署都提出了挑战。在这种情况下，激活函数的设计在宽 FFNs 中起着相对更重要的作用。在 LLMs 中，ReLU 有几种替代方案。其中之一是高斯误差线性单元（GeLU），它可以看作是 ReLU 的平滑版本。GeLU 函数不是通过输入的符号来控制输出，而是通过百分位数 $\Pr(h \leq \mathbf{h})$ 加权其输入。在这里 h 是一个 d -维向量，其条目是从标准正态分布高斯(0, 1)⁹ 中抽取的。具体来说，GeLU 函数被定义为

$$\begin{aligned} \sigma_{\text{gelu}}(\mathbf{h}) &= \mathbf{h} \Pr(h \leq \mathbf{h}) \\ &= \mathbf{h} \Phi(\mathbf{h}) \end{aligned} \quad (2.27)$$

$\Phi(\mathbf{h})$ 是高斯(0, 1)的累积分布函数，可以通过方便的方式实现 [Hendrycks and Gimpel, 2016]。GeLU 函数已被采用在多个 LLM 中，如 BERT、GPT-3 和 BLOOM。

另一类在 LLMs 中流行的激活函数族是基于门控线性单元（GLU）的函数。GLU 的基本形式由以下给出 $\{v^*\}$

$$\sigma_{\text{glu}}(\mathbf{h}) = \sigma(\mathbf{h}\mathbf{W}_1 + \mathbf{b}_1) \odot (\mathbf{W}_2 + \mathbf{b}_2) \quad (2.28)$$

在 $\mathbf{W}_1 \in \mathbb{R}^{d \times d}$ 、 $\mathbf{b}_1 \in \mathbb{R}^d$ 、 $\mathbf{W}_2 \in \mathbb{R}^{d \times d}$ 和 $\mathbf{b}_2 \in \mathbb{R}^d$ 是模型参数。 $\sigma(\cdot)$ 的不同选择会导致 GLU 函数的不同版本。例如，如果将 $\sigma(\cdot)$ 定义为 GeLU 函数，我们将得到 GeGLU 函数

$$\sigma_{\text{geglu}}(\mathbf{h}) = \sigma_{\text{gelu}}(\mathbf{h}\mathbf{W}_1 + \mathbf{b}_1) \odot (\mathbf{W}_2 + \mathbf{b}_2) \quad (2.29)$$

此激活函数已在 Gemma 等 LLMs 中成功应用。

作为另一个例子，考虑 $\sigma(\cdot)$ 为 Swish 函数 $\sigma_{\text{swish}}(\mathbf{h}) = \mathbf{h} \odot \text{Sigmoid}(c\mathbf{h})$ [Ramachandran 等人, 2017]。然后，SwiGLU 函数由以下给出

$$\sigma_{\text{swiglu}}(\mathbf{h}) = \sigma_{\text{swish}}(\mathbf{h}\mathbf{W}_1 + \mathbf{b}_1) \odot (\mathbf{W}_2 + \mathbf{b}_2) \quad (2.30)$$

PaLM 和 LLaMA 系列均基于 SwiGLU 函数。有关 GLUs 的更多讨论，读者可参考 Shazeer [2020] 的作品。

⁹ $\Pr(h \leq \mathbf{h})$ is an informal notation. It refers to a vector, with each entry representing the percentile for the corresponding entry of \mathbf{h} .

2.2.2.3 移除偏差项

另一种流行的模型设计是在LLMs中使用的仿射变换中移除偏差项。这种处理可以应用于层归一化、QKV注意力的输入变换以及FFNs。例如，我们可以修改公式（2.25）以获得没有偏差项的FFN

$$\text{FFN}(\mathbf{h}) = \sigma(\mathbf{h}\mathbf{W}_h)\mathbf{W}_f \quad (2.31)$$

Chowdhery等人[2022]报告称，去除偏差项有助于提高LLMs的训练稳定性。这种方法已在几个最近的LLMs中应用，例如LLaMA和Gemma。

2.2.2.4 其他问题

许多LLM还涉及对其位置嵌入模型的修改。例如，可以用旋转位置嵌入替换正弦位置编码，以便学习到的LLM能更好地处理长序列。这些模型将在第2.3节中讨论。

注意，虽然模型修改在训练LLM时很常见，但可以通过许多不同的方式提高训练的稳定性。例如，随着训练的进行增加批大小已被发现对某些LLM很有用。一般来说，实现稳定高效的大规模LLM训练需要精心设计的设置，包括学习计划、优化器选择、训练并行性、混合精度训练等等。其中一些问题高度工程化，因此，我们通常需要多次训练运行才能获得令人满意的LLM。

2.2.3 分布式训练

训练LLMs需要大量的计算资源。提高训练效率的常见方法之一是使用大规模分布式系统。幸运的是，随着人工智能中神经网络的发展，面向深度学习的软件和硬件也得到了开发，这使得实现LLMs和执行计算变得更加容易。例如，现在人们可以轻松地使用深度学习软件框架和多GPU机器微调LLM。然而，扩大LLMs的训练规模仍然具有挑战性，需要投入大量努力来开发稳定高效的分布式训练的硬件和软件系统。

分布式训练的一个重要考虑因素是并行性。存在几种并行形式：数据并行、模型并行、张量并行和流水线并行。尽管有不同方式在设备间分配计算，但这些并行方法基于一个类似的想法：训练问题可以被划分为可以同时执行的小任务。训练LLM的并行性问题已被广泛研究[Narayanan等人，2021；Fedus等人，2022]。在此，我们概述基本概念。

- 数据并行。这种方法是训练神经网络最广泛使用的并行方法之一。为了说明，考虑最简单的情况，其中在梯度下降中使用标准delta规则

$$\theta_{t+1} = \theta_t - lr \cdot \frac{\partial L_{\theta_t}(\mathcal{D}_{\text{mini}})}{\partial \theta_t} \quad (2.32)$$

在新的参数 θ_{t+1} 通过将最新的参数 θ_t 沿着负损失梯度的方向更新一个小步 lr 获得的地方。 $\frac{\partial L_{\theta_t}(\mathcal{D}_{\text{mini}})}{\partial \theta_t}$ 是损失相对于参数 θ_t 的梯度，并在训练样本的minibatch $\mathcal{D}_{\text{mini}}$ 上计算。在数据并行中，我们将 $\mathcal{D}_{\text{mini}}$ 划分为 N 个更小的批次，记为 $\{\mathcal{D}^1, \dots, \mathcal{D}^N\}$ 。然后，我们将这些批次分配给 N 个工作者，每个工作者对应一个批次。一旦数据被分配，这些工作者可以同时工作。通过汇总工作者计算出的梯度，获得整个minibatch的梯度，如下所示

$$\frac{\partial L_{\theta_t}(\mathcal{D}_{\text{mini}})}{\partial \theta_t} = \underbrace{\frac{\partial L_{\theta_t}(\mathcal{D}^1)}{\partial \theta_t}}_{\text{worker 1}} + \underbrace{\frac{\partial L_{\theta_t}(\mathcal{D}^2)}{\partial \theta_t}}_{\text{worker 2}} + \dots + \underbrace{\frac{\partial L_{\theta_t}(\mathcal{D}^N)}{\partial \theta_t}}_{\text{worker N}} \quad (2.33)$$

在理想情况下，当工人协调良好且通信开销小，数据并行可以实现训练速度提高近 N 倍。

- 模型并行。尽管数据并行简单有效，但它要求每个工作节点运行整个LLM并执行完整的正向和反向过程。随着LLM的增大，有时在单个设备上加载和执行LLM变得不可行。在这种情况下，我们可以将LLM解耦成更小的组件，并在不同的设备上运行这些组件。一种简单的方法是将层堆栈中连续的层分组，并将每个组分配给一个工作节点。工作节点按照层堆栈中层的顺序操作，即在正向传递中，我们从低级层到高级层处理输入，在反向传递中，我们从高级层到低级层传播误差梯度。例如，考虑一个具有 L 堆叠块的Transformer解码器。为了分配计算负载，每个块被分配给一个工作节点。请参见以下插图，以了解此模型正向和反向传递的单次运行。

Worker L	B_L (\uparrow) B_L (\downarrow)
...	...
Worker 2	B_2 (\uparrow) B_2 (\downarrow)
Worker 1	B_1 (\uparrow) B_1 (\downarrow)

这里 B_l 表示计算块 l ，符号 \uparrow 和 \downarrow 分别表示正向和反向传递。请注意，此并行方法强制工作器按顺序运行，因此一个工作器必须等待前一个工作器完成其工作。这导致设备大部分时间处于空闲状态。在实际系统中，通常将模型并行与其他并行机制一起使用，以最大限度地利用设备。

- 张量并行。并行也可以在单个计算步骤中执行。一个常见的例子是将一个大参数矩阵分割成块，分别将输入张量与这些块相乘，然后将这些乘积的结果连接起来形成输出。例如，考虑FFN子层中代表 $\mathbf{h} \in \mathbb{R}^d$ 与参数矩阵 $\mathbf{W}_h \in \mathbb{R}^{d \times d_h}$ 的乘法（见公式(2.25)）。我们可以将矩阵 $\mathbf{W}_h \in \mathbb{R}^{d \times d_h}$ 垂直切割成一系列 M 子矩阵

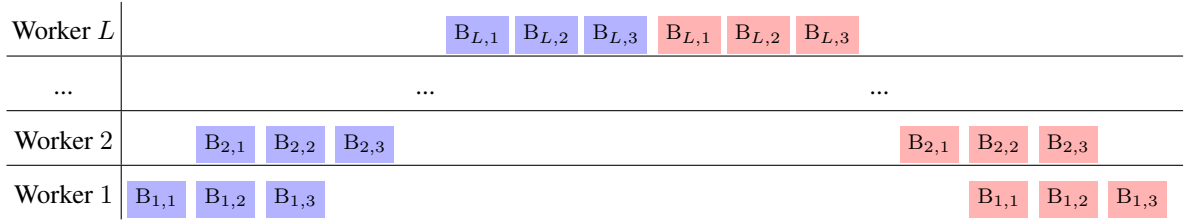
$$\mathbf{W}_h = [\mathbf{W}_h^1 \quad \mathbf{W}_h^2 \quad \dots \quad \mathbf{W}_h^M] \quad (2.34)$$

每个子矩阵 \mathbf{W}_h^k 的形状为 $d \times \frac{d_h}{M}$ 。 \mathbf{h} 与 \mathbf{W}_h 的乘积可以表示为

$$\begin{aligned}\mathbf{h}\mathbf{W}_h &= \mathbf{h} \begin{bmatrix} \mathbf{W}_h^1 & \mathbf{W}_h^2 & \dots & \mathbf{W}_h^M \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{h}\mathbf{W}_h^1 & \mathbf{h}\mathbf{W}_h^2 & \dots & \mathbf{h}\mathbf{W}_h^M \end{bmatrix}\end{aligned}\quad (2.35)$$

我们可以分别对 $\{\mathbf{h}\mathbf{W}_h^1, \mathbf{h}\mathbf{W}_h^2, \dots, \mathbf{h}\mathbf{W}_h^M\}$ 进行矩阵乘法。结果，我们将大型矩阵乘法分布在多个设备上，每个设备可能具有相对较小的内存。从现代GPU的设计角度来看，GPU上的张量并行提供了两层、基于瓦片的并行计算方法。首先，在较高层次上，我们将矩阵乘法分解为可以直接适应GPU内存的子矩阵乘法。然后，在较低层次上，我们使用针对GPU特别优化的基于瓦片的并行算法在GPU上执行这些子矩阵乘法。

- 管道并行。在上面，在模型并行中，我们描述了一种将模型组件组分散到多个设备上的简单方法。但这种方法效率低下，因为在处理过程中每次只有一个设备被激活。管道并行通过在不同设备上的计算之间引入重叠来解决这个问题[Harlap等人, 2018; Huang等人, 2019]。为此，一批样本被分成多个微批次，然后这些微批次由每个工作者按常规处理。一旦一个微批次被工作者处理并通过给下一个，下一个微批次立即占用相同的工作者。换句话说，我们创建了一个管道，如果给管道多个作业，不同的计算步骤可以重叠。以下是一个处理3个微批次的管道并行的示意图。



这里 $B_{l,k}$ 表示第 k 个微批次的处理由第 l 个工作者完成。理想情况下，我们希望最大化微批次数，从而最小化工作者的空闲时间。然而，在实践中，使用小的微批次通常会降低GPU利用率并增加任务切换成本。这反过来可能会降低整个系统的吞吐量。

。

并行处理的最终目标是实现效率的线性增长，即单位时间内可以处理的样本数量与设备数量呈线性增长。然而，分布式训练很复杂，除了我们选择的并行方法外，还受到许多因素的影响。一个问题，通常与分布式系统相关，是通信成本。我们可以将分布式系统视为一组网络节点。这些节点中的每一个都可以执行本地计算或将数据传递给其他节点。如果

存在大量此类节点，跨它们分发和收集数据将非常昂贵。有时，并行带来的时间节省被大型网络的通信开销所抵消。大规模分布式系统另一个问题是节点同步引入了额外的成本。正如常发生的那样，一些节点可能需要更长的时间来工作，导致其他节点等待最慢的节点。虽然我们可以使用异步训练来处理计算资源的异质性，但这可能导致梯度过时和非保证收敛。此外，随着网络中节点数量的增加，在训练过程中出现崩溃节点的可能性也更大。在这种情况下，我们需要确保整个系统具有容错性。在许多实际设置中，为了提高可扩展性，需要考虑包括架构设计、数据传输和计算重叠、负载均衡、内存带宽等问题。

训练LLMs的计算成本如此之高，尽管已经使用分布式训练，研究人员和工程师仍然经常采用各种模型压缩和加速方法来提高训练效率 [Weng, 2021]。一个例子是混合精度训练，其中在每个节点上使用低精度数据（如FP16和FP8数据）进行梯度计算，并使用单精度或双精度数据（如FP32/FP64数据）更新模型 [Micikevicius等人, 2018]。这种方法中的一个关键操作是梯度累积，需要在节点间累积和同步梯度。然而，由于浮点加法的非结合性，这可能导致不同节点上累积梯度的微小数值差异，这可能会影响模型收敛和最终性能。如果参与分布式训练的节点数量很多，这个问题就更加明显，尤其是在低精度数值计算可能遇到溢出和下溢问题，以及不同硬件设备之间不一致的情况下。因此，分布式系统的设计需要考虑这些数值计算问题，以确保满意的结果和收敛。

2.2.4 缩放定律

LLMs的成功表明，使用更多资源训练更大的语言模型可以提高模型性能。研究人员将其解释为LLMs的缩放定律。更具体地说，缩放定律描述了LLMs的性能与其训练属性之间的关系，例如模型大小、用于训练的计算量以及训练数据量。例如，Hestness等人[2017]表明，深度神经网络的性能是训练数据大小的幂律函数。最初，当训练数据量不大时，模型性能缓慢提高。随后，当使用更多训练数据时，模型进入快速性能提升阶段，性能曲线类似于幂律曲线。最终，性能提升再次变慢，更多数据不会导致显著收益。图2.3显示了此类曲线的示例。

在NLP中，一种传统观点认为，随着训练规模的扩大，性能提升将在某个点上消失。然而，最近的研究表明，如果我们从更大的规模来考虑这个问题，扩大训练规模仍然是获得更强大的LLMs的有效方法。例如，无论是闭源还是开源的LLMs，都可以从更多数据中受益，即使已经使用了数万亿个标记进行训练。

随着模型训练规模的增加，LLMs展现出新的能力，被称为

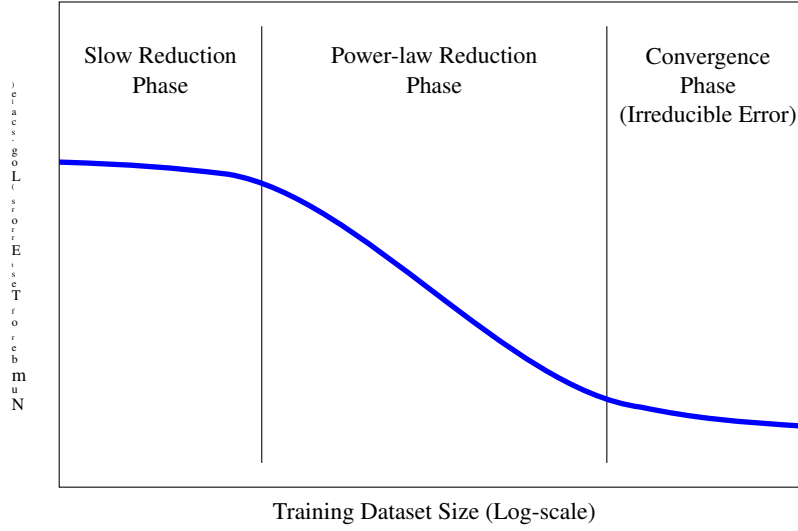


图2.3: 针对感兴趣变量的测试误差缩放定律 (例如, 训练数据集大小) [Hestness等人, 2017]。缩放定律的曲线可以分为三个阶段。最初, 当使用更多训练数据时, 测试错误数量缓慢减少, 但这种效果仅维持短暂时间。在第二阶段, 测试错误数量急剧减少, 曲线变为幂律曲线。之后, 在第三阶段, 误差减少速度再次放缓。请注意, 存在无法消除的不可减少误差, 无论训练数据量如何。

LLMs的涌现能力。例如, Wei等人[2022b]研究了不同模型大小和计算资源量下LLMs的缩放特性。他们的工作表明, 当我们把模型大小扩展到一定水平时, 某些能力会涌现出来。涌现能力的出现证明了缩放训练在提升LLMs性能中的作用, 并在一定程度上激励研究人员不断尝试训练更大的模型。随着更大更强的LM不断出现, 我们对缩放定律的理解也在不断成熟。这有助于研究人员预测LLMs在训练过程中的性能, 并估算实现给定性能水平所需的最小计算资源。

为了理解模型性能如何随着训练过程中考虑的各种因素而变化, 通常将模型性能表示为这些因素的函数。例如, 在最简单的情况下, 我们可以将LLM的损失或误差表示为单个感兴趣变量的函数。然而, 没有普遍适用的缩放定律可以描述这种关系。相反, 提出了不同的函数来拟合LLM的学习曲线。

设 x 为感兴趣的变量 (例如模型参数的数量) 和 $\mathcal{L}(x)$ 为给定 x (例如测试数据上的交叉熵损失) 的模型损失)。 $\mathcal{L}(x)$ 的最简单形式是幂律

$$\mathcal{L}(x) = ax^b \quad (2.36)$$

在 a 和 b 是通过经验估计的参数。尽管这个函数很简单, 但它成功地解释了语言模型和机器翻译系统在模型大小 (用 N 表示) 和训练数据集大小 (用 D 表示) 方面的缩放能力 [Gordon 等人, 2021; Hestness 等人, 2017]。例如, Kaplan 等人 [2020] 发现, 在初始瞬态期之后, 他们的语言模型的性能随着 N 或 D 的幂律提高, 并使用 $\mathcal{L}(N) = \left(\frac{N}{8.8 \times 10^{13}}\right)^{-0.076}$ 和 $\mathcal{L}(D) = \left(\frac{D}{5.4 \times 10^{13}}\right)^{-0.095}$ (表示这些关系, 见图 2.4)。

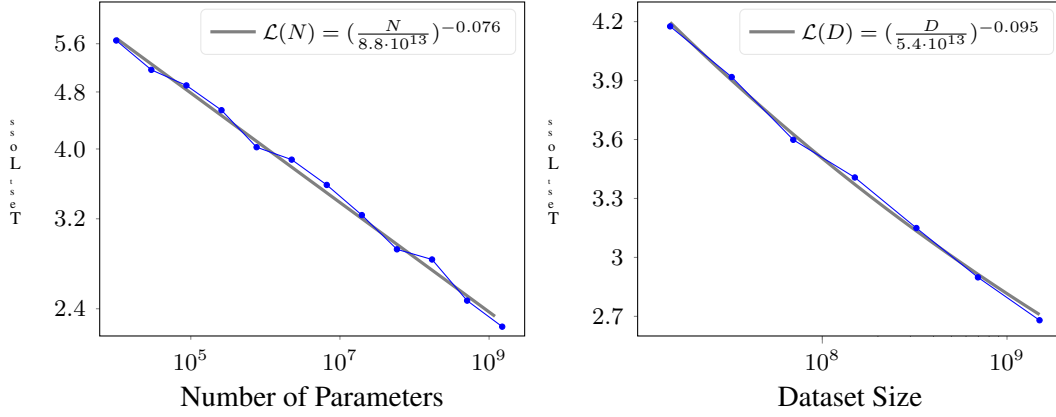


图2.4: 测试损失与模型大小(N)和训练数据集大小(D)的关系 (数据点用于说明目的)。我们将测试损失作为 N 的函数绘制, 其中 N 定义为 $\mathcal{L}(N) = \left(\frac{N}{8.8 \times 10^{13}}\right)^{-0.076}$, 以及作为 D 的函数绘制, 其中 D 定义为 $\mathcal{L}(D) = \left(\frac{D}{5.4 \times 10^{13}}\right)^{-0.095}$ [Kaplan等人, 2020]。

对这种缩放定律的改进是向幂律中添加一个不可约误差项。此时, $\mathcal{L}(x)$ 的形式由以下给出

$$\mathcal{L}(x) = ax^b + \epsilon_{\infty} \quad (2.37)$$

在 ϵ_{∞} 是不可约误差, 它解释了由于未知变量引起的误差, 即使 $x \rightarrow \infty$ 存在。方程式 (2.37) 是设计 LLM 放缩定律最广泛使用的形式之一。例如, Rosenfeld 等人 [2020] 开发了一种涉及模型放缩和数据集放缩的放缩定律, 如下所示

$$\mathcal{L}(N, D) = aN^b + cD^d + \epsilon_{\infty} \quad (2.38)$$

一个此类公式的例子是Chinchilla缩放定律。它表明每个标记的测试损失是 N 和 D 的逆比例函数之和, 还有一个额外的不可约误差项。Hoffmann等人[2022]将此缩放定律表示为

$$\mathcal{L}(N, D) = \underbrace{\frac{406.4}{N^{0.34}}}_{\text{model scaling}} + \underbrace{\frac{410.7}{D^{0.28}}}_{\text{dataset scaling}} + \underbrace{1.69}_{\text{irreducible error}} \quad (2.39)$$

所有上述缩放定律均基于单调函数。因此, 它们不能涵盖具有拐点的函数, 例如双下降曲线。作为回应, 研究人员已经探索了更复杂的函数来拟合学习曲线。此类函数的例子可以在Alabdulmohsin等人[2022]和Caballero等人[2023]的工作中找到。

缩放定律的重要性在于为LLM研究提供方向性指导: 如果我们仍然处于幂律曲线区域, 使用更多资源来训练更大的模型是一个非常具有前途的方向。虽然这个结果“迫使”大型研究团队和公司投资更多计算资源来训练更大的模型, 这非常昂贵, 但缩放定律不断将人工智能的边界推向更远。另一方面, 理解缩放定律有助于研究人员在训练LLM时做出决策。例如, 考虑到计算

re手头资料，LLMs 的性能可能预测已连接。

关于本节中的缩放定律，有一点需要注意。对于LLM，较低的测试损失并不总是意味着在所有下游任务上的性能更好。为了适应LLM，有多个步骤，如微调和提示，可能会影响最终结果。因此，不同下游任务的缩放定律在实践中可能不同。

2.3 长序列建模

我们已经看到，在大规模训练中，通过使用更多的数据和计算资源，我们可以开发出更大的语言模型。然而，扩展也可以在其他方向发生。例如，在许多应用中，LLMs被调整以处理显著长的序列。一个有趣的例子是我们在一个正常长度的广泛文本上预训练一个LLM，然后将其应用于处理非常长的标记序列，远远超出预训练中遇到的长度。在这里，我们使用 $\Pr(y\{v^*\}x)$ 来表示文本生成概率，其中 x 是上下文， y 是生成的文本。长序列建模问题大致分为三种类型。

- 基于长上下文（即， x 是一个长序列）的文本生成。例如，我们为非常长的文本生成一个简短的摘要。
- 长文本生成（即 y 是一个长序列）。例如，我们根据几个关键词生成一个长故事。
- 基于长上下文的长文本生成（即 x 和 y 都是长序列）。例如，我们将一篇长文档从中文翻译成英文。

最近，NLP 研究人员对在涉及极长输入文本的任务中应用和评估 LLMs 的兴趣日益增加。想象一个 LLM，它读取一个包含数万行代码的 C++ 源文件，并概述与源文件对应的程序功能。这种能够处理广泛文本上下文模型有时被称为长上下文 LLM。在本节中，我们将限制自己讨论长上下文 LLM，但这里讨论的方法也可适用于其他问题。

对于Transformer，处理长序列在计算上非常昂贵，因为自注意力机制的计算成本随着序列长度的增加而呈二次增长。这使得为非常长的输入训练和部署此类模型变得不可行。两条研究思路试图将Transformer适应于长上下文语言建模。

- 第一次探索了从长序列数据中学习自注意力模型的高效训练方法和模型架构。
- 其他则将预训练的LLMs适配以处理长序列，无需或仅需少量微调。

这里，我们将简要讨论前者，因为它可以在关于高效Transformer架构的一般讨论中找到[Tay等人，2020；Xiao和Zhu，2023]。我们将专注于后者，

突出近期LLMs中流行的方法。我们还将讨论这些长序列模型的优势和局限性。

2.3.1 从HPC视角的优化

我们开始讨论，从高性能计算的角度考虑对标准Transformer模型的改进。尽管这些改进并非专门为LLMs设计，但它们已被广泛应用于各种深度学习模型[Kim等人, 2023]。常用的方法之一是采用Transformer的低精度实现。例如，我们可以使用8位或16位定点数据类型进行算术运算，而不是32位或64位浮点数据类型。使用这些低精度数据类型可以提高效率和内存吞吐量，从而更容易处理更长的序列。另一种方法是使用硬件感知技术改进Transformer。例如，在现代GPU上，可以通过使用自注意力函数的IO感知实现来提高Transformer的效率[Dao等人, 2022; Kwon等人, 2023]。

另一种处理长序列的方法是通过序列并行化 [Li 等人, 2023b; Korthikanti 等人, 2023]。具体来说，考虑将查询 \mathbf{q}_i 在位置 i 关注到键 \mathbf{K} 和值 \mathbf{V} 的一般问题。我们可以按行划分 \mathbf{K} ，得到一组子矩阵 $\{\mathbf{K}^{[1]}, \dots, \mathbf{K}^{[n_u]}\}$ ，每个子矩阵对应序列的一个片段。同样，我们可以得到 \mathbf{V} 的子矩阵，记为 $\{\mathbf{V}^{[1]}, \dots, \mathbf{V}^{[n_u]}\}$ 。然后，我们将每一对 $\mathbf{K}^{[u]}$ 和 $\mathbf{V}^{[u]}$ 分配给一个计算节点（例如，GPU 集群的 GPU）。分配的节点可以并行运行，从而并行化注意力操作。

回忆一下，自注意力模型的输出可以写成

$$\text{Att}_{\text{qkv}}(\mathbf{q}_i, \mathbf{K}, \mathbf{V}) = \sum_{j=0}^{m-1} \alpha_{i,j} \mathbf{v}_j \quad (2.40)$$

在 $\alpha_{i,j}$ 是位置 i 和 j 之间的注意力权重。在 Transformers 中， $\alpha_{i,j}$ 通过对 \mathbf{q}_i 和 \mathbf{k}_j 之间的点积的缩放版本进行归一化得到。令 $\beta_{i,j}$ 表示 \mathbf{q}_i 和 \mathbf{k}_j 之间的注意力分数。我们有

$$\beta_{i,j} = \frac{\mathbf{q}_i \cdot \mathbf{k}_j}{\sqrt{d}} + \text{Mask}(i, j) \quad (2.41)$$

在 $\text{Mask}(i, j)$ 是 (i, j) 的掩码变量时，我们定义注意力权重 $\alpha_{i,j}$ 为

$$\begin{aligned} \alpha_{i,j} &= \text{Softmax}(\beta_{i,j}) \\ &= \frac{\exp(\beta_{i,j})}{\sum_{j'} \exp(\beta_{i,j'})} \end{aligned} \quad (2.42)$$

在每一个计算节点上，我们需要实现这些方程。给定分配给该节点的键和值，计算方程 (2.42) 右侧的分子（即， $\exp(\beta_{i,j})$ ）是直接的，因为所有所需信息都存储在节点上。然而，计算方程 (2.42) 右侧的分母涉及到对所有 j' 的 $\exp(\beta_{i,j'})$ 的求和，这

需要将数据传输到其他节点。为了说明，假设 \mathbf{v}_j 和 \mathbf{k}_j 放置在节点 u 上。我们可以将式(2.42)重写为

$$\alpha_{i,j} = \frac{\overbrace{\exp(\beta_{i,j})}^{\text{node } u}}{\underbrace{\sum_{\mathbf{k}_{j'} \in \mathbf{K}^{[1]}} \exp(\beta_{i,j'})}_{\text{node 1}} + \cdots + \underbrace{\sum_{\mathbf{k}_{j'} \in \mathbf{K}^{[u]}} \exp(\beta_{i,j'})}_{\text{node } u} + \cdots + \underbrace{\sum_{\mathbf{k}_{j'} \in \mathbf{K}^{[n_u]}} \exp(\beta_{i,j'})}_{\text{node } n_u}} \quad (2.43)$$

在标记法 $\mathbf{k}_{j'} \in \mathbf{K}^{[u]}$ 表示 $\mathbf{k}_{j'}$ 是一个 $\mathbf{K}^{[u]}$ 的行向量。在直接实现中，我们首先分别对相应的节点执行求和 $\{\sum_{\mathbf{k}_{j'} \in \mathbf{K}^{[u]}} \exp(\beta_{i,j'})\}$ 。然后，我们从不同的节点收集这些求和结果以将它们组合成最终结果。这对应于并行处理环境中的集体操作。此类操作有许多高效实现，例如 all-reduce 算法。因此，所有 $\exp(\beta_{i,j})$ 值的总和可以使用集体通信工具包中的优化例程计算。

给定注意力权重 $\{\alpha_{i,j}\}$ ，然后我们使用公式(2.40)计算注意力结果。问题可以重新表述为

$$\text{Att}_{\text{qkv}}(\mathbf{q}_i, \mathbf{K}, \mathbf{V}) = \underbrace{\sum_{\mathbf{v}_{j'} \in \mathbf{V}^{[1]}} \alpha_{i,j'} \mathbf{v}_{j'}}_{\text{node 1}} + \cdots + \underbrace{\sum_{\mathbf{v}_{j'} \in \mathbf{V}^{[u]}} \alpha_{i,j'} \mathbf{v}_{j'}}_{\text{node } u} + \cdots + \underbrace{\sum_{\mathbf{v}_{j'} \in \mathbf{V}^{[n_u]}} \alpha_{i,j'} \mathbf{v}_{j'}}_{\text{node } n_u} \quad (2.44)$$

与公式(2.43)类似，公式(2.44)可以作为并行处理中的求和程序实现。首先，同时在不同节点上执行值的加权求和。然后，我们通过集体操作从这些节点收集结果。

注意，尽管本节主要关注长序列建模，但序列并行的许多动机来自深度网络的分布式训练方法，如第2.2.3节所述。因此，这些方法的实现可以基于相同的并行处理库。

2.3.2 高效架构

Transformer应用于长序列的一个困难是，自注意力在序列长度方面具有二次时间复杂度。此外，在推理过程中维护一个键值缓存（或简称为KV缓存），其大小随着处理的标记数量增加而增加。尽管KV缓存随着序列长度的增加而线性增长，但对于非常长的输入序列，内存占用变得显著，甚至无法部署LLMs来完成此类任务。因此，长上下文LLMs的模型架构通常远离标准Transformer，转而开发更高效的变体和替代方案。

一种方法是使用稀疏注意力而不是标准自注意力。这一系列模型基于这样的想法：在关注给定标记时，只有少数标记被认为是重要的，因此大多数标记之间的注意力权重接近于零。

因此，我们可以剪枝大部分注意力权重，并以压缩形式表示注意力模型。为了说明，考虑自注意力模型

$$\text{Att}_{\text{qkv}}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \alpha(\mathbf{Q}, \mathbf{K})\mathbf{V} \quad (2.45)$$

在何处通过 $\alpha(\mathbf{Q}, \mathbf{K}) \in \mathbb{R}^{m \times m}$ 获得注意力权重矩阵

$$\begin{aligned} \alpha(\mathbf{Q}, \mathbf{K}) &= \text{Softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d}} + \mathbf{Mask}\right) \\ &= \begin{bmatrix} \alpha_{0,0} & 0 & 0 & \dots & 0 \\ \alpha_{1,0} & \alpha_{1,1} & 0 & \dots & 0 \\ \alpha_{2,0} & \alpha_{2,1} & \alpha_{2,2} & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \alpha_{m-1,0} & \alpha_{m-1,1} & \alpha_{m-1,2} & \dots & \alpha_{m-1,m-1} \end{bmatrix} \end{aligned} \quad (2.46)$$

每行向量 $[\alpha_{i,0} \dots \alpha_{i,i} 0 \dots 0]$ 对应于将 i -th 令牌分配到序列中每个令牌的分佈。由于语言模型仅根据它们的左上下文预测下一个令牌，因此我们通常将注意力模型在位置 i 的输出写为

$$\begin{aligned} \text{Att}_{\text{qkv}}(\mathbf{q}_i, \mathbf{K}_{\leq i}, \mathbf{V}_{\leq i}) &= [\alpha_{i,0} \dots \alpha_{i,i}] \begin{bmatrix} \mathbf{v}_0 \\ \vdots \\ \mathbf{v}_i \end{bmatrix} \\ &= \sum_{j=0}^i \alpha_{i,j} \mathbf{v}_j \end{aligned} \quad (2.47)$$

在位置 i 之前， $\mathbf{K}_{\leq i} = \begin{bmatrix} \mathbf{k}_0 \dots \mathbf{k}_i \\ \vdots \\ \vdots \end{bmatrix}$ 和 $\mathbf{V}_{\leq i} = \begin{bmatrix} \mathbf{v}_0 \dots \mathbf{v}_i \\ \vdots \\ \vdots \end{bmatrix}$ 是键和值。

在自注意力 $[\alpha_{i,0} \dots \alpha_{i,i}]$ 的原始版本中，假设其是密集的，即大部分值非零。在稀疏注意力中， $[\alpha_{i,0} \dots \alpha_{i,i}]$ 的一些条目被认为是非零的，其余条目在计算中简单地被忽略。假设 $G \subseteq \{0, \dots, i\}$ 是非零条目的索引集。对于语言模型，稀疏注意力模型在位置 i 的输出由以下公式给出

$$\text{Att}_{\text{sparse}}(\mathbf{q}_i, \mathbf{K}_{\leq i}, \mathbf{V}_{\leq i}) = \sum_{j \in G} \alpha'_{i,j} \mathbf{v}_j \quad (2.48)$$

这里 $\{\alpha'_{i,j}\}$ 已归一化到 G 。因此，它们的值与原始注意力权重不同（实际上我们有 $\alpha'_{i,j} > \alpha_{i,j}$ ）。模型的稀疏性由 G 的大小决定。稀疏注意力模型在定义 G 的方式上有所不同。一种简单的方法是基于启发式设计的模式来定义 G 。例如，一个广泛使用的模式涉及让 G 覆盖一个位于位置 i 附近的标记窗口 [Parmar 等人, 2018]。

虽然稀疏注意力通过使用稀疏操作减少了计算，但这些模型仍然存在重大限制，因为我们必须保留整个KV缓存（即， $\mathbf{K}_{\leq i}$ 和 $\mathbf{V}_{\leq i}$ ）

在推理过程中。如果序列非常长，存储此缓存将变得非常内存密集。为了解决这个问题，我们可以考虑一种不同的注意力模型形式，其中KV缓存不会被显式保留。线性注意力就是这样一种方法[Katharopoulos等人，2020]。它使用核函数 $\phi(\cdot)$ 将每个查询和键分别投影到点 $\mathbf{q}'_i = \phi(\mathbf{q}_i)$ 和 $\mathbf{k}'_i = \phi(\mathbf{k}_i)$ 。通过在这样变换下移除Softmax函数¹⁰，得到的注意力模型形式如下

$$\begin{aligned} \text{Att}_{\text{kv}}(\mathbf{q}_i, \mathbf{K}_{\leq i}, \mathbf{V}_{\leq i}) &\approx \text{Att}_{\text{linear}}(\mathbf{q}'_i, \mathbf{K}'_{\leq i}, \mathbf{V}_{\leq i}) \\ &= \frac{\mathbf{q}'_i \mu_i}{\mathbf{q}'_i \nu_i} \end{aligned} \quad (2.49)$$

在 μ_i 和 ν_i 是通过递归形式计算出的变量

$$\mu_i = \mu_{i-1} + \mathbf{k}'_i{}^T \mathbf{v}_i \quad (2.50)$$

$$\nu_i = \nu_{i-1} + \mathbf{k}'_i{}^T \mathbf{k}'_i \quad (2.51)$$

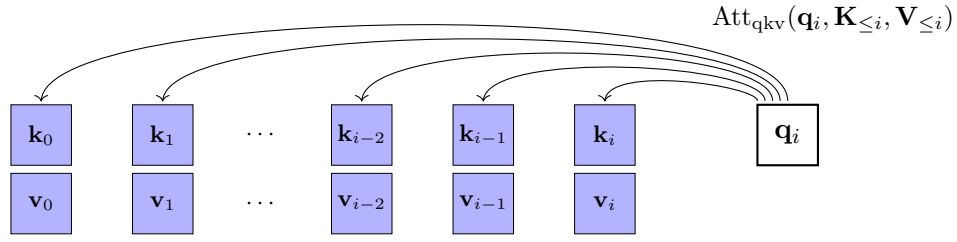
μ_i 并且 ν_i 可以被视为到达位置 i 的历史表示。这个模型的一个好处是我们不需要保留所有过去的查询和值。相反，只需使用最新的表示 μ_i 和 ν_i 。因此，每一步的计算成本是常数，并且该模型可以很容易地扩展以处理长序列。

实际上，当我们采用循环模型的观点时，这种对长序列的顺序方法自然产生。这些模型一次读取一个标记（或少量标记），使用这些输入更新循环状态，然后在下一个标记到达之前丢弃它们。每个步骤的输出仅基于循环状态，而不是所有先前状态。内存占用由固定大小的循环状态决定。循环模型可用于实时学习场景，其中数据以流的形式到达，并且可以在任何时间步进行预测。在NLP中，将循环模型应用于语言建模是学习序列表示的最早成功尝试之一。尽管Transformer已被用作LLM的基础架构，但循环模型仍然是强大的模型，尤其是在开发高效的LLM方面。最近，循环模型在语言建模中开始复兴，并被重新考虑为Transformer的有希望替代方案[Gu和Dao，2023]。图2.5显示了本小节中讨论的模型比较。

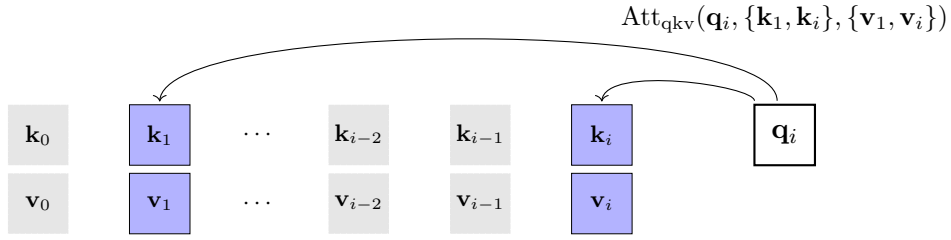
2.3.3 缓存和内存

基于标准Transformer架构的LLMs是全球模型。这些模型的推理涉及存储整个左上下文，以便为未来的标记做出预测。这需要一个KV缓存，其中包含所有先前生成的表示（即键和值）。

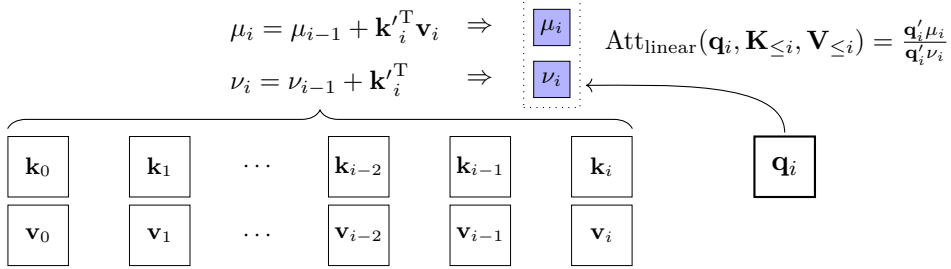
¹⁰In the new space after this transformation, the Softmax normalization can be transformed into the simple scaling normalization.



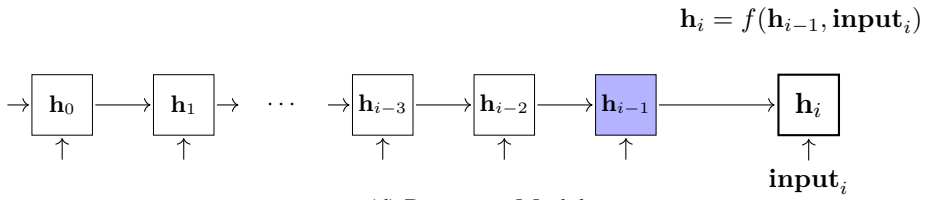
(a) Standard Self-attention



(b) Sparse Attention



(c) Linear Attention



(d) Recurrent Models

图2.5: 自注意力、稀疏注意力、线性注意力和循环模型的说明。蓝色方框 = 缓存状态以产生位置 i 的输出。 $f(\cdot)$ = 一个循环单元。

令牌被保留，缓存成本随着推理的进行而增长。在上面，我们讨论了通过高效注意力方法，如稀疏注意力和线性注意力，来优化此缓存的方法。另一个可能与前文讨论重叠的想法是通过额外的记忆模型显式编码上下文。

2.3.3.1 固定大小 KV 缓存

一种直接的方法是使用固定大小的内存模型来表示键和值。假设我们有一个内存Mem，它保留上下文信息。我们可以将位置 i 的注意力操作写成一般形式

$$\text{Att}(\mathbf{q}_i, \text{Mem}) = \text{Att}_{\text{qkv}}(\mathbf{q}_i, \mathbf{K}_{\leq i}, \mathbf{V}_{\leq i}) \quad (2.52)$$

在这个模型中，Mem 简单地是 KV 缓存，即 $\text{Mem} = (\mathbf{K}_{\leq i}, \mathbf{V}_{\leq i})$ 。因此，Mem 的大小由 i 决定。如果我们把 Mem 定义为一个固定大小的变量，那么执行 $\text{Att}(\mathbf{q}_i, \text{Mem})$ 的成本将是固定的。设计 Mem 有几种不同的方法。

- 一种最简单的方法是考虑一个固定大小的先前键和值窗口。因此，Mem 由以下公式给出

$$\text{Mem} = (\mathbf{K}_{[i-n_c+1, i]}, \mathbf{V}_{[i-n_c+1, i]}) \quad (2.53)$$

n_c 表示窗口大小。记号 $\mathbf{K}_{[i-n_c+1, i]}$ 和 $\mathbf{V}_{[i-n_c+1, i]}$ 表示从 $i - n_c + 1$ 到 i 的键和值。¹¹ 这个模型可以看作是一种局部注意力模型。

- 它还可以将Mem定义为两个摘要向量的对，这导致历史记录的更紧凑表示。总结先前键和值的一个简单方法就是使用它们的移动平均值。例如，Mem可以定义为先前 n_c 键和值的未加权移动平均值

$$\text{Mem} = \left(\frac{\sum_{j=i-n_c+1}^i \mathbf{k}_j}{n_c}, \frac{\sum_{j=i-n_c+1}^i \mathbf{v}_j}{n_c} \right) \quad (2.54)$$

另一种方法是使用加权移动平均

$$\text{Mem} = \left(\frac{\sum_{j=i-n_c+1}^i \beta_{j-i+n_c} \mathbf{k}_j}{\sum_{j=1}^{n_c} \beta_j}, \frac{\sum_{j=i-n_c+1}^i \beta_{j-i+n_c} \mathbf{v}_j}{\sum_{j=1}^{n_c} \beta_j} \right) \quad (2.55)$$

这里 $\{\beta_1, \dots, \beta_{n_c}\}$ 是系数，它们可以是作为模型参数学习，或者通过启发式方法确定。例如，它们可以被设置为递增系数（即， $\beta_1 < \beta_2 < \dots < \beta_{n_c-1} < \beta_{n_c}$ ），以便给靠近 i 的位置更大的权重。我们可以将移动平均扩展到包括所有位置直到 i 。这导致键和值的累积平均值，以以下形式给出

$$\text{Mem} = \left(\frac{\sum_{j=0}^i \mathbf{k}_j}{i+1}, \frac{\sum_{j=0}^i \mathbf{v}_j}{i+1} \right) \quad (2.56)$$

¹¹更正式地，我们写作 $\mathbf{K}_{[i-n_c+1, i]}$ 和 $\mathbf{V}_{[i-n_c+1, i]}$ 。有时为了简化符号，我们用 $\{\mathbf{k}_{i-n_c+1}, \dots, \mathbf{k}_i\}$ 表示 $\mathbf{K}_{[i-n_c+1, i]}$ ，用 $\{\mathbf{v}_{i-n_c+1}, \dots, \mathbf{v}_i\}$ 表示 $\mathbf{V}_{[i-n_c+1, i]}$ 。

通常， t 他累积平均值可以用递归公式表示 公式

$$\text{Mem}_i = \frac{(\mathbf{k}_i, \mathbf{v}_i) + i \cdot \text{Mem}_{i-1}}{i + 1} \quad (2.57)$$

当前位置和前一个位置的累积平均值分别表示为 Mem_i 和 Mem_{i-1} 。该模型的一个优点是，在推理过程中我们只需要存储一个键值对，而不是存储所有的键值对。请注意，上述内存模型与循环模型相关，并且已经使用更先进的技术来开发Transformer中自注意力机制的替代方案 [Ma等, 2023]。

- 内存Mem也可以是一个神经网络。在每一步，它都以前一个内存的输出和当前模型的状态作为输入，并产生新的内存输出。这个神经网络可以表示为以下函数

$$\text{Mem} = \text{Update}(S_{kv}, \text{Mem}_{\text{pre}}) \quad (2.58)$$

当前步骤和前一步骤的内存输出分别表示为Mem和 Mem_{pre} 。 S_{kv} 是一组键值对，表示模型的最近状态。这种公式是通用的，允许我们通过选择不同的 $\text{Update}(\cdot)$ 和 S_{kv} 配置来开发各种内存模型。例如，如果 S_{kv} 只包含最新的键值对 $(\mathbf{k}_i, \mathbf{v}_i)$ 并且 $\text{Update}(\cdot)$ 被定义为循环单元，那么式(2.58)可以表示为一个类似RNN的模型

$$\text{Mem} = f((\mathbf{k}_i, \mathbf{v}_i), \text{Mem}_{\text{pre}}) \quad (2.59)$$

在 $f(\cdot)$ 是一个循环细胞的情况下。为了提高效率，循环也可以应用于段级建模。一种简单的方法是将序列划分为段，并将 S_{kv} 视为一个段。将循环模型应用于 $\text{Update}(\cdot)$ 将导致在段上操作的内存模型。一个特殊例子是我们将 $\text{Update}(\cdot)$ 定义为一个 FIFO 函数，该函数将 S_{kv} 添加到内存中，并从内存中移除最旧的键值段，给出

$$\text{Mem} = \text{FIFO}(S_{kv}, \text{Mem}_{\text{pre}}) \quad (2.60)$$

考虑一个包含两个段落的内存，一个用于当前段，另一个用于前一段。在注意力操作中，每个位置可以访问两个最接近的连续段的历史键值对。这本质上定义了一个局部内存，但及其变体已被广泛用于段级循环模型 [Dai 等人, 2019; Hutchins 等人, 2022; Bulatov 等人, 2022]。

- 上述内存模型可以扩展以涉及多个内存。这种方法的例子是压缩Transformer [Rae等人, 2019]。它采用两个不同的固定大小内存：一个用于建模局部上下文（表示为Mem），另一个用于建模和压缩长期历史（表示为CMem）。该模型中的KV缓存是Mem和CMem的组合。注意力函数可以写成

$$\text{Att}_{\text{com}}(\mathbf{q}_i, \text{Mem}, \text{CMem}) = \text{Att}_{\text{qkv}}(\mathbf{q}_i, [\text{Mem}, \text{CMem}]) \quad (2.61)$$

在[Mem, CMem]是Mem和CMem的联合内存。与其他段级模型一样，压缩Transformer模型在序列的段上操作。每个段是一系列 n_s 个连续的标记，我们用 S_{kv}^k 表示对应于第 k 个段标记的关键值对。当一个新的段到达时，Mem以FIFO的方式更新：我们将 S_{kv}^k 中的 n_c 个键值对追加到Mem中，然后从Mem中弹出 n_s 个最旧的键值对，该值由

$$\text{Mem} = \text{FIFO}(S_{kv}^k, \text{Mem}_{\text{pre}}) \quad (2.62)$$

弹出的键值对随后用于更新压缩内存CMem。这些 n_s 键值对通过压缩网络压缩成 $\frac{n_s}{c}$ 键值对。CMem是一个FIFO，将压缩的 $\frac{n_s}{c}$ 键值对追加到队列尾部，并丢弃队列中的第一个 $\frac{n_s}{c}$ 键值对。它由以下给出

$$\text{CMem} = \text{FIFO}(C_{kv}^k, \text{CMem}_{\text{pre}}) \quad (2.63)$$

C_{kv}^k 表示压缩后的键值对集合。在压缩Transformer模型中隐含的是，局部上下文应以最小信息损失显式表示，而长距离上下文可以更压缩。

- 我们已经看到，全局和局部上下文都很有用，可以使用注意力模型进行建模。这种观点促使我们将注意力模型扩展到结合局部和长期记忆[Ainslie等人, 2020; Zaheer等人, 2020; Gupta和Berant, 2020]。一种简单但广泛使用的方法是将序列的前几个标记纳入注意力中，作为全局标记。这种方法通常与其他稀疏注意力模型一起应用。将序列的全局标记纳入的优点是，它有助于平滑在注意力权重计算中使用的Softmax函数的输出分布，从而在上下文大小非常大时稳定模型性能[Xiao等人, 2024]。然而，一个缺点是，使用固定大小的全局内存可能会导致信息丢失。在处理长序列时，我们需要扩大KV缓存以充分表示上下文，但这反过来又增加了计算成本。

图2.6展示了上述方法的插图。请注意，虽然我们在此处关注KV缓存的优化，但这个问题与上一节中讨论的问题密切相关。我们迄今为止提到的所有方法都可以广泛归类为高效注意力方法，这些方法在各种Transformer变体中得到广泛应用。

2.3.3.2 基于内存的模型

上述讨论的记忆建模基于KV缓存的更新，得到的模型通常被称为内部记忆。我们现在考虑另一组模型，称为外部记忆，这些模型作为独立模型来访问LLMs的大规模上下文。许多此类模型基于在机器学习中广泛讨论的记忆方法[Bishop, 2006]。一个常见的例子是最邻近算法：我们在数据存储中存储上下文表示，并尝试找到最相似

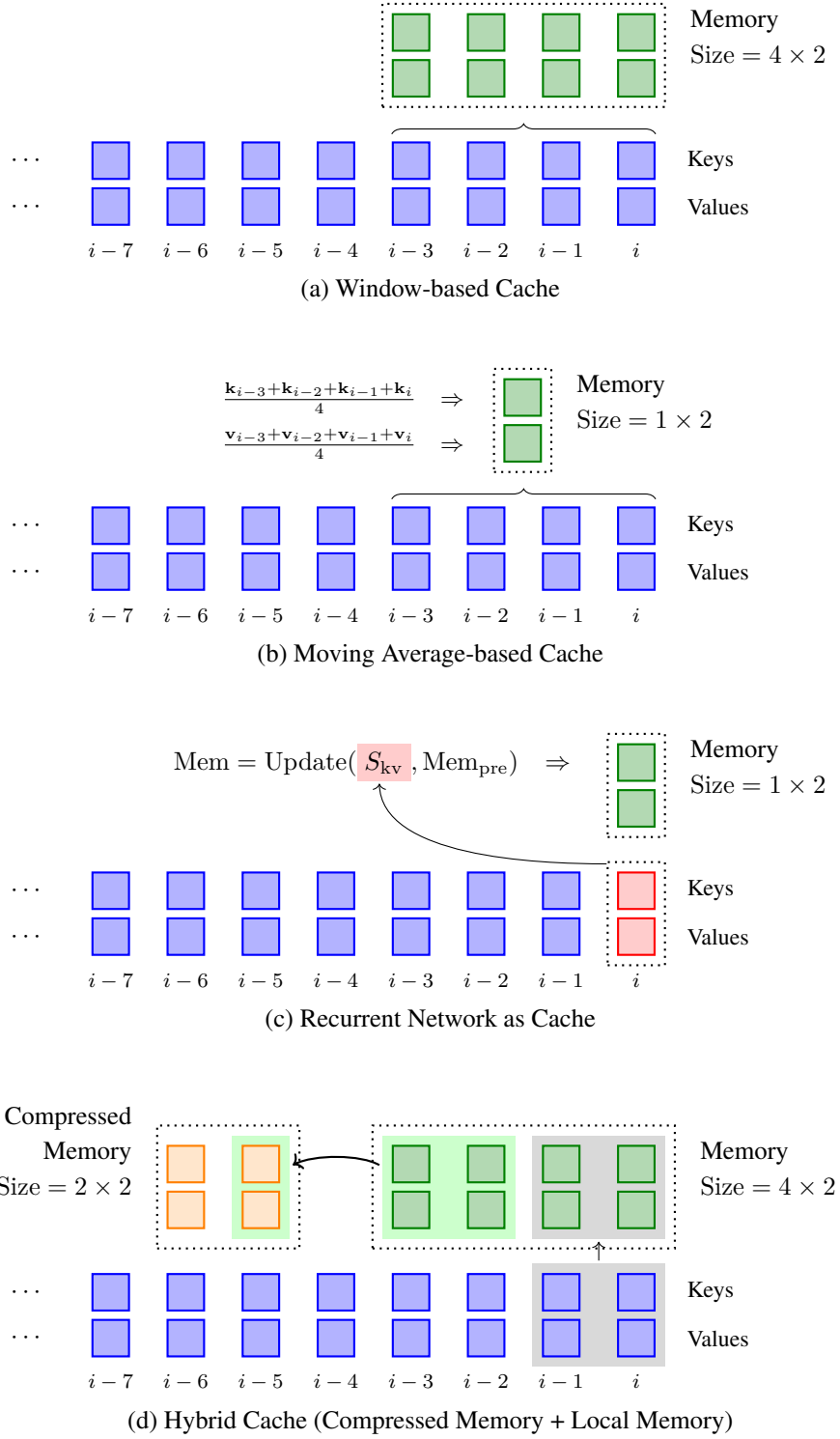


图2.6: LLM中固定大小KV缓存的示意图。蓝色方框表示LLM推理过程中生成的键和值，绿色方框表示存储或编码在主内存中的键和值，橙色方框表示存储或编码在压缩内存中的键和值。

存储表示以匹配给定查询。检索到的上下文表示随后用于提高此查询的注意力。

这里，我们考虑 k -最近邻 (k -NN) 方法，这是最流行的基于内存的方法之一。由于本节的重点是语言建模，我们将数据存储器中的一个样本定义为与某些上下文状态相对应的键值对。请注意，“上下文”在这里是一个广泛的概念，而不仅仅是文本生成中的序列前缀。例如，有人可能会将整个数据集视为预测标记的上下文。这使我们能够在一系列序列中检索最接近的上下文情况，而不是给定的序列前缀。尽管在本小节中我们将限制自己仅对单个序列的上下文建模，但我们讨论的是一个相对更一般的情况。

假设我们有一个键集 $\{k_j\}$ ，其对应值为 $\{v_j\}$ ，并且我们把这些键值对存储在一个向量数据库¹²中。对于每个查询 q_i ，我们通过扩大以 q_i 为中心的球面半径，直到它包含 k 个数据点在 $\{k_j\}$ 中，来找到其 k 个最近邻。这导致了一组 k 个键及其对应值，表示为 Mem_{knn} 。与之前一样，我们将 Mem 表示为查询的本地内存，例如相邻标记的 KV 缓存。我们的目标是让查询 q_i 关注本地内存 Mem 和长期内存 Mem_{knn} 。当然，有几种方法可以将 Mem 和 Mem_{knn} 纳入注意力模型。例如，我们可能简单地将它们组合成一个单一的 KV 缓存 $[\text{Mem}, \text{Mem}_{knn}]$ ，并通过标准的 QKV 注意力将 q_i 关注到 $[\text{Mem}, \text{Mem}_{knn}]$ 。或者我们可能使用 Mem 和 Mem_{knn} 在单独的注意力步骤中。这种方法的例子是 Wu 等人 [2021] 开发的模型。它线性组合了两种类型的注意力，给出

$$\text{Att}(q_i, \text{Mem}, \text{Mem}_{knn}) = g \odot \text{Att}_{\text{local}} + (1 - g) \odot \text{Att}_{knn} \quad (2.64)$$

$$\text{Att}_{\text{local}} = \text{Att}(q_i, \text{Mem}) \quad (2.65)$$

$$\text{Att}_{knn} = \text{Att}(q_i, \text{Mem}_{knn}) \quad (2.66)$$

这里 $g \in \mathbb{R}^d$ 是系数向量，它可以是一个学习到的门的输出。

给定上述基于 k -NN 的内存模型，剩余的任务是确定哪些键值对保留在数据存储中。对于标准语言建模任务，我们将序列中之前看到的标记视为上下文，因此我们可以将这些标记的所有键和值添加到数据存储中。在这种情况下，基于 k -NN 的注意力模型本质上等同于一个稀疏注意力模型 [Gupta 等人, 2021]。

另一种方法是，我们可以将上下文从一个序列扩展到一系列序列。例如，我们可能会收集训练数据集中所有序列中的键值对并将它们添加到数据存储中，以建模更大的上下文。因此，LLMs 可以根据广义上下文预测标记。这种方法的一个问题是，如果涉及许多序列，计算成本会很大。由于这些序列是我们训练数据的一部分，我们可以在运行 LLMs 之前在数据存储中为向量构建和优化索引。因此，检索相似向量的效率可以非常高，就像在大多数向量数据库中一样。

实际上，上述所有方法都可以被视为基于检索的方法的实例。我们不仅可以利用检索结果来改进注意力，还可以以其他方式应用这种方法。 k -NN-based 搜索的一个应用是 k -NN 语言模型（或 k -NN LM）[Khandelwal 等人, 2020]。其想法是，尽管它试图通过在表示学习中结合最近邻来扩展自注意力中使用的上下文，但在实践中，Transformers 中的相似隐藏状态通常对后续位置中的相似标记有很高的预测性。

¹²A vector database, or vector store, is a database that provides highly optimized retrieval interfaces for finding stored vectors that closely match a query vector.

在 k -NN LM 中，数据存储库中的每个项目都是一个键值对 (z, w) ，其中 z 代表 LLM 在某个位置的隐藏状态， w 代表相应的预测。创建数据存储库的一种典型方法是从训练数据集中收集每个序列每个位置的 Transformer 层堆叠的输出向量和相应的下一个标记。在推理期间，给定一个前缀，我们有一个表示 \mathbf{h}_i 。给定这个表示，我们首先在数据存储库中搜索 k 个最接近匹配的数据项 $\{(z_1, w_1), \dots, (z_k, w_k)\}$ 。这里 $\{w_1, \dots, w_k\}$ 被认为是预测的参考标记，因此可以用来根据 \mathbf{h}_i 指导标记预测。利用参考标记的一种常见方法是在词汇表 V 上定义一个分布，

$$\Pr_{knn}(\cdot|\mathbf{h}_i) = \text{Softmax}([-d_0 \quad \dots \quad -d_{|V|}]) \quad (2.67)$$

d_v 等于 \mathbf{h}_i 和 z_j 之间的距离，如果 w_j 等于 V 的 v -th 条目，否则等于 0。我们使用一个具有系数 λ 的线性函数，在基于检索的分布 $\Pr_{knn}(\cdot|\mathbf{h}_i)$ 和 LLM 输出分布 $\Pr_{lm}(\cdot|\mathbf{h}_i)$ 之间进行插值。

$$\Pr(\cdot|\mathbf{h}_i) = \lambda \cdot \Pr_{knn}(\cdot|\mathbf{h}_i) + (1 - \lambda) \cdot \Pr_{lm}(\cdot|\mathbf{h}_i) \quad (2.68)$$

然后，像往常一样，我们可以通过最大化概率 $\Pr(y|\mathbf{h}_i)$ 来选择下一个标记 y 。

与信息检索 (IR) 系统一样，数据存储库也可以管理文本并提供对相关文本的访问，以供查询。例如，我们可以在具有全文索引的搜索引擎中存储一组文本文档，然后搜索与给定基于文本的查询匹配的文档。将 IR 技术应用于 LLMs 导致了一个称为检索增强生成 (RAG) 的通用框架。RAG 框架的工作原理如下。我们使用上下文 \mathbf{x} 作为查询，并通过高效的 IR 技术从数据存储库中找到最相关的文档片段 $k \{c_1, \dots, c_k\}$ 。这些搜索结果通过提示模板 $g(\cdot)$ ¹⁴ 与原始上下文相结合，从而为 LLM 生成增强输入。

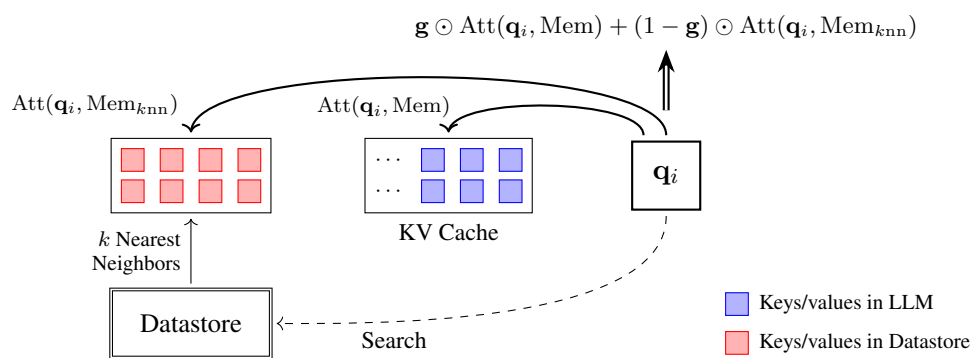
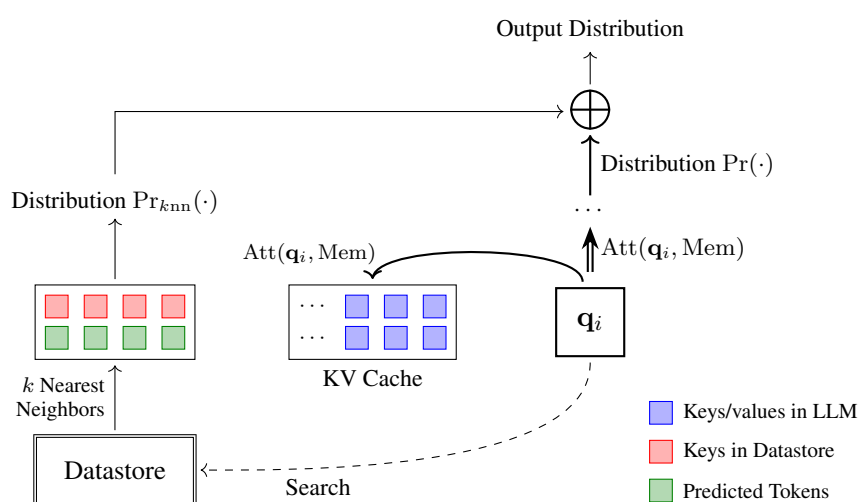
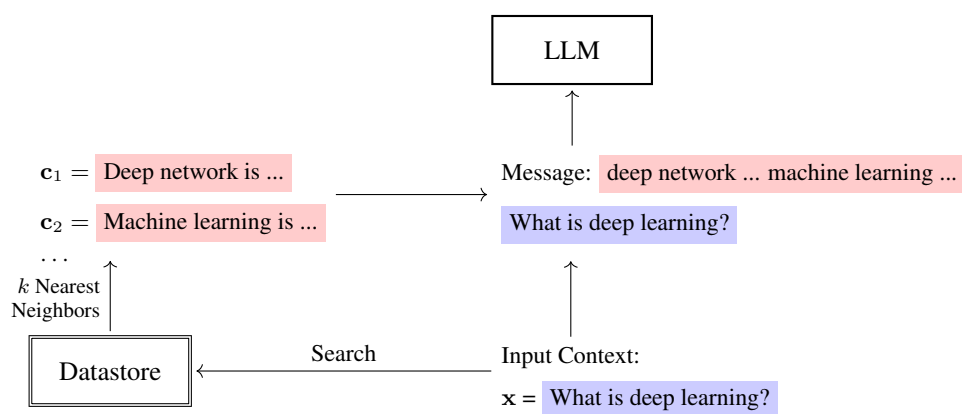
$$\mathbf{x}' = g(c_1, \dots, c_k, \mathbf{x}) \quad (2.69)$$

然后，我们使用 \mathbf{x}' 作为上下文，并使用模型 $\Pr(y|\mathbf{x}')$ 预测以下文本。RAG 的一个优点是我们不需要修改 LLMs 的架构，而是通过额外的 IR 系统增强 LLMs 的输入。图 2.7 显示了在 LLMs 中使用不同外部内存的比较。

¹³In piratical applications, queries are typically generated using a query generation system, which may expand it with variations of tokens and query intent.

¹⁴For example, the template could be:

消息 = $\{ *c_1 * \} \dots \{ *c_k * \}$ 输入: $\{ *x * \}$ 输出:

(a) k -NN 搜索增强注意力(b) k -NN 语言模型

(c) Retrieval-augmented Generation

图2.7: 语言模型的外部存储（或数据存储）的示意图。

2.3.3.3 内存容量

一个LLM中的内存模型，以简单的键值缓存或数据存储的形式，可以广泛地被视为上下文信息的编码器。理想情况下，在我们说一个内存模型在标记预测中代表了整个上下文之前，我们需要确保该模型可以准确地表示上下文的任何部分。标准的KV缓存就是这样一种模型，它完全存储了所有过去的历史。在这种情况下，该模型被认为具有足够的记忆上下文的能力。然而，在许多实际应用中，并不需要完全记忆。相反，目标是使LLM能够访问重要的上下文信息。因此，开发了高效且压缩的内存模型，如本节所述。请注意，序列越长，对于低容量内存模型来说，捕捉重要上下文信息就越困难。因此，在处理长上下文时，通常的做法是简单地增加模型容量。

虽然高容量模型通常是有利的，但它们难以训练和部署。一个具有挑战性的场景是标记以流的形式到达，上下文持续增长。为这类任务开发LLM是困难的，因为我们需要在极长的序列上训练Transformer。解决这种困难的一种可能方法是使用非参数方法，例如基于检索的方法。例如，如上所述，我们可以使用向量数据库来存储先前生成的键值对，从而通过这种外部记忆模型来表示上下文。尽管这种方法绕过了在Transformer中表示长上下文的挑战，但构建和更新外部记忆模型在计算上非常昂贵。这些模型更常用于上下文在推理过程中预先给出并固定的问题，因此不适合流上下文建模。

在上下文大小持续增长的情况下，应用固定大小的内存模型是一种常用的方法。例如，在循环模型中，任意长度的序列可以被总结为一组隐藏状态，这使得我们每一步的计算成本固定。虽然循环模型最初在深度学习早期应用于NLP时被发现不太擅长处理序列建模中的长距离依赖，但最近的进展表明，它们的变体现在在建模极长序列方面非常有效。[Bulatov等人，2022；Hutchins等人，2022；Munkhdalai等人，2024；Ma等人，2024]。

没有LLMs中内存容量的通用定义。一种简单的方法可能考虑用于保留上下文信息所需的存储量。例如，内存容量可以通过Transformers中的KV缓存大小或检索方法中使用的向量数据库的大小来定义。相关概念是模型复杂性。在机器学习中，有几种方法可以定义模型的复杂性。其中最简单的方法之一是计数参数数量。然而，应强调的是，这里讨论的内存模型主要服务于存储信息，而不是增加可训练参数。因此，具有大内存容量的模型不一定更复杂。尽管如此，在实践中确定内存模型的容量并不简单。一般来说，我们需要在最大化性能和控制内存占用之间进行权衡。

2.3.4 头和层之间的共享

在Transformers中，KV缓存是一种可以在多个维度上动态调整的数据结构，例如头、层和序列长度。例如，考虑一个具有 L 层的LLM。每一层有 τ 个注意力头，每个头产生一个 d_h -维输出。在

推理过程中，我们存储最多 m 个标记的键和值。此缓存机制的时空复杂度为 $O(L \cdot \tau \cdot d_h \cdot m)$ 。正如我们之前所看到的，可以通过缓存更少的标记的键和值来降低这种复杂度。例如，在滑动窗口注意力中，使用固定大小的窗口来缓存局部上下文中的键和值。并且该模型的空间复杂度为 $O(L \cdot \tau \cdot d_h \cdot m_w)$ ，其中 m_w 是窗口的大小。

除了减少 m ，我们还可以在其他维度上减小 KV 缓存的大小。一个广泛使用的方法是在多头自注意力中启用跨头共享。回想第 2.1.1 节，多头自注意力使用多个查询、键和值的集合（每个集合称为一个头），每个头都执行常规的 QKV 注意力机制。这可以表示为

$$\text{Output} = \text{Merge}(\text{head}_1, \dots, \text{head}_\tau) \mathbf{W}^{\text{head}} \quad (2.70)$$

在何处头 $j \in \mathbb{R}^{d_h}$ 使用标准 QKV 注意力函数计算

$$\text{head}_j = \text{Att}_{\text{qkv}}(\mathbf{q}_i^{[j]}, \mathbf{K}_{\leq i}^{[j]}, \mathbf{V}_{\leq i}^{[j]}) \quad (2.71)$$

这里， $\mathbf{q}_i^{[j]}$ 、 $\mathbf{K}_{\leq i}^{[j]}$ 和 $\mathbf{V}_{\leq i}^{[j]}$ 是投影到第 j 个特征子空间的查询、键和值。因此，这个模型可以解释为对一组特征子空间进行并行注意力（见图 2.8 (b)）。KV 缓存需要保留所有这些头部的键和值，即 $\{(\mathbf{K}_{\leq i}^{[1]}, \mathbf{V}_{\leq i}^{[1]}), \dots, (\mathbf{K}_{\leq i}^{[\tau]}, \mathbf{V}_{\leq i}^{[\tau]})\}$ 。

一种对多头注意力模型的改进，称为多查询注意力（MQA），是在头部之间共享键和值，同时允许查询对每个头部都是唯一的 [Shazeer, 2019]。在 MQA 中，存在一组单一的键和值 $(\mathbf{K}_{\leq i}, \mathbf{V}_{\leq i})$ 。此外，还有 τ 个查询 $\{\mathbf{q}_i^{[1]}, \dots, \mathbf{q}_i^{[\tau]}\}$ ，每个对应不同的头部。对于每个头部，我们有

$$\text{head}_j = \text{Att}_{\text{qkv}}(\mathbf{q}_i^{[j]}, \mathbf{K}_{\leq i}, \mathbf{V}_{\leq i}) \quad (2.72)$$

图 2.8 (c) 说明了此模型。通过共享键和值，KV 缓存的尺寸将是 $O(L \cdot d_h \cdot m)$ 。

分组查询注意力（GQA）是多头注意力和 MQA [Ainslie et al., 2023] 的自然扩展。在 GQA 中，头被分为 n_g 组，每组对应一组共享的键和值。因此，我们有 n_g 组键和值 $\{(\mathbf{K}_{\leq i}^{[1]}, \mathbf{V}_{\leq i}^{[1]}), \dots, (\mathbf{K}_{\leq i}^{[n_g]}, \mathbf{V}_{\leq i}^{[n_g]})\}$ 。参见图 2.8 (d) 的说明。设 $g(j)$ 为第 j -th 个头的组 ID。GQA 模型可以表示为

$$\text{head}_j = \text{Att}_{\text{qkv}}(\mathbf{q}_i^{[j]}, \mathbf{K}_{\leq i}^{[g(j)]}, \mathbf{V}_{\leq i}^{[g(j)]}) \quad (2.73)$$

GQA 的 KV 缓存大小为 $O(L \cdot n_g \cdot d_h \cdot m)$ 。GQA 的一个好处是，我们可以通过调整 n_g 在计算效率和模型表达能力之间进行权衡。当 $n_g = \tau$ 时，该模型成为标准的多头注意力模型。相比之下，当 $n_g = 1$ 时，它成为 GQA 模型。

共享也可以在层之间进行。这种方法属于共享权重和共享激活方法家族，这些方法已在 Transformers 中得到了广泛的应用 [Dehghani 等人, 2018; Lan 等人, 2020]。例如，可以共享 KV 激活或注意力

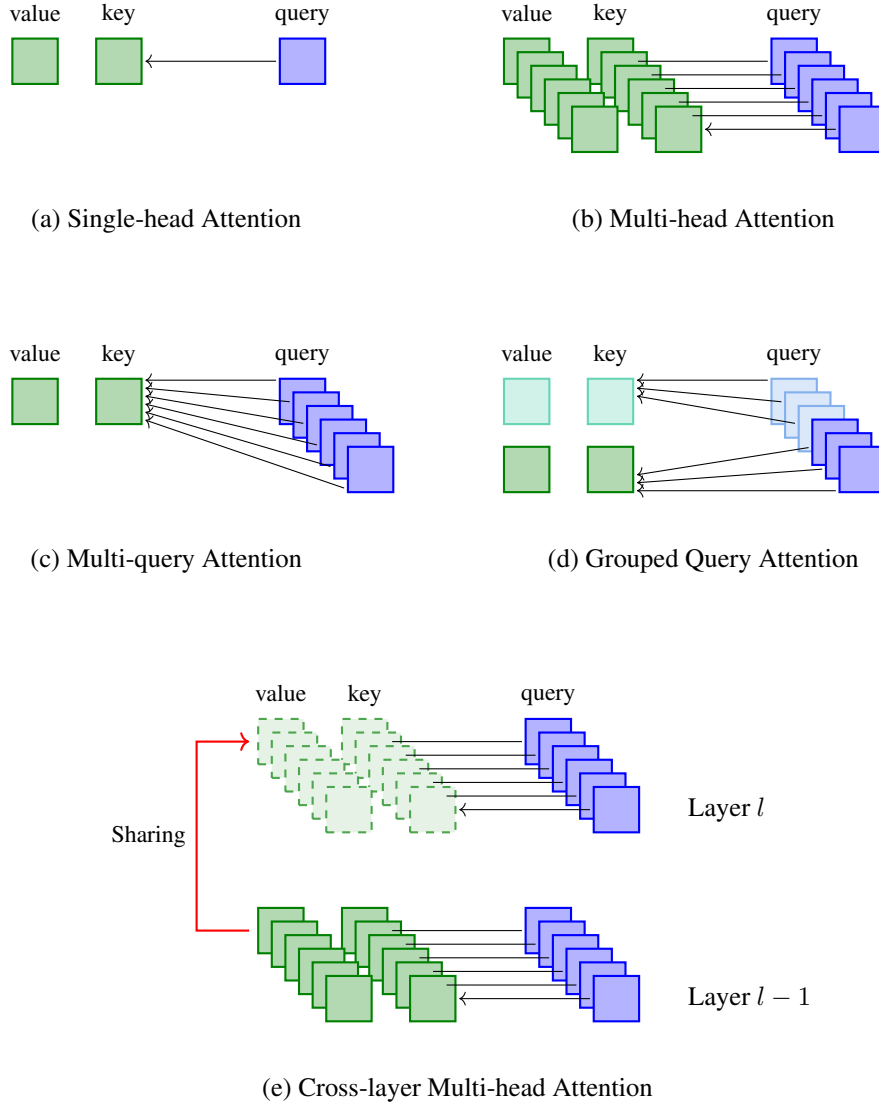


图2.8: 基于不同多头和共享机制的QKV注意力示意图。(a) =单头注意力, 和 (b-e) =多头注意力。

权重跨层以减少计算和内存占用 [Xiao 等人, 2019; Brandon 等人, 2024]。图 2.8 (e) 展示了该方法的一个示意图, 其中一层中的查询直接访问较低层级的 KV 缓存。

2.3.5 位置外推和内插

由于Transformer层对输入无序, 我们需要在输入标记中编码位置信息。为此, 通常会在标记嵌入中添加位置嵌入, 然后将这些组合嵌入作为输入馈送到Transformer层堆栈。在这种情况下, 位置 i 的嵌入可以表示为

$$\mathbf{e}_i = \mathbf{x}_i + \text{PE}(i) \quad (2.74)$$

$\mathbf{x}_i \in \mathbb{R}^d$ 表示标记嵌入， $\mathbf{PE}(i) \in \mathbb{R}^d$ 表示位置嵌入。一般来说，标记嵌入 \mathbf{x}_i 是一个位置无关的向量，因此使用位置嵌入 $\mathbf{PE}(i)$ 来编码位置上下文。一种直接的方法是将 $\mathbf{PE}(i)$ 视为可学习的变量，并与其他模型参数一起进行训练。这样，我们可以为每个位置学习一个独特的表示，从而区分序列中不同位置出现的标记。

使用学习向量表示的位置可以在训练和测试时序列长度相似的任务中表现良好。然而，在实践中，我们通常在训练过程中对序列长度施加限制，以防止过高的计算成本，但希望在推理过程中应用于更长的序列。在这种情况下，使用学习到的位置嵌入具有明显的缺点，因为训练阶段没有观察到位置的嵌入。

一种建模位置信息的替代方法是开发可以泛化的位置嵌入：一旦训练完成，嵌入模型就可以用来处理更长的序列。假设我们在最大长度为 m_l 的序列上训练一个位置嵌入模型，并且我们希望将该训练模型应用于长度为 m ($m \gg m_l$) 的序列。如果嵌入模型在从训练数据中观察到的位置范围有限，那么这个模型将无法处理该范围之外的新数据。参见图 2.9 (a) 的说明，其中，如果嵌入模型缺乏外推能力，则无法对训练域之外的数据点进行建模。

有几种方法可以使位置嵌入模型泛化。它们可以分为两类。

- 外推。在观测数据点（即位置）上学习的模型可以直接用于为原始范围之外的数据点分配有意义的值。例如，假设我们有一系列数字 1, 2, ..., 10，我们想要理解一个新数字 15 的含义。知道这些数字是用于排序的自然数，我们可以轻松推断出 15 是一个在 10 之后出现的数字，即使 15 之前没有被观察到。图 2.9 (b) 展示了这种方法的一个例子，其中学习到一个函数来拟合特定范围内的数据点，然后将其应用于估计该范围之外的数据点的值。
- 插值。这种方法将更大的数据点范围映射到原始观测范围。例如，假设我们有一个设计用于范围 [1, 10] 的模型。当给定新的范围 [1, 20] 时，我们可以通过将每个数字除以 2 来缩小它，从而将所有数字都拟合到 [1, 10]。这种缩放允许我们使用在范围 [1, 10] 上训练的模型来描述扩展范围 [1, 20] 中的数据点。参见图 2.9 (c) 以了解此方法的说明。

实际上，许多系统中的位置嵌入已经实现了一定程度的一般化。例如，正弦编码，最常见的位置嵌入方法，使用正弦和余弦函数，这些函数可以自然地扩展到任何长度的序列。尽管这种方法看起来直接且简单，但在我们显著扩展序列进行处理时，它并不表现良好。在本小节中，我们将讨论几种基于外推或内插的替代方法。

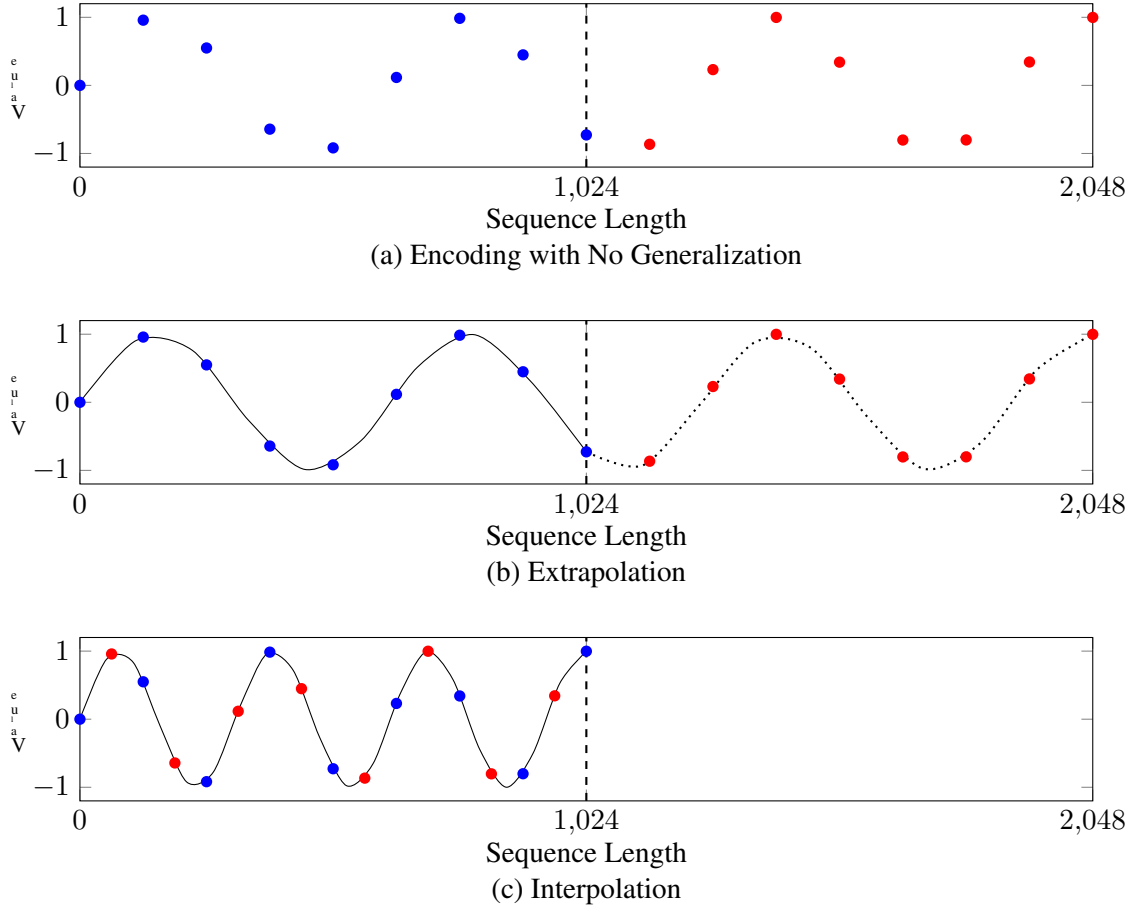


图2.9: 不同位置嵌入方法在一系列位置上的示意图。蓝色点代表训练过程中观察到的位置, 红色点代表测试时新观察到的位置。在子图 (a) 中, 编码模型只记住训练过程中看到的点, 无法泛化。在子图 (b) 和 (c) 中, 模型可以通过外推和内插进行泛化。

2.3.5.1 可学习偏差的注意力

一个与公式 (2.74) 相关的问题是嵌入模型独立地处理每个标记, 因此忽略了不同标记之间的距离。对此模型的一种常见改进, 称为相对位置嵌入, 是考虑标记之间的成对关系[Shaw等人, 2018]。这一想法的总体思路是获取任意一对位置的偏移量并将其纳入自注意力模型。具有相对位置嵌入的最简单自注意力形式之一如下:

$$\text{Att}_{\text{qkv}}(\mathbf{q}_i, \mathbf{K}_{\leq i}, \mathbf{V}_{\leq i}) = \sum_{j=0}^i \alpha(i, j) \mathbf{v}_j \quad (2.75)$$

$$\alpha(i, j) = \text{Softmax}\left(\frac{\mathbf{q}_i \mathbf{k}_j^T + \text{PE}(i, j)}{\sqrt{d}} + \text{Mask}(i, j)\right) \quad (2.76)$$

此模型与原始自注意力模型之间的唯一区别是, 在此新模型中, 查询-键乘积中添加了一个偏置项 $\text{PE}(i, j)$ 。直观上, $\text{PE}(i, j)$ 可以解释为位置对 i 和 j 的距离惩罚。当 i 离开 j 时, 该值

$\text{PE}(\{v^*\})$ 减少。

$\text{PE}(\{v^*\})$ 可以用几种不同的方式定义。在这里，我们考虑相对位置嵌入的 T5 版本，称为 T5 偏置 [Raffel 等人, 2020]。对于每个查询 q_i 和键 k_j 的每一对，它们之间的偏移被定义为¹⁵

$$d(i, j) = i - j \quad (2.77)$$

一个简单的偏置 $\text{PE}(i, j)$ 设计是为具有相同偏移量的所有查询-键对共享相同的可学习变量，即 $\text{PE}(i, j) = u_{i-j}$ ，其中 u_{i-j} 是对应于偏移量 $i - j$ 的变量。然而，简单地为每个偏移量分配一个唯一值将限制此模型仅限于观察到的偏移量。当 $i - j$ 大于最大训练偏移量时，模型无法泛化。

T5 偏差反而采用该模型的一般化。它不是为每个查询-键偏移分配一个唯一的偏差项，而是将差异偏移分组到“桶”中，每个桶对应一个可学习的参数。更具体地说， $n_b + 1$ 桶的偏差项如下所示。

- 对于桶 0 到 $\frac{n_b+1}{2} - 1$ ，每个桶对应一个偏移量，即桶 0 \leftrightarrow 偏移量 0，桶 1 \leftrightarrow 偏移量 1，桶 2 \leftrightarrow 偏移量 2，依此类推。我们将其表示为 $b(i - j) = i - j$ 。
- 对于桶 $\frac{n_b+1}{2}$ 到 n_b ，每个桶的大小以对数方式增加。例如，给定偏移量 $i - j \geq \frac{n_b+1}{2}$ 的桶号可以定义为

$$b(i - j) = \frac{n_b + 1}{2} + \left\lfloor \frac{\log(i - j) - \log(\frac{n_b+1}{2})}{\log(\text{dist}_{\max}) - \log(\frac{n_b+1}{2})} \cdot \frac{n_b + 1}{2} \right\rfloor \quad (2.78)$$

在参数 dist_{\max} 通常被设置为一个相对较大的数字，以指示我们可能遇到的最大的偏移量。

- 当 $i - j > \text{dist}_{\max}$ 时，我们将 $i - j$ 放入最后一个桶中。换句话说，桶 n_b 包含所有未分配给前一个桶的偏移量。

一起，这些可以表示为函数

$$b(i - j) = \begin{cases} i - j & 0 \leq i - j < \frac{n_b+1}{2} \\ \min(n_b, \frac{n_b+1}{2} + \left\lfloor \frac{\log(i-j) - \log(\frac{n_b+1}{2})}{\log(\text{dist}_{\max}) - \log(\frac{n_b+1}{2})} \cdot \frac{n_b+1}{2} \right\rfloor) & i - j \geq \frac{n_b+1}{2} \end{cases} \quad (2.79)$$

图 2.10 展示了这些桶的示意图。我们看到在前半部分的桶中，每个桶只与 $i - j$ 的一个值相关联，而在后半部分，桶的大小随着 $i - j$ 的增长而增加。最后一个桶被设计用来处理任意长度的序列。

¹⁵For language modeling, a query is only allowed to attend to its left-context, and so we have $i - j \geq 0$. In the more general case of self-attention, where a token can attend to all tokens in the sequence, we may have negative offsets when $i < j$.

