

## 2 大语言模型架构

随着数据资源和计算能力的爆发式增长，语言模型的参数规模和性能表现实现了质的飞跃，迈入了大语言模型 (Large Language Model, LLM) 的新时代。凭借着**庞大的参数量**和**丰富的训练数据**，大语言模型不仅展现出了强大的泛化能力，还催生了新智能的涌现，勇立生成式人工智能 (Artificial Intelligence Generated Content, AIGC) 的浪潮之巅。当前，大语言模型技术蓬勃发展，各类模型层出不穷。这些模型在广泛的应用场景中已经展现出与人类比肩甚至超过人类的能力，引领着由 AIGC 驱动的新一轮产业革命。本章将深入探讨大语言模型的相关背景知识，并分别介绍 Encoder-only、Encoder-Decoder 以及 Decoder-only 三种主流模型架构。通过列举每种架构的代表性模型，深入分析它们在网络结构、训练方法等方面的主要创新之处。最后，本章还将简单介绍一些非 Transformer 架构的模型，以展现当前大语言模型研究百花齐放的发展现状。

\* 本书持续更新，GIT Hub 链接为：<https://github.com/ZJU-LLMs/Foundations-of-LLMs>。

## 2.1 大数据 + 大模型 → 新智能

在自然语言处理的前沿领域，大语言模型正以其庞大的模型规模、海量数据的吞吐能力和卓越的模型性能，推动着一场技术革新的浪潮。当我们谈论“大语言模型”之大时，所指的不仅仅是**模型规模的庞大**，也涵盖了**训练数据规模的庞大**，以及由此衍生出的**模型能力的强大**。这些模型如同探索未知领域的巨轮，不仅在已有的技术上不断突破性能的极限，更在新能力的探索中展现出惊人的潜力。

截止 2024 年 6 月，国内外已经见证了超过百种大语言模型的诞生，这些大语言模型在学术界和工业界均产生了深远的影响。图 2.1 展示了其中一些具有重要影响力的模型。

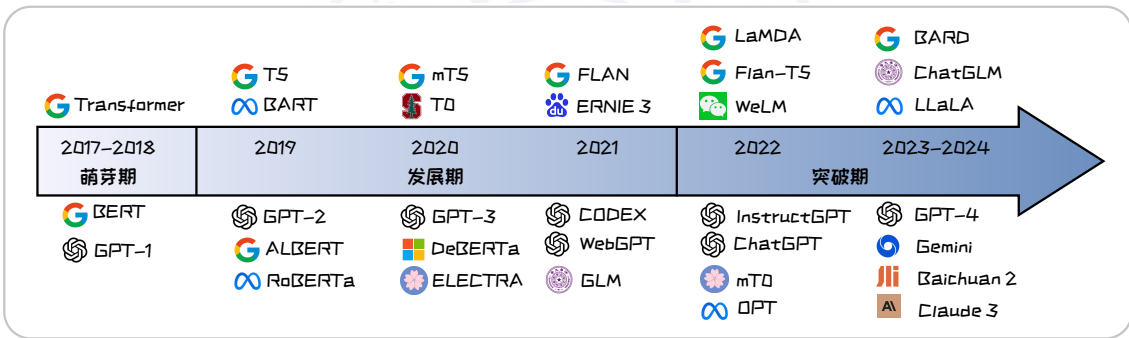


图 2.1: 大语言模型涌现能力的三个阶段。

大语言模型的发展历程可以大致划分为三个阶段。2017 至 2018 年是基础模型的**萌芽期**，以 Transformer 架构的诞生和 BERT[11]、GPT-1[27] 模型的问世为标志，开启了预训练语言模型的新纪元。2019 至 2022 年是大语言模型的**发展期**，通过 GPT-2<sup>1</sup>、T5[29] 以及 GPT-3[5] 等模型在参数规模以及能力上的大幅提升，研究者开始深入探索大语言模型的潜力。2022 年起则是大语言模型的**突破期**，ChatGPT<sup>2</sup> 以及 GPT-4<sup>3</sup> 等模型的发布标志着大语言模型相关技术的显著进步。同时，各大公司