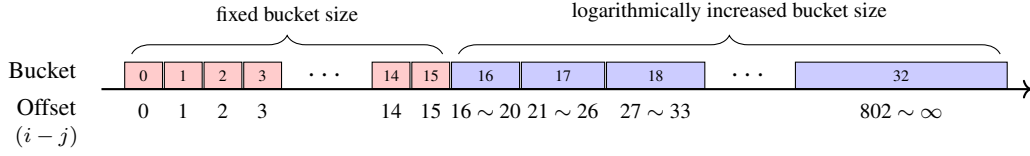


图2.10: T5模型中将查询键偏移分配到桶中的示意图 ($n_b = 32$ 和 $\text{dist}_{\max} = 1024$)。方框代表桶。在桶的前半部分, 我们使用固定的桶大小。在桶的后半部分, 我们以对数方式增加桶大小。最后一个桶包含所有之前桶未覆盖的查询键偏移。

所有桶中的 $\text{PE}(i, j)$ 共享相同的偏差项 $u_{b(i-j)}$ 。将 $\text{PE}(i, j) = u_{b(i-j)}$ 代入公式(2.76), \mathbf{q}_i 和 \mathbf{k}_j 的注意力权重变为¹⁶

$$\alpha(i, j) = \text{Softmax}\left(\frac{\mathbf{q}_i \mathbf{k}_j^T + u_{b(i-j)}}{\sqrt{d}} + \text{Mask}(i, j)\right) \quad (2.81)$$

参数 $\{u_0, \dots, u_{n_b}\}$ 在训练期间被学习为公共参数。应强调, 此模型可以泛化到长序列。这是因为具有相似查询-键偏移的 $\text{PE}(i, j)$ s 共享相同的参数, 并且鉴于在训练中大型查询-键偏移很少, 这种共享策略对于实现良好的泛化特别重要。在实践中, 我们通常将 n_b 设置为适中的数值, 因此它可以帮助控制位置嵌入模型的过拟合。

2.3.5.2 带有非学习偏差的注意力

相对位置嵌入模型基于查询-键乘积的自注意力中的学习偏差。一种替代方法是通过对这些偏差使用启发式方法赋予固定值, 而不是在特定数据集上训练它们。这种基于启发式方法的一个优点是它不依赖于训练过程, 因此一旦设置偏差, 就可以直接应用于任何序列。

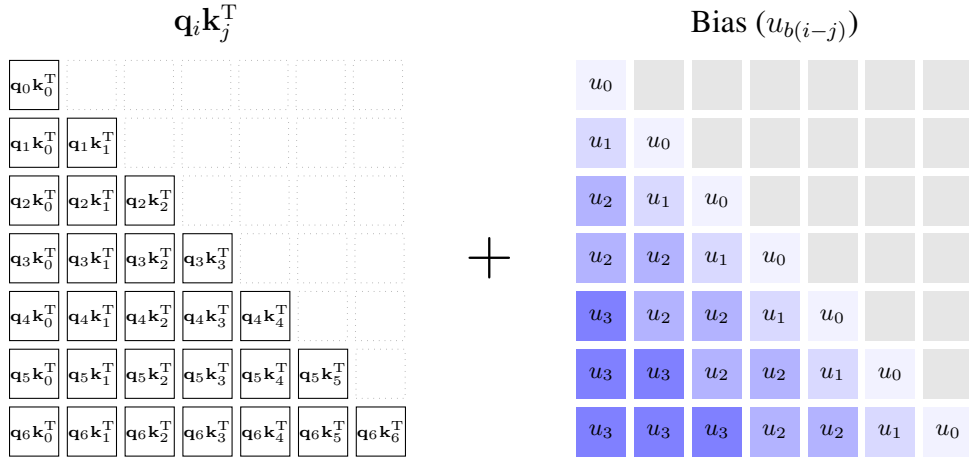
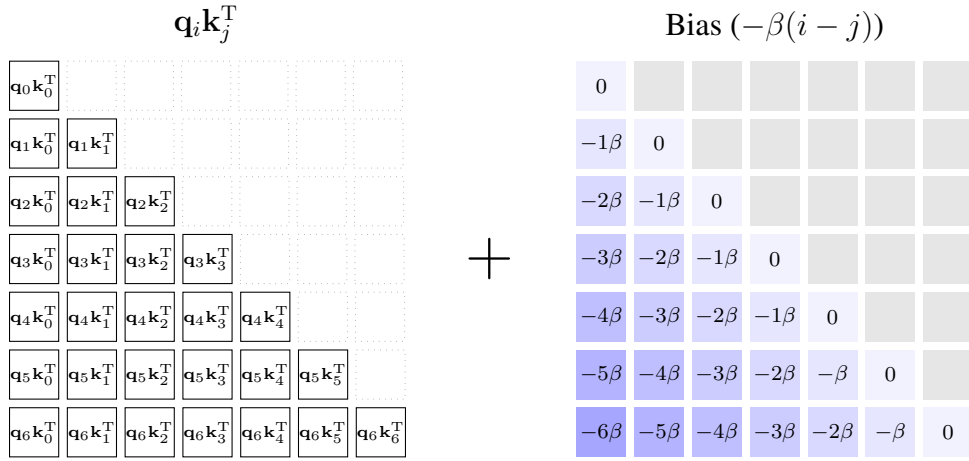
一个这样的方法示例是Press等人[2022]的方法, 称为带有线性偏置的注意力或简称ALiBi。在ALiBi方法中, 偏置项被定义为负的缩放查询-键偏移

$$\begin{aligned} \text{PE}(i, j) &= -\beta \cdot (i - j) \\ &= \beta \cdot (j - i) \end{aligned} \quad (2.82)$$

在 β 是缩放因子的情况下。将此项添加到查询键乘积中, 我们得到新的形式

¹⁶Note that, in Raffel et al. [2020]’s T5 model, the rescaling operation for the query-key product is removed. The attention weight $\alpha(i, j)$ is then given by

$$\alpha(i, j) = \text{Softmax}(\mathbf{q}_i \mathbf{k}_j^T + u_{b(i-j)} + \text{Mask}(i, j)) \quad (2.80)$$

(a) The T5 bias ($n_b = 3$ and $\text{dist}_{\max} = 5$)

(b) The ALiBi bias

图2.11: 具有偏差的查询键乘积（上方为=T5偏差，下方为=的ALiBi偏差）。偏差的颜色范围从表示小绝对值的浅蓝色到表示大绝对值的深蓝色。

注意权重

$$\alpha(i, j) = \text{Softmax}\left(\frac{q_i k_j^T + \beta \cdot (j - i)}{\sqrt{d}} + \text{Mask}(i, j)\right) \quad (2.83)$$

此模型可以解释为在 j 每次向 i 移动一步时，向 $q_i k_j^T$ 添加一个固定惩罚。因此，我们不需要将其适应一系列序列长度，并且可以将其用于模拟任意长度的序列。参见图 2.11 以比较 T5 偏差和 ALiBi 偏差。

通常，标量 β 应在验证数据集上调整。然而，Press 等人 [2022] 发现，将 β 设置为以 $\frac{1}{2^a}$ 为公比的几何递减值，在多头注意力机制的各种任务上表现良好。具体来说，对于涉及 n_{head} 的自注意力子层

Entry	Query-Key Bias ($\text{PE}(i, j)$)
T5 [Raffel et al., 2020]	$u_{b(i-j)}$
ALiBi [Press et al., 2022]	$-\beta \cdot (i - j)$
Kerple [Chi et al., 2022]	$-\beta_1 (i - j)^{\beta_2}$ (power) $-\beta_1 \log(1 + \beta_2 (i - j))$ (logarithmic)
Sandwich [Chi et al., 2023]	$\sum_{k=1}^{\bar{d}/2} \cos((i - j)/10000^{2k/\bar{d}})$
FIRE [Li et al., 2024b]	$f(\psi(i - j)/\psi(\max(m_{\text{len}}, i)))$

表2.4: 查询键偏差作为相对位置嵌入。 β 、 β_1 、 β_2 、 \bar{d} 和 m_{len} 是超参数。在T5模型中, $b(i - j)$ 表示分配给 $i - j$ 的桶。在FIRE模型中, $\psi(\cdot)$ 是一个单调递增的函数, 如 $\psi(x) = \log(cx + 1)$, 而 $f(\cdot)$ 是一个FFN。

头, 第 k -个头的标量由以下公式给出

$$\beta_k = \frac{1}{2^{\frac{8}{k}}} \quad (2.84)$$

ALiBi方法提供了一种简单的相对位置嵌入形式。还有其他类似的方法, 用于使用偏移量 $i - j$ 设计查询-键偏差。表2.4显示了此类偏差的比较。顺便提一下, 方程 (2.82) 右侧的形式与传统基于特征系统的长度特征非常相似。例如, 在统计机器翻译系统中, 这些特征被广泛用于建模词序问题, 从而产生在不同翻译任务中具有良好泛化能力的模型[Koehn, 2010]。

2.3.5.3 旋转位置嵌入

与正弦嵌入类似, 旋转位置嵌入基于嵌入所有维度的硬编码值 [Su 等人, 2024]。回想一下, 在正弦嵌入模型中, 位置被表示为不同频率的正弦和余弦函数的组合。然后, 将这些嵌入添加到标记嵌入中, 形成Transformer层堆栈的输入。旋转位置嵌入相反, 将位置上下文建模为复空间中标记嵌入的旋转。这导致了一种以乘法嵌入形式表达的模式

$$\mathbf{e}_i = \mathbf{x}_i R(i) \quad (2.85)$$

在 $R(i) \in \mathbb{R}^{d \times d}$ 是表示对标记嵌入 $\mathbf{x}_i \in \mathbb{R}^d$ 执行的旋转的旋转矩阵。

为了简单起见, 我们首先考虑只有两个维度的嵌入, 稍后返回到更一般公式的讨论。假设我们有一个二维标记嵌入 $\mathbf{x} = \begin{bmatrix} x_1 & x_2 \end{bmatrix}$ 。我们可以将其表示为平面上的一个向量, 起点在原点 $(0, 0)$ 并终止于 (x_1, x_2) 。这个向量的逆时针旋转是指围绕原点移动向量, 同时保持其大小, 如图2.12 (a) 所示。旋转的程度通常由一个特定的角度定义, 表示为 θ 。旋转可以是

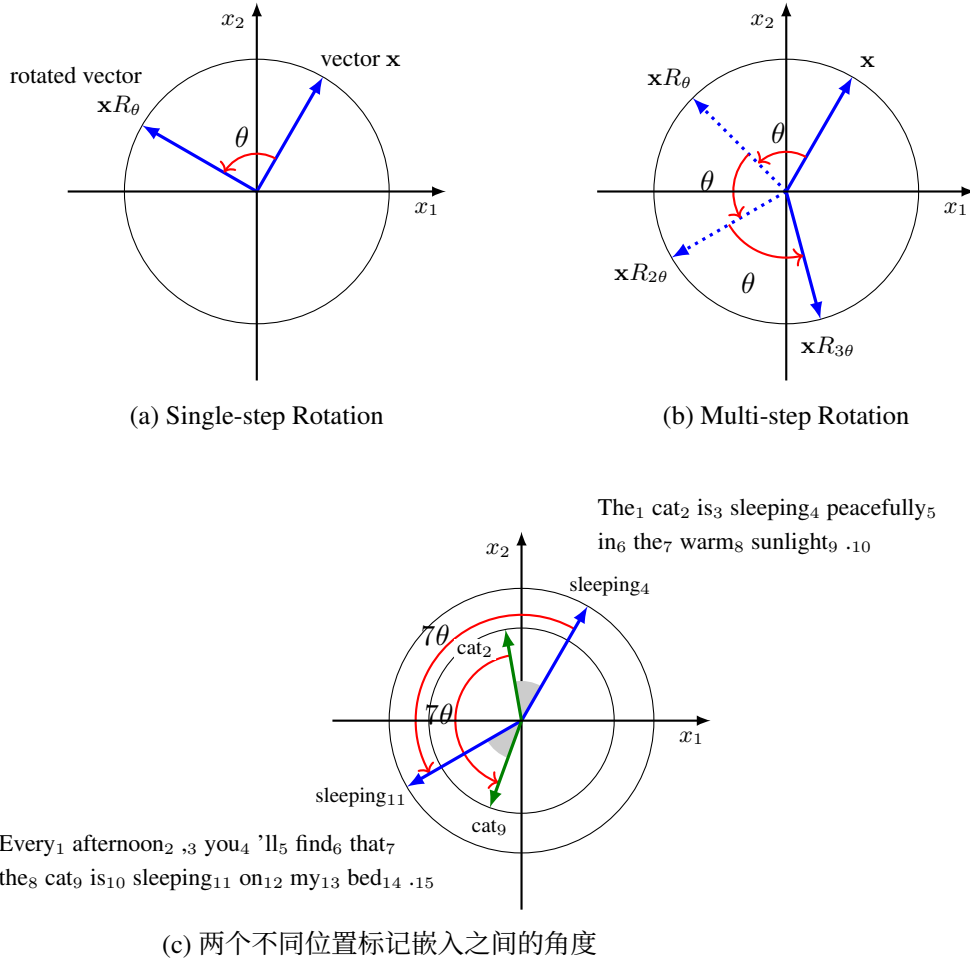


图2.12: 平面中向量旋转的示意图。子图 (a) 和 (b) 分别显示了向量的一次旋转和多次旋转。子图 (c) 显示了标记 cat 和 $sleeping$ 在两个不同句子中的嵌入。我们通过在每个标记上附加下标来显示这些句子，以指示其位置。如果我们把标记表示为向量，我们可以通过旋转这些向量来添加位置信息。这种旋转保留了向量之间的“距离”。例如，给定 cat 和 $sleeping$ 在两个句子中的距离相同，它们嵌入之间的角度在旋转过程中也保持相同。

用数学形式表示

$$\begin{aligned}
 \text{Ro}(\mathbf{x}, \theta) &= \mathbf{x}R_\theta \\
 &= \begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{bmatrix} \\
 &= \begin{bmatrix} \cos \theta \cdot x_1 - \sin \theta \cdot x_2 & \sin \theta \cdot x_1 + \cos \theta \cdot x_2 \end{bmatrix} \quad (2.86)
 \end{aligned}$$

在 $R_\theta = \begin{bmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{bmatrix}$ 是旋转矩阵的情况下。如果在同一向量上执行两个或更多旋转，我们可以进一步旋转该向量。这源于连续旋转的合成本身也是一个旋转的事实。更正式地说，将向量旋转 θ 角度 t 次

可以表示为

$$\begin{aligned}\text{Ro}(\mathbf{x}, t\theta) &= \mathbf{x}R_{t\theta} \\ &= \begin{bmatrix} \cos t\theta \cdot x_1 - \sin t\theta \cdot x_2 & \sin t\theta \cdot x_1 + \cos t\theta \cdot x_2 \end{bmatrix}\end{aligned}\quad (2.87)$$

如果我们把 t 解释为表示为 \mathbf{x} 的标记在序列中的位置，那么我们会发现上述方程定义了一个简单的位置嵌入模型。如图 2.12 (b) 所示，我们从位置 0 开始移动标记。每次我们向前移动一步，向量就旋转 θ 角度。到达位置 t 时，具有位置上下文的标记表示由 $\text{Ro}(\mathbf{x}, t\theta)$ 给出。由于旋转不改变嵌入的幅度，因此保留了标记的原始“意义”。当嵌入旋转时，位置信息被注入其中。

一个理解向量旋转的流行方法是将其定义在复数空间中。通过一个双射线性映射，将二维欧几里得空间中的每个向量 $\mathbf{x} = \begin{bmatrix} x_1 & x_2 \end{bmatrix}$ 转换为复数空间中的 $\mathbf{x}' = x_1 + \mathbf{i}x_2$ 是很容易的。然后， \mathbf{x} 以角度 $t\theta$ 的旋转对应于乘以 $e^{\mathbf{i}t\theta}$ 。鉴于 $e^{\mathbf{i}t\theta} = \cos t\theta + \mathbf{i} \sin t\theta$ ，旋转操作可以重新表示为以下形式

$$\begin{aligned}\mathbf{x}R_{t\theta} &\mapsto \mathbf{x}'e^{\mathbf{i}t\theta} \\ &= (x_1 + \mathbf{i}x_2)(\cos t\theta + \mathbf{i} \sin t\theta) \\ &= \cos t\theta \cdot x_1 - \sin t\theta \cdot x_2 + \mathbf{i}(\sin t\theta \cdot x_1 + \cos t\theta \cdot x_2)\end{aligned}\quad (2.88)$$

这里我们将标记表示 $\mathbf{x}'e^{\mathbf{i}t\theta}$ 记为 $C(\mathbf{x}, t\theta)$ 。位置 t 和 s 的标记表示的内积可以写为

$$\langle C(\mathbf{x}, t\theta), C(\mathbf{y}, s\theta) \rangle = (\mathbf{x}'\overline{\mathbf{y}'})e^{\mathbf{i}(t-s)\theta}\quad (2.89)$$

\mathbf{y}' 是 \mathbf{y}' 的复共轭。如所见，此内积的结果涉及一个项 $t - s$ ，因此它可以模拟两个标记之间的偏移。

现在我们回到二维欧几里得空间中的表示。 $\text{Ro}(\mathbf{x}, t\theta)$ 和 $\text{Ro}(\mathbf{y}, s\theta)$ 的点积可以写成 $(t - s)\theta$ 的函数

$$\begin{aligned}\text{Ro}(\mathbf{x}, t\theta)[\text{Ro}(\mathbf{y}, s\theta)]^T &= \mathbf{x}R_{t\theta}[\mathbf{y}R_{s\theta}]^T \\ &= \mathbf{x}R_{t\theta}[R_{s\theta}]^T\mathbf{y}^T \\ &= \mathbf{x}R_{(t-s)\theta}\mathbf{y}^T\end{aligned}\quad (2.90)$$

给定这个结果，如果我们将 $\text{Ro}(\mathbf{x}, t\theta)$ 和 $\text{Ro}(\mathbf{y}, s\theta)$ 作为查询和键，那么自注意力操作将隐式地涉及相对位置上下文的建模。

这个旋转位置嵌入可以扩展到多维嵌入。对于 d -维度的标记嵌入 $\mathbf{x} = \begin{bmatrix} x_1 & x_2 & \dots & x_d \end{bmatrix}$ ，我们可以将其视为 $\frac{d}{2}$ -维度的复向量 $\mathbf{x}' = \begin{bmatrix} x_1' & x_2' & \dots & x_{d/2}' \end{bmatrix} = \begin{bmatrix} x_1 + \mathbf{i}x_2 & x_3 + \mathbf{i}x_4 & \dots & x_{d-1} + \mathbf{i}x_d \end{bmatrix}$ ，其中每个连续的项对形成一个复数。然后，旋转位置嵌入在

复空间由以下给出

$$C(\mathbf{x}, t\theta) = \sum_{k=1}^{d/2} x'_k e^{it\theta_k} \vec{e}_k \quad (2.91)$$

\vec{e}_k 是在 k -th 坐标上具有单个非零值而在其他地方为 0 的标准基向量 [Biderman 等人, 2021]。

尽管这个公式涉及一个复杂的表达式，但在 d -维欧几里得空间中的等价形式相对容易理解。我们可以将其写为

$$\text{Ro}(\mathbf{x}, t\theta) = \begin{bmatrix} x_1 & x_2 & \dots & x_d \end{bmatrix} \begin{bmatrix} R_{t\theta_1} & & & \\ & R_{t\theta_2} & & \\ & & \ddots & \\ & & & R_{t\theta_{d/2}} \end{bmatrix} \quad (2.92)$$

在 $R_{t\theta_k} = \begin{bmatrix} \cos t\theta_k & \sin t\theta_k \\ -\sin t\theta_k & \cos t\theta_k \end{bmatrix}$ 。 $\theta = [\theta_1, \dots, \theta_{d/2}]$ 是控制不同维度旋转角度的参数。通常， θ_k 设置为 $10000^{-\frac{2(k-1)}{d}}$ ，这与正弦嵌入中的设置类似。

在实用实现中，式 (2.92) 可以改写为一个仅依赖于向量逐元素乘积和加法的形式。

$$\text{Ro}(\mathbf{x}, t\theta) = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{d-1} \\ x_d \end{bmatrix}^T \odot \begin{bmatrix} \cos t\theta_1 \\ \cos t\theta_1 \\ \vdots \\ \cos t\theta_{d/2} \\ \cos t\theta_{d/2} \end{bmatrix}^T + \begin{bmatrix} -x_2 \\ x_1 \\ \vdots \\ -x_d \\ x_{d-1} \end{bmatrix}^T \odot \begin{bmatrix} \sin t\theta_1 \\ \sin t\theta_1 \\ \vdots \\ \sin t\theta_{d/2} \\ \sin t\theta_{d/2} \end{bmatrix}^T \quad (2.93)$$

最后，我们将公式 (2.85) 重写以获得位置 i 的嵌入形式

$$\mathbf{e}_i = \text{Ro}(\mathbf{x}_i, i\theta) \quad (2.94)$$

2.3.5.4 位置插值

在位置插值中，我们的目标是将新序列中的位置映射到与训练中观察到的范围相匹配。假设训练序列长度从 0 到 m_l 。当测试时 $m > m_l$ ，我们表示 $[0, m]$ 中的位置，以便我们的表示适合 $[0, m_l]$ 。

为了说明，考虑上述描述的旋转位置嵌入模型。每个标记的嵌入由一个模型 $\text{Ro}(\mathbf{x}\{\mathbf{v}^*21\})$ 描述，其中 $\{\mathbf{v}^*22\}$ 是参数。 $\text{Ro}(\mathbf{x}\{\mathbf{v}^*23\})$ 可以表示为两个周期函数的线性组合（见公式）。

(2.93))

$$\cos i\theta = \begin{bmatrix} \cos i\theta_1 & \dots & \cos i\theta_{d/2} \end{bmatrix} \quad (2.95)$$

$$\sin i\theta = \begin{bmatrix} \sin i\theta_1 & \dots & \sin i\theta_{d/2} \end{bmatrix} \quad (2.96)$$

θ_k 是 k 的指数函数, 形式为

$$\theta_k = b^{-\frac{2(k-1)}{d}} \quad (2.97)$$

b 是基数。 $\cos i\theta_k$ 和 $\sin i\theta_k$ 的周期是

$$T_k = 2\pi \cdot b^{\frac{2(k-1)}{d}} \quad (2.98)$$

位置插值背后的关键思想是调整这个周期, 以便新的位置可以编码在范围 $[0, m_l]$ 内。实现这一目标的一种方法是将 T_k 按 $\frac{m}{m_l}$ 缩放, 给出

$$T'_k = \frac{m}{m_l} \cdot 2\pi \cdot b^{\frac{2(k-1)}{d}} \quad (2.99)$$

因此, $[0, m]$ 中的所有点都压缩到了 $[0, m_l]$ 。这种线性缩放可以通过修改嵌入模型的输入[Chen等人, 2023c]轻松实现。具有线性位置插值的新的模型如下所示

$$\text{Ro}'(\mathbf{x}_i, i\theta) = \text{Ro}(\mathbf{x}_i, \frac{m_l}{m} i\theta) \quad (2.100)$$

另一种位置插值的方法是对基 $\{v^*\}$ 进行缩放。假设基 b 被缩放为 λ 。我们希望这个新模型在 θ (的最后维度 (即维度 $\frac{d}{2}$)) 的周期与线性位置插值模型的周期相等。这可以表示为

$$2\pi \cdot (\lambda b)^{\frac{2(\frac{d}{2}-1)}{d}} = \frac{m}{m_l} \cdot 2\pi \cdot b^{\frac{2(\frac{d}{2}-1)}{d}} \quad (2.101)$$

解这个方程, 我们得到

$$\begin{aligned} \lambda &= \left(\frac{m}{m_l} \right)^{\frac{d}{2(\frac{d}{2}-1)}} \\ &= \left(\frac{m}{m_l} \right)^{\frac{d}{d-2}} \end{aligned} \quad (2.102)$$

这给出一个嵌入模型

$$\text{Ro}'(\mathbf{x}_i, i\theta) = \text{Ro}(\mathbf{x}_i, i\theta') \quad (2.103)$$

¹⁷This method was first proposed in https://www.reddit.com/r/LocalLLaMA/comments/141z7j5/ntkaware_scaled_rope_allows_llama_models_to_have/

哪里

$$\theta' = \left[(\lambda b)^{-\frac{0}{d}}, (\lambda b)^{-\frac{2}{d}}, \dots, (\lambda b)^{-\frac{d-2}{d}} \right] \quad (2.104)$$

请注意，对基进行缩放提供了一种非均匀的方法来缩放不同维度上的周期 θ 。这种方法已被发现有助于将LLMs扩展到更长的序列，并且已经开发出几种改进方法 [Peng等人, 2024; Ding等人, 2024]。

2.3.6 注释

本节中，我们介绍了多种长上下文语言建模方法。我们通过讨论与这些方法相关的一些有趣问题来结束本节。

2.3.6.1 长上下文需求

一个长上下文LLM的最终目标之一是这些模型可以精确编码无限上下文。所谓的无限上下文更多地指的是LLM可以持续读取单词。这激励了可以处理极长上下文或流数据的LLM。如第2.3.3节所述，使用固定大小的内存模型来处理持续扩展的上下文是常见的。许多这样的系统基于循环架构或其变体，因为它们本质上适合于模拟过去输入效应无限持续的时间序列问题。实现无限内存的另一种方法是开发自我注意力模型的替代方案，例如，可以使用连续空间注意力模型来编码上下文，从而消除对上下文长度的依赖 [Martins等人, 2022]。

当研究长上下文LLM时，自然会想知道哪些机制可以解释在语言模型中使用长上下文。我们能否将无限上下文的表示压缩成一个相对较小的模型？所有上下文标记对预测下一个标记都有用吗？当LLM看到上下文时，它们是如何为标记预测做准备的？我们能否提前知道哪些上下文信息对预测至关重要？对所有这些问题的普遍答案并不明显，但它们激发了可解释模型后续研究，并且已经发现了一些有趣的结果。例如，Deletang等人 [2024] 进行了广泛的实验，表明LLM是强大的上下文压缩器。尽管将预测模型视为压缩模型在机器学习中已经研究了很长时间，但它也为我们理解LLM扩展定律提供了见解。Pal等人 [2023] 和Wu等人 [2024] 研究了直到当前步骤学到的特征，尽管不是有意为之，是否已经足够用于预测后续步骤的标记。请注意，在语言模型中需要长上下文高度依赖于我们解决的问题。相关问题是LLM的应用和评估。例如，在摘要任务中，我们可能只需要提炼和关注文本的几个关键方面，而在类似检索的任务中，我们需要“记住”整个上下文，以便可以访问相关信息。我们将在本小节稍后讨论评估问题。

2.3.6.2 预训练或调整LLMs？

训练LLMs需要巨大的计算成本。尽管在长序列数据上训练LLMs很简单，但对于大数据集来说，训练在计算上变得难以控制。

在通用数据集上预训练LLMs是一种常见做法，然后通过适度的微调努力进行适配。例如，具有相对或旋转位置嵌入的LLMs可以直接在预训练阶段对大规模数据进行训练。虽然生成的模型可能在推理阶段表现出一些推断长度的能力，但更有效的方法是在更长的序列上对它们进行微调。

理想情况下，我们希望使用标准的Transformer架构预训练LLMs，并将它们适应到新任务中。这使我们能够使用许多现成的LLMs，并有效地将它们适应以处理长序列。然而，当采用新的架构时，似乎不可避免地需要从头开始训练这些模型。这为开发长上下文LLMs带来了实际困难，因为我们不能利用已经开发好的、预训练的模型，而必须自己训练它们。另一方面，微调仍然是适应具有与预训练不同的架构的LLMs的有效方法。一个例子是具有外部记忆的模型。在这些模型中，预训练的LLMs是固定的，重点是这些LLMs如何与记忆模型协作。例如，在RAG中，通常微调LLMs以改进它们对检索增强输入的使用。另一个为长上下文建模微调LLMs的例子是，我们在微调阶段用稀疏注意力模型替换了具有全注意力模型的LLM。预训练的LLM提供了用于不同模型中模型参数的初始值，然后该模型按常规进行微调。

2.3.6.3 评估长上下文LLMs

评估长上下文LLM非常重要，但这是NLP中的一个新问题。一般思路是，如果我们向LLM输入一个长上下文，然后我们可以从LLM的输出中检查它是否理解整个上下文并在预测后续标记时利用它。在传统的NLP研究中，此类评估通常旨在检验NLP模型处理长距离依赖的能力。然而，最近LLM中使用的上下文大小远大于几年前NLP系统中使用的上下文大小。这促使研究人员为长上下文LLM开发新的评估基准和指标。

一种方法是使用困惑度指标。然而，尽管这种方法表面上很简单，但它往往更多地反映了LLMs利用局部上下文的能力，而不是全局上下文的能力。因此，开发针对长上下文LLMs的特定评估方法很有吸引力。流行的方法包括各种合成任务，在这些任务中使用人工生成或修改的数据来评估长上下文LLMs的具体能力。例如，在针尖上的针¹⁸和密钥检索任务[Mohtashami和Jaggi, 2024; Chen等人, 2023c]中，要求LLMs从大量给定文本中识别和提取一小块相关信息。这里的假设是，具有足够记忆的LLM在处理新信息时应记住文本的早期部分。因此，这种LLM可以从文本中挑选出相关的细节，这些细节可能稀疏且隐藏在大量无关信息中。或者，在复制记忆任务（或简称为复制任务）中，使用LLMs重复输入文本或特定段落的多次。这些任务最初是为了测试循环模型能够保留和回忆先前看到的标记的程度[Hochreiter和Schmidhuber, 1997; Arjovsky等人, 2016]，并且已被用于评估最近的LLMs[Bulatov等人, 2022; Gu和Dao, 2023]。

¹⁸https://github.com/gkamradt/LLMTest_NeedleInAHaystack

另一种评估长上下文LLMs的方法是在涉及非常长的输入序列的NLP任务上测试它们。例如包括长文档或多个文档摘要、长文档问答、代码补全等。这种方法的一个好处是它可以与用户期望相一致。

尽管已经开发了许多方法，但仍然没有一种通用的方法来评估长上下文LLMs [刘等, 2024c]。一个问题在于，这些方法大多关注LLMs的特定方面，而不是它们建模非常长上下文的基本能力。即使一个LLM可以从输入中挑选出适当的文本片段，我们也不能说它真正理解了整个上下文。相反，它可能只是记住了上下文的一些重要部分，或者简单地通过在预训练中学习的模型回忆答案。此外，许多任务中使用的数据规模较小且相对初步，导致评估结果与实际应用性能之间存在差异。更有趣的问题是，LLMs的结果受许多其他因素和实验设置的影响，例如，使用不同的提示可以导致非常不同的结果。这使得评估变得更加具有挑战性，因为改进可能不仅仅来自对长上下文的更好建模，而且存在过度夸大我们结果的风险。尽管如此，在长上下文LLMs的开发和评估中仍有许多未解之谜。例如，这些模型仍然受到诸如上下文长度受限和高延迟等限制。研究这些问题可能有助于证明未来的有价值方向。

2.4 摘要

本章中，我们讨论了LLMs及其相关技术。这可以被认为是LLMs的通用介绍，尽管不是全面的，为后续章节中更高级主题的讨论奠定了基础。此外，我们还探讨了两种扩展LLMs的方法。第一种侧重于LLMs的大规模预训练，这对于开发最先进的模型至关重要。第二种侧重于将LLMs适应于长输入的方法，包括优化注意力模型、设计更高效和压缩的KV缓存、整合记忆模型以及探索更好的位置嵌入。

LLMs的强大之处在于它们能够突破为有限数量的特定任务训练NLP模型的限制。相反，LLMs通过简单的标记预测任务从大量文本中学习——我们预测句子中给定其先前标记的下一个标记。一种普遍的观点是，通过重复进行大量的标记预测任务，LLMs可以获取一些关于世界和语言的知识，然后将这些知识应用于新任务。因此，LLMs可以通过将其构造成给定提示的后续标记预测任务来执行任何任务。这种语言模型中出现的功能来自多个维度，例如扩大训练规模、模型大小和上下文大小。不可否认的是，目前开发大型语言模型的基本原则是扩展定律，尽管仅仅增加模型大小尚未证明足以实现AGI。这些持续扩展的LLMs被发现具有通用语言理解、生成和推理的能力。最近，人们发现，在推理时扩大计算能力也可以在复杂推理任务中带来显著的改进[OpenAI, 2024]。

鉴于其惊人的能力，LLMs已引起相当大的兴趣，无论是从技术还是应用角度来看。因此，对LLMs的研究兴趣激增也导致了一个

大量新技术和模型。然而，鉴于该领域的快速发展，我们并不试图提供关于LLMs所有方面的全面文献综述。尽管如此，人们仍然可以从一般性综述[赵等，2023；米纳尼等，2024]或更专注于特定主题的讨论[阮等，2024]中了解LLMs。

CHAPTER 3

提示

在LLM的上下文中，*prompting*指的是向LLM提供特定输入或提示以生成所需输出或执行任务的方法。例如，如果我们想让LLM将一句话从英语翻译成中文，我们可以这样提示它

将文本从英语翻译成中文。

早起的鸟儿有虫吃。

翻译： _____

提示对于LLM至关重要，因为它直接影响这些模型理解和响应用户查询的有效性。精心设计的提示可以引导LLM生成更准确、相关和上下文适当的响应。此外，这个过程可以迭代优化。通过分析LLM的响应，用户可以调整他们的提示，使其更紧密地符合他们的特定需求。鉴于提示在应用LLM中的重要性，提示设计已成为使用LLM的用户和开发者的必备技能。这导致了一个活跃的研究领域，称为提示工程，其中我们设计有效的提示，以更好地利用LLM并增强其在实际应用中的实用性。

与提示相关的一个重要概念是在上下文中学习。当提示一个LLM时，我们可以向上下文中添加新信息，例如解决问题的演示。这允许LLM从上下文中学习如何解决问题。以下是一个通过几个演示如何根据情感极性对文本进行分类来提示LLMs的例子。

这里有一些文本分类的例子。示例 1：我们一起度过了一个愉快的晚餐。→ 标签：正面 示例 2：我对延误感到沮丧。→ 标签：负面 “那个评论相当伤害人。” 的标签是什么？ 标签：

上下文学习通常被视为LLM在预训练后出现的一种涌现能力。尽管LLM可以被训练或调整以执行新任务，但上下文学习提供了一种无需任何训练或调整努力的高效方式来适应这些模型。也许这是LLM最显著的特征之一：它们确实在预训练期间学习了关于世界和语言的一般知识，我们可以轻松地将这些知识应用于新的挑战。此外，上下文学习反映了使AI系统更具通用性和用户友好的更广泛趋势。不再需要专门的工程师为每个独特任务微调模型，用户可以以更直观的方式与LLM互动，只需提供示例或根据需要调整上下文即可。

在这一章中，我们关注LLMs中的提示技术。我们首先考虑了几种在提示工程中常用的有趣提示设计。然后，我们讨论了一系列

对thes的改进 e 方法。最后，我们探索自动化的方法。 ng 提示设计。

3.1 通用提示设计

本节介绍了提示设计的基本概念，以及如何为各种NLP任务提示LLMs的示例。由于提示的有效性高度依赖于所使用的LLMs，因此提示在不同LLMs之间往往有所不同，这使得为所有LLMs和下游任务提供全面的提示列表变得困难。因此，本讨论不针对任何特定的LLM。相反，目标是提供提示设计的指导原则。

3.1.1 基础

*prompt*这个术语被用于多种不同的方式。在本章中，我们将提示定义为LLM的输入文本，用 x 表示。LLM通过最大化概率 $\Pr(y|x)$ 生成文本 y 。在这个生成过程中，提示充当我们进行预测的条件，它可以包含任何有助于描述和解决问题的信息。

提示可以通过提示模板（或简称模板）[刘等，2023a]获得。模板是一段包含占位符或变量的文本，其中每个占位符都可以用特定信息填充。以下是两个用于向LLM请求周末建议的模板。

请给我一些建议，让我这个周末过得愉快。

——

如果{*前提*}，你有什么建议来度过一个有趣的周末。

——

在第一个模板中，我们只是指示LLM返回一些建议。因此，模板只是一段没有变量的文本。在第二个模板中，变量{*前提*}需要由用户指定，以提供做出建议的前提。例如，如果我们输入

premise = the weather is nice this weekend

然后我们可以生成一个提示

如果这个周末天气好，你有什么建议来度过一个愉快的周末吗？

——

我们也可以设计一个包含多个变量的模板。以下是一个示例，其中我们比较了两句话在语义相似性方面的差异。

这里是一个句子 {**sentence1**} 这里是另一个句子 {**sentence2**} 计算两个句子之间的语义相似度

一种流行的提示格式方法是按“名称：内容”风格编写每个输入或输出。例如，我们可以描述两个名叫约翰和大卫的人之间的对话，并使用LLM继续对话。此类提示的模板如下：

约翰： {**utterance1**}
大卫： {**utterance2**}
约翰： {**utterance3**}
大卫： {**utterance4**}
约翰： {**utterance5**}
大卫： {**utterance6**}
约翰： {**utterance7**}
大卫： _____

“名称：内容”格式可以用来定义我们希望LLM执行的任务。例如，给定“Q”和“A”分别是“问题”和“答案”的常用缩写，我们可以使用以下模板进行问答。

Q: {**问题**} A:

此格式可用于描述更复杂的任务。例如，以下是一个为翻译任务提供规范的示例

任务：翻译 源语言：英语 目标语言：中文 风格：正式文本 模板：翻译以下句子： {**sentence**}

在实用系统中，通常以键值对的形式表示和存储此类数据，例如JSON格式¹。

当问题难以用属性方式描述时，更常见的是用清晰详细的方式来指导LLMs。有许多方法可以做到这一点。一种

¹The JSON representation is

示例是将角色分配给LLMs并提供足够的上下文。以下是一个指导LLM扮演专家并回答儿童问题的模板。

你是一位对深度学习领域有广泛知识的计算机科学家。请尽可能使用简单例子，向一个大约10岁的孩子解释以下与计算机相关的概念。

{*概念*}

这里文本“您是一位计算机科学家...深度学习。”有时被称为系统信息，并提供以帮助LLM理解其被要求执行的任务的上下文或约束。

3.1.2 在上下文中学习

学习可以在推理过程中发生。情境学习是一种方法，其中提示涉及解决问题的演示，LLMs 可以从这些演示中学习如何解决新问题。由于我们在此过程中不更新模型参数，情境学习可以被视为一种在无需额外训练或微调的情况下，高效激活和重组预训练中学习到的知识的方法。这使得 LLMs 能够快速适应新问题，推动预训练 LLMs 在无需特定任务调整的情况下所能达到的边界。

上下文学习可以通过比较三种方法来阐述：零样本学习、单样本学习和少样本学习。零样本学习，正如其名所示，不涉及传统的“学习”过程。它相反地直接应用LLMs来解决训练期间未观察到的新的问题。在实践中，我们可以反复调整提示来引导LLMs生成更好的响应，而不展示解决问题的步骤或提供示例。考虑以下示例。假设我们想使用一个LLM作为助手，帮助纠正英语句子。一个零样本学习提示如下

```
{
  "Task": "Translation"
  "Source language": "English"
  "Target language": "Chinese"
  "Style": "Formal text"
  "Template": "Translate the following sentence: {*sentence*}"
}
```

SYSTEM You are a helpful assistant, and are great at grammar correction.

USER You will be provided with a sentence in English. The task is to output the correct sentence.

Input: She don't like going to the park.

Output: _____

这里用灰色字表示提示的不同字段。

在一次性学习中，我们通过添加如何纠正句子的演示来扩展这个提示，从而允许LLM从新添加的经验中学习。

系统：你是一个有用的助手，擅长语法纠正。演示：你将得到一个英文句子。任务是输出正确的句子。输入：There is many reasons to celebrate. 输出：There are many reasons to celebrate. 用户：你将得到一个英文句子。任务是输出正确的句子。输入：She don't like going to the park. 输出：

此外，我们可以添加更多演示以实现少样本学习运行。

She doesn't like going to the park.

在少样本学习中，我们本质上提供了一种模式，将一些输入映射到相应的输出。LLM试图在做出预测时遵循这种模式，前提是提示中包含足够多的演示，尽管通常数量较小。它也是

可以使用更简单的模式来实现这一点。例如，可以使用以下几个提示进行从中文到英文的单词翻译。

DEMO 现在 → 现在 来 →
来 去 → 去 男孩 → 男孩
USER 女孩 → ____

如果LLM足够强大，少样本学习可以使其能够解决复杂问题，例如数学推理。例如，考虑以下任务：求两个数的和，然后将和除以它们的乘积。

DEMO 12 5 → $(12 + 5)/(12 \times 5) = 0.283$
3 1 → $(3 + 1)/(3 \times 1) = 1.33$
-9 4 → $(-9 + 4)/(-9 \times 4) = 0.138$
15 15 → $(15 + 15)/(15 \times 15) = 0.133$
USER 19 73 → ____

在许多实际应用中，情境学习的有效性在很大程度上依赖于提示的质量和预训练LLM的基本能力。一方面，我们需要付出大量的提示工程努力，以开发出有助于LLM从演示中更有效地学习的适当提示。另一方面，更强的LLM可以更好地利用情境学习来执行新任务。例如，假设我们希望使用一个LLM将因纽特语翻译成英语。如果LLM缺乏对因纽特语数据的预训练，其对因纽特语的理解将很弱，无论我们如何提示，模型在翻译中的表现都难以良好。在这种情况下，我们需要继续用更多的因纽特语数据训练LLM，而不是试图找到更好的提示。

可能很有趣的是探索在预训练期间上下文学习如何出现以及为什么它在推理期间有效。一种简单的理解是，LLMs 获得了一些解决问题的知识，但存在许多可能的预测，当模型面对新问题时，这些预测很难区分。提供演示可以引导 LLMs 跟随“正确”的路径。此外，一些研究人员尝试从几个不同的角度来解释上下文学习，包括贝叶斯推理 [Xie 等人, 2022]，梯度下降 [Dai 等人, 2023; Von Oswald 等人, 2023]，线性回归 [Akyürek 等人, 2023]，元学习 [Garg 等人, 2022] 等。

3.1.3 提示工程策略

设计提示非常经验主义。一般来说，有多种方法可以提示LLM执行相同任务，我们需要进行多次试错运行以找到令人满意的提示。为了更有效地编写好的提示，可以遵循某些策略。常见的提示原则示例包括

- 尽可能清晰地描述任务。当我们应用一个LLM来解决一个问题时，我们需要提供精确、具体和清晰的问题描述，并指导LLM按照我们的期望执行。当我们希望LLM的输出满足某些期望时，这一点尤为重要。例如，假设我们对于气候变化很感兴趣。一个简单的提示来要求LLM提供一些信息是

告诉我关于气候变化的事情。

由于这个指令过于笼统，LLM可能会生成一个涉及气候变化任何方面的响应，这可能不符合我们的特定兴趣。在这种情况下，我们可以使用具体且详细的提示。一个这样的例子是

提供对气候变化原因和影响的详细解释，包括对全球温度、天气模式和海平面的影响。此外，讨论可能的解决方案和正在采取的行动以减轻这些影响。

现在假设我们打算向一个10岁的孩子解释气候变化。我们可以进一步调整上述提示。

解释气候变化的原因和影响给一个10岁的孩子听。谈谈它如何影响天气、海平面和温度。还要提到人们正在做的一些帮助事情。尽量用简单的话来说明，不要超过500字。

- 引导LLM进行思考。LLM展现出惊人的“思考”能力。一个常见的例子是，经过良好发展的LLM在数学推理任务中取得了令人印象深刻的性能，这些任务被认为是具有挑战性的。在提示工程中，LLM的“思考”能力需要通过适当的提示来激活，特别是对于需要大量推理努力的问题。在许多情况下，被指示“思考”的LLM与被指示直接执行任务的相同LLM相比，可以产生完全不同的结果。例如，Kojima等人[2022]发现，只需在每个提示的末尾简单添加“让我们一步步思考”，就可以提高LLM在多个推理任务上的性能。LLM可以通过多种方式被提示“思考”。一种方法是指示LLM在得出最终答案之前为解决问题生成推理步骤。例如，考虑解决数学问题的任务。下面是针对此任务的简单提示。

您是一位数学家。您将得到一个数学问题。请解决问题。

由于解决数学问题需要详细的推理过程，如果LLMs试图直接计算出答案，它们可能会犯错误。因此，我们可以在LLMs得出结论之前，明确要求它们遵循给定的推理过程。

您是一位数学家。在解决数学问题时，您将遵循以下详细的推理步骤。

第一步：问题解释。

数学家仔细聆听您的查询，并理解您所提出的数学挑战的复杂细节。

步骤 2：策略制定。

从他们丰富的知识中汲取，数学家选择最适合数学问题类型的有效策略，无论是代数、微积分还是几何。

步骤 3：详细计算。

以精确和专业，数学家逐步进行必要的计算，遵循所有数学原理。

步骤 4：解决方案审查。

在提供最终答案之前，数学家仔细检查计算的正确性，并为解决方案提供简洁的解释或理由。

您将得到一个数学问题。请解决问题。

{*问题*}

另一种引导LLMs“思考”的方法是通过与LLMs的多轮交互。例如，作为第一步，我们可以指示LLMs直接解决问题

您将得到一个数学问题。请解决问题。

{*问题*}

现在我们对这个问题有一个初步答案。作为第二步，我们提示LLMs评估答案的正确性，并在必要时重新修改以找到更好的解决方案。

您将得到一个数学问题及其解决方案。评估该解决方案的正确性，并识别任何存在的错误。然后，自己解决该问题。

问题：{*问题*}

解决方案：{*solution*}

这里提出的提示与LLMs中推理问题的长期研究密切相关。由于这个主题涵盖了大量方法，因此无法对所有相关问题进行完整讨论。但在第3.2节中，我们将更详细地讨论如何通过更多推理来改进提示。

- 提供参考信息。如前文所述，我们可以在提示中包含演示，并允许LLM在上下文中从这些演示中学习如何执行任务。事实上，鉴于LLM在语言理解方面的卓越能力，我们可以将任何类型的文本添加到提示中，因此这些模型可以根据丰富的上下文进行预测。在许多应用中，我们拥有与用户查询相关的各种信息。我们通常不希望LLM进行无约束的预测，而是希望LLM产生仅限于相关文本的输出。一个这样的例子是RAG，其中用户查询的相关文本是通过调用IR系统提供的，我们提示LLM根据提供的这些相关文本生成响应。以下提示显示了示例。

您是一位能够针对输入查询生成答案的专家。现在您已经得到了一个查询及其对应的环境信息。请根据这些环境信息生成一个答案。请注意，您需要用自己的话来提供答案，而不仅仅是复制提供的环境信息。{*v*}

上下文信息：{*IR结果*}

查询：{*查询*}

If the context information is highly reliable, we can even restrict LLMs to answering using only the provided text. An example prompt is shown as follows

您是一位专家，负责从输入查询中生成答案。您已经收到一个查询和相应的上下文信息，这些信息以表格形式组织，其中每一行代表一条有用的记录。请仅使用这些上下文信息生成答案。请确保您用自己的话来提供答案。

上下文信息：{*表*}

查询：{*查询*}

——

处理现实世界问题时，我们通常对问题有先验知识和额外信息，这些信息有助于产生更好的答案。在提示中考虑此类信息通常有助于提高结果。

- 关注提示格式。一般来说，LLMs 的性能对我们输入的提示非常敏感。有时对提示的微小修改可能导致模型输出的巨大变化。一个有趣的例子是，改变提示中句子的顺序可能会导致 LLMs 生成不同的结果。为了使提示易于阅读并减少歧义，通常以确保清晰的方式对它们进行格式化。一个例子是我们为提示定义了几个字段，并在每个字段中填写不同的信息。另一个例子是我们可以使用 LLMs 可以理解和生成自然语言和代码的代码风格提示。以下是一个执行翻译的代码风格提示示例，其中提供了一个演示。

[英语] = [我有一个苹果。] [德语]
= [Ich habe einen Apfel.] [英语] =
[我有一个橙子。] [德语] = ____

LLMs 可以接收各种格式的文本。这使我们能够使用控制字符、XML 标签和特定格式来表示复杂数据。并且指定输入和输出的格式或结构是有用的。例如，我们可以使用引号来分隔文本部分，并相应地提示 LLM（例如，在提示中添加“输入文本由双引号分隔”这样的句子）。

以上，我们仅讨论了编写良好提示的几种策略。当然，有许多这样的方法，并且需要通过实践来发展自己的方法。感兴趣的读者可以参考各种在线文档以获取更多信息，例如 OpenAI 关于 GPT 系列模型的指南²。

²See <https://platform.openai.com/docs/guides/prompt-engineering/six-strategies-for-getting-better-results>.

3.1.4 更多示例

在这个子节中，我们考虑了更多示例，以提示LLMs执行各种NLP任务。这里的动机不是提供这些任务的标准化提示，而是使用简单的示例来说明如何提示LLMs处理NLP问题。

3.1.4.1 文本分类

文本分类可能是NLP中最常见的问题之一。许多任务可以广泛地分为给定的文本分配预定义标签。在这里，我们考虑情感分析中的极性分类问题。我们选择极性分类进行说明，因为它是最受欢迎和定义最明确的文本分类任务之一。在极性分类的一般设置中，我们需要将给定的文本分类到三个类别之一：负面、正面或中性。下面是一个简单的提示来完成这项任务（为了便于阅读，我们在提示中突出显示了任务描述）。

分析以下文本的极性并将其分类为正面、负面或中性。

文本：{v*}

餐厅的服务比预期的慢，有点令人沮丧。

文本的极性可以归类为负面。

为了使示例完整，我们展示了LLM生成的响应（下划线文本）。

尽管答案正确，但LLM给出这个答案不是通过标签，而是通过描述结果的文本。问题是LLM被设计用来生成文本，而不是为文本分配标签并将分类问题视为文本生成问题。因此，我们需要另一个系统将LLM的输出映射到标签空间（称为标签映射），即我们从“文本的极性可以分类为负面”中提取“负面”。在大多数情况下，这是微不足道的，因为我们可以通过简单的启发式方法识别标签词。但有时，LLM可能不会使用这些标签词来表达分类结果。在这种情况下，问题变得更加复杂，因为我们需要某种方法将生成的文本或单词映射到预定义的标签词。

一种从LLMs中诱导输出标签的方法是将问题重新构造成一个填空任务。例如，以下展示了一个类似于填空的任务提示，用于极性分类。

分析以下文本的极性并将其分类为正面、负面或中性。

文本：{v*}

餐厅的服务比预期的慢，有点令人沮丧。

文本的极性为负

我们可以使用LLMs来完成文本并填充空白，使其包含最合适的词语。理想情况下，我们希望填充的词语是积极的、消极的或中性的。然而，LLMs并不能保证生成这些标签词语。解决这个问题方法之一是将预测限制在标签词语的集合中，并选择概率最高的一个。然后，输出标签为

$$\text{label} = \arg \max_{y \in Y} \Pr(y|\mathbf{x}) \quad (3.1)$$

y 表示填入空白的单词， Y 表示标签词集合{积极,负面,中性}。

另一种使用LLMs生成标签的方法是通过提示来约束输出。例如，我们可以提示LLMs在受控的词集中进行预测。以下是一个示例。

分析以下文本的极性并将其分类为正面、负面或中性。

文本: {v*}

餐厅的服务比预期的慢，有点令人沮丧。

文本的极性是什么？

仅回答：正面、负面或中性。

负_____

情感分析是一个常见的NLP问题，LLMs可能通过预训练或微调已经很好地理解了。因此，我们可以通过简单的指令来提示LLMs执行任务。然而，对于新的分类问题，可能需要提供有关任务的额外详细信息，例如分类标准，以便LLMs能够正确执行。为此，我们可以在提示中添加更详细的任务描述和/或展示分类示例。为了说明，考虑以下示例。

分析以下文本的极性并将其分类为积极、消极或中性。以下是每个类别的含义：积极：这表示文本传达了积极的情绪或态度。例如，表达快乐、满意、兴奋或钦佩的文本被认为是积极的。消极：这指的是表达负面情绪或态度的文本。它包括悲伤、愤怒、挫败或批评的感觉。中性：中性情绪用于描述不表现出明显积极或消极情绪的文本，而是传达信息性、事实性或冷漠的语气。文本：餐厅的服务比预期慢，这有点令人沮丧。文本的极性是什么？消极

虽然使用LLMs进行分类问题似乎很简单，但仍有一些问题没有得到很好的解决。例如，当处理大量类别时，有效地提示LLMs仍然具有挑战性。请注意，如果我们面临一个非常困难的分类问题并且有一定数量的标记数据，微调LLMs或采用类似“BERT + 分类器”的架构也是可取的。

3.1.4.2 信息提取

许多NLP问题可以被视为信息提取问题，涉及从非结构化文本中识别或提取特定信息片段。这些信息可以包括命名实体、关系、事件和其他相关数据点。信息提取的目标是将原始数据转换为易于分析和在各种下游应用中使用的形式。

信息提取涵盖了广泛的问题，我们无法在此讨论所有问题。相反，我们从命名实体识别任务开始——这是一个长期以来一直是自然语言处理领域关注的问题。命名实体识别是一个将文本中的关键信息检测和分类到特定组的过程。这些关键信息，被称为命名实体，通常包括专有名词，并被归类到不同的类别，如人物、地点、组织、日期、货币值和百分比。首先考虑一个简单的例子，从文本中提取人名³。

³The text is from <https://edition.cnn.com/travel>

识别提供的文本中所有的个人姓名。

文本: {v*}

英国真的做得那么糟糕吗，还是说疫情之后旅行趋势已经改变？对于欧洲旅游组织首席执行官汤姆·詹金斯来说，是后者。他说：“我认为英国做得很好，但我们将会看到更多的人去欧洲，”他还提到，2024年“据说美国对英国的旅游需求略有下降。”

在提供的文本中识别出的姓名是：Tom Jenkins

然后，我们可以提取所有命名实体。

识别并将提供的文本中的所有命名实体分类到人员名称、地点、日期和组织等类别中。每个实体及其类型在一行中列出。

文本: {v*}

英国真的做得那么糟糕吗，还是说疫情之后旅行趋势已经改变？对于欧洲旅游组织首席执行官汤姆·詹金斯来说，是后者。他说：“我认为英国做得很好，但我们将会看到更多的人去欧洲，”他还提到，2024年“据说美国对英国的旅游需求略有下降。”

1. Tom Jenkins - 人名

2. 欧洲旅游组织 - 组织

3. 英国 - 位置

4. 欧洲 - 位置

5. 2024 - 日期

给定这些已识别的命名实体，我们可以使用其他信息提取任务进一步处理文本。例如，我们可以识别这些命名实体之间的关系（称为关系抽取）。相应的提示如下。

给定一段文本和其中识别出的命名实体列表，分析并描述这些实体之间的关系。解释每个实体如何与其它实体在上下文中相关联。

文本：{v*}

英国真的做得那么糟糕吗，还是说疫情之后旅行趋势已经改变？对于欧洲旅游组织首席执行官汤姆·詹金斯来说，是后者。他说：“我认为英国做得很好，但我们将会看到更多的人去欧洲，”他还提到，2024年“据说美国对英国的旅游需求略有下降。”

命名实体：

汤姆·詹金斯，欧洲旅游组织，英国，欧洲，2024

汤姆·詹金斯 - 欧洲旅游组织：汤姆·詹金斯是欧洲旅游组织的首席执行官。这表明这是一个专业协会，汤姆·詹金斯很可能在指导该组织活动方面发挥着重要作用，尤其是与欧洲旅游业相关的事务。

...

如果LLMs已经针对信息提取任务进行了指令遵循的微调，那么执行各种信息提取任务相对容易。例如，以下是一个信息提取的提示模板。

您将获得一段文本。您的任务是{*任务描述*}

文本：{*text*}

这里任务描述是任务的说明。说明的例子包括

extract the keywords from the text.

extract the key events from the text.

detect the coreference and anaphoric links between entities from the text.

等等。当然，我们可以提供更详细的任务描述，以更准确地提示LLMs。

3.1.4.3 文本生成

在NLP中，术语*text generation*被广泛使用，涵盖了一系列不同的问题，其中许多问题集中在基于给定文本生成文本。这些问题可以大致分为两类：1) 基于输入文本的连续写作（称为文本补全），2) 将输入文本转换为另一种文本（称为文本转换）。在此，我们说明了前者的提示，并在以下讨论中考虑后者的提示。

一个简单的文本补全示例是从几个给定的句子开始继续写一个故事。

您是一位擅长写故事的作家。您被提供了一些句子作为开头。请从提供的句子继续讲述故事。

故事开始：

一个阳光明媚的下午，莉莉在她家后院的一块石头下发现了一把隐藏的金色小钥匙。好奇之下，她捡起了钥匙，想知道它能打开什么。就在这时，她的猫，胡须，开始兴奋地喵喵叫，引领她走向花园边缘的老橡树。

莉莉跟着Whiskers，心中既兴奋又有些紧张。那棵老橡树对她来说一直显得神秘莫测，其扭曲的枝条即使在最晴朗的日子里也会投下长长的影子。当他们到达树下时，Whiskers猛地冲到前面，开始用爪子拨弄一堆秋叶下埋藏的东西...

同样，我们可以根据会话上下文创建一个对话。

您将获得一段游客和出租车司机的对话。请完成这段对话。

游客：你能带我去市中心博物馆吗？请？出租车司机：当然可以！第一次来这个城市吗？游客：是的，附近有什么必看的景点吗？

出租车司机：博物馆之后一定要去看看植物园。它们就在附近，非常美丽。

游客：太好了，谢谢你的建议！出租车司机：没问题。祝您玩得愉快！

文本补全的另一个应用是根据用户需求生成文本。需求可以包括风格、语气、长度以及文本应包含的任何特定内容。例如，用户可能需要一个正式的报告、一个创意故事或一段编程代码。以下是一个生成五个字的规范中文诗的提示。

你是一位诗人。请写一首传统中国诗。以下是要求。

主题：春天的复苏 结构：五言绝句 情感：通过生动的春天意象传达快乐和更新的感觉。

春意盎然

翠柳风中舞，
红花雨后新。
山明水又绿，
天宽鸟自频。

如果LLM被训练生成语言和代码，我们可以提示它执行代码补全任务。这里有一个例子。

Please write a Python function to calculate the average of a list of numbers.

```
def calculate_average(numbers):  
    if numbers: # Check if the list is not empty  
        return sum(numbers) / len(numbers)  
    else:  
        return 0 # Return 0 if the list is empty
```

3.1.4.4 文本转换

典型文本转换任务包括机器翻译、摘要、文本风格转换等。用于执行这些任务的提示与之前呈现的提示没有显著差异。在此，我们提供一些示例以展示这些提示。

将以下文本从英语翻译成西班牙语。

快速棕色的狐狸跳过懒惰的狗。El rápido zorro m
arrón salta sobre el perro perezoso.

总结以下文章，不超过50字：

文章：

近年来，城市地区面临着与可持续性和环境管理相关的重大挑战。城市正经历着污染率上升、交通拥堵加剧和对基础设施需求增加。这种增长导致了众多环境问题，包括空气和水污染水平升高、废物生产增加以及公共服务压力增大...

城市地区正面临着可持续性挑战，如污染加剧、交通拥堵和基础设施需求...

将此文本改写为正式语气。

文本：{v*}

嗨，最近怎么样？好久不见！

你好，最近怎么样？我们上次见面已经很久了！

3.1.4.5 问答

问答格式本质上是简单的。对于给定的问题，有一个与之对应的答案。例如，在开放域问答中，我们期望系统在用户提交问题后返回一个答案。通用问答的提示模板可以是：

{*question*}

Question: {*question*}

Answer: _____

问答在NLP中很重要，因为许多问题都可以被表述为问答任务。特别是，许多最近的推理任务都是以问答的形式定义的。例如，在MMLU基准测试[Hendrycks等人，2021]中，每个示例都包含一个多项选择题，并且要求LLMs选择正确答案。以下是一个关于如何回答这个数据集中问题的示例提示。

$(Z, *)$ 这是一个包含 $a * b = a + b + 1$ 的群, 对于所有 a, b 在 Z 中。 a 的逆元是 (A) 0 (B) -2 (C) $a - 2$ (D) $(2 + a) * -1$ D

另一个广泛使用的基准是GSM8K数据集[Cobbe等人, 2021]。它包含数千个小学数学应用题。对于每个问题, 我们使用LLMs以自然语言返回一个解决方案。例如, 以下是在GSM8K中解决一个问题的提示。

杰瑞的两个女儿在不同的棒球队打球。她们每个赛季都有8场比赛。每个球队每场比赛都要练习4小时。如果每场比赛持续2小时, 杰瑞总共要在球场上观看他的女儿们打球和练习多少小时?

A: Jerry将花费8场比赛 * 每场比赛2小时 = $\ll 8 * 2 = 16 \gg 16$ 小时观看一个女儿玩游戏。他将花费16 * 2 = $\ll 16 * 2 = 32 \gg 32$ 小时观看两个女儿玩游戏。他将花费8场比赛 * 4小时练习 = $\ll 8 * 4 = 32 \gg 32$ 小时观看一个女儿练习。他将花费32 * 2 = $\ll 32 * 2 = 64 \gg 64$ 小时观看两个女儿练习。他将花费总共32小时观看比赛 + 64小时观看练习 = $\ll 32 + 64 = 96 \gg 96$ 小时。#### 96

玛丽从商店买了六个苹果。从她买的苹果中, 对于她吃的每一个苹果, 她从剩下的苹果中种了两棵树。玛丽吃了多少个苹果?

她种了八棵树。这意味着她使用了那数量的一半苹果, 也就是8棵树 / 2棵苹果/棵树 = $\ll 8 / 2 = 4 \gg 4$ 个苹果。这意味着她种了她买的六个苹果中的四个, 剩下只有6个苹果 - 4个苹果 = $\ll 6 - 4 = 2 \gg 2$ 个苹果要吃。#### 2

Boris 有 100 个苹果。Beck 比Boris 少 23 个苹果。如果 Boris 给 Beck 10 个苹果, 现在 Beck 比Boris 少多少个苹果?

A: Boris开始时有100个苹果, 给了Beck 10个苹果, 剩下100个苹果 - 10个苹果 = $\ll 100 - 10 = 90 \gg 90$ 个苹果。最初, Beck比Boris少23个苹果, 给他一个起始计数为100 - 23 = $\ll 100 - 23 = 77 \gg 77$ 个苹果。从Boris那里收到10个苹果后, Beck的总数达到77个苹果 + 10个苹果 = $\ll 77 + 10 = 87 \gg 87$ 个苹果。因此, Beck现在比Boris少3 - 87 = $\ll 90 - 87 = 3 \gg 3$ 个苹果。#### 3

这里采用了几次提示。LLM不仅从这些问题-解决方案对的演示中学习了解决问题的方法，还学习了格式化输出的方法。例如，计算结果的最终结果遵循####标记， $\ll \dots \gg$ 注释了详细的计算步骤（称为计算注释）⁴。

3.2 高级提示方法

本章迄今为止，我们介绍了与LLM提示相关的基本概念，并呈现了多个用于NLP任务的提示。现在，我们考虑几种增强提示有效性的技术。

3.2.1 思维链

我们在本章和前几章中多次遇到了思维链（CoT）的概念[Wei等人，2022c；Chowdhery等人，2022]。CoT方法提供了一种简单的方式来提示LLMs为复杂问题生成逐步推理，从而以更类似于人类的方式完成任务。而不是直接得出结论，CoT方法指导LLMs生成推理步骤或从提示中提供的详细推理过程演示中学习。为了说明CoT，我们考虑了文献中通常描述的代数计算问题。假设我们被给出一个代数问题

Calculate the average of the numbers 2, 4, and 6.

我们可以将其视为问题并提示LLM回答。

Q: 请计算数字2、4和9的平均值。A: 答案是6。

对于LLM直接给出正确答案似乎很困难。一个简单的改进是在提示中添加类似问题的示例，这样LLM就可以从这些示例中学习。

Q: 请计算数字 1、3、5 和 7 的平均值。

A: 答案是 4。

Q: 请计算数字2、4和9的平均值。

A: 答案是 7。_____

这里的问题是，尽管我们已经展示了一个类似的问题-答案对，但对于LLM来说，推理出正确答案仍然很困难。在CoT中，LLM不仅可以从

⁴During prediction, a calculator is used when we see $\ll \dots \gg$. More specifically, once the LLM encounters “=” in a $\ll \dots \gg$, then the calculator calculates the expression on the left-hand side of “=”. This method helps reduce the calculation errors made by LLMs.

问题与答案之间的对应，但他们可能从用于推导答案的详细问题解决步骤中获得更多。为此，我们可以在提示中融入一些推理步骤以获得CoT提示。

请计算数字1、3、5和7的均方。A: 计算每个数的平方: $1^2 = 1$, $3^2 = 9$, $5^2 = 25$, 和 $7^2 = 49$ 。求平方和, $1 + 9 + 25 + 49 = 84$ 。总共有4个数。将总和除以项目数, $84/4 = 21$ 。答案是21。

Q请计算数字2、4和{v*}的平均值
9. A: 计算 $2 + 4 + 9$, 等于 15。有三个数字。将总和除以数量, 得到 $15/3 = 5$ 。答案是 5。

这里我们用绿色突出显示推理步骤。通过为类似问题提供详细的推理过程，LLM学会推理，从而生成一条通往正确答案的解决问题的路径。

使用CoT提示的优势有多个。首先，CoT允许LLM将复杂问题分解为更小、更连续的推理步骤。这在某种程度上反映了人类解决问题的行为，使其特别适用于需要详细、多步骤推理的任务。其次，CoT使推理过程更加透明和可解释。由于所有推理步骤都可见，我们可以理解和解释结论是如何得出的。第三，如果用户能理解推理过程背后的逻辑，他们更有可能信任LLM的预测。这在将LLM应用于医学、教育和金融等领域尤为重要。第四，CoT是一种情境学习方法，因此适用于大多数经过良好训练的现成LLM。此外，CoT提供了有效的方法来适应不同类型的问题。它甚至可以通过探索各种替代推理路径来激发更多创造性的解决方案，这些路径在直接得出结论时可能并不明显。

上述方法需要提供一个或多个CoT推理示例，通常称为少量样本CoT方法。相比之下，零样本CoT方法不需要此类示例。它相反地通过在提示中包含特定指令来逐步提示LLMs进行推理。例如，下面是一个零样本CoT提示。

Q: 请计算数字2、4和9的平均值。

A: 让我们一步步思考。

我们三个数字: 2、4和9。将这些数字相加, $2 + 4 + 9 = 15$ 。确定数字的数量, 在这种情况下是三个。平均值是通过将总和除以元素数量来计算的。完成除法得到 $15/3 = 5$ 。因此答案是5。

遵循“让我们一步步思考”的指示，LLM被提示生成详细的推理步骤。正如Kojima等人[2022]的研究中讨论的那样，使用此类指示提示可能导致LLM仅生成推理步骤而没有明确的结论。在这种情况下，可以使用第二轮提示来从这些推理步骤中提取答案。例如，Kojima等人[2022]创建了一个第二提示，该提示结合了输入和输出

在第一轮提示中。使用这个组合输入，LLM可以继续其推理过程，然后生成正确答案。此外，还可以提示LLM使用除“让我们一步步思考”之外的指令进行推理，例如“让我们逻辑地思考”和“请先展示你的思考步骤”。

虽然我们使用代数推理问题说明了CoT方法，但这些方法可以应用于各种不同的问题。CoT的典型问题解决场景包括数学推理、逻辑推理、常识推理、符号推理、代码生成等。请参阅图3.1以获取更多关于在各项任务中应用CoT的示例。

CoT今天是最活跃的提示工程领域之一。这不仅提高了LLM提示的性能，还为研究LLM推理能力开辟了广泛的方法。尽管我们在这个部分关注了CoT的基本思想，但它可以从几个方面进行改进。例如，我们可以将推理过程视为搜索许多可能路径的问题，其中每个路径可能由多个中间状态（即推理步骤）组成。一般来说，我们希望搜索空间定义良好且足够大，这样我们更有可能找到最优结果。因此，当前LLM研究的一个领域是设计更好的表示推理过程的结构，使LLM能够应对更复杂的推理挑战。这些结构包括基于树的[姚等，2024]，基于图的[Besta等，2024]等。通过使用这些推理路径的紧凑表示，LLM可以探索更广泛的决策路径，类似于系统2思维⁵。另一条研究路线侧重于通过多轮交互提示LLM。这包括将复杂问题分解为子问题，验证和细化模型输出，采用模型集成等。请注意，这些方法和涉及的问题不仅限于CoT。事实上，它们通常被用作更通用的方法来提高LLM，而CoT可以被视为测试LLM能力的一种方式。我们将在以下小节中看到一些这些问题的讨论。

在离开对CoT的讨论之前，我们应该考虑其实际局限性。其中之一是在少量样本的CoT场景中，需要详细的、多步骤的推理演示，这可能难以自动或手动获得。此外，没有标准方法将复杂问题分解为更简单的解决问题的步骤。这通常严重依赖于用户的经验。此外，中间步骤中的错误也可能影响最终结论的准确性。对于关于CoT的优缺点进一步讨论，感兴趣的读者可以参考该主题的最新调查[Chu等，2023；Yu等，2023；张等，2023a]。

3.2.2 问题分解

我们已经看到，LLMs可以通过将复杂问题分解为更简单的解决任务来从中受益。这种做法可以被视为一个更广泛范式的一个例子，即问题分解，这在心理学和计算机科学中已被广泛探索和讨论。从心理学的角度来看，复杂问题解决是指使用有助于克服障碍的知识来解决问题的过程。

⁵System 1 and System 2 thinking, as described by Kahneman [2011], represent two different modes of cognitive processing. System 1 is fast, automatic, intuitive, and emotional. This mode of thinking operates effortlessly and quickly, and is often what guides our daily decisions, judgments, and impressions. System 2 is slow, deliberate, and analytical. It is activated when we need to perform complex computations.