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

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

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

和研究机构也纷纷推出了自己的模型，例如百川智能的百川大模型 [44]，百度的文心一言等，推动了大语言模型的快速发展。

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

### 2.1.1 大数据 + 大模型 → 能力增强

在数字化浪潮的推动下，数据如同汇聚的洪流，而模型则如同乘风破浪的巨舰。数据规模的增长为模型提供了更丰富的信息源，意味着模型可以学习到更多样化的语言模式和深层次的语义关系。而模型规模的不断扩大，极大地增加了模型的表达能力，使其能够捕捉到更加细微的语言特征和复杂的语言结构。在如此庞大的模型参数规模以及多样化的训练数据共同作用下，模型内在对数据分布的拟合能力不断提升，从而在复杂多变的数据环境中表现出更高的适应性和有效性 [7]。

然而模型规模和数据规模的增长并非没有代价，它们带来了更高的计算成本和存储需求，这要求我们在模型设计时必须在资源消耗和性能提升之间找到一个恰当的平衡点。为了应对这一挑战，大语言模型的扩展法则（Scaling Laws）应运而生。这些法则揭示了模型的能力随模型和数据规模的变化关系，为大语言模型的设计和 optimization 提供了宝贵的指导和参考。本章节将深入介绍两种扩展法则：OpenAI 提出的 Kaplan-McCandlish 扩展法则以及 DeepMind 提出的 Chinchilla 扩展法则。

#### 1. Kaplan-McCandlish 扩展法则

2020 年，OpenAI 团队的 Jared Kaplan 和 Sam McCandlish 等人 [16] 首次探究了神经网络的性能与数据规模  $D$  以及模型规模  $N$  之间的函数关系。他们在不同规模的数据集（从 2200 万到 230 亿个 Token）和不同规模的模型下（从 768 到 15 亿个参数）进行实验，并根据实验结果拟合出了两个基本公式：

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

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

这里的  $L(N)$  表示在数据规模固定时，**不同模型规模**下的交叉熵损失函数，反映了模型规模对拟合数据能力的影响。相应地， $L(D)$  表示在模型规模固定时，**不同数据规模**下的交叉熵损失函数，揭示了数据量对模型学习的影响。 $L$  的值衡量了模型拟合数据分布的准确性，值越小表明模型对数据分布的拟合越精确，其**自身学习能力**也就越强大。

实验结果和相关公式表明，模型的性能与模型模型以及数据规模这两个因素均高度正相关。然而，在模型规模相同的情况下，模型的**具体架构**对其性能的影响相对较小。因此，扩大模型规模和丰富数据集成为了提升大型模型性能的两个关键策略。

此外，OpenAI 在进一步研究计算预算的最优分配时发现，总计算量  $C$  与数据量  $D$  和模型规模  $N$  的乘积近似成正比，即  $C \approx 6ND$ 。在这一条件下，如果计算预算增加，为了达到最优模型性能，数据集的规模  $D$  以及模型规模  $N$  都应同步增加。但是**模型规模的增长速度应该略快于数据规模的增长速度**。具体而言，两者的最优配置比例应当为  $N_{opt} \propto C^{0.73}, D_{opt} \propto C^{0.27}$ 。这意味着，如果总计算预算增加了 10 倍，模型规模应扩大约 5.37 倍，而数据规模应扩大约 1.86 倍，以实现模型的最佳性能。

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

## 2. Chinchilla 扩展法则

谷歌旗下 DeepMind 团队对“模型规模的增长速度应该略高于数据规模的增长速度”这一观点提出了不同的看法。在 2022 年，他们对更大范围的模型规模（从 7000 万到 1600 亿个参数）以及数据规模（从 50 亿到 5000 亿个 Token）进行了深入的实验研究，并据此提出了 Chinchilla 扩展法则 [15]：

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

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

DeepMind 同样探索了计算预算的最优分配问题，最终得出数据集规模  $D$  与模型规模  $N$  的最优配置为  $N_{opt} \propto C^{0.46}, D_{opt} \propto C^{0.54}$ 。这一结果表明，**数据集量  $D$  与模型规模  $N$  几乎同等重要**，如果总计算预算增加了 10 倍，那么模型规模以及数据规模都应当扩大大约 3.16 倍。谷歌后续在 2023 年 5 月发布的 PaLM 2 的技术报告 [2] 中也再次证实了这一观点，进一步强调了数据规模在提升模型性能中的重要性。