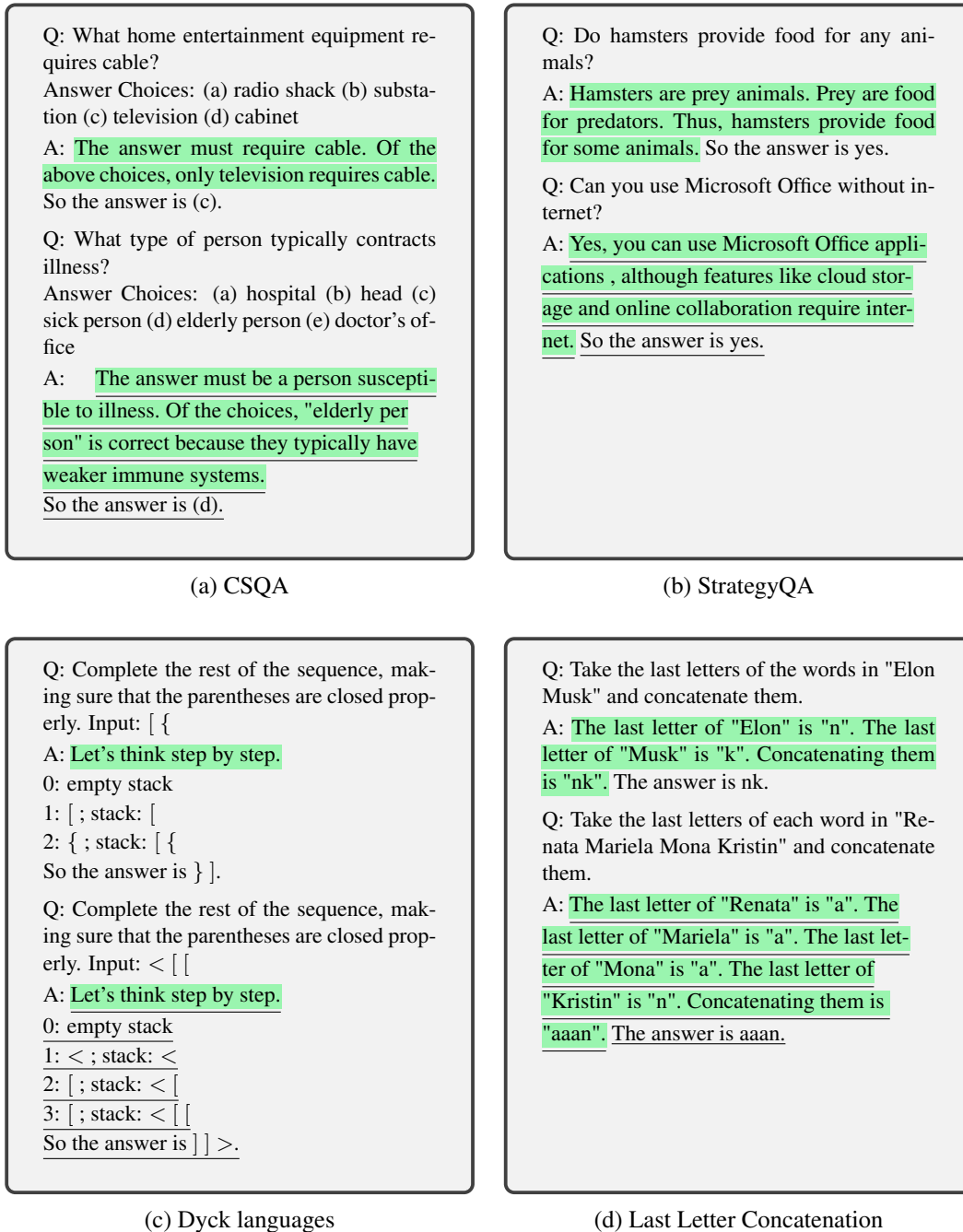


图3.1: 四个不同推理任务中的CoT, 包括CSQA、StrategyQA、Dyck语言和最后一个字母连接。CoT部分以绿色突出显示。

问题⁶。对于复杂问题, 通常没有标准或明确的解决方案路径。然而, 采用分解问题的策略通常是有利的, 这样可以更容易地以更少的努力解决相应的子问题。例如, 考虑写一篇关于人工智能风险的博客。如果我们只是向LLM发出指令“请写一篇关于人工智能风险的博客”, LLM可能会生成具有任意结构的博客。

⁶A relatively formal definition can be found in Frensch and Funke [2014]'s book: *complex problem-solving occurs to overcome barriers between a given state and a desired goal state by means of behavioral and/or cognitive, multi-step activities*.

写作风格。更好的方法可能是概述博客并提供每个部分的更详细信息。考虑以下提示

您是一位博客作者。请按照以下提供的提纲撰写一篇关于人工智能风险的博客。

- 介绍人工智能，其相关性以及理解其风险对青少年的重要性。
- 隐私担忧 讨论人工智能如何通过在线互动侵犯个人隐私。
- 错误信息 探索人工智能在传播错误信息和影响年轻人决策中的作用。
- 网络霸凌
突出AI工具在网络欺凌中的应用及其对心理健康的影响。
- 安全使用AI的技巧 提供负责任使用AI的指南并促进批判性思维。
- 结论
总结 m AI 焦点并鼓励与 AI 的主动互动 伦理。

这里我们为每个部分提供标题和主要要点。然后，LLM可以使用这种结构通过填充这些部分的内容来分解写作任务。请注意，博客的结构可以由人类提供，甚至可以自动生成。例如，我们可以先用LLM生成提纲，然后要求它遵循这个提纲来完成写作。

在计算机科学中，分解复杂问题是软件和硬件系统设计中常用的策略。一个著名的例子是分而治之范式，它通常用于设计可以将计算问题简化为更简单、更易于管理的算法。例如，考虑一个确定文档是否讨论了人工智能风险的问题。我们可以用以下提示指令LLM。

您提供了一个文本。请确定它是否讨论了人工智能的风险。

{*文档*}

如果文档很长，计算将很昂贵。或者，我们可以将文档分成相对较短的段落，并在每个段落上执行相同的任务。这些段落可以并行处理以进一步降低计算成本。接下来，我们确定

每个段落与AI风险主题的相关性。然后使用另一个提示生成最终输出。

您的任务是确定一篇文本是否讨论了人工智能的风险。该文本已被划分为几个部分，您已经获得了每个部分与人工智能风险主题的相关性。基于此，请提供您的最终结果。

段落 1: { *与主题1的相关性* } 段落 2:
{ *与主题2的相关性* } 段落 3: { *与主题3的相关性* }

...

现在让我们回到对提示中问题分解的更一般性讨论。虽然问题分解可以应用于各种NLP问题，但最近它在推理任务中得到了更广泛的讨论和测试。对于复杂的推理任务，我们通常需要一个多步骤的推理路径来得出正确的结论。我们可以通过三种不同的方式使用LLMs来实现这一点。首先，LLMs可以直接得出结论。换句话说，它们可以在没有明确的推理过程中进行预测，并且存在一个隐藏且不可解释的推理机制。其次，LLMs被提示生成一个多步骤的推理路径，以得出结论，就像CoT一样。然而，我们只运行一次LLMs，并且所有推理的中间步骤都在一个预测中生成。第三，我们将原始问题分解成多个子问题，这些子问题要么在LLMs的单独运行中解决，要么使用其他系统解决。在这里，我们关注第三种方法，它与问题分解密切相关。然而，需要注意的是，更全面的讨论可以涵盖所有这些方法，而前两种方法在本章中已经有所讨论。

一个通用的问题分解框架涉及两个元素。

- 子问题生成。这涉及到将输入问题分解为多个子问题。
- 子问题解决。这涉及到通过推理解决每个子问题，并得出中间和最终结论。

这两个问题可以用不同的方式建模，导致各种问题分解方法。一种方法是将它们视为两步过程中的单独步骤。例如，考虑本小节开头描述的博客写作任务。在第一步中，我们将整个问题一次性分解为子问题（即，概述博客）。在第二步中，我们按顺序或另一个顺序解决子问题（即，根据需要为每个部分填充内容）。此过程的最终输出结合了解决每个子问题的结果。虽然这种方法简单直接，但它假设问题是组合性的，因此更适合写作和代码生成等任务。

然而，许多现实世界的问题需要复杂的推理。这些问题的关键特征是推理步骤可能不是固定的。推理路径可能因问题而异，每一步推理都可能依赖于先前步骤的结果。在

在这些情况下，预先使用固定子问题生成是不理想的。相反，应根据输入问题动态生成子问题，并在推理过程中尽可能实时生成。与设计分而治之算法相比，这使得问题分解更具挑战性。理想情况下，我们希望共同设计子问题生成和子问题解决的系统。但更实际且广泛使用的方法是采用单独的模型来完成这些任务。实现这一点的直接方法是通过提示或调整模型来适应LLM来完成这些任务。

这里我们考虑了一种基于上述想法的方法，称为最少到最多提示法 [周等，2023b]。这种方法的动力源于解决困难推理问题的挑战——那些不能仅仅通过从几个例子中泛化来解决的问题。对于这些问题，更有效的解题策略是遵循一个逐步的子问题序列，系统地引导到结论。更具体地说，在最少到最多提示法中，通过提示LLM指令和/或演示来执行子问题生成。例如，下面是用于最少到最多提示法中子问题生成的2次提示。

TASK Your task is to decompose a problem into several sub-problems. You will be given a few examples to illustrate how to achieve this.

DEMO Q: In a community, 5% of the population are infants, 15% are children, 40% are adults, and 40% are seniors. Which group makes up the largest portion of the population?

A: To answer the question “Which group makes up the largest portion of the population?”, we need to know: “How many percent are infants?”, “How many percent are children?”, “How many percent are adults?”, “How many percent are seniors?”.

Q: Alice, Bob, and Charlie brought beads for their group project in their craft class. Alice has twice as many beads as Bob, and Bob has five times as many beads as Charlie. If Charlie has 6 beads, how many beads can they use for their craft project?

A: To answer the question “How many beads can they use for their craft project?”, we need to know: “How many beads does Bob have?”, “How many beads does Alice have?”.

USER Q: The environmental study conducted from 2015 to 2020 revealed that the average temperature in the region increased by 2.3 degrees Celsius. What was the duration of the environmental study?

A: To answer the question “What was the duration of the environmental study?”, we need to know: “When did the environmental study start?”, “When did the environmental study end?”.

通过学习示例，LLM可以为回答新问题“环境研究的持续时间是多少？”（用蓝色和橙色突出显示）生成两个子问题。给定这些子问题，我们按顺序解决它们。对于每个子问题，我们以所有先前生成的QA对作为上下文，然后生成答案。对于上面的示例，

我们需要 通过提示LLM解决第一个子问题，类似于{v*} e 这

The environmental study conducted from 2015 to 2020 revealed that the average temperature in the region increased by 2.3 degrees Celsius.

SUB-PROB1 Q: When did the environmental study start?

A: The environmental study started in 2015.

一旦我们解决了第一个子问题，我们就继续解决第二个。这次，我们将第一个子问题和其对应的答案都包含在输入中。

The environmental study conducted from 2015 to 2020 revealed that the average temperature in the region increased by 2.3 degrees Celsius.

SUB-PROB1 Q: When did the environmental study start?

A: The environmental study started in 2015.

SUB-PROB2 Q: When did the environmental study end?

A: The environmental study ended in 2020.

最后，我们使用LLM解决所有子问题的答案给出的原始问题。

The environmental study conducted from 2015 to 2020 revealed that the average temperature in the region increased by 2.3 degrees Celsius.

SUB-PROB1 Q: When did the environmental study start?

A: The environmental study started in 2015.

SUB-PROB2 Q: When did the environmental study end?

A: The environmental study ended in 2020.

FINAL Q: What was the duration of the environmental study?

A: The duration of the environmental study was 5 years.

最小到最大方法为提示LLMs分别生成和解决子问题提供了一个基本方法。我们可以从几个方面进行改进。一种简单的改进是应用各种高级提示技术，这不需要对问题分解框架进行更改。例如，我们可以将CoT纳入提示中，以增强子问题生成和解决的推理性能。

另一种改进是探索更好的问题分解和组织问题解决路径的方法。为了描述这些方法，我们将使用符号 p_0 来表示

输入问题，并使用符号 $\{p_1, \dots, p_n\}$ 表示与 p_0 对应的子问题。对于从少到多的提示，我们将 p_0 分解为 $\{p_1, \dots, p_n\}$ ，如下所示

$$\{p_1, \dots, p_n\} = G(p_0) \quad (3.2)$$

$G(\cdot)$ 表示子问题生成函数。然后，我们依次解决子问题 $\{p_1, \dots, p_n\}$ ，得到一系列答案 $\{a_1, \dots, a_n\}$ 。在回答第 i 个子问题 p_i 时，我们将原始问题 p_0 和所有之前看到的问题-答案对纳入上下文中进行预测。答案 a_i 由以下公式给出

$$a_i = S_i(p_i, \{p_0, p_{<i}, a_{<i}\}) \quad (3.3)$$

在 $p_{<i} = \{p_1, \dots, p_{i-1}\}$ 和 $a_{<i} = \{a_1, \dots, a_{i-1}\}$ 的地方。 $S_i(\cdot)$ 表示解决给定上下文 $\{p_0, p_{<i}, a_{<i}\}$ 的子问题 p_i 的函数。最后一步是生成原始问题的答案 p_0 ，它可以以类似于式(3.3)的方式表达。

$$a_0 = S_0(p_0, \{p_{\leq n}, a_{\leq n}\}) \quad (3.4)$$

一种改进此模型的方法是修改 $G(\cdot)$ 函数，以便模型可以动态生成答案。我们可以在解决问题的过程中逐个生成每个子问题，而不是一次性生成所有子问题 [Dua 等人, 2022]。为此，我们可以将公式 (3.2) 替换为

$$p_i = G_i(p_0, \{p_{<i}, a_{<i}\}) \quad (3.5)$$

因此，我们获得了一个以逐步方式进行操作的子问题生成模型。在每一步 i 中，我们首先通过提示一个LLM原始问题 p_0 和问题解决历史 $\{p_{<i}, a_{<i}\}$ 来生成子问题 p_i 。然后，我们使用相同的或不同的LLM，基于相同的环境信息（见公式 (3.3)）为这个子问题生成答案 a_i 。这种方法通过允许LLM在中间推理步骤中动态生成和解决子问题，有效地扩展了LLM的推理能力。因此，推理路径不是预先固定的，模型可以在问题解决过程中选择和调整它们的推理策略。

另一种改进上述模型的方法是专注于开发更好的子问题求解器。在我们的前一次讨论中，我们将 $S_i(\cdot)$ 限制为被提示解决子问题 p_i 的 LLM。实际上，我们可以将这个功能扩展到任何能够解决子问题的系统。例如， $S_i(\cdot)$ 可以调用 IR 系统，从而让我们能够访问更广泛的数据以解决问题。另一个例子是使用 $S_i(\cdot)$ 作为计算器，在数学问题解决中准确计算结果。如果子问题 p_i 复杂且需要多个中间问题解决步骤，也可以进一步将 p_i 分解为更小的子问题。例如， $S_i(\cdot)$ 可以定义为生成和解决子问题的递归程序。这把递归引入问题解决中，并允许我们通过迭代分解来解决问题。因此，我们可以为问题解决定义一个层次结构 [Khot 等人, 2023]。

如果我们对上述公式进行进一步的推广，我们可以将其视为一个强化学习问题。一种典型的方法是将问题解决过程建模为决策过程。在这个过程中，每一步都根据当前状态采取行动。这些行动

可以包括所有用于子问题生成和解决的函数（即， $G_i(\cdot)$ 和 $S_i(\cdot)$ ）。因此，动作序列对应于一个问题解决路径。由于强化学习问题的讨论超出了本章范围，我们省略了此学习任务的精确描述。尽管如此，开发一个确定何时以及如何生成和解决子问题的代理或控制器也是一个自然的选择。

在NLP中，问题分解与多跳问答研究的长线相关 [Mavi等人, 2024]。这项任务要求系统从多段文本中收集和结合信息，以提供一个对复杂问题的准确答案。例如，要回答“阿尔伯特·爱因斯坦出生的国家的首都是什么？”这个问题，我们需要知道“阿尔伯特·爱因斯坦出生在哪里？”和“德国的首都是什么？”。该领域和相关领域早期的工作已经研究了问题分解的问题，尽管这些方法可能不是基于LLM的。例如，一种流行的方法是开发一个额外的神经模型来生成更简单的问题，这些问题针对原始问题的不同方面 [Andreas等人, 2016; Talmor和Berant, 2018; Min等人, 2019]。这个问题生成器可以批量或顺序地创建问题。

广泛来说，问题分解也与NLP中的组合性问题相关[Droz dov等人, 2022; Press等人, 2023]。例如，在语义解析中，我们通过将自然语言句子分解成组成部分，并基于这些部分的意义以及组合它们的规则来理解句子，将自然语言句子映射到结构化的意义表示中。在早期对该领域的研究中，高度组合性的句子被认为对测试系统更容易，因为分解这样的句子并组合其部分的意义相对简单。然而，当需要对新数据进行更多泛化以建模组合性时，任务变得更加困难。在这种情况下，我们希望系统具有更好的组合泛化能力。在最近关于LLMs的研究中，这个问题在组合推理任务（如SCAN⁷）中经常被讨论，因为它被认为是测试LLMs语言理解和推理能力的重要方面。这也为开发和检验问题分解方法提出了新的任务。

在LLMs中，问题分解的一个有趣应用是工具使用。在某些情况下，将外部工具集成到LLMs中是必要的，以便访问在训练或微调期间不可用的准确数据。例如，LLMs可以与API集成以获取实时数据，如天气更新、股市价格或新闻源，使它们能够对用户查询提供最新的响应。当使用工具时，LLMs的预测可能包括指示如何调用外部API的位置和方式的标记。这需要将问题分解为子问题，其中一些由LLMs处理，而另一些由外部工具处理。更详细的讨论将在第3.2.5节中介绍。

3.2.3 自我精炼

在许多情况下，LLM的预测可能不准确或不正确。鉴于当前LLM可以执行如精炼和校正等任务，探索这些模型自我精炼其输出的方法是有意义的。自我精炼是人类心理中的常见现象。

⁷The SCAN tasks (Simplified versions of the CommAI Navigation tasks) are designed to evaluate the ability of LLMs to perform compositional generalization [Lake and Baroni, 2018]. They involve translating natural language commands into a sequence of actions. For example, a command “jump opposite left and walk thrice” can be translated into the action sequence “LTURN LTURN JUMP WALK WALK WALK”.

活动与日常行为。例如，在设计产品时，设计师可能会首先创建一个基本原型，然后在评估和测试后对设计进行细化，以提升用户体验和功能。细化可以迭代多次，直到设计令人满意。预测-再细化的思想也可以在自然语言处理（NLP）中找到。一个早期例子是Brill的标记器[Brill, 1992]，其中句子的初始词性标注结果可以使用基于规则的系统进行迭代细化。在深度学习的时代，关于序列到序列问题的大量工作，如语法纠错和文本重写，也可以被视为这一主题的例子。

我们可以提示LLMs进行自我优化。考虑一个简单的例子，我们首先使用一个LLM将中文文本翻译成英文，然后使用相同的模型来优化翻译。以下是我们可以提示模型进行翻译的方法：

以下文本从中文翻译成英文：{v*}

中文：

一系列考古发现奠定了红山文化在中华文明起源研究中的重要地位。最新公布的研究成果认为，大约从距今5800年开始，中华大地上各个区域相继出现较为明显的社会分化，进入文明起源的加速阶段。

英文：翻译文本：

一系列发现巩固了红山文化在中国文明起源研究中的重要作用。最新研究表明，大约5800年前，中国许多地区开始经历显著的社会分化，进入文明起源的加速阶段。

Then, we can prompt it to refine the translation:

您将获得一段中文文本及其英文翻译。如有必要，请对翻译进行润色以提高其准确性和流畅度。

中文：

一系列考古发现奠定了红山文化在中华文明起源研究中的重要地位。最新公布的研究成果认为，大约从距今5800年开始，中华大地上各个区域相继出现较为明显的社会分化，进入文明起源的加速阶段。

英文：翻译文本：

一系列发现巩固了红山文化在中国文明起源研究中的重要作用。最新研究结果表明，大约5800年前，中国许多地区开始经历显著的社会分化，进入文明起源的加速阶段。

细化：

一系列考古发现巩固了红山文化在中国文明起源研究中的重要作用。最新研究结果表明，大约5800年前，中国各地开始经历显著的社会分化，进入文明起源的加速阶段。

这里对输入翻译的修改以绿色突出显示。此过程可以是迭代的：在每一步中，我们使用前一步的翻译作为输入，并提示LLM生成可能更好的翻译。

在上述示例中，通过向LLM提示简单的指令如“请改进它！”实现了初始模型输出的改进。然而，这种改进仅基于LLM的指令遵循能力，没有关于如何以及在哪里改进模型输出的指导或监督。更有效的方法是考虑对需要改进的具体方面的反馈。例如，我们可以向LLM提示“请纠正翻译中的所有语法错误”，这样模型在改进过程中可以更多地关注语法错误纠正。

一个使用LLMs进行自我优化的通用框架包括三个步骤[Madaan等人，2024]。

- 预测。我们使用一个LLM来生成初始模型输出。
- 反馈收集。我们获取关于模型输出的反馈。
- 精炼。我们使用LLM根据反馈精炼模型输出。

最后两个步骤可以重复多次，从而导致一个迭代自我完善的过程。在这个框架中，接收准确和详细的反馈至关重要，因为它指导后续的完善步骤。一般来说，可以通过各种方式获得反馈 [Fernandes 等人，2023；Pan 等人，2024]。我们可以手动检查模型的预测以识别问题。此外，我们还可以使用反馈模型自动生成反馈。例如，在某个标记数据上训练的奖励模型可以帮助评估模型输出并返回一个表示其质量的分数。

本章的重点是LLM提示，我们考虑使用LLM来提供反馈。为了说明，让我们考虑一个生成对用户问题良好回答的例子：

- 首先，我们像往常一样提示一个LLM回答输入问题。

生成对以下问题的回答：“今天有哪些主要的环境问题？”

响应：

今天的主要环境问题包括气候变化、空气和水污染、森林砍伐、生物多样性丧失和海洋退化。

- 然后，使用以下提示，我们再次提示相同的LLM以获取对生成的响应的反馈。

您将获得一个问题及其相应的回答。评估该回答的准确性和深度，并识别任何事实错误或需要更多细节以增强理解的部分。

问题：

今天有哪些主要的环境问题？

响应：

今天的主要环境问题包括气候变化、空气和水污染、森林砍伐、生物多样性丧失和海洋退化。

反馈：

提供的响应准确，并突出了关键环境问题。然而，它将受益于对这些问题的更具体和详细的描述。以下是一些建议以供改进：

1. 包含具体的环境问题示例。
2. 添加关于主要污染物的信息。
3. 指定受影响最严重的地区。

- 根据LLM产生的反馈，我们提示它细化响应。

您将获得一个问题、其对应的回答以及对该回答的一些反馈。

问题：

今天有哪些主要的环境问题？

响应：

今天的主要环境问题包括气候变化、空气和水污染、森林砍伐、生物多样性丧失和海洋退化。

反馈：

提供的响应准确，并突出了关键环境问题。然而，它将受益于对这些问题的更具体和详细的描述。以下是一些建议以供改进：

1. 包含具体的环境问题示例。
2. 添加关于主要污染物的信息。
3. 指定受影响最严重的地区。

现在根据提供的反馈细化响应。

精炼响应：

关键环境问题包括气候变化，证据是全球气温上升和恶劣天气模式；城市和工业区域严重的空气和水污染；亚马逊等地区的滥砍滥伐，导致生物多样性丧失；以及海洋退化，以珊瑚礁白化和广泛过度捕捞为标志。

理想情况下，如果采用强大的LLM，我们希望它能够在不额外训练的情况下执行所有三个步骤。另一方面，如果我们有足够的标记数据来完成感兴趣的任务，我们可以通过监督学习来提高LLM的性能。例如，我们可以微调LLM以更好地适应细化任务，或者使用特定于任务的模型，这些模型不一定基于LLM [Welleck等人，2023；Schick等人，2023]。从更广泛的角度来看，提高LLM以进行自我细化任务可以被视为一个对齐问题。例如，已经发现可以通过RLHF激活一些自我纠正能力 [Ganguli等人，2023]。然而，讨论这些问题超出了本章的范围。更深入的讨论可以在第4章找到。

在LLMs中，自我完善与几个概念相关，这些概念揭示了这些模型的心理方面，例如自我反思的能力。一种观点认为，如果LLMs能够进行自我反思，它们的预测可以变得更加准确，甚至具有自我纠正的能力。这种自我反思可以通过多种方式激活，例如，通过提示这些LLMs进行更深入和细致的思考，或者通过提供模型可以从中学习和反思的例子。为了说明，我们在此考虑Li等人[2023a]的工作中提出的“先思考后生成”（DTG）方法，其中LLMs被提示进行思考。在DTG中，我们得到一个可能包含错误的初始模型输出。然后，LLMs被提示识别这种模型输出的错误类型并提供改进的输出。以下是针对中英翻译任务的DTG提示模板。

Given the Chinese sentence: {**source**}
 The English translation is: {**target**}
 Please first detect the type of error, and then refine the translation.
 Error Type:

我们旨在首先预测错误类型（红色），然后生成精炼的翻译（蓝色）。这个过程由指示“请先检测错误类型，然后精炼翻译”引导。这鼓励LLMs最初进行深思熟虑的分析，然后给出更好的结果。由于错误类型预测和精炼在LLMs的单次运行中执行，这种方法将反馈和精炼的步骤合并为一个过程。

在上述提示中，我们假设我们使用的LLM能够审查输入翻译并正确识别其错误类型。然而，这又带来了新的困难，因为模型可能不擅长在翻译中找到错误。这反过来又会导致额外的微调或提示工程工作。因此，一个更简单的方法是减少错误识别的负担，仅使用LLM进行审议。为此，我们可以用随机翻译替换输入翻译，并分配一个默认错误类型。以下是一个此类提示的示例。

给定以下中文句子：Translated Text: 给定以下中文句子：
 一系列考古发现奠定了红山文化在中华文明起源研究中的重要地位。
 The English translation is: 中文翻译是：
各种创新技术重新定义了现代艺术在当代文化研究中的重要性。
 请首先检测错误类型，然后细化翻译。
 错误类型： 翻译错误

在这个例子中，输入翻译不是由LLMs生成的，而是从数据集中随机抽取的。因此，它只是源句的一个错误翻译，我们可以相应地设置错误类型。然后，LLMs通过将源句和错误翻译作为输入来生成一个新的翻译。这种提示的设计也可以被视为通过“负面证据”[Marcus, 1993]激活LLMs的学习能力，从而使它们能够通过对比分析进行反思并产生更好的结果。尽管如此，这种方法不依赖于任何反馈，并且可以通过简单的提示来增强单个LLMs预测的性能。

注意，虽然DTG是非迭代的，但在NLP中迭代学习和改进是常用的。这些迭代方法的优势在于它们模仿人类的学习和问题解决，其中持续的反馈和调整导致结果逐步改进。迭代方法可以应用于各种LLM提示问题。例如，在问题分解中，可以在每一步将新的子问题和它们的解决方案纳入上下文中，因此LLMs可以逐步接近原始问题的解决方案。另一方面，迭代方法引发了一些非迭代方法中不存在的问题，例如，

早期步骤中的错误可能会对后续的问题解决产生负面影响，而确定何时停止迭代通常需要额外的工程努力。

3.2.4 集成

模型集成在文本生成中的NLP文献中被广泛讨论。这个想法是将两个或更多模型的预测结合起来以生成更好的预测。这项技术可以直接应用于LLMs。例如，我们可以收集一组LLMs并在相同的输入上运行每个模型。最终输出是这些模型的组合预测。

对于LLM提示，通过结合基于不同提示的预测也可以提高性能。假设我们有一个LLM和一组针对同一任务的提示。我们可以用每个提示运行这个LLM，然后结合预测。例如，下面是三个不同的文本简化提示模板。

Make this text simpler.

{*text*}

—

Condense and simplify this text.

{*text*}

—

Rewrite for easy reading.

{*text*}

—

每个这些提示都将导致不同的预测，我们可以考虑所有三个预测来生成最终的预测。

形式上，令 $\{x_1, \dots, x_K\}$ 为执行相同任务的 K 提示。给定一个 LLM $\Pr(\cdot|\cdot)$ ，我们可以使用 $\hat{y}_i = \arg \max_{y_i} \Pr(y_i|x_i)$ 找到每个 x_i 的最佳预测。这些预测可以组合成一个新的预测：

$$\hat{y} = \text{Combine}(\hat{y}_1, \dots, \hat{y}_K) \quad (3.6)$$

这里 $\text{Combine}(\{v^*\})$ 是组合模型，可以以几种不同的方式设计。例如，我们可以通过投票或识别与其他模型重叠最多的一种来选择最佳预测。模型组合的另一种方法是进行标记预测时的模型平均。设 \hat{y}_j 为模型组合在第 j -步预测的标记。预测 \hat{y}_j 的概率由以下公式给出

$$\hat{y}_j = \arg \max_{y_j} \sum_{k=1}^K \log \Pr(y_j|x_k, \hat{y}_1, \dots, \hat{y}_{j-1}) \quad (3.7)$$

在LLM提示的集成中，通常使用多样化的提示是有利的，以便组合可以捕捉更广泛的潜在响应。这种做法在集成学习中很常见，因为多样性有助于平均出任何单个模型或配置可能存在的偏差和错误。从贝叶斯观点来看，我们可以将提示 x 视为一个潜在变量，给定感兴趣的问题 p 。这允许给定 p 的 y 的预测分布被写成对所有可能的提示的边缘分布 $\Pr(y|x)$ 。

$$\Pr(y|p) = \int \Pr(y|x) \Pr(x|p) dx \quad (3.8)$$

积分计算 y 的总概率，通过考虑所有可能的 x 值，并按其在 p 给定的可能性进行加权。这里 $\Pr(y|x)$ 由LLM给出，而 $\Pr(x|p)$ 是问题的提示的先验分布。这是一个好模型，因为积分有效地考虑了 x 选择的不确定性，确保最终的预测分布 $\Pr(y|p)$ 稳健，并包含所有潜在的提示变化和偏差。然而，由于 x 可能的空间是无限的，直接计算这个积分可能是计算上不可行的。解决这个问题的方法之一是采用蒙特卡洛抽样等方法，使用可管理的有限数量的提示来近似积分。

虽然贝叶斯处理在数学上定义良好，但在NLP中，通常假设一个非信息性或均匀先验，并专注于构建一组多样化的提示。因此，输出可以使用如公式（3.6）所述的简单组合模型来计算。创建高质量、多样化的提示的问题已在CoT和其他情境学习领域得到研究。大多数研究集中在将各种演示示例纳入不同的提示中。在此，我们列出了一些这些方法。

- 给定一个问题，我们手动创建多个演示，并针对不同的提示使用不同的演示。
- 给定一个问题，我们使用LLMs自动生成演示和提示。
- 给定一个提示，我们通过改变提示中演示的顺序来创建不同的提示。
- 给定一个提示，我们使用LLMs生成多个相似的提示。
- 给定一个提示，我们将其转换为其他形式，例如将其翻译成其他语言。

当然，在实践中，我们可以结合这些方法以实现更大的多样性。这里的潜在假设是，多样化的提示可以导致多样化的模型输出。这在我们处理的问题相对较新且困难时尤其如此。对于更强健、更鲁棒的LLM，相似提示的输出方差可能不大。在这种情况下，涉及多个提示的好处可能相对较小。

除了为LLM提供多样化的提示外，另一种方法是利用LLM输出中的固有差异。生成多个输出的简单方法是从假设空间中采样输出。这对于LLM来说很简单，因为它们的搜索算法天生适合产生多个结果。例如，如果使用束搜索，我们可以轻松收集束中的所有完整假设以生成多个输出。

候选人。此外，我们还可以修改搜索算法以实现更强大的采样方法并探索更广泛的优质假设范围。

下一个问题是如何结合不同的模型预测。这使我们回到了如公式 (3.6) 所述的模型组合问题。在这里，我们考虑自洽性方法，它输出的不是概率最高的预测，而是与其他预测最一致的预测[Wang等人, 2022a; 2023b]。首先，像往常一样，用CoT提示一个LLM，并通过采样生成多个推理路径。然后，统计这些推理路径中每个答案的频率。最终输出的是频率最高的答案。下面展示了通过用CoT提示一个LLM得到的三个不同预测。

预测 1 (正确) :

三個朋友玩一個遊戲，他們拋一枚公平的硬幣。每個朋友拋一次硬幣。恰有一個朋友拋出正面朝上的概率是多少？

A: 让我们一步步思考。

每个抛硬币事件都是独立事件，有50%的概率落地为正面（H）和50%的概率落地为反面（T）。考虑三次抛掷的可能结果：HHH、HHT、HTH、HTT、THH、THT、TTH、TTT。在这些结果中，只有HHH、HHT、HTH和THH是相关的。我们只对恰好有一个正面的场景感兴趣：HTT、THT和TTH。在总共8种结果中，有3种这样的结果，给出概率为3/8。

预测 2 (正确) :

三個朋友玩一個遊戲，他們拋一枚公平的硬幣。每個朋友拋一次硬幣。恰有一個朋友拋出正面朝上的概率是多少？

A: 让我们一步步思考。

使用二项概率公式，其中 p (成功或正面) 为 0.5，且每个 3 次试验（翻转）是独立的，我们寻找恰好一次成功（正面）。恰好 k 次成功的 n 次试验的公式由二项式系数公式 $\binom{n}{k} p^k (1-p)^{n-k}$ 给出。将值代入，我们得到 $\binom{3}{1} (0.5)^1 (0.5)^{3-1} = 3 \cdot 0.5 \cdot 0.25 = 0.375$ 。

Prediction 3 (Wrong):

三個朋友玩一個遊戲，他們拋一枚公平的硬幣。每個朋友拋一次硬幣。恰有一個朋友拋出正面朝上的概率是多少？

A: 让我们一步步思考。

由于每次抛硬币有50%的概率落地正面和50%的概率落地反面，而我们正在寻找只有三次抛掷中的一次结果是正面的情况，我们可能会首先认为这只是一个选择哪次抛掷是正面的简单问题。有三次抛掷，所以有人可能会错误地假设选择这种特定结果的机会是1/3。因此，他们可能会得出恰好一次正面的概率是 $1/3 = 33.3\%$ 。

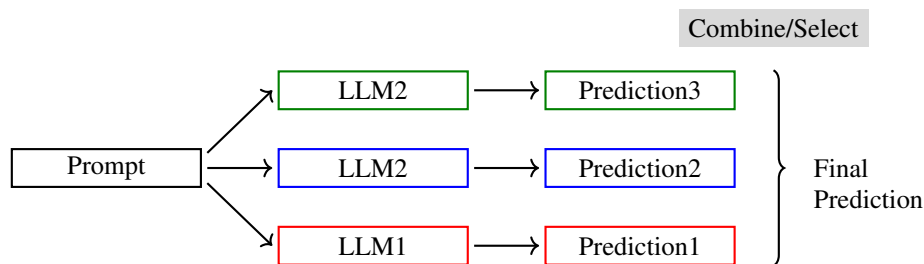
预测1和2正确地识别了恰好翻转一个硬币的三种情况，两者都获得了37.5%的概率。预测3中的推理未能考虑到三次抛硬币可能的总结果数，因此给出了错误的33.3%的答案。因此，我们选择37.5%作为最终答案，因为它是一致意见。

自洽性为在候选集中确定最佳预测提供了一个标准。由于在此方法中提示和模型是固定的，因此它不是严格意义上的提示集成方法。相反，它可以被视为输出集成方法的一个实例，也称为假设选择方法，这些方法在NLP中已被长期探索，尤其是在文本生成问题中[Xiao等人, 2013]。在这些方法中，通过改变模型架构或参数生成多个输出。然后根据某些标准对每个输出进行评分，并根据这些评分重新排序输出。定义评分函数有多种方式，例如测量输出之间的协议，并使用更强的模型重新评分每个输出⁸。图3.2显示了不同LLM集成方法的比较。

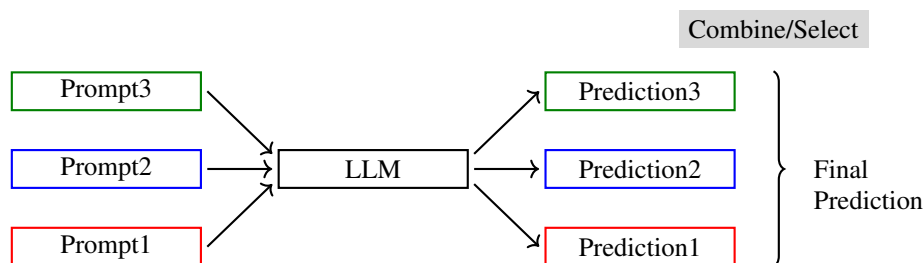
现在，让我们简要回顾一下本节中已讨论的方法，例如问题分解和自我完善。显然，这些方法通过将更多“选择”引入推理过程来增强决策。在某种程度上，它们都涉及评估和提供对LLMs结果的反馈。例如，在自我完善中，我们需要为改进LLMs的预测提供建议，而在输出集成中，我们从一组候选者中选择最佳输出。从这个意义上说，这些方法属于更广泛的预测-验证方法类别，其中首先进行预测，然后进行验证和改进。这里的基本问题涉及验证和评估推理结果或中间步骤。这个问题与RLHF中训练奖励模型的问题有些相关，尽管RLHF解决的是不同方面。事实上，验证器在LLMs推理中的开发已被探索并实施。大多数工作，而不是开发基于启发式的推理时间算法，而是以监督方式学习验证器。一种直接的方法是将验证器训练为二元分类器，例如将答案分类

⁸ An interpretation of self-consistency is to view it as a minimum Bayes risk search process. It searches for the best output by minimizing the Bayes risk. More specifically, a risk function $R(\mathbf{y}, \mathbf{y}_r)$ is defined on each pair of outputs (denoted by $(\mathbf{y}, \mathbf{y}_r)$), representing the cost of replacing \mathbf{y} with \mathbf{y}_r . Given a set of outputs Ω , the risk of an output $\mathbf{y} \in \Omega$ is given by

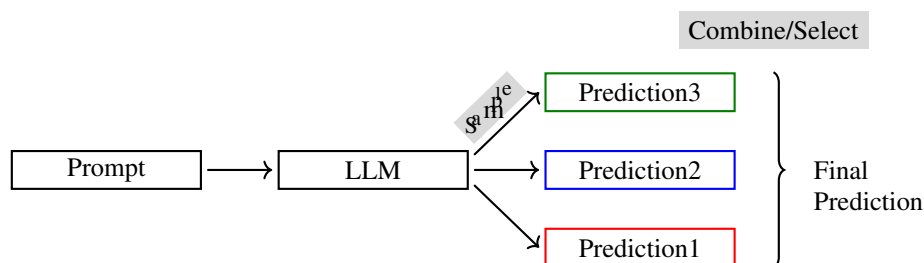
$$\begin{aligned} \text{Risk}(\mathbf{y}) &= \mathbb{E}_{\mathbf{y}_r \sim \text{Pr}(\mathbf{y}_r|\mathbf{x})} R(\mathbf{y}, \mathbf{y}_r) \\ &= \sum_{\mathbf{y}_r \in \Omega} R(\mathbf{y}, \mathbf{y}_r) \cdot \text{Pr}(\mathbf{y}_r|\mathbf{x}) \end{aligned} \quad (3.9)$$



(a) Model Ensembling



(b) Prompt Ensembling



(c) Output Ensembling

图3.2: LLM的集成方法。在标准模型集成 (a) 中, 使用多个在架构或参数上有所不同的LLM。每个LLM接收相同的提示并产生一个预测。这些预测被组合以生成最终的预测。在提示集成 (b) 中, 我们有一个LLM和多个提示。LLM为每个提示产生一个预测, 并且这些预测通常被组合。在输出集成 (c) 中, LLM在给定提示的情况下对预测空间进行多次采样。这可以被视为提升LLM自身性能的方法。请注意, 这些集成方法可以组合起来以增加预测的多样性。例如, 我们可以同时使用提示集成和输出集成以获得更多样化的预测。

尽管这些验证器通常用作评分模型, 但它们正确或错误, 给定一个问题的推理路径, 验证器可以用来评分整个路径 (称为基于结果的方法) [Cobbe 等人, 2021], 或者每个单独的推理步骤 (称为基于过程的方法) [Uesato 等人, 2022; Lightman 等人, 2024]。

3.2.5 RAG和工具使用

RAG通常用于标准LLMs (仅依赖预训练知识) 在生成的文本中缺乏准确性和深度时。通过从外部数据库和文档中提取,

RAG可以显著提高响应质量，确保它们在上下文中相关且在事实上正确。这种方法在需要高事实准确性和最新信息的场景中特别有用，例如复杂问题回答。

RAG的概念在前面章节中多次被提及。为了完整性，我们在此概述RAG涉及的关键步骤。

- 我们准备了一个文本集合，这些文本被视为我们可以访问的额外知识来源。
- 我们检索给定查询的相关文本。
- 我们输入检索到的文本和查询到LLM中，然后提示其生成最终预测。

步骤 1 和 2 可以通过使用外部信息检索系统来实现。例如，我们可以将文本集合存储在向量数据库中，然后通过基于向量的搜索技术检索最相似的文本。由于信息检索不是本章的重点，我们将假设此类系统是现成的，并直接使用它们。

这里我们展示如何提示LLMs利用检索到的文本。为了说明，考虑一个使用LLMs回答以下问题的例子。

Where will the 2028 Olympics be held?

我们可以简单地将这个问题输入到在线搜索引擎中。然后，它将返回在互联网上找到的相关文本片段，例如，

(维基百科)

The 2028 Summer Olympics, officially the Games of the XXXIV Olympiad and commonly known as Los Angeles 2028 or LA28, is an upcoming international multi-sport event scheduled to take place from July 14-30, 2028, in the United States. ...

(体育新闻)

In 2028, Los Angeles will become the third city, following London and Paris respectively, to host three Olympics after hosting the Summer Games in 1932 and 1984. It will also be the first time the United States has hosted an Olympic Games since the 2002 Winter Games in Salt Lake City. ...

...

我们可以使用这些检索到的文本作为额外上下文，并提示LLM根据这些文本生成响应。下面是一个示例RAG提示。

您的任务是回答以下问题。为此，提供了相关文本。请根据这些文本回答问题。

问题：

2028年奥运会将在哪里举行？

相关文本 1：

2028年夏季奥运会，正式名称为第三十四届奥林匹克运动会，通常被称为洛杉矶2028年奥运会或LA28...

相关文本 2：

在2028年，洛杉矶将成为继伦敦和巴黎之后，第三个举办三次奥运会的城市，之后...

...

2028年奥运会将在洛杉矶举行。

此提示假定提供的文本与问题相关，并期望LLM使用这些文本生成忠实于事实的回答。然而，信息检索系统有时可能会提供不相关或不正确的文本，这可能导致LLM产生错误的答案。解决此问题的一个简单方法是通过提高信息检索系统的准确性。尽管如此，与大多数AI系统一样，错误仍然可能发生。因此，还需要增强LLM的鲁棒性，以便即使在输入不准确的情况下也能做出合理的预测。下面是一个新的提示，使LLM更忠实于事实，并允许它在提供的信息不准确时选择不回答问题。

Your task is to answer the following question. To help you with this, relevant texts are provided. Please base your answer on these texts.

Please note that your answers need to be as accurate as possible and faithful to the facts. If the information provided is insufficient for an accurate response, you may simply output "No answer!".

Question:

Where will the 2028 Olympics be held?

Relevant Text 1:

The 2024 Summer Olympics, officially the Games of the XXXIII Olympiad and branded as Paris 2024, were an international multi-sport event ...

...

No answer!

在这个例子中，LLM 拒绝回答，因为提供的信息不足且与问题无关。

RAG和微调是使用特定任务数据适应LLMs的常见方法。标准RAG无需训练，可以直接应用于LLMs。为了进一步提高RAG，它

也可能微调LLMs，尽管这需要一些训练努力。例如，我们可以使用人工标注的数据来微调LLMs，以监督它们学习拒绝回答。请注意，虽然上面显示的示例似乎很简单，但RAG并非微不足道。从提示工程的角度来看，不同的用例可能需要不同的提示，尽管我们的目标相对“贪婪”，即开发一种通用的提示策略，该策略可以适应不同的任务。在许多情况下，我们需要控制我们在预测中依赖检索到的上下文程度。有时，LLMs必须严格从提供的文本中推导出响应，而在其他时候，如果提供的文本不足，它们可能需要使用其预训练的知识来生成响应。RAG有许多方面，例如改进检索系统，这些内容无法在本章中涵盖。感兴趣的读者可以参考RAG技术的调查报告以获取更多信息[Li等，2022；Gao等，2023c]。

一个我们在这里讨论RAG的原因是，它可以被广泛地视为一般问题分解框架的一个实例（参见第3.2.2节）。RAG将问题解决分为两个步骤。在第一步中，我们从各种知识源中收集与给定查询相关和支持的信息。在第二步中，我们使用LLMs根据收集到的信息生成响应。如果我们进一步扩展问题分解的概念，我们会发现许多需要使用外部系统或工具的任务可以被视为类似的问题。这样的一个例子是在LLMs中的工具使用。在许多应用中，LLMs需要使用外部数据库、API甚至模拟工具来生成准确的响应。例如，LLMs可以访问金融市场实时数据以提供最新的投资建议或与医疗保健数据库集成以提供个性化的医疗见解。这种集成通过允许LLMs与外部系统交互，在某些情况下影响或控制外部系统，从而扩展了LLMs的功能。因此，LLMs更像自主代理而不是单纯的文本生成器[Franklin和Graesser, 1996]。

工具使用问题范围广泛。在此，我们将讨论范围缩小到可以通过调用外部API来解决一些子问题的任务[Parisi等人，2022；高等人，2023b]。再次考虑要求一个LLM回答“2028年奥运会将在哪里举行？”的例子。假设LLM可以访问网络搜索工具。然后我们可以提示LLM使用网络搜索来回答问题，如下所示

您的任务是回答以下问题。您可以使用外部工具，例如网络搜索，来协助您。

问题：

2028年奥运会将在哪里举行？

关于此问题的信息如下：

`{tool: web-search, query: "2028 Olympics"}` 因此
答案是：洛杉矶

这里 `{tool: web-search, query: "2028 Olympics"}` 表示使用查询词“2028奥运会”对网络搜索系统进行请求。当LLM看到这个字符串时，它会执行网络搜索并使用结果替换字符串。然后，在预测的后续步骤中，LLM将这个网络搜索结果作为上下文来生成正确的答案。

考虑另一个例子，我们要求LLM解决一个数学问题。

问题：

一个游泳池需要用水填充。游泳池长10米，宽4米，深2米。计算游泳池的体积（以立方米为单位），然后确定填充它需要多少升水（考虑到1立方米等于1000升）。

解决方案：

要解决这个问题，LLM 首先需要使用长方体体积公式来计算水池的体积：长度×宽度×深度。因此，体积是 $10\text{ m} \times 4\text{ m} \times 2\text{ m}$

$= \{\text{tool: calculator, expression: } 10 * 4 * 2\} \text{ m}^3$ 。接下来，为了找出需要多少升水，LLM 将立方米体积乘以 1000（因为 1 立方米等于 1000 升）。因此， 80×1000

$= \{\text{tool: calculator, expression: } 80 * 1000\} \text{ 升}$ 。

这里字符串 `{tool: calculator, expression: 10 * 4 * 2}` 触发调用数学解释器来计算表达式的结果。请注意，结果（即，80）将替换

`{tool: calculator, expression: 10 * 4 * 2}` 并可以在以下标记预测中引用。例如，在解决问题的最后一步中，使用 80 而不是

`{tool: calculator, expression: 10 * 4 * 2}`。

A key difference between the tool use examples here and the previously discussed RAG examples is that in tool use, external functions can be called during inference. In contrast, in RAG, the retrieved texts are provided before the prediction process begins. However, from the language modeling perspective, they are actually doing the same thing: before generating the final result, we use external tools, either manually or automatically, to obtain sufficient and relevant context. A high-level interpretation of these approaches is that they both rely on an “agent” that can determine where and how to call external functions to generate the context necessary for prediction.

工具使用的问题在于原始的LLM没有训练生成工具使用的必要标记。因此，我们需要微调LLM以适应这些任务[Schick等人，2024]。由于本章重点在于提示，我们将不展示这个微调过程的细节。简单来说，我们首先需要标注数据。对于每个微调示例，我们将输出中需要使用外部工具的部分替换为预定义的命令或标记。然后，我们使用这些标记数据来像往常一样微调LLM的参数。因此，LLM可以获得生成调用外部工具命令的能力。在推理过程中，我们可以在模型输出中执行这些工具使用命令，以获得外部工具的帮助。

3.3 学习提示

到目前为止，在本章中，我们考虑了几个基本提示策略及其各种改进。然而，我们讨论的所有提示都是手动设计的。这导致了一系列问题：首先，设计高质量的提示本身就很困难，需要大量的手动工作。例如，通常需要进行大量的实验来识别最有效的提示。由于不同的LLM可能对某些提示的响应更好，因此

提示类型，开发普遍有效的提示可能更加资源密集。其次，手动提示设计高度依赖人类专业知识，这可能会限制方法的多样性，并忽略那些对人类来说并不立即明显的可能有效的提示。第三，人类创建的提示可能复杂且冗余，导致LLM的输入更长，计算成本更高。

在这一节中，我们讨论自动化提示技术。这些方法旨在自动创建、优化和表示提示，以便更有效地解决下游任务。特别是，我们在此考虑三个问题。

- 如何自动化设计并优化LLMs的提示过程？
- 存在除了字符串之外的其他表示提示的形式吗？我们如何学习这样的表示？
- 如何使提示更加简洁紧凑，从而降低其复杂性和长度？

请注意，我们可以调查这些问题的设置有很多。例如，我们可能指定提示专门为特定LLM开发，或者开发独立于所使用的LLM。这些设置可能导致不同的方法和应用场景，但这些方法在某些方面可能存在重叠。在以下讨论中，我们将涵盖几个不同的场景，并讨论各种方法之间的联系。

3.3.1 提示优化

给定提示设计困难且劳动密集，使用机器学习模型来发现特定任务的优化提示（称为自动提示设计或提示优化）是很有吸引力的。这种方法可以广泛地被视为自动化机器学习（AutoML）的一个实例，其目标是减少或消除对专家驱动的手动设计机器学习模型的依赖。尽管我们在这里的关注点是提示设计，但提示本身是离散结构。因此，设计提示与设计机器学习模型（如离散模型架构）非常相似。也许最相关的领域之一是神经架构搜索（NAS），其中通过探索可能的神经网络空间来识别最优秀的神经网络[Zoph和Le, 2016; Elsken等人, 2019]。如果我们把提示优化看作是一个搜索过程，那么我们可以描述一个包含以下组件的通用提示优化框架：

- 提示搜索空间。这定义了算法可以探索的所有可能的提示。例如，可以编辑一些种子提示以生成一组多样化的候选提示。
- 性能估计。一旦选择了一个提示，就需要对其进行评估。例如，一种直接的方法是将其输入到LLM中，并测量其在验证集上的性能。
- 搜索策略。搜索过程通常与许多人工智能系统中所使用的相同。在每一步中，系统都会在搜索空间中探索一组有希望的提示。

评估它们。此过程在探索更多提示时继续。搜索的结果是搜索停止前观察到的最佳性能提示。

这是一个非常通用的框架，不同的提示优化系统在各个组件的设计上可能有所不同。一种广泛使用的方法是以LLM为基础来开发这些组件。最初，提供一些提示。然后，以下过程会迭代，直到满足停止标准：1) 在验证集上评估提示；2) 通过仅保留最有希望的提示来维护候选池；3) 通过使用LLM从候选池中推断相似提示来创建新的提示。这种方法的一个好处是，它允许我们使用现成的LLM来执行上述任务，而无需进行大量系统开发。为了实现这一点，我们可以提示或微调LLM以适应这些任务。在这里，我们考虑了周等人[2023c]基于LLM的提示优化方法。它包括以下步骤。

- 初始化。让 C 代表我们打算探索的候选提示池。第一步是将初始提示添加到 C 中。我们可以通过几种方式来完成这项任务。一种简单的方法是为特定任务手动创建这些提示。然而，在许多情况下，当人类对如何编写有效的任务提示了解有限时，开发提示变得具有挑战性。在这些情况下，使用 LLM 生成提示是可取的。例如，我们可以直接指示 LLM 生成提示，并向它们提供任务描述。

你被分配了一个使用LLMs完成的任务。请编写一个提示来指导LLMs。

{*任务描述*}

这种方法很简单，但仍需要人类提供的任务描述。另一种方法是使用LLMs根据任务的输入和输出示例生成提示。以下是一个提示模板。

您将获得几个输入输出对的任务。请编写执行此任务的说明。

输入：{*input1*} 输出：{*output1*} 输入：

{*input2*} 输出：{*output2*}

...

因此，LLMs 可以从提供的输入和输出中推断出对应任务的指令。

- 评估。一旦我们获得候选池 C ，我们需要评估 C 中的提示。一种方法是将每个提示输入到LLM中，并评估其在下游的结果

任务。例如，我们可以使用预定义的指标来评估给定输入的LLM输出，或者使用输出的对数似然作为提示质量的一个衡量标准。

- 剪枝。如果 C 包含大量提示，则对其中的不具前景提示进行剪枝是合理的，从而减少后续步骤的计算负担。这是一个标准的剪枝问题。对于每个提示的评估分数，一个简单的方法是只保留一定百分比的提示，其余的丢弃。
- 扩展。扩展是搜索算法中用于探索搜索空间中不同状态的关键操作。这里的扩展操作可以定义为函数

$$C' = \text{Expand}(C, f) \quad (3.10)$$

C' 是使用模型 f 从 C 生成的新的提示集合。如果我们把 f 看作一个 LLM，我们可以通过指示 f 根据以下内容生成新的和相关的提示来执行扩展操作。以下是一个示例。

以下是LLM的提示。请提供一些执行相同任务的新提示。

输入：{*提示*}

——

然后，我们将 C 替换为 C' 。评估、剪枝和扩展的步骤可以重复进行，因此我们可以逐步探索更广泛的提示范围。

在提示优化中，扩展步骤起着关键作用，因为它定义了我们如何探索搜索空间，我们的目标是找到以最小努力获得的最优结果。对此步骤的一种改进是将问题视为一个释义任务。一种简单的方法是应用现成的释义系统，这些系统基于LLMs或其他模型，将输入提示转换为语义等价的形式[Jiang等人, 2020]。或者，我们可以为每个标记定义特定的编辑操作，如插入和修改。可以通过应用这些操作将给定的提示编辑成新的提示[Prasad等人, 2023]。此外，还可以进行进一步的评价和修剪，以过滤掉低质量的提示。除了将提示生成视为一个释义问题外，我们还可以通过学习反馈来提高扩展过程中的提示质量[Pryzant等人, 2023]。这种方法与第3.2.3节中讨论的自我改进问题有些相关。可以使用LLM生成关于输入提示的反馈，然后根据此反馈进行修订。这个反馈和修订周期可以重复多次，直到结果收敛或达到预期结果。

另一种提示优化的方法是应用经典优化技术。例如，可以将问题表述为进化计算问题，其中提示被视为候选方案，随着优化的进行，这些候选方案会代代进化 [Guo 等人, 2024]。由于相关领域已经开发了众多强大的优化算法，因此它们可以直接应用于此问题。

实际上，我们可能会倾向于使用现有的LLM API来实现上述步骤。然而，这种做法将强烈依赖于LLM的推理和在上下文中的学习能力。如果这些LLM不够强大且缺乏对任务的适应性，它们可能会在搜索中引入错误，例如在扩展过程中生成错误的提示。在这种情况下，训练更适合任务的模型更为可取。在这个研究方向中，一种方法是通过强化学习，它已被广泛用于解决离散决策和优化问题。例如，Deng等人[2022]通过将基于FFN的适配器集成到LLM中开发了一个提示生成器。提示生成器被训练为一个典型的策略网络，但在训练过程中，只有适配器的参数被更新，而模型的其余参数保持不变。在训练过程中，通过使用另一个LLM测试生成的提示来获得奖励，类似于上述讨论的评估方法。一旦训练完成，提示生成器就被用来生成新的提示。

注意，在我们的讨论中，提示被简单地视为一系列标记的序列，提示优化的输出就是这样一种序列。然而，在严格意义上，提示具有复杂的结构，包括用户输入、指令和演示等不同领域。虽然我们讨论的方法大多是通用的，但许多关于提示优化的工作都集中在学习更好的指令上。具体来说，目标是生成基于给定任务的指令，以有效地引导LLMs。当然，提示优化的概念也可以扩展到学习提示的其他部分。例如，在CoT中，对学习选择或生成演示的研究兴趣很大 [刘等，2022；鲁宾等，2022；张等，2023b]。学习指令与学习演示之间的一个区别是，使用LLMs生成高质量的演示相对容易，学习演示的重点通常是如何从候选集中采样适当的演示。相比之下，学习指令的困难部分在于预训练的LLMs不适合预测指令的质量，并且将这些指令应用于下游任务在计算上很昂贵。这使得优化方法的应用成本高昂，探索各种指令提出了重大挑战。

3.3.2 软提示

尽管开发自然语言提示，无论是手动还是自动，都是一个简单且广泛应用的方法，但它也带来了一些问题。一个问题就是自然语言提示可能很复杂且长，当通过LLMs处理时会产生显著的计算负担。在许多应用中，用户可能需要重复执行任务，将相同的长提示输入到LLMs中大量次数显然是不高效的。另一个问题是，虽然提示通常在常规LLM输入中以离散的标记序列（称为硬提示）表示，但LLMs将它们编码为低维实值向量。这引发了关于是否存在更紧凑和高效的方式来表示提示的问题。

在这个子节中，我们介绍了软提示的概念，这可以被视为提示的隐藏、分布式表示。在提示LLMs时，我们关注的是传达任务或问题以引发所需响应。我们可以将硬提示定义为用户直接输入到LLMs中以引导响应的显式、预定义文本序列。相比之下，我们可以将软提示视为嵌入在LLMs中的隐式、可适应的提示模式。与硬提示不同，硬提示用自然语言表达，应该对用户可理解，

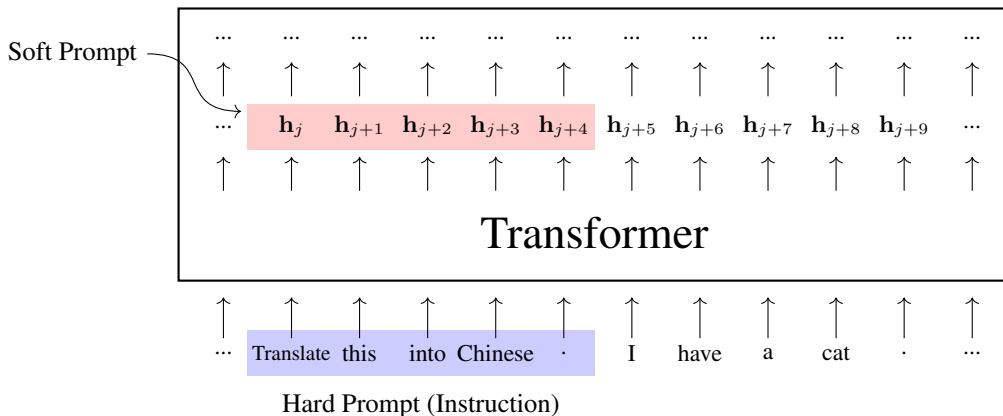


图3.3: 硬提示和软提示的说明。在这里, 硬提示是我们输入到LLM以执行任务的指令。LLM通常将此指令编码, 与指令对应的中间表示可以视为某种软提示。

人类, 软提示是以对模型更易理解的方式而不是对人类易理解的方式编码的。为了说明这一点, 考虑一个简单的提示

将句子翻译成中文。

已完成!

这里, 指令“将句子翻译成中文”可以被视为一个硬提示, 用标记序列 $c_1 \dots c_5$ 表示。将这些标记输入到LLM中, 它们被转换成一系列对应于标记的实值向量 $h_1 \dots h_5$ 。我们可以大致将 $h_1 \dots h_5$ 视为一个软提示, 如图3.3所示。

虽然上述示例表明可以通过转换硬提示来生成软提示, 但它们之间并不一定有直接的对应关系。事实上, 我们甚至不需要使用有意义的文本来解释软提示。相反, 它们只是LLMs中的隐藏状态, 可以通过连续优化作为模型的标准参数来学习。这种处理方法使我们能够探索超越文本的提示方法。作为另一个好处, 软提示为编码我们如何引导LLMs生成特定输出提供了密集、低维和可学习的表示。这些表示的训练和应用所需的计算成本远低于处理长硬提示所需的成本。这种方法在LLM推理应用中具有极大的实用价值, 在这些应用中, 相同的提示被反复使用。

3.3.2.1 使用较少提示调整LLMs

一个明显的将LLM适应特定任务的方法是简单地使用标记数据进行微调。这导致了一系列LLM对齐方法, 例如监督式微调, 通过将给定提示的响应与监督信号对齐来更新模型参数。微调后的LLM将任务相关信息嵌入到模型参数中, 因此这些

模型在处理与微调中相似的提示时可以正确响应。

如果我们进一步探讨这个想法，我们可以期待在微调过程中，LLMs尽可能地吸收关于任务提示的知识。因此，提示信息部分地被捕获在模型参数中，微调后的LLMs可以以更少的提示来完成任务。在这里，我们考虑一种简单的提示形式，其中只包括一个指令（用 \mathbf{c} 表示）和一个用户输入（用 \mathbf{z} 表示）。提示可以使用以下元组来表示

$$\mathbf{x} = (\mathbf{c}, \mathbf{z}) \quad (3.11)$$

给定一组提示-响应对 $\mathcal{D} = \{(\mathbf{x}, \mathbf{y})\}$ ，微调的目标是最小化在此集合上产生的总损失。一种流行的方法是相对于模型参数 θ 最小化负对数似然（即最大化对数似然）：

$$\begin{aligned} \hat{\theta} &= \arg \max_{\theta} \sum_{(\mathbf{x}, \mathbf{y}) \in \mathcal{D}} \log \Pr_{\theta}(\mathbf{y}|\mathbf{x}) \\ &= \arg \max_{\theta} \sum_{(\mathbf{x}, \mathbf{y}) \in \mathcal{D}} \log \Pr_{\theta}(\mathbf{y}|\mathbf{c}, \mathbf{z}) \end{aligned} \quad (3.12)$$

在 $\Pr_{\theta}(\cdot|\cdot)$ 是具有参数 θ ⁹的LLM预测的概率。

一般来说，每个微调示例中的指令应遵循提示设计指南，例如，一个好的指令应尽可能清晰，并详细描述任务。然而，上述方程中描述的方法并不限制指令的任何特定形式。这种灵活性使我们能够以我们想要的方式指导LLMs。考虑一个例子，我们打算指导LLMs将英语句子翻译成中文。当然，如本章前面所述，我们可以使用以下指令提示LLMs：

Translate the following sentence from English to Chinese.

如果我们想使指令更简单，我们可以将其重新表述为更简单的形式

Translate this into Chinese.

甚至，我们可以将指令定义为单个短语

Translate!

通过一定的微调努力，我们可以使LLMs遵循这些指令中的任何一个。从高效提示的角度来看，在提示中简化指令具有计算优势。例如，我们可以使用像“翻译！”这样的简单指令来执行通常需要更复杂和详细指令的任务。这可以使后续

⁹In practice, we initialize θ with the parameters obtained from pre-training, and then adjust θ moderately to ensure that the results after fine-tuning do not deviate too much from the pre-trained results.

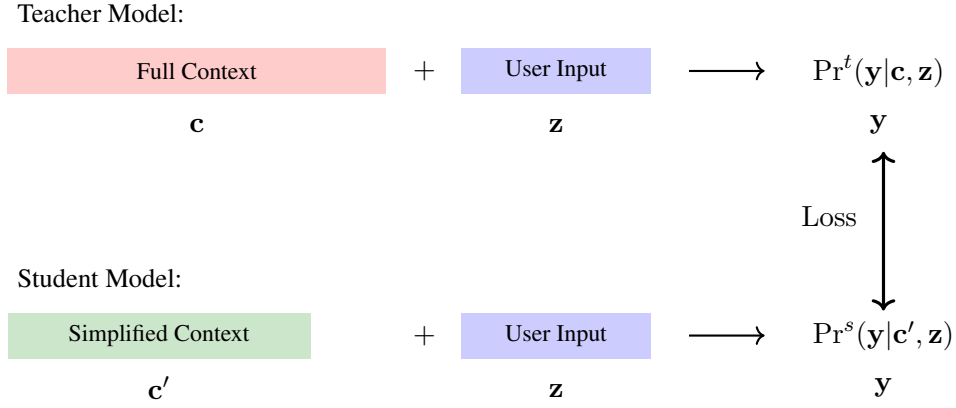


图3.4: 上下文蒸馏的说明[Snell等人, 2022]。教师模型是一个标准的LLM, 它将上下文和用户输入作为模型输入, 并产生一个预测作为模型输出。然后, 我们简化上下文(例如, 简化提示中的指令)并使用学生模型根据简化的上下文和用户输入进行预测。学生模型通过最小化两个模型产生的预测之间的损失进行训练。

推理过程中的提示要容易得多。另一方面, 使用过于简化的指令微调LLMs可能会对模型的泛化造成伤害。由于简化指令可能导致信息丢失, LLMs更有可能过度拟合微调数据, 并无法泛化到这些指令之外。在涉及复杂和简化指令的微调场景中, 这个问题更为严重, 因为可用于微调的标记数据通常有限, 而适应各种指令的成本很高。

一种将LLMs适应简化指令的替代方法是知识蒸馏。例如, 我们考虑上下文蒸馏方法[Snell等人, 2022]。该方法的目标是学习一个学生模型, 该模型可以利用来自训练良好的指令遵循教师模型的简化指令。图3.4展示了这种方法的一个说明。构建教师模型遵循标准微调过程: 我们首先收集一定量的数据, 包括指令、用户输入和正确响应, 然后我们继续使用此数据集训练预训练模型。对于构建学生模型, 我们需要构建一个新的数据集 \mathcal{D}' , 其中每个样本都是一个由指令、相应的简化指令和用户输入组成的元组, 表示为 $\mathbf{x}' = (c, c', z)$ 。通过最小化定义在教师和学生模型输出上的损失函数来执行知识蒸馏。

$$\hat{\theta} = \arg \min_{\theta} \sum_{\mathbf{x}' \in \mathcal{D}'} \text{Loss}(\Pr^t(\cdot|\cdot), \Pr_{\theta}^s(\cdot|\cdot), \mathbf{x}') \quad (3.13)$$

在 $\Pr^t(\cdot|\cdot)$ 表示预训练的教师模型, 而 $\Pr_{\theta}^s(\cdot|\cdot)$ 表示具有参数 θ 的学生模型的情况下, 为了简化符号, 我们将 $\text{Loss}(\Pr^t(\cdot|\cdot), \Pr_{\theta}^s(\cdot|\cdot), \mathbf{x})$ 简写为 Loss 。常用的损失是序列级损失, 其基本形式为:

$$\text{Loss} = \sum_{\mathbf{y}} \Pr^t(\mathbf{y}|\mathbf{c}, \mathbf{z}) \log \Pr_{\theta}^s(\mathbf{y}|\mathbf{c}', \mathbf{z}) \quad (3.14)$$

但是这个函数在计算上不可行, 因为它需要对一个指数级大的输出数量进行求和。这种方法的一个变体是使用教师模型生成的输出训练学生模型。对于每个样本, 我们使用教师模型来生成一个输出

$\hat{y} = \arg \max_y \log \Pr^t(y|c, z)$ 。然后我们考虑 \hat{y} 作为学习的目标，损失函数由下式给出

$$\text{Loss} = \log \Pr_{\theta}^s(\hat{y}|\mathbf{c}', \mathbf{z}) \quad (3.15)$$

另一种方法是，我们可以最小化两个模型输出的概率分布之间的距离[Askeel等, 2021]。例如，损失函数可以定义为两个输出分布之间的KL散度

$$\text{Loss} = \text{KL}(\mathbf{P}^t \parallel \mathbf{P}_{\theta}^s) \quad (3.16)$$

哪里

$$\mathbf{P}^t = \Pr^t(\cdot|\mathbf{c}, \mathbf{z}) \quad (3.17)$$

$$\mathbf{P}_{\theta}^s = \Pr_{\theta}^s(\cdot|\mathbf{c}', \mathbf{z}) \quad (3.18)$$

尽管我们限制了自己指令知识蒸馏方面，但这里讨论的方法是通用的。通过学习教师模型的输出，提示中的知识可以蒸馏到学生模型的参数中。因此，蒸馏模型可以被视为编码某种形式的软提示。这种方法可以应用于提示学习中的许多其他问题，例如压缩长上下文以及将软提示作为LLMs的具体组件进行学习。

3.3.2.2 学习软提示以实现参数高效微调

所有参数的更新是适应感兴趣任务中LLMs的常见方法。尽管微调被认为比预训练计算成本更低，但在实际应用中仍然成本高昂。这个问题促使开发参数高效的微调方法，旨在最小化需要更新的参数数量。

一种称为前缀微调的方法是在每个Transformer层的输入前附加一系列可训练的向量或前缀[Li和Liang, 2021]。这些前缀可以被视为软提示，作为指导模型在特定任务下行为的额外上下文。在微调期间，我们只需要学习前缀以嵌入特定任务的知识。因此，这种方法是高效的，因为它只修改模型的一小部分，而不是调整整个模型参数集。

具体来说，设深度 l 的层的输入表示为 $\mathbf{H}^l = \mathbf{h}_0^l \mathbf{h}_1^l \dots \mathbf{h}_m^l$ 。该层的输出可以表示为

$$\mathbf{H}^{l+1} = \text{Layer}(\mathbf{H}^l) \quad (3.19)$$

在前缀微调中，我们通过在序列 $\mathbf{h}_0^l \mathbf{h}_1^l \dots \mathbf{h}_m^l$ 的开头添加几个向量来扩展该序列，我们将这些向量表示为 $\mathbf{p}_0^l \mathbf{p}_1^l \dots \mathbf{p}_n^l$ 。因此， \mathbf{H}^l 可以写成以下形式

$$\mathbf{H}^l = \underbrace{\mathbf{p}_0^l \mathbf{p}_1^l \dots \mathbf{p}_n^l}_{\text{trainable}} \underbrace{\mathbf{h}_0^l \mathbf{h}_1^l \dots \mathbf{h}_m^l}_{\text{previous layer output}} \quad (3.20)$$

层的输出是最后的 $m + 1$ 表示

$$\begin{aligned}\bar{\mathbf{H}}^{l+1} &= \text{Layer}(\mathbf{H}^l)[-m-1:] \\ &= \mathbf{h}_0^{l+1} \mathbf{h}_1^{l+1} \dots \mathbf{h}_m^{l+1}\end{aligned}\quad (3.21)$$

在 $[-m-1:]$ 表示提取序列中最后 $m + 1$ 个元素的切片操作。给定 \mathbf{H}^{l+1} ，下一层的输入可以表示为与式(3.20)相同的形式：

$$\begin{aligned}\mathbf{H}^{l+1} &= \mathbf{p}_0^{l+1} \mathbf{p}_1^{l+1} \dots \mathbf{p}_n^{l+1} \bar{\mathbf{H}}^{l+1} \\ &= \mathbf{p}_0^{l+1} \mathbf{p}_1^{l+1} \dots \mathbf{p}_n^{l+1} \mathbf{h}_0^{l+1} \mathbf{h}_1^{l+1} \dots \mathbf{h}_m^{l+1}\end{aligned}\quad (3.22)$$

这里每个 $\mathbf{p}_i \in \mathbb{R}^d$ 可以被视为一个可学习的参数。在训练过程中， $\mathbf{p}_0^l \mathbf{p}_1^l \dots \mathbf{p}_n^l$ 按照常规进行训练，原始 Transformer 模型的参数保持不变。

图3.5展示了翻译任务中前缀微调的示意图。在此，仅通过接收输出（即中文翻译）的错误梯度来更新前缀向量 \mathbf{p}_0^l 和 \mathbf{p}_1^l 。通过调整这些向量以适应翻译任务，模型相应地做出调整。这使得 \mathbf{p}_0^l 和 \mathbf{p}_1^l 作为提示，激活LLM执行任务，无需像“将以下句子从英语翻译成中文”这样的明确输入提示。在测试时，我们将优化的 \mathbf{p}_0^l 和 \mathbf{p}_1^l 添加到层中，然后LLM将翻译输入句子。请注意，前缀微调引入了额外的 $L \times n \times d$ 参数，其中 L 是层数， n 是前缀的数量， d 是每个前缀的维度。然而，这个数字与LLM中的总参数数量相比要小得多，这使得微调过程非常高效。

虽然前缀微调很简单，但仍需要对LLMs进行修改。另一种方法是将软提示与LLMs分离，这样我们可以保留原始模型架构，使其在不同任务部署时更高效，无需调整核心模型。一种这样的方法是提示微调[Lester等人, 2021]。与前缀微调类似，提示微调包含可训练的向量，以便LLMs可以通过调整这些向量来适应给定的任务。然而，提示微调的不同之处在于它只修改嵌入层。

回忆在LLMs中，每个输入标记 z_i 都由嵌入 \mathbf{e}_i 表示。这些嵌入通常通过标记嵌入模型学习得到，然后作为LLMs的真实输入使用，取代了符号表示的标记。在提示微调中，在标记嵌入序列的开头添加了多个伪嵌入 $\mathbf{p}_0 \dots \mathbf{p}_n$ 。因此，LLMs的实际输入可以表示为

$$\underbrace{\mathbf{p}_0 \mathbf{p}_1 \dots \mathbf{p}_n}_{\text{trainable}} \quad \underbrace{\mathbf{e}_0 \mathbf{e}_1 \dots \mathbf{e}_m}_{\text{token embeddings}}$$

注意，伪嵌入不需要对应自然语言中的任何标记。相反，这些嵌入可以被视为“软提示嵌入”，用于条件化LLMs。通过在特定任务数据上训练软提示嵌入，它们学会自适应地与标记嵌入 $\mathbf{e}_0 \dots \mathbf{e}_m$ 交互，并指导LLMs的行为。由于提示调整不会改变预训练LLMs的底层参数，因此被认为是一种轻量级且高效的超参数调整方法，在提高特定任务性能的同时保持其泛化能力。参见图3.6以了解提示调整的说明。

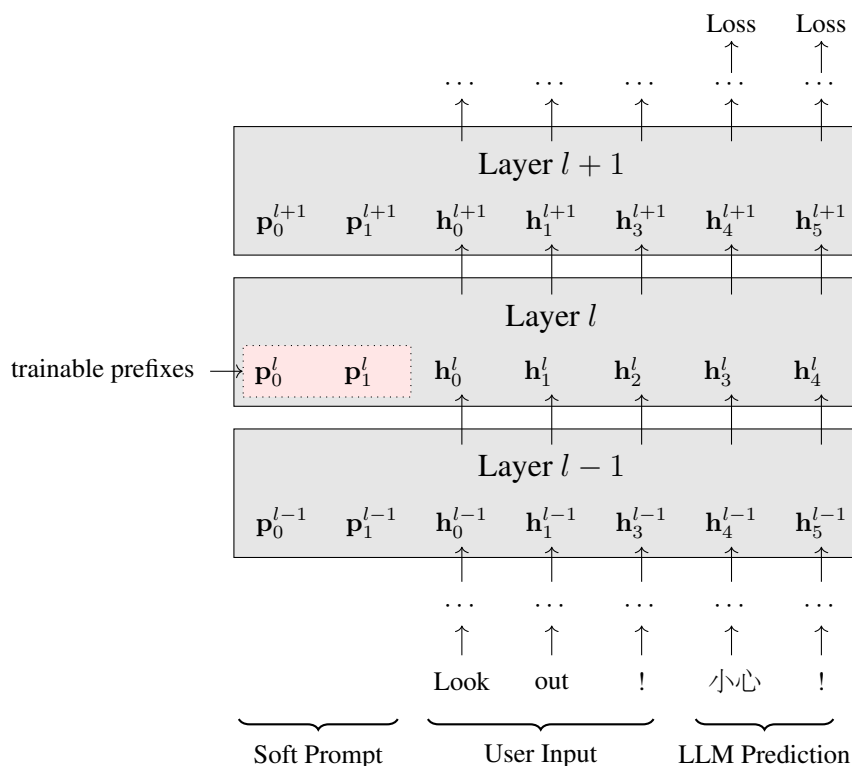
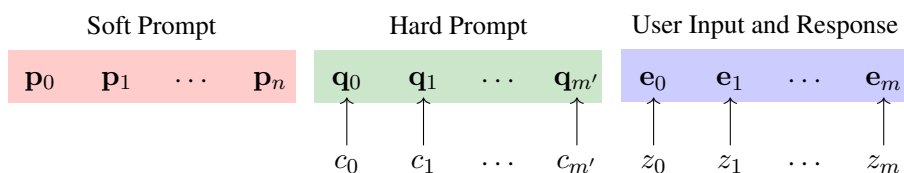


图3.5: 翻译任务前缀微调的说明 (小心! \rightarrow 小心!)。对于每一层, 我们在开头添加两个前缀 p_0^l 和 p_1^l 。LLM 被训练以最小化给定输入的预测损失。在这个过程中, 只有前缀被优化, 其余参数保持不变。因此, 该模型可以非常高效地适应给定的任务。在推理时间, LLM 与优化后的前缀一起工作, 可以在不需要显式硬提示的情况下执行任务。

由于 $p_0 p_1 \dots p_n$ 本身是一个序列, 我们可以使用序列模型来更好地表示它。例如, 一个 Transformer 模型可以编码这个序列, 然后将得到的表示用作 LLM 的输入。换句话说, 我们可以开发一个额外的模型来编码软提示。另一种改进提示的方法是结合软提示和硬提示, 从而利用两种类型 [Liu 等人, 2023b]。在嵌入序列中, 我们可以安排或穿插这些提示。这将导致不同的提示模式。例如, 一个使用两种类型提示的简单模式是