此外，Chinchilla 扩展法则进一步提出，理想的数据集大小应当是模型规模的 20 倍。例如，对于一个 7B（70 亿参数）的模型，最理想的训练数据集大小应为 140B（1400 亿）个 Token。但先前很多模型的预训练数据量并不够，例如 OpenAI 的 GPT-3[5] 模型的最大版本有 1750 亿参数，却只用了 3000 亿 Token 进行训练；同样，微软的 MT-NLG[35] 模型拥有 5300 亿参数，而训练用的 Token 数量却只有 2700 亿。因此，DeepMind 推出了数据规模 20 倍于模型规模的 Chinchilla 模型（700 亿参数，1.4 万亿 Token），最终在性能上取得了显著突破。

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



2.1.2 大数据 + 大模型 → 能力扩展

如图2.2所示，模型训练数据规模以及参数数量的不断提升，不仅带来了上述学习能力的稳步增强，还为大模型“解锁”了一系列**新的能力**<sup>4</sup>，例如上下文学习能力、常识推理能力、数学运算能力、代码生成能力等。值得注意的是，这些新能力并非通过在特定下游任务上通过训练获得，而是随着模型复杂度的提升凭空自然涌现<sup>5</sup>。这些能力因此被称为大语言模型的涌现能力（Emergent Abilities）。



图 2.2: 大语言模型能力随模型规模涌现，图片由 GPT-4o 生成。

涌现能力往往具有突变性和不可预见性。类似于非线性系统中的“相变”，即系统在某个阈值点发生显著变化，这些能力也并没有一个平滑的、逐渐积累的过程，而是在模型达到一定规模和复杂度后，很突然地显现 [32]。例如，在 GPT 系列的演变中，可以观察到一些较为典型的涌现能力。

- **上下文学习：**上下文学习（In-Context Learning）是指大语言模型在推理过程中，能够利用输入文本的上下文信息来执行特定任务的能力。具备了上下文学习能力的模型，在很多任务中**无需额外的训练**，仅**通过示例或提示**即可理解任务要求并生成恰当的输出。在 GPT 系列中，不同版本的模型在上下文学习能力上有显著差异。早期的 GPT-1 和 GPT-2 在上下文学习方面的能力非常

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

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

有限，通常无法直接利用上下文信息进行准确的推理和回答。GPT-3 的 130 亿参数版本则在上下文学习方面取得了显著进步，能在提供的上下文提示下完成一些常见任务。然而，对于更加复杂或特定领域的任务，其性能仍有限。具有 1750 亿参数的 GPT-3 最大版本以及后续的 GPT-4 模型展现出强大的上下文理解和学习能力，可以基于少量示例完成各类高度复杂的任务。

- **常识推理：**常识推理（Commonsense Reasoning）能力赋予了大语言模型**基于常识知识和逻辑进行理解和推断**的能力。它包括对日常生活中普遍接受的事实、事件和行为模式的理解，并利用这些知识来回答问题、解决问题和生成相关内容。GPT-1 和 GPT-2 在常识推理方面的能力非常有限，常常会出现错误的推断或缺乏详细的解释。而 GPT-3 的较大版本能够在大多数情况下生成合理和连贯的常识性回答。至于具有 1750 亿参数的 GPT-3 最大版本以及后续的 GPT-4 等模型，则能够在处理高度复杂的常识推理任务时展现逻辑性、一致性和细节丰富性。
- **代码生成：**代码生成（Code Generation）能力允许大语言模型**基于自然语言描述自动生成编程代码**。这包括理解编程语言的语法和语义、解析用户需求、生成相应代码，以及在某些情况下进行代码优化和错误修复。GPT-1 和 GPT-2 仅能生成非常简单的代码片段，但是无法有效理解具体的编程需求。130 亿参数的 GPT-3 模型出现时，已经能很好地处理常见的编程任务和生成结构化代码片段，但在极其复杂或特定领域的任务上仍有限。在参数量达到 1750 亿时，模型则能够处理复杂编程任务，多语言代码生成，代码优化和错误修复等，展示出高质量的代码生成和理解能力。
- **逻辑推理：**逻辑推理（Logical Reasoning）能力使大语言模型能够基于给定信息和规则进行**合乎逻辑的推断和结论**。这包括简单的条件推理、多步逻辑推理、以及在复杂情境下保持逻辑一致性。GPT-1 和 GPT-2 作为早期的生成

预训练模型，在逻辑推理方面的能力非常有限，甚至对于 130 亿参数版本的 GPT-3 模型而言，虽然能处理一部分逻辑推理任务，但在复杂度和精确性上仍存在一定局限性。直到 1750 亿参数版本，GPT-3 才能够处理复杂的逻辑推理任务，生成详细和连贯的推理过程。

- .....

这些涌现能力使得大语言模型可以在不进行专项训练的前提下完成各类任务，但同时也带来了诸多挑战，包括模型的可解释性、信息安全与隐私、伦理和公平性问题，以及对计算资源的巨大需求等。解决这些挑战需要在技术、法律和社会层面进行综合考量，以确保大语言模型的健康发展和可持续进步。

## 2.2 大语言模型架构概览

在语言模型的发展历程中，Transformer[42] 框架的问世代表着一个划时代的转折点。其独特的自注意力（Self-Attention）机制极大地提升了模型**对序列数据的处理能力**，在**捕捉长距离依赖关系**方面表现尤为出色。此外，Transformer 框架对**并行计算的支持**极大地加速了模型的训练过程。当前，绝大多数大语言模型均以 Transformer 框架为核心，并进一步演化出了三种经典架构，分别是 Encoder-only 架构，Decoder-only 架构以及 Encoder-Decoder 架构。这三种架构在设计和功能上各有不同。本节将简要介绍这三种架构的设计理念与预训练方式，并分析它们之间的区别以及各自的演变趋势。

### 2.2.1 主流模型架构的类别

本小节将从设计理念、训练方式等角度对 Encoder-only 架构，Decoder-only 架构以及 Encoder-Decoder 架构分别进行简要介绍。



## 1. Encoder-only 架构

Encoder-only 架构仅选取了 Transformer 中的编码器（Encoder）部分，用于接收输入文本并生成与上下文相关的特征。具体来说，Encoder-only 架构包含三个部分，分别是**输入编码**部分，**特征编码**部分以及**任务处理**部分，具体的模型结构如图2.3所示。其中输入编码部分包含**分词**、**向量化**以及**添加位置编码**三个过程。而特征编码部分则是由多个相同的**编码模块**（Encoder Block）堆叠而成，其中每个编码模块包含**自注意力模块**（Self-Attention）和**全连接前馈模块**。任务处理模块是针对任务需求专门设计的模块，其可以由用户针对任务需求自行设计。Encoder-only 架构模型的预训练阶段和推理阶段在输入编码和特征编码部分是一致的，而任务处理部分则需根据任务的不同特性来进行定制化的设计。