$c_0 \dots c_{m'}$ 表示硬提示, $q_0 \dots q_{m'}$ 表示相应的嵌入序列。

这里我们考虑了在 LLMs 中插入软提示的方法。但我们跳过了训练这些软提示的细节, 并假设读者熟悉标准的监督学习过程, 即最大化给定模型的正确模型输出的似然性。

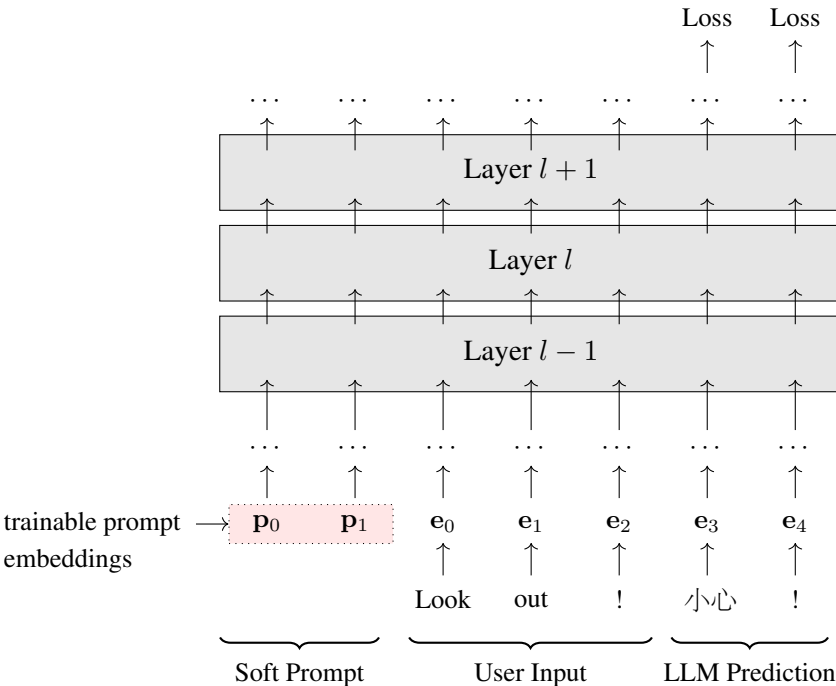


图3.6: 翻译任务中提示调整的说明(小心! → 小心!)。与使用固定的文本提示不同, 软提示是可学习的嵌入, 它们被添加到嵌入序列的开头。在微调期间, 仅优化这些提示嵌入以有效地调整LLM以适应给定的任务。一旦优化, 提示嵌入就用于指导LLM在接收到新数据时执行任务。

输入。事实上, 学习软提示可能与LLM微调中的许多问题相关。例如, 如果我们将其视为上下文压缩问题, 我们可以应用之前描述的知识蒸馏方法。在Mu等人[2024]的研究中, 提示被压缩并表示为几个伪标记, 这些伪标记被附加到每个输入序列中。这些伪标记的嵌入被优化以模仿标准提示模型的预测。换句话说, 提示知识是从教师模型蒸馏到伪标记中的。

广义上讲, 许多参数高效的微调方法可以被视为学习某种形式的软提示[Lialin等, 2023]。当我们对一个LLM的部分进行微调以完成一项任务时, 这个过程本质上可以看作是将与任务相关的提示信息注入到该特定模型部分。参数高效微调的另一种广泛使用的方法是在现有模型层之间添加一个适配器层。这种方法允许我们仅对特定任务的适配器层进行微调, 而不会改变底层架构或重新训练整个模型。从这个意义上讲, 适配器层可以被视为编码提示和任务相关信息的软提示, 并与原始LLM交互以帮助其适应。总之, 图3.7展示了在LLM中使用软提示的不同方法的比较。

3.3.2.3 使用压缩学习软提示

另一种学习软提示的方法是从压缩的角度出发。作为一个简单的例子, 考虑使用连续表示来近似长上下文的问题 [Wingate 等人, 2022]。假设我们有一个用户输入 z 及其上下文 c (例如长指令和演示)。现在我们要开发上下文的压缩表示, 表示为 $\{v^*\}$ 。

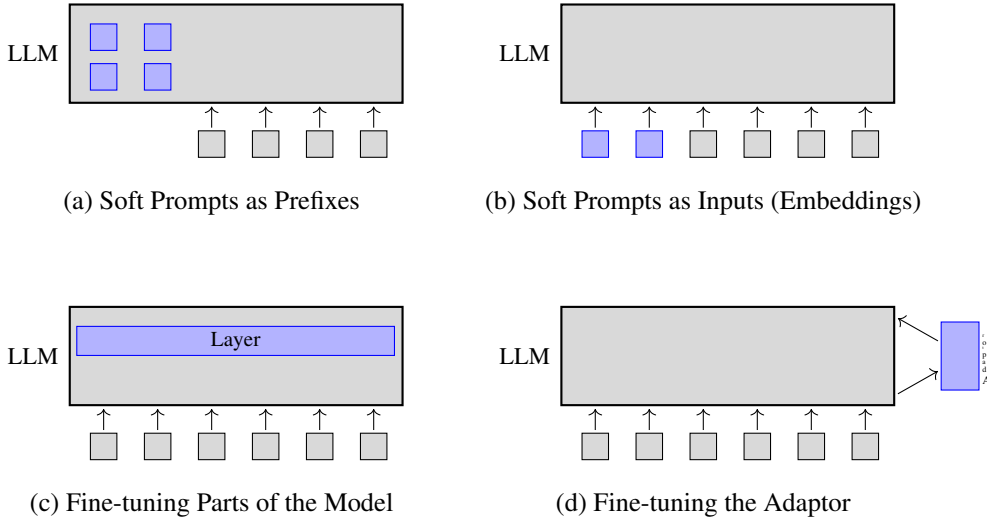


图3.7: 在LLM中使用软提示的说明。可调软提示以蓝色显示，在微调期间参数固定的组件以灰色显示。在子图(a)中，软提示是附加到LLM每一层的词缀。在子图(b)中，软提示被用作LLM的输入嵌入。在子图(c)和(d)中，软提示被广泛视为模型组件，用于任务适应的微调。

通过 σ ，使得基于 z 和 σ 的预测尽可能接近基于 z 和 c 的预测。此目标可以表示为

$$\hat{\sigma} = \arg \min_{\sigma} s(\hat{y}, \hat{y}_{\sigma}) \quad (3.23)$$

在 $\hat{y} = \arg \max_y \Pr(y|c, z)$ 和 $\hat{y}_{\sigma} = \arg \max_{y_{\sigma}} \Pr(y_{\sigma}|\sigma, z)$ 分别表示在完整上下文和压缩上下文下的 LLM 预测，其中函数 $s(\cdot, \cdot)$ 通常表示损失或相似度度量，旨在最小化两个上下文表示之间的预测差异。

一个实现这一目标的一般框架是知识蒸馏，其中 \hat{y} 和 \hat{y}_{σ} 可以分别看作教师模型和学生模型的预测。这种形式化将我们的讨论与之前讨论的上下文蒸馏问题联系起来。可以通过类比方程(3.15)和(3.16)获得训练目标。例如，一个简单的训练目标如下：

$$\hat{\sigma} = \arg \max_{\sigma} \log \Pr(\hat{y}|\sigma, z) \quad (3.24)$$

另一种方法是，我们可以最小化输出分布之间的KL散度，给出

$$\hat{\sigma} = \arg \min_{\sigma} \text{KL}(\Pr(\cdot|c, z) \parallel \Pr(\cdot|\sigma, z)) \quad (3.25)$$

与公式 (3.15) 和 (3.16) 中的模型不同之处在于，这里将压缩的上下文表示为实值向量（称为提示嵌入），而不是表示为正常标记。通过应用上述方法，我们将上下文从标记序列 c 蒸馏到嵌入 σ 中。请注意，教师模型 $\Pr(\cdot|c, z)$ 和学生模型 $\Pr(\cdot|\sigma, z)$ 可能不共享相同的架构或模型设置。在实践中，我们通常希望教师模型是

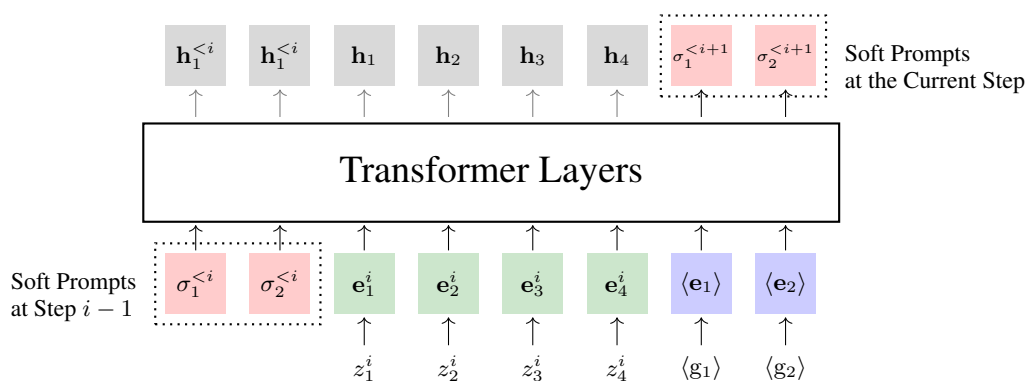


图3.8: 将上下文段压缩为软提示 ($\kappa = 2$ 和 $m_i = 4$) 的示意图。LLM的输入包括前一步的软提示 ($\sigma_1^{<i}$ 和 $\sigma_2^{<i}$)，段落的标记 (z_1, z_2, z_3 , 和 z_4)，以及摘要标记 ($\langle g_1 \rangle$ 和 $\langle g_2 \rangle$)。据此，LLM按常规操作。然后我们提取对应于摘要标记的最后Transformer层的输出。这些输出可以视为累积到该段之前的软提示。

更强，而学生模型应该更小、更高效。

当将完整上下文压缩为连续表示是学习软提示的直接方法时，它需要一个能够处理长输入序列的教师模型。然而，在许多情况下，上下文如此之长，以至于应用一个LLM过于昂贵或不可行。对长输入序列进行建模可以归入长上下文LLM的高效方法广泛家族。已经开发了许多技术来解决此问题。例如，可以在推理过程中使用固定大小的KV缓存来存储每一步的过去信息。本书中详细讨论了高效的Transformer架构和长上下文LLM。对于这些主题的更详细讨论，感兴趣的读者可以参考第2章。

也存在专门设计用于将长上下文压缩为软提示的方法。在这里，我们以Chevalier等人[2023]的方法为例。基本思想是通过在上下文序列中累积固定大小的上下文表示来逐步学习软提示。给定一个长上下文，我们首先将其划分为若干个段 z^1, \dots, z^K 。然后我们按顺序处理这些段，每次生成到目前为止已处理的上下文的表示，记为 $\sigma^{<i+1}$ 。为此，引入了一些摘要标记 $\langle g_1 \rangle, \dots, \langle g_\kappa \rangle$ 。在每一步中，我们取一个段 $z^i = z_1^i \dots z_{m_i}^i$ ，以及先前的上下文表示 $\sigma^{<i}$ 和摘要标记 $\langle g_1 \rangle, \dots, \langle g_\kappa \rangle$ 作为输入，并使用LLM在最后一个Transformer层产生相应的隐藏表示序列。这个过程的一个示例如图3.8所示。

这里 $\sigma^{<i}$ 实质上是一个记忆。该模型以 RNN 的方式运行。每次我们取一个片段，并通过编码先前的记忆状态和片段来更新这个记忆。因此，在最后一个片段产生的 $\sigma^{<i}$ 是整个上下文序列的表示。用于学习这些表示的 Transformer 模型可以是一个标准的 LLM，但我们需要对其进行微调以适应这个上下文表示任务。

请注意，在这里我们仅仅将 *prompt* 和 *context* 视为相似项，尽管它们并不相同。尽管我们对概念 *prompt* 有点“误用”，但我们通常可以将其视为一种上下文。从这个角度来看，这里讨论的方法可以应用于一般的文本压缩问题。

3.3.3 提示长度缩减

虽然软提示提供密集的隐藏表示，但它们不可直接解释。缺乏可解释性可能成为用户试图理解其输入如何影响LLM输出的重大障碍。此外，尽管软提示在微调和部署方面效率高，但它们不够灵活，不允许在不进行大量微调或修改的情况下轻松调整。这种不灵活性可能会限制它们在需要频繁更改提示的动态环境中的效用。

一种开发高效提示的替代方法是简化用于提示的文本。例如，下面是一个用于回答医疗和财务问题的提示。

任务涉及开发一种能够理解和回应用户在各个领域查询的语言模型，尤其侧重于医疗和金融领域。考虑到潜在查询的广泛范围，从医疗诊断的细节到金融法规的细微差别，该模型必须确保全面理解和准确回应。

问题：

什么是使用人工智能诊断心血管疾病的最佳实践？

我们可以通过删除不重要的部分来简化任务描述。

The task involves developing a language model capable of understanding and responding to user inquiries across various domains, with a particular emphasis on healthcare and finance. Considering the broad range of potential queries, from the specifics of medical diagnoses to the nuances of financial regulations, The model must ensure a comprehensive understanding and accurate responses.

我们也可以将其改写为更简短的文本。

The task involves developing a language model focused on healthcare and finance, capable of understanding and accurately responding to a wide range of user inquiries.

这个问题可以被视为一个经典的NLP问题——文本简化。因此，所使用的方法可以是一般的，而不仅限于简化提示的问题。实现这一点有许多方法。一种简单的方法是定义一些启发式规则，并识别可以消除而不丢失关键信息的冗余词汇。例如，我们可以从每个标记对整体意义的贡献来检查序列中的每个标记，并移除那些提供最小价值的标记[Li等人，2023c；Jiang等人，2023b]。另一种方法是将问题框架化为一个序列到序列的任务。对于文本简化的标记数据，我们可以训练一个编码器-解码器模型，将每个输入文本转换为简化形式。此外，鉴于许多

LLMs 已经过微调和校准以执行文本简化任务，使用这些模型简化提示非常简单。例如，我们可以提示一个 LLM 在某些约束下简化文本，例如限制简化文本的长度。

3.4 摘要

在这一章中，我们讨论了与LLM提示相关的一系列问题。我们的讨论主要集中在两个方面：

- 如何设计基本提示以引导LLMs的预测并优化这些提示以实现更有效和高效的解决问题？
- 如何自动化提示的设计和表示？

解决方案涉及一般提示设计以及更高级的技术，如CoT和提示学习，这些在最近的研究中已被广泛探讨。

在NLP中，提示可以被视为与LLMs共同发展的技术，从某种意义上说，它为这些模型在广泛的实际问题领域中的实际应用打开了大门。事实上，如果我们稍微扩展提示的概念，它可以追溯到机器学习和NLP的早期。例如，许多NLP系统使用手工制作的特征和模板来“提示”特定任务。想象一下开发一个特征来指示文本是正式的还是非正式的。我们可以将这个特征输入到机器翻译系统中，以根据输入文本的类型来条件化翻译。

现代提示概念的广泛应用始于NLP领域大型预训练模型的兴起。最初，这些模型，如BERT，主要通过微调来适应特定的下游任务。然而，研究人员很快发现，通过设计特定的“提示”——在输入中添加某些词语或句子——可以触发模型对特定任务做出响应，而无需进行广泛的微调。这促使NLP社区开发并应用通用的基础模型，这些模型可以通过提示来解决各种任务，而无需改变底层架构和预训练过程。

提示方法最初在较小的模型上进行了实验，后来在像GPT-3这样的大型模型上展示了令人印象深刻的性能，能够在各种任务中根据简单的提示生成高质量的文本。随着提示技术的演变，提示工程成为了一个关键的研究领域。正如本章所讨论的，它广泛涉及设计有效的提示以最大化模型性能，包括手工制作和自动生成的提示。最近的研究探讨了如何通过少样本学习、零样本学习和CoT推理等技术来提高提示的有效性，使LLMs能够在广泛的场景中有效地工作。关于提示的一般讨论可以非常广泛，我们无法在本章中涵盖所有细节。对于更高级的提示技术，读者可以参考最近的综述。包括上下文学习[Li, 2023; Dong等, 2022], CoT[Chu等, 2023; Yu等, 2023; Zhang等, 2023a], 高效提示[Chang等, 2024]和通用提示工程[Liu等, 2023c; Chen等, 2023a]。

注意，尽管我们理想情况下希望在不调整模型架构和参数的情况下开发通用提示方法，但提示的结果通常在很大程度上取决于给定LLMs的质量和大小。对于更强的模型，如商业化的在线LLMs，简单的提示可能足以指导这些模型正确执行任务。在这种情况下，提示工程相对容易，尽管我们仍然需要付出一定的努力来使LLMs正常工作。相比之下，如果LLMs不够强大，我们可能需要仔细设计提示以实现预期的结果。在许多情况下，微调仍然是必要的，以使模型适应复杂的提示策略。

CHAPTER 4

对齐

对齐在NLP中不是一个新概念，但其含义在不同领域和不同时间有所变化。在传统NLP中，术语*alignment*通常指将两个集合中的对应元素链接起来的任务，例如将中文句子和英文句子中的单词对齐。随着LLMs在NLP研究中的重要性日益增加，这个术语被更广泛地用来指代将模型输出与人类期望对齐。对齐解决的问题可能是模型的输出可能无法与用户意图的具体目标或上下文相一致。例如，预训练的LLMs可能无法遵循用户指令，因为它们没有被训练去这样做。另一个例子是，LLMs可能会生成有害内容或延续其训练数据中固有的偏见。这为确保LLM输出不仅准确相关，而且符合伦理和非歧视性提出了新的挑战。

简单预训练LLMs可能导致各种对齐问题。我们的最终目标是解决或减轻所有这些问题，以确保LLMs既准确又安全。这里有一个有趣的问题：由于大型语言模型是在大量数据上训练的，我们有理由相信，如果我们有足够的数据，涵盖各种任务并与人类偏好一致，预训练可以使LLMs足够准确和安全，甚至可能消除对齐的需要。然而，现实是几乎不可能收集到涵盖所有任务或充分代表人类偏好的数据。这使得仅通过预训练来实现模型对齐变得困难，或者至少，在这个阶段，对齐仍然是LLMs发展中非常必要和关键的一步。

在这一章中，我们将关注LLMs的对齐方法。我们将首先讨论一般的对齐任务。然后，我们将考虑两种广泛使用的方法，分别称为指令对齐和人类偏好对齐。前者采用监督微调技术，引导LLMs生成紧密遵循用户指令的输出。另一方面，后者通常依赖于强化学习技术，其中LLMs根据人类的反馈进行训练。虽然这些方法由不同的目标驱动，但它们通常一起使用，以开发良好对齐的LLMs。

4.1 LLM 对齐概述

对齐可以通过多种不同的方式实现。由于这个问题本身复杂且多面，需要结合技术考虑，我们需要为LLM对齐采用不同的方法。在这里，我们考虑了三种广泛使用的LLM对齐方法。

第一种方法是使用标记数据微调LLMs。这种方法很简单，因为它只是将预训练LLM的现有训练扩展到适应特定任务。这种方法的例子是监督微调（SFT），其中LLM在包含特定任务指令及其预期输出的数据集上进行进一步训练。与原始训练集相比，SFT数据集通常要小得多，但这些数据非常专业。SFT的结果是LLM可以学习根据用户指令执行任务。例如，通过使用一组问答对微调LLM，模型可以回答特定问题，即使这些问题没有直接包含在SFT数据集中。这种方法特别证明

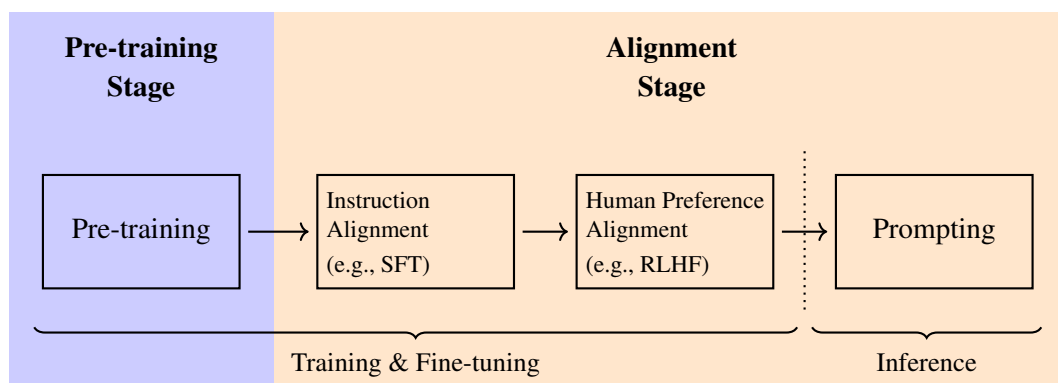


图4.1：开发LLM的预训练然后对齐方法的示意图。在预训练阶段，我们使用下一个标记预测在大量数据上训练一个LLM。然后，在对齐阶段，我们将LLM与用户指令、意图和偏好对齐。这包括指令对齐、人类偏好对齐和提示。

当描述输入输出关系相对容易且数据标注直接时很有用。

第二种方法是使用奖励模型微调LLMs。对齐中的一个困难是人类的价值观和期望复杂且难以描述。在许多情况下，即使是人类自己，表达什么是道德上正确或文化上适当的也可能具有挑战性。因此，收集或注释微调数据并不像SFT那样简单。此外，对齐LLM不仅仅是拟合数据，换句话说，人类标注的有限样本通常不足以全面描述这些行为。我们真正需要的是教会模型如何确定哪些输出更符合人类偏好，例如，我们不仅希望输出在技术上准确，而且要与人类的期望和价值观相一致。一个想法是开发一个类似于人类专家的奖励模型。这个奖励模型会在LLM生成与人类偏好更接近的响应时对其进行奖励，就像老师向学生提供反馈一样。为了获得这样的奖励模型，我们可以从人类偏好数据中训练一个评分函数。然后，训练好的奖励模型被用作指南来调整和细化LLM。这把LLM对齐任务框架化为强化学习任务。结果方法，如从人类反馈中进行强化学习（RLHF），已被证明在适应LLMs以遵循人类行为和社会规范的细微差别方面特别成功。

第三种方法是在推理期间而不是在训练或微调期间进行对齐。从这个角度来看，在LLM中的提示也可以被视为一种对齐形式，但它不涉及训练或微调。因此，我们可以以最低的成本动态适应LLM以处理各种任务。在推理时间进行对齐的另一种方法是重新评分LLM的输出。例如，我们可以开发一个评分系统来模拟人类对LLM输出的反馈（如奖励模型）并优先考虑那些获得更多积极反馈的输出。

上述三种方法通常在预训练完成后按顺序使用：我们首先执行SFT，然后进行RLHF，最后在推理过程中以某种方式提示LLM。这大致将LLM的开发分为两个阶段——预训练阶段和对齐阶段。图4.1展示了这一过程。由于提示技术已在上一章中进行了深入讨论，我们将在本章的剩余部分专注于基于微调的对齐方法。

4.2 指令对齐

LLMs的一个特点是它们可以遵循用户提供的提示来执行各种任务。在许多应用中，提示由一个简单的指令和用户输入组成，我们希望LLM遵循这个指令正确地完成任务。LLMs的这种能力也称为指令遵循能力。例如，以下是一个提示，我们希望LLM提取要点并为长篇文章提供一个简洁的摘要。

说明总结以下文本为三句话。输入夏令时（DST） - 在夏季将时钟向前移动一小时的过程 - 于1916年在德国开始...输出

此任务要求LLM理解指令“用三句话总结此文本”并相应地进行总结。然而，LLM通常被训练用于预测下一个标记，而不是生成遵循指令的输出。将预训练的LLM应用于上述示例可能会使模型继续撰写输入文章而不是总结要点。指令对齐（或指令微调）的目的是调整LLM以准确响应用户指令和意图。本节其余部分将讨论与指令对齐相关的一些问题，包括微调LLM以遵循指令、生成或收集指令数据以及指令对齐的泛化。

4.2.1 监督微调

一种将LLMs适应指令的简单方法是通过使用标注的输入-输出对来微调这些模型[Ouyang等人，2022；Wei等人，2022a]。与标准语言模型训练不同，我们在这里不希望最大化生成完整序列的概率，而是最大化给定其前缀生成剩余序列的概率。这种方法使得指令微调与预训练略有不同。SFT数据是一组这样的输入-输出对（用*S*表示），其中每个输出是对应输入指令的正确响应。例如，下面是一个SFT数据集

x (instruction + user input)	y (output)
Summarize the following article. Article: In recent years, solar energy has seen unprecedented growth, becoming the fastest-growing ...	{*summary*}
Extract the main financial figures from the following earnings report. Report: The company reported a revenue of \$10 million in the first quarter with a profit margin of 15% ...	Revenue: \$10 million, Profit Margin: 15%
Classify the following email as spam or not spam. Text: Congratulations! You’ve won a \$500 gift card. Click here to claim now.	Spam
Provide a solution to the following technical issue. Issue: my computer is running slow and often freezes.	First, check for ...

在指令被突出显示的地方。此数据集包含多个不同NLP问题的指令及其对应输出，因此我们可以微调一个LLM以同时处理多个任务。

设 $\mathbf{x} = x_0 \dots x_m$ 为一个输入序列（例如，指令 + 用户输入）和 $\mathbf{y} = y_1 \dots y_n$ 为相应的输出序列。在 SFT 中，我们旨在最大化给定输入 \mathbf{x} 的输出 \mathbf{y} 的概率。考虑一个具有预训练参数 $\hat{\theta}$ 的 LLM。微调目标可以表示为：

$$\tilde{\theta} = \arg \max_{\hat{\theta}^+} \sum_{(\mathbf{x}, \mathbf{y}) \in \mathcal{D}} \log \Pr_{\hat{\theta}^+}(\mathbf{y}|\mathbf{x}) \quad (4.1)$$

$\tilde{\theta}$ 表示通过微调优化的参数， $\hat{\theta}^+$ 代表对 $\hat{\theta}$ 的调整。在这里，我们将省略上标 +，并使用 θ 来表示 $\hat{\theta}^+$ 以保持符号简洁。但读者应记住，微调是从预训练参数而不是随机初始化的参数开始的。

目标函数 $\log \Pr_{\theta}(y_i|\mathbf{x}, y_{<i})$ 通过对 \mathbf{y} 中在输入 \mathbf{x} 和所有先前标记 $y_{<i}$ 条件下的标记的 log-概率进行求和来计算：

$$\log \Pr_{\theta}(\mathbf{y}|\mathbf{x}) = \sum_{i=1}^n \log \Pr_{\theta}(y_i|\mathbf{x}, \mathbf{y}_{<i}) \quad (4.2)$$

这个公式等同于最小化交叉熵损失。

注意，最小化条件对数概率 $\log \Pr_{\theta}(\mathbf{y}|\mathbf{x})$ 不是一个标准的语言模型训练问题。如果我们把 \mathbf{x} 和 \mathbf{y} 连接成一个单一序列，基于联合对数概率 $\log \Pr_{\theta}(\mathbf{x}, \mathbf{y})$ 的更一般形式的语言模型，即我们在序列 $\text{seq}_{\mathbf{x}, \mathbf{y}} = [\mathbf{x}, \mathbf{y}]$ 的所有标记上最小化损失。我们可以使用链式法则来写出这个序列的概率

$$\begin{aligned} \log \Pr_{\theta}(\text{seq}_{\mathbf{x}, \mathbf{y}}) &= \log \Pr_{\theta}(\mathbf{x}, \mathbf{y}) \\ &= \underbrace{\log \Pr_{\theta}(\mathbf{x})}_{\text{set to 0}} + \underbrace{\log \Pr_{\theta}(\mathbf{y}|\mathbf{x})}_{\text{loss computation}} \end{aligned} \quad (4.3)$$

方程右侧有两个项。我们可以简单地设置第一个项 $\log \Pr_{\theta}(\mathbf{x})$ 为 0，仅关注第二个项 $\log \Pr_{\theta}(\mathbf{y}|\mathbf{x})$ 进行损失计算。因此，训练可以使用标准的 LLMs 实现。对于序列 $\text{seq}_{\mathbf{x}, \mathbf{y}}$ ，我们首先像往常一样进行前向传播。然后，在反向传播过程中，我们强制将对应于 \mathbf{x} 的损失设置为 0。图 4.2 展示了此过程的示意图。

通过将 $\log \Pr_{\theta}(\text{seq}_{\mathbf{x}, \mathbf{y}})$ 作为目标函数，我们可以使用语言模型训练的常规形式来描述 SFT：

$$\tilde{\theta} = \arg \max_{\theta} \sum_{(\mathbf{x}, \mathbf{y}) \in \mathcal{D}} \log \Pr_{\theta}(\text{seq}_{\mathbf{x}, \mathbf{y}}) \quad (4.4)$$

上述考虑的问题本质上是一个单轮预测问题，其中 LLM 根据单个输入生成响应，没有任何进一步的交互或用户反馈。输入一次性处理，输出生成。这在

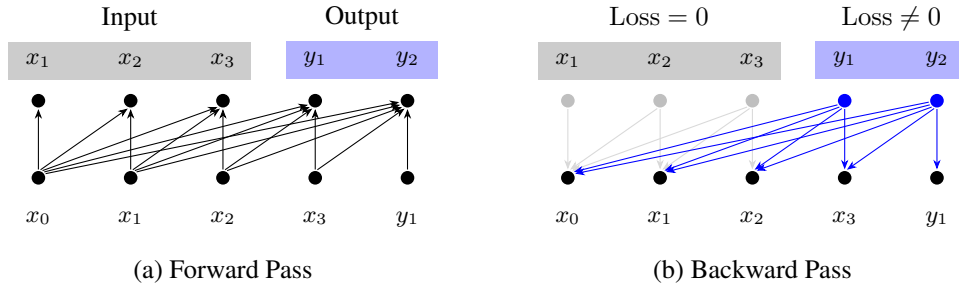


图4.2: LLMs的监督微调示意图。我们将输入和输出连接成一个单一序列。在正向传播过程中, 我们像往常一样运行LLM。在反向传播过程中, 我们只计算输出部分的损失, 并将输入部分的损失简单地设置为0。

场景中只问一个问题, 并给出一个答案, 没有后续问题或澄清。然而, 在实践中, 我们有时必须处理多轮预测问题, 例如, 一个大型语言模型在多个回合中进行对话。在这种情况下, LLM不仅根据初始输入生成响应, 还结合后续输入, 这些输入可能会完善或扩展早期交互。例如, 我们可以使用LLM作为医疗助手聊天机器人, 并与用户进行对话, 如下所示

用户 我最近感觉非常累。聊天机器人 我很遗憾听到这个消息。除了感到疲劳, 你有没有注意到其他症状? 用户 是的, 我也经常感到头痛。聊天机器人 这些症状持续了多久? 用户 大概一周了。聊天机器人 可能最好咨询一下医疗保健专业人士。你想让我帮你安排预约吗? 用户 是的, 请。可以在下班后吗? 聊天机器人 当然, 我可以安排。下周三和周四下午5点后有可用的时段。哪一天对你更合适? ...

在这个任务中, 有多个对话回合, 每个回合都涉及根据用户的请求或问题以及对话历史生成一个响应。假设我们有 K 个对话回合, 用 $\{x^1, y^1, x^2, y^2, \dots, x^K, y^K\}$ 表示。这里 x^k 和 y^k 分别表示每个回合 k 的用户请求和响应。生成响应的对数概率可以写成 $\log \Pr_{\theta}(y^k | x^1, y^1, \dots, x^k)$ 。我们的目标然后是最大化这些对数概率的总和

$$\tilde{\theta} = \arg \max_{\theta} \sum_{k=1}^K \log \Pr_{\theta}(y^k | x^1, y^1, \dots, x^k) \quad (4.5)$$

一个直接的实现方法涉及计算每个 k 的条件概率。然而，这需要运行 LLM K 次，每次都增加对话历史以进行预测。一个更有效的方法是在 LLM 单次运行中对所有响应进行损失计算。为此，我们将对话表示为序列 $\text{seq}_{\mathbf{x}^1, \mathbf{y}^1, \dots, \mathbf{x}^K, \mathbf{y}^K} = [\mathbf{x}^1, \mathbf{y}^1, \dots, \mathbf{x}^K, \mathbf{y}^K]$ （或简称为 seq ）。此序列的对数概率由以下公式给出

$$\begin{aligned}
 \log \Pr_{\theta}(\text{seq}) &= \log \Pr_{\theta}(\mathbf{x}^1, \mathbf{y}^1, \dots, \mathbf{x}^K, \mathbf{y}^K) \\
 &= \underbrace{\log \Pr_{\theta}(\mathbf{x}^1)}_{\text{set to 0}} + \underbrace{\log \Pr_{\theta}(\mathbf{y}^1 | \mathbf{x}^1)}_{\text{loss computation}} + \dots + \\
 &\quad \underbrace{\log \Pr_{\theta}(\mathbf{x}^K | \mathbf{x}^1, \mathbf{y}^1, \dots, \mathbf{y}^{K-1})}_{\text{set to 0}} + \\
 &\quad \underbrace{\log \Pr_{\theta}(\mathbf{y}^K | \mathbf{x}^1, \mathbf{y}^1, \dots, \mathbf{x}^K)}_{\text{loss computation}}
 \end{aligned} \tag{4.6}$$

这里的关键是，我们忽略生成用户输入的损失，如图4.3所示。因此，我们只计算给定对话历史生成响应的概率，换句话说，等式(4.6)右侧的值实际上等于等式(4.5)右侧的值。与等式(4.4)一样，可以通过最大化训练数据集 \mathcal{D} 上的对数似然来训练这个多轮预测模型：

$$\tilde{\theta} = \arg \max_{\theta} \sum_{\text{seq} \in \mathcal{D}} \log \Pr_{\theta}(\text{seq}) \tag{4.7}$$

在实现上述SFT方法看似简单，因为它们在本质上与常规语言模型训练相同，但在实践中仍有一些问题需要考虑。例如，

- SFT需要标记数据。这使得SFT与预训练有很大不同，预训练使用原始文本作为训练数据，且易于获取。与其他监督机器学习问题一样，SFT中的数据标注和选择并非简单任务。一般来说，我们希望开发出既大量又高质量的SFT数据，并且这些数据应与LLM将要执行的任务高度相关。另一方面，需要用较少的数据微调LLM以最小化计算和数据构建成本。通常，LLM的质量高度依赖于SFT中使用的数据。因此，此类数据必须谨慎开发并检查。正如我们将在后面的子节中看到的那样，通过更先进的数据构建技术，SFT可以更加高效和有效。
- SFT对于LLMs来说仍然计算成本高昂，因为它们规模庞大。因此，维护和更新此类模型需要大量资源。例如，在LLM中对数十亿参数应用梯度更新需要大量的计算能力和内存。这通常需要高性能计算环境，其运营成本高昂。为了解决这些挑战，已经探索了各种优化策略，例如剪枝、量化和使用更高效的训练算法。特别是，对参数高效的微调方法产生了极大的兴趣，这些方法旨在在不需要大量计算资源的情况下保持最先进的性能。在第3章中，我们看到了应用如软提示等技术可以

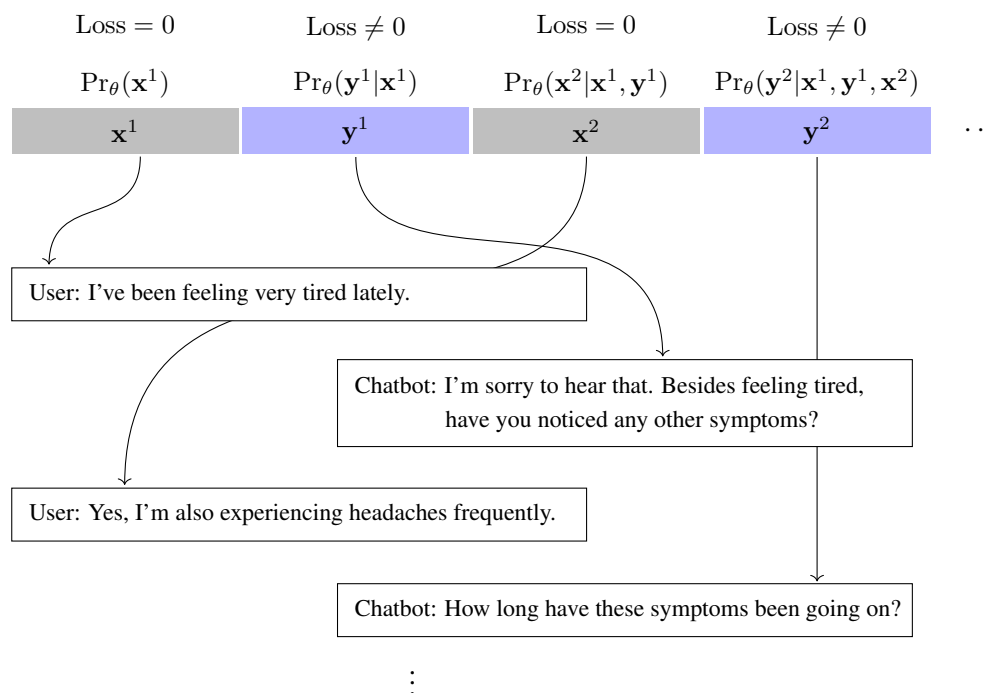


图4.3: 监督微调对话模型的示意图。在此, LLM充当聊天机器人, 根据对话历史对每个请求进行响应。对话通过用户和聊天机器人之间的交替进行。在SFT中, 我们将整个对话视为一个序列, 就像在标准LLM中一样, 但只计算LLM的响应损失。

使微调过程更高效。关于参数高效方法的进一步讨论, 读者可以参考相关论文[Houlsby等人, 2019; Hu等人, 2022; Han等人, 2024]。

- SFT可以被视为在预训练之后的后训练步骤。它是一个独立的训练阶段, 旨在保留初始预训练的优势, 同时融入新的调整。这看起来可能有些矛盾, 因为使用更多数据更新预训练的LLM可能会使模型忘记一些先前的知识。想象一下, 我们拥有大量SFT数据并对LLM进行广泛微调的场景。在这种情况下, LLM可能会对数据进行过拟合, 这反过来可能会降低泛化性能或导致灾难性遗忘。缓解这一问题的常见策略是采用正则化和早期停止技术。另一种实际的方法是使用较小的学习率来温和地调整LLM的权重。此外, 使用来自不同来源和问题领域的数据进行微调也可能有益。然而, 在实践中, SFT步骤通常需要仔细检查, 并且需要大量的工程和实验努力来优化。

4.2.2 精细调优数据采集

微调数据非常重要, 因此最近在LLM领域的研究主要集中在开发各种用于指令微调的数据集。与机器学习中的大多数工作一样, 数据获取通常有两种方法——手动数据生成和自动数据生成。

4.2.2.1 手动生成数据

一种直接的方法是招募人工标注员为感兴趣的任務创建输入-输出对。与传统的NLP中的数据标注不同，例如文本分类，标注员只需根据指南将标签分配给收集到的文本，为LLMs创建微调数据需要更多步骤和努力，因此更具挑战性。假设我们想要获取英语到中文的机器翻译任务的微调数据。第一步是编写一个提示模板来描述任务并清晰地格式化问题。例如，

指令 将文本从英语翻译成中文。用户输入 $\{*\text{text}*\}$
输出 $\{*\text{translation}*\}$

然后，我们收集源文本和目标文本的成对（即中文文本及其对应的翻译），并将变量 $\{*\text{text}*\}$ 和 $\{*\text{translation}*\}$ 替换以生成微调样本。例如，给定一对英文和中文句子

How's the weather today? → 今天天气 么样?
 $\{*\text{text}*\}$ $\{*\text{translation}*\}$

我们可以使用提示模板生成一个微调样本，如下所示

今天天气怎么样?

这是，

$x =$ 将英文翻译成中文.\n 今天天气怎么样? $y =$

我们可以使用这个 (x, y) 对来微调LLM，如前一小节所述。

一个困难在于，对于同一任务，有非常多种不同的编写提示模板的方式，不同的人可能会制作出质量、复杂度各异的提示模板。有时，我们可能会编写过于复杂或冗长的指令作为提示模板。有时，我们甚至可能不清楚目标任务是什么以及如何描述它。一个广泛采用的战略是，鉴于已经存在许多确立的NLP问题和基准，为现有的NLP任务创建提示模板[Bach等人, 2022; Wang等人, 2022b; Mishra等人, 2022]。在这种情况下，标注者可以提供原始任务描述和许多示例。然后，他们可以使用自己的方式来表达如何提示LLM执行任务。请注意，尽管这种方法可以简化创建和编写提示的过程，我们仍然需要标注框架和众包系统来管理工作和进行质量控制。例如，我们通常需要设计标注指南和

一个用于编写提示模板的统一格式，尤其是在许多注释员为同一任务做出贡献时。从现有的NLP任务中诱导提示的一个优点是，一旦开发出提示模板，就很容易使用原始任务中的注释样本生成提示。例如，给定一个用于英译中的双语数据集，我们可以通过上面的模板中填充该数据集中的句子对，轻松创建多个微调示例。

另一种方法是直接使用互联网上自然存在的数据。一个常见的例子是从问答网站上收集问答对来微调LLMs以进行开放域问答任务[Joshi等人，2017]。许多问答基准都是这样构建的，因为存在如此多的问题类型，以至于不可能由一小群人想出所有这些问题。相反，使用这些数据可以确保LLM微调数据在数量和质量方面都处于良好或可接受的水平。

除了利用现有资源外，开发微调数据集的另一种简单方法是通过众包数据。一种简单的方法是允许用户输入任何问题，之后回答要么是手动给出，要么由LLM自动生成，然后手动标注和纠正。因此，可以捕捉到真实用户行为，从而收集大量“新”问题的输入和输出，这些问题是传统NLP任务所不涵盖的。

与微调数据集构建相关的问题是，我们通常希望数据尽可能多样化。许多研究发现，增加微调数据的多样性可以提高LLMs的鲁棒性和泛化能力。因此，在LLMs微调数据集中涉及更多样化的提示和任务引起了相当大的兴趣。我们将在第4.2.4节中进一步讨论微调的泛化。

4.2.2.2 自动生成数据

手动数据生成的局限性在于，质量和多样性在很大程度上取决于人类经验和创造力。因此，如果我们想让LLMs处理广泛的任務，即有效地执行任何指令，依赖于人类标注数据对LLMs进行微调通常效率低下。此外，此类数据的覆盖范围可能有限，数据甚至可能包含标注者自身引入的偏差。一种替代方法是自动生成数据。例如，我们可以通过众包收集大量问题，并使用调优良好的LLM生成问题的答案。然后，将这些问答对作为常规微调样本使用。这种方法虽然非常简单，但已被广泛用于为LLMs生成大规模微调数据。

上述生成合成微调数据的方法类似于用于NLP数据增强的方法。如果我们有一个LLM，我们就可以对任何输入产生预测。对不同的输入重复此过程可以让我们创建足够的微调样本。这种方法特别适用于使用调优良好的LLM来微调新的LLM。然而，这种方法的一个缺点是它依赖于人工制作或收集的输入来生成数据，这可能会发现不适合泛化LLM。在许多LLM应用中，一个重大挑战来自于用户问题的广泛范围和请求，其中许多不在现有的NLP任务和数据集中。在这些情况下，不仅需要生成预测，还需要生成输入本身。

这里我们将自指导作为例子来说明如何生成LLM微调

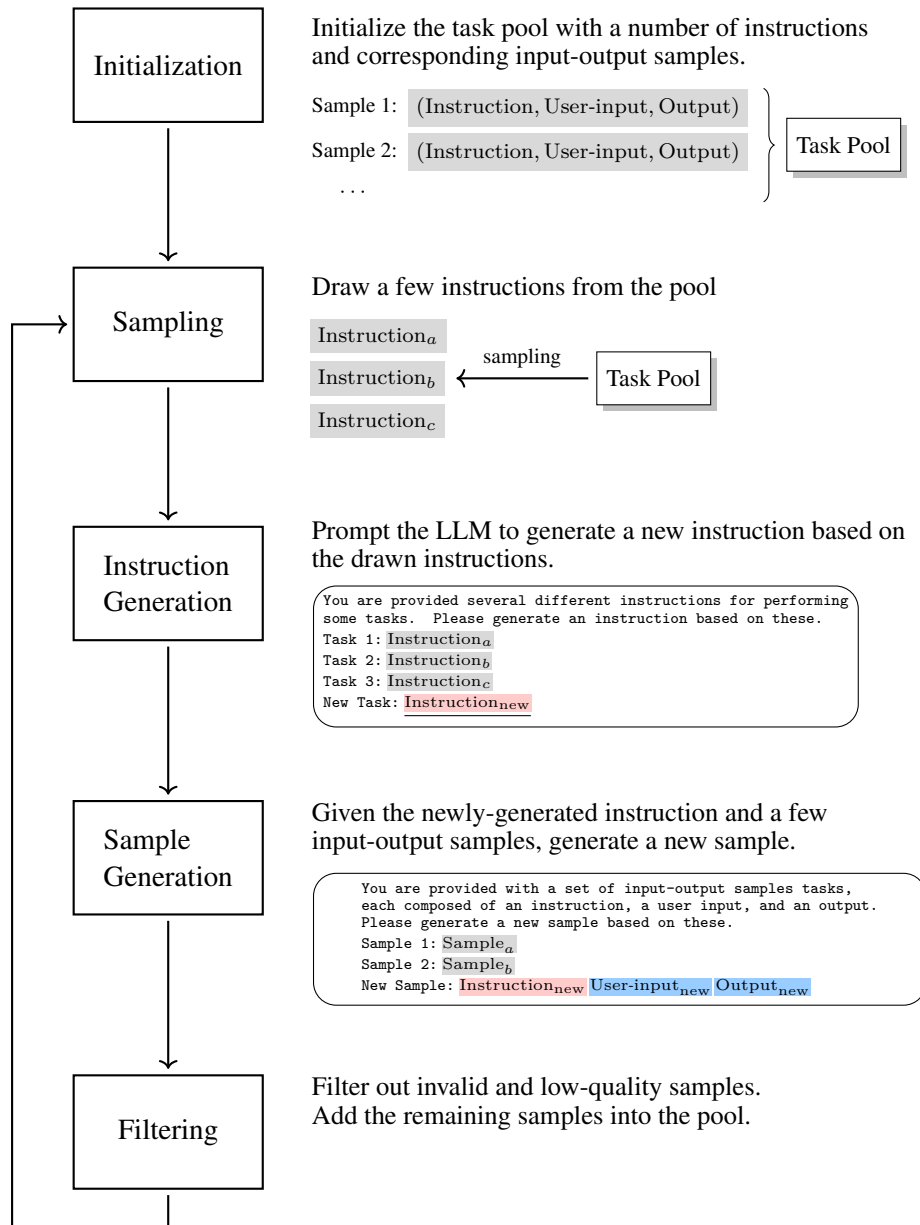


图4.4: 自指令说明 [王等, 2023b]。此方法维护一个指令和相应的输入输出样本池。最初, 池中包含一定数量的手工制作指令和样本。每次, 我们从池中抽取一些指令。然后提示一个LLM根据抽取的指令生成新的指令和样本。最后, 将新生成的指令和样本过滤并添加到池中。

samples [王等, 2023d; 霍诺维奇等, 2023]。想法是我们可以通过学习其他指令来提示一个LLM创建一个新的指令。给定这个指令, LLM可以填写其他字段(如用户输入)并产生预测。图4.4展示了自指令的示意图。在此, 我们简要概述了涉及的关键步骤。

- 自我指令算法维护一个任务池。最初它包含一些种子手工制作的任務, 每个任务都有一个指令和输入输出样本。随着算法的进行, 将向此池中添加由LLM生成的指令和样本。

- 在每一步，从指令池中抽取少量指令。例如，我们可以随机选择一些人工编写的指令和一些LLM生成的指令以确保多样性。
- 选定的指令随后用作演示示例。因此，LLM 可以从这些示例中情境学习并生成新的指令。以下是一个用于提示 LLM 的示例模板。

您被提供了执行某些任务的不同指令。请根据这些指令生成一条指令。

任务 1: {指令1} 任务
2: {指令2} 任务 3: {
指令3} 任务 4: {指令
4} 新任务:

- 给定生成的指令，然后提示LLM通过填写剩余的输入字段并生成相应的输出来完成样本。下面是提示模板。

您提供了一组输入输出样本，每个样本由一个指令、用户输入和输出组成。请根据这些样本生成一个新的样本。

样本 1: {指令1} 输入: {用户输入1} 输出: {输出1} 样本 2: {指令2} 输入: {用户输入2} 输出: {输出2} 新样本: {新指令}

- 这个新生成的样本通过一些启发式规则（如过滤掉与池中已有的样本或指令相似的样本或指令）进行检验。如果通过，则将样本和指令添加到池中。

此生成过程可以重复多次以获得足够的微调样本。注意，上面我们仅展示了简单的提示模板以生成指令和微调样本。当然，我们可以开发更好的模板以生成更多样化和准确的指令和微调样本。例如，对于某些任务如文本分类，

LLM可能会产生有偏见的预测，例如，大多数生成的样本属于单个类别。在这种情况下，我们可以调整不同字段生成的顺序。更具体地说，我们可以指定输出（即类别）的一些先验，并提示LLM根据指令和输出生成用户输入。这种方法类似于输入反转，其中LLM根据指定的输出生成输入[Longpre等人，2023]。

使用LLM生成的指令和微调样本是开发LLM的常见方法，尤其是在手动开发此类数据如此昂贵以至于大多数研究组负担不起的情况下。在几个调优良好的LLM中，它们的微调数据集包括一定量的合成数据，这已被证明是有用的[欧阳等，2022；太里等，2023；江等，2023]。关于LLM微调的合成数据生成已有进一步的研究。例如，可以通过引入进化算法[徐等，2024]生成更多样化的指令，或者将合成数据作为更高级微调过程中的监督信号[陈等，2024b]。最近，对在预训练阶段使用合成数据也产生了相当大的兴趣[古纳塞卡拉等，2023；阿拉尔等，2024]。

在许多应用中，现实场景通常是，给定一个任务，我们可以收集或标注相对少量的微调数据，例如，我们可以招募专家为特定领域的QA任务创建问题。但这类数据的数量和多样性通常不足。在这种情况下，我们可以使用自指令技术生成更多样化的问答对，从而扩充微调数据。这为从一组微调样本开始启动LLM提供了一种方法。请注意，使用自生成数据是一种常见做法，并且长期以来一直应用于NLP。例如，这种方法已被成功用于解析和机器翻译[Charniak，1997；Sennrich等，2016]。

4.2.3 使用较少数据进行微调

随着指令微调的日益突出，对大规模、高质量微调数据的需求激增。例如，FLAN微调数据集，由1,836个任务编制而成，包含1500万个样本[Longpre等人，2023]。使用如此大规模的数据集微调LLM通常是一个计算密集型任务，尤其是在LLM中更新大量参数需要大量资源。缓解这一问题的方法之一是探索高效的模型训练方法，例如，可以使用参数高效的方法来更新模型的一小部分。然而，许多微调数据集包含大量合成数据，其中错误和偏差仍然不可避免。

另一种高效微调的方法是仅考虑最相关和最具影响力的示例进行微调。这样，我们可以在保持模型更新质量的同时减少需要处理的数据量。为此，有几种方法可以实现。例如，周等人[2023a]通过精心设计提示并从各种NLP任务中收集样本，构建了一个仅包含1,000个样本的指令遵循数据集。他们表明，使用此数据集微调的LLaMa 65B模型可以与或优于经过更多微调努力模型的性能。这表明，LLM可以适应响应各种任务，而无需在所有类型的指令遵循数据上都必须进行微调。陈等人[2024a]开发了一个基于GPT-3.5模型的系统，用于评估每个指令遵循样本的质量。因此，他们可以从现有数据集中选择高质量样本，使用更少的微调样本展示更好的微调性能。研究人员还开发了使用{v*}选择或过滤数据的方法。

启发式方法[赵等, 2024; 葛等, 2024], 或者优先考虑对微调过程有更大影响的数据[Xia等, 2024]。事实上, 这些方法中的大多数都可以看作是更大数据选择和过滤方法家族的实例。并且, 通常情况下, 使用更高质量(但可能更少)的数据对训练NLP模型是有益的。

指令微调的发现与传统NLP观点有所不同: 模型处理复杂问题的能力可以通过少量标注数据激活, 而不是需要大量监督数据来进行广泛训练。一种可能的解释是, 在预训练期间已经学会了根据指令生成正确响应的能力, 但在推理期间这种指令-响应映射的概率并不高。微调可以略微调整模型以使其遵循指令, 所需的训练努力远低于预训练。这与所谓的表面对齐假设密切相关, 该假设认为学习主要发生在预训练期间, 而随后的微调或对齐阶段对LLM的潜在知识库的贡献并不显著[Zhou等人, 2023a]。由于模型的核心能力和知识已经在预训练中确立, 因此可以通过相对较小的训练微调努力实现与用户需求的对齐。这暗示了用非常少的数据微调LLM的可能性。在另一个方向上, 可能没有必要将微调限制在成对的指令-响应数据上。例如, Hewitt等人[2024]发现, 仅通过在LLM上微调响应, 而不需要相应的指令, 就可以隐式地实现指令遵循。

与这里讨论相关的一个概念是样本效率。如果一个机器学习方法能够从少量训练示例中有效地学习, 那么它被称为样本高效的。从这个意义上讲, 与预训练相比, 指令微调是样本高效的。从机器学习的角度来看, 样本高效的方法可以被视为高效地采样数据空间的方法, 并且由于它们能够最大限度地利用稀缺数据, 因此具有优势。因此, 基于采样的学习技术, 如许多强化学习算法, 可以从这些样本高效的方法中受益。例如, 在人类偏好对齐中, 我们可以通过奖励模型 [刘等, 2024b] 高效地采样偏好数据, 或者提高策略学习中的采样效率 [王等, 2024]。

4.2.4 指令泛化

在许多机器学习和NLP问题中, 训练一个能够泛化的模型是一个基本目标。例如, 在文本分类中, 我们希望我们的模型能够正确分类在训练过程中未见过的文本。然而, 泛化在指令微调中提出了额外的挑战。我们希望指令微调的LLM不仅能够在任务中为不同的输入生成适当的响应, 而且能够根据不同的指令准确执行各种任务。为了说明这个问题, 考虑一个LLM $\Pr(y|c, z)$, 其中 c 是指令, z 是用户输入, y 是对应的模型输出(即响应)。假设这个模型的性能是通过一个指标来评估的, 写成 $\text{Performance}(\Pr(y|c, z))$ 或简写为 $P(c, z, y)$ 。非正式地说, 当我们说这个模型可以在给定的任务(由指令 c^* 指示)中泛化时, 我们意味着可能存在一个值 ϵ , 使得在新输入上的平均性能高于这个值:

$$\frac{1}{|Z|} \sum_{z' \in Z} P(c^*, z', y') > \epsilon \quad (4.8)$$

\mathcal{Z} 是新输入的集合, z' 和 y' 分别是该集合中的一个输入及其对应的输出。

同样, 我们可以说, 如果所有指令输入对的平均性能超过某个 ϵ , 则该模型可以在任务之间泛化:

$$\frac{1}{|\mathcal{D}|} \sum_{(c', z') \in \mathcal{D}} P(c', z', y') > \epsilon \quad (4.9)$$

\mathcal{D} 是新指令输入对的集合。

这里, 我们需要处理两个维度的变化: 指令和用户输入。这使得泛化问题非常复杂, 因为直观上, 一个模型需要从大量任务以及与每个任务相关联的不同输入输出对中学习, 以实现良好的泛化。正如我们在本书中多次讨论的那样, 实现这种泛化比预训练的成本要低得多。一般来说, 用指令响应数据微调LLM在一定程度上可以导致模型在新的任务上产生遵循指令的效果。尽管如此, 人们普遍认为, 仍然需要一定的努力来适应LLM, 使它们能够广泛理解和执行指令。

一种泛化指令微调的方法是增加微调数据的多样性。在早期关于指令微调的研究中, 研究人员开发了大量的数据集, 涵盖了广泛的NLP任务以及每个任务的多种指令[Wang等人, 2022b; Sanh等人, 2022; Longpre等人, 2023]。通过将这些任务转换为统一的格式, 可以微调一个具有足够大量样本的LLM, 例如, 已经有一些涉及超过100个NLP任务和100万样本的指令微调数据集。然而, 这些早期数据集主要关注现有的学术问题, 而不是用户在现实应用中想要解决的问题。最近的工作大多转向解决新的和更实际的问题。例如, 人们对于构建包含SOTA模型对真实用户查询的大规模和复杂演示及响应的数据集表现出浓厚的兴趣[Wang等人, 2023c; Teknium, 2023]。

或许大型且多样化的微调数据集的使用源于尝试在不同维度上扩展LLMs。确实, 扩展定律已被广泛用于激励开发各种不同的指令微调LLMs。将指令微调扩展到使LLM遵循广泛指令是合理的。然而, 从LLM对齐的角度来看, 扩展指令微调可能不是实现泛化的有效方法。

一个问题是, 指令微调依赖于监督学习, 该学习通过指令-响应映射来泛化和执行任务。然而, 这种方法无法捕捉到细微或复杂的人类偏好(例如, 语气、风格或主观质量), 因为这些难以编码为显式的指令-响应数据。此外, 泛化性能受限于指令-响应数据集的多样性和质量。鉴于这些限制, 我们宁愿在指令微调之后采用偏好模型作为额外的微调步骤, 以便LLMs可以进一步泛化(见第4.3节)。

另一种观点是, 一些指令-响应映射可能在预训练期间就已经学习到了, 因此预训练的LLM已经编码了这些映射。然而, 由于我们通常不知道在预训练中使用了哪些数据, 很难判断在微调中是否需要学习这样的映射。相关问题是关于分布外泛化

主要在预训练或微调期间实现。虽然直接回答这个问题超出了本章的范围，但已经证明在大型和多样化的数据集上进行预训练对于提高分布外性能是有效的 [Hendrycks 等人, 2020; Radford 等人, 2021; Gunasekar 等人, 2023]。这引发了一个有趣的问题：如果一个 LLM 在规模上已经得到了很好的预训练，那么微调对于分布外泛化可能就不是那么必要了，因为模型可能已经遇到了足够的分布变化。这促使研究人员以适度的努力微调 LLM 或探索新的方法来实现指令遵循。例如，正如前一小节所讨论的，指令遵循可以通过在少量精心选择的指令-响应对上进行微调来实现 [Zhou 等人, 2023a]，甚至可以通过使用并非明确设计来这样做的方法 [Kung 和 Peng, 2023]。

上述讨论提供了两种不同的策略：一种需要扩大微调数据集以实现更大的多样性，另一种需要小型但必要的微调数据集以实现高效的 LLM 适应。然而，在实践中，涉及多样化的指令通常有帮助。在许多情况下，我们需要针对特定目的调整我们的 LLM。但是，在预训练期间可能已经编码了广泛的指令遵循映射的 LLM，即使在适度的微调下，也可能倾向于作为通用指令执行者。一个有趣的现象是，当在数学数据上进行微调时，结果 LLM 可能不会专门从事数学输出。相反，这个模型可能会对一般指令做出正常响应，例如，如果被要求这样做，它可以生成诗歌 [Hewitt, 2024]。这并不是坏事，但它表明 LLM 可能不容易改变其遵循一般指令的本质。在这种情况下，使用更多样化的数据进行额外的调整可能有助于调整 LLM 遵循指令的方式，尤其是对于我们要解决的任务。

4.2.5 使用弱模型来改进强模型

截至目前，我们已经探索了基于标记数据的各种指令微调方法。许多此类方法的局限性之一是它们需要数据由人类标注或由强大的 LLM 生成，这可以在微调中提供准确的监督信号。然而，在许多情况下，我们手头的 LLM 已经足够强大（或者至少在解决问题的某些方面具有优势），因此很难找到一个优越的模型用于监督。即使是人类专家，当问题变得复杂时，提供正确和详细的答案可能很困难，有时甚至是不可能的。例如，面对一份极长的文档，专家在没有进行彻底和耗时审查的情况下，可能很难识别任何不一致性、细微偏见或缺失的关键点。

可以在这个时候问：我们能否使用弱 LLM 来监督强 LLM？这似乎是一个重大挑战，但它可能反映了这样一个未来场景，即我们需要监督比人类或任何其他 AI 系统更智能的 AI 系统 [Burns 等人, 2023b]。使用较小、较简单的模型来改进较大、较复杂的模型的难题也被称为弱到强泛化问题。尽管还没有成熟的弱到强泛化方法，但使用较小模型来协助较强模型在 LLM 的几个领域确实已被证明是有用的。

对于指令微调，应用弱 LLM 的最简单方法之一是使用这些模型生成合成微调数据。假设我们有一个输入集合 X ，其中每个输入包括一个指令和必要的用户输入。对于每个 $x \in X$ ，我们使用弱 LLM $\text{Pr}^w(\cdot)$ 生成预测 $\hat{y} = \arg \max_y \text{Pr}^w(y|x)$ 。然后，强 LLM $\text{Pr}_\theta^s(\cdot)$ 可以

在生成的预测上训练（见公式（4.1））：

$$\tilde{\theta} = \arg \max_{\theta} \sum_{\mathbf{x} \in X} \log \Pr_{\theta}^s(\hat{\mathbf{y}}|\mathbf{x}) \quad (4.10)$$

θ 是模型参数。

上述形式将微调问题转化为知识蒸馏问题，换句话说，我们从弱模型中提取知识到强模型。因此，我们可以采用各种知识蒸馏方法来实现这一目标。然而，从知识蒸馏的角度解释弱到强的微调并不简单。一个主要问题是强模型可能只是模仿或过度拟合弱模型的错误，而无法泛化。例如，微调后的强模型仍然无法解决弱模型无法准确预测的难题。幸运的是，这一研究领域的初步实验已经显示出积极和有希望的结果。例如，Burns等人[2023a]发现，使用GPT-2级别的监督微调预训练的强GPT-4模型可以提高多个NLP任务的泛化能力。为了衡量弱模型如何提高强模型的泛化能力，我们定义以下术语：

- 弱性能（ P_{weak} ）。这是弱模型的测试集性能，可以视为基线性能。
- 弱到强性能（ $P_{\text{weak} \rightarrow \text{strong}}$ ）。这是使用弱模型微调的强模型的测试集性能。
- 强天花板性能（ P_{ceiling} ）。这是使用真实数据进行微调的强模型的测试集性能。例如，我们使用人工标注的预测来微调强模型，并将得到的模型作为天花板。

然后，性能差距恢复（PGR）可以定义为

$$\text{PGR} = \max \left\{ 0, \frac{P_{\text{weak} \rightarrow \text{strong}} - P_{\text{weak}}}{P_{\text{ceiling}} - P_{\text{weak}}} \right\} \quad (4.11)$$

此指标衡量弱到强模型可以恢复多少天花板模型和弱模型之间的性能差距。PGR为1表示弱到强微调可以完全关闭性能差距，而PGR为0表示没有改进。在Burns等人[2023a]的研究中，显示PGR在22个NLP分类任务上可以达到约0.8。需要注意的是，尽管弱到强微调的潜力很有希望，但实现实质性的弱到强泛化仍然是一个需要进一步研究的具有挑战性的目标[Aschenbrenner, 2024]。

微调LLMs使用弱监督只是使用小模型改进大模型的一种选择。尽管本节主要关注微调LLMs，但我们在此也提到了其他方法，以提供更完整的讨论（参见图4.5以了解这些方法的说明）。

- 而不是使用小型模型生成合成数据，也可以简单地基于这些模型引入知识蒸馏损失。例如，一个简单的损失

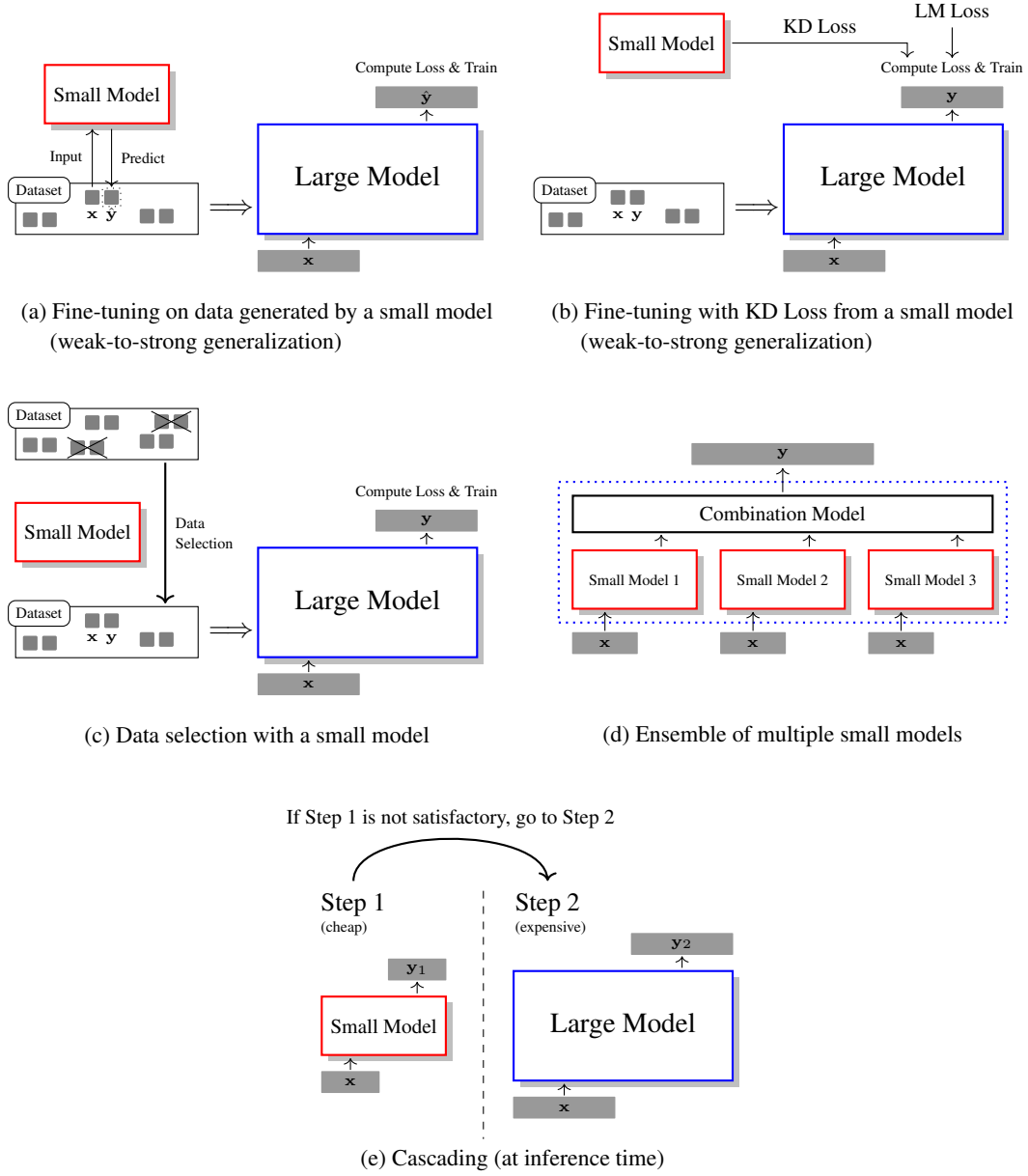


图4.5: 在LLMs中使用小模型改进大模型的说明。一种方法涉及使用较小模型进行大模型的微调或预训练。这包括生成合成数据 (a)、引入辅助损失 (b) 和选择适当的数据 (c)。另一种方法涉及结合小模型和大模型。这包括通过聚合多个小模型学习一个强大模型 (d)，以及级联小模型与大模型 (e)。

函数可以定义为测量小模型和大模型之间差异的：

$$\text{Loss}_{\text{kd}} = \text{KL}(\text{Pr}^w(\cdot|\mathbf{x}) \parallel \text{Pr}_\theta^s(\cdot|\mathbf{x})) \quad (4.12)$$

然后，我们可以将这个损失添加到语言模型的原始损失中，并得到以下训练目标

$$\tilde{\theta} = \arg \max_{\theta} \sum_{(\mathbf{x}, \mathbf{y}) \in \mathcal{D}} \log \text{Pr}_\theta^s(\mathbf{y}|\mathbf{x}) - \lambda \cdot \text{Loss}_{\text{kd}} \quad (4.13)$$

在 \mathcal{D} 是输入和输出对的集合， λ 是插值的系数的情况下。此方法可以在预训练或微调阶段使用。我们可以调整 λ 来控制小模型对训练的影响程度。例如，我们可以逐渐减小 λ ，使训练更多地依赖于原始语言建模损失，因为大模型变得越来越强大。

- 另一种将小型模型纳入LLM预训练和微调的方法是使用它们进行数据选择或过滤。给定一个序列，我们可以使用一个小型模型来计算似然或交叉熵。这些量可以用作选择或过滤数据的标准。例如，似然低或交叉熵高的序列可能被排除在训练集之外，因为它们与小型模型学习到的分布不太一致。相反，似然高或交叉熵低的序列可以优先考虑，确保训练集中在更相关或高质量的数据上。
- 集成学习是通过结合多个弱模型来构建强大模型的一种简单而有效的方法。将此技术应用于LLMs非常直接。我们可以聚合多个小型模型或专用子模型预测的分布，并从聚合结果中得出最终预测。这种聚合可以使用如多数投票、加权平均或堆叠等方法进行。
- 小型模型也可以在推理时使用，以提高整体效率。假设我们有一个大模型，它运行缓慢但更准确，还有一个小模型，它运行速度快但不太准确。在模型级联中，小模型首先处理输入数据，快速生成初步结果。如果这些结果满足某些预定义的标准，它们可以直接使用。然而，如果初始结果不够好，输入就会被传递给更大、更准确的模型以产生更好的结果。这种方法显著降低了计算成本和延迟，因为小模型可以有效地处理许多输入，而无需访问大模型。

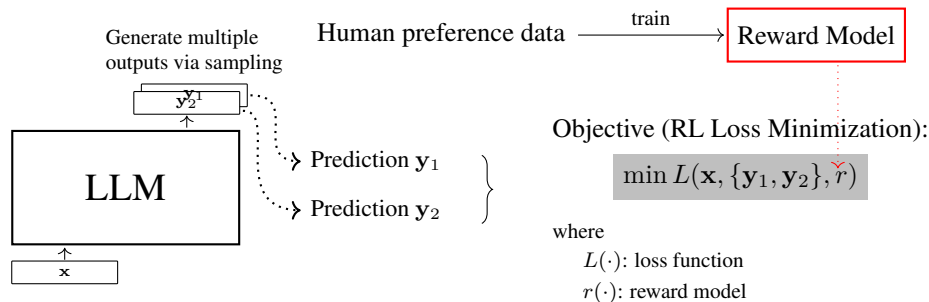
4.3 人类偏好对齐：RLHF

本章迄今为止，我们专注于使用输入-输出配对的有标签数据微调LLMs。这种方法使我们能够通过监督学习来调整LLMs以遵循指令。然而，在许多应用中，LLMs不仅需要遵循指令，还需要以更符合人类价值观和偏好的方式行事。考虑一个场景，用户询问一个LLM如何入侵计算机系统。如果LLM没有适当对齐，它可能会通过提供如何执行这项非法活动的详细信息来回应。相反，更理想的做法可能是建议用户不要参与非法活动，并概述此类行为的后果。实现这一点的困难在于，LLM在此类场景中适当回应所需的伦理细微差别和上下文考虑并不总是容易编码到微调数据集中。更具挑战性的是，通常，人类本身也无法精确表达自己的偏好。

在这个部分，我们讨论了一种名为基于人类反馈的强化学习或简称RLHF的替代性LLM微调方法[Christiano等人，2017；Stiennon等人，2020]。RLHF背后的基本思想是LLM可以通过使用奖励来从模型输出的比较中学习。



(a) Supervised fine-tuning (maximizing the prediction probability given the input)



(b) Reinforcement Learning from Human Feedback

图4.6: 监督微调与基于人类反馈的强化学习。在监督微调中, 我们通过最大化给定输入的预测概率来优化LLM。在基于人类反馈的强化学习中, 我们首先在人类偏好数据上训练一个奖励模型(在每一对预测中, 评估者被要求选择他们更喜欢哪一个)。然后, 我们使用这个奖励模型来监督LLM在微调过程中的学习。

模型(见图4.6)。为此, 我们可以招募人类专家, 让他们在LLM生成的输出对之间表明他们的偏好。这些偏好数据用于训练一个可以预测LLM输出感知质量的奖励模型。一旦训练完成, 奖励模型通过为新LLM生成的输出分配分数来提供反馈。LLM使用这些分数通过强化学习算法更新其参数。在本节的其余部分, 我们首先介绍强化学习的基本知识, 以促进讨论, 然后讨论训练奖励模型和将LLM与这些模型对齐的方法。

4.3.1 强化学习基础

我们首先探讨强化学习的一些基本概念。请注意, 这里使用的符号与前面章节中使用的略有不同, 因为我们希望使我们的描述与强化学习文献中的描述更加一致。尽管如此, 我们将展示这个符号如何与语言建模符号相对应。对于已经熟悉强化学习技术的读者, 可以跳过或快速浏览本小节。

一个通用的强化学习框架描述了智能体如何与动态环境交互。这种交互被建模为智能体对环境状态的响应所采取的一系列动作。在每个时间步, 智能体观察当前状态, 根据其策略选择一个动作, 执行该动作, 然后从环境中以奖励和新状态的形式接收反馈。这种观察-行动-接收反馈的序列被重复

直到代理实现其目标。

强化学习系统涉及多个组件：

- 代理。这是强化学习中的学习器或决策者。在LLMs的上下文中，它可以被视为LLM本身。
- 环境。这包括与代理交互的所有外部事物。但在LLMs中，环境更多地是指代理（例如，一个LLM）接收反馈和学习的框架，而不是物理或虚拟空间。
- 状态 (s)。状态表示环境的当前情况。对于语言模型中的标记序列，一个时间步长的状态可以看作是我们迄今为止观察到的标记，即我们用来预测下一个标记的上下文标记。例如，我们可以定义 $(\mathbf{x}, \mathbf{y}_{<t})$ 为在时间步长 t 预测下一个标记时的状态。
- 动作 (a)。动作表示代理可以做出的可能决策。我们可以把它们看作词汇表中的可能预测标记。
- 奖励 (R)。奖励是环境对动作成功度的反馈。例如， $r(s, a, s')$ 表示代理在状态 s 执行动作 a 并移动到下一个状态 s' 所获得的奖励。如果给定状态-动作序列，我们可以将时间步 t 的奖励表示为 $r_t = r(s_t, a_t, s_{t+1})$ 。此外，请注意，如果决策过程是确定性的，我们可以省略 s_{t+1} ，因为它可以通过 s_t 和 a_t 来确定。在这种情况下，我们可以使用 $r(s_t, a_t)$ 作为 $r(s_t, a_t, s_{t+1})$ 的简称。
- 策略 (π)。对于LLM，策略被定义为LLM在给定先前上下文标记的情况下预测标记的概率分布。形式上，这可以表示为

$$\pi(a|s) = \Pr(y_t|\mathbf{x}, \mathbf{y}_{<t}) \quad (4.14)$$

在 a 对应于标记 y_t ，且 s 对应于上下文 $(\mathbf{x}, \mathbf{y}_{<t})$ 。图 4.7 阐述了如何在强化学习框架中将一个 LLM 视为一个策略。

- 值函数 ($\{v^*\}$)。状态值函数（或简称值函数）评估从特定状态 s 开始并遵循特定策略 π 的代理的期望折现回报（即累积奖励）。它定义为：

$$\begin{aligned} V(s) &= \mathbb{E}\left[r(s_0, a_0, s_1) + \gamma r(s_1, a_1, s_2) + \gamma^2 r(s_2, a_2, s_3) + \cdots \mid s_0 = s, \pi\right] \\ &= \mathbb{E}\left[r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots \mid s_0 = s, \pi\right] \\ &= \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s, \pi\right] \end{aligned} \quad (4.15)$$

在 $\gamma \in [0, 1]$ 中， γ 是调整未来奖励重要性的折现因子， $s_0 = s$ 表示代理从状态 s 开始，期望 \mathbb{E} 是对所有可能的轨迹（即状态-动作序列）进行计算。同样，一个动作值函数（或

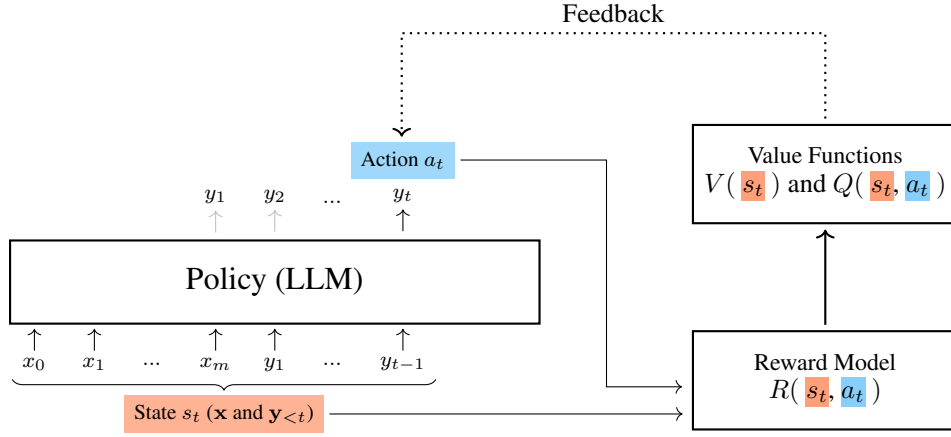


图4.7: 强化学习中作为策略的LLM。在每一步 t , LLM根据模型输入 x 和之前生成的标记 $y_{<t}$ 预测一个标记 y_t 。这个过程可以被视为一个强化学习问题, 其中 y_t 作为动作, $(x, y_{<t})$ 作为状态, 预测分布 $\Pr(y_t|x, y_{<t})$ 作为策略。一旦 y_t 被预测, LLM将 $(x, y_{<t})$ 和 y_t 输入到奖励模型, 该模型评估所选标记对实现所需文本结果的有效贡献。这种评估生成奖励分数, 用于计算价值函数 $V(s_t)$ 和 $Q(s_t, a_t)$ 。这些函数随后为LLM提供反馈并指导策略训练。

Q值函数)衡量从状态 s 开始采取行动 a 并随后遵循策略 π 的预期回报

$$Q(s, a) = \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s, a_0 = a, \pi \right] \quad (4.16)$$

$a_0 = a$ 表示在初始状态下采取的动作是 a_0 。

强化学习的目标是学习一个策略, 该策略最大化代理在长期内获得的累积奖励 (或回报)。给定一个状态-动作序列 $\tau = \{(s_1, a_1), \dots, (s_T, a_T)\}^1$, 该序列上的累积奖励可以表示为

$$R(\tau) = \sum_{t=1}^T r_t \quad (4.17)$$

该累积奖励在状态-动作序列空间中的期望以以下形式给出

$$\begin{aligned} J(\theta) &= \mathbb{E}_{\tau \sim \mathcal{D}} [R(\tau) \mid \pi_{\theta}] \\ &= \sum_{\tau \in \mathcal{D}} \Pr_{\theta}(\tau) R(\tau) \\ &= \sum_{\tau \in \mathcal{D}} \Pr_{\theta}(\tau) \sum_{t=1}^T r_t \end{aligned} \quad (4.18)$$

¹We assume the state-action sequence begins with s_1 and a_1 , rather than s_0 and a_0 , to align with the notation commonly used in this chapter, where the prediction y typically starts from y_1 . Of course, it is also common to denote a state-action sequence as $\{(s_0, a_0), \dots, (s_T, a_T)\}$ or $\{(s_0, a_0), \dots, (s_{T-1}, a_{T-1})\}$ in the literature. But this variation in notation does not affect the discussion of the models presented here.

$\tau \sim \mathcal{D}$ 表示 τ 从状态-动作序列空间 \mathcal{D} 中抽取，下标 θ 表示策略的参数。 $J(\theta)$ 也称为性能函数。

然后，训练目标是最大化 $J(\theta)$ ：

$$\tilde{\theta} = \arg \max_{\theta} J(\theta) \quad (4.19)$$

现在，我们有一个简单的强化学习方法：1) 我们采样一组状态-动作序列；然后，2) 我们使用性能函数评估每个序列；然后，3) 我们更新模型以最大化这个性能函数。如果我们取公式 (4.18) 并使用梯度下降来优化策略，这种方法就构成了策略梯度方法[Williams, 1992]。

注意，在许多NLP问题中，例如机器翻译，奖励通常是稀疏的。例如，只有在完整句子的末尾才会收到奖励。这意味着对于所有 $t < T$ ， $r_t = 0$ ，并且只有当 $t = T$ 时， r_t 才是非零的。理想情况下，人们可能更喜欢即时和频繁（密集）的反馈，因此策略的训练可以更容易和更高效。虽然已经提出了几种方法来解决稀疏奖励问题，例如奖励塑造，但我们将继续假设一个稀疏奖励设置，其中只有在完成预测后才能获得奖励。

模型描述在公式 (4.17-4.19) 中，建立了一种基本的强化学习形式，并且已经开发了许多该模型的变体和改进。在展示那些更复杂的模型之前，让我们花一点时间从策略梯度的角度来解释目标函数 $J(\theta)$ ：在梯度下降中，我们需要计算 $J(\theta)$ 关于 θ 的梯度：

$$\begin{aligned} \frac{\partial J(\theta)}{\partial \theta} &= \frac{\partial \sum_{\tau \in \mathcal{D}} \Pr_{\theta}(\tau) R(\tau)}{\partial \theta} \\ &= \sum_{\tau \in \mathcal{D}} \frac{\partial \Pr_{\theta}(\tau)}{\partial \theta} R(\tau) \\ &= \sum_{\tau \in \mathcal{D}} \Pr_{\theta}(\tau) \frac{\partial \Pr_{\theta}(\tau) / \partial \theta}{\Pr_{\theta}(\tau)} R(\tau) \\ &= \sum_{\tau \in \mathcal{D}} \Pr_{\theta}(\tau) \frac{\partial \log \Pr_{\theta}(\tau)}{\partial \theta} R(\tau) \end{aligned} \quad (4.20)$$

在某些情况下，我们将假设 \mathcal{D} 中的每个序列都是等可能的（即， $\Pr_{\theta}(\tau) = 1/|\mathcal{D}|$ ）。在这种情况下，我们可以简化公式 (4.20)，只需考虑项 $\frac{\partial \log \Pr_{\theta}(\tau)}{\partial \theta}$ 和 $R(\tau)$ ：

$$\frac{\partial J(\theta)}{\partial \theta} = \frac{1}{m} \sum_{\tau \in \mathcal{D}} \frac{\partial \log \Pr_{\theta}(\tau)}{\partial \theta} R(\tau) \quad (4.21)$$

这个结果的一个优点是， $R(\tau)$ 不需要可微，这意味着我们可以在强化学习中使用任何类型的奖励函数。

通过将序列 τ 的生成视为马尔可夫决策过程，我们可以进一步

推导 $\frac{\partial \log \Pr_\theta(\tau)}{\partial \theta}$ ，并得到：

$$\begin{aligned} \frac{\partial \log \Pr_\theta(\tau)}{\partial \theta} &= \frac{\partial}{\partial \theta} \log \prod_{t=1}^T \pi_\theta(a_t|s_t) \Pr(s_{t+1}|s_t, a_t) \\ &= \frac{\partial}{\partial \theta} \sum_{t=1}^T \underbrace{\log \pi_\theta(a_t|s_t)}_{\text{policy}} + \frac{\partial}{\partial \theta} \sum_{t=1}^T \underbrace{\log \Pr(s_{t+1}|s_t, a_t)}_{\text{dynamics}} \end{aligned} \quad (4.22)$$

梯度分解为两部分：策略梯度和动力学梯度。策略组件， $\log \pi_\theta(a_t|s_t)$ ，决定了在状态 s_t 下采取行动 a_t 的对数概率，并由 θ 参数化。动力学组件， $\log \Pr(s_{t+1}|s_t, a_t)$ ，表示在采取行动 a_t 后从状态 s_t 转移到状态 s_{t+1} 的对数概率。在典型的强化学习设置中，动力学不受策略参数 θ 的直接影响，因此它们的导数通常是零。在这种情况下，因此，式(4.22)可以简化为：

$$\frac{\partial \log \Pr_\theta(\tau)}{\partial \theta} = \frac{\partial}{\partial \theta} \sum_{t=1}^T \log \pi_\theta(a_t|s_t) \quad (4.23)$$

换句话说，我们只专注于优化策略，而不关心其底层动力学。

替换 等式 (4.23) 代入等式 (4.21)，并展开 $R(\tau)$ ，然后获得

$$\frac{\partial J(\theta)}{\partial \theta} = \frac{1}{|\mathcal{D}|} \sum_{\tau \in \mathcal{D}} \frac{\partial}{\partial \theta} \left(\sum_{t=1}^T \log \pi_\theta(a_t|s_t) \sum_{t=1}^T r_t \right) \quad (4.24)$$

虽然这种策略梯度方法很简单，但它存在一个问题，即估计梯度的方差可能非常高，这使得学习过程变得嘈杂且效率低下。这种高方差问题的一个原因是奖励在不同步骤或场景中可能差异很大。想象一下，在一个动作决策序列中，奖励模型倾向于将小奖励分配给好的动作（例如， $R_t = 2$ ）并将大惩罚分配给差的动作（例如， $R_t = -50$ ）。这种对好和差动作的奖励尺度变化可能导致整个序列的总奖励非常低，即使它包括好的动作。

一种降低梯度方差的简单方法是将基线 b 从 $\sum_{t=1}^T r_t$ 中减去，得到 $\sum_{t=1}^T r_t - b$ 。² 在这里，基线可以解释为一个参考点。通过将奖励围绕这个基线进行中心化，我们消除了奖励信号中的系统性偏差，使更新更加稳定，并对单个奖励的极端波动不那么敏感。

²In fact, the use of a baseline b does not change the variance of the total rewards $\sum_{t=1}^T r_t$. However, it is important to note that while introducing a baseline does not alter the overall variance of the rewards, it helps reduce the variance of the gradient estimates. This is because subtracting the baseline from the total rewards effectively reduces fluctuations around their mean, which makes the gradient estimates more stable. In general, the operation $\sum_{t=1}^T r_t - b$ centers the rewards around zero (e.g., b is defined as the expected value of $\sum_{t=1}^T r_t$), which can lead to reduced variance in the product $\sum_{t=1}^T \log \pi_\theta(a_t|s_t) (\sum_{t=1}^T r_t - b)$.

此具有基线的策略梯度模型可以给出

通过

$$\begin{aligned}
 \frac{\partial J(\theta)}{\partial \theta} &= \frac{1}{|\mathcal{D}|} \sum_{\tau \in \mathcal{D}} \frac{\partial}{\partial \theta} \left(\sum_{t=1}^T \log \pi_{\theta}(a_t | s_t) \right) \left(\sum_{t=1}^T r_t - b \right) \\
 &= \frac{1}{|\mathcal{D}|} \sum_{\tau \in \mathcal{D}} \frac{\partial}{\partial \theta} \left[\sum_{t=1}^T \log \pi_{\theta}(a_t | s_t) \left(\sum_{k=1}^T r_k - b \right) \right] \\
 &= \frac{1}{|\mathcal{D}|} \sum_{\tau \in \mathcal{D}} \frac{\partial}{\partial \theta} \left[\sum_{t=1}^T \log \pi_{\theta}(a_t | s_t) \left(\sum_{k=1}^{t-1} r_k + \sum_{k=t}^T r_k - b \right) \right] \quad (4.25)
 \end{aligned}$$

这里我们将 $\sum_{k=1}^T r_k$ 写作两个项 $\sum_{k=1}^{t-1} r_k$ 和 $\sum_{k=t}^T r_k$ 的和，以区分在时间步 t 的动作之前和之后的累积奖励。请注意，在马尔可夫决策过程中，给定当前状态，未来与过去是独立的。因此，在时间步 t 采取的动作不能影响在 t 之前收到的奖励，换句话说，在 t 之前奖励已经由 t 的动作“固定”。项 $\sum_{k=1}^{t-1} r_k$ 不对梯度有贡献，可以省略，从而得到方程 (4.25) 的简化版本

$$\frac{\partial J(\theta)}{\partial \theta} = \frac{1}{|\mathcal{D}|} \sum_{\tau \in \mathcal{D}} \frac{\partial}{\partial \theta} \left[\sum_{t=1}^T \log \pi_{\theta}(a_t | s_t) \left(\sum_{k=t}^T r_k - b \right) \right] \quad (4.26)$$

同时请注意，移除 $\sum_{k=t}^T r_k$ 可以进一步降低梯度的方差。

有许多定义基线 $\{v^*\}$ 的方法。在这里，我们考虑状态 s_t 的值函数，即处于状态 s_t 的估计值： $V(s_t) = \mathbb{E}(r_t + r_{t+1} + \dots + r_T)$ 。因此，我们有

$$\begin{aligned}
 A(s_t, a_t) &= \sum_{k=t}^T r_k - b \\
 &= \sum_{k=t}^T r_k - V(s_t) \quad (4.27)
 \end{aligned}$$

$\sum_{k=t}^T r_k$ 表示实际收到的回报， $V(s_t)$ 表示预期回报。简称为 $A(s_t, a_t)$ (或 A_t) 的被称为第 t 步的优势，它量化了动作 a_t 相对于从状态 s_t 开始遵循策略的预期值的相对收益。

通过使用优势函数 $A(s_t, a_t)$ ， $J(\theta)$ 的梯度可以写成如下形式

$$\frac{\partial J(\theta)}{\partial \theta} = \frac{1}{|\mathcal{D}|} \sum_{\tau \in \mathcal{D}} \frac{\partial}{\partial \theta} \left(\sum_{t=1}^T \log \pi_{\theta}(a_t | s_t) A(s_t, a_t) \right) \quad (4.28)$$

此优化目标对应于强化学习中的优势演员-评论家 (A2C) 方法 [Mnih 等人, 2016]。在此方法中，演员旨在学习一个策略。它使用公式 (4.28) 更新策略参数，以帮助更多地关注可能提高性能的动作。另一方面，评论家更新其对价值函数的估计，该函数用于计算优势函数 $\{v^*\}$ ，从而作为评估者。

策略由演员学习。

在A2C方法中, $A(s_t, a_t)$ 通常表示为动作值函数 $Q(s_t, a_t)$ 和状态值函数 $V(s_t)$ 的差

$$A(s_t, a_t) = Q(s_t, a_t) - V(s_t) \quad (4.29)$$

首先看, 这个模型可能看起来很难开发, 因为它需要两个独立的子模型分别计算 $Q(s_t, a_t)$ 和 $V(s_t)$ 。幸运的是, 考虑到 $Q(s_t, a_t)$ 可以定义为回报 $r_t + V(s_{t+1})$, 我们可以将公式 (4.29) 重写为

$$A(s_t, a_t) = r_t + V(s_{t+1}) - V(s_t) \quad (4.30)$$

或者, 引入折扣因子 γ 以获得更一般的形式

$$A(s_t, a_t) = r_t + \gamma V(s_{t+1}) - V(s_t) \quad (4.31)$$

$A(s_t, a_t) = r_t + \gamma V(s_{t+1}) - V(s_t)$ 也被称为时间差分 (TD) 误差。我们需要训练一个用于值函数 $V(s_t)$ 的评论员网络, 然后使用它来计算优势函数³。

到这一点为止, 我们已经花费了大量篇幅讨论强化学习的基础知识, 特别是如何推导 A2C方法的优化目标。然而, 强化学习是一个庞大的领域, 许多技术细节无法在此涵盖。感兴趣的读者可以参考强化学习书籍以获取更多详细信息[Sutton and Barto, 2018; Szepesvári, 2010]。尽管如此, 我们现在已经拥有了进一步讨论RLHF所需的知识。在接下来的小节中, 我们将回到LLM对齐的讨论, 展示如何使用A2C方法与人类偏好对齐。

4.3.2 训练奖励模型

我们已经表明, 奖励模型在通用强化学习框架中起着非常重要的作用, 并构成了计算价值函数的基础。我们现在考虑训练这些奖励模型的问题。

在RLHF中, 奖励模型是一个将一对输入和输出标记序列映射到标量的神经网络。给定输入 \mathbf{x} 和输出 \mathbf{y} , 奖励可以表示为

$$r = \text{Reward}(\mathbf{x}, \mathbf{y}) \quad (4.33)$$

在 $\text{Reward}(\cdot)$ 是奖励模型的情况下。 r 可以解释为在给定输入 \mathbf{x} 的情况下, 输出 \mathbf{y} 与期望行为对齐程度的度量。如前一小节所述, \mathbf{x}

³The training loss for the value network (or critic network) in A2C is generally formulated as the mean squared error between the computed return $r_t + \gamma V(s_{t+1})$ and the predicted state value $V(s_t)$. Suppose that the value network is parameterized by ω . The loss function is given by

$$\mathcal{L}_v(\omega) = \frac{1}{M} \sum (r_t + \gamma V_\omega(s_{t+1}) - V_\omega(s_t))^2 \quad (4.32)$$

where M is the number of training samples, for example, for a sequence of T tokens, we can set $M = T$.

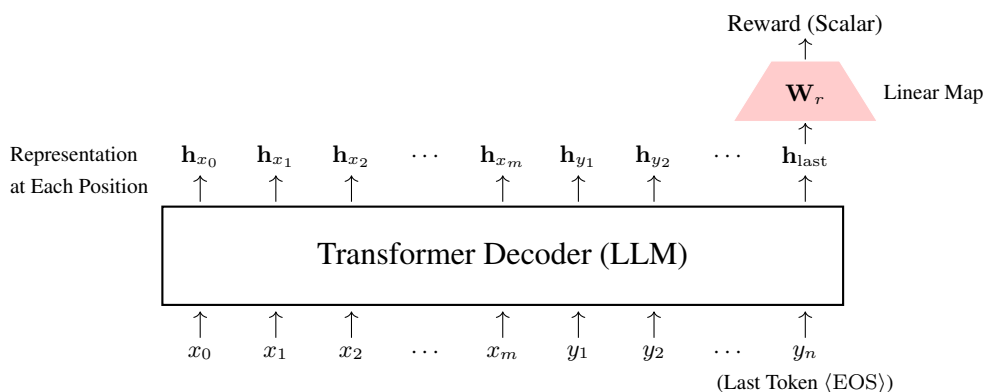


图4.8：基于Transformer的奖励模型架构。此模型的主要组件仍然是LLM。我们使用Transformer解码器作为序列表示模型。我们提取解码器最后一个位置的表示作为整个序列的表示 $[x, y]$ 。然后，我们通过线性变换将此表示映射到标量，这作为 $[x, y]$ 的奖励分数。

并且 y 被假定为完整文本。这意味着奖励模型评估了提供完整语义内容的输入和输出之间的关系。例如，在应用奖励模型时，它将在输出序列 y 的每个位置 t 上分配一个值 0（或另一个预定的值）。只有在最终位置 $t = n$ 时，奖励模型才会生成实际的奖励分数。为了保持符号简洁，从现在起我们将使用 $r(x, y)$ 来表示奖励模型 $\text{Reward}(x, y)$ 。

有多种实现奖励模型的方法。一种简单的方法是基于预训练的LLM构建奖励模型。更具体地说，我们可以将 x 和 y 连接起来形成一个单个标记序列 $\text{seq}_{x,y} = [x, y]$ 。我们像往常一样在这个序列上运行预训练的LLM，并在每个位置获得最顶层Transformer层的表示。然后，我们取最后一个位置的表示（表示为 h_{last} ）并通过线性变换将其映射到标量：

$$r(\mathbf{x}, \mathbf{y}) = \mathbf{h}_{\text{last}} \mathbf{W}_r \quad (4.34)$$

在 \mathbf{h}_{last} 是一个 d 维向量，且 \mathbf{W}_r 是一个 $d \times 1$ 线性映射矩阵。该奖励模型的架构如图4.8所示。

为了训练奖励模型，第一步是收集一组生成输出的人类反馈。给定输入 x ，我们使用 LLM 生成多个候选输出 $\{y_1, \dots, y_N\}$ 。人类反馈可以通过几种方式获得：

- 成对比较（成对排名）。给定两个不同的输出，人类专家选择哪一个更好。
- 评分。人类专家为每个输出提供分数或评级。这个分数通常是一个连续或离散的数值，例如一个量表上的分数（例如，1-5星，或1-10分）。在某些情况下，评级可能是二元的，表示“是/否”或“正面/负面”偏好。

- 列表排序。要求人类专家对给定的可能输出集进行排序或排列。

这里我们考虑成对比较反馈，因为它是最简单和最常见的人类反馈形式之一，在RLHF中应用。在这种设置中，每次从候选池 $\{y_1, \dots, y_N\}$ 中随机抽取两个输出 (y_a, y_b) 。然后，将这两对输出展示给人类专家，并要求他们根据特定标准，如清晰度、相关性和准确性，决定他们更喜欢哪一个。人类反馈可以编码为二进制标签， $y_a \succ y_b$ 表示更喜欢 y_a ， $y_b \succ y_a$ 表示更喜欢 y_b 。

一个简单且广泛用于描述此类成对比较的模型是Bradley-Terry模型[Bradley和Terry, 1952]。它是一个概率模型，用于估计一个物品被另一个物品偏好的概率。将此模型适应到此处使用的符号，我们可以将 y_a 比 y_b 更受偏好的概率写成以下形式

$$\begin{aligned} \Pr(y_a \succ y_b | \mathbf{x}) &= \frac{e^{r(\mathbf{x}, y_a)}}{e^{r(\mathbf{x}, y_a)} + e^{r(\mathbf{x}, y_b)}} \\ &= \frac{e^{r(\mathbf{x}, y_a) - r(\mathbf{x}, y_b)}}{e^{r(\mathbf{x}, y_a) - r(\mathbf{x}, y_b)} + 1} \\ &= \text{Sigmoid}(r(\mathbf{x}, y_a) - r(\mathbf{x}, y_b)) \end{aligned} \quad (4.35)$$

当训练奖励模型时，我们希望最大化这个偏好概率。基于Bradley-Terry模型的损失函数如下：

$$\mathcal{L}_r(\phi) = -\mathbb{E}_{(\mathbf{x}, y_a, y_b) \sim \mathcal{D}_r} [\log \Pr_\phi(y_a \succ y_b | \mathbf{x})] \quad (4.36)$$

在 (\mathbf{x}, y_a, y_b) 是从由输出和它们对应的输入组成的偏好对的人类标注数据集 \mathcal{D}_r 中抽取的情况下， ϕ 表示奖励模型的参数，包括Transformer解码器的参数和线性映射矩阵 \mathbf{W}_r 。在实践中，假设 (\mathbf{x}, y_a, y_b) 从 \mathcal{D}_r 中均匀采样，我们可以用求和来代替期望

$$\mathcal{L}_r(\phi) = -\frac{1}{|\mathcal{D}_r|} \sum_{(\mathbf{x}, y_a, y_b) \in \mathcal{D}_r} \log \Pr_\phi(y_a \succ y_b | \mathbf{x}) \quad (4.37)$$

训练奖励模型的目标是找到最优参数 $\hat{\phi}$ ，以最小化这个损失函数，该函数由以下给出

$$\hat{\phi} = \arg \min_{\phi} \mathcal{L}_r(\phi) \quad (4.38)$$

由于奖励模型本身也是LLM，我们可以直接重用Transformer训练过程来优化奖励模型。与训练标准LLM的不同之处在于，我们只需要将交叉熵损失替换为如公式（4.37）所述的成对比较损失。在训练奖励模型后，我们可以将训练好的奖励模型 $r_{\hat{\phi}}(\cdot)$ 应用于监督目标LLM以实现对齐。

值得注意，尽管我们训练奖励模型进行成对排名，但在对齐过程中，我们将其应用于独立评分每个输入输出对。成对

排名目标确保奖励模型对输出之间的细微差异敏感，但我们依赖于奖励模型产生的连续分数来指导LLM的优化。这种方法的一个优点是，我们可以选择或组合各种排名损失函数，并且仍然以与我们在本小节中所做相同的方式应用产生的奖励模型。这种一致性确保了无论在奖励模型训练期间使用哪种特定的排名损失，都可以为对齐LLM提供一个统一的框架。

4.3.3 训练LLMs

获得奖励模型后，我们随后通过A2C方法训练策略（即LLM）。回想第4.3.1节，一个状态-动作序列或轨迹 τ 可以通过效用函数进行评估

$$U(\tau; \theta) = \sum_{t=1}^T \log \pi_{\theta}(a_t | s_t) A(s_t, a_t) \quad (4.39)$$

在 $A(s_t, a_t)$ 是在状态 s_t 下采取行动 a_t 的优势。 $A(s_t, a_t)$ 的估计定义为 TD 错误 $r_t + \gamma V(s_{t+1}) - V(s_t)$ ，其中值函数 $V(s_t)$ 使用奖励模型进行训练。

给定这个效用函数，基于A2C的损失函数可以写成以下形式

$$\begin{aligned} \mathcal{L}(\theta) &= -\mathbb{E}_{\tau \sim \mathcal{D}} [U(\tau; \theta)] \\ &= -\mathbb{E}_{\tau \sim \mathcal{D}} \left[\sum_{t=1}^T \log \pi_{\theta}(a_t | s_t) A(s_t, a_t) \right] \end{aligned} \quad (4.40)$$

\mathcal{D} 是状态-动作序列的空间。通常，训练策略的目标是使这个损失函数最小化

$$\tilde{\theta} = \arg \min_{\theta} \mathcal{L}(\theta) \quad (4.41)$$

如果我们将问题映射回语言建模问题并采用LLMs的符号，损失函数可以写成：

$$\mathcal{L}(\theta) = -\mathbb{E}_{(\mathbf{x}, \mathbf{y}) \sim \mathcal{D}} [U(\mathbf{x}, \mathbf{y}; \theta)] \quad (4.42)$$

哪里

$$U(\mathbf{x}, \mathbf{y}; \theta) = \sum_{t=1}^T \log \pi_{\theta}(y_t | \mathbf{x}, \mathbf{y}_{<t}) A(\mathbf{x}, \mathbf{y}_{<t}, y_t) \quad (4.43)$$

这里 $\pi_{\theta}(y_t | \mathbf{x}, \mathbf{y}_{<t}) = \Pr_{\theta}(y_t | \mathbf{x}, \mathbf{y}_{<t})$ 是由 θ 参数化的 LLM。

通常，我们在RLHF中并没有一个包含输入输出的标注数据集 \mathcal{D} ，而是一个只包含输入的数据集。在这种情况下，输出通常是LLM做出的预测。

损失函数随后定义为

$$\mathcal{L}(\theta) = -\mathbb{E}_{\mathbf{x} \sim \mathcal{D}} \mathbb{E}_{\mathbf{y} \sim \pi_{\theta}(\cdot|\mathbf{x})} [U(\mathbf{x}, \mathbf{y}; \theta)] \quad (4.44)$$

\mathcal{D} 表示仅输入的数据集，而 $\mathbf{y} \sim \pi_{\theta}(\cdot|\mathbf{x})$ 表示输出 \mathbf{y} 由策略 $\pi_{\theta}(\cdot|\mathbf{x})$ 进行采样。

上述公式提供了LLMs的A2C方法的基本形式。此模型的改进版本在RLHF中更常用。在接下来的讨论中，我们仍将使用强化学习符号来简化表述，稍后我们将回到语言建模符号。

一种常见的策略梯度方法的改进是使用重要性采样来细化对 $U(\tau; \theta)$ 的估计。这可以写成

$$U(\tau; \theta) = \sum_{t=1}^T \frac{\pi_{\theta}(a_t|s_t)}{\pi_{\theta_{\text{ref}}}(a_t|s_t)} A(s_t, a_t) \quad (4.45)$$

这里我们将对数概率对数 $\pi_{\theta}(a_t|s_t)$ 替换为比率 $\frac{\pi_{\theta}(a_t|s_t)}{\pi_{\theta_{\text{ref}}}(a_t|s_t)}$ 。 θ_{ref} 表示先前策略的参数（例如，我们从其中开始训练的初始模型）。因此， $\frac{\pi_{\theta}(a_t|s_t)}{\pi_{\theta_{\text{ref}}}(a_t|s_t)}$ ，也称为比率函数，可以解释为当前策略 π_{θ} 和先前策略 $\pi_{\theta_{\text{ref}}}$ （称为参考策略）之间的对数概率比率。通过使用比率函数，我们根据当前策略与参考策略下动作的似然性重新加权观察到的奖励。当 $\frac{\pi_{\theta}(a_t|s_t)}{\pi_{\theta_{\text{ref}}}(a_t|s_t)} > 1$ 时，动作 a_t 相比参考策略更受当前策略的青睐。相比之下，当 $\frac{\pi_{\theta}(a_t|s_t)}{\pi_{\theta_{\text{ref}}}(a_t|s_t)} < 1$ 时，动作 a_t 相比当前策略更不受青睐⁴。

考虑一个更一般的情况，其中我们希望使用其期望奖励来评估策略（也可参见公式（4.18））

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_{\theta}} [R(\tau)] \quad (4.46)$$

这里 $\tau \sim \pi_{\theta}$ 表示序列 τ 是由策略 π_{θ} 生成的。或者，我们也可以用另一种形式写出 $J(\theta)$

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_{\theta_{\text{ref}}}} \left[\frac{\text{Pr}_{\theta}(\tau)}{\text{Pr}_{\theta_{\text{ref}}}(\tau)} R(\tau) \right] \quad (4.47)$$

它不难发现，这些方程的右侧本质上相同，因为 $\mathbb{E}_{\tau \sim \pi_{\theta_{\text{ref}}}} \left[\frac{\text{Pr}_{\theta}(\tau)}{\text{Pr}_{\theta_{\text{ref}}}(\tau)} R(\tau) \right] = \sum_{\tau} \text{Pr}_{\theta_{\text{ref}}}(\tau) \frac{\text{Pr}_{\theta}(\tau)}{\text{Pr}_{\theta_{\text{ref}}}(\tau)} R(\tau) = \sum_{\tau} \text{Pr}_{\theta}(\tau) R(\tau) = \mathbb{E}_{\tau \sim \pi_{\theta}} [R(\tau)]$

注意，这个等价性仅在在整个序列空间上执行期望时成立。然而，在实践中，我们通常只使用策略学习中的策略采样相对较小数量的序列。因此，采样方法本身很重要。公式（4.47）提供了一种有趣的方法来分离采样和奖励计算过程：我们首先使用基线策略（具有 θ_{ref} ）来采样一系列序列，然后使用目标策略（具有 θ ）来计算期望奖励。这样，我们分离了用于收集数据的策略和用于计算梯度的策略。这种方法避免了直接从评估策略中采样的需要，这在生成目标策略的序列昂贵或困难的情况下可能是有益的。在强化学习中， $\mathbb{E}_{\tau \sim \pi_{\theta_{\text{ref}}}} \left[\frac{\text{Pr}_{\theta}(\tau)}{\text{Pr}_{\theta_{\text{ref}}}(\tau)} R(\tau) \right]$ 通常被称为代理目标。

公式（4.47）也可以从策略梯度角度进行解释。对于 $\mathbb{E}_{\tau \sim \pi_{\theta_{\text{ref}}}} \left[\frac{\text{Pr}_{\theta}(\tau)}{\text{Pr}_{\theta_{\text{ref}}}(\tau)} R(\tau) \right]$ ， $\theta = \theta_{\text{ref}}$ 处的梯度由以下给出

$$\frac{\partial}{\partial \theta} \mathbb{E}_{\tau \sim \pi_{\theta_{\text{ref}}}} \left[\frac{\text{Pr}_{\theta}(\tau)}{\text{Pr}_{\theta_{\text{ref}}}(\tau)} R(\tau) \right] \Big|_{\theta=\theta_{\text{ref}}} = \mathbb{E}_{\tau \sim \pi_{\theta_{\text{ref}}}} \left[\frac{\partial \text{Pr}_{\theta}(\tau) |_{\theta=\theta_{\text{ref}}}}{\partial \theta} R(\tau) \right] \quad (4.48)$$

右侧是策略梯度方法中使用的标准形式，意味着我们计算方向

模型（如公式（4.47）以及公式（4.39）所示）存在的问题是梯度估计的方差通常很高，这使得学习过程不稳定。为了缓解这个问题，通常会采用诸如截断等技术来限制重要性权重并防止大的更新。截断版本的效用函数（也称为截断代理目标函数）如下所示：

$$U_{\text{clip}}(\tau; \theta) = \sum_{t=1}^T \text{Clip}\left(\frac{\pi_{\theta}(a_t|s_t)}{\pi_{\theta_{\text{ref}}}(a_t|s_t)}\right) A(s_t, a_t) \quad (4.49)$$

$$\text{Clip}\left(\frac{\pi_{\theta}(a_t|s_t)}{\pi_{\theta_{\text{ref}}}(a_t|s_t)}\right) = \min\left(\frac{\pi_{\theta}(a_t|s_t)}{\pi_{\theta_{\text{ref}}}(a_t|s_t)}, \text{bound}\left(\frac{\pi_{\theta}(a_t|s_t)}{\pi_{\theta_{\text{ref}}}(a_t|s_t)}, 1 - \epsilon, 1 + \epsilon\right)\right) \quad (4.50)$$

这里函数 $\text{bound}\left(\frac{\pi_{\theta}(a_t|s_t)}{\pi_{\theta_{\text{ref}}}(a_t|s_t)}, 1 - \epsilon, 1 + \epsilon\right)$ 将比值函数约束在范围 $[1 - \epsilon, 1 + \epsilon]$ 内。

进一步改进上述模型是在优化中考虑信任区域 [Schulman 等人, 2015]。在强化学习中，策略的大幅更新可能导致不稳定性，其中代理在更新后可能开始表现得更差。一个合理的想法是在信任区域内优化模型，这指的是围绕当前参数估计的模型表现良好的区域。将信任区域纳入的一种方法是对策略更新的规模施加约束，确保当前策略不会与参考策略显著偏离。这可以通过向目标函数添加基于当前策略与参考策略之间某种形式的差异的惩罚来实现。这种惩罚的简单形式是当前策略与参考策略下序列 τ 的对数概率之差：

$$\text{Penalty} = \log \pi_{\theta}(\tau) - \log \pi_{\theta_{\text{ref}}}(\tau) \quad (4.51)$$

实际上，这个惩罚可以通过只考虑策略概率而忽略动态来近似。这给出

$$\text{Penalty} = \sum_{t=1}^T \log \pi_{\theta}(a_t|s_t) - \sum_{t=1}^T \log \pi_{\theta_{\text{ref}}}(a_t|s_t) \quad (4.52)$$

通过将此惩罚包含在优化目标中，我们鼓励当前策略保持接近参考策略，限制可能导致学习不稳定的大幅更新。

我们可以将这个惩罚项纳入截断代理目标函数中，并得到

$$U_{\text{ppo-clip}}(\tau; \theta) = U_{\text{clip}}(\tau; \theta) - \beta \text{Penalty} \quad (4.53)$$

β 是惩罚的权重。这种训练方法称为近端策略优化（PPO），它是 LLMs 和其他许多领域中使用的最受欢迎的强化学习方法之一 [Schulman 等, 2017]。

现在我们可以将训练 LLMs 的目标写成 PPO 的形式。

the parameter update at the point $\theta = \theta_{\text{ref}}$ on the optimization surface.

$$U(\mathbf{x}, \mathbf{y}; \theta) = U_{\text{ppo-clip}}(\mathbf{x}, \mathbf{y}; \theta) - \beta \text{Penalty} \quad (4.54)$$

哪里

$$U_{\text{ppo-clip}}(\mathbf{x}, \mathbf{y}; \theta) = \sum_{t=1}^T \text{Clip}\left(\frac{\pi_{\theta}(y_t|\mathbf{x}, \mathbf{y}_{<t})}{\pi_{\theta_{\text{ref}}}(y_t|\mathbf{x}, \mathbf{y}_{<t})}\right) A(\mathbf{x}, \mathbf{y}_{<t}, y_t) \quad (4.55)$$

$$\begin{aligned} \text{Penalty} &= \log \Pr_{\theta}(\mathbf{y}|\mathbf{x}) - \log \Pr_{\theta_{\text{ref}}}(\mathbf{y}|\mathbf{x}) \\ &= \sum_{t=1}^T \log \Pr_{\theta}(y_t|\mathbf{x}, \mathbf{y}_{<t}) - \sum_{t=1}^T \log \Pr_{\theta_{\text{ref}}}(y_t|\mathbf{x}, \mathbf{y}_{<t}) \end{aligned} \quad (4.56)$$

尽管这里的符号看起来有点繁琐，但PPO的想法很简单：我们通过结合目标策略和参考策略的裁剪似然比以及优势函数来开发一个目标，然后施加一个惩罚，以确保策略更新不会太大。基于PPO的RLHF在图4.9中展示。

总结来说，实现RLHF需要构建四个模型，所有模型均基于Transformer解码器架构。

- 奖励模型 ($r_{\phi}(\cdot)$ 其中 ϕ 表示参数)。奖励模型从人类偏好数据中学习，以预测每对输入和输出标记序列的奖励。它是一个Transformer解码器，后面跟着一个线性层，将序列（输入和输出的拼接）映射到实值奖励分数。
- 价值模型或价值函数 ($V_{\omega}(\cdot)$ ，其中 ω 表示参数)。价值函数接收奖励模型的奖励分数，并训练预测从状态开始可以获得的总奖励的期望值。它通常与奖励模型具有相同的架构。
- 参考模型 ($\pi_{\theta_{\text{ref}}}(\cdot) = \Pr_{\theta_{\text{ref}}}(\cdot)$ ，其中 θ_{ref} 表示参数)。参考模型是作为策略训练起点的基线LLM。在RLHF中，它代表模型的先前版本或未接受人类反馈训练的模型。它用于在输出空间中进行采样，并有助于策略训练的损失计算。
- 目标模型或策略 ($\pi_{\theta}(\cdot) = \Pr_{\theta}(\cdot)$ 其中 θ 表示参数)。此策略规定了LLM在给定上下文的情况下如何决定最合适的下一个标记。它在奖励模型和价值模型的监督下进行训练。

实际上，这些模型需要按照一定顺序进行训练。首先，我们需要使用其他模型来初始化它们。例如，奖励模型和价值模型可以使用预训练的LLM进行初始化，而参考模型和目标模型可以使用经过指令微调的模型进行初始化。请注意，在此阶段，参考模型已准备好使用，将不再进行更新。其次，我们需要收集人类偏好数据，并在此数据上训练奖励模型。第三，我们同时使用奖励模型训练价值模型和政策。在每个输出标记序列的位置，我们更新价值模型

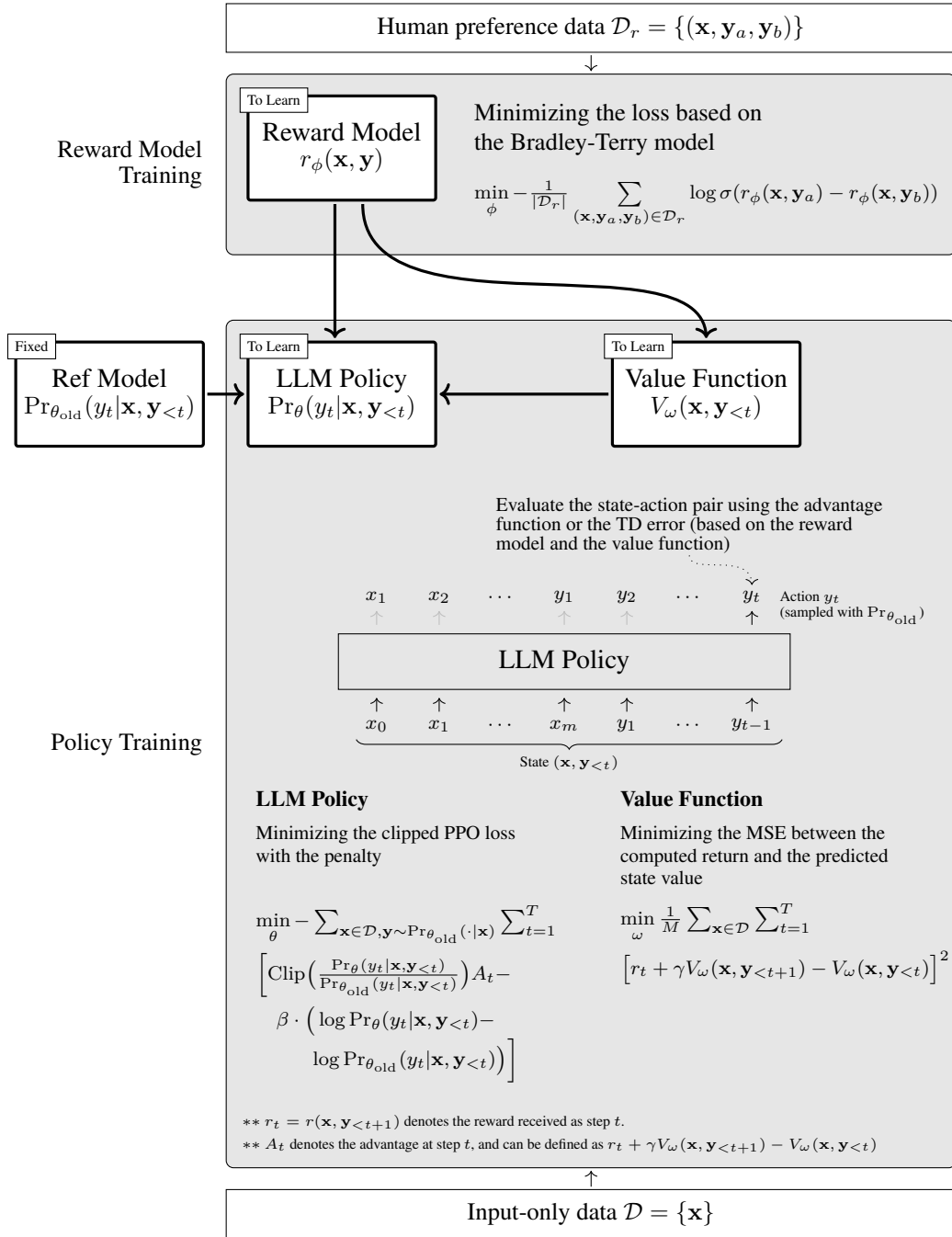


图4.9: RLHF的说明。第一步是收集人类偏好数据并使用这些数据训练奖励模型。一旦奖励模型优化完成, 连同参考模型, 我们继续训练策略和价值函数。在每一步预测中, 我们计算基于PPO的损失总和并更新策略的参数。这需要访问奖励模型、参考模型和价值函数。同时, 我们通过最小化均方误差损失来更新价值函数的参数。

通过最小化值预测的均方误差 (MSE) 误差, 并且通过最小化PPO损失来更新策略。

4.4 改进的人性化偏好对齐

在上一节中，我们回顾了强化学习的基本概念和RLHF的一般框架。在本节中，我们将讨论RLHF的一些改进和实现人类偏好对齐的替代方法。

4.4.1 更好的奖励建模

在4.3.2节中，我们强调了从人类偏好中学习以及使用成对排名损失来训练奖励模型的任务。在这里，我们考虑更多奖励建模的方法。我们的讨论将相对通用，由于奖励模型在许多强化学习问题中得到广泛应用，因此我们将很容易将此处讨论的方法应用于RLHF和相关应用。

4.4.1.1 监督信号

训练奖励模型可以广泛地被视为一个排序问题，其中模型学习为输出分配分数，以便它们的顺序反映人类所指示的偏好。从排序的角度来看，有几种方法来训练奖励模型。

一种方法是将成对排名扩展到列表排名。对于数据集中的每个样本，我们可以使用LLM生成多个输出，并请人类专家对这些输出进行排序。例如，给定一组四个输出 $\{y_1, y_2, y_3, y_4\}$ ，它们的一个可能的顺序可以是 $y_2 \succ y_3 \succ y_1 \succ y_4$ 。对列表排序的一个非常简单的方法是累积成对比较损失。例如，我们可以通过累积所有输出对的损失来定义列表损失：

$$\mathcal{L}_{\text{list}} = -\mathbb{E}_{(\mathbf{x}, Y) \sim \mathcal{D}_r} \left[\frac{1}{N(N-1)} \sum_{\substack{y_a \in Y, y_b \in Y \\ y_a \neq y_b}} \log \Pr(y_a \succ y_b | \mathbf{x}) \right] \quad (4.57)$$

在 Y 是输出列表，且 N 是列表中的输出数量。 $\Pr(y_a \succ y_b | \mathbf{x})$ 可以使用Bradley-Terry模型定义，即 $\Pr(y_a \succ y_b | \mathbf{x}) = \text{Sigmoid}(r(\mathbf{x}, y_a) - r(\mathbf{x}, y_b))$ 。在这里，我们省略了 $\Pr(\cdot)$ 上的 ϕ 上标以保持符号简洁。

一种扩展到Bradley-Terry模型的列表排序方法可能涉及一种考虑整个输出列表而不是仅考虑成对比较的排序机制。其中一个这样的模型是Plackett-Luce模型，它将Bradley-Terry模型推广到处理排名中的多个项目[Plackett, 1975]。在Plackett-Luce模型中，对于列表中的每个项目，我们定义一个反映该项目相对于其他项目被选择的相对强度的“价值”。对于这里的奖励建模问题，列表 Y 中 y 的价值可以定义为

$$\alpha(y) = \exp(r(\mathbf{x}, y)) \quad (4.58)$$

然后从 Y 中选择 y 的概率由以下公式给出

$$\begin{aligned}\Pr(y \text{ is selected} | \mathbf{x}, Y) &= \frac{\alpha(y)}{\sum_{y' \in Y} \alpha(y')} \\ &= \frac{\exp(r(\mathbf{x}, y))}{\sum_{y' \in Y} \exp(r(\mathbf{x}, y'))}\end{aligned}\quad (4.59)$$

假设 \dot{Y} 是一个有序列表 $y_{j_1} \succ y_{j_2} \succ \cdots \succ y_{j_N}$ 。这个有序列表的整体对数概率可以定义为在每一步选择阶段的条件对数概率之和，给出如下

$$\begin{aligned}\log \Pr(\dot{Y} | \mathbf{x}) &= \log \Pr(y_{j_1} \succ y_{j_2} \succ \cdots \succ y_{j_N} | \mathbf{x}) \\ &= \log \Pr(y_{j_1} | \mathbf{x}, \{y_{j_1}, y_{j_2}, \dots, y_{j_N}\}) + \\ &\quad \log \Pr(y_{j_2} | \mathbf{x}, \{y_{j_2}, \dots, y_{j_N}\}) + \\ &\quad \cdots + \\ &\quad \log \Pr(y_{j_N} | \mathbf{x}, \{y_{j_N}\}) \\ &= \sum_{k=1}^N \log \Pr(y_{j_k} | \mathbf{x}, \dot{Y}_{\geq k})\end{aligned}\quad (4.60)$$

在 $\dot{Y}_{\geq k}$ 表示在 k -th 阶段未选择的输出列表的子集，即 $\dot{Y}_{\geq k} = \{y_{j_k}, \dots, y_{j_N}\}$ 。给定对数概率 $\log \Pr(\dot{Y} | \mathbf{x})$ ，我们可以根据 Plackett-Luce 模型定义损失函数，通过

$$\mathcal{L}_{\text{pl}} = -\mathbb{E}_{(\mathbf{x}, \dot{Y}) \sim \mathcal{D}_r} [\log \Pr(\dot{Y} | \mathbf{x})] \quad (4.61)$$

存在许多其他成对和列表方法来建模排名，例如 RankNet [Burgess 等人, 2005] 和 ListNet [Cao 等人, 2007]。所有这些方法都可以归类为学习排名的大类，其中大多数适用于建模人类偏好的问题。然而，讨论这些方法超出了本章的范围。感兴趣的读者可以参考有关此主题的书籍以获取更多详细信息 [刘, 2009; 李, 2011]。

除了成对和列表排名之外，使用点wise方法训练奖励模型为捕捉人类偏好提供了一种替代方法。与关注不同输出之间相对排名的方法不同，pointwise方法独立处理每个输出。例如，人类专家可能会为单个输出分配一个分数，如五点评分。目标是调整奖励模型，使其输出与这些分数一致。通过回归技术实现pointwise训练是一种简单的方法，其中每个输出的奖励被视为目标变量。令 $\varphi(\mathbf{x}, y)$ 为人类根据 \mathbf{x} 分配给 y 的分数。可以通过最小化预测奖励 $r(\mathbf{x}, y)$ 与实际人类反馈 $\varphi(\mathbf{x}, y)$ 之间的损失函数来训练pointwise奖励模型，通常基于均方误差或其他回归损失。例如，损失函数可以是

$$\mathcal{L}_{\text{point}} = -\mathbb{E} [\varphi(\mathbf{x}, y) - r(\mathbf{x}, y)]^2 \quad (4.62)$$

虽然逐点方法在概念上更简单，可以直接指导奖励模型到

预测分数，它们在RLHF中不一定总是最佳选择。问题是这些方法可能难以处理人类反馈的高方差，尤其是在不同专家对类似输出提供不一致分数时。因为它们侧重于拟合绝对分数而不是相对差异，评分的不一致性可能导致模型性能不佳。此外，拟合特定评分输出可能会阻碍泛化，尤其是在RLHF中训练数据通常非常有限的情况下。相比之下，考虑相对偏好的方法可以促进更普遍的成功和失败模式的学习。尽管如此，在某些情况下，点值方法可能仍然适用。例如，在训练数据丰富且获取准确、一致注释的成本低的情况下，点值方法可以证明是有效的。

实际上，为了使训练奖励模型的监督信号更加鲁棒，我们还可以在训练引入额外的正则化项。例如，如果我们把方程(4.54)中的第一项 $U_{\text{ppo-clip}}(\mathbf{x}, \mathbf{y}; \theta)$ 视为一种广义奖励，那么第二项（即惩罚项）可以看作是奖励模型的一种正则化形式，只不过这里的目的是训练策略而不是奖励模型。另一个例子是，Eisenstein等人[2023]基于奖励的平方和开发了一个正则化项，并将其添加到RLHF中的成对比较损失中：

$$\begin{aligned}\mathcal{L}_{\text{reg}} &= \mathcal{L}_{\text{pair}} + (-\mathbb{E}_{(\mathbf{x}, \mathbf{y}_a, \mathbf{y}_b) \sim \mathcal{D}_r} [r(\mathbf{x}, \mathbf{y}_a) + r(\mathbf{x}, \mathbf{y}_b)]^2) \\ &= -\mathbb{E}_{(\mathbf{x}, \mathbf{y}_a, \mathbf{y}_b) \sim \mathcal{D}_r} [\log \Pr_{\phi}(\mathbf{y}_a \succ \mathbf{y}_b | \mathbf{x})] \\ &\quad - \mathbb{E}_{(\mathbf{x}, \mathbf{y}_a, \mathbf{y}_b) \sim \mathcal{D}_r} [r(\mathbf{x}, \mathbf{y}_a) + r(\mathbf{x}, \mathbf{y}_b)]^2\end{aligned}\tag{4.63}$$

通过此正则化项进行优化可以帮助缓解奖励模型的欠确定性问题⁵。

4.4.1.2 稀疏奖励与密集奖励

如第4.3节所述，RLHF中的奖励非常稀疏：它们仅在序列末尾观察到，而不是在整个生成过程中连续观察到。处理稀疏奖励一直是强化学习中的一个关注点，也是许多实际应用中的挑战之一。例如，在机器人领域，通常需要调整奖励函数以简化优化，而不是仅仅依赖于序列末尾的奖励。已经开发出各种方法来解决这个问题。一种常见的方法是奖励塑造，其中原始函数被修改以包括中间奖励，从而提供更及时的反馈。此外，还可以采用课程学习来按顺序构建任务，使复杂性逐渐增加。这可以帮助模型首先掌握简单任务，为它们技能发展过程中的更复杂挑战做好准备。有许多这样的方法可以减轻稀疏奖励的影响，例如蒙特卡洛方法和内在动机。大多数这些方法都是通用的，关于它们的讨论可以在强化学习的更广泛文献中找到，例如Sutton和Barto [2018]的书籍。

尽管我们在此处没有详细讨论缓解稀疏奖励的方法，但一个有趣的问题出现了：为什么稀疏奖励在RLHF中如此成功？回想第4.3.1节，每个时间步收到的监督信号 t 并不是当前动作的奖励，而是

⁵A model is called underdetermined if there are multiple alternative sets of parameters that can achieve the same objective.

相当于是从 t 到最后一个时间步的累积奖励的形式。这种监督信号在序列上是密集的，因为序列末尾获得的奖励可以回传到那个时间步，无论它是什么时间步。换句话说，稀疏奖励被转换成了密集的监督信号。此外，从奖励塑造的角度来看，Ng 等人 [1999] 表明， t 处的奖励可以定义为

$$r'(s_t, a_t, s_{t+1}) = r(s_t, a_t, s_{t+1}) + f(s_t, a_t, s_{t+1}) \quad (4.64)$$

$r'(\cdot)$ 是转换后的奖励函数， $r(\cdot)$ 是原始奖励函数， $f(\cdot)$ 是塑造奖励函数。为确保在转换后的奖励函数下策略的最优性，塑造奖励函数可以表示为

$$f(s_t, a_t, s_{t+1}) = \gamma \Phi(s_{t+1}) - \Phi(s_t) \quad (4.65)$$

$\Phi(s)$ 被称为状态 s 的势值。如果我们定义 $\Phi(s)$ 为共同值函数，如公式 (4.15) 所示，并将公式 (4.65) 代入公式 (4.64)，我们得到

$$r'(s_t, a_t, s_{t+1}) = r(s_t, a_t, s_{t+1}) + \gamma V(s_{t+1}) - V(s_t) \quad (4.66)$$

有趣的是，这个函数与 PPO 中使用的优势函数完全相同。这将基于优势的方法与奖励塑造联系起来：优势本质上是一种塑造的奖励。

另一方面，采用序列末尾奖励的一个原因在于 RLHF 任务的本质。与代理与动态环境交互的传统强化学习环境不同，RLHF 任务通常涉及基于语言或其他高级认知过程的复杂决策。这些过程不易于频繁且有意义的中间奖励，因为动作的质量和适宜性只能在观察其在整个序列或任务更大背景中的影响后才能完全评估。在这种情况下，基于人类反馈的奖励信号，尽管非常稀疏，但通常非常具有信息性和准确性。因此，这种稀疏性，加上人类反馈的高度信息性和准确性，可以使学习既稳健又高效。

4.4.1.3 精细奖励

对于许多应用，我们的目标将比仅仅评估整个文本更为复杂。例如，在情感分析中，我们通常不仅确定文本的情感，还需要通过将其与文本中讨论的主题的特定方面相关联来更详细地分析情感。考虑以下句子：“手机的相机非常好，但电池续航令人失望。”在这个例子中，我们需要分别分析关于相机和电池所表达的情感。这种分析，称为基于方面的情感分析，有助于比一般情感分析提供更细致的客户评论理解。

对于奖励建模问题，我们通常还需要对序列的不同部分进行建模。一种简单直接的方法是将序列划分为不同的段，然后计算每个段的奖励 [Wu 等人, 2023b]。假设一个输出标记

序列 y 可以根据某些标准分为 n_s 个段 $\{\bar{y}_1, \dots, \bar{y}_{n_s}\}$ 。我们可以使用奖励模型来评估这些段中的每一个。通过将 x 、 y 和 \bar{y}_k 作为输入到奖励模型中，第 k 个段的奖励分数由以下公式给出

$$r^k = r(\mathbf{x}, \mathbf{y}, \bar{\mathbf{y}}_k) \quad (4.67)$$

然后整个输出序列的奖励分数由以下公式给出

$$r(\mathbf{x}, \mathbf{y}) = \sum_{k=1}^{n_s} r(\mathbf{x}, \mathbf{y}, \bar{\mathbf{y}}_k) \quad (4.68)$$

这里 $r(\mathbf{x}, \mathbf{y})$ 可以像往常一样用于训练策略。

此模型的问题在于，在段落级别训练奖励模型不如从整个文本的人类偏好中学习那么直接，因为难以获得段落级别的人类偏好数据。对于类似评分的问题（例如，我们根据其虚假信息的程度对段落进行评分），一个简单的方法是为每个段落分配一个评分，并使用点积方法训练奖励模型。例如，我们可以使用一个强大的LLM来评分序列 $\bar{y}_1 \dots \bar{y}_{k-1}$ 和 $\bar{y}_1 \dots \bar{y}_k$ ，并获得评分 $s(\bar{y}_1 \dots \bar{y}_{k-1})$ 和 $s(\bar{y}_1 \dots \bar{y}_k)$ 。然后我们可以定义段落 \bar{y}_k 的评分为 $s(\bar{y}_1 \dots \bar{y}_k)$ 和 $s(\bar{y}_1 \dots \bar{y}_{k-1})$ 之间的差值

$$s(\bar{\mathbf{y}}_k) = s(\bar{\mathbf{y}}_1 \dots \bar{\mathbf{y}}_k) - s(\bar{\mathbf{y}}_1 \dots \bar{\mathbf{y}}_{k-1}) \quad (4.69)$$

使用这些段级别得分，我们可以使用回归损失函数训练奖励模型

$$\mathcal{L}_{\text{rating}} = -\mathbb{E}_{\bar{\mathbf{y}}_k} [s(\bar{\mathbf{y}}_k) - r(\mathbf{x}, \mathbf{y}, \bar{\mathbf{y}}_k)]^2 \quad (4.70)$$

有时，对齐可以被视为一个分类问题，例如，我们评估一个片段是否存在道德问题。在这种情况下，片段可以被标记为道德或不道德，无论是通过人类还是使用额外的分类器。给定片段的标签，我们可以使用某些分类损失函数来训练奖励模型。例如，假设当片段被分类为不道德时， $r(\mathbf{x}, \mathbf{y}, \bar{\mathbf{y}}_k) = 1$ ，否则 $r(\mathbf{x}, \mathbf{y}, \bar{\mathbf{y}}_k) = -1$ ⁶。训练二元分类模型的 hinge 损失由以下公式给出

$$\mathcal{L}_{\text{hinge}} = \max(0, 1 - r(\mathbf{x}, \mathbf{y}, \bar{\mathbf{y}}_k) \cdot \hat{r}) \quad (4.71)$$

$\hat{r} \in \{1, -1\}$ 表示真实标签。

剩余问题在于如何将 y 分割成段。一种方法是定义固定长度的分割，将 y 分成等长块。然而，这并不总是理想的，因为序列的内容可能无法很好地与固定边界对齐。另一种方法是根据特定的语言或语义线索对 y 进行分割，例如句子边界、主题转换或文本中的其他有意义的结构。这种分割可以通过使用语言分割系统或提示LLMs识别序列中的自然断点来实现。另一种方法是使用基于复杂性的动态分割方法。

⁶To allow the reward model to output categories, we can replace the linear layer described in Section 4.3.2 with a Softmax layer.

序列。例如，可以在奖励分数发生显著变化的地方定义段，这可能对应于正在建模的任务的变化。

4.4.1.4 奖励模型组合

奖励模型可以被视为环境的代理。由于真实环境通常过于复杂或未知，因此开发一个完美的环境代理通常是不可能的。因此，过度调整LLMs与这个不完美的代理可能会导致性能下降，即过度优化问题[Stiennon等人, 2020; Gao等人, 2023a]⁷。我们也可以通过Goodhart定律来解释这一点，该定律指出：*when a measure becomes a target, it ceases to be a good measure* [Goodhart, 1984]。

解决过度优化问题不容易，目前还没有成熟的解决方案。理想的方法可能是开发一个能够完美捕捉任务真实目标的预言奖励模型，并防止代理“欺骗”。然而，由于现实世界环境的复杂性和定义所有有助于期望结果的相关因素的挑战，创建这样的模型极其困难。相反，更实际的方法是结合多个奖励模型，从而缓解使用单一特定奖励模型时出现的训练目标与真实目标之间的不匹配[Coste等人, 2024]。

给定一组奖励模型，将它们组合是直接的，在某些情况下，我们可以简单地将这个问题视为集成学习问题。一种简单但常见的方法是对这些模型的输出进行平均，以获得更精确的奖励估计：

$$r_{\text{combine}} = \frac{1}{K} \sum_{k=1}^K w_k \cdot r_k(\mathbf{x}, \mathbf{y}) \quad (4.72)$$

在集成中， $r_k(\cdot)$ 是第 k 个奖励模型， w_k 是 $r_k(\cdot)$ 的权重， K 是奖励模型的数量。这种组合奖励可以用来监督策略的训练。实际上，有许多种方法来组合不同的模型，例如，可以使用贝叶斯模型平均或开发一个融合网络来学习如何结合不同模型的预测。或者，可以将这个任务构造成一个多目标优化问题，并使用多个奖励模型同时训练策略。这些方法已经在优化和机器学习的文献中进行了广泛讨论[Miettinen, 1999; Bishop, 2006]。

除了模型组合方法之外，另一个重要问题是如何收集或构建多个不同的奖励模型。其中一种最简单的方法是采用集成学习方法，例如从给定数据集的不同子集或各种数据源中开发不同的奖励模型。对于RLHF，也可以根据对对齐的不同方面的考虑来构建奖励模型。例如，我们可以开发一个奖励模型来评估输出的事实准确性，另一个奖励模型来评估

⁷This problem is also called **reward hacking** or **reward gaming** [Krakovna et al., 2020; Skalse et al., 2022; Pan et al., 2022], which refers to the phenomenon where the agent attempts to trick the reward model but fails to align its actions with the true intended objectives of the task. Imagine a student who is assigned homework and is rewarded with points or praise for completing it. The student might then find ways to finish the homework with minimal effort to maximize the reward, such as copying and pasting solutions from the internet or previous assignments, rather than solving the problems themselves.

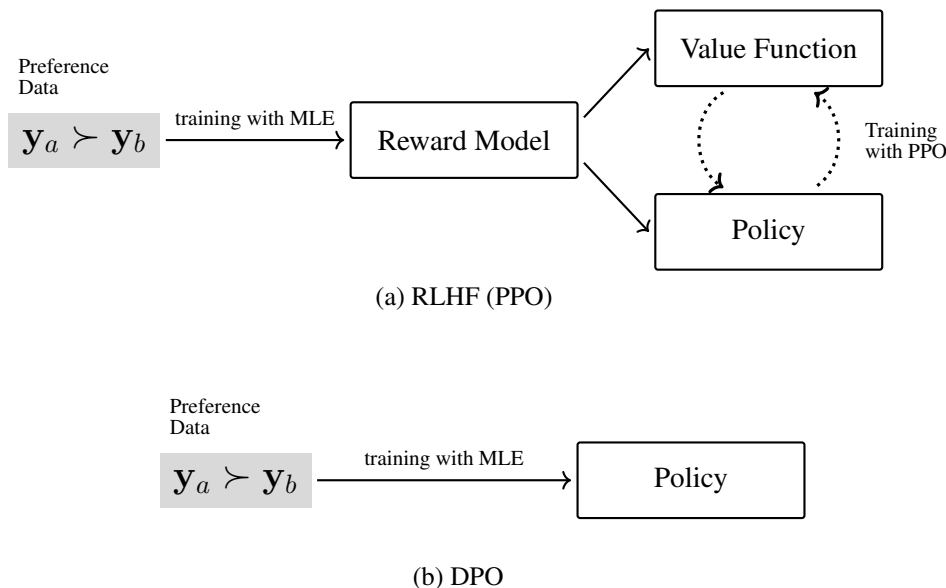


图4.10: 标准RLHF (PPO) 与DPO。在RLHF中, 使用人类偏好数据来训练奖励模型, 然后将其用于训练策略以及价值函数。在DPO中, 人类偏好数据的利用更为直接, 策略是在此数据上训练, 无需进行奖励模型训练。

输出完整度。这两个模型相互补充, 可以结合使用以提高输出的整体评估。另一种方法是采用不同的现成LLM作为奖励模型。这种方法简单实用, 因为已经有很多成熟的LLM, 我们只需使用它们, 无需或只需进行少量修改。虽然这个问题与这里的讨论不太相关, 但有一个有趣的问题: 一个与其他LLM保持一致的LLM能否超越这些LLM? 乍一看可能不行。部分原因在于目标LLM仅仅基于有限的监督模仿其他LLM, 因此无法很好地捕捉这些监督者的行为细微差别。然而, 鉴于LLM强大的泛化能力, 这种方法实际上可能非常有益。例如, 使用开源或商业LLM作为奖励模型已在对齐LLM方面显示出强大的性能, 甚至在几个流行任务上实现了最先进的结果 [Lambert等人, 2024]。

4.4.2 直接偏好优化

尽管学习奖励模型是强化学习中的一个标准步骤, 但它使得整个训练过程比监督训练复杂得多。训练一个可靠的奖励模型本身就不是一件容易的任务, 一个训练不当的奖励模型会极大地影响策略学习的成果。我们现在考虑一种替代的对齐方法, 称为直接偏好优化 (DPO), 通过消除显式建模奖励的需要来简化训练框架 [Rafailov 等人, 2024]。这种方法直接根据用户偏好优化策略, 而不是开发一个单独的奖励模型。因此, 我们可以以类似于监督学习的方式实现人类偏好的对齐。图 4.10 展示了标准 RLHF 方法和 DPO 方法的比较。

在推导DPO目标之前, 让我们首先回顾RLHF中使用的策略训练目标。如第4.3.3节所述, 策略通常通过优化损失函数进行训练

具有惩罚项。DPO方法假设一个简单的损失函数，其中输出 y 的质量在给定输入 x 的情况下通过奖励模型 $r(x, y)$ 进行评估。因此，训练目标如下

$$\tilde{\theta} = \arg \min_{\theta} \mathbb{E}_{\mathbf{x} \sim \mathcal{D}} \mathbb{E}_{\mathbf{y} \sim \pi_{\theta}(\cdot|\mathbf{x})} \left[\underbrace{-r(\mathbf{x}, \mathbf{y})}_{\text{loss}} + \beta \underbrace{(\log \pi_{\theta}(\mathbf{y}|\mathbf{x}) - \log \pi_{\theta_{\text{ref}}}(\mathbf{y}|\mathbf{x}))}_{\text{penalty}} \right] \quad (4.73)$$

注意，在这个优化问题中，只有项 $\pi_{\theta}(\mathbf{y}|\mathbf{x})$ 依赖于目标策略 $\pi_{\theta}(\cdot)$ 。奖励模型 $r(\mathbf{x}, \mathbf{y})$ 和参考模型 $\pi_{\theta_{\text{ref}}}(\mathbf{y}|\mathbf{x})$ 假设给定 x 和 y 后是固定的。与 PPO 相比，这是一个较强的假设，但如后文所示，它简化了问题，对于推导 DPO 目标至关重要。

由于 θ 是我们想要优化的变量，我们将方程 (4.73) 的右侧重新排列，以将 $\pi_{\theta}(\mathbf{y}|\mathbf{x})$ 作为独立项隔离：

$$\begin{aligned} \tilde{\theta} &= \arg \min_{\theta} \mathbb{E}_{\mathbf{x} \sim \mathcal{D}} \mathbb{E}_{\mathbf{y} \sim \pi_{\theta}(\cdot|\mathbf{x})} [\beta \log \pi_{\theta}(\mathbf{y}|\mathbf{x}) - \beta \log \pi_{\theta_{\text{ref}}}(\mathbf{y}|\mathbf{x}) - r(\mathbf{x}, \mathbf{y})] \\ &= \arg \min_{\theta} \mathbb{E}_{\mathbf{x} \sim \mathcal{D}} \mathbb{E}_{\mathbf{y} \sim \pi_{\theta}(\cdot|\mathbf{x})} [\log \pi_{\theta}(\mathbf{y}|\mathbf{x}) - (\log \pi_{\theta_{\text{ref}}}(\mathbf{y}|\mathbf{x}) + \frac{1}{\beta} r(\mathbf{x}, \mathbf{y}))] \\ &= \arg \min_{\theta} \mathbb{E}_{\mathbf{x} \sim \mathcal{D}} \mathbb{E}_{\mathbf{y} \sim \pi_{\theta}(\cdot|\mathbf{x})} \left[\underbrace{\log \pi_{\theta}(\mathbf{y}|\mathbf{x})}_{\text{dependent on } \theta} - \underbrace{\log \pi_{\theta_{\text{ref}}}(\mathbf{y}|\mathbf{x}) \exp(\frac{1}{\beta} r(\mathbf{x}, \mathbf{y}))}_{\text{not dependent on } \theta} \right] \quad (4.74) \end{aligned}$$

这个方程定义目标函数为 y 的对数概率分布函数与另一个 y 的函数之间的差异。这种目标函数的形式似乎并不“理想”，因为我们通常更喜欢看到两个分布之间的差异，这样我们可以将这种差异解释为分布之间的一种某种发散。一个简单的想法是将第二个项（即 $\log \pi_{\theta_{\text{ref}}}(\mathbf{y}|\mathbf{x}) \exp(\frac{1}{\beta} r(\mathbf{x}, \mathbf{y}))$ ）转换为 y 的域上的对数概率分布。如果我们把 $\pi_{\theta_{\text{ref}}}(\mathbf{y}|\mathbf{x}) \exp(\frac{1}{\beta} r(\mathbf{x}, \mathbf{y}))$ 当作 y 的未归一化概率，我们可以通过除以归一化因子将其转换为归一化概率：

$$Z(\mathbf{x}) = \sum_{\mathbf{y}} \pi_{\theta_{\text{ref}}}(\mathbf{y}|\mathbf{x}) \exp(\frac{1}{\beta} r(\mathbf{x}, \mathbf{y})) \quad (4.75)$$

因此，我们可以通过以下方式定义一个概率分布：

$$\pi^*(\mathbf{y}|\mathbf{x}) = \frac{\pi_{\theta_{\text{ref}}}(\mathbf{y}|\mathbf{x}) \exp(\frac{1}{\beta} r(\mathbf{x}, \mathbf{y}))}{Z(\mathbf{x})} \quad (4.76)$$

我们然后将公式 (4.74) 重写为

$$\begin{aligned}
\tilde{\theta} &= \arg \min_{\theta} \mathbb{E}_{\mathbf{x} \sim \mathcal{D}} \mathbb{E}_{\mathbf{y} \sim \pi_{\theta}(\cdot|\mathbf{x})} \left[\log \pi_{\theta}(\mathbf{y}|\mathbf{x}) - \log \frac{\pi_{\theta_{\text{ref}}}(\mathbf{y}|\mathbf{x}) \exp\left(\frac{1}{\beta} r(\mathbf{x}, \mathbf{y})\right)}{Z(\mathbf{x})} \right. \\
&\quad \left. - \log Z(\mathbf{x}) \right] \\
&= \arg \min_{\theta} \mathbb{E}_{\mathbf{x} \sim \mathcal{D}} \mathbb{E}_{\mathbf{y} \sim \pi_{\theta}(\cdot|\mathbf{x})} \left[\log \pi_{\theta}(\mathbf{y}|\mathbf{x}) - \log \pi^*(\mathbf{y}|\mathbf{x}) - \log Z(\mathbf{x}) \right] \\
&= \arg \min_{\theta} \mathbb{E}_{\mathbf{x} \sim \mathcal{D}} \left[\mathbb{E}_{\mathbf{y} \sim \pi_{\theta}(\cdot|\mathbf{x})} \left[\log \pi_{\theta}(\mathbf{y}|\mathbf{x}) - \log \pi^*(\mathbf{y}|\mathbf{x}) \right] \right. \\
&\quad \left. - \mathbb{E}_{\mathbf{y} \sim \pi_{\theta}(\cdot|\mathbf{x})} [\log Z(\mathbf{x})] \right] \\
&= \arg \min_{\theta} \mathbb{E}_{\mathbf{x} \sim \mathcal{D}} \left[\underbrace{\text{KL}(\pi_{\theta}(\cdot|\mathbf{x}) \parallel \pi^*(\cdot|\mathbf{x}))}_{\text{KL divergence}} - \underbrace{\log Z(\mathbf{x})}_{\text{constant wrt. } \theta} \right] \tag{4.77}
\end{aligned}$$

由于对数 $Z(\mathbf{x})$ 与 θ 无关，它不影响 $\arg \min_{\theta}$ 操作的结果，可以从中移除。现在我们获得了一个新的训练目标，通过最小化 $\pi_{\theta}(\cdot|\mathbf{x})$ 和 $\pi^*(\cdot|\mathbf{x})$ 之间的 KL 散度来找到最优策略 π_{θ} 。

$$\tilde{\theta} = \arg \min_{\theta} \mathbb{E}_{\mathbf{x} \sim \mathcal{D}} [\text{KL}(\pi_{\theta}(\cdot|\mathbf{x}) \parallel \pi^*(\cdot|\mathbf{x}))] \tag{4.78}$$

显然，此优化问题的解由以下公式给出 $\{\mathbf{v}^*\}$

$$\begin{aligned}
\pi_{\theta}(\mathbf{y}|\mathbf{x}) &= \pi^*(\mathbf{y}|\mathbf{x}) \\
&= \frac{\pi_{\theta_{\text{ref}}}(\mathbf{y}|\mathbf{x}) \exp\left(\frac{1}{\beta} r(\mathbf{x}, \mathbf{y})\right)}{Z(\mathbf{x})} \tag{4.79}
\end{aligned}$$

给定此方程，我们可以使用目标模型 $\pi_{\theta}(\mathbf{y}|\mathbf{x})$ 、参考模型 $\pi_{\theta_{\text{ref}}}(\mathbf{y}|\mathbf{x})$ 和归一化因子 $Z(\mathbf{x})$ 来表示奖励 $r(\mathbf{x}, \mathbf{y})$ ：

$$r(\mathbf{x}, \mathbf{y}) = \beta \left(\log \frac{\pi_{\theta}(\mathbf{y}|\mathbf{x})}{\pi_{\theta_{\text{ref}}}(\mathbf{y}|\mathbf{x})} + \log Z(\mathbf{x}) \right) \tag{4.80}$$

这是有趣的，因为我们最初试图使用奖励模型 $r(\mathbf{x}, \mathbf{y})$ 学习策略 $\pi_{\theta}(\cdot)$ ，但最终获得了一个基于策略的奖励模型表示。根据定义在公式 (4.80) 中的奖励模型，我们可以将其应用于 Bradley-Terry 模型来计算偏好概率（也可参见第 4.3.2 节）：

$$\begin{aligned}
\Pr_{\theta}(\mathbf{y}_a \succ \mathbf{y}_b|\mathbf{x}) &= \text{Sigmoid}(r(\mathbf{x}, \mathbf{y}_a) - r(\mathbf{x}, \mathbf{y}_b)) \\
&= \text{Sigmoid} \left(\beta \left(\log \frac{\pi_{\theta}(\mathbf{y}_a|\mathbf{x})}{\pi_{\theta_{\text{ref}}}(\mathbf{y}_a|\mathbf{x})} + \log Z(\mathbf{x}) \right) - \right. \\
&\quad \left. \beta \left(\log \frac{\pi_{\theta}(\mathbf{y}_b|\mathbf{x})}{\pi_{\theta_{\text{ref}}}(\mathbf{y}_b|\mathbf{x})} + \log Z(\mathbf{x}) \right) \right) \\
&= \text{Sigmoid} \left(\beta \log \frac{\pi_{\theta}(\mathbf{y}_a|\mathbf{x})}{\pi_{\theta_{\text{ref}}}(\mathbf{y}_a|\mathbf{x})} - \beta \log \frac{\pi_{\theta}(\mathbf{y}_b|\mathbf{x})}{\pi_{\theta_{\text{ref}}}(\mathbf{y}_b|\mathbf{x})} \right) \tag{4.81}
\end{aligned}$$

这个公式很优雅，因为它将奖励差异转换为比率函数的差异，我们不需要计算 $Z(\mathbf{x})$ 的值。直接结果是，我们不再需要奖励模型，只需要目标策略和参考模型来计算偏好概率。最后，我们可以通过最小化以下 DPO 损失函数来训练目标策略

$$\mathcal{L}_{\text{dpo}}(\theta) = -\mathbb{E}_{(\mathbf{x}, \mathbf{y}_a, \mathbf{y}_b) \sim \mathcal{D}_r} [\log \Pr_{\theta}(\mathbf{y}_a \succ \mathbf{y}_b | \mathbf{x})] \quad (4.82)$$

该损失函数的形式与 RLHF 中用于训练奖励模型的函数非常相似（参见公式（4.36））。但应注意，这里的损失函数依赖于策略的参数（即 θ ），而不是奖励模型的参数（即 ϕ ）。

DPO 的主要优势在于其简单性和效率。DPO 的目标非常直接——它直接优化基于偏好的反馈，而不是依赖于单独开发的奖励模型。此外，DPO 通常更具有样本效率，因为它从固定数据集中学习，无需使用 PPO 中使用的计算成本高昂的采样过程。这使得 DPO 成为人类偏好对齐的流行方法，尤其是在通过强化学习开发和应用奖励模型具有挑战性时。

DPO 可以广泛地被视为一种离线强化学习方法，其中训练数据是预先收集和固定的，且没有探索。相比之下，像 PPO 这样的在线强化学习方法，需要与环境交互（使用奖励模型作为代理）来探索新状态，也具有其独特的优势。在线强化学习的一个好处是它允许代理通过学习实时反馈来持续适应环境的变化。这意味着，与离线方法不同，在线方法不受预先收集数据的静态性质的限制，可以发现新的问题解决策略。此外，探索可以帮助代理覆盖更广泛的状态-动作对，从而提高泛化能力。这对于 LLMs 来说可能是一个重要的优势，因为泛化被认为是在应用如此大型模型时的一个关键方面。

4.4.3 自动偏好数据生成

尽管从人类偏好中学习是调整 LLMs 的有效且流行的方法，但标注偏好数据成本高昂。使用人类反馈不仅面临可扩展性有限的难题，而且可能引入偏差，因为人类反馈本质上具有主观性。因此，人们可以转向 AI 反馈方法来解决这些可扩展性和一致性方面的问题，而不会受到人类标注员相关限制的影响。

与用于指令微调的数据生成一样，使用 LLM 生成偏好数据很简单。给定一组输入，我们首先使用 LLM 生成输出对。然后，我们提示 LLM 对每对输出及其对应输入之间的偏好进行标记。以下是一个提示 LLM 为消费者服务响应对生成偏好标签的示例。

考虑一个客户服务场景，其中客户提出一个请求。您将审查对这个请求的两个回应。请指出哪个回应更受欢迎。请注意，一个好的回应应该是礼貌的、清晰的和简洁的。它应该直接解决客户的问题，提供有用的信息或解决方案，并保持积极的态度。

请求：

Hello, I noticed that my order hasn't arrived yet, though it was scheduled to arrive several days ago. Could you please update me on its status? Thank you!

响应 A：

I'm very sorry for the delay and understand how disappointing this can be. We're doing our best to sort this out quickly for you.

响应 B：

Hey, stuff happens! Your package will get there when it gets there, no need to stress.

响应A更受欢迎。

一旦我们收集了这样的偏好标签，我们就可以使用它们，连同输出对和输入一起训练奖励模型。当然，我们可以考虑展示几个示例或使用高级提示技术，如CoT，以提高标签性能。例如，我们可以在提示中包含一个示例，说明基于CoT理由，为什么两个响应中的一个更受欢迎。

除了偏好标签外，我们还可以获得每个标签的概率 [Lee 等人，2023]。一种简单的方法是从LLM输出的概率中提取标签标记的概率，例如“A”和“B”。然后，我们可以使用Softmax函数或其他归一化技术将这些概率重新归一化到标签的分布上。这些偏好标签的概率可以作为训练奖励模型的点监督信号，如第4.4.1节所述。

数据生成虽然容易扩展，但通常需要确保数据准确且多样化。在这里，数据质量和多样性问题不仅涉及偏好的标注，还包括模型的输入和输出。因此，我们通常需要使用各种技术来获取大规模、高质量的数据。例如，可以通过使用不同的LLM、提示、情境演示等方式生成多样化的模型输出和标注[Cui等人，2024]。Dubois等人[2024]报告称，成对偏好数据的变异性对于从人类或AI反馈中训练LLM非常重要。

虽然从AI反馈中学习具有高度可扩展性和一般客观性，但这种方法更适合于有明确任务和可用客观性能指标的情况。相比之下，从人类反馈中学习更有利于将AI系统与人类价值观、偏好以及需要理解微妙或主观背景的复杂现实任务相一致。这些方法可以结合使用，以训练从人类洞察力和AI反馈的可扩展性中受益的LLM。

4.4.4 步骤对齐

到目前为止，我们关于对齐的讨论主要集中于使用奖励模型来评估整个输入输出序列对。这些方法可以很容易地适应可以通过检查是否包含所需结果来检查输出正确性的场景。例如，在计算数学表达式的任务中，如果答案是正确的，奖励模型可以提供积极反馈，如果答案是错误的，则提供消极反馈。然而，在许多需要复杂推理的问题中，仅仅检查最终结果是否正确对于学习是不够的。想象一下，一个学生只被给出了一个具有挑战性的数学问题的最终答案。知道最终答案是正确还是错误并不能帮助学生找出他们出错的地方以及如何计算正确答案。更好的方法是通过逐步分解问题解决过程来指导学生，并鼓励理解这些步骤背后的基本概念和逻辑。

在第三章中，我们研究了CoT方法，以提示LLMs明确写出达到结论或解决问题所需的中间步骤或推理过程。我们看到了将问题分解成更小的部分可以使理解解决方案路径更容易，并提高输出的准确性。这些方法可以自然地扩展到LLMs的对齐，即我们在推理的中间步骤监督模型。考虑一个推理任务，其中LLM为给定输入生成一系列推理步骤 $\bar{y} = \{\bar{y}_1, \dots, \bar{y}_{n_s}\}$ 。推理的结果假设包含在最后一步 \bar{y}_{n_s} 中，并且可以轻松验证。对于这个推理问题，Uesato等人[2022]将LLM微调方法分为两类：

- 基于结果的方案。仅在验证最终结果时进行监督。这是我们本章讨论的从人类反馈中学习的一种标准方法。例如，LLM被优化以最大化某种形式的奖励 $r(x, y)$ 。
- 基于过程的途径。除了最后一步外，监督还涉及所有中间步骤。为此，我们需要开发一个在每个步骤提供监督信号的模型，并开发能够利用此类监督信号的损失函数。

图4.11显示了一个示例数学问题两个LLM的输出。尽管LLM在这两种情况下都给出了正确的最终答案，但在第二个输出中，它在解决问题的过程中犯了错误。基于结果的方法忽略了这些错误，并对整个解决方案给予积极反馈。相比之下，基于过程的方法可以考虑到这些错误，并在详细的推理步骤上提供额外的指导。

一个基于过程的方法的重要问题是，我们需要在（可能）漫长的推理路径中获得步骤级反馈。我们可以从现有数据集中收集或生成对应于问题的推理路径。然后，人类专家对这些路径中的每个步骤进行标注以验证其正确性。这些标注可以直接用于训练LLMs或作为奖励模型中的奖励。然而，在实践中，通常引入更丰富的标注 [Lightman等人, 2024]。除了`correct`和`incorrect`标签外，一个步骤还可以被标记为`neutral`，以表明虽然该步骤在技术上可能是正确的，但它可能在整体推理过程中仍然存在问题。此外，为了提高数据标注的效率，可以采用主动学习等技术。识别明显的错误通常不会对

Question (from The International Junior Math Olympiad):

Jessica has a lot of apps on her phone. The number of apps she has is divisible by 5. If she downloaded 10 more apps, she would still have less than 50 apps. What is the maximum original number of apps Jessica could have?