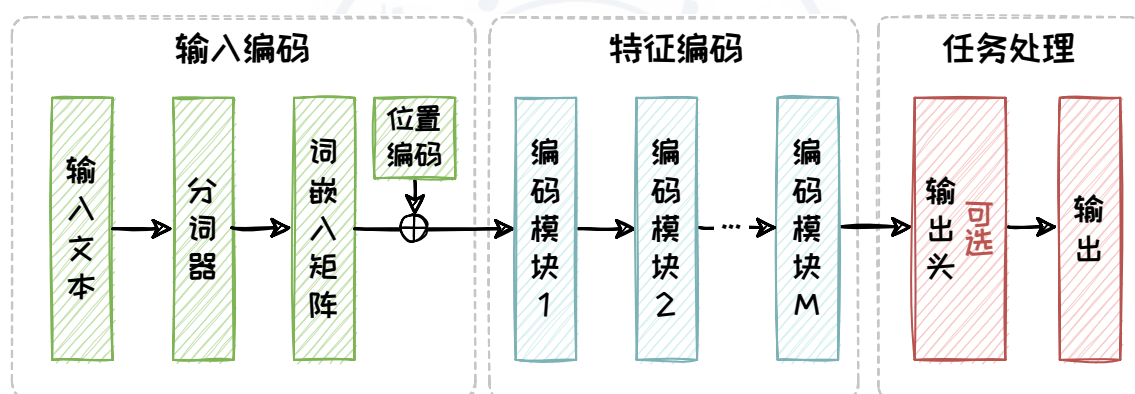


图 2.3: Encoder-only 架构。

在**输入编码**部分，原始输入文本会被分词器（Tokenizer）拆解为 Token 序列，随后通过词表和词嵌入（Embedding）矩阵映射为向量序列，确保文本信息得以数字化表达。具体的过程和细节将在3.1 章节中被具体介绍。接着为了保留文本中单词的顺序信息，每个向量序列会被赋予位置编码（Positional Encoding）。在**特征编码**部分，先前得到的向量序列会依次通过一系列编码模块，这些模块通过自注意力机制和前馈网络进一步提取和深化文本特征。**任务处理**部分在预训练阶段和下游任务适配阶段一般有所差别。在预训练阶段，模型通常使用全连接层作为输出头，

用于完成掩码预测等任务。而在下游任务适配阶段，输出头会根据具体任务需求进行定制。例如，对于情感分析或主题分类等判别任务，只需要添加一个分类器便可直接输出判别结果。但对于文本摘要生成等生成任务，则需要添加一个全连接层，逐个预测后续的 Token。但以这种形式来完成生成任务存在着诸多的限制，例如在每次生成新的 Token 时，都需要重新计算整个输入序列的表示，这增加了计算成本，也可能导致生成的文本缺乏连贯性。本章将在 2.3 节中对 Encoder-only 架构进行更具体的介绍。

## 2. Encoder-Decoder 架构

为了弥补 Encoder-only 架构在文本生成任务上的短板，Encoder-Decoder 架构在其基础上引入了一个解码器 (Decoder)，并采用交叉注意力机制来实现编码器与解码器之间的有效交互。