Response 1:

- \bar{y}_1 : x must be a multiple of 5, i.e.,
 $x \equiv 0 \pmod{5}$. x
- \bar{y}_2 : After downloading 10 apps, the total
total number of apps should still be less
than 50: $x + 10 < 50$. Solving this
inequality: $x < 40$. x
- \bar{y}_3 : Since x must be a multiple of 5 and
less than 40, the largest value of x is 35. x
- \bar{y}_4 : Verify the result. If $x = 35$, then after
downloading 10 apps, Jessica would
have: $35 + 10 = 45$ which satisfy
 $45 < 50$. x
- \bar{y}_5 : The maximum original number of
apps is 35. x
- ↑
correct

Response 2:

- \bar{y}_1 : The number of apps is divisible by 5,
so x must be a multiple of 5. x
- \bar{y}_2 : If she downloads 10 more apps, her
new total will be $x + 10$, and this must
be less than 50. Therefore, $x + 10 < 50$.
Simplifying this gives $x < 40$. x
- \bar{y}_3 : x must be less than or equal to
the largest multiple of 5, which is 40. problematic x
- \bar{y}_4 : But $x + 10$ should not be more
than or equal to 50. So we need to
subtract 5 from 40. problematic x
- \bar{y}_5 : Therefore, the final result is 35. x
- ↑
correct

图4.11: 两个LLM对数学问题的回答。在回答1中, 最终结果和所有推理步骤都是正确的。在回答2中, 最终结果是正确的, 但在推理过程中有错误(用红色突出显示)。对于基于结果的方法, 两个回答都被认为是正确的。对于基于过程的方法, 回答2中的错误可以在奖励模型中考虑。

从推理错误中学习。相反, 标注模型自信认为是正确但实际上有问题的步骤通常更有效。

给定一组步骤级标注推理路径及其对应的输入, 我们可以训练一个奖励模型来提供反馈以监督策略学习。奖励模型可以被视为一个分类模型, 因此其架构可以是一个带有堆叠 Softmax 层的 Transformer 解码器。在步骤 k 时, 奖励模型将问题描述(用 \mathbf{x} 表示)和迄今为止生成的推理步骤(用 $\bar{\mathbf{y}}_{\leq k}$ 表示)作为输入, 并输出一个关于标签集 $\{\text{correct}, \text{incorrect}\}$ 或 $\{\text{correct}, \text{incorrect}, \text{neutral}\}$ 的概率分布。然后, 使用学习到的奖励模型通过评估每一步的正确性来评估推理路径。建模正确性的简单方法是对被分类为 *correct* 的步骤数量进行计数, 由以下公式给出

$$r(\mathbf{x}, \mathbf{y}) = \sum_{k=1}^{n_s} \delta(\text{correct}, C(\mathbf{x}, \bar{\mathbf{y}}_{\leq k})) \quad (4.83)$$

在 $C(\mathbf{x}, \bar{\mathbf{y}}_{\leq k})$ 表示具有最大概率的标签处。我们还可以使用对数概率

对整个路径的奖励进行分类以定义

$$r(\mathbf{x}, \mathbf{y}) = \sum_{k=1}^{n_s} \log \Pr(\text{correct} | \mathbf{x}, \bar{\mathbf{y}}_{\leq k}) \quad (4.84)$$

在 $\text{correct} | \mathbf{x}, \bar{\mathbf{y}}_{\leq k}$ 的概率表示奖励模型生成的 *correct* 标签的概率。然后可以使用奖励分数 $r(\mathbf{x}, \mathbf{y})$ 来像往常一样在 RLHF 中训练策略。

虽然我们将讨论限制在数学问题上，但这里描述的方法是通用的，可以应用于涉及多步推理和决策的广泛任务。此外，在评估步骤的质量时，我们可以考虑各种方面，而不仅仅是其正确性。例如，在对话系统中，响应不仅必须准确，而且必须在整个对话的多个回合中上下文适当。如果一个模型提供了正确的响应但未能保持与当前对话的连贯性，步骤级反馈可以帮助模型识别和纠正这种差异。此外，请注意，基于过程的方法与第4.4.1.3节中讨论的细粒度奖励建模方法相关。所有这些方法本质上旨在通过将输出分解为更小、更易于管理的步骤来为LLMs提供更详细的监督。然而，基于过程的方法更侧重于根据其先前步骤评估步骤的正确性，而第4.4.1.3节中的方法强调独立评估每个步骤。

LLMs逐步对齐的想法具有很大的应用潜力，尤其是考虑到LLMs在最近向更复杂的推理任务转变。例如，GPT-o1和GPT-o3模型都采用了更先进的推理技术（如长内部CoT）来解决科学和数学推理等具有挑战性的问题[OpenAI, 2024]。这些任务通常依赖于长而复杂的推理路径，因此，在推理过程中引入详细的监督信号似乎是必不可少的。此外，从实际角度来看，对长推理路径的有效监督不仅提高了推理性能，还有助于模型消除冗余或不必要的推理步骤，从而降低推理复杂度并提高效率。

4.4.5 推理时对齐

在这个部分，我们探索了多种方法来使模型与人类偏好和标注对齐。然而，许多此类方法的显著局限性在于LLMs必须进行微调。对于RLHF及其变体，使用奖励模型训练LLMs可能计算成本高昂且不稳定，导致在应用这些方法时复杂性和成本增加。在这种情况下，我们可以考虑在推理时对齐模型，从而避免涉及额外复杂性和努力的步骤。

一种简单实现推理时对齐的方法是使用奖励模型从LLM生成的 N 个替代输出中选择最佳的一个，这种方法被称为Best-of- N 采样（BoN采样）。我们可以将BoN采样视为一种重新排序。事实上，重新排序方法长期以来一直被广泛应用于NLP任务中，如机器翻译。它们通常应用于训练复杂模型成本高昂的情况。在这种情况下，直接重新排序输出可以以非常低的成本⁸将这些复杂模型纳入其中。

⁸Reranking methods can also help us explore what are known as model errors and search errors, although these

在BoN采样过程中，LLM接收输入序列 \mathbf{x} 并生成 N 个不同的输出序列 $\{\hat{\mathbf{y}}_1, \dots, \hat{\mathbf{y}}_N\}$ ：

$$\{\hat{\mathbf{y}}_1, \dots, \hat{\mathbf{y}}_N\} = \arg\text{TopN}_{\mathbf{y}} [\Pr(\mathbf{y}|\mathbf{x})] \quad (4.85)$$

在 $\arg\text{TopN}$ 操作返回最大化函数 $\Pr(\mathbf{y}|\mathbf{x})$ 的前- N 个输出。这些输出可以通过多种方式生成，具体取决于模型使用的搜索算法（例如采样或束搜索）。一旦生成了 N 个最佳输出候选，就使用奖励模型来评估和选择最佳输出：

$$\hat{\mathbf{y}}_{\text{best}} = \max\{r(\mathbf{x}, \hat{\mathbf{y}}_1), \dots, r(\mathbf{x}, \hat{\mathbf{y}}_N)\} \quad (4.86)$$

值得注意，BoN采样结果也受到 N -best列表多样性的影响。这是大多数重排序方法中普遍存在的问题。通常，我们希望 N -best输出候选者具有较高的质量，但彼此之间足够不同。在许多文本生成系统中， N -best输出非常相似，通常只相差一两个词。在LLMs中，多样性问题更为严峻，因为LLMs生成的 N -best输出在措辞上可能不同，但它们的语义意义通常非常相似。在实践中，可以通过调整模型超参数和/或采用不同的LLMs来生成更多样化的输出候选者以进行重排序。然而，与许多实际系统一样，我们需要在选择高质量候选者和确保生成输出足够多样化之间做出权衡。

BoN采样也可以用于训练LLMs。一种密切相关的方法是拒绝采样。在此方法中，我们首先通过奖励模型从 N -best列表中选择“最佳”输出，然后将这些选定的输出用于微调LM。通过这种方式，我们可以通过比RLHF更简单的方法将人类偏好引入LLMs的训练中。许多LLMs已采用拒绝采样进行微调[Nakano等人，2021；Touvron等人，2023b]。

4.5 摘要

在这一章中，我们探讨了多种对齐LLMs的技术。特别是，我们讨论了微调方法，这些方法使LLMs能够遵循指令并将它们与人类偏好对齐。微调LLMs的一个好处是计算效率。与基于大规模神经网络优化的预训练不同，微调是一个训练后的步骤，因此计算成本较低。此外，它更适合解决在预训练中难以解决的问题，例如人类价值观对齐。对对齐问题的广泛关注也导致了对该主题的研究论文激增，这给编写本章带来了挑战，因为很难涵盖所有最新的技术。然而，我们已尽力提供对对齐基本方法的相对详细介绍，例如

issues are not often discussed in the context of LLMs. For example, suppose we have an old model and a new, more powerful model. We can use the new model to select the best output from the N -best list of the old model as the oracle output. The performance difference between the oracle output and the top-1 output of the original N -best list reflects the performance gain brought by the new model. If the performance gain is significant, we can say that the old model has more model errors. If the gain is small, it may indicate that the issue lies in search errors, as the best candidates were not found.

指令微调 and RLHF。

虽然我们在这章中关注了LLM对齐技术，但术语 *AI alignment* 是一个广泛的概念。它通常指的是确保人工智能系统的行为与人类价值观、目标和期望相一致的过程。人工智能对齐的想法可以追溯到人工智能的早期。关于人工智能对齐的一个广泛引用的描述来自数学家和计算机科学家诺伯特·维纳（Norbert Wiener）的一篇文章[Wiener, 1960]。引言如下

If we use, to achieve our purposes, a mechanical agency with whose operation we cannot efficiently interfere ... we had better be quite sure that the purpose put into the machine is the purpose which we really desire.

当时，AI对齐对研究人员来说是一个遥远的担忧。但如今，它极大地影响着各种AI系统的设计。例如，在机器人领域，对齐对于确保自主机器人安全地与人类及其环境互动至关重要。在自动驾驶中，汽车不仅要遵守交通法规，还要做出复杂、实时的决策，优先考虑人类安全，避免事故，并解决道德困境。

在当前的AI研究中，对齐通常是通过开发一个类似于真实目标的代理目标，并将AI系统引导到这个目标上实现的。然而，设计AI对齐的目标非常困难。一个原因是人类价值观多样化且往往具有情境依赖性，这使得很难将它们提炼成一个单一、普遍适用的目标函数。此外，现实世界环境的复杂性，其中价值观和目标经常冲突或随时间演变，进一步增加了对齐工作的难度。即使我们能够定义一个适当的目标，AI系统也可能找到意想不到的方式来实现它，从而导致“不匹配”的结果，这些结果虽然在技术上满足目标，但以有害或反生产的方式实现。

这些挑战促使并正在推动人工智能研究向更一致的系统发展，无论是通过开发新的感知世界机制，还是通过更高效和更具普遍性的方法来适应这些系统以完成特定任务。更重要的是，随着人工智能系统变得更加强大和智能，尤其是在最近大型语言模型在处理许多挑战性问题中展现出非凡能力的情况下，人工智能对齐的需求变得更加紧迫。研究人员开始关注人工智能安全，并警告社区，他们需要谨慎地开发和发布人工智能系统，以防止这些系统出现偏差 [Russell, 2019; Bengio 等人, 2024]。

CHAPTER 5

推理

一旦我们预训练和微调了一个LLM，我们就可以将其应用于对新数据的预测。这个过程称为推理，其中LLM根据输入计算不同可能输出的概率，并选择概率最大的输出。推理问题通常以下列形式表示：

$$\hat{y} = \arg \max_y \Pr(y|x) \quad (5.1)$$

在 x 是输入序列， y 是可能的输出序列， \hat{y} 是最佳输出序列的情况下。

这是NLP中最广泛采用的公式之一，可以追溯到基于概率模型的语音识别和机器翻译系统早期。尽管对于某些应用，例如使用非常小的语言模型预测一个标记，解决这个问题似乎很简单，但对于大多数情况，计算挑战来自于计算 $\Pr(y|x)$ 和执行 $\arg \max$ 操作。因此，我们希望在本章中解决的问题包括：1) 在给定训练好的LLM的情况下，有效地计算预测概率，2) 设计一个有效的（次优）搜索 \hat{y} 。

在较高层次上，这些问题是人工智能的基本问题，已经得到了广泛的研究。因此，许多已建立的技术可以直接应用，例如，可以使用贪婪搜索算法来实现高效的推理系统。另一方面，针对特定模型的优化，如针对Transformers的高效注意力模型，可以考虑以进一步提高效率。但是，在许多实际应用中，我们仍然需要在准确性和效率之间进行权衡，通过仔细结合各种技术。

LLMs中推理问题的重要性还在于许多应用场景需要处理极其长的序列。最近的研究发现，在推理过程中注入额外的提示和上下文信息，如长思维链提示，可以显著提高LLMs的性能。这为LLMs的扩展提供了一种新方法：通过增加推理时的计算量可以获得更好的结果。例如，通过推理时间扩展，OpenAI [2024] 的o1和Deepseek [2025] 的R1系统在复杂推理和编程任务上展示了令人印象深刻的性能。这反过来又促使NLP领域更加关注高效推理的问题。

在这一章中，我们将介绍LLM推理的基本概念和算法，包括预填充-解码框架、搜索（解码）算法和推理性能评估指标。然后，我们将介绍提高LLM推理效率的方法，涵盖一系列加快系统和压缩模型的技术。最后，我们将讨论推理时间缩放，这被认为是推理优化的重要应用。

5.1 预填充和解码

在这个部分，我们介绍了预填充解码框架，这是最常用于解释和实现LLM推理过程的框架。我们首先介绍符号和背景知识，然后描述框架的细节，例如LLM推理的解码算法。

5.1.1 前言

尽管我们在本书中多次描述了LLMs，但我们首先简要定义符号以方便后续讨论，并使本章自成体系。

\mathbf{x} : 输入标记序列。从概念上讲，它等同于一个“提示”，包括指令、用户输入以及任何作为LLM输入的附加上下文。 \mathbf{x} 包含 $m + 1$ 个标记，用 $x_0 \dots x_m$ 表示，其中 x_0 是起始符号 $\langle \text{SOS} \rangle$ 。 \mathbf{y} : 输出标记序列，也称为对输入的响应。 \mathbf{y} 包含 n 个标记，用 $y_1 \dots y_n$ 表示。 $y_{<i}$: 位于 i 之前的输出标记，即 $y_{<i} = y_1 \dots y_{i-1}$ 。 $\Pr(\mathbf{y}|\mathbf{x})$: 使用LLM生成 \mathbf{y} 给定 \mathbf{x} 的概率。如果LLM由 θ 参数化，我们可以将其写为 $\Pr_{\theta}(\mathbf{y}|\mathbf{x})$ 。 $[\mathbf{x}, \mathbf{y}]$: \mathbf{x} 和 \mathbf{y} 的连接标记序列。即 $[\mathbf{x}, \mathbf{y}] = x_0 \dots x_m y_1 \dots y_n$ 。偶尔，我们使用 $\text{seq}_{\mathbf{x}, \mathbf{y}}$ 来表示 $[\mathbf{x}, \mathbf{y}]$ 。 $\Pr([\mathbf{x}, \mathbf{y}])$: 使用LLM生成标记序列 $[\mathbf{x}, \mathbf{y}]$ 的概率。

如第(5.1)式所述，LLM推理的目标是最大化 $\Pr(\mathbf{y}|\mathbf{x})$ 。在NLP中，建模这种条件概率是常见的。乍一看，这似乎是一个序列到序列的问题，其中我们使用编码-解码模型将一个序列转换为另一个序列。然而，我们不是在讨论序列到序列问题或编码-解码架构。相反，正如前几章所讨论的，这个问题可以通过使用仅解码器模型来解决。为此，我们可以将log-scale概率 $\log \Pr(\mathbf{y}|\mathbf{x})$ 解释为 $\log \Pr([\mathbf{x}, \mathbf{y}])$ 和 $\log \Pr(\mathbf{x})$ 之间的差异

$$\log \Pr(\mathbf{y}|\mathbf{x}) = \log \Pr([\mathbf{x}, \mathbf{y}]) - \log \Pr(\mathbf{x}) \quad (5.2)$$

在何处通过运行LLM在序列 $[\mathbf{x}, \mathbf{y}]$ 和 \mathbf{x} 上分别获得 $\log \Pr([\mathbf{x}, \mathbf{y}])$ 和 $\log \Pr(\mathbf{x})$ 。例如，我们可以通过链式计算生成 \mathbf{x} 的概率。

规则

$$\begin{aligned}
 \log \Pr(\mathbf{x}) &= \log \Pr(x_0 \dots x_m) \\
 &= \log [\Pr(x_0) \Pr(x_1|x_0) \cdots \Pr(x_m|x_0 \dots x_{m-1})] \\
 &= \underbrace{\log \Pr(x_0)}_{=0} + \sum_{j=1}^m \log \Pr(x_j|\mathbf{x}_{<j}) \\
 &= \sum_{j=1}^m \log \Pr(x_j|\mathbf{x}_{<j})
 \end{aligned} \tag{5.3}$$

换句话说，我们在 \mathbf{x} 的每个位置计算标记预测对数概率，并将所有这些对数概率相加。

在LLM的常见实现中，然而，我们不需要计算输入序列的对数概率，而是使用LLM直接计算以下形式输出序列的对数概率 $\{v^*\}$

$$\log \Pr(\mathbf{y}|\mathbf{x}) = \sum_{i=1}^n \log \Pr(y_i|\mathbf{x}, \mathbf{y}_{<i}) \tag{5.4}$$

在 $[\mathbf{x}, \mathbf{y}_{<i}]$ 表示预测 y_i 的上下文。我们使用 $\Pr(y_i|\mathbf{x}, \mathbf{y}_{<i})$ 来表示 $\Pr(y_i|[\mathbf{x}, \mathbf{y}_{<i}])$ ，遵循文献中常用的符号表示法。

现在，我们在解决方程（5.1）中描述的推理问题时有两个子问题：

- 模型计算：我们以高效的方式对 $\Pr(y_i|\mathbf{x}, \mathbf{y}_{<i})$ 进行建模和计算。
- 搜索：我们在对数概率 $\Pr(\mathbf{y}|\mathbf{x})$ 方面找到最优（或次优）输出序列。

第二个子问题是NLP中的一个经典问题。我们将在5.1.3节中展示，存在几个经过充分研究的算法可以应用于高效搜索可能的输出序列空间。第一个子问题需要语言模型在给定一系列上下文标记的情况下，对词汇表 V 生成一个分布。我们可以通过训练一个Transformer解码器来实现这一点，该解码器输出分布

$$\Pr(\cdot|\mathbf{x}, \mathbf{y}_{<i}) = \text{Softmax}(\mathbf{H}\mathbf{W}^o)_{m+i} \tag{5.5}$$

$$\mathbf{H} = \text{Dec}([\mathbf{x}, \mathbf{y}_{<i}]) \tag{5.6}$$

这里 $\text{Dec}(\{v^*\})$ 生成一系列表示，每个表示对应输入序列的一个位置。因此，如果我们输入 $[\mathbf{x}, \mathbf{y}_{<i}]$ 到 LLM 中， \mathbf{H} 是一个 $i' \times d$ 矩阵，其中 d 是每个表示的维度， $i' = m + i$ 是上下文标记的数量。然后我们可以使用 Softmax 层将这些表示转换为标记的分布。 $\mathbf{W}^o \in \mathbb{R}^{d \times |V|}$ 是 Softmax 层的线性映射矩阵， $\mathbf{H}\mathbf{W}^o$ 将 \mathbf{H} 中的 d -维表示转换为 $|V|$ -维表示。使用下标 $m + i$ 表示仅在位置 $m + i$ 的表示上执行 Softmax 函数。参见图 5.1 以了解此架构的说明。

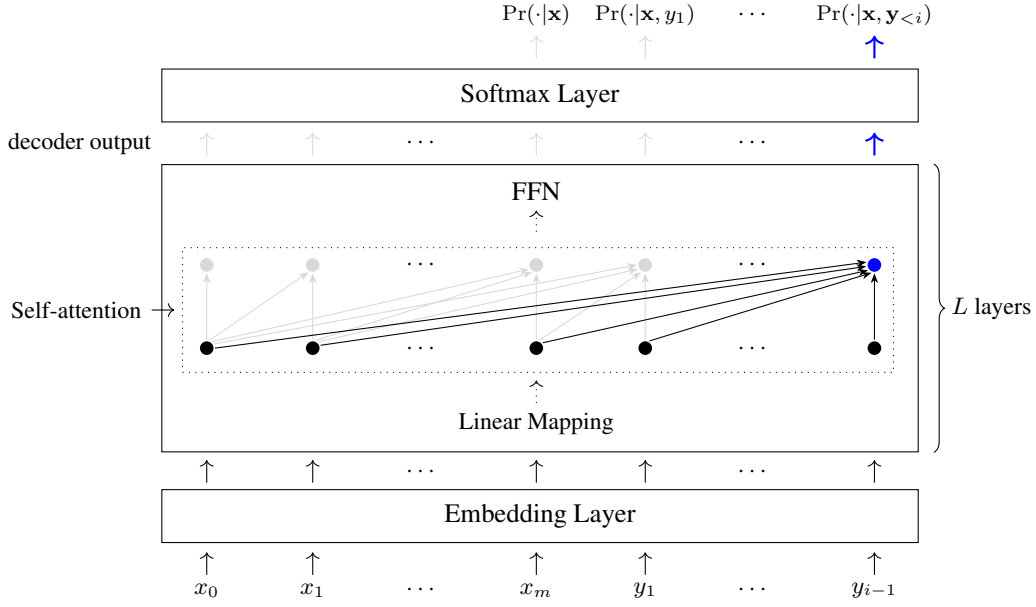


图5.1: LLM的仅解码器架构。解码器由一个嵌入层和一系列Transformer层组成。在每个Transformer层中，输入通过一个线性映射、一个自注意力网络和一个FFN。解码器的输出是一系列表示，作为Softmax层的输入，该层为每个位置生成一个标记分布。

$\text{Dec}(\cdot)$ 是一个由嵌入网络和多个堆叠的自注意力网络和FFN网络组成的Transformer解码网络。我们在此处不会详细讨论Transformer，因为读者可以很容易地从文献中了解这些模型。然而，值得注意的是，推理的难度部分来自于Transformer中自注意力机制的使用。回想一下，单头自注意力的通用形式如下：

$$\text{Att}_{\text{qkv}}(\mathbf{q}_{i'}, \mathbf{K}, \mathbf{V}) = \text{Softmax}\left(\frac{\mathbf{q}_{i'} \mathbf{K}^T}{\sqrt{d}}\right) \mathbf{V} \quad (5.7)$$

在位置 i' (的查询 $\mathbf{q}_{i'} \in \mathbb{R}^d$ ，即 y_i) 的位置，以及 \mathbf{K} 和 $\mathbf{V} \in \mathbb{R}^{i' \times d}$ 分别是键和值，直到 i' 。

在推理的每一步中，我们调用自注意力函数 $\text{Att}_{\text{qkv}}(\cdot)$ ，然后是 FFN，以生成一个 d 维度的表示，该表示整合了当前标记及其左侧上下文的信息。此过程通过 L 层自注意力和 FFN 重复进行，形成一个 Transformer 层的堆栈。这个堆栈中 L 层的输出是最终的表示。

每次，该模型将位置 i' 关注到所有先前位置，这导致 $2i'$ 个向量乘积 ($\mathbf{q}_{i'} \mathbf{K}^T$ 的 i' 次和 $\text{Softmax}(\frac{\mathbf{q}_{i'} \mathbf{K}^T}{\sqrt{d}})$ 与 \mathbf{V} 的 i' 次乘积)。因此，生成长度为 len 的序列具有 $O(L \times len^2)$ 的自注意力网络时间复杂度。显然，由于与序列长度相关的二次时间复杂度，该模型的推理对于长序列来说很慢。因此，许多针对 Transformers 和替代模型的改进都集中在比这种二次时间复杂度更快的有效方法上，例如稀疏注意力机制和线性时间模型。关于有效 Transformers 的详细讨论可以在前几章中找到，本节将专注于

标准Transformer架构。

注意，在自注意力中，层的查询、键和值是从相同输入（即前一层输出）的线性映射。一旦生成新的键值对，它将在后续推理步骤中反复使用。与其在推理过程中重新生成这些键值对，不如将它们存储在称为键值缓存或KV缓存的机构中。因此， (\mathbf{K}, \mathbf{V}) 可以简单地被视为一个KV缓存。此缓存按以下方式更新

$$\mathbf{K} = \text{Append}(\mathbf{K}, \mathbf{k}_{i'}) \quad (5.8)$$

$$\mathbf{V} = \text{Append}(\mathbf{V}, \mathbf{v}_{i'}) \quad (5.9)$$

在位置 i' 处， $(\mathbf{k}_{i'}, \mathbf{v}_{i'})$ 是新生成的键值对， $\text{Append}(\mathbf{a}, \mathbf{b})$ 表示一个将行向量 \mathbf{b} 添加到矩阵 \mathbf{a} 的函数。图 5.2 展示了 Transformer 解码器如何与 KV 缓存一起工作。

最后，计算 $\log \Pr(\mathbf{y}|\mathbf{x})$ 的过程总结如下：

1. 我们将 \mathbf{x} 和 \mathbf{y} 连接成一个序列 $[\mathbf{x}, \mathbf{y}]$ 。对于此序列的每个位置 i' ，我们执行以下步骤。

(a) 我们计算位置 i' 的标记嵌入，并将得到的嵌入作为初始表示输入到 Transformer 层的堆栈中。(b) 在每个 Transformer 层中，我们首先通过自注意力网络传递输入表示，然后通过 FFN。在自注意力网络中，输入表示被转换为 $\mathbf{q}_{i'}$ 、 $\mathbf{k}_{i'}$ 和 $\mathbf{v}_{i'}$ 。然后，我们使用 $\mathbf{k}_{i'}$ 和 $\mathbf{v}_{i'}$ （更新 KV 缓存 (\mathbf{K}, \mathbf{V}) （见公式 (5.8-5.9)））。然后，我们通过将 $\mathbf{q}_{i'}$ 关注到 (\mathbf{K}, \mathbf{V}) 来计算注意力模型的输出（见公式 (5.7)）。(c) 如果 $i' > m$ （即 $i = i' - m \geq 0$ ），我们取 Transformer 堆栈的输出，并通过 Softmax 层计算标记预测概率 $\Pr(y_i|\mathbf{x}, \mathbf{y}_{<i})$ （见公式 (5.5)）。

2. 当达到序列的末尾时，我们通过 $i \in [1, n]$ 上对 $\log \Pr(y_i|\mathbf{x}, \mathbf{y}_{<i})$ 求和来获得 $\log \Pr(\mathbf{y}|\mathbf{x})$ （参见公式 (5.4)）。

5.1.2 双阶段框架

如我们所见，语言建模是一个标准的自回归过程，其中每个标记逐个生成，基于之前的标记。对于 Transformer，这要求模型维护一个存储过去表示的 KV 缓存，并将新生成的表示关注到这个 KV 缓存中。如果我们从计算 KV 缓存的角度来看模型 $\Pr(\mathbf{y}|\mathbf{x})$ ，那么将推理分为两个阶段是自然而然的：

- 预填充。预填充阶段计算输入序列 \mathbf{x} 的 KV 缓存。它被称为预填充，因为模型为每个标记准备和存储键值对

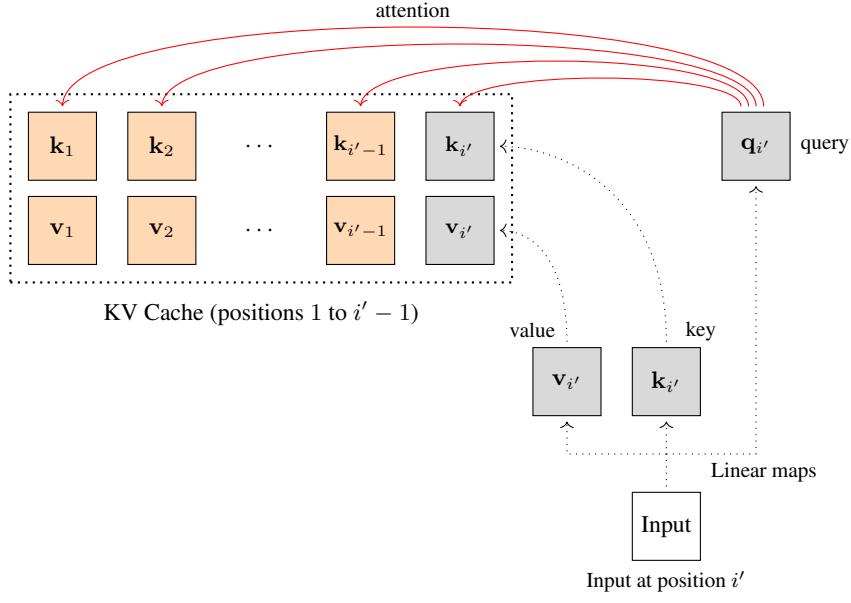
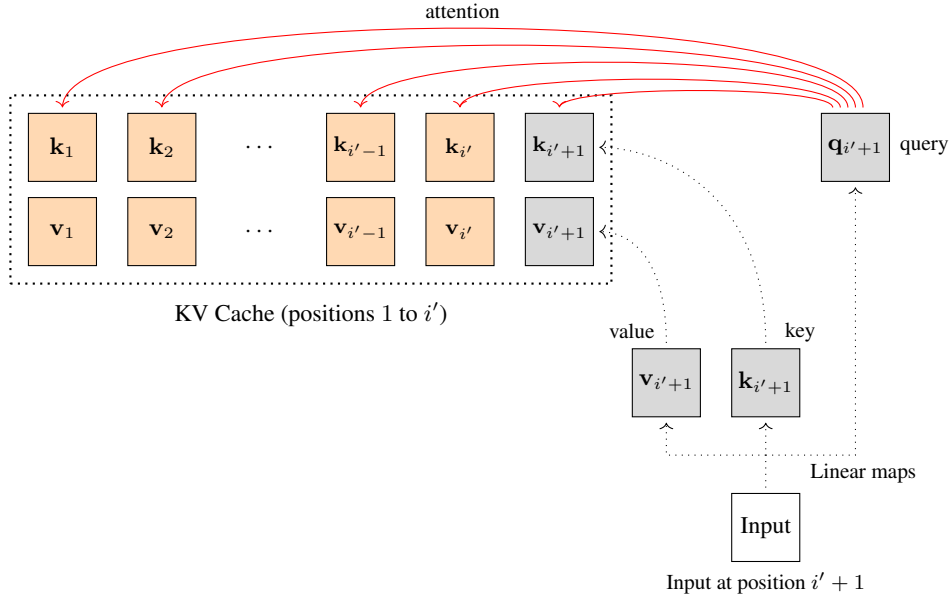
(a) Updating the KV Cache at Position i' (b) Updating the KV Cache at Position $i' + 1$

图5.2: KV缓存的示意图。我们在一个位置更新KV缓存, 执行注意力操作, 然后移动到下一个位置重复此过程。

在输入实际推理开始之前。在LLM中的预填充过程可以表示为

$$\text{cache} = \text{Dec}_{\text{kv}}(\mathbf{x}) \quad (5.10)$$

$\text{Dec}_{\text{kv}}(\cdot)$ 是解码网络（即与 $\text{Dec}(\cdot)$ 相同），但它返回KV

缓存在自注意力机制中而不是输出表示中。缓存是一个列表，由

$$\text{cache} = \{\text{cache}^1, \dots, \text{cache}^L\} \quad (5.11)$$

缓存^{*l*}表示第 l 层的键值对。

- 解码。解码阶段继续根据KV缓存生成标记，如图5.2所示。当新的标记输入到解码器时，我们通过添加新的键值对来更新每一层的KV缓存。然后使用更新的缓存进行自注意力计算。当满足某些停止标准时，例如生成的标记是结束符号时，标记生成停止。解码的目的是找到最佳预测序列，该序列由以下公式给出

$$\hat{\mathbf{y}} = \arg \max_{\mathbf{y}} \Pr(\mathbf{y} | \text{cache}) \quad (5.12)$$

这里我们使用 $\Pr(\mathbf{y} | \text{cache})$ 而不是 $\Pr(\mathbf{y} | \mathbf{x})$ 来强调解码过程实际上依赖于 KV 缓存而不是 \mathbf{x} 。

预填充和解码过程如图5.3所示。请注意，这两个过程都是自回归的。然而，如表5.1所示，它们在几个方面有所不同，这导致在实际应用中实现方式非常不同。

本质上，虽然预填充的底层模型基于标记预测，但它可以被视为一个编码过程。这是因为我们的目标不是生成标记，而是为解码阶段的后续步骤构建一个上下文表示（即KV缓存）。从这个意义上讲，它与BERT相似，其中我们将输入序列编码成一系列上下文化的标记表示。另一方面，与BERT生成双向序列表示不同，预填充基于标准的语言建模任务，因此是单向的。请注意，由于整个序列 \mathbf{x} 一次性输入到模型中，所有查询可以一起打包，并在 \mathbf{x} 上并行执行自注意力操作。设 \mathbf{Q} 为打包到一个矩阵中的查询。预填充中的自注意力模型可以定义为

$$\text{Att}_{\text{qkv}}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d}} + \mathbf{Mask}\right)\mathbf{V} \quad (5.13)$$

在 $\{\mathbf{v}^*\}$ 处不变。掩码 $\in \mathbb{R}^{(m+1) \times (m+1)}$ 是一个确保每个标记只关注自身以及序列中先前的标记的掩码。它通过将掩码中对应未来标记的值设置为一个大负数来表示，例如，对于查询 \mathbf{q}_i 和键 \mathbf{k}_j ，如果 $i < j$ ，我们设置条目 (i, j) 的值为 $-\infty$ 。使用单个自注意力计算处理序列的一个优点是我们可以更好地利用现代 GPU 的并行计算能力，从而加快预填充。一般来说，预填充过程被认为是计算密集型的。这是因为将多个计算操作合并为一个操作减少了数据传输次数，性能瓶颈通常来自计算能力而不是内存带宽。

解码是一种标准的从左到右的文本生成过程。标记序列通过基于KV缓存逐个预测标记而自回归地生成。每次生成新标记时，我们需要根据公式 (5.7) 将其关注到前面的标记上。因此，解码

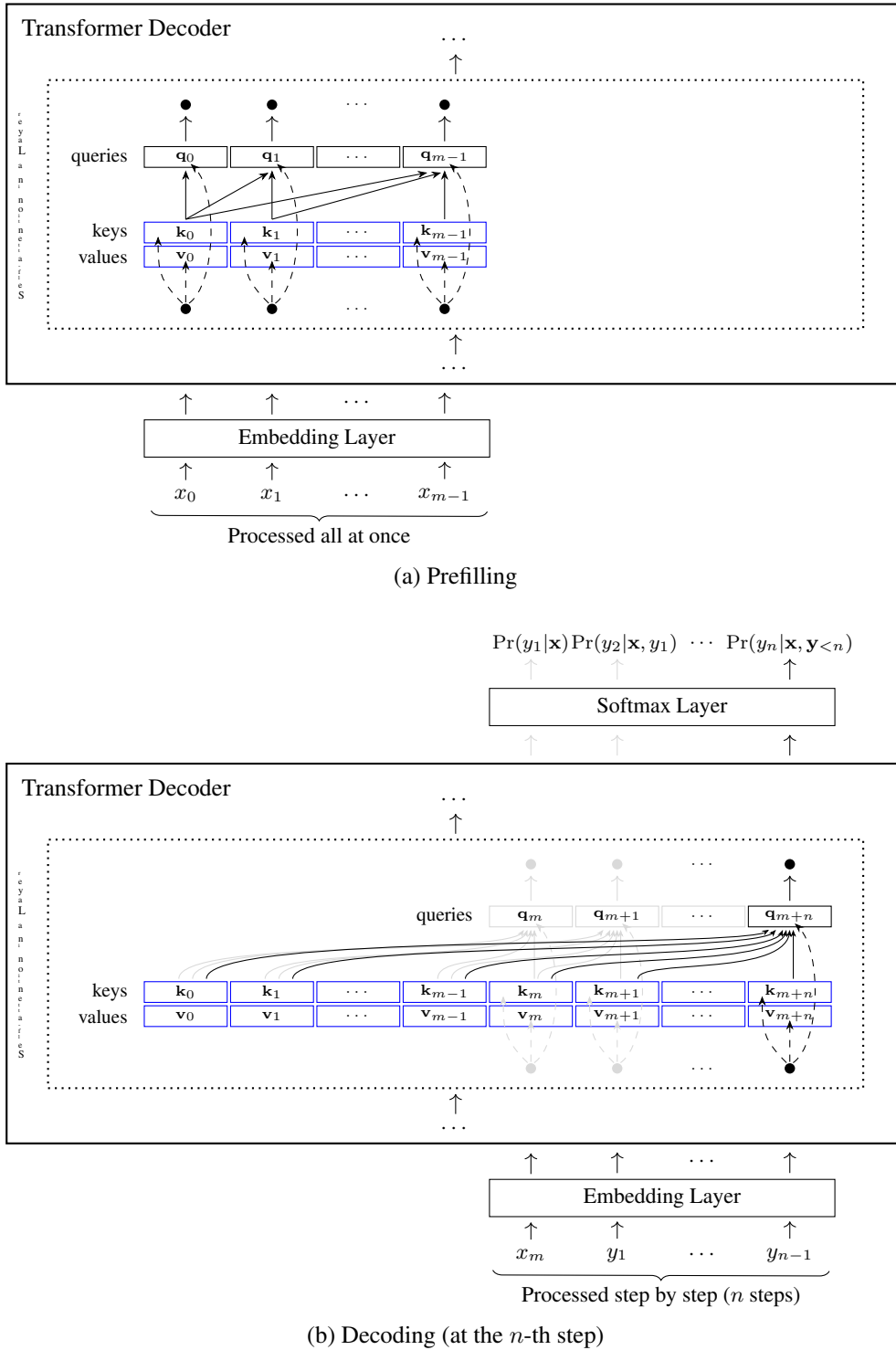


图5.3: 预填充和解码过程的说明。在预填充过程中，整个输入序列一起处理，并填充KV缓存。在解码过程中，LLM根据预填充的KV缓存逐步生成输出序列。

由于频繁访问KV缓存，该过程受内存限制。随着更多标记的生成，解码成本显著增加。在大多数情况下，解码的计算成本高于预填充。请注意，这不仅是因为在解码过程中，LLM逐个生成标记并反复更新KV缓存。正如我们将在下一个小节中看到的，

	Prefilling	Decoding
Goal	Set up initial context \mathbf{x} .	Continue generating tokens \mathbf{y} after the initial input.
All-at-once Visibility	Tokens in \mathbf{x} are presented all at once.	Tokens in \mathbf{y} are presented sequentially, that is, predicting a token requires waiting for the previous tokens to be predicted first.
Context Use	Build the context or encoded representation of the input.	Use the cached key-value pairs (from prefilling) to generate further tokens.
Resource Limitation	Compute-bound	Memory-bound
Computational Cost	High	Very High

表5.1: 预填充与解码。

在解码过程中，我们可能需要探索多个不同的标记序列，这使得问题更加复杂，并进一步增加了其成本。

5.1.3 解码算法

到目前为止，我们关于LLM推理的讨论主要集中于模型计算问题，即如何计算 $\Pr(\mathbf{y}|\mathbf{x})$ 。现在我们转向搜索问题的讨论。该问题可以表述为：给定一个LLM $\Pr(\mathbf{y}|\mathbf{x})$ ，在给定输入序列 \mathbf{x} （或生成的KV缓存）的情况下，我们如何高效地搜索最佳输出序列 $\hat{\mathbf{y}}$ 。直观地，我们可以考虑所有输出序列，计算每个序列的预测概率，然后选择具有最高概率的输出序列。这种方法可以保证全局最优解，但直接穷举搜索对于LLM来说是不切实际的，因为可能的输出序列数量随着 \mathbf{y} 长度的增加而呈指数增长。

在实践中，各种启发式搜索算法，如贪婪搜索和基于采样的搜索，通常被用来近似求解。这些方法中的每一种都提供了搜索质量和计算效率之间的权衡。因此，搜索问题变成了探索和利用之间的平衡行为，其目标是找到一种高效的策略，在不探索整个空间的情况下产生高质量的输出。

在更详细地讨论这些方法之前，让我们首先非正式地定义什么是搜索空间以及它是如何表示的。在LLM推理中，我们将一个假设定义为输入和输出序列的元组。由于在推理过程中 \mathbf{x} 是固定的，我们可以简单地考虑每个假设为一个输出序列。表示为 \mathcal{Y} 的搜索空间是模型可以生成的所有可能的假设（即输出序列）的集合。LLM推理的搜索问题可以重新表述为

$$\hat{\mathbf{y}} = \arg \max_{\mathbf{y} \in \mathcal{Y}} \Pr(\mathbf{y}|\mathbf{x}) \quad (5.14)$$

在NLP中， \mathcal{Y} 通常以树形数据结构表示，以方便搜索。图5.4显示了由小词汇表生成的搜索树示例。在此示例中，一个节点

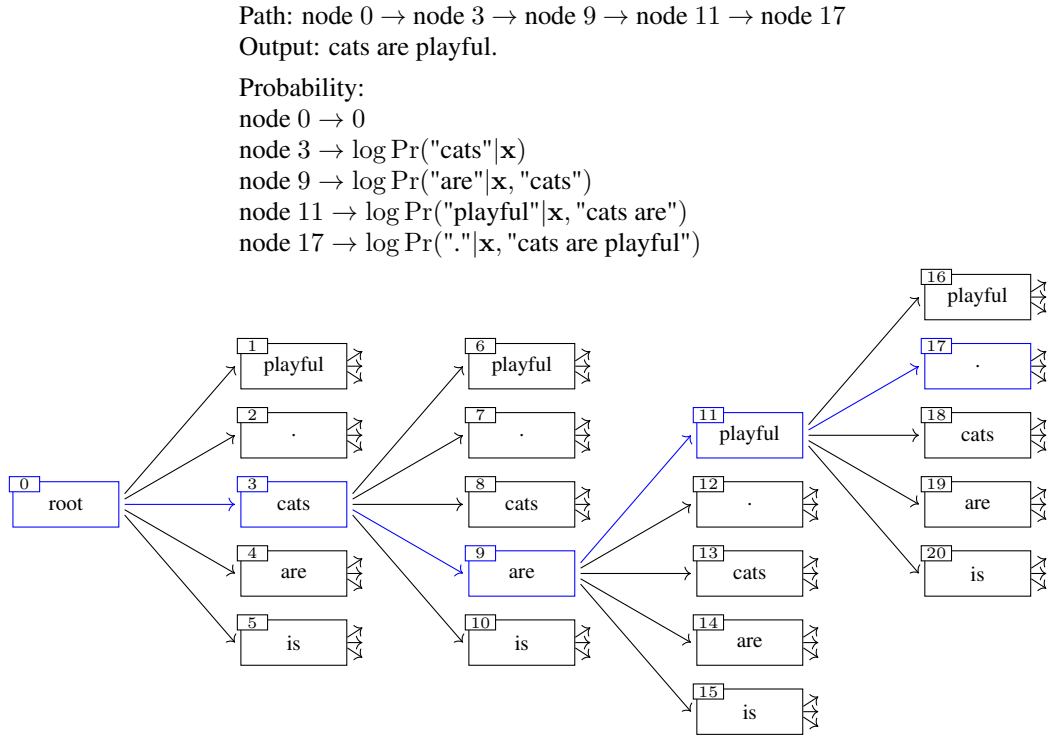


图5.4: 解码的搜索树。在每个节点, 我们通过考虑所有可能的标记来扩展树, 每个标记都导致一个新的节点, 该节点代表文本的潜在延续。在这里, 我们突出显示了一条通过节点0、3、9、11和17的路径。该路径表示输出序列“{v*}”, 其对数概率可以通过累积这些节点的对数概率来计算。

表示可以被许多序列共享的前缀子序列。搜索从树的根节点开始, 这可以被视为所有可以生成的序列的开始¹。每个子节点通过向序列添加一个词汇表中的标记来扩展其父节点的前缀, 同时还包括给定前缀预测该标记的概率。这个过程继续进行, 每个节点进一步分支成更多的子节点, 每个子节点代表序列的新可能扩展, 添加另一个标记。因此, 搜索树变得更深更宽, 代表随着更多标记的附加, 潜在序列的数量不断增加。这种结构使我们能够高效地遍历可能的序列, 根据从根节点到该节点的路径上累积的对数概率来评估每个序列。例如, 在图5.4中, 从根节点到节点17的路径对应于输出序列“Cats are playful.”。预测对数概率 $\log \Pr(\mathbf{y}|\mathbf{x})$ 是此路径上所有节点的对数概率之和。

通常, 搜索树按层次组织, 其中每一层包含与根节点相同距离的所有节点。因此, 在树上的广度优先搜索本质上执行从左到右的标记生成。同一层的节点对应于相同长度的序列。随着搜索的进行, 新标记被附加到这些序列中, 逐步扩展它们。

设 Y_i 为 LLM 在步骤 i 生成的序列集合。 Y_i 可以通过将 Y_{i-1} 中的每个序列与词汇表 V 中所有可能的下一个标记进行扩展来获得。

¹Here, since the predictions in LLMs are based on \mathbf{x} , we can think of the root as a representation of \mathbf{x} .

以下递归形式

$$Y_i = Y_{i-1} \times V \quad (5.15)$$

$Y_{i-1} \times V$ 表示 Y_{i-1} 和 V (的笛卡尔积, 即 Y_{i-1} 中的每个序列都与 V 中的每个标记连接。注意, 如果 Y_{i-1} 中的序列是完整的 (例如, 以 $\langle \text{EOS} \rangle$ 标记结束), 则不会进一步展开。设 $\Psi(Y_i)$ 为 Y_i 中所有完整序列的集合。那么, 搜索空间可以表示为

$$\mathcal{Y} = \Psi(Y_1) \cup \Psi(Y_2) \cup \dots \cup \Psi(Y_{n_{\max}}) \quad (5.16)$$

n_{\max} 是序列的最大长度。

大多数解码算法遵循这种逐层搜索过程。然而, \mathcal{Y} 包含大量指数级序列, 在如此庞大的空间中进行直接搜索在计算上是不可行的。因此, 实用的解码算法通常依赖于策略来剪枝搜索空间并避免探索低质量序列。例如, 在每一步解码中, Y_i 可以通过以下方式获得

$$Y_i = \text{Prune}(Y_{i-1} \times V) \quad (5.17)$$

在 $\text{Prune}(\cdot)$ 是一个选择性移除不太可能产生高质量结果序列的函数。一般来说, 我们期望 $|Y_i| \ll |Y_{i-1}| \cdot |V|$ 。因此, 我们可以大幅减少每一步需要考虑的序列数量, 确保计算负载不会随着序列长度的增加而呈指数增长。

接下来, 我们将介绍这些解码算法。其中一些已在序列到序列模型中讨论过, 而另一些则在 LLMs 中更常用。

5.1.3.1 贪婪解码

贪婪搜索 (或贪婪解码) 是 NLP 中最广泛使用的解码方法之一, 尤其是在文本生成任务, 如机器翻译中。贪婪搜索背后的思想很简单: 在生成过程中的每一步, 它都选择具有最高预测概率的下一个标记。对于每个序列 $\mathbf{y} = y_1 \dots y_i \in Y_{i-1} \times V$, 我们可以使用 $\log \Pr(\mathbf{y}|\mathbf{x})$ 来评估它。这个对数概率可以通过注意到以下事实来轻松计算:

$$\begin{aligned} \log \Pr(\mathbf{y}|\mathbf{x}) &= \log \Pr(y_1 \dots y_i|\mathbf{x}) \\ &= \underbrace{\log \Pr(\mathbf{y}_{<i}|\mathbf{x})}_{\text{accumulated up to the parent node}} + \underbrace{\log \Pr(y_i|\mathbf{x}, \mathbf{y}_{<i})}_{\text{newly computed for the current node}} \end{aligned} \quad (5.18)$$

这里第一个项是根节点到父节点的路径对数概率之和, 该值已在之前的解码步骤中计算。在步骤 i , 我们只需要计算第二个项, 即由 LLM 产生的标准标记预测对数概率。

步骤 i 中的“最佳”标记随后被选择为

$$\begin{aligned}
 y_i^{\text{top1}} &= \arg \max_{y_i \in V} \log \Pr(y_1 \dots y_i | \mathbf{x}) \\
 &= \arg \max_{y_i \in V} \left[\underbrace{\log \Pr(\mathbf{y}_{< i} | \mathbf{x})}_{\text{fixed wrt. } y_i} + \log \Pr(y_i | \mathbf{x}, \mathbf{y}_{< i}) \right] \\
 &= \arg \max_{y_i \in V} \log \Pr(y_i | \mathbf{x}, \mathbf{y}_{< i})
 \end{aligned} \tag{5.19}$$

因此，到步骤 i 生成的“最佳”序列由以下给出

$$\mathbf{y}^{\text{top1}} = y_1 \dots y_{i-1} y_i^{\text{top1}} \tag{5.20}$$

最后， Y_i 只包含这个序列

$$Y_i = \{\mathbf{y}^{\text{top1}}\} \tag{5.21}$$

贪婪选择在一步解码中如图5.5 (a) 所示。贪婪搜索在LLM推理中提供了计算效率和实现简单性。然而，其主要缺点在于其次优性质——高质量的序列可能在解码的早期阶段被剪枝。因此，对于需要速度和简单性的任务，贪婪搜索具有吸引力。对于需要更好的搜索结果的任务，如同时探索多个潜在路径的束搜索等替代策略更可取。

5.1.3.2 光束解码

光束搜索（或光束解码）是贪婪搜索的自然扩展。与在每一步选择最可能的单个标记不同，光束搜索在每一步维护一定数量的最佳候选者，称为“光束宽度”。参见图5.5 (b) 以了解光束搜索的示例。

设 K 为光束宽度。给定一个父节点，它对应于前缀 $y_1 \dots y_{i-1}$ ，我们可以通过以下方式选择前- K 个下一个标记

$$\{y_i^{\text{top1}}, \dots, y_i^{\text{topK}}\} = \arg \text{TopK}_{y_i \in V} \Pr(y_i | \mathbf{x}, \mathbf{y}_{< i}) \tag{5.22}$$

$\arg \text{TopK}$ 是一个函数，它对所有可能的下一个标记的预测概率进行排序，并选择前 K 个候选者。给定这些标记，第 i 步的前- K 序列由以下公式给出

$$\mathbf{y}^{\text{top1}} = y_1 \dots y_{i-1} y_i^{\text{top1}} \tag{5.23}$$

...

$$\mathbf{y}^{\text{topK}} = y_1 \dots y_{i-1} y_i^{\text{topK}} \tag{5.24}$$

然后，我们可以定义 Y_i 为

$$Y_i = \{\mathbf{y}^{\text{top1}}, \dots, \mathbf{y}^{\text{topK}}\} \tag{5.25}$$

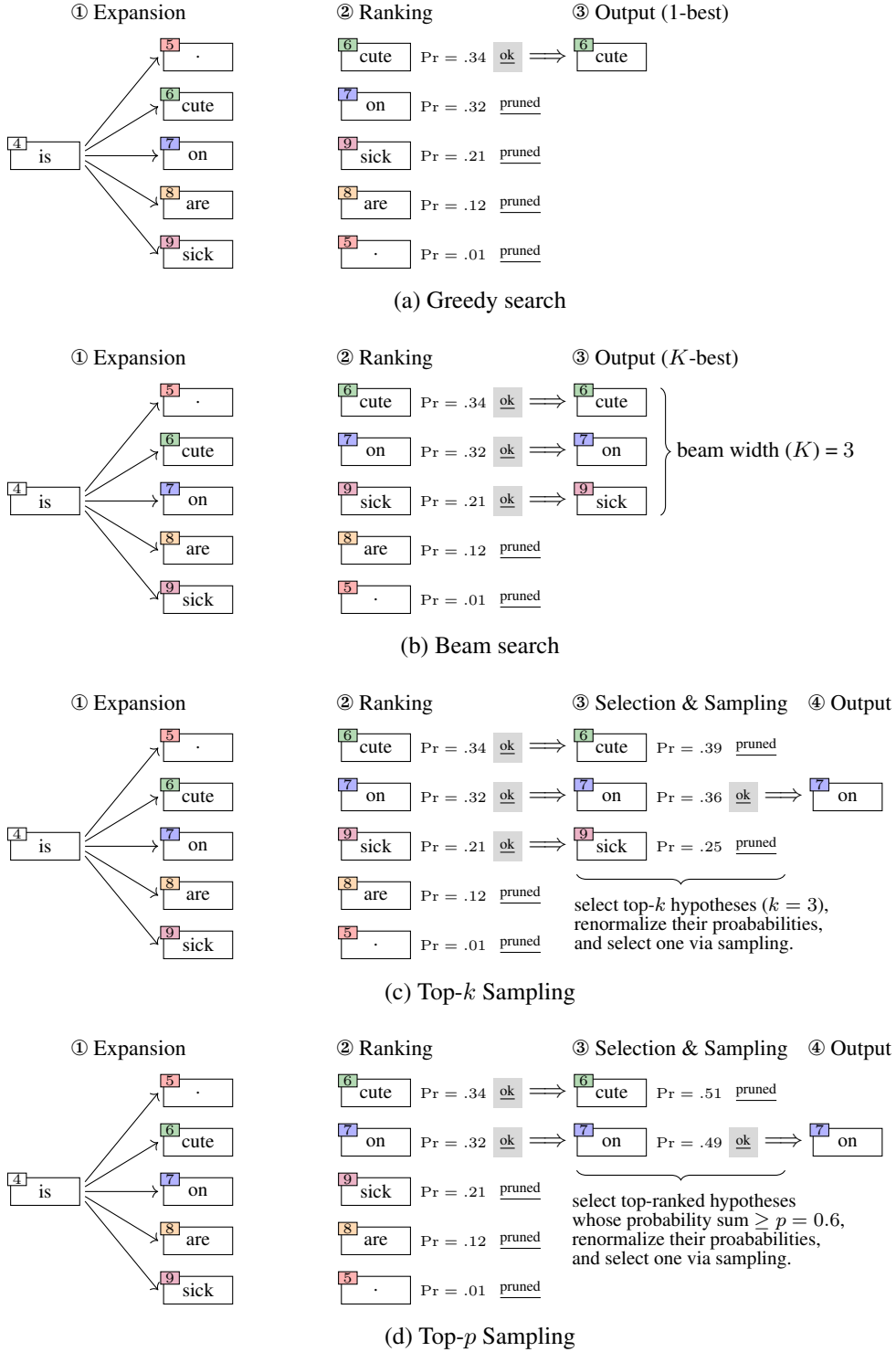


图5.5: 贪婪解码、束解码、top- k 解码和top- p 解码方法（在一个解码步骤中的）示意图。

我们可以调整光束宽度 K 以平衡搜索效率和准确性。但非常宽的光束可能并不有帮助。在许多实际应用中，选择一个相对较小的 K 数值，例如 $K = 2$ 或 $K = 4$ ，通常足以在 LLM 推理中实现令人满意的性能。

5.1.3.3 基于采样的解码

贪婪搜索和束搜索都生成确定性输出，即给定一个LLM，每次处理相同输入时模型的输出总是相同的。贪婪搜索和束搜索的确定性特性确保了在需要一致结果的应用中具有可预测性和可靠性，例如在正式文档生成中，变化的输出可能会导致混淆或错误。另一方面，这些方法的缺点是缺乏多样性和灵活性。例如，在故事生成或对话代理等创造性任务中，由确定性系统生成的通用或重复性输出往往不够吸引人。

为了在LLM输出中添加变化，我们可以使用基于采样的解码方法。有两种常用的方法。

- Top- k 采样。这种方法在生成过程的每一步都从最可能的 k 个候选者中选择下一个标记[Fan等人, 2018]。设 V_i 为top- k 采样的选择池。我们可以将其定义为

$$\bar{V}_i = \{y_i^{\text{top}1}, \dots, y_i^{\text{top}k}\} \quad (5.26)$$

在 $\{y_i^{\text{top}1}, \dots, y_i^{\text{top}k}\}$ 是基于它们的预测概率选择的top- k 令牌（参见公式（5.22））。一旦确定选择池，我们就在 V_i 上重新计算预测概率分布。这样做的一种最简单的方法是重新归一化它们的概率：

$$\bar{\text{Pr}}(y_i|\mathbf{x}, \mathbf{y}_{<i}) = \frac{\text{Pr}(y_i|\mathbf{x}, \mathbf{y}_{<i})}{\sum_{y_j \in \bar{V}_i} \text{Pr}(y_j|\mathbf{x}, \mathbf{y}_{<i})} \quad (5.27)$$

另一种方法是使用Softmax函数来计算分布：

$$\bar{\text{Pr}}(y_i|\mathbf{x}, \mathbf{y}_{<i}) = \frac{\exp(u_{y_i})}{\sum_{y_j \in \bar{V}_i} \exp(u_{y_j})} \quad (5.28)$$

u_{y_i} 是标记 y_i 的logit。然后，我们从这个分布中采样一个标记 \bar{y}_i ：

$$\bar{y}_i \sim \bar{\text{Pr}}(y_i|\mathbf{x}, \mathbf{y}_{<i}) \quad (5.29)$$

相应的序列是 $\bar{\mathbf{y}} = y_1 \dots y_{i-1} \bar{y}_i$ ，且 Y_i 由以下给出

$$Y_i = \{\bar{\mathbf{y}}\} \quad (5.30)$$

- Top- $\{v^*\}$ 采样。这种方法，也称为核采样，遵循与top- $\{v^*\}$ 采样类似的程序。它不是从固定大小的候选池中抽取，而是从具有累积概率高于预定义阈值 $\{v^*\}$ 的最小令牌集中选择下一个令牌[Holtzman等人, 2020]。通过这种方式，我们防止预测从可能导致不连贯或无意义的输出的长尾低概率令牌中选择。要在top- $\{v^*\}$ 采样方法中获得候选池，我们可以按预测概率对所有令牌进行排序。然后，从概率最高的令牌开始，我们继续向候选池中添加令牌，直到

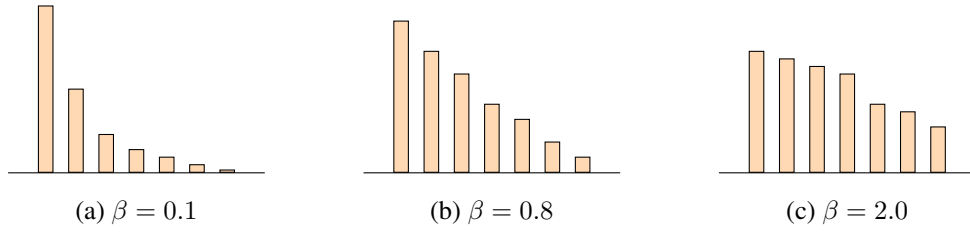


图5.6: 不同温度参数 β 下由Softmax函数生成的分布的直方图估计

累积概率达到或超过 $\{v^*\}$ 的池中标记，我们称这时候选池的大小为 k_p 。候选池可以表示为

$$\bar{V}_i = \{y_i^{\text{top}1}, \dots, y_i^{\text{top}k_p}\} \quad (5.31)$$

后续步骤，如分布重整化和采样，与 top- k 采样方法（参见公式(5.27-5.30)）相同。

查看图5.5(c-d)以了解top- k 和top- p 采样方法的示例。通过将选择限制在更小的高概率标记集，这些方法在随机性和连贯性之间取得平衡。它们允许输出更加多样化，同时仍然保持合理的相关性和流畅性。然而， k 或 p 的值必须仔细选择：如果 k 或 p 太小，输出可能仍然过于确定（更像是贪婪解码），如果 k 或 p 太大，LLM可能会产生退化输出。

为了进一步控制令牌选择过程的随机性，通常通过使用具有温度参数的Softmax函数来获得重归一化分布 $\text{Pr}(\cdot)$ ，给定为

$$\bar{\text{Pr}}(y_i | \mathbf{x}, \mathbf{y}_{<i}) = \frac{\exp(u_{y_i} / \beta)}{\sum_{y_j \in \bar{V}_i} \exp(u_{y_j} / \beta)} \quad (5.32)$$

这里 β 是一个温度参数 β ，它控制从 logits 导出的概率分布的尖锐程度。在图 5.6 中，我们展示了涉及由上述函数生成的不同温度分布的简单示例。当温度设置为更高的值时，得到的概率分布变得更加均匀，因为 logits 之间的差异减小。这意味着候选池中的每个标记都有更大的机会被选中，从而导致生成的输出更加多样化。相比之下，当温度设置为较低的值时，分布变得更加尖锐，使得高概率标记更有可能被选中，这通常会导致更确定性的输出。例如，如果我们将 p 设置为 1，将 β 设置为一个非常小的数字（接近零），则 top- p 采样方法将等同于贪婪搜索方法。

5.1.3.4 带惩罚项的解码

一种常见的改进文本生成中的解码方法是对搜索目标进行修改。例如，可以将最大后验概率（MAP）解码替换为最小贝叶斯风险（MBR）解码 [Kumar 和 Byrne, 2004]，其中焦点从选择单个

最可能输出是选择一个在可能输出分布中使预期风险最小化的输出。在这里，我们探讨了将惩罚项纳入解码的方法。这些方法提供了一种简单而有效的方法来使解码更具可控性。

从公式 (5.14) 中回忆，解码的目标是最大化输出序列的可能性。引入惩罚项后，目标被扩展以包括惩罚或奖励生成文本中某些行为的额外因素。新目标的一般形式由以下给出

$$\hat{\mathbf{y}} = \arg \max_{\mathbf{y} \in \mathcal{Y}} [\Pr(\mathbf{y}|\mathbf{x}) - \lambda \cdot \text{Penalty}(\mathbf{x}, \mathbf{y})] \quad (5.33)$$

在何处惩罚(\mathbf{x}, \mathbf{y})是一个函数，它量化了生成的序列 \mathbf{y} 在给定输入 \mathbf{x} 的情况下违反某些约束或表现出不良行为的程度。惩罚(\cdot)的设计非常灵活，因此我们可以将其纳入广泛的约束或先验知识中。下面，我们介绍一些常见的惩罚函数类型。

- 重复惩罚。重复惩罚会阻止模型生成重复或冗余的文本。惩罚函数可能会衡量生成序列中重复标记或短语的频率，并按其出现次数施加相应的惩罚。
- 长度惩罚。长度惩罚确保生成的序列符合期望的长度。例如，在文本摘要任务中，惩罚函数可以对过短或过长的输出进行惩罚。
- 多样性惩罚。多样性惩罚促进生成文本的多样性。例如，在束搜索中，我们可以测量生成假设之间的相似性，并鼓励模型探索不同的假设。
- 基于约束的惩罚。基于约束的惩罚强制执行与生成文本的内容或风格相关的特定约束。例如，在机器翻译中，惩罚函数可能会惩罚偏离期望语气或术语的输出。

通常，我们可以将 $\text{Penalty}(\mathbf{x}, \mathbf{y})$ 视为一个函数，它定义了给定输入序列 \mathbf{x} 的情况下生成输出序列 \mathbf{y} 的表面形式的成本。或者，这个函数可以定义为评估在生成 \mathbf{y} 时LLM的隐藏状态。例如，Su等人[2022]开发了一个惩罚项，该惩罚项计算预测标记的表示与先前生成的标记的表示之间的最大距离。因此，搜索目标将惩罚退化输出，例如具有许多重复的文本。

该方法如公式 (5.33) 所述，是通用的，并且可以轻松地适应不同的搜索算法。例如，在贪婪搜索中，我们可以在每个解码步骤中保持最大化 $\Pr(\mathbf{y}|\mathbf{x}) - \lambda \cdot \text{Penalty}(\mathbf{x}, \mathbf{y})$ 的单个序列；在基于采样的搜索中，我们可以根据 $\Pr(\mathbf{y}|\mathbf{x}) - \lambda \cdot \text{Penalty}(\mathbf{x}, \mathbf{y})$ 对序列进行排名和选择，以形成候选池。

5.1.3.5 投机性解码

推测性解码源于推测执行的概念，其中系统对未来动作做出有根据的猜测并在事先执行它们。如果猜测正确，那么

结果立即可用，这加快了处理速度。在LLM推理的情况下，假设我们有两个模型。一个是较小的、较快的模型（称为草稿模型），另一个是完整、更准确的模型（称为验证模型）。这两个模型代表了LLM推理中的两个基线：草稿模型效率高但不太准确；验证模型通常是我们要运行的模型，但它非常慢。给定一个前缀，我们首先使用草稿模型来推测一系列可能的未来标记。这是一个标准的自回归解码过程，但由于草稿模型的高效率，在实践中仍然很快。然后，验证模型并行评估推测的标记。它检查预测的标记是否正确或需要调整。请注意，由于我们可以一次性处理这些标记，因此验证可以同时对所有标记进行，而不是逐个标记进行。如果推测的标记是正确的，则接受它们，并继续下一组标记的过程。如果它们是不正确的，则丢弃错误的推测，并使用验证模型生成正确的标记。

更具体地说，让我们看看Leviathan等人[2023]的工作中提出的投机解码方法。在此方法中，草案模型是一个小型语言模型，表示为 $\Pr_q(y_i|\mathbf{x}, \mathbf{y}_{<i})$ ，而验证模型是一个正常的LLM，表示为 $\Pr_p(y_i|\mathbf{x}, \mathbf{y}_{<i})$ 。目标是，给定一个前缀，我们使用草案模型进行自回归预测，直到 τ 个标记。然后，验证模型被用来在投机预测开始出现错误的地方生成最后一个标记。图5.7说明了解码过程中的一个步骤。

投机解码算法可以概括如下。

- 给定前缀 $[\mathbf{x}, \mathbf{y}_{\leq i}]$ ，我们使用草稿模型来预测接下来的 τ 个连续标记，记为 $\{\hat{y}_{i+1}, \dots, \hat{y}_{i+\tau}\}$ 。这是一个逐个标记的生成过程，表示为

$$\hat{y}_{i+t} = \arg \max_{y_{i+t}} \Pr_q(y_{i+t}|\mathbf{x}, \mathbf{y}_{\leq i}, \hat{y}_{i+1} \dots \hat{y}_{i+t-1}) \quad (5.34)$$

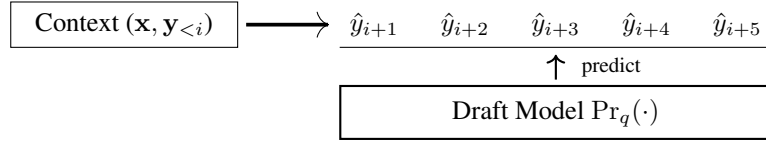
- 我们使用验证模型评估 $\{\hat{y}_{i+1}, \dots, \hat{y}_{i+\tau}\}$ ，即我们计算 $\{\Pr_p(\hat{y}_{i+1}|\mathbf{x}, \mathbf{y}_{\leq i}), \dots, \Pr_p(\hat{y}_{i+\tau}|\mathbf{x}, \mathbf{y}_{\leq i}, \hat{y}_{i+1} \dots \hat{y}_{i+\tau-1})\}$ 。请注意，我们可以并行计算这些概率，因此此验证步骤是高效的。
- 我们确定接受推测标记的最大数量。为了使符号简洁，我们分别用 $q(\hat{y}_{i+t})$ 和 $p(\hat{y}_{i+t})$ 表示 $\Pr_q(\hat{y}_{i+t}|\mathbf{x}, \mathbf{y}_{\leq i}, \hat{y}_{i+1} \dots \hat{y}_{i+t-1})$ 和 $\Pr_p(\hat{y}_{i+t}|\mathbf{x}, \mathbf{y}_{\leq i}, \hat{y}_{i+1} \dots \hat{y}_{i+t-1})$ 。然后我们定义，如果 $q(\hat{y}_{i+t}) \leq p(\hat{y}_{i+t})$ ，则我们接受这个推测。相比之下，如果 $q(\hat{y}_{i+t}) > p(\hat{y}_{i+t})$ ，我们以概率 $1 - \frac{p(\hat{y}_{i+t})}{q(\hat{y}_{i+t})}$ 拒绝这个推测。从 \hat{y}_{i+1} 开始，定义接受的最大连续推测标记数量为

$$n_a = \min \left\{ t - 1 \mid 1 \leq t \leq \tau, r_t > \frac{p(\hat{y}_{i+t})}{q(\hat{y}_{i+t})} \right\} \quad (5.35)$$

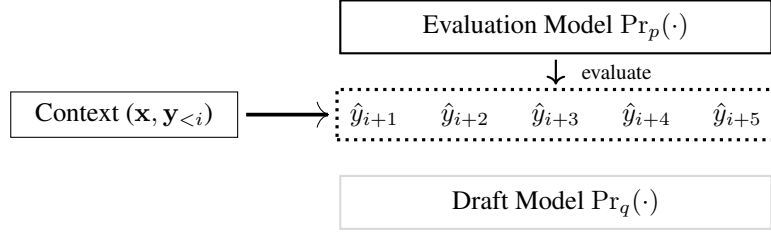
在 r_t 是从均匀分布 $U(0, 1)$ 中抽取的变量。

- 给定 n_a ，我们保留推测的标记 $\{\hat{y}_{i+1}, \dots, \hat{y}_{i+n_a}\}$ 。然后我们使用验证模型在 $i + n_a + 1$ 进行新的预测

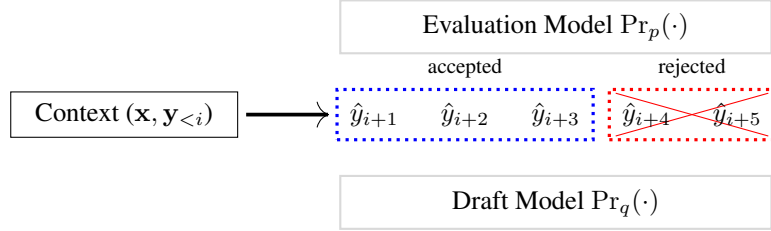
$$\bar{y}_{i+n_a+1} = \arg \max_{y_{i+n_a+1}} \Pr_p(y_{i+n_a+1}|\mathbf{x}, \mathbf{y}_{\leq i}, \hat{y}_{i+1} \dots \hat{y}_{i+n_a}) \quad (5.36)$$



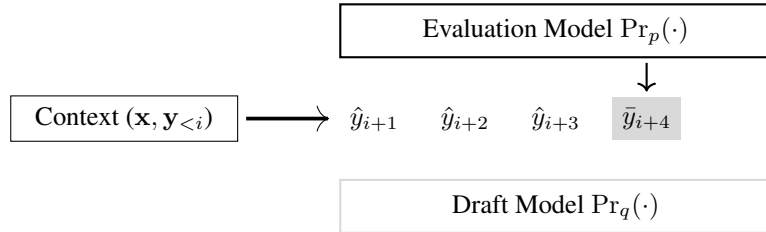
(a) 预测给定上下文下的下一个 τ 个标记，使用草稿模型 ($\tau = 5$)



(b) Evaluate the predicted tokens using the evaluation model



(c) Determine the number of accepted tokens



(d) 使用评估模型预测接受后的新标记

图5.7: 推测解码一步的说明。目标是使用草稿模型尽可能预测尽可能多的下一个标记。有四个子步骤。给定上下文，我们首先使用草稿模型预测下一个 τ 标记。然后，我们使用评估模型并行评估这些预测。接下来，我们确定可以接受的预测标记的最大数量。最后，我们使用评估模型预测这些已接受标记之后的下一个标记。

- 以上，我们描述了推测解码的一步。结果序列（包括上下文和预测标记）如下所示

$$\begin{array}{ccc}
 \underbrace{[\mathbf{x}, \mathbf{y}_{<i}]}_{\text{Context}} & \underbrace{\hat{y}_{i+1} \dots \hat{y}_{i+n_a}}_{\substack{n_a \text{ tokens} \\ \text{predicted using} \\ \text{the draft model}}} & \underbrace{\bar{y}_{i+n_a+1}}_{\substack{\text{One token} \\ \text{predicted using} \\ \text{the verification model}}}
 \end{array}$$

一旦我们完成这一步，我们将预测的标记 $\{\hat{y}_{i+1}, \dots, \hat{y}_{i+n_a}, \bar{y}_{i+n_a+1}\}$ 添加到上下文中，并重复上述过程。

实际上，我们通常希望使用较小的草稿模型，以便预测 $\{\hat{y}_{i+1}, \dots, \hat{y}_{i+n_a}\}$ 的计算成本更低。但一个非常小的草稿模型准确性较低，可能导致较小的 n_a 。因此，我们需要仔细选择草稿模型，以在计算效率和准确性之间进行权衡。

5.1.3.6 停止标准

停止标准是LLM推理的关键组成部分。它们通常涉及规则或条件，用于指定模型应在解码过程中何时停止生成文本。大多数LLM都经过训练以生成一个序列结束标记（例如， $\langle \text{EOS} \rangle$ 或 $\langle /s \rangle$ ）来表示生成文本的结束。因此，最简单的策略之一是在生成此标记时停止生成过程。对于同时探索多个假设的beam search，过程可以继续，直到生成给定数量的完整序列。

在实用应用中，通常不希望生成非常长的序列，因此我们需要降低解码成本和不必要的冗余。一个常用的停止标准是输出最大长度。模型在生成预定数量的标记后停止生成文本。或者，我们可以根据实际成本停止解码，例如计算资源或时间限制。例如，在实时应用如聊天机器人中，解码可能需要在一定时间限制后停止，以确保响应性。

另一种方法是设计基于LLM行为的停止标准。例如，如果预测下一个标记的概率低于某个阈值，则可以停止解码。除了基于概率的停止之外，还可以实现一个重复检测模块，如果模型开始重复标记或短语超过预定义的限制，则触发模型停止。这有助于防止冗余或不连贯的输出。

5.1.4 LLM 推理的评估指标

评估LLMs在推理过程中的性能涉及各种指标，以评估这些模型满足预期标准的情况，例如准确性、鲁棒性、可用性和效率。与大多数NLP系统一样，我们可以使用基于准确性的指标来评估LLMs，例如困惑度和F1分数。我们还可以通过测试它们处理模糊或具有挑战性的输入（包括对抗性、扰动或分布外数据）的能力来检查它们的鲁棒性。此外，可以通过衡量生成的输出在流畅性、连贯性、相关性和多样性方面与用户期望的匹配程度来评估可用性。人类评估者可以对文本的自然度进行评分或评估响应是否在上下文中适当且逻辑上一致。还可以包括道德和公平性指标，以确保LLMs避免持续偏见或生成有害内容。

所有上述提到的评估指标本质上都集中在评估输出质量上。鉴于部署和应用LLMs的高成本，效率指标对于从业者来说也非常重要。以下是一些常用的效率指标 [Nvidia, 2025]:

- 请求延迟。此指标衡量从向LLM发送请求到收到完整响应所花费的总时间。这包括数据传输、模型处理和将输出返回给用户的时间。
- 吞吐量。它指的是模型每秒可以处理的标记或请求数量。
- 首次标记时间（TTFT）。此指标衡量从请求开始发送到生成响应的第一个标记所需的时间。如果数据传输不消耗太多时间，那么TTFT主要是预填充和预测第一个标记的时间。
- 令牌间延迟（ITL）。此指标指的是生成第一个令牌之后的每个后续令牌所需的时间。它反映了解码过程的效率。
- 每秒令牌数（TPS）。此指标量化了模型每秒可以生成的令牌数量。
- 资源利用率。这涉及到在推理过程中测量模型的计算资源使用（例如，CPU和GPU利用率）以及内存消耗。

除了这些指标外，能源效率和成本效率是大规模部署LLMs的实际考虑因素。能源效率衡量模型在推理过程中消耗的电能。另一方面，成本效率评估与部署和维护模型相关的总费用。

一般来说，选择合适的评估指标取决于具体任务和应用。虽然以质量为导向的指标对于评估LLM至关重要，但效率指标对于它们在现实世界中的应用同样关键。一个全面的评估框架应包括这两组指标，以准确估计LLM的性能和实用性。

5.2 高效推理技术

在实用应用中，我们通常希望系统尽可能高效。对于LLM推理，这通常涉及两种类型的改进：减少内存需求和加速系统。例如，我们可以修改Transformer架构以避免处理非常长的输入序列时的内存爆炸。另一个例子是我们可以通过压缩输入序列来减少计算开销，同时保留它们的语义信息。此外，还可以采用量化、剪枝等技术来进一步优化内存使用和推理速度。

高效推理是一个涵盖多个LLM子领域（如架构设计和模型压缩）的广泛主题。其中大部分主题已在之前的

章节。例如，在第2章中，我们讨论了高效的Transformer架构和长上下文LLM；在第3章中，我们讨论了用于减少提示长度的提示压缩方法。在本节中，我们关注在LLM部署和提供服务中常用的技术。

5.2.1 更多缓存

在现实世界的应用中，将频繁请求及其对应的响应存储在缓存中是一种常见的做法。当新的请求击中缓存时，系统可以直接从缓存中检索响应，而不是重新计算结果。一种直接的实施方法是键值数据存储（例如，哈希表），它将输入序列映射到其LLM生成的输出序列。在最简单的情况下，我们可以收集频繁查询，使用LLM生成其响应，并将这些查询-响应对存储在数据存储中。这创建了一个基本的序列级缓存机制，允许系统在输入序列与缓存查询完全匹配时绕过LLM计算。

一种简单的缓存机制扩展是缓存前缀及其对应的隐藏状态。给定数据集 \mathcal{D} 中的一个输入序列 \mathbf{x} ，我们可以像标准预填充阶段一样处理它。因此，我们获得一系列前缀及其对应的 KV 缓存状态：

$$\begin{aligned} x_0 (\mathbf{x}_{<1}) &\Rightarrow \text{cache}_{<1} \\ x_0 x_1 (\mathbf{x}_{<2}) &\Rightarrow \text{cache}_{<2} \\ &\dots \\ x_0 x_1 \dots x_{m-1} (\mathbf{x}_{<m}) &\Rightarrow \text{cache}_{<m} \end{aligned}$$

缓存 $\{v^*\}$ 表示前缀 $\mathbf{x}_{<i}$ (的KV缓存，也参见公式(5.10))。所有这些映射都可以存储在前缀缓存中以实现高效重用。

当处理与之前在 \mathcal{D} 中看到的序列共享公共前缀的新序列时，我们可以加载相应的缓存隐藏状态，而不是重新计算它们。具体来说，如果新的输入 \mathbf{x}' 有 $\mathbf{x}_{<k}$ (即， $\mathbf{x}'_{<k} = \mathbf{x}_{<k}$ 对于某个 $k \leq m$)，我们可以使用 $\text{cache}_{<k}$ 初始化 KV 缓存，并且只计算剩余标记 $\mathbf{x}'_{\geq k}$ 的隐藏状态。

通常，我们可以维护一个键值数据存储，将频繁遇到的前缀映射到它们的预计算键值缓存。可以使用前缀标记的哈希来执行查找，从而实现缓存状态的常数时间访问。必须注意管理内存使用，因为存储所有可能的前缀对于大数据集可能是不切实际的。实际系统通常采用最近最少使用 (LRU) 缓存方法或其他策略，在计算节省和内存限制之间取得平衡。

5.2.2 批处理

批处理在LLM推理中指的是将多个输入序列同时作为一个组（称为批次）进行处理，而不是逐个处理。由于现代GPU擅长并行处理，批处理允许它们在单次正向传递中计算多个序列，使硬件保持完全占用。因此，在规模上提供LLM时，批处理对于

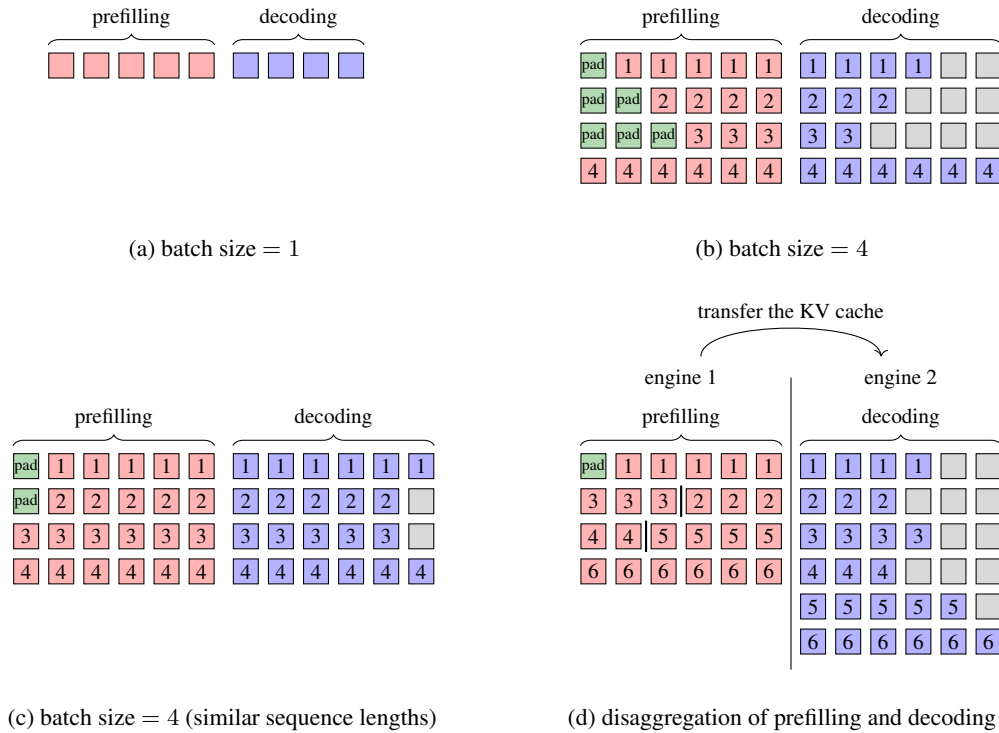


图5.8: 基本批处理方法的示意图。我们使用2D布局来表示批处理, 其中每个方块代表一个标记。红色方块表示预填充阶段的标记, 蓝色方块表示解码阶段的标记, 绿色方块表示填充标记, 灰色方块对应无意义的标记。子图 (a) 和 (b) 分别比较了批处理大小为1和4的情况。子图 (c) 显示了将具有相似长度的序列分组到同一批中的策略。子图 (d) 说明了预填充和解码的分解。在这种方法中, 我们可以通过将多个短序列连接成一个长序列以进行联合处理, 更好地利用GPU的并行性。这允许我们在最小化填充标记数量的同时, 最大化批处理中处理的标记数量。然而, 作为权衡, 我们需要在预填充阶段后将KV缓存复制到解码引擎并重新组织, 这引入了额外的数据传输开销。

提高计算效率并最大化硬件利用率²。

为了说明批处理的概念, 图5.8 (a-b) 展示了分别以1和4为批量大小的简单示例。当使用1个批量大小时 (即不进行批处理), GPU每次处理一个输入序列。因此, 处理是顺序的: 下一个序列必须等待当前计算完成。相比之下, 当使用4个批量大小时, GPU可以在单个正向传递中同时处理四个序列。由于输入序列的长度不同, 我们需要使用填充技术来标准化它们的长度。在这里, 我们使用左填充, 即在短序列的开头添加虚拟标记, 以便批处理中的所有序列都具有相同的长度以进行预填充。对于解码, 为所有这些序列同时生成标记, 并且生成过程一直持续到最长的序列完成。

上述示例暗示了吞吐量和延迟之间的权衡, 这在设计和实现LLM推理系统中是一个非常重要的考虑因素。如果我们选择较小的批量大小, 延迟会降低, 因为单个推理运行中需要处理的标记会更少。想象一下我们只有一个序列。生成完成后, 结果立即可用, 没有额外的计算开销。然而, 这种低延迟的优势是以未充分利用并行计算资源为代价的, 因为并行性为

²See <https://docs.nvidia.com/deeplearning/performance/dl-performance-gpu-background/index.html#understand-perf> for a simple evaluation.

GPU在顺序处理期间大部分时间处于空闲状态。另一方面，如果我们使用更大的批量，我们可以更好地利用并行性，因为GPU可以被大规模矩阵计算占用。因此，我们可以在相同的时间内处理更多的标记，并且吞吐量得到提高。然而，由于结果仅在批量中的最后一个标记被预测时获得，延迟会更高。

实际上，我们通常更喜欢使用稍大的批次，但尽量用相似长度的序列填充批次以减少填充标记的数量并提高设备利用率。例如，我们可以将短时间内到达的用户请求分组到桶中，每个桶包含相似长度的序列。然后，我们可以用同一桶中的序列填充批次，这样就可以最小化浪费的计算资源，如图5.8（c）所示。

另一种在LLMs中实现批处理的方法是将预填充和解码过程分解[吴等，2023a；帕特尔等，2024；钟等，2024]。例如，我们可以在一个GPU上执行预填充，在另一个GPU上执行解码。分解的一个优点是我们可以重新排列批次的输入序列以更好地填充，因为预填充和解码之间没有干扰。例如，我们可以将多个短序列连接成一个较长的序列，从而确保批次中序列的长度尽可能一致，如图5.8（d）所示。这样，我们可以最大化预填充阶段的吞吐量。然而，作为权衡，我们需要将KV缓存传输到执行解码的设备，这也产生了额外的通信开销。通常，这种方法需要一个高带宽、低延迟的网络来实现最佳性能。

在这个部分，我们将讨论上述基本批处理策略的几个改进。其中大部分基于聚合架构，即解码和预填充可以被视为在同一设备上执行的一个模型的不同阶段。

5.2.2.1 调度

一个实用的LLM推理系统通常由两个组件组成：

- 调度器。其主要作用是根据当前系统负载和任务优先级，高效地将任务（即输入序列）排队和调度到推理引擎。这通常涉及各种批处理策略，这些策略将某些请求分组在一起，以某种方式最大化处理效率。
- 推理引擎。它负责LLMs的实际执行，处理按顺序到达的请求。如前所述，此引擎涉及预填充和解码过程。

此架构如图5.9所示。将调度纳入批量处理提供了一种灵活的方式来优化系统的吞吐量和延迟，从而在它们之间实现更好的平衡。例如，图5.8（a）和（b）中显示的批量方法可以被认为是一种最简单的调度策略，称为请求级调度。在这种策略中，一旦一个批次被填满并发送到引擎，整个批次的处理就不能

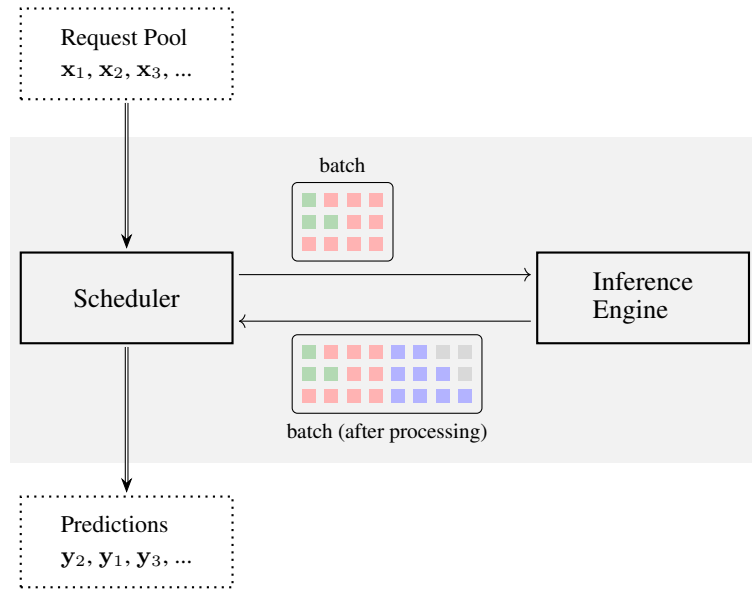


图5.9: 涉及调度器和推理引擎的LLM推理架构示意图。每次，调度器选择一定数量的用户请求形成一个批次并发送到推理引擎。调度器可以在推理过程中与推理引擎交互并调整批次，例如在批次处理开始时和在每次标记预测开始时。

中断。调度器等待此批次处理完毕后再处理下一个批次 [Timonin 等人, 2022]。

一种更复杂的调度策略，称为基于迭代的调度，在每个标记预测步骤与推理引擎交互，而不是在序列级别。这种方法允许在推理过程中进行动态批量调整，如图5.10所示。这种细粒度控制允许系统实时优先处理关键标记或序列。例如，如果紧急请求在某个解码步骤到达，调度器可以将此请求添加到批次中，以便尽可能早地处理。在以下小节中，我们将讨论基于迭代调度的批量方法。

5.2.2.2 持续分批

连续批处理是Orca系统中使用的一种基于迭代的调度方法 [Yu et al., 2022]。在此方法中，一个迭代指的是整个预填充过程或单个解码步骤。例如，给定一个输入序列 $x = x_0 \dots x_m$ 和一个输出序列 $y = y_1 \dots y_n$ ，总共有 $n + 1$ 次迭代：一次用于预填充， n 次用于生成输出标记（每个标记一次）。在调度过程中，可以在迭代之间调整批次。例如，我们可以在某些迭代中向批次添加新的输入序列，或者在某些迭代中从批次中移除完整的序列，即使批次处理尚未完成。

连续批处理的一般过程包括以下步骤：

- 最初，根据推理引擎的处理能力和当前用户请求，创建一个包含一个或多个输入序列的批次。然后，将批次输入到推理引擎中。

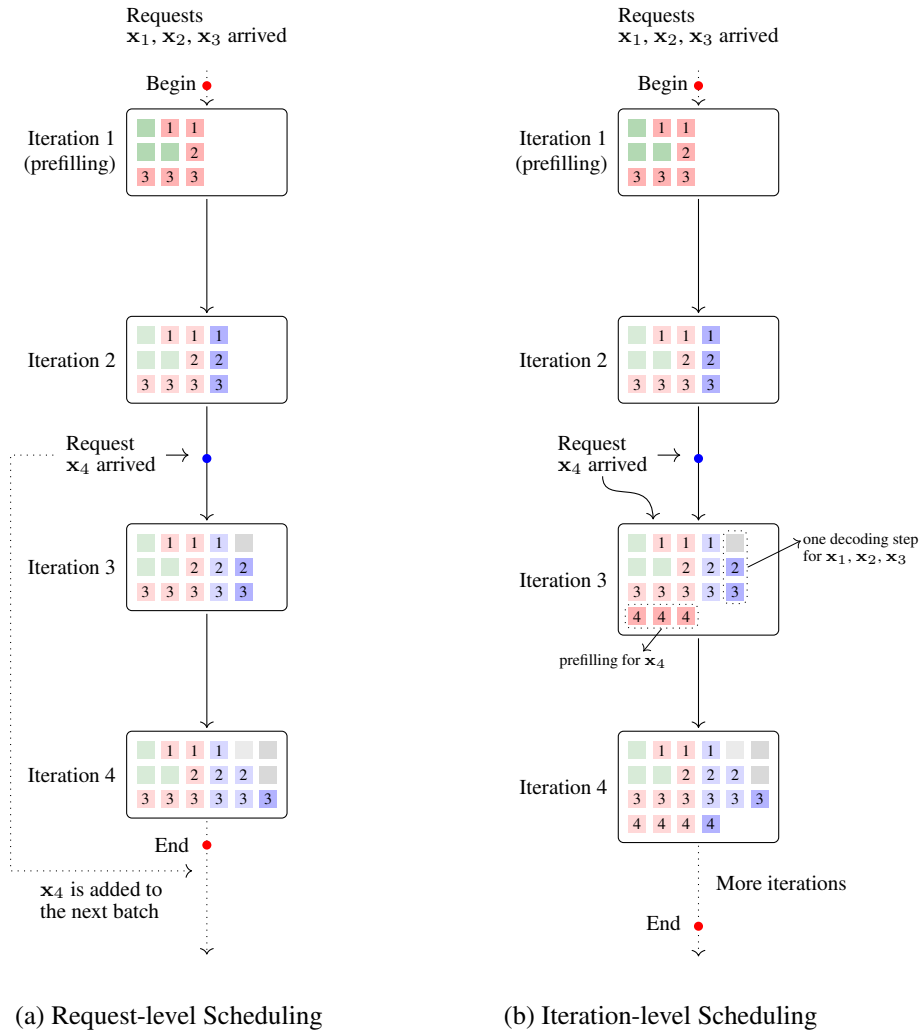


图5.10: 请求级调度和迭代级调度的说明。在请求级调度中，一旦创建并发送批次到推理引擎，我们无法调整批次。换句话说，调度仅在批次处理完成后发生。在迭代级调度中，我们可以在批次处理期间进行调度。例如，如果在推理过程中某个时刻收到新的请求，我们可以将其添加到批次中并继续处理。

- 推理引擎按批次逐次迭代处理。每次迭代后，调度器可能会以下列方式之一调整批次：

- 如果批次中的某个序列完成生成（即生成序列结束符号），则该序列将从批次中移除。
- 如果有新用户请求到达且推理引擎有额外的处理能力，则将其添加到批次中。
- 如果没有序列被添加到或从批次中移除，则批次保持不变。

- 处理仅在所有序列完成且没有新的用户请求到达时终止。

查看图5.11以获取连续批次的示例。在此示例中，我们从一个用户开始

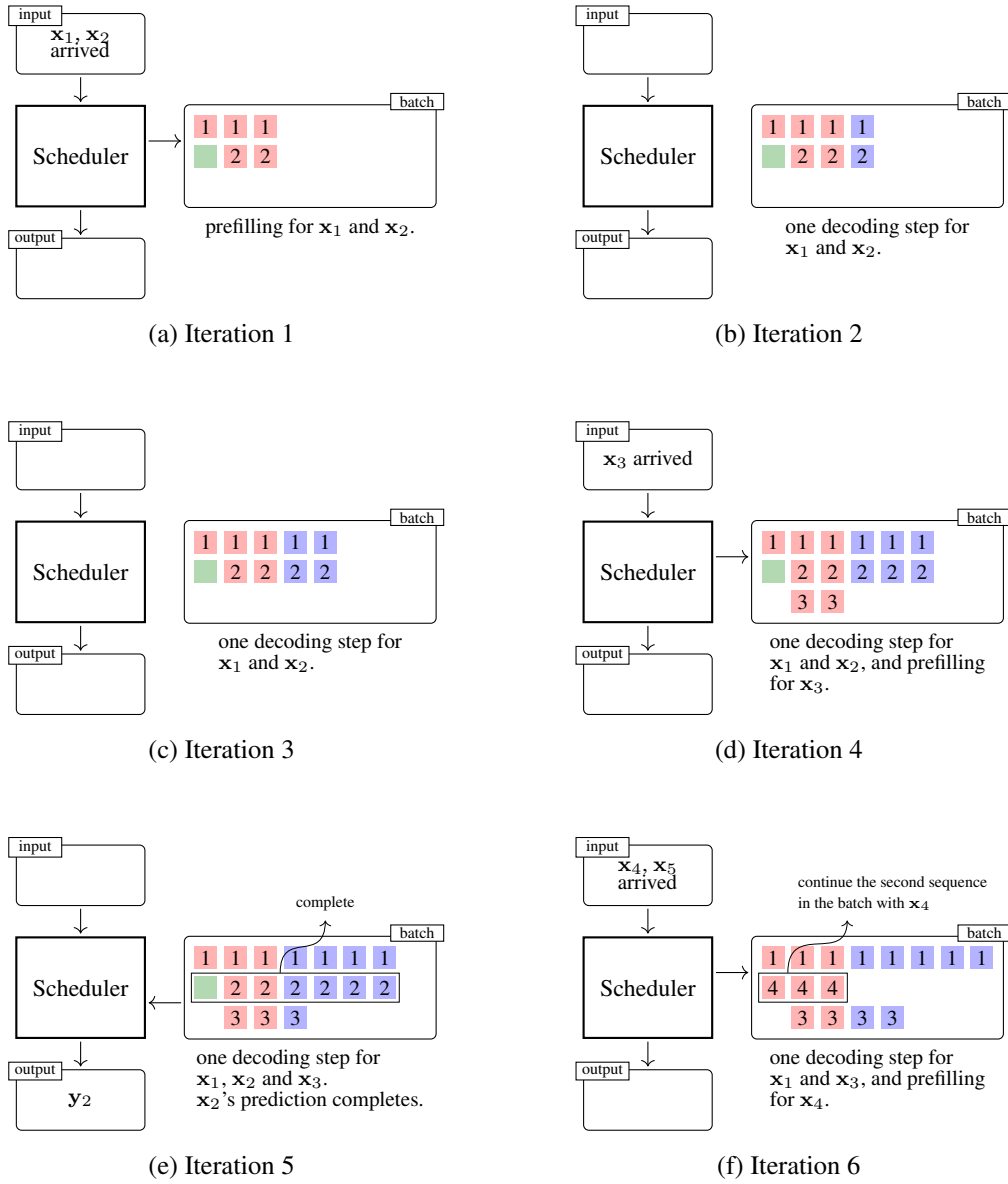


图5.11：连续批量调整的说明。与固定一批输入序列并完成处理（如请求级批量）不同，连续批量在推理过程中动态更新批量。只要存在可用的计算能力，系统就会持续接受并添加新的请求（例如， x_3 和 x_4 ）到当前批量中。

请求数据， x_1 和 x_2 。这两个序列被打包成一个批次并发送到推理引擎进行处理。在引擎完成两次迭代后，一个新的用户请求 x_3 到达。此时，调度器通过添加 x_3 来调整批次。推理引擎随后继续处理更新后的批次。请注意，推理引擎现在以不同的方式处理不同的序列： x_1 和 x_2 继续进行解码过程（即预测下一个标记），而 x_3 则进行预填充过程。经过一段时间后， x_2 的生成完成。此时，还有两个用户请求 x_4 和 x_5 到达。调度器从批次中移除已完成的序列 x_2 ，并考虑到推理引擎当前的负载，将 x_4 添加到批次中。然而， x_5 必须等待批次中的另一个序列完成才能被添加。

连续批处理背后的想法是通过尽可能处理更多序列来保持推理引擎充分利用，从而最大化计算资源的使用。连续批处理与标准批处理（见图5.8）的一个关键区别在于，在连续批处理中，预填充和解码可以跨不同序列同时发生，而在标准批处理中，这两个阶段在整个批次中是顺序执行的。如第5.1.2节所述，预填充被认为是一个计算密集型过程，而解码被认为是一个内存密集型过程。重叠预填充和解码背后的直觉是减少计算和数据传输的空闲时间。考虑两个小批量：一个用于预填充，一个用于解码。当预填充小批量保持GPU忙碌时，解码小批量可以同时执行内存传输。

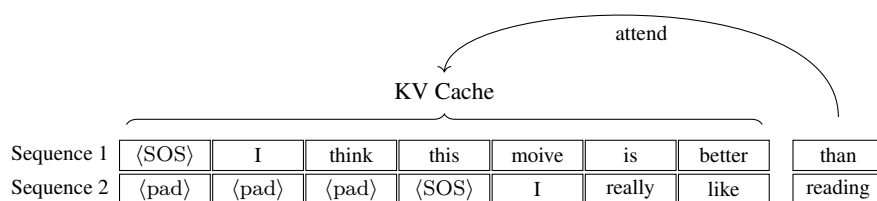
另一个连续批处理和标准批处理之间的区别是，连续批处理是优先填充，而标准批处理是优先解码 [Agrawal 等人, 2024]。在连续批处理中，一旦推理引擎有额外的计算资源，调度器将添加新的请求到批次中。换句话说，这些新添加的请求将尽可能早地被处理用于填充。这种方法提高了系统吞吐量，但代价是增加了延迟，因为新添加的请求延长了早期请求的处理时间。相比之下，在标准批处理中，一旦批次创建，我们必须等待批次中的最后一个序列完成，然后才能处理新的请求。这确保了相对较低的延迟，但导致设备利用率和系统吞吐量较低。

重要的是要注意，连续批处理的开销是我们需要不断重新组织批次，这涉及到在内存中重新排列数据。每次添加新请求时，调度器都需要重新评估和优化当前的批次结构。这种动态调整可能会产生额外的内存和计算开销，尤其是在批次频繁调整时。因此，虽然这种方法可以提高吞吐量，但它也可能导致内存碎片化增加，在某些情况下，还会引入额外的延迟。

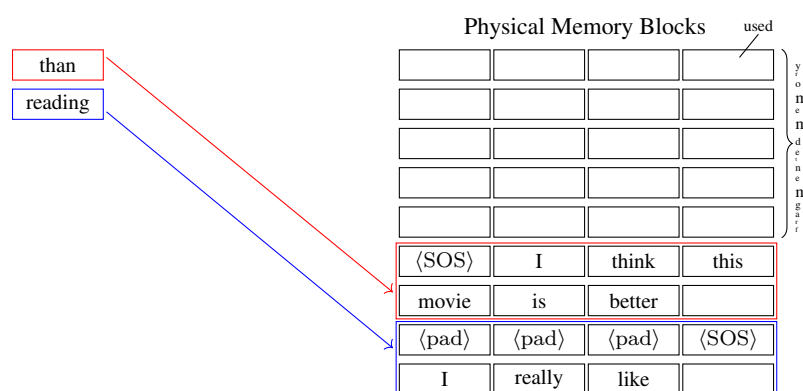
5.2.2.3 分页注意力

分页注意力（或分页 KV 缓存）是 vLLM 系统中 [Kwon 等人, 2023] 使用的一种技术。受操作系统分页的启发，它通过解决动态批处理场景中可变长度序列的碎片化内存分配，优化了 LLM 推理过程中的内存使用——尤其是对于 KV 缓存。PagedAttention 的理念是将 KV 缓存的大内存需求分解成更易于管理的“页面”或内存块。这样，我们不需要在连续的内存中存储整个序列的 KV 缓存。相反，KV 缓存被分成固定大小的块（类似于操作系统的内存页面），可以在物理内存中非连续分配。PagedAttention 的一个优点是它实现了灵活的内存管理，支持动态序列增长，而无需进行昂贵的重新分配或复制大连续内存区域。请注意，PagedAttention 并非专门为批处理设计。但它确实有助于提高批处理推理场景中的内存效率，在这些场景中，内存管理要求更高且更复杂。

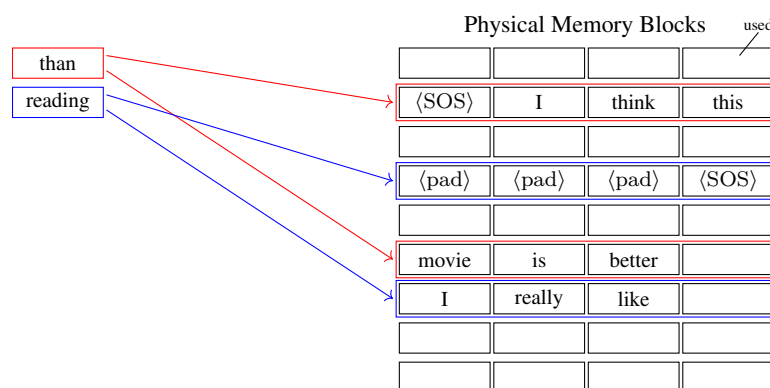
考虑图5.12中的简单内存分配示例，其中对由两个序列组成的批次执行自注意力。对于每个序列，我们需要根据自注意力的要求，将当前标记关注到该序列的KV缓存中的键值对。在



(a) Two sequences in a batch



(b) Memory allocation for KV caching in standard self-attention



(c) Memory allocation for KV caching in PagedAttention

图5.12: PagedAttention中内存分配的示意图。批处理中有两个序列，如图子图 (a) 所示。由于内存碎片化，KV缓存存储在标准自注意力中的一个大空闲内存块中（见图子图 (b)），但未使用碎片化内存。相比之下，在PagedAttention中（见图子图 (c)），KV缓存被分成更小的块，因此适合碎片化内存。

标准自注意力实现，KV 缓存存储在连续的内存块中，使我们能够高效地访问这个连续的内存。然而，在分页的 KV 缓存系统中，KV 缓存被划分为更小、固定大小的内存块，这些块不一定连续。这些较小的 KV 缓存块可以更有效地分配到碎片化的内存区域，从而提高内存利用率。将数据块分发的另一个好处是

不同内存块之间的KV缓存使得缓存过程可以并行化。例如，如果输入序列很长且内存带宽足够，那么在多个内存块上并行写入和读取序列不同段的关键字和值向量将是有益的。

通常，将连续数据存储在非连续区域可能会导致问题，例如，访问碎片化数据需要额外的寻址时间，这会降低I/O效率。然而，在处理大规模数据（例如，对极大矩阵进行乘法运算）时，我们通常不会一次性处理所有数据，而是将其划分为更小的块进行块级计算。从这个角度来看，对注意力计算进行分区也是合理的。如果分页策略设计得当，内存访问的额外开销可以最小化，而内存利用率的提高可以非常显著。

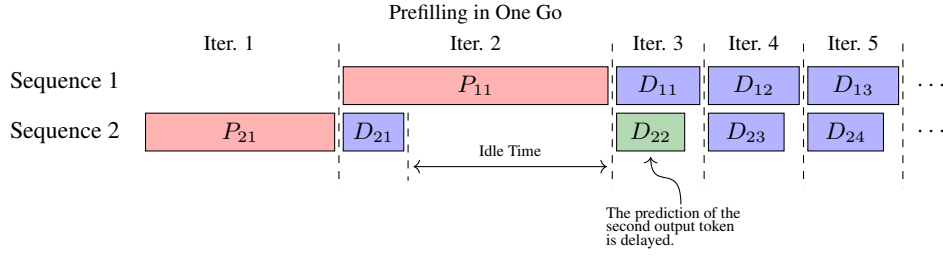
5.2.2.4 分块预填充

我们已经看到，在迭代级调度中，不同序列的预填充和解码可以同时发生。这可以被视为一种预填充优先策略，可以最大化吞吐量。然而，如果输入序列非常长且预填充过程主导了计算，那么这样的迭代可能需要很长时间。在这种情况下，其他序列的解码必须等待预填充完成，从而导致生成输出标记的延迟增加。因此，虽然预填充优先策略对于最大化硬件利用率是有效的，但它们可能会在标记生成延迟方面引入显著的可变性，尤其是在系统处理长序列和短序列混合的情况下。

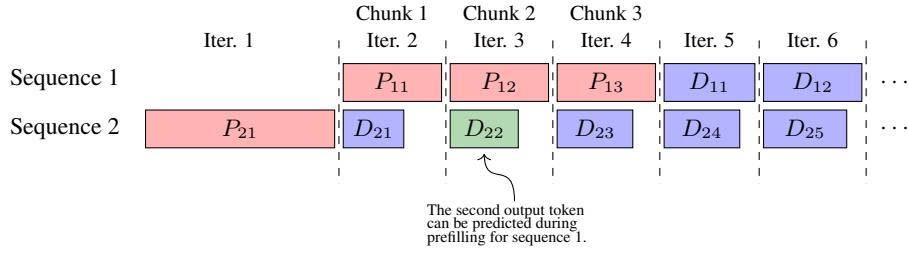
一种减少解码延迟的简单方法是将批处理中的不同序列的计算进行比较。一种方法是按块划分序列并逐块进行预填充。这种方法通常被称为块预填充，一次处理每个序列的较小部分，使系统能够更好地平衡序列间的计算负载[Agrawal等人, 2023]。通过选择合适的块大小，我们可以确保当两个序列的预填充和解码重叠时，它们在同一迭代中的处理时间往往相似。因此，解码空闲时间减少，整体吞吐量提高。

图5.13展示了在几次迭代中的分块预填充的示意图。在这个例子中，批次包含两个序列。第一个序列的整个预填充过程被分为三个预填充步骤，从而产生了标记为 P_{11} 、 P_{12} 和 P_{13} 的块。每个块对应一次迭代，因此可以与一个解码步骤重叠。这样，在第一个序列的预填充过程中，我们可以执行三个解码步骤，而不是像标准迭代级调度那样只执行一个解码步骤。因此，解码过程的空闲时间减少，并且可以更早地生成输出标记。

分块预填充通过重叠预填充和解码来提高解码效率，但代价是额外的内存开销和调度复杂性。在标准预填充中，我们一次性处理整个输入序列，一次性构建KV缓存。相比之下，在分块预填充中，每个块需要单独的前向传递来计算其注意力输出并更新KV缓存。因此，在处理后续块的同时，我们需要维护早期块的KV缓存。这也妥协了在单次遍历中完成整个序列预填充的并行性。在实践中，通常可以通过选择合适的块大小来平衡吞吐量和延迟。



(a) Simple Iteration-level Scheduling



(b) Chunked Prefilling

图5.13: 简单迭代式调度与分块预填充的比较。 P_{xy} 表示序列 x 的第 y 次预填充步骤, D_{xy} 表示序列 x 的第 y 次解码步骤。在简单迭代式调度(或预填充优先级调度)中, 由于预填充被视为单个迭代, D_{22} 必须等待序列1的预填充完成。在分块预填充中, 预填充过程可以分解为多个步骤。因此, D_{22} 可以在序列1的预填充期间执行(即, 在 P_{12} 期间)。

值得注意, 本小节中讨论的方法可以广泛归类为基于优先级的调度方法。在这些方法中, 我们可以对某些请求或某些预填充或解码步骤给予优先级, 以便系统资源以更好地与特定性能目标相匹配的方式分配。例如, 如上所述, 我们可能优先考虑解码以最小化标记生成延迟, 或者在批处理场景中优先考虑预填充以最大化整体吞吐量。实践者可以为现实应用中的特定需求和操作约束设计自定义优先级策略, 例如用户定义的请求截止日期和重要性级别。

5.2.3 并行化

并行化是一种广泛使用的扩展LLM推理的方法, 尤其是在大规模部署中。在第1章中, 我们讨论了几种常见的并行化策略, 用于并行化LLM预训练, 例如模型并行性、张量并行性和流水线并行性。我们还讨论了易于在分布式计算系统中部署的高效架构。例如, 在MoE模型中, 我们将不同的专家分配到不同的设备上³。只有针对给定输入的活跃专家被执行, 这显著提高了计算效率, 同时保持了模型质量。许多这些方法可以直接应用于LLM推理, 只需进行最小修改。

³In LLMs, the experts are typically modular FFNs. So each expert is a part of the FFN component in the Transformer architecture.

然而，将这些并行化技术应用于推理与预训练相比带来了新的挑战。这些问题在实时或低延迟推理场景中尤为突出，在这些场景中，设备间的负载不平衡和通信开销可能会显著影响性能。例如，与预训练不同，预训练中可以提前准备批次，而推理必须实时处理可变长度的序列。这使得维护最佳设备利用率变得更加困难，并复杂化了异构计算资源之间的调度。相关问题是负载均衡。当在短时间内收到大量请求时，系统必须高效地在可用设备之间分配工作负载。例如，现实世界的请求通常由于任务类型和提示长度的差异而表现出高度可变的计算需求。这种可变性使得简单的静态负载均衡方法无效，因此我们需要使用更细粒度的策略，这些策略可以适应运行时条件。当我们在异构硬件上部署系统并且存在严格的延迟约束时，问题变得更加复杂。

在LLM的开发中，并行化与LLM服务密切相关。通常，构建一个高质量的LLM服务系统并非易事——它通常需要结合多种技术，如架构设计、工作负载分配以及针对LLM的硬件/软件优化。因此，LLM服务构成一个极其广泛的主题，通常需要大量的工程专业知识。在此，我们不会深入探讨LLM服务的细节。对于相关概念和技术，读者可以参考相关开源系统（如vLLM⁴、TensorRT-LLM⁵和TGI⁶）以及论文[Pope等人，2023；Li等人，2024a]。

5.2.4 注释

我们已经在本章和前几章中考虑了许多提高LLM效率的方法。尽管这些方法解决不同的问题，但它们本质上都是在各种性能因素之间探索权衡。一个重要的权衡是在推理速度和准确性之间。例如，量化、剪枝和知识蒸馏等技术可以显著减少计算开销和延迟，但可能会在模型性能上引入轻微的退化。相反，保留完整精度或使用更大的模型可以提高准确性，但代价是更慢的推理和更高的资源需求。

另一个在LLM推理中重要的考虑因素是内存-计算权衡。正如在计算机系统设计，我们需要考虑生成输出所需的内存使用和计算之间的平衡。特别是，在推理期间存储中间结果，如KV缓存，可以显著减少冗余计算，但代价是增加了内存使用。在KV缓存中，存储过去的注意力状态可以避免对先前标记的自注意力重新计算，从而减少每个标记的计算时间。然而，随着标记数量的增加，KV缓存的内存占用也随之增加，尤其是在处理非常长的序列或并行处理多个序列时。为此，已经开发出各种技术，通过部分重新计算中间状态来减少内存消耗。例如，分块或窗口注意力将注意力范围限制在最近的标记子集，以减少KV缓存大小，但代价是减少了上下文或如果必须重新处理过去信息，则增加了额外的计算。

⁴<https://github.com/vllm-project/vllm>

⁵<https://github.com/NVIDIA/TensorRT-LLM>

⁶<https://github.com/huggingface/text-generation-inference>

请注意，考虑内存-计算权衡是一个非常普遍的原则。它可以扩展到注意力机制和Transformer之外，应用于系统设计的其他组件。一个例子是数据精度的选择。使用FP16或INT8等低精度格式可以减少内存使用和内存带宽需求，从而有效缓解对内存子系统的压力。然而，低精度可能会导致数值不稳定性或轻微的精度下降，需要仔细校准或重新训练。因此，这种权衡也可以被视为内存-计算-精度三角形，其中某一维度的改进可能以牺牲另一维度为代价。

除了速度、准确性和内存之外，还有其他几个维度也会影响LLM推理效率。其中一些维度在本章中已讨论，而另一些尚未讨论。以下是我们对这些维度的概述。

- 吞吐量与延迟：在大规模多用户LLM服务场景中，我们通常旨在最大化系统吞吐量。例如，如本节所述，我们可以将多个请求一起批量处理，以增加同时处理的令牌数量。然而，批量处理会增加等待时间，并可能导致每个请求的延迟更高，尤其是对于短或交互式请求。相比之下，为了降低延迟而进行优化通常需要单独或以更小的批量处理请求，这会导致硬件资源利用率不足并降低吞吐量。实现良好的平衡取决于服务质量要求和用户交互模式。
- 泛化与专业化：通用型LLM被训练以使用单一组参数执行广泛的任務。虽然灵活，但它们可能在特定任务上效率或准确性较低。专用模型可以为针对的应用程序提供更好的性能和更低的推理成本。然而，维护多个专用模型会增加系统复杂性和存储需求。在维护单个通用模型与多个专用模型之间进行权衡是重要的系统级设计选择。
- 能源效率与性能：高性能推理通常需要在强大的加速器上以高吞吐量运行大型模型，这会消耗相当多的能源。这可能会对边缘部署或对能源敏感的环境造成问题。像模型压缩这样的技术可以提高能源效率，但通常会在输出质量或延迟方面有所下降。因此，能源限制为优化LLM推理引入了另一个重要维度。

5.3 推理时缩放

缩放定律可以被认为是指导LLM发展的基本原理之一。在前几章中，我们多次讨论了扩大训练数据、模型大小和计算能力可以有效提高预训练性能。事实上，缩放定律也适用于下游阶段，如微调和推理（见图5.14）。在这里，我们考虑推理时间缩放，这是近年来许多LLM广泛采用来解决复杂

条件其预测以更多先验信息。一个例子是少样本提示。它通过多个输入输出示例增强上下文，因此模型可以从这些示例中隐式地学习任务行为，而无需参数更新。在少样本提示的基础上，我们可以使用思维链提示来鼓励模型在最终答案之前产生中间推理步骤。请注意，思维链提示是解决推理问题最重要的方法之一。通过明确提供问题解决的中间步骤，我们可以提示模型将复杂任务分解成更简单的子任务，这被发现对生成准确和可解释的输出非常有益。

超出通过示例或推理步骤扩展提示，另一种上下文扩展方法涉及动态融入外部知识。这通常通过RAG实现。RAG系统首先根据当前输入从大量文档或数据库中检索相关文档片段。然后，这些检索到的信息被添加到LLM提供的上下文中。这实际上扩展了上下文，包括及时或专业的外部知识。通过这样做，模型将响应建立在外部来源中找到的特定知识上。因此，LLM可以生成不仅与输入相关，而且事实准确且最新的响应。

然而，随着上下文增长，这些方法通常受到有限上下文窗口长度的限制。虽然模型架构和技术（如高效注意力模型）不断进化以支持更长的上下文，但处理极长输入仍然具有挑战性。增加的计算成本是一个因素。更重要的是，当上下文窗口变得非常大时，模型可能难以有效地关注最相关的信息（例如，“迷失在中间”现象）。因此，有效的上下文扩展不仅关乎添加更多信息，还关乎在模型的处理能力内战略性地选择、结构和呈现最相关的信息。

这里我们省略了这些方法的详细讨论，因为它们已经在之前的章节中有所涉及。请参阅第2章和第3章以获取更多详细信息，包括提示、RAG和长序列建模方法。

5.3.2 搜索缩放

在LLMs中，解码是一个搜索过程，旨在高效地找到给定输入序列的最佳输出序列。搜索缩放（或解码缩放）通常涉及两个方面：缩放输出长度和缩放搜索空间。

输出长度缩放是指增加推理过程中生成的标记数量。这在需要长文本生成的任务中尤为重要，例如故事写作。最近，在数学问题解决和代码生成中，生成具有长思考路径的输出表现出强大的性能。例如，鼓励模型在生成最终答案之前生成思考路径已被发现对执行复杂推理非常有益。这一想法已被广泛应用于开发最近的推理LLM，如OpenAI [2024] 的o1和Deepseek [2025] 的R1。我们将在第5.3.4节中进一步讨论输出长度缩放。

搜索空间的扩展，另一方面，指的是在搜索过程中考虑的候选输出序列集合的扩展，以便找到更高质量的输出。如第5.1.3节所述，一个简单的例子是在束搜索中，我们增加束宽以允许在每个解码步骤中并行探索更多的候选序列。这增加了找到更佳输出的机会

发现更好的输出，尤其是在最优解无法从局部决策中立即显现的任务中。

除了解码算法调整外，还可以探索紧凑结构来编码大量输出。例如，我们可以构建和导航推理步骤的树或图[Yao等人, 2024]。在这个范例中，每个节点代表一个部分解决方案或中间步骤，边代表推理状态之间的转换。这种结构化搜索使模型能够同时考虑多条路径。另一个相关方向是受蒙特卡洛树搜索启发的解码，其中模型基于学习到的启发式方法或外部奖励模型随机探索和评分不同的路径。

搜索缩放是一个非常通用的概念，它通常隐含在利用搜索结构、启发式和模型不确定性的搜索过程设计中。上述许多方法之前已经讨论过，尽管它们最初并不是以缩放为主要目标。然而，搜索缩放本质上伴随着计算成本。例如，增加搜索宽度直接导致更高的内存使用和更长的推理时间。在实践中，通常存在一个收益递减的点，进一步扩大搜索空间在显著的计算成本下只会带来输出质量的微小改进。因此，一个有效的策略通常涉及在缩放和计算可行性之间找到一个最佳平衡。

5.3.3 输出集成

如果我们有多个模型输出，通常将它们结合起来以减轻单个模型错误的影响并综合出更优越的最终输出是有益的。每个模型可能捕捉到底层数据分布的不同方面或具有独特的优势和劣势。通过集成，我们可以平均出单个预测中存在的噪声或随机错误，从而得到更稳定和可靠的输出。在LLM集成中，最简单的方法之一是平均每个模型从下一个标记的概率分布，并使用这个平均分布选择最佳标记。或者，如果我们把问题视为一个离散的决策任务，可以采用多数投票。更复杂的方法可能涉及根据单独的评分函数重新排序不同模型生成的候选输出，甚至可以使用元学习器智能地结合预测。

输出集成中的“缩放”以运行多个模型或采样多个输出为代价。这不仅增加了推理的延迟，还导致了管理多个模型的多余复杂性。但是，随着更多模型的添加，输出质量并不总是无限期地提高。在某些情况下，当集成中的组件模型数量超过某个阈值时，输出集成的益处可能会减少。相反，当个别模型多样化（即它们犯不同的错误）时，集成的益处通常更大，即使组件模型的数量相对较少。因此，通常的做法是使用一组多样化的LLMs，这些LLMs在训练数据、模型架构或微调目标上有所不同。

在LLMs中，“扩展”通常意味着为了质量而“增大”事物，并投入更多资源。然而，除了提高质量之外，扩展还可以意味着更多。它还可以表示提高鲁棒性（使系统更不容易出错且更可靠）和探索（覆盖更广泛的潜在解决方案）。在输出集成中，这些维度自然地整合在一起。例如，对不同的模型输出进行平均或投票的这种行为本身就是

直接策略以增强对单个模型失败的鲁棒性。此外，通过有意包含各种模型，集成增加了发现新颖或更优解决方案的机会。在这种情况下，扩展不仅限于使模型更大或运行更长——它还意味着使推理更鲁棒、更具探索性和适应性。

5.3.4 生成和验证思维路径

到目前为止，我们将推理时间缩放视为一种用于缩放推理各个方面（如序列长度、模型大小和/或搜索策略）的一般方法。事实上，一个成功的应用是使用推理时间缩放来增强LLMs的推理能力。正如我们所见，通过使用思维链方法可以提高LLMs的推理性能。因此，我们可以利用思维链提示来生成中间推理步骤并得出正确答案。然而，推理问题通常非常复杂，我们无法通过提供简单的思维链提示来获得高质量的解决方案。例如，在解决数学问题时，我们通常需要推理一系列步骤。在每一步中，我们需要计算出一些中间结果，验证它，然后确定下一步要做什么。推理路径不是一个固定的模式，而是一个动态生成的思维过程，通常涉及试错、回溯和自我纠正。这需要更复杂的提示策略或搜索算法来导航这种复杂的推理。在本小节中，我们重点关注超越简单思维链的推理缩放方法，以更有效地解决复杂的推理问题。

在较高层次上，扩展LLM推理的方法可以分为两类：

- 无监督方法。这些方法旨在在不要求修改或重新训练预训练参数的情况下提高推理能力。相反，它们专注于推理过程中的技术，例如复杂的提示策略（例如，思维链）和算法对推理过程的控制（例如，搜索）。
- 基于训练的方法。这些方法涉及进一步训练或微调模型参数，以显式提高推理能力，例如在包含推理示例的数据集上进行监督微调（例如，带有逐步解答的数学问题）。

以下，我们首先讨论无训练方法，然后讨论基于训练的方法。

5.3.4.1 带验证器的解决方案级搜索

给定一个输入序列（例如，一个数学问题），存在许多可能的输出序列（例如，问题的解决方案）。如果我们有一个模型来评估或验证每个解决方案，我们可以选择最好的一个。这是像best-of- N 采样这样的方法背后的基本原理，其中生成多个输出，并根据某种选择机制选择最佳结果。这样的选择过程可以被视为一个搜索问题，它涉及两个组件：

- 搜索算法。这定义了用于探索可能输出序列（解）空间并生成一组候选的策略。它可以从简单的独立采样到更复杂的搜索技术，如第5.1.3节所述。

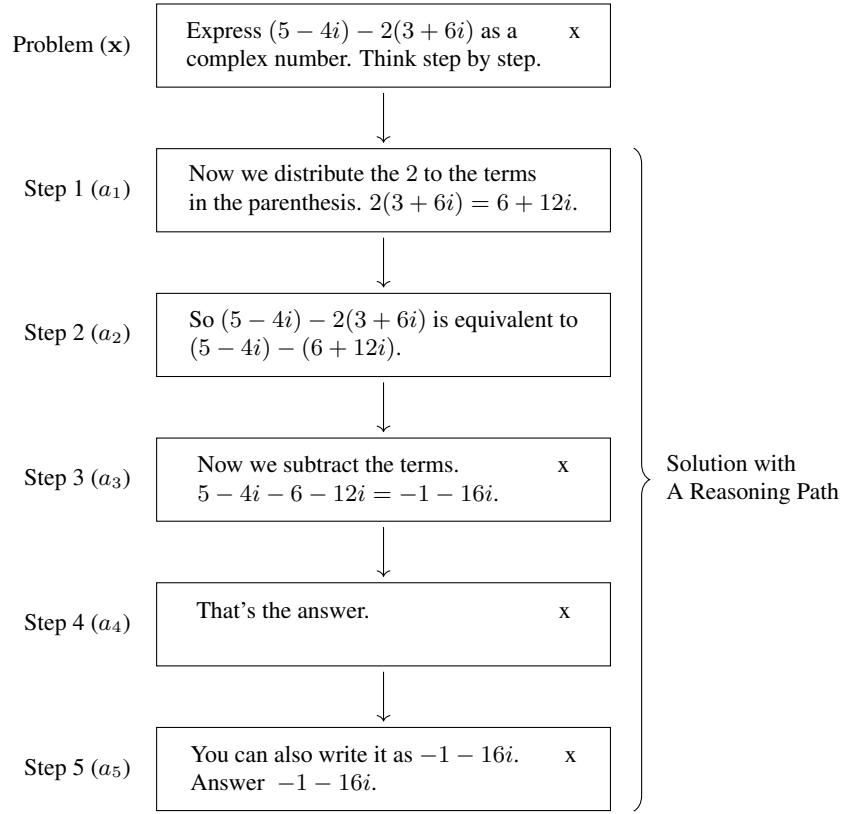


图5.15: 多步推理的说明。此示例来自PRM800K数据集[Lightman等人, 2024]。给定一个数学问题, LLM被提示生成一个由多个推理步骤组成的思维路径(或推理路径)。每个步骤基于前一步的结果解决一个子问题。原始问题的答案包含在最后一步。

- 验证器。这是一个负责评估搜索算法生成的每个候选解的质量、正确性或实用性的模型或函数。它提供分数、概率或判断, 使系统能够从候选者中选择最佳者。验证器可以是另一个LLM, 甚至是一组预定义的规则或启发式方法。

给定一个输入问题 x , 我们定义一个输出解 y 可以表示为一个推理步骤的序列:

$$\mathbf{y} = (a_1, a_2, \dots, a_{n_r}) \quad (5.37)$$

在 a_i 是第 i 个推理步骤, 而 a_{n_r} 是最后一个步骤, 其中应包含问题的答案。参见图 5.15 以了解多步骤推理路径的示例。

搜索算法可以高效地生成一组候选解

$$\mathcal{D}_c = \{\mathbf{y}_1, \dots, \mathbf{y}_K\} \quad (5.38)$$

然后, 我们可以使用一个验证器, 该验证器通过函数 $V(\mathbf{y})$ 对每个解决方案进行评估, 以对 \mathcal{D}_c 中的候选者进行评分。最终输出是验证器选出的最佳候选者

$$\hat{\mathbf{y}} = \arg \max_{\mathbf{y} \in \mathcal{D}_c} V(\mathbf{y}) \quad (5.39)$$

尽管验证整个推理路径是可能的，但一个更简单的替代方案是仅验证最终的推理步骤。这样，验证函数 $V(y)$ 就简化为仅依赖于包含在 a_{n_r} 中的最终答案。这可以通过各种方式实现，具体取决于问题的性质和预期的答案格式。

- 对于某些数学和编码问题，我们可以使用现成的工具作为验证器。例如包括数学定理的证明检查器、代码执行的解释器或编译器，以及用于验证程序正确性并针对预定义测试用例的单元测试系统。
- 如果有用于评估答案的有标签数据，例如人类偏好数据，我们可以在这些数据上训练一个奖励模型。然后，学到的奖励模型被用作验证器，为每个候选答案分配一个标量分数。
- 如果没有现有系统或合适的奖励模型，我们可以使用另一个LLM作为验证器。这个LLM被提示评估候选答案的质量。它可能是一个更强大的模型，或者使用特定“评估器”提示的相同LLM。
- 另一种方法是设计基于启发式的简单验证器。常用的方法是对一组候选答案进行多数投票，选择出现频率最高的答案。

基于这些验证器，我们可以搜索以获得一组候选解决方案进行选择。一种简单的策略，通常被称为并行缩放 [Brown 等人, 2024; Snell 等人, 2024]，涉及通过独立运行基本 LLM K 次来生成 K 个候选解决方案。在这个过程中，我们可以调整采样中的温度来控制输出的多样性。验证器随后评估这些 K 个完整解决方案中的每一个，得分最高的一个被选为最终输出。这在概念上与 best-of- N 采样非常相似，我们在前面的章节中主要将其描述为使用奖励模型从一组采样输出中选择最佳方案的方法。

另一种方法是顺序缩放，它通过增量构建一系列解决方案 [Gou 等人, 2024; 张等人, 2024]。它从由 LLM 通过提示生成的初始解决方案开始。然后，我们使用验证器（通常是同一个 LLM）来评估解决方案。这可以看作是一个批评阶段。这一阶段的输出是一些形式的反馈，如指明错误的文本批评或提出改进建议，反映解决方案质量的数值分数，甚至可以指导下一代的修订计划或中间步骤。然后，将这种反馈与原始问题和当前解决方案一起用于提示 LLM 生成可能改进的解决方案。这可以看作是一个细化阶段。这种批评-细化循环可以重复进行，形成一个迭代循环：

$$\mathbf{y}_{k+1} = \text{Refine}(\mathbf{x}, \mathbf{y}_k, \text{Feedback}(\mathbf{y}_k)) \quad (5.40)$$

其中 $\text{Feedback}(\mathbf{y}_k)$ 表示验证器的反馈。 $\text{Refine}(\cdot)$ 函数通过向 LLM 提示原始问题 \mathbf{x} 、前一个解 \mathbf{y}_k 和此反馈来生成改进的解 \mathbf{y}_{k+1} 。该过程可以迭代 K 次，或者直到解

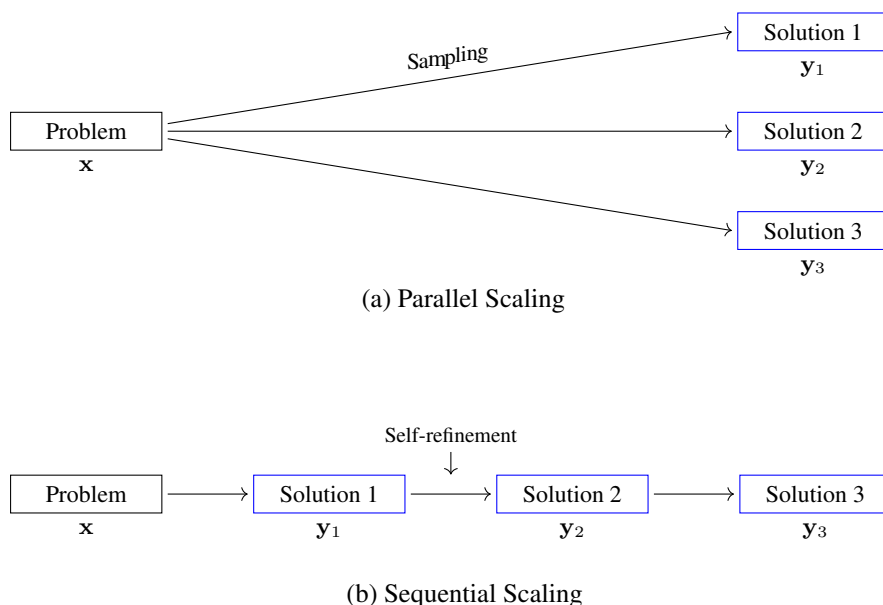


图5.16: 并行缩放和顺序缩放的示意图。在并行缩放中, 我们通过独立运行LLM多次来获得多个解决方案。在顺序缩放中, LLM生成一个初始解。然后, 我们使用LLM迭代地对其进行细化, 每次细化都产生一个新的、可能更好的解决方案。

质量, 如验证者评估, 收敛到一个令人满意的水平。这种迭代框架, 通过生成、评估（批评）和修订的循环逐步改进解决方案, 正是自我完善[Shinn等, 2023; Madaan等, 2024]的构成。在这种情况下, 验证者的角色不仅仅是从静态集合中挑选出最佳完整解决方案, 而是积极引导生成过程本身。

查看图5.16以了解并行缩放和顺序缩放的示例。请注意, 还有其他执行搜索和获得不同候选解集的方法。一种替代方法是按树结构组织搜索。这种方法通常被称为树搜索, 提供了一种更结构化的方式来探索可能的推理路径空间。在解决方案级搜索中, 树中的每个节点代表一个完整的解决方案。在搜索过程中, 我们需要将一个节点扩展到一组子节点, 代表可以在验证中考虑的新解决方案。扩展过程通常涉及取一个现有解决方案（父节点）并使用LLM生成变体或替代解决方案。

5.3.4.2 带验证器的步骤级搜索

当上述讨论的方法主要关注在最终选择之前生成完整解决方案时, 搜索过程也可以更深入地集成到推理路径的逐步生成中。这导致了一种执行带有验证器的步骤级搜索的方法, 其中指导或剪枝发生在中间推理步骤 $\{a_1, \dots, a_{n_k}\}$ 而不是仅在完整解决方案 y 形成之后。

这种细粒度控制对于复杂的推理问题特别有益, 在这些问题中, 一个错误的中间步骤可能会使整个后续推理链无效。通过在每个中间步骤进行评估或引导生成, LLM可以更有效地探索推理空间, 可能早期剪枝无望的路径或分配更多资源来探索更可能的路径。

步骤级搜索与验证器也可以建模为树搜索问题。在这个范例中，每个节点（或状态）对应一个部分推理路径， $a_{\leq i}$ ，表示迄今为止采取的 i 推理步骤的序列（即从根节点到当前节点的路径）。搜索过程的目标是从一个初始空路径开始，探索潜在的状态空间，以找到一个构成正确解决方案的完整路径。请注意，我们在这里使用 $a_{\leq i}$ 来表示部分推理路径，而不是 $y_{\leq i}$ 。虽然这使符号与表示完整解决方案（ y ）或解决方案级搜索中的完整路径的符号不一致，但它有助于突出关注单个动作或步骤。

核心组件是具有验证器的步骤级搜索：

- 节点表示。节点是一个部分推理路径 $a_{\leq i} = (a_1, \dots, a_i)$ 。根节点是空路径，终端节点是完整的推理路径。
- 节点扩展。给定当前部分路径 $a_{\leq i}$ ，使用LLM生成一个或多个候选下一步推理步骤 $\{a_{i+1}^{(1)}, \dots, a_{i+1}^{(M)}\}$ 。每个候选步骤，当附加到 $a_{\leq i}$ 后，形成一个新的潜在部分路径 $a_{\leq i+1} = (a_1, \dots, a_i, a_{i+1}^{(j)})$ 。
- 验证。验证器 $V(\cdot)$ 在当前部分路径 $a_{\leq i} = (a_1, \dots, a_i)$ 和原始问题 x 的上下文中评估新生成步骤的质量。与解决方案级别的验证一样，步骤级别的验证器可能会输出一个数值分数、一个分类标签和文本反馈。
- 搜索。这决定了搜索空间如何被探索。基于验证器的评估，搜索策略决定哪些部分路径需要进一步扩展，哪些需要剪枝，以及探索的顺序。

此步骤验证允许对推理过程进行动态调整。如果步骤 a_{i+1} 被认为是不正确或无希望的，则搜索算法可以回溯并从 $a_{\leq i}$ 或甚至从更早的节点 $a_{\leq i'}$ (where $i' < i$) 探索替代步骤。相反，如果步骤被高度评价，资源可以集中在扩展该路径上。参见图 5.17 以了解带有验证器的步骤级搜索的说明。

显然，这个搜索框架与第5.1.3节中讨论的用于解码LLM方法的搜索框架非常相似。例如，束搜索在每个生成步骤中维护一组最有希望的 K 部分序列。这是一种步骤级搜索，其中“验证器”隐式地是LLM自己的概率模型，“搜索”是修剪机制以维持束大小。

然而，如这里所述的具有显式验证器的步骤级搜索与标准解码存在差异。其中之一是验证器可以是一个比生成型LLM的原始输出概率更为复杂的组件。步骤级验证器的设计基本上遵循解决方案级验证的设计。一个步骤级验证器可能是一个语言模型，它评估在先前的路径上下文单个推理步骤的质量。这个LLM甚至可以被微调以增强其验证能力。或者，对于具有明确定义规则的领域，它可能是一个符号引擎或一系列程序性检查。此外，验证器可以设计为根据当前部分路径预测未来的效用或成功的可能性，从强化学习中的价值函数中汲取灵感。人类

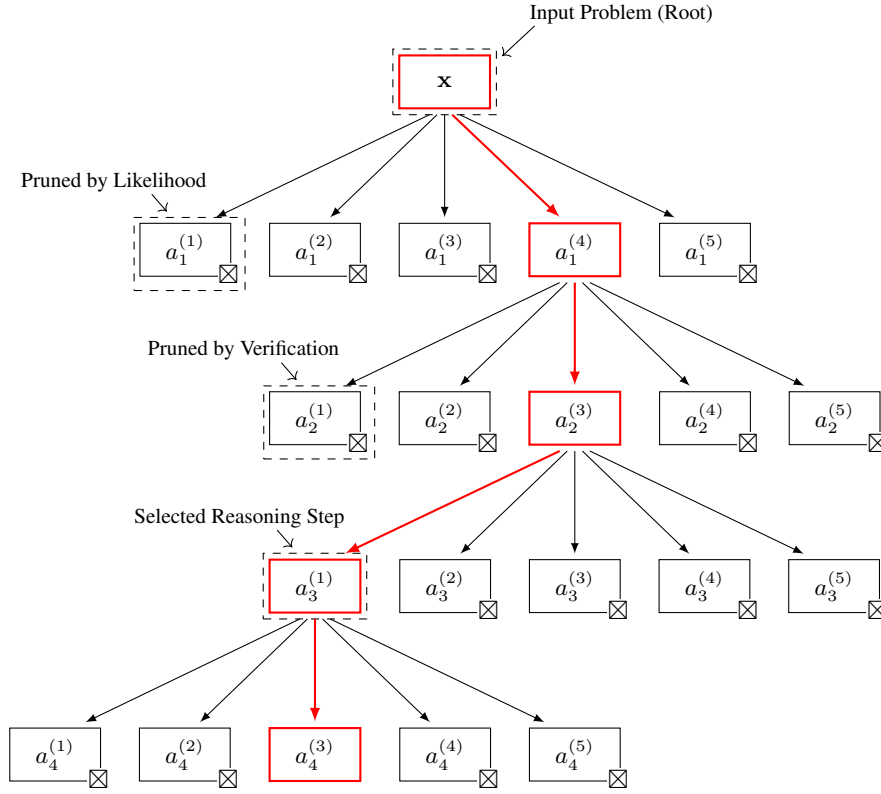


图5.17: 带有验证器的步级搜索示意图。 $a_i^{(j)}$ 是第 j 个推理步骤的第 i 个候选者, \boxtimes 是 LLM 输出概率剪枝的候选者, \boxtimes 是验证器剪枝的候选者。以输入问题作为根节点, 我们在每次扩展中生成多个推理步骤来扩展树。每个候选者可以通过似然度 (如在标准解码中) 或步级验证进行剪枝。然后, 未剪枝的候选者被扩展以生成进一步的推理步骤。该过程重复进行, 直到生成一个完整的推理链, 达到最终答案, 或者达到预定义的搜索限制。

专业知识也可以融入以对关键步骤进行判断, 尤其是在高风险场景中。

一个此类步骤级验证器的例子, 尤其是在使用人类反馈评估中间进度时, 是过程奖励模型 (PRM)。PRM 通常是一个独立的语言模型, 用于为部分路径 $a_{\leq i}$ 中的每个推理步骤 a_i 输出一个标量奖励。与仅评估最终解决方案的输出奖励模型 (ORMs) 相比, 它提供了更直接和更细粒度的监督信号。然而, PRMs 的发展依赖于步骤级的人类注释, 例如对不同下一步的偏好。为每个中间步骤收集监督需要更多的劳动强度, 并且比简单地标记最终结果需要人类注释者付出更多的认知努力。

一种开发步骤级验证训练数据的替代方法是使用 LLMs 自动生成此类注释。例如, 我们可以选择一个强大的 LLM, 称为教师模型, 并提示它首先为给定问题生成一个完整的推理路径。然后, 在这个路径中的每个中间步骤, 我们可以提示相同的教师 LLM (或另一个有能力的 LLM) 生成除了它最初选择的步骤之外的其他几个备选下一步。然后, 可以再次提示教师 LLM 评估这些备选方案。这些评估结果 (例如, 正确与错误) 可以随后作为数据注释。或者, 可以利用 PRMs 的泛化能力。我们可以在步骤级验证更容易的任务上训练一个 PRM, 然后将这个 PRM 泛化到其他任务, 这些任务需要很少或没有额外的

训练。

请注意，步骤级验证也伴随着它自己的问题。频繁的验证，尤其是如果使用LLM作为验证器，会显著增加计算成本和延迟。设计有效的步骤级验证器本身就是一个非平凡的任务。一个不准确的验证器可能会过早地丢弃好的推理路径或未能识别出有缺陷的路径，从而误导搜索。这使得此类系统的开发更加复杂和困难。

5.3.4.3 鼓励长期思考

到目前为止，在本小节中，大多数方法都是隐式基于一个简单想法：生成更长的推理路径可以帮助。除了CoT和带有验证的搜索之外，我们还可以考虑其他方法来实现这一点。例如，我们可以通过明确要求扩展思考来提示LLM。除了直接提示之外，我们还可以对解码过程本身进行修改，例如调整标记限制或对短输出应用惩罚。另一种方法是采用多阶段生成方案，其中模型逐步构建其推理。

5.3.4.4 基于训练的缩放

除了考虑无训练的推理时间缩放方法外，我们还希望考虑通过进一步训练修改其参数以改进LLM内在推理能力的方法。虽然此类基于训练的缩放方法通常需要额外的训练成本和计算资源，但它们可以直接将更强的推理技能植入模型参数中，从而带来更有效和高效的推理性能。我们甚至可以将它们与无训练方法结合起来，以获得更好的推理时间缩放结果。

尽管我们在这里的讨论仅限于推理问题，但基于训练的缩放方法很常见。其中大部分已在第四章中讨论过。在这里，我们将简要描述这些方法如何应用于提高推理问题的推理时间缩放。

- 推理数据微调。增强推理最直接的方法之一是通过在专门为推理任务定制的数据集上微调预训练的LLM。这些数据集可以从简单的输入-输出对到更结构化的格式，包括逐步推理过程。典型例子包括数学文字问题数据集、逻辑推理练习或带有解释的代码生成。通过在这样数据上训练，模型学习常见的推理模式，因此在测试时可以生成详细且连贯的推理路径。
- 强化学习推理。如果我们把验证器视为奖励模型，那么我们可以看到前一小节中讨论的方法是将奖励模型直接应用于推理问题，尽管它们是无训练的。当然，我们可以将此奖励模型应用于LLM微调。这遵循了强化学习的标准范式。给定一个奖励模型，LLM作为策略，使用强化学习算法进行微调。LLM生成推理步骤或完整解决方案，从奖励模型接收反馈（奖励），并更新其参数以产生输出

这些奖励的最大化。此过程使LLM的输出与高质量推理的概念相一致，从而鼓励LLM生成更可靠的推理路径。另一个关键问题是奖励模型的训练。通常，这个奖励模型可以是评估最终答案的正确性或质量的成果奖励模型，或者是一个评估每个中间推理步骤质量的进程奖励模型，正如在步骤级验证的上下文中所讨论的。在某些情况下，我们甚至可以根据简单的规则开发奖励模型，例如给予更长的输出奖金。

- 知识蒸馏推理。在此方法中，训练一个更小、更高效的学生LLM来模仿更大、更强的教师LLM的推理输出或内部表示。教师模型可能为各种问题生成详细的推理步骤。然后，学生模型学习重现这些高质量的推理演示。这种策略通过在较小的模型中部署它们，使得更强的推理能力在推理时更易于访问，这些模型在推理时的计算成本更低。
- 迭代优化。基于训练的扩展也可以涉及迭代优化。例如，一个LLM可以生成一系列问题的解决方案。然后，这些解决方案及其推理路径被验证，无论是通过人类还是自动验证器。随后，正确的推理路径被添加到训练数据中，LLM在此基础上进一步微调。这形成了一个循环，其中LLM通过重复的生成、批评和学习，逐步提高其推理能力。

这些基于训练的缩放方法的主要优势是它们赋予了LLM更强的固有推理能力。这直接以多种方式促进了推理时间的改进：它可以导致更高效的推理，因为LLM可能需要更少的搜索或更少的生成样本来找到正确的解决方案。此外，生成的步骤或解决方案的质量更高。因此，一个经过良好训练的LLM可能比仅依赖上下文学习或无训练推理方案的LLM更有效地将其学习到的推理能力推广到新问题。

另一方面，与无训练方法相比，基于训练的方法也面临着挑战。为推理创建高质量、大规模的训练数据集可能成本高昂且劳动密集。特别是对于最大的LLM或使用RL时，微调过程本身可能计算密集，需要大量的工程努力。还存在模型过度拟合到训练数据中存在的特定类型问题或推理风格的风险，这可能会限制其在分布外任务上的性能。

5.4 摘要

在这一章中，我们讨论了LLM的推理问题。我们提出了LLM推理的预填充-解码框架和相关解码算法。然后，我们描述了几种高效推理的技术。我们还讨论了推理时间缩放，这被认为是提高LLM推理能力最重要的方法之一。

推理在序列数据上一直是人工智能领域的一个关注点 [Wozengraft and Reiffen, 1961; Viterbi, 1967; Forney, 1972]。在自然语言处理（NLP）的背景下，这一领域的研究可以追溯到语音识别和统计机器翻译的早期阶段 [Koehn, 2010]，当时的研究人员

面对高效搜索广阔假设空间以找到最可能的输出序列的挑战。随后开发了如束搜索和各种剪枝策略等技术，以使这一计算变得可行。当时，模型相对较弱，大部分研究集中在开发强大的搜索算法以减少搜索错误。这些基础思想继续影响着现代方法。

随着我们进入深度学习方法主导的时代，基于深度神经网络的模型变得极其强大。即使是非常简单的搜索算法，这些模型也能取得优异的结果。在这种情况下，推理不再像以前那样“重要”，研究关注点逐渐转向模型架构、训练方法和模型的扩展。

然而，历史往往重演。随着LLMs的兴起，推理再次引起了广泛关注。这种新的关注主要表现在两个方面：

- LLMs的推理成本非常高。例如，在高并发、低延迟场景中高效部署LLMs仍然是一个具有挑战性的问题，这使得推理效率至关重要。在这种情况下，高效的架构设计、优化的搜索算法以及各种推理优化策略具有重大的实际意义。
- 输入和输出序列长度在复杂任务中显著增加。特别是在数学推理等任务中，序列长度的增长进一步突出了推理效率的重要性。此外，最近证明，扩展推理过程是一种有效提高模型推理能力的方法。因此，实现高效的推理扩展正成为一个特别有前景的研究方向。

推理现在是一个广泛的主题，涵盖了众多技术。它不仅涉及模型架构和解码算法的开发，而且越来越多地受到部署LLMs所需复杂工程和高级系统级优化的影响。许多这些技术超出了NLP或特定AI领域的范围。相反，LLM推理优化的前沿现在深入到传统上被认为是核心计算机科学和工程领域的领域。这种系统性的视角为推理问题的研究带来了许多新想法。不幸的是，本章无法涵盖所有相关技术——实际上，这本身就是一个几乎不可能完成的任务。最终，更好地理解 and 掌握这些技术的最佳方式可能仍然在于动手实践。

Bibliography

- [Agrawal等人, 2023] Amey Agrawal, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav S Gula-vani, 和 Ramachandran Ramjee. Sarathi: 通过利用分块预填充的解码来高效进行llm推理。 *arXiv preprint arXiv:2308.16369*, 2023。 [Agrawal等人, 2024] Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav Gulavani, Alexey Tumanov, 和 Ramachandran Ramjee. 通过Sarathi-Serve}驯服{Throughput-Latency}权衡在{LLM}推理中。在 *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, 第117-134页, 2024。 [Ainslie等人, 2020] Joshua Ainslie, Santiago Ontanon, Chris Alberti, Vaclav Cvicek, Zachary Fisher, Philip Pham, Anirudh Ravula, Sumit Sanghai, Qifan Wang, 和 Li Yang. Etc: 在transformers中编码长和结构化输入。在 *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 第268-284页, 2020。
- [Ainslie等人, 2023] Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebron和Sumit Sanghai. Gqa: 从多头检查点训练广义多查询Transformer模型。在 *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 第4895-4901页, 2023。
- [Akyürek等人, 2023] Ekin Akyürek, Dale Schuurmans, Jacob Andreas, Tengyu Ma和Denny Zhou. 什么是上下文学习算法? 使用线性模型的调查。在 *Proceedings of The Eleventh International Conference on Learning Representations*, 2023。
- [Alabdulmohsin等人, 2022] Ibrahim M Alabdulmohsin, Behnam Neyshabur和Xiaohua Zhai. 重新审视语言和视觉中的神经缩放定律。 *Advances in Neural Information Processing Systems*, 35: 22300–22312, 2022。
- [Allal 等人, 2024] Loubna Ben Allal, Anton Lozhkov, 和 Daniel van Strien. cosmopedia: 如何创建大规模合成数据以进行预训练。 <https://huggingface.co/blog/cosmopedia>, 2024。
- [Almazrouei 等人, 2023] Ebtesam Almazrouei, Hamza Alobeidli, Abdulaziz Alshamsi, Alessandro Cap- pelli, Ruxandra Cojocaru, M rouane Debbah,  tienne Goffinet, Daniel Hesslow, Julien Launay, Quentin Malartic, Daniele Mazzotta, Badreddine Noune, Baptiste Pannier, 和 Guilherme Penedo。 开放语言模型的猎鹰系列。 *arXiv preprint arXiv:2311.16867*, 2023。
- [Andreas 等人, 2016] Jacob Andreas, Marcus Rohrbach, Trevor Darrell 和 Dan Klein. 神经模块网络。在 *Proceedings of the IEEE conference on computer vision and pattern recognition*, 第39–48页, 2016。
- [Arjovsky等人, 2016] Martin Arjovsky, Amar Shah和Yoshua Bengio. 单位演化循环神经网络。在 *International conference on machine learning*, 第1120–1128页, 2016。
- [Aschenbrenner, 2024] Leopold Aschenbrenner. 情境意识: 未来十年, 2024。 URL <https://situational-awareness.ai/>。
- [Askill 等人, 2021] Amanda Askill, 白云涛, 陈安娜, 德雷恩, Deep Ganguli, 汤姆·亨尼汉, 安迪·琼斯, 尼古拉斯·约瑟夫, 本杰明·曼, Nova DasSarma, 内森·埃尔哈格, 扎克·哈特菲尔德·多兹, 丹尼·赫南德斯, 杰克逊·科尔尼奥, 卡马尔·恩杜西, 凯瑟琳·奥尔松, 达里奥·阿莫迪, 汤姆·B·布朗, 杰克·克拉克, 山姆·麦克安德利斯, 克里斯·奥拉, 和贾里德·卡普兰。 通用语言助手作为对齐的实验室。 *arXiv preprint arXiv:2112.00861*, 2021。
- [Bach等人, 2022] Stephen H. Bach, Victor Sanh, 郑欣勇, Albert Webson, Colin Raffel, Nihal V. Nayak, Abheesht Sharma, Taewoon Kim, M. Saiful Bari, Thibault F vry, Zaid Alyafeai, Manan Dey, Andrea Santilli, Sun Zhiqing, Srulik Ben-David, 徐灿文, Gunjan Chhablani, 王汉, Jason Alan Fries, Maged Saeed AlShaibani, Shanya Sharma, Urmish Thakker, Khalid Almubarak, 唐翔宇, Dragomir R. Radev, Jiang Tian-Jian Mike, Alexander M. Rush. Promptsources: 自然语言提示的集成开发环境和存储库。在 *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, 第93-104页, 2022年。

[Bengio等人, 2003] Yoshua Bengio, Réjean Ducharme, Pascal Vincent和Christian Jauvin. 一个神经概率语言模型。 *Journal of Machine Learning Research*, 3:1137–1155, 2003。

[Bengio等人, 2006] Yoshua Bengio, Pascal Lamblin, Dan Popovici和Hugo Larochelle. 深度网络的贪婪层训练。 *Advances in neural information processing systems*, 19, 2006。

[Bengio等人, 2024] Yoshua Bengio, Geoffrey Hinton, Andrew Yao, Dawn Song, Pieter Abbeel, Trevor Darrell, Yuval Noah Harari, Ya-Qin Zhang, Lan Xue, Shai Shalev-Shwartz, Gillian K. Hadfield, Jeff Clune, Tegan Maharaj, Frank Hutter, Atilim Gunes Baydin, Sheila A. McIlraith, Qiqi Gao, Ashwin Acharya, David Krueger, Anca Dragan, Philip Torr, Stuart Russell, Daniel Kahneman, Jan Markus Brauner, 和Sören Mindermann. 在快速进步中管理极端AI风险。 *Science*, 384(6698): 842–845, 2024。

[Bentivogli和Giampiccolo, 2011] Luisa Bentivogli和Danilo Giampiccolo. Pascal识别文本蕴涵挑战 (rte-7) 在tac 2011。 <https://tac.nist.gov/2011/RTE/>, 2011。

[Besta 等人, 2024] Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, 和Torsten Hoefler. 思维图：用大型语言模型解决复杂问题。在 *Proceedings of the AAAI Conference on Artificial Intelligence*, 第 38 卷, 第 17682–17690 页, 2024 年。

[Biderman等人, 2021] Stella Biderman, Sid Black, Charles Foster, Leo Gao, Eric Hallahan, Horace He, Ben Wang, 以及Phil Wang. 旋转嵌入：一场相对革命。 <https://blog.eleuther.ai/rotary-embeddings/>, 2021。

[主教, 2006] 克里斯托弗·M·主教。 *Pattern Recognition and Machine Learning*。斯普林格, 2006。

[Blum 和 Mitchell, 1998] Avrim Blum 和 Tom Mitchell. 结合有标签和无标签数据与协同训练。在 *Proceedings of the eleventh annual conference on Computational learning theory*, 第 92-100 页, 1998。

[布拉德利和特里, 1952] 拉尔夫·艾伦·布拉德利和米尔顿·E·特里。不完全区组设计的秩分析：I. 配对比较法。 *Biometrika*, 39(3/4): 324–345, 1952。

[Brandon等人, 2024] William Brandon, Mayank Mishra, Aniruddha Nrusingha, Rameswar Panda和Jonathan Ragan Kelly. 使用跨层注意力减少transformer键值缓存大小。 *arXiv preprint arXiv:2405.12981*, 2024。

[Brill, 1992] Eric Brill. 一个简单的基于规则的词性标注器。在 *Speech and Natural Language: Proceedings of a Workshop Held at Harriman, New York, February 23-26, 1992*, 1992。

[Briski, 2025] Kari Briski. 如何通过规模定律推动更智能、更强大的AI, 2025。

[布朗等人, 2024] 布拉德利·布朗, 乔丹·尤拉夫斯基, 瑞安·厄利希, 罗纳德·克拉克, 吴克·V·莱, 克里斯托弗·雷, 以及阿齐亚·米尔霍塞尼。大型语言猴子：通过重复采样扩展推理计算。 *arXiv preprint arXiv:2407.21787*, 2024。

[布朗等人, 1993] Peter F. Brown, Stephen A. Della Pietra, Vincent J. Della Pietra, 和Robert L. Mercer. 统计机器翻译的数学：参数估计。 *Computational Linguistics*, 19(2): 263–311, 1993。