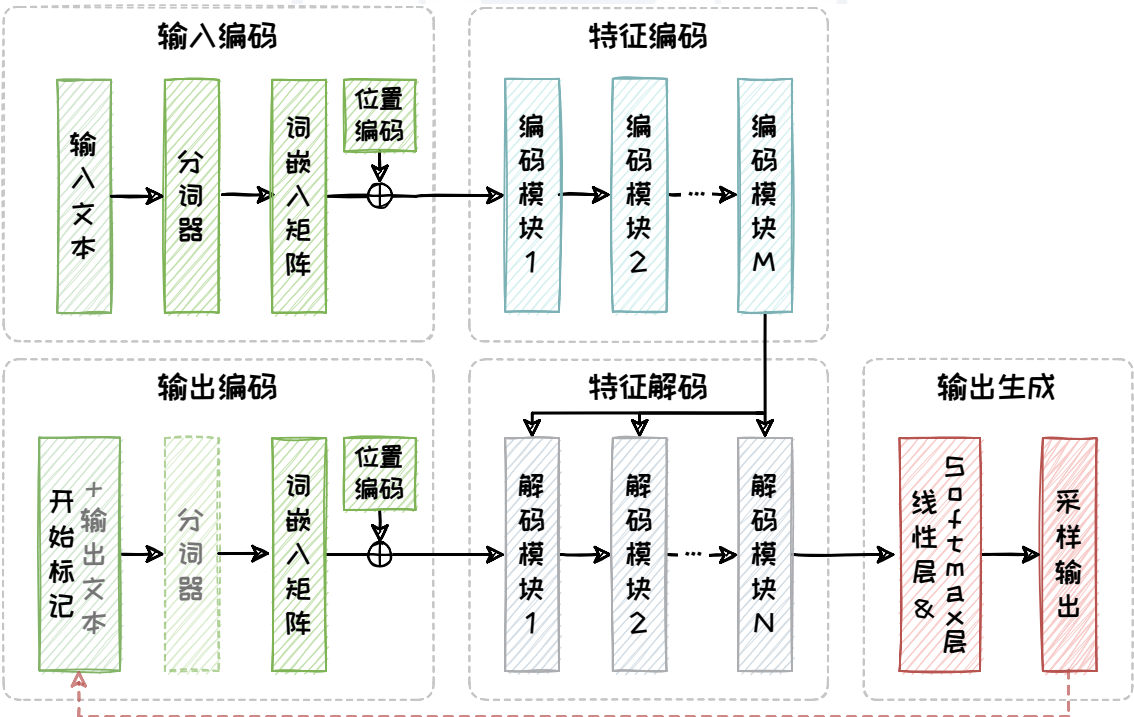


图 2.4: Encoder-Decoder 架构。其中分词器和输出文本只在训练阶段存在，而实现“自回归”的红色虚线只在推理阶段存在。

具体来说，解码器包含了**输出编码**、**特征解码**以及**输出生成**三个部分。其中**输出编码**与编码器中的输入编码结构相同，包含分词、向量化以及添加位置编码三个过程，将原始输入文本转换化为带有位置信息的向量序列。此外，**特征解码**部分与特征编码部分在网络结构上也高度相似，包括掩码自注意力（Masked Self-Attention）模块，交叉注意力模块和全连接前馈模块。其中掩码自注意力模块确保模型只关注上文，不会“预见”未来的信息，从而可以在无“下文泄露”的条件下，进行“自回归”的训练和推理。而交叉注意力模块则负责处理从编码模块向解码模块传递相关信息。**输出生成**部分则由一个线性层以及一个 Softmax 层组成，负责将特征解码后的向量转换为词表上的概率分布，并从这个分布中采样得到最合适的 Token 作为输出。

图2.4展示了 Encoder-Decoder 架构的具体工作流程。在训练阶段，样本中同时包含了输入和真实（Ground Truth）输出文本。其中输入文本首先被输入编码部分转化为向量序列，接着在特征编码模块中被多个堆叠起来的编码模块进一步处理，从而被转化为上下文表示。而输出文本之前会被添加特殊的开始标记 [START]，然后在输出编码部分被分词、词嵌入和位置编码处理后，并行输入到特征解码模块中。接着解码模块使用 Teacher Forcing 技术，在每轮预测时，使用真实输出文本中的已知部分作为输入，并结合从最后一个编码块得到的上下文信息，来预测下一个 Token，计算预测的 Token 和真实 Token 之间的损失，通过反向传播更新模型参数。

在推理阶段，由于缺少了真实的输出文本，所以输出序列原始状态只有开始标记 [START]，也不再需要分词器。模型需要通过自回归的方式，在每轮采样生成 Token 后，会将其拼接到输出序列中，用于下一轮预测。这个过程循环进行，直到生成特定的结束标记 [end] 或达到模型设定的最大输出长度。在这一过程中，由于每轮的输入依赖于上一轮的采样结果，因此只能一步步地串行输出。在2.4节中会针对 Encoder-Decoder 架构进行更具体的介绍。

### 3. Decoder-only 架构

为了有效缩减模型的规模以及降低整体的计算复杂度，Decoder-only 架构摒弃了 Encoder-Decoder 架构中的编码器部分以及与编码器交互的交叉注意力模块。在这种架构下，模型仅使用解码器来构建语言模型。这种架构利用“自回归”机制，在给定上文的情况下，生成流畅且连贯的下文。