[布朗等人, 2020] 托姆·布朗, 本杰明·曼, 尼克·赖特, 梅拉妮·苏比亚, 贾里德·D·卡普兰, 普拉夫拉·达里瓦尔, 阿夫林德·尼尔拉坎坦, 普拉纳夫·希亚姆, 吉里什·萨斯特里, 阿曼达·阿斯克, 桑迪尼·阿加瓦尔, 阿里埃尔·赫伯特·沃斯, 格雷琴·克鲁格, 汤姆·亨尼汉, 雷文·查尔德, 阿迪蒂亚·拉梅什, 丹尼尔·齐格勒, 杰弗里·吴, 克莱门斯·温特, 克里斯·赫斯, 马克·陈, 埃里克·西格勒, 马泰乌什·利特温, 斯科特·格雷, 本杰明·切斯, 杰克·克拉克, 克里斯托弗·伯纳, 山姆·麦克安德利斯, 亚历克·拉德福德, 伊利亚·苏茨克维, 达里奥·阿莫迪。语言模型是少样本学习者。 *Advances in neural information processing systems*, 33:1877–1901, 2020。

[Bubeck等人, 2023] Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott M. Lundberg, Harsha Nori, Hamid Palangi, Marco Túlio Ribeiro, 和 Yi Zhang. 人工智能通用智能的火花：早期实验

与 gpt-4. *arXiv preprint arXiv:2303.12712*, 2023。

[Bulatov等人, 2022] Aydar Bulatov, Yury Kuratov, 和Mikhail Burtsev。循环记忆转换器。*Advances in Neural Information Processing Systems*, 35:11079–11091, 2022。

[Burges等人, 2005] Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton和Greg Hullender。使用梯度下降进行排序学习。在*Proceedings of the 22nd international conference on Machine learning*, 第89-96页, 2005年。

[Burns等人, 2023] Collin Burns, Pavel Izmailov, Jan Hendrik Kirchner, Bowen Baker, Leo Gao, Leopold Aschenbrenner, Yining Chen, Adrien Ecoffet, Manas Joglekar, Jan Leike, Ilya Sutskever和Jeff Wu。从弱到强泛化：利用弱监督激发强大能力。*arXiv preprint arXiv:2312.09390*, 2023a。

[Burns等人, 2023] Collin Burns, Jan Leike, Leopold Aschenbrenner, Jeffrey Wu, Pavel Izmailov, Leo Gao, Bowen Baker, 以及Jan Hendrik Kirchner。从弱到强的泛化, 2023b。URL <https://openai.com/index/weak-to-strong-generalization>。

[Caballero等人, 2023] Ethan Caballero, Kshitij Gupta, Irina Rish和David Krueger。破坏性神经缩放定律。在*ICLR 2023 Workshop on Mathematical and Empirical Understanding of Foundation Models*, 2023。

[曹等, 2007] 曹哲, 秦涛, 刘铁岩, 蔡明锋, 李航。从成对方法到列表方法的学习排序。在*Proceedings of the 24th international conference on Machine learning*, 第 129–136 页, 2007。

[Chang等人, 2024] Kaiyan Chang, Songcheng Xu, Chenglong Wang, Yingfeng Luo, Tong Xiao, 和Jingbo Zhu。大型语言模型的高效提示方法：综述。*arXiv preprint arXiv:2404.01077*, 2024。

[查尼雅克, 1997] 尤金·查尼雅克。基于上下文无关语法和词频统计的统计解析。AAAI/IAAI, 2005(598-603):18, 1997。

[陈等, 2023] 陈邦豪, 张兆丰, 尼古拉·朗格内, 朱胜欣。在大语言模型中释放提示工程潜力：全面综述。*arXiv preprint arXiv:2310.14735*, 2023a。

[陈等, 2023] 李长陈, 李诗洋, 严军, 王海, 卡帕·古纳拉特纳, 维卡斯·亚达夫, 唐正, 维贾伊·斯里尼瓦桑, 周天意, 黄恒, 金红霞。Alpagasus：用更少的数据训练更好的羊驼。*arXiv preprint arXiv:2307.08701*, 2023b。

[陈等, 2024] 李长陈, 李诗洋, 严军, 王海, 卡帕·古纳拉特纳, 维卡斯·亚达夫, 唐正, 维贾伊·斯里尼瓦桑, 周天意, 黄恒, 金红霞。Alpagasus：用更少的数据训练更好的羊驼。在*The Twelfth International Conference on Learning Representations*, 2024a。

[陈等, 2023] 陈守源, 黄仕敏, 陈亮健, 田远东。通过位置插值扩展大型语言模型的上下文窗口。*arXiv preprint arXiv:2306.15595*, 2023c。

[陈等, 2020] 陈天龙, 乔纳森·弗兰克尔, 常诗雨, 刘思佳, 张洋, 王张阳, 迈克尔·卡宾。预训练BERT网络的彩票假设。*Advances in neural information processing systems*, 33:15834–15846, 2020。

[陈等, 2024] 陈子翔, 邓毅和, 袁辉灼, 季开轩, 顾全全。自我博弈微调将弱语言模型转换为强语言模型。*arXiv preprint arXiv:2401.01335*, 2024b。

[Chevalier等人, 2023] Alexis Chevalier, Alexander Wettig, Anirudh Ajith和Danqi Chen。将语言模型适应于压缩上下文。在*Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 第3829-3846页, 2023。

[Chi等人, 2022] Ta-Chung Chi, Ting-Han Fan, Peter J Ramadge和Alexander Rudnicky。Kerple：

内核化相对位置嵌入用于长度外推。 *Advances in Neural Information Processing Systems*, 35:838 6–8399, 2022. [Chi et al., 2023] Ta-Chung Chi, Ting-Han Fan, Alexander Rudnicky和Peter Ramadge. 通过感受野分析视角剖析Transformer长度外推。在 *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 第13522–13537页, 2023。

[Chiang等人, 2023] Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, 和Eric P. Xing. Vicuna: 一个让gpt-4印象深刻的开源聊天机器人, 90%*的chatgpt质量, 2023年3月。URL <https://lmsys.org/blog/2023-03-30-vicuna/>。

[Chowdhery等人, 2022] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Rein Er Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayanan Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, 和Noah Fiedel. Palm: 通过路径扩展语言模型。 *arXiv preprint arXiv:2204.02311*, 2022。

[克里斯蒂亚诺等人, 2017] 保罗·F·克里斯蒂亚诺, 简·莱克, 汤姆·布朗, 米尔詹·马蒂奇, 肖恩·莱格, 达里奥·阿莫迪。从人类偏好中进行深度强化学习。 *Advances in neural information processing systems*, 第30卷, 2017年。

[Chu等人, 2023] 郑楚, 陈景昌, 陈强龙, 余伟江, 何涛, 王昊天, 彭伟华, 刘明, 秦冰, 刘婷。思维链推理综述: 进展、前沿与未来。 *arXiv preprint arXiv:2309.15402*, 2023。

[Chung等人, 2022] Hyung Won Chung, Le Hou, S. Longpre, Barret Zoph, Yi Tay, William Fedus, Eric Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Dasha Valter, Sharan Narang, Gaurav Mishra, Adams Wei Yu, Vincent Zhao, Yanping Huang, Andrew M. Dai, Hongkun Yu, Slav Petrov, Ed Huaihsin Chi, Jeff Dean, Jacob Devlin, Adam Roberts, Denny Zhou, Quoc V. Le, 以及Jason Wei. 扩展指令微调语言模型。 *arXiv preprint arXiv:2210.11416*, 2022。

[Clark等人, 2019] Kevin Clark, Minh-Thang Luong, Quoc V Le和Christopher D Manning. Electra: 将文本编码器作为判别器而非生成器进行预训练。在 *Proceedings of International Conference on Learning Representations*, 2019。

[Cobbe等人, 2021] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Heitsch, 以及John Schulman. 训练验证者解决数学文字问题。 *arXiv preprint arXiv:2110.14168*, 2021。

[Conneau等人, 2020] Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Édouard Grave, Myle Ott, Luke Zettlemoyer, 以及Veselin Stoyanov. 大规模无监督跨语言表示学习。在 *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 第8440–8451页, 2020。

[Coste等人, 2024] Thomas Coste, Usman Anwar, Robert Kirk和David Krueger. 奖励模型集成有助于减轻过度优化。在 *The Twelfth International Conference on Learning Representations*, 2024。

[Cui等人, 2024] Ganqu Cui, Lifan Yuan, Ning Ding, Guanming Yao, Bingxiang He, Wei Zhu, Yuan Ni, Guotong Xie, Ruobing Xie, Yankai Lin, Zhiyuan Liu, 和Maosong Sun. ULTRAFEEDBACK:

增强语言模型与缩放AI反馈。在 *Proceedings of the 41st International Conference on Machine Learning*, 第235卷, 第9722-9744页, 2024年。

[戴等, 2023] 戴迈, 孙宇涛, 董力, 郝亚如, 马树明, 隋志芳, 魏福如. 为什么GPT能在上下文中学习? 语言模型秘密地作为元优化器执行梯度下降。在

Findings of the Association for Computational Linguistics: ACL 2023, 第4005–4019页, 2023。

[Dai等人, 2019] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime G Carbonell, Quoc Le, 和 Ruslan Salakhutdinov. Transformer-xl: 超越固定长度上下文的注意力语言模型。在 *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 第2978-2988页, 2019。

[Dao等人, 2022] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra和Christopher Ré. Flashattention: 具有I/O感知的快速且内存高效的精确注意力机制。 *Advances in Neural Information Processing Systems*, 35:16344–16359, 2022。

[Deepseek, 2025] Deepseek. Deepseek-r1: 通过强化学习激励llms的推理能力。 *arXiv preprint arXiv:2501.12948*, 2025。

[Dehghani等人, 2018] Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit和 Łukasz Kaiser. 通用Transformer。 *arXiv preprint arXiv:1807.03819*, 2018。

[Deletang 等人, 2024] Gregoire Deletang, Anian Ruoss, Paul-Ambroise Duquenne, Elliot Catt, Tim Genewein, Christopher Mattern, Jordi Grau-Moya, Li Kevin Wenliang, Matthew Aitchison, Laurent Orseau, Marcus Hutter, 以及 Joel Veness. 语言建模是压缩。在 *The Twelfth International Conference on Learning Representations*, 2024。

[Deng等人, 2022] 明凯邓, 王建宇, 谢承平, 王艺涵, 郭汉, 舒天民, 宋萌, 邢宇, 和胡志挺。 Rlprompt: 使用强化学习优化离散文本提示。在 *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, 第3369-3391页, 2022。

[Devlin 等人, 2019] Jacob Devlin, Ming-Wei Chang, Kenton Lee 和 Kristina Toutanova. Bert: 用于语言理解的深度双向变换器的预训练。在 *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, 第 4171-4186 页, 2019。

[Ding等人, 2024] Yiran Ding, Li Lyna Zhang, Chengruidong Zhang, Yuanyuan Xu, Ning Shang, Jiahang Xu, Fan Yang, 以及Mao Yang. Longrope: 将llm上下文窗口扩展到超过200万令牌。 *arXiv preprint arXiv:2402.13753*, 2024。

[Dolan和Brockett, 2005] Bill Dolan和Chris Brockett. 自动构建一个句子释义语料库。在 *Proceedings of Third International Workshop on Paraphrasing (IWP2005)*, 2005。

[东等, 2019] 李东, 杨楠, 王文辉, 魏福如, 刘晓东, 王宇, 高建峰, 周明, 和晓鸥. 统一语言模型预训练用于自然语言理解和生成。 *Advances in neural information processing systems*, 32, 2019。

[Dong等人, 2022] Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Zhiyong Wu, Baobao Chang, Xu Sun, Jingjing Xu, 以及Zhifang Sui. 关于情境学习的调查。 *arXiv preprint arXiv:2301.00234*, 2022。

[Dong等人, 2021] 易和东, 让-巴蒂斯特·科尔德里尼, 以及安德烈亚斯·卢卡斯. 注意力并非一切: 纯注意力随着深度的增加呈双指数级下降。在 *International Conference on Machine Learning*, 第2793–2803页。PMLR, 2021。

[Drozhdov等人, 2022] Andrew Drozdov, Nathanael Schärli, Ekin Akyürek, Nathan Scales, Xinying Song, Xinyun Chen, Olivier Bousquet, 和Denny Zhou. 基于大型语言模型的组合语义解析。在 *Proceedings of The Eleventh International Conference on Learning Representations*, 2022。

[Dua 等人, 2022] Dheeru Dua, Shivanshu Gupta, Sameer Singh 和 Matt Gardner. 复杂问题的连续提示分解。在 *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, 第 1251–1265 页, 2022。

[Dubey 等人, 2024] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, 等人。Llama 3 模型群。arXiv preprint arXiv:2407.21783, 2024。

[杜博瓦等人, 2024] Yann Dubois, 陈雪晨 李, 罗汉 陶里, 田艺 张, 伊沙恩 古拉贾尼, 吉米 巴, 卡洛斯 格斯特林, 佩尔西 S 李昂, 以及 稻叶 稔典雄。Alpacafarm: 一种从人类反馈中学习的方法的模拟框架。Advances in Neural Information Processing Systems, 36, 2024。

[Eisenstein 等人, 2023] Jacob Eisenstein, Chirag Nagpal, Alekh Agarwal, Ahmad Beirami, Alex D'Amour, DJ Dvijotham, Adam Fisch, Katherine Heller, Stephen Pfohl, Deepak Ramachandran, 以及 Peter Shaw。助人或放牧? 奖励模型集成减轻但并未消除奖励黑客行为。arXiv preprint arXiv:2312.09244, 2023。

[Elsken 等人, 2019] Thomas Elsken, Jan Hendrik Metzen, 和 Frank Hutter。神经架构搜索: 综述。Journal of Machine Learning Research, 20(55): 1–21, 2019。

[Erhan 等人, 2010] Dumitru Erhan, Aaron Courville, Yoshua Bengio 和 Pascal Vincent。为什么无监督预训练有助于深度学习? 在 Proceedings of the thirteenth international conference on artificial intelligence and statistics, 第201-208页, 2010年。

[Fan 等人, 2018] Angela Fan, Mike Lewis 和 Yann Dauphin。层次化神经故事生成。在 Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), 第889-898页, 2018。

[Fan 等人, 2019] Angela Fan, Edouard Grave 和 Armand Joulin。需求驱动下减少 transformer 深度与结构化 dropout。在 Proceedings of International Conference on Learning Representations, 2019。

[Fedus 等人, 2022] William Fedus, Barret Zoph 和 Noam Shazeer。切换变压器: 通过简单高效的稀疏性扩展到万亿参数模型。The Journal of Machine Learning Research, 23(1):5232–5270, 2022。

[Fernandes 等人, 2023] Patrick Fernandes, Aman Madaan, Emmy Liu, António Farinhas, Pedro Henrique Martins, Amanda Bertsch, José G. C. de Souza, Shuyan Zhou, Tongshuang Wu, Graham Neubig, 以及 André F. T. Martins。桥梁差距: 关于自然语言生成中(人类)反馈整合的综述。Transactions of the Association for Computational Linguistics, 11:1643–1668, 2023。

[Forney, 1972] GDJR Forney。存在符号间干扰时数字序列的最大似然序列估计。IEEE Transactions on Information theory, 18(3): 363–378, 1972。

[Franklin 和 Graesser, 1996] Stan Franklin 和 Art Graesser。是一个智能体, 还是一个程序? : 自主智能体的分类法。在 International workshop on agent theories, architectures, and languages, 第21-35页。Springer, 1996。

[Frensch 和 Funke, 2014] Peter A Frensch 和 Joachim Funke。Complex problem solving: The European perspective。心理学出版社, 2014。

[盖尔等人, 2019] 特雷弗·盖尔, 埃里希·埃尔森, 萨拉·胡克。深度神经网络稀疏状态。arXiv preprint arXiv:1902.09574, 2019。

[冈古利等, 2023] 深度冈古利, 阿曼达·阿斯凯尔, 尼古拉斯·施伊弗, 托马斯·I·廖, 卡米勒·卢科·西乌特, 安娜·陈, 安娜·戈尔迪, 阿齐亚·米尔霍塞尼, 凯瑟琳·奥尔松, 丹尼·赫南德斯, 多恩·德雷恩, 达斯汀·李, 埃利·特兰·约翰逊, 伊桑·佩雷斯, 杰克逊·科尔尼恩, 杰米·科尔, 贾里德·穆勒, 约书亚·兰道, 卡马尔·恩杜西, 卡里娜·阮, 莱安·洛维特, 迈克尔·塞利托, 内森·埃尔哈格, 诺埃米·马尔科, 诺瓦·达斯萨, 奥利弗·劳施, 罗伯特·拉斯恩比, 罗宾·拉尔森, 山姆·林格, 桑迪潘·库杜, 萨乌拉夫·卡达瓦特, 斯科特·约翰逊, 肖娜·克拉维克, 谢尔·埃尔·舒克, 塔梅拉·兰厄姆, 蒂莫西·特利恩·劳顿, 汤姆·亨利汉, 特里斯坦·休姆, 苑涛·白, 扎克·哈特菲尔德-多兹, 本·曼恩, 达里奥·阿莫迪, 尼古拉斯·约瑟夫, 山姆·麦克安德利斯, 汤姆·布朗, 克里斯托弗·奥拉, 杰克·克拉克, 塞缪尔·R·鲍曼, 以及贾里德·卡普兰。大型语言模型中道德自我纠正的能力。arXiv preprint arXiv:2302.07459, 2023。

[高等人, 2023] 高Leo, John Schulman, 和 Jacob Hilton。奖励模型过度优化的缩放定律。在 International Conference on Machine Learning, 第10835–10866页。PMLR, 2023a。

[高等人, 2023] 高露玉, 阿曼·马达安, 周淑媛, 尤里·阿隆, 刘鹏飞, 杨一鸣, 杰米·卡拉恩, 格雷厄姆·纽比格。Pal: 程序辅助语言模型。在 *International Conference on Machine Learning*, 第10764–10799页。PMLR, 2023b。

[高etal.,2023] 高云帆, 熊云, 高欣宇, 贾康翔, 潘金流, 毕宇熙, 戴毅, 孙佳伟, 王浩奋。大型语言模型的检索增强生成: 综述。 *arXiv preprint arXiv:2312.10997*, 2023c。

[Garg等人, 2022] Shivam Garg, Dimitris Tsipras, Percy S Liang和Gregory Valiant。变换器在上下文中可以学习什么? 简单函数类的一个案例研究。 *Advances in Neural Information Processing Systems*, 35:30583–30598, 2022。

[Ge等人, 2024] 元歌, 刘一伦, 胡驰, 孟伟斌, 陶世民, 赵晓峰, 马红霞, 张丽, 陈博兴, 杨浩, 李贝, 肖彤, 朱静波。聚类和排名: 通过专家对齐的质量估计实现多样性保护的指令选择。 *arXiv preprint arXiv:2402.18191*, 2024。

[2024年Gemma团队] Google DeepMind Gemma团队。Gemma: 基于Gemini研究和技术的开源模型, 2024。

[Goodhart, 1984] 查尔斯·A·E·古德哈特。 *Problems of monetary management: the UK experience*。斯普林格, 1984。

[Gordon等人, 2021] Mitchell A Gordon, Kevin Duh和Jared Kaplan。神经机器翻译的数据和参数缩放定律。在 *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, 第5915–5922页, 2021。

[郭等, 2024] 郭志斌, 邵志红, 龚叶云, 杨宇久, 端楠, 陈伟竹, 等。Critic: 大型语言模型可以通过工具交互式批评进行自我纠正。在 *The Twelfth International Conference on Learning Representations*, 2024。

[Gu和Dao, 2023] Albert Gu和Tri Dao。Mamba: 使用选择性状态空间的线性时间序列建模。 *arXiv preprint arXiv:2312.00752*, 2023。

[Gunasekar等人, 2023] Suriya Gunasekar, Yi Zhang, Jyoti Aneja, Caio César Teodoro Mendes, Allie Del Giorno, Sivakanth Gopi, Mojan Javaheripi, Piero Kauffmann, Gustavo de Rosa, Olli Saarikivi, Adil Salim, Shital Shah, Harkirat Singh Behl, Xin Wang, Sébastien Bubeck, Ronen Eldan, Adam Tauman Kalai, Yin Tat Lee, Yuanzhi Li。你只需要教科书。 *arXiv preprint arXiv:2306.11644*, 2023。

[Guo等人, 2024] 青燕郭, 王瑞, 郭俊良, 李贝, 宋凯涛, 谭旭, 刘国清, 边江, 杨玉久。将大型语言模型与进化算法相结合产生强大的提示优化器。在 *The Twelfth International Conference on Learning Representations*, 2024。

[古普塔和贝兰特, 2020] 安尼克·古普塔和乔纳森·贝兰特。Gmat: 用于Transformer的全局内存增强。 *arXiv preprint arXiv:2006.03274*, 2020。

[Gupta等人, 2021] Ankit Gupta, Guy Dar, Shaya Goodman, David Ciprut和Jonathan Berant。通过p-k注意力实现内存高效的Transformer。在 *Proceedings of the Second Workshop on Simple and Efficient Natural Language Processing*, 第39–52页, 2021。

[汉等, 2021] 徐汉, 张正艳, 丁宁, 顾宇仙, 刘晓, 火宇琪, 邱杰中, 张亮, 韩文涛, 黄敏丽, 金琴, 兰艳艳, 刘阳, 刘志远, 卢志武, 邱锡鹏, 宋瑞华, 唐杰, 文继荣, 袁金辉, 赵新, 赵威, 朱军。预训练模型: 过去、现在和未来。 *AI Open*, 2:225–250, 2021。

[Han等人, 2024] Zeyu Han, Chao Gao, Jinyang Liu, Jeff Zhang, 和Sai Qian Zhang。大型模型的参数高效微调: 全面综述。 *arXiv preprint arXiv:2403.14608*, 2024。

[Harlap等人, 2018] Aaron Harlap, Deepak Narayanan, Amar Phanishayee, Vivek Seshadri, Nikhil Deva-nur, Greg Ganger, 以及Phil Gibbons。Pipedream: 快速高效的管道并行DNN训练。 *arXiv preprint arXiv:1806.03377*, 2018。

[He等人, 2019] Kaiming He, Ross Girshick和Piotr Dollár。重新思考ImageNet预训练。在

- Proceedings of the IEEE/CVF International Conference on Computer Vision*, 第4918-4927页, 2019年。
- [He等人, 2021] 彭程 何, 刘晓东, 高建峰, 陈伟竹. Deberta: 解码增强的BERT与解耦注意力。在 *Proceedings of International Conference on Learning Representations*, 2021。
- [Hendrycks和Gimpel, 2016] Dan Hendrycks和Kevin Gimpel. 高斯误差线性单元 (gelu)。 *arXiv preprint arXiv:1606.08415*, 2016。
- [Hendrycks等人, 2020] Dan Hendrycks, Xiaoyuan Liu, Eric Wallace, Adam Dziedziec, Rishabh Krishnan和Dawn Song. 预训练的Transformer提高了分布外鲁棒性。在 *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 第2744-2751页, 2020。
- [Hendrycks等人, 2021] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, 以及Jacob Steinhardt. 测量大规模多任务语言理解。在 *Proceedings of International Conference on Learning Representations*, 2021。
- [Hestness等人, 2017] Joel Hestness, Sharan Narang, Newsha Ardalani, Gregory Diamos, Heewoo Jun, Hassan Kianinejad, Md Mostofa Ali Patwary, Yang Yang, 以及Yanqi Zhou. 深度学习扩展是可预测的, 经验上。 *arXiv preprint arXiv:1712.00409*, 2017。
- [Hewitt, 2024] 约翰·休伊特。无需指令调整的指令遵循, 2024。URL <https://nlp.stanford.edu/~johnhew/instruction-following.html>。
- [Hewitt等人, 2024] 约翰·休伊特, 刘纳森·F·刘, 李佩奇, 以及克里斯托弗·D·曼宁。无需指令微调的指令遵循。 *arXiv preprint arXiv:2409.14254*, 2024。
- [Hochreiter 和 Schmidhuber, 1997] Sepp Hochreiter 和 Jürgen Schmidhuber. 长短期记忆。 *Neural computation*, 9(8): 1735–1780, 1997。
- [Hoffmann等人, 2022] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, 以及Laurent Sifre. 训练计算最优的大型语言模型。 *arXiv preprint arXiv:2203.15556*, 2022。
- [Holtzman等人, 2020] Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes和Yejin Choi. 神经文本退化的奇特案例。在 *International Conference on Learning Representations*, 2020。
- [Honovich 等人, 2023] Honovich, 托马斯·斯奇亚洛姆, 奥梅·列维和蒂莫·施克。非自然指令: 无需 (几乎) 人工劳动调整语言模型。在 *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 第 14409–14428 页, 2023。
- [Houlsby等人, 2019] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, 和Sylvain Gelly. 针对NLP的参数高效迁移学习。在 *Proceedings of the 36th International Conference on Machine Learning*, 第2790–2799页。P MLR, 2019。
- [胡等, 2022] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yanzhi Li, Shean Wang, Lu Wang 和 Weizhu Chen. LoRA: 大型语言模型的低秩自适应。在 *International Conference on Learning Representations*, 2022。
- [黄, 2009] 梁黄。基于动态规划的NLP搜索算法。在 *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Tutorial Abstracts*, 2009。
- [黄等, 2019] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Mia Xu Chen, Dehao Chen, Hyoungho Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, and Zhifeng Chen. Gpipe: 使用管道并行性高效训练巨型神经网络。 *Advances in neural information processing systems*, 32, 2019。
- [Hutchins等人, 2022] DeLesley Hutchins, Imanol Schlag, Yuhuai Wu, Ethan Dyer和Behnam Neyshabur. 块循环Transformer。 *Advances in neural information processing systems*, 35:

33248–33261, 2022。

[Jelinek, 1998] 弗雷德里克·杰利内克。 *Statistical methods for speech recognition*。麻省理工学院出版社, 1998年。

[江等, 2023] Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, L  lio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timoth  e Lacroix, 和William El Sayed. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023a。

[Jiang等人, 2023] 汇强江, 吴倩辉, 林钦业, 杨雨晴, 邱丽丽。LlmLingua: 压缩大语言模型推理的提示以加速推理。在 *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 第13358–13376页, 2023b。

[Jiang等人, 2020] 郑宝江, 徐福福, 荒木淳, 以及纽比格·格雷厄姆。我们如何知道语言模型知道什么? *Transactions of the Association for Computational Linguistics*, 8:423–438, 2020。

[Jiao等人, 2020] 小琪·焦, 尹一春, 商立锋, 姜新, 陈晓, 李琳琳, 王芳, 刘群。Tinybert: 自然语言理解的BERT蒸馏。在 *Findings of the Association for Computational Linguistics: EMNLP 2020*, 第4163–4174页, 2020。

[Joshi等人, 2017] Mandar Joshi, Eunsol Choi, Daniel S Weld, 和Luke Zettlemoyer. Triviaqa: 一个用于阅读理解的规模庞大的远程监督挑战数据集。在 *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 第1601–1611页, 2017。

[Joshi等人, 2020] Mandar Joshi, Danqi Chen, Yinhan Liu, Daniel S Weld, Luke Zettlemoyer, 和Omer Levy。Spanbert: 通过表示和预测跨度来改进预训练。 *Transactions of the association for computational linguistics*, 8:64–77, 2020。

[Jurafsky 和 Martin, 2008] Dan Jurafsky 和 James H. Martin. *Speech and Language Processing (2nd ed.)*. Prentice Hall, 2008。

[卡尼曼, 2011] 丹尼尔·卡尼曼。 *Thinking, fast and slow*。麦克米伦, 2011。

[Kaplan等人, 2020] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Re-won Child, Scott Gray, Alec Radford, Jeffrey Wu, 以及Dario Amodei。神经语言模型的缩放定律。 *arXiv preprint arXiv:2001.08361*, 2020。

[Katharopoulos等人, 2020] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas和Fran  ois Fleuret。Transformers是RNNs: 具有线性注意力的快速自回归Transformer。在 *International conference on machine learning*, 第5156–5165页。PMLR, 2020。

[Khandelwal等人, 2020] Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer和Mike Lewis。通过记忆实现泛化: 最近邻语言模型。在 *International Conference on Learning Representations*, 2020。

[Khot等人, 2023] Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark和Ashish Sabharwal。分解提示: 解决复杂任务的模块化方法。在 *Proceedings of The Eleventh International Conference on Learning Representations*, 2023。

[金等, 2023] Sehoon Kim, Coleman Hooper, Thanakul Wattanawong, Minwoo Kang, Ruohan Yan, Hasan Genc, Grace Dinh, Qijing Huang, Kurt Keutzer, Michael W. Mahoney, Yakun Sophia Shao, 以及Amir Gholami。Transformer推理的全栈优化: 综述。 *arXiv preprint arXiv:2302.14017*, 2023。

[Kirkpatrick等人, 2017] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, 和Raia Hadsell。克服神经网络中的灾难性遗忘。 *Proceedings of the national academy of sciences*, 114(13): 3521–3526, 2017。

- [科恩, 2010] 菲利普·科恩。 *Statistical Machine Translation*。剑桥大学出版社, 2010年。
- [Kojima等人, 2022] Kojima Takeshi, Gu Shixiang Shane, Reid Machel, Matsuo Yutaka, Iwasawa Yusuke. 大型语言模型是零样本推理者。 *Advances in neural information processing systems*, 35:22199–22213, 2022。
- [Korthikanti等人, 2023] Vijay Anand Korthikanti, Jared Casper, Sangkug Lym, Lawrence McAfee, Michael Andersch, Mohammad Shoeybi和Bryan Catanzaro. 在大型Transformer模型中减少激活重计算。 *Proceedings of Machine Learning and Systems*, 5, 2023。
- [Krakovna等人, 2020] Victoria Krakovna, Jonathan Uesato, Vladimir Mikulik, Matthew Rahtz, Tom Everitt, Ramana Kumar, Zac Kenton, Jan Leike, 和Shane Legg. 规格游戏: AI独创性的另一面。 <https://deepmind.google/discover/blog/specification-gaming-the-flip-side-of-ai-ingenuity>, 2020。
- [库马尔和伯恩, 2004] 沙恩卡·库马尔和威廉·伯恩。统计机器翻译的最小贝叶斯风险解码。在 *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics: HLT-NAACL 2004*, 第169-176页, 2004。
- [Kung和Peng, 2023] Po-Nien Kung和Nanyun Peng. 模型真的学会了遵循指令吗? 指令微调的实证研究。 *arXiv preprint arXiv:2305.11383*, 2023。
- [Kwon等人, 2023] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E Gonzalez, Hao Zhang, 和Ion Stoica. 使用分页注意力的大语言模型服务的有效内存管理。 *arXiv preprint arXiv:2309.06180*, 2023。
- [湖和巴罗尼, 2018] 布雷登·湖和马可·巴罗尼。没有系统性的泛化: 关于序列到序列循环网络的组合技能。在 *International conference on machine learning*, 第2873–2882页。PMLR, 2018。
- [Lambert等人, 2024] Nathan Lambert, Valentina Pyatkin, Jacob Morrison, LJ Miranda, Bill Yuchen Lin, Khyathi Chandu, Nouha Dziri, Sachin Kumar, Tom Zick, Yejin Choi, Noah A. Smith, 以及Han-naneh Hajishirzi. Rewardbench: 评估语言模型的奖励模型。 *arXiv preprint arXiv:2403.13787*, 2024。
- [Lample和Conneau, 2019] Guillaume Lample和Alexis Conneau. 跨语言语言模型预训练。 *arXiv preprint arXiv:1901.07291*, 2019。
- [Lan等人, 2020] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma和Radu Soricut. Albert: 一种用于语言表示自监督学习的轻量级BERT。在 *Proceedings of International Conference on Learning Representations*, 2020。
- [李等, 2023] 哈里森·李, 萨姆拉特·帕塔莱, 哈桑·曼苏尔, 凯莉·卢, 托马斯·梅斯纳德, 约翰·费雷特, 科尔顿·毕晓普, 伊森·霍尔, 维克多·卡布内, 阿布纳夫·拉斯特吉。Rlaif: 利用AI反馈扩展人类反馈强化学习。 *arXiv preprint arXiv:2309.00267*, 2023。
- [Lester等人, 2021] Brian Lester, Rami Al-Rfou和Noah Constant. 规模对参数高效提示调整的力量。在 *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, 第3045–3059页, 2021。
- [Leviathan等人, 2023] Yaniv Leviathan, Matan Kalman和Yossi Matias. 通过投机解码从变换器中进行快速推理。在 *Proceedings of International Conference on Machine Learning*, 第19274–19286页。PMLR, 2023。
- [刘易斯等人, 2020] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, 和Luke Zettlemoyer. Bart: 用于自然语言生成、翻译和理解的去噪序列到序列预训练。在 *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 第7871-7880页, 2020。
- [李等, 2024] 李宝林, 姜彦凯, 维贾伊·加德帕利, 以及迪韦什·蒂瓦里。LLM推理服务: 近期进展和机遇综述。 *arXiv preprint arXiv:2407.12391*, 2024a。

[李等, 2023] 李贝, 王瑞, 郭俊良, 宋凯涛, 谭旭, 哈尼·哈桑, 阿鲁尔·梅内塞斯, 肖彤, 边江, 朱静波. 先思考后生成: 文本生成增强提示框架. *arXiv preprint arXiv:2305.19835*, 2023a。

[李, 2011] 汪海. *Learning to Rank for Information Retrieval and Natural Language Processing*. 在线访问: Morgan & Claypool 综合收藏五. Morgan & Claypool 出版社, 2011. ISBN 9781608457076。

[李等, 2022] 华阳李, 苏一轩, 蔡登, 王岩, 刘乐茂. 关于检索增强文本生成的调查. *arXiv preprint arXiv:2202.01110*, 2022。

[李等, 2024] 李山达, 尤崇, 古鲁·古鲁甘内什, 约书亚·艾因斯利, 圣地亚哥·奥塔农, 马兹尼尔·扎希尔, 苏米特·桑海, 杨一鸣, 桑吉夫·库马尔, 斯里纳特·博贾纳帕利. 相对位置的功能插值改进了长上下文变换器. 在 *The Twelfth International Conference on Learning Representations*, 2024b。

[李等, 2023] 李胜贵, 薛复兆, 查伊塔尼亚·巴拉瓦尔, 李勇斌, 以及游洋. 序列并行性: 从系统视角的长序列训练. 在 *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 第2391–2404页, 2023b。

[李和梁, 2021] 李祥丽·李和珀西·梁. 前缀调整: 优化连续提示以生成. 在 *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, 第4582–4597页, 2021。

[李, 2023] 李银恒. 在上下文学习中的零样本提示设计实用调查. 在 *Proceedings of the 14th International Conference on Recent Advances in Natural Language Processing*, 第641–647页, 2023。

[李等, 2023] 李宇成, 董博, 弗朗克·古林, 林成华. 压缩上下文以提高大型语言模型的推理效率。在 *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 第342–6353页, 2023c。

[Lialin等人, 2023] Vladislav Lialin, Vijeta Deshpande和Anna Rumshisky. 缩小以扩大规模: 参数高效微调指南. *arXiv preprint arXiv:2303.15647*, 2023。

[Lightman等人, 2024] Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, 以及 Karl Cobbe. 让我们一步步验证. 在 *The Twelfth International Conference on Learning Representations*, 2024。

[刘等, 2024] 刘爱新, 冯贝, 薛冰, 王冰轩, 吴博超, 卢成达, 赵成刚, 邓成琪, 张晨宇, 阮崇, 等. Deepseek-v3 技术报告. *arXiv preprint arXiv:2412.19437*, 2024a。

[刘等, 2022] 刘嘉昌, 沈丁汉, 张一哲, 威廉·B·多兰, 劳伦斯·卡林, 陈伟竹. 什么使gpt-3的上下文示例变得优秀? 在 *Proceedings of Deep Learning Inside Out (DeeLIO 2022): The 3rd Workshop on Knowledge Extraction and Integration for Deep Learning Architectures*, 第100–114页, 2022。

[刘等, 2023] 刘鹏飞, 袁伟哲, 傅金兰, 姜正宝, 林博明, 内布比·纽比格. 预训练、提示和预测: 自然语言处理中提示方法的系统综述. *ACM Computing Surveys*, 55(9):1–35, 2023a。

[刘等, 2024] 天琪 刘, 赵耀, 里沙布·乔希, 米沙·卡哈曼, 穆罕默德·萨利赫, 刘彼得, 刘佳璐. 统计拒绝采样改进偏好优化. 在 *The Twelfth International Conference on Learning Representations*, 2024b。

[刘, 2009] 刘铁岩. 信息检索中的学习排序. *Foundations and Trends® in Information Retrieval*, 3(3): 225–331, 2009。

[刘等, 2023] 小刘, 郑艳艳, 杜正晓, 丁明, 钱宇杰, 杨志林, 唐杰. Gpt 也懂. *AI Open*, 2023b。

[刘等, 2023] 刘晓晓, 王静怡, 孙军, 袁晓涵, 董国梁, 迪鹏, 王文海, 王东霞。大型语言模型的提示框架: 综述。 *arXiv preprint arXiv:2311.12785*, 2023c。

[刘等, 2024] 刘新宇, 赵润松, 黄鹏程, 肖春阳, 李贝, 王晶刚, 肖彤, 朱景波。遗忘曲线: 评估长上下文模型记忆能力的一种可靠方法。在 *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, 第 4667–4682 页, 2024c。

[刘等人, 2019] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, 和 Veselin Stoyanov. Roberta: 一种鲁棒的BERT预训练方法。 *arXiv preprint arXiv:1907.11692*, 2019。

[Longpre 等人, 2023] Shayne Longpre, Le Hou, Tu Vu, Albert Webson, Hyung Won Chung, Yi Tay, Denny Zhou, Quoc V. Le, Barret Zoph, Jason Wei, 以及 Adam Roberts. The flan collection: 设计有效指令微调的数据和方法。在 *International Conference on Machine Learning*, 第 22631–22648 页。PMLR, 2023。

[Ma等, 2023] Xuezhe Ma, Chunting Zhou, Xiang Kong, Junxian He, Liangke Gui, Graham Neubig, Jonathan May, 和 Luke Zettlemoyer. Mega: 配备移动平均的门控注意力。在 *The Eleventh International Conference on Learning Representations*, 2023。

[Ma等, 2024] Xuezhe Ma, Xiaomeng Yang, Wenhan Xiong, Beidi Chen, Lili Yu, Hao Zhang, Jonathan May, Luke Zettlemoyer, Omer Levy, 和 Chunting Zhou. Megalodon: 具有无限上下文长度的高效llm预训练和推理。 *arXiv preprint arXiv:2404.08801*, 2024。

[Madaan等人, 2024] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegrefe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, 和 Peter Clark。自我改进: 带有自我反馈的迭代改进。 *Advances in Neural Information Processing Systems*, 36, 2024。

[Manning, 2022] Christopher D Manning. 人类语言理解与推理。 *Daedalus*, 151(2): 127–138, 2022。

[马库斯, 1993] 加里·F·马库斯。语言习得中的负面证据。 *Cognition*, 46(1): 53–85, 1993。

[Martins等人, 2022] Pedro Henrique Martins, Zita Marinho, 以及 André FT Martins. ∞ -former: 无限记忆Transformer-former: 无限记忆Transformer。在 *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 第5468–5485页, 2022。

[Mavi等人, 2024] Vaibhav Mavi, Anubhav Jangra 和 Adam Jatowt. 多跳问答。 *Foundations and Trends® in Information Retrieval*, 17(5): 457–586, 2024。

[Michel等人, 2019] Paul Michel, Omer Levy 和 Graham Neubig. 十六个头真的比一个好吗? *Advances in neural information processing systems*, 32, 2019。

[Micikevicius等人, 2018] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, 以及 Hao Wu. 混合精度训练。在 *Proceedings of International Conference on Learning Representations*, 2018。

[米蒂宁, 1999] 卡伊萨·米蒂宁。 *Nonlinear multiobjective optimization*, 第12卷。斯普林格科学和商业媒体, 1999年。

[Mikolov等人, 2013] 托马斯·米科洛夫, 凯·陈, 格雷格·科拉多, 以及杰弗里·戴恩。在向量空间中高效估计词表示。在 $\{v^*\}$, 2013a。

[Mikolov等人, 2013] 托马斯·米科洛夫, 伊利亚·苏茨克维尔, 凯·陈, 格雷格·科拉多, 以及杰弗里·戴恩。词语和短语的分布式表示及其组合性。在 *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, 第3111–3119页, 2013b。

[Min等人, 2019] Sewon Min, Victor Zhong, Luke Zettlemoyer和Hannaneh Hajishirzi。通过问题分解和重新评分的多跳阅读理解。在*Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 第6097-6109页, 2019。

[Minaee等人, 2024] Shervin Minaee, Tomas Mikolov, Narjes Nikzad, Meysam Chenaghlu, Richard Socher, Xavier Amatriain, 以及Jianfeng Gao。大型语言模型: 综述。 *arXiv preprint arXiv:2402.06196*, 2024。

[Mishra等人, 2022] Swaroop Mishra, Daniel Khashabi, Chitta Baral和Hannaneh Hajishirzi。通过自然语言众包指令实现跨任务泛化。在*Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 第3470–3487页, 2022。

[Mnih等人, 2016] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Tim Harley, Timothy P Lillicrap, David Silver, 以及Koray Kavukcuoglu。异步深度强化学习方法。在*Proceedings of the 33rd International Conference on International Conference on Machine Learning*, 第1928-1937页, 2016。

[莫哈塔沙米和贾吉, 2024] 阿米尔基万·莫哈塔沙米和马丁·贾吉。Transformer的随机访问无限上下文长度。 *Advances in Neural Information Processing Systems*, 36, 2024。

[Mu等人, 2024] Jesse Mu, 李翔, 和Noah Goodman。使用梗概标记学习压缩提示。 *Advances in Neural Information Processing Systems*, 36, 2024。

[Munkhdalai等人, 2024] Tsendsuren Munkhdalai, Manaal Faruqui和Siddharth Gopal。留下无上下文: 高效的无限上下文变换器与infini-attention。 *arXiv preprint arXiv:2404.07143*, 2024。

[中野等, 2021] 中野理一郎, 雅各布·希尔顿, 苏奇尔·巴拉吉, 吴杰, 欧阳龙, 金晶娜, 克里斯托弗·赫斯, 沙坦努·贾因, 维奈特·科萨拉朱, 威廉·桑德斯, 江旭, 卡尔·科布, 蒂娜·埃隆杜, 格雷琴·克鲁格, 凯文·巴顿, 马修·奈特, 本杰明·切斯, 约翰·舒尔曼。Webgpt: 带有人类反馈的浏览器辅助问答。 *arXiv preprint arXiv:2112.09332*, 2021。

[Narayanan等人, 2021] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, Amar Phanishayee, 以及Matei Zaharia。使用megatron-lm在GPU集群上高效进行大规模语言模型训练。在*Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 第1-15页, 2021年。

[Ng等人, 1999] Andrew Y Ng, Daishi Harada, 和Stuart J Russell。奖励变换下的策略不变性: 理论与应用于奖励塑造。在*Proceedings of the Sixteenth International Conference on Machine Learning*, 第278-287页, 1999。

[Nvidia, 2025] Nvidia. Nvidia nim llms 性能基准。 <https://docs.nvidia.com/nim/benchmarking/llm/latest/metrics.html>, 2025。检索日期: 2025-03-17。

[OpenAI, 2024] OpenAI. 使用llms进行推理的学习, 2024年9月。URL <https://openai.com/index/learning-to-reason-with-llms/>。

[欧阳等, 2022] 欧阳, 吴杰夫, 姜旭, 迪戈·阿尔梅达, 卡罗尔·L·韦恩赖特, 帕梅拉·米什金, 张冲, 桑迪尼·阿加瓦尔, 卡塔琳娜·斯拉马, 亚历克斯·雷, 约翰·舒尔曼, 雅各布·希尔顿, 弗莱泽·凯尔顿, 卢克·米勒, 麦迪·西蒙斯, 阿曼达·阿斯科尔, 彼得·韦林德, 保罗·F·克里斯蒂亚诺, 简·莱克, 以及瑞安·洛。通过人类反馈训练语言模型以遵循指令。 *Advances in Neural Information Processing Systems*, 35:27730–27744, 2022。

[Pal等人, 2023] Koyena Pal, Jiuding Sun, Andrew Yuan, Byron C Wallace和David Bau。未来镜头: 从单个隐藏状态预测后续标记。在*Proceedings of the 27th Conference on Computational Natural Language Learning (CoNLL)*, 第548-560页, 2023。

[潘等, 2022] Alexander Pan, Kush Bhatia, 和 Jacob Steinhardt. 奖励不具体化的影响: 映射和缓解不匹配模型。在 *International Conference on Learning Representations*, 2022。

[潘等, 2024] 潘良明, 迈克尔·萨克斯, 许文达, 迪帕克·纳塔尼, 王欣怡, 王威廉杨。自动纠正大型语言模型: 调查多样化的自动纠正策略。

Transactions of the Association for Computational Linguistics, 12:484–506, 2024。

[Parisi等人, 2022] Aaron Parisi, 赵耀, Noah Fiedel. Talm: 工具增强语言模型。
arXiv preprint arXiv:2205.12255, 2022。

[Parisi等人, 2019] 德国 I Parisi, Ronald Kemker, Jose L Part, Christopher Kanan, 和 Stefan Wermter. 使用神经网络的持续终身学习: 综述。 *Neural networks*, 113:54–71, 2019。

[Parmar等人, 2018] Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Noam Shazeer, Alexander Ku 和 Dustin Tran. 图像变换器。在 *International conference on machine learning*, 第4055–4064页。PMLR, 2018。

[Patel等人, 2024] Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Íñigo Goiri, Saeed Maleki, 和 Ricardo Bianchini. Splitwise: 使用相位分割的高效生成式llm推理。在 *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, 第118–132页。IEEE, 2024。

[Penedo 等人, 2023] Guilherme Penedo, Quentin Malartic, Daniel Hesslow, Ruxandra Cojocaru, Alessandro Cappelli, Hamza Alobeidli, Baptiste Pannier, Ebtesam Almazrouei 和 Julien Launay. 针对 falcon llm 的提炼网络数据集: 使用网络数据超越精心制作的语料库, 以及仅使用网络数据。 *arXiv preprint arXiv:2306.01116*, 2023。

[Peng等人, 2024] Bowen Peng, Jeffrey Quesnelle, Honglu Fan 和 Enrico Shippole. YaRN: 大型语言模型的效率上下文窗口扩展。在 *The Twelfth International Conference on Learning Representations*, 2024。

[Pennington等人, 2014] Jeffrey Pennington, Richard Socher 和 Christopher D. Manning. Glove: 用于词表示的全局向量。在 *Proceedings of Empirical Methods in Natural Language Processing (EMNLP)*, 第1532–1543页, 2014。

[Peters等人, 2018] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, 和 Luke Zettlemoyer. 深度上下文文化词表示。在 *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, 2018。

[Plackett, 1975] Robin L Plackett. 排列分析。 *Journal of the Royal Statistical Society Series C: Applied Statistics*, 24(2):193–202, 1975。

[Pope等人, 2023] Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Jonathan Heek, Kefan Xiao, Shivani Agrawal, 和 Jeff Dean. 高效扩展transformer推理。在 *Proceedings of Machine Learning and Systems*, 2023。

[Prasad 等人, 2023] Archiki Prasad, Peter Hase, Xiang Zhou 和 Mohit Bansal. Grips: 用于提示大型语言模型的梯度自由、基于编辑的指令搜索。在 *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, 第3845–3864页, 2023。

[Press等人, 2022] Ofir Press, Noah Smith 和 Mike Lewis. 训练短, 测试长: 具有线性偏差的注意力机制实现输入长度外推。在 *Proceedings of International Conference on Learning Representations*, 2022。

[Press等人, 2023] Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A Smith 和 Mike Lewis. 在语言模型中测量和缩小组合性差距。在 *Findings of the Association for Computational Linguistics: EMNLP 2023*, 第5687–5711页, 2023。

[Pryzant等人, 2023] Reid Pryzant, Dan Iter, Jerry Li, Yin Tat Lee, Chenguang Zhu, 和 Michael Zeng。

自动提示优化使用“梯度下降”和束搜索。在 *The 2023 Conference on Empirical Methods in Natural Language Processing*, 2023。

[邱等, 2020] 邱锡鹏, 孙天翔, 徐一戈, 邵云帆, 戴宁, 黄宣静。自然语言处理预训练模型: 综述。 *Science China Technological Sciences*, 63(10): 1872–1897, 2020。

[Radford 等人, 2018] Alec Radford, Karthik Narasimhan, Tim Salimans 和 Ilya Sutskever。通过生成预训练改进语言理解。 *OpenAI*, 2018。

[Radford 等人, 2019] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei 和 Ilya Sutskever。语言模型是无监督的多任务学习器。 *OpenAI blog*, 1(8), 2019。

[Radford 等人, 2021] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, 以及 Ilya Sutskever。从自然语言监督中学习可迁移的视觉模型。在 *International conference on machine learning*, 第 8748–8763 页。PMLR, 2021。

[Rae 等人, 2019] Jack W Rae, Anna Potapenko, Siddhant M Jayakumar, Chloe Hillier, 和 Timothy P Lillicrap。压缩变换器用于长序列建模。在 *International Conference on Learning Representations*, 2019。

[Rafailov 等人, 2024] Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, 和 Chelsea Finn。直接偏好优化: 你的语言模型其实是一个奖励模型。 *Advances in Neural Information Processing Systems*, 36, 2024。

[Raffel 等人, 2020] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, 以及 Peter J. Liu。使用统一的文本到文本转换器探索迁移学习的极限。 *Journal of Machine Learning Research*, 21(140): 1–67, 2020。

[拉马钱德兰等人, 2017] 普拉吉特·拉马钱德兰、巴雷特·佐夫和吴国伟·莱。寻找激活函数。 *arXiv preprint arXiv:1710.05941*, 2017。

[Rolnick 等人, 2019] David Rolnick, Arun Ahuja, Jonathan Schwarz, Timothy Lillicrap 和 Gregory Wayne。持续学习的经验回放。 *Advances in Neural Information Processing Systems*, 32, 2019。

[Rosenfeld 等人, 2020] Jonathan S Rosenfeld, Amir Rosenfeld, Yonatan Belinkov 和 Nir Shavit。在 *Proceedings of International Conference on Learning Representations* 中关于跨尺度泛化错误的构造性预测。

[阮等, 2024] 阮俊豪, 孟龙, 山伟桥, 肖彤, 朱景波。关于 llm 调查的综述。
<https://github.com/NiuTrans/ABigSurveyOfLLMs>, 2024。

[Rubin 等人, 2022] Ohad Rubin, Jonathan Herzig 和 Jonathan Berant。学习检索用于情境学习的提示。在 *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 第 2655–2671 页, 2022。

[Russell, 2019] 斯图尔特·罗素。
Human Compatible: Artificial Intelligence and the Problem of Controls。维京出版社, 2019年。

[Sanh 等人, 2020] Victor Sanh, Thomas Wolf 和 Alexander Rush。移动剪枝: 通过微调实现自适应稀疏性。 *Advances in Neural Information Processing Systems*, 33:20378–20389, 2020。

[Sanh 等人, 2022] Victor Sanh, Albert Webson, Colin Raffel, Stephen Bach, Lintang Sutawika, Zaid Alyafeai, Antoine Chaffin, Arnaud Stiegler, Arun Raja, Manan Dey, M Saiful Bari, Canwen Xu, Urmish Thakker, Shanya Sharma Sharma, Eliza Szczechla, Taewoon Kim, Gunjan Chhablani, Nihal Nayak, Debajyoti Datta, Jonathan Chang, Mike Tian-Jian Jiang, Han Wang, Matteo Manica, Sheng Shen, Zheng Xin Yong, Harshit Pandey, Rachel Bawden, Thomas Wang, Trishala Neeraj, Jos Rozen, Abheesht Sharma, Andrea Santilli, Thibault Fevry, Jason Alan Fries, Ryan Teehan, Teven Le Scao, Stella Biderman, Leo Gao, Thomas Wolf, 和 Alexander M Rush。多任务提示训练实现零样本任务泛化。在 *Proceedings of International Conference on Learning Representations*, 2022。

[施克等人, 2023] Timo Schick, Jane A. Yu, 江正宝, 法比奥·佩特罗尼, 帕特里克·刘易斯, 高蒂尔·伊扎卡尔, 尤清飞, 克里斯托φ·罗斯·纳尔姆帕蒂斯, 爱德华·格雷夫, 塞巴斯蒂安·里德尔。PEER: 一个协作语言模型。在

Proceedings of The Eleventh International Conference on Learning Representations, 2023。

[施克等人, 2024] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, 以及Thomas Scialom。Toolformer: 语言模型可以自学使用工具。 *Advances in Neural Information Processing Systems*, 36, 2024。

[施密德huber, 2015] 朱尔根·施密德huber. 神经网络中的深度学习: 概述。 *Neural networks*, 61:85–117, 2015。

[Schulman等人, 2015] 约翰·舒尔曼, 谢尔盖·莱文, 菲利普·莫里茨, 迈克尔·乔丹, 皮埃特·阿贝贝。信任域策略优化。在 *Proceedings of the 32nd International Conference on International Conference on Machine Learning-Volume 37*, 第1889-1897页, 2015年。

[Schulman等人, 2017] 约翰·舒尔曼, 菲利普·沃尔斯基, 普拉夫拉·达里瓦尔, 亚历克·拉福德, 以及奥列格·克里莫夫。近端策略优化算法。 *arXiv preprint arXiv:1707.06347*, 2017。

[Sennrich等人, 2016] Rico Sennrich, Barry Haddow和Alexandra Birch。利用单语数据改进神经机器翻译模型。在 *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 第86-96页, 2016年。

[Seo等人, 2017] Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi和Hannaneh Hajishirzi。双向注意力流用于机器理解。在 *Proceedings of International Conference on Learning Representations*, 2017。

[香农, 1951] 克劳德·香农。印刷英语的预测和熵。 *Bell system technical journal*, 30(1):50–64, 1951。

[Shaw等人, 2018] Peter Shaw, Jakob Uszkoreit和Ashish Vaswani。带相对位置表示的自注意力。在 *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, 第464-468页, 2018。

[Shazeer, 2019] Noam Shazeer。快速Transformer解码: 你需要一个写头就足够了。 *arXiv preprint arXiv:1911.02150*, 2019。[Shazeer, 2020] Noam Shazeer。Glu变体改进Transformer。 *arXiv preprint arXiv:2002.05202*, 2020。

[沈等, 2020] 沈生, 董振, 叶佳宇, 马林健, 姚哲伟, 阿米尔·戈拉米, 迈克尔·W·马霍尼, 和库尔特·库泽特。Q-bert: 基于Hessian的BERT超低精度量化。在 *Proceedings of the AAAI Conference on Artificial Intelligence*, 第34卷, 第8815-8821页, 2020。

[Shinn等人, 2023] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan和Shunyu Yao。Reflexion: 具有语言强化学习的语言代理。 *Advances in Neural Information Processing Systems*, 36:8634–8652, 2023。

[Shoeybi等人, 2019] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper和Bryan Catanzaro。Megatron-LM: 使用模型并行训练数十亿参数的语言模型。 *arXiv preprint arXiv:1909.08053*, 2019。

[Skalse等人, 2022] Joar Skalse, Nikolaus Howe, Dmitrii Krashennnikov和David Krueger。定义和表征奖励游戏。 *Advances in Neural Information Processing Systems*, 35:9460–9471, 2022。

[Snell 等人, 2022] Charlie Snell, Dan Klein 和 Ruiqi Zhong。通过蒸馏上下文学习。 *arXiv preprint arXiv:2209.15189*, 2022。[Snell 等人, 2024] Charlie Snell, Jaehoon Lee, Kelvin Xu 和 Aviral Kumar。最优地扩展 llm 测试时计算比扩展模型参数更有效。 *arXiv preprint arXiv:2408.03314*, 2024。

[Snell 等人, 2025] Charlie Victor Snell, Jaehoon Lee, Kelvin Xu 和 Aviral Kumar. 最优地扩展 LLM 测试时计算比扩展参数进行推理更有效。在 *The Thirteenth International Conference on Learning Representations*, 2025。

[Socher 等人, 2013] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, 和 Christopher Potts. 在 *Proceedings of the 2013 conference on empirical methods in natural language processing* 中, 关于情感树库上的语义组合性的递归深度模型。在 2013 年, 第 1631–1642 页。

[Song 等人, 2019] Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, 和 Tie-Yan Liu. 质量: 掩码序列到序列预训练用于语言生成。在 *International Conference on Machine Learning*, 第 5926–5936 页。PM LR, 2019。

[Stiennon 等人, 2020] Nisan Stiennon, Long Ouyang, Jeffrey Wu, Daniel Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, 以及 Paul F Christiano. 利用人类反馈进行总结学习。 *Advances in Neural Information Processing Systems*, 33:3008–3021, 2020。

[苏等, 2024] 建林 苏, 穆尔塔达·艾哈迈德, 吕宇, 潘胜锋, 博文, 刘云峰. Ro-former: 具有旋转位置嵌入的增强型 Transformer。 *Neurocomputing*, 568:127063, 2024。

[苏等, 2022] 苏一璇, 蓝天, 王岩, 丹尼·约加塔玛, 孔凌鹏, 奈杰尔·科利尔. 用于神经文本生成的对比框架。 *Advances in Neural Information Processing Systems*, 35:21548–21561, 2022。

[Sun 等人, 2020] 智庆孙, 余红坤, 宋晓丹, 刘仁杰, 杨一鸣, 周登宇. Mobilebert: 一种紧凑型任务无关的 BERT, 适用于资源受限设备。在 *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 第 2158–2170 页, 2020。

[Sutskever 等人, 2014] 伊利亚·苏茨克维, 奥里奥尔·维尼亚尔斯, 和 国外·V·莱. 使用神经网络的序列到序列学习。 *Advances in neural information processing systems*, 27, 2014。

[Sutton 和 Barto, 2018] Richard S. Sutton 和 Andrew G. Barto. *Reinforcement Learning: An Introduction (2nd ed.)*. 麻省理工学院出版社, 2018。

[Szepesvári, 2010] Csaba Szepesvári. 强化学习算法。 *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 4(1):1–103, 2010。

[Talmor 和 Berant, 2018] Alon Talmor 和 Jonathan Berant. 网络作为回答复杂问题的知识库。 *arXiv preprint arXiv:1803.06643*, 2018。

[Taori 等人, 2023] Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, 和 Tatsunori B. Hashimoto. 斯坦福 Alpaca: 一种遵循指令的 Llama 模型。 https://github.com/tatsu-lab/stanford_alpaca, 2023。

[Tay 等人, 2020] Yi Tay, Mostafa Dehghani, Dara Bahri 和 Donald Metzler. 高效 Transformer: 综述。 *CoRR*, abs/2009.06732, 2020。

[Team 等人, 2024] Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, 等人. Gemma 2: 在实用规模上改进开放语言模型。 *arXiv preprint arXiv:2408.00118*, 2024。

[Teknium, 2023] Teknium. Openhermes 2.5: 通用 LLM 助手的合成数据开放集, 2023。 URL <https://huggingface.co/datasets/teknium/OpenHermes-2.5>。

[Timonin 等人, 2022] Denis Timonin, BoYang Hsueh 和 Vinh Nguyen. 使用 nvidia triton 推理服务器加速大型 Transformer 模型的推理。 <https://developer.nvidia.com/blog/accelerated-inference-for-large-transformer-models-using-nvidia-fastertransformer-and-nvidia-triton>, 2022 年。

[Touvron 等人, 2023] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, 和 Guillaume Lample. Llama: 开放且高效的基语言模型。 *arXiv preprint arXiv:2302.13971*, 2023a。

[Touvron等人, 2023] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, 以及 Thomas Scialom. Llama 2: 开放基础和微调聊天模型。arXiv preprint arXiv:2307.09288, 2023b。

[渡边等, 2022] 约翰逊·渡边, 内特·库什曼, 拉马纳·库马尔, 弗朗西斯·宋, 诺亚·西格尔, 丽莎·王, 安东尼亚·克雷斯韦尔, 杰弗里·欧文, 伊琳娜·希金斯。基于过程和结果反馈解决数学文字问题。arXiv preprint arXiv:2211.14275, 2022。

[Vaswani等人, 2017] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, 和 Illia Polosukhin。注意力即所需。在 *Proceedings of Advances in Neural Information Processing Systems*, 第30卷, 2017年。

[Viterbi, 1967] 安德鲁·J·维特比。卷积码的错误界限和渐近最优解码算法。*IEEE Transactions on Information Theory*, 1967。

[冯·奥斯瓦尔德等人, 2023] 约翰内斯·冯·奥斯瓦尔德, 艾温德·尼克拉斯松, 埃托雷·兰达佐, 若昂·萨克拉门托, 亚历山大·莫尔金采夫, 安德烈·日莫金诺夫, 以及马克西姆·弗拉基米罗夫。通过梯度下降在上下文中学习的Transformer。在 *Proceedings of International Conference on Machine Learning*, 第35151–35174页。PMLR, 2023。

[王等, 2024] 王成龙, 周航, 胡一民, 胡宇夫, 李贝, 刘同然, 肖同, 朱静波。Esrl: 用于序列生成的基于采样的高效强化学习。在 *Proceedings of the AAAI Conference on Artificial Intelligence*, 第19107–19115页, 2024。

[王等, 2023] 王丽媛, 张星星, 苏航, 朱军。持续学习的全面调查: 理论、方法和应用。arXiv preprint arXiv:2302.00487, 2023a。

[王等, 2019] 王强, 李贝, 肖彤, 朱景波, 李长亮, 黄德瑞, 赵丽达 S. 学习深度变换器模型进行机器翻译。在 *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 第1810–1822页, 2019。

[王等, 2022] 王雪志, 魏杰, 舒尔曼斯, 吴国, 艾德·奇, 周登宇。语言模型中的理由增强集成。arXiv preprint arXiv:2207.00747, 2022a。

[王等, 2023] 王雪志, 魏杰, 舒尔曼斯, 吴国威·莱, 陈鸿志·蔡, 纳兰·沙兰, 乔阿坎什·乔杜里, 周登尼。自洽性提高语言模型中的思维链推理。在 *Proceedings of The Eleventh International Conference on Learning Representations*, 2023b。

[王等, 2022] 一中王, 斯瓦鲁普·米什拉, 佩加·阿里波尔莫拉巴希, 叶哈内赫·科尔迪, 阿米尔雷扎·米尔扎伊, 阿瑟瓦·纳伊克, 阿琼·阿什克, 阿鲁特·塞尔万·达纳谢卡拉恩, 安贾娜·阿伦库马尔, 大卫·斯塔普, 艾沙恩·帕塔克, 亚尼斯·卡拉曼诺拉基斯, 海志·加里·莱, 伊山·普罗希特, 伊沙尼·蒙达尔, 雅各布·安德森, 柯比·库兹尼亚, 克里玛·多西, 昆塔尔·库马尔·帕尔, 马特雷亚·帕特尔, 梅赫拉德·莫拉德沙希, 米希尔·帕马尔, 米尔拉利·普罗希特, 尼拉杰·瓦什尼, 帕尼·罗希塔·卡扎, 普尔基特·维尔马, 拉维谢哈·辛格·普里, 鲁尚·卡里亚, 萨万·多西, 沙伊拉贾·基尤尔·桑帕特, 西达尔塔·米什拉, 苏贾恩·雷迪A, 苏曼塔·帕特罗, 塔纳伊·迪希特, 和许东·申。超自然指令: 通过在1600+ nlp任务上的声明性指令进行泛化。在 *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, 第5085–5109页, 2022b。

[王等, 2023] 王一仲, 哈米什·伊维森, 普拉迪普·达西吉, 杰克·赫塞尔, 图沙尔·霍特, 奇亚蒂·拉加维·钱杜, 大卫·瓦登, 凯尔西·麦克米伦, 诺亚·A·史密斯, 伊兹·贝尔塔吉, 和汉纳内赫·哈吉什尔齐。骆驼能走多远? 探索在开放资源上的指令调整状态。 *Advances in Neural Information Processing Systems*, 36:74764–74786, 2023c。

[王等, 2023] 王一仲, 科迪·耶加内, 米什拉·斯瓦鲁普, 刘·阿莉萨, 诺亚·A·史密斯, 卡沙比·丹尼尔, 和哈吉什尔齐·汉纳内。自我指令: 将语言模型与自生成的指令对齐。在 *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 第13484–13508页, 2023d。

[王等, 2023] 王振一, 杨恩能, 沈丽, 黄恒。超越持续学习的深度学习中遗忘的综合调查。 *arXiv preprint arXiv:2307.09218*, 2023e。

[Warstadt等人, 2019] Alex Warstadt, Amanpreet Singh和Samuel R Bowman. 神经网络可接受性判断。 *Transactions of the Association for Computational Linguistics*, 7:625–641, 2019。

[Wei等人, 2022] Jason Wei, Maarten Bosma, Vincent Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M Dai, 以及Quoc V Le。微调的语言模型是零样本学习器。在 *Proceedings of International Conference on Learning Representations*, 2022a。

[魏等, 2022] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, 以及William Fedus。大型语言模型的涌现能力。 *arXiv preprint arXiv:2206.07682*, 2022b。

[魏等, 2022] Jason Wei, 王雪志, Dale Schuurmans, Maarten Bosma, Brian Ichter, 夏飞, Ed H. Chi, Quoc V. Le, 以及周登宇。思维链提示引发大型语言模型的推理。 *Advances in Neural Information Processing Systems*, 35:24824–24837, 2022c。

[Welleck等人, 2023] Sean Welleck, Ximing Lu, Peter West, Faeze Brahman, Tianxiao Shen, Daniel Khoshnab和Yejin Choi。通过学习自我纠正生成序列。在 *Proceedings of The Eleventh International Conference on Learning Representations*, 2023。

[温, 2021] 李莉安·温。如何在多个GPU上训练真正的大型模型? *lilianweng.github.io*, 2021年9月。URL <https://lilianweng.github.io/posts/2021-09-25-train-large/>。

[维纳, 1960] 诺伯特·维纳。自动化的一些道德和技术后果: 随着机器学习, 它们可能会以令程序员困惑的速度发展出不可预见的策略。 *Science*, 131(3410): 1355–1358, 1960。

[威廉姆斯等人, 2018] Adina Williams, Nikita Nangia和Samuel Bowman。通过推理进行句子理解的广泛覆盖挑战语料库。在 *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, 第1112–1122页, 2018。

[威廉姆斯, 1992] 罗纳德·J·威廉姆斯。用于连接主义强化学习的简单统计梯度跟踪算法。 *Machine learning*, 8:229–256, 1992。

[Wingate等人, 2022] David Wingate, Mohammad Shoeybi和Taylor Sorensen。语言模型中的提示压缩和对比性条件化以实现可控性和毒性降低。在 *Findings of the Association for Computational Linguistics: EMNLP 2022*, 第5621–5634页, 2022。

[沃森格拉夫和里芬, 1961] 约翰·M·沃森格拉夫和巴尼·里芬。 *Sequential Decoding*。麻省理工学院出版社, 1961年。

[吴等, 2023] 邓阳吴, 钟银敏, 张子力, 刘胜宇, 刘方月, 孙元航, 黄刚, 刘璇哲, 金欣。大型语言模型的快速分布式推理服务。 *arXiv preprint arXiv:2305.05920*, 2023a。

[吴等人, 2024] 威尔逊·吴, 约翰·X·莫里斯, 和利昂内尔·莱文。语言模型是否为未来的标记做计划? *arXiv preprint arXiv:2404.00859*, 2024。

[吴等人, 2021] 吴宇辉, 马克斯·诺曼·拉贝, 德莱塞利·哈钦斯, 以及克里斯蒂安·塞格迪。记忆变压器。在 *Proceedings of International Conference on Learning Representations*, 2021。

[吴等人, 2023] 赵秋 吴, 胡玉诗, 石伟佳, 迪里·努哈, 苏拉内·阿拉内, 阿姆纳布罗卢·普里特维拉贾, 诺亚·A·史密斯, 玛丽·奥斯滕多夫, 哈纳内·哈吉什尔齐。细粒度人类反馈为语言模型训练提供更好的奖励。在 *Thirty-seventh Conference on Neural Information Processing Systems*, 2023b。

[Xia 等人, 2024] 孟州 Xia, Sathika Malladi, Suchin Gururangan, Sanjeev Arora 和 Danqi Chen。Less: 选择有影响力的数据进行针对性指令调整。 *arXiv preprint arXiv:2402.04333*, 2024。

[小等, 2024] 广选小, 田远东, 陈贝迪, 韩松, 和 Mike Lewis。带注意力汇聚的流式语言模型。在 *Proceedings of The Twelfth International Conference on Learning Representations*, 2024。

[肖晓和朱, 2023] 肖晓和朱静波。变换器导论: 自然语言处理视角。 *arXiv preprint arXiv:2311.17633*, 2023。

[小等, 2013] 小彤, 朱静波, 刘同然。集成和提升统计机器翻译系统。 *Artificial Intelligence*, 195: 496–527, 2013。

[Xiao 等人, 2019] Tong Xiao, Yinqiao Li, Jingbo Zhu, Zhengtao Yu, 和 Tongran Liu。为快速Transformer共享注意力权重。在 *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI-19)*, 第5292–5298页, 2019。

[谢等, 2022] Sang Michael Xie, Aditi Raghunathan, Percy Liang, 和 Tengyu Ma。情境学习的解释作为隐式贝叶斯推理。在 *Proceedings of International Conference on Learning Representations*, 2022。

[Xin 等人, 2020] Ji Xin, Raphael Tang, Jaeeun Lee, Yaoliang Yu 和 Jimmy Lin。Deebert: 用于加速BERT推理的动态早期退出。在 *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 第2246–2251页, 2020。

[Xu 等人, 2024] 徐, 孙庆锋, 郑凯, 耿锡波, 赵普, 冯佳湛, 陶长阳, 林清伟, 姜大新。Wizardlm: 赋能大型预训练语言模型遵循复杂指令。在 *The Twelfth International Conference on Learning Representations*, 2024。

[杨等, 2024] 杨安, 杨宝松, 张贝臣, 惠斌元, 郑波, 余博文, 李成远, 刘大恒, 黄飞, 魏浩然, 等。Qwen2.5 技术报告。 *arXiv preprint arXiv:2412.15115*, 2024。

[杨等, 2019] 邱吉尔·杨, 戴子航, 杨一鸣, 贾迈·卡本内尔, 拉苏尔·萨拉克哈图丁诺夫, 和国克·V·莱。Xlnet: 用于语言理解的广义自回归预训练。 *Advances in neural information processing systems*, 32, 2019。

[姚等, 2024] 韵宇 姚丹, 余迪安, 赵杰夫, 伊扎克·沙弗兰, 汤姆·格里菲斯, 曹远, 和纳拉辛汉·卡蒂克。思维树: 使用大型语言模型进行深思熟虑的问题解决。 *Advances in Neural Information Processing Systems*, 36, 2024。

[Yarowsky, 1995] David Yarowsky。与监督方法相媲美的无监督词义消歧。在 *Proceedings of the 33rd annual meeting of the association for computational linguistics*, 第189–196页, 1995年。

[Yu 等人, 2022] Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, 和 Byung-Gon Chun。Orca: 一种用于{基于Transformer的}生成模型的分布式服务系统。在 *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, 第521–538页, 2022。

[Yu 等人, 2023] Zihan Yu, Liang He, Zhen Wu, Xinyu Dai, 和 Jiajun Chen。朝着更好的思维链提示策略: 一项调查。 *arXiv preprint arXiv:2310.04959*, 2023。

[Zaheer 等人, 2020] Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, C. Alberti, S. Ontañón, Philip Pham, Anirudh Ravula, Qifan Wang, L. Yang, 和 A. Ahmed。大鸟: 用于长序列的Transformer。 *Advances in neural information processing systems*, 33:17283–17297, 2020。

[Zellers等人, 2018] Rowan Zellers, Yonatan Bisk, Roy Schwartz和Yejin Choi. Swag: 一个用于基于常识推理的大规模对抗数据集。在*Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 第93-104页, 2018年。

[张和森尼克, 2019] 张表和里科·森尼克。均方根层归一化。 *Advances in Neural Information Processing Systems*, 32, 2019。

[张等, 2024] 张云翔, 穆罕默德·卡利法, 拉贾努根·洛格斯瓦兰, 杰基姆·金, 李梦泰, 李洪拉克, 和王路。小型语言模型需要强大的验证器来自我纠正推理。在 *ACL (Findings)*, 2024。

[张等, 2023] 张铸生, 姚瑶, 张阿斯顿, 唐翔宇, 马新北, 何志伟, 王一鸣, 马克·盖斯廷, 王瑞, 刘功深, 和赵海。点燃语言智能: 从思维链推理到语言代理的搭便车指南。 *arXiv preprint arXiv:2311.11797*, 2023a。

[张等, 2023] 张卓胜, 张骏, 李穆, 和亚历克斯·斯莫拉。在大语言模型中的自动思维链提示。在 *The Eleventh International Conference on Learning Representations*, 2023b。

[赵等, 2024] 赵浩, 安德留申科, 克罗切, 和弗拉马尔松。长文本更有利于对齐: 指令微调的一个简单但难以击败的基线。 *arXiv preprint arXiv:2402.04833*, 2024。

[赵等, 2023] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Z. Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jianyun Nie, 以及Ji rong Wen。大型语言模型的调查。 *arXiv preprint arXiv:2303.18223*, 2023。

[钟等, 2024] 钟银民, 刘胜宇, 陈俊达, 胡建波, 朱一博, 刘璇哲, 金欣, 张浩。{DistServe}: 为高吞吐量优化的大型语言模型服务中的预填充分解和解码。在 *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, 第 193–210 页, 2024。

[周等, 2023] Chunting Zhou, Pengfei Liu, Puxin Xu, Srini Iyer, Jiao Sun, Yuning Mao, Xuezhe Ma, Avia Efrat, Ping Yu, Lili Yu, Susan Zhang, Gargi Ghosh, Mike Lewis, Luke Zettlemoyer, 和Omer Levy. Lima: 少即是多对齐。 *arXiv preprint arXiv:2305.11206*, 2023a。

[周等, 2023] Denny Zhou, Nathanael Schärli, 侯磊, 魏杰, Nathan Scales, 王雪志, Dale Schuurmans, 崔克莱尔, Olivier Bousquet, Quoc V. Le, 以及Ed H. Chi。从少到多提示使大型语言模型能够进行复杂推理。在 *Proceedings of The Eleventh International Conference on Learning Representations*, 2023b。

[周等, 2020] 王春树 周可文 徐, 陶戈, 朱利安 麦卡利, 徐克, 魏福如。伯特失去耐心: 快速且鲁棒的早期退出推理。 *Advances in Neural Information Processing Systems*, 33:18330–18341, 2020。

[周等, 2023] 周永超, 穆雷萨努安德烈, 韩子文, 凯里安·帕斯特, 皮蒂斯西尔维乌, 陈哈里斯, 以及巴吉米。大型语言模型是人类级别的提示工程师。在 *The Eleventh International Conference on Learning Representations*, 2023c。

[Zoph和Le, 2016] Barret Zoph和Quoc Le. 基于强化学习的神经架构搜索。在 *Proceedings of International Conference on Learning Representations*, 2016。

[Zoph等人, 2020] Barret Zoph, Golnaz Ghiasi, Tsung-Yi Lin, Yin Cui, Hanxiao Liu, Ekin Dogus Cubuk, 和Quoc Le。重新思考预训练和自训练。 *Advances in neural information processing systems*, 33:3833–3845, 2020。

索引

k -NN, 76 k -NN语言模型
, 76 k -NN语言建模, 76 k -
最近邻, 76

A2C, 178动作值函数, 174优
势, 178优势演员-评论家, 178
代理, 47 ALiBi, 85对齐, 46
具有线性偏置的注意力, 85自
动化机器学习, 139自动提示
设计, 139 AutoML, 139自主
代理, 137

BART, 19 BERT, 1 最佳
- N 采样, 200 BoN采样
, 200 布拉德利-特里模
型, 181

计算注释, 115灾难性遗忘, 35
因果语言建模, 9思维链, 115
思维链提示, 53完成, 6组合泛
化, 124连续批处理, 226CoT
, 115COT提示, 53跨语言语言
模型, 28累积奖励, 175

解码, 209 故意-然后生成, 128
演示, 6 直接偏好优化, 193 文
档旋转, 20 DPO, 193 DTG, 12
8

突现能力, 64个外部
记忆, 74

外推法, 82

少量样本 COT 提示, 54

门控线性单元, 59 高斯误
差线性单元, 59 GeLU, 5
9 GLU, 59 GPT, 1 GQA
, 80 分组查询注意力, 80

硬提示, 142人类偏好对齐, 15
5

ICL, 53 ICT, 6重要性采样
, 183上下文学习, 6, 53, 9
6推理引擎, 225推理时间缩
放, 234输入反转, 166指令
对齐, 155指令微调, 43, 15
7干扰, 29内部存储器, 74插
值, 82不可约误差, 65基于
迭代的调度, 226

键值缓存, 68, 207 K
V缓存, 68, 207

标签映射, 106 从人类反馈中学
习, 47 最少到最多提示, 121 长
上下文LLM, 66

掩码语言建模, 1, 9 mBERT
, 28 基于记忆的方法, 74 M
QA, 80 多语言 BERT, 28 多
查询注意力, 80

NAS, 139 神经架构搜索, 1
39 下一个句子预测, 12

NSP, 12核采样, 21
6

离线强化学习, 196次单次COT
提示, 54个ORM, 243个结果奖励模型, 243个基于结果的方法, 198个过度优化问题, 192

并行缩放, 240 性能估计, 139 性能函数, 176 性能差距恢复, 170 乱序语言模型, 11 PGR, 170 Plackett-Luce 模型, 187 PPO, 51, 184 预填充, 207 前缀微调, 146 前缀语言模型, 16 PRM, 243 问题分解, 117 流程奖励模型, 243 基于流程的方法, 198 提示嵌入, 150 提示工程, 96 提示优化, 139 提示搜索空间, 139 提示工程, 51 近端策略优化, 51, 184

Q值函数, 175

RAG, 77比例函数, 183修正线性单元, 59 基于人类反馈的强化学习, 47, 156拒绝采样, 201关系抽取, 109ReLU, 59请求级调度, 225检索增强生成, 77返回, 175奖励游戏, 192奖励黑客, 192奖励模型, 48

RLHF, 47, 156
RoBERTa, 27

样例高效, 167缩放定律, 63调度器, 225自治, 132自我指导, 163 自监督学习, 3自训练, 3句子重排, 20序列编码模型, 3序列生成模型, 4顺序缩放, 240SFT, 47, 155单轮预测, 158软提示, 142跨度掩码, 19投机解码, 218投机执行, 218状态值函数, 174强上限性能, 170子问题生成, 120子问题解决, 120表面对齐假设, 167监督微调, 47监督微调, 155监督学习, 2代理目标, 183T5, 15TD, 179时间差分, 179文本补全, 110文本转换, 110标记删除, 19标记掩码, 19 Transformer, 1翻译语言建模, 29 信任域, 184

无监督学习, 2

弱性能, 170 弱到强泛化, 169
弱到强性能, 170

XLMS, 28

零样本COT, 54零
样本学习, 45