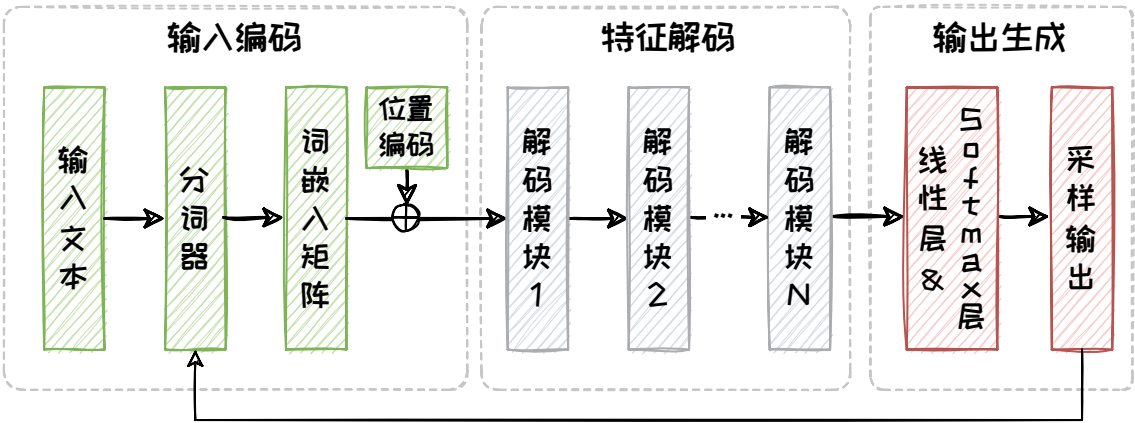


图 2.5: Decoder-only 架构。

Decoder-only 架构同样包含了三个部分，分别是**输入编码**部分、**特征解码**部分以及**输出生成**部分，其具体的模型结构如图2.5所示。Decoder-only 架构的核心特点在于省略了每个编码模块中的交叉注意力子模块，这也是其与传统 Encoder-Decoder 架构中解码器部分的主要区别。在2.5 节中将会对 Decoder-only 架构进行更具体的介绍。

#### 2.2.2 模型架构的功能对比

上述的 Encoder-only、Encoder-Decoder 和 Decoder-only 这三种模型架构虽然都源自于 Transformer 框架，但他们在注意力矩阵上有着显著区别，这也造就了他们在功能以及最终适用任务上的不同。接下来将针对注意力矩阵以及适用任务两个方面对这三种架构的主要区别进行分析。



## 1. 注意力矩阵

注意力矩阵 (Attention Matrix) 是 Transformer 中的核心组件，用于计算输入序列中各个 Token 之间的依赖关系。通过注意力机制，模型可以在处理当前 Token 时，灵活地关注序列中其他 Token 所携带的信息，决定了在这一过程中哪些 Token 能够相互影响。如图2.6所示，三种架构在注意力矩阵上有着显著差异。

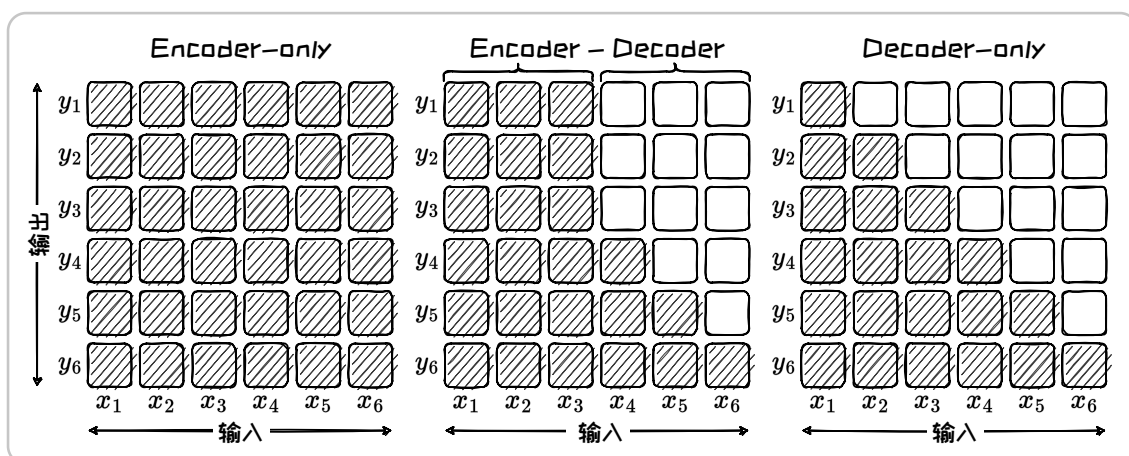


图 2.6: 三种架构的注意力矩阵。

**Encoder-only** 架构中的注意力矩阵来自于自注意力模块，用于捕捉输入序列中各个 Token 之间的关系。Encoder-only 架构的注意力矩阵呈现出“完全”的注意力，即对于每个 Token 的理解都依赖于整个输入序列中的所有 Token。例如，在将输入 Token  $x_i$  转换为上下文向量  $y_i$  的过程中，模型能够综合利用从  $x_1 \sim x_n$  的所有输入信息，这就是所谓的**双向注意力机制**。在这种双向注意力机制的作用下，模型能够同时利用前后文信息，深入理解复杂的语义联系和上下文依赖。

**Encoder-Decoder** 架构中的注意力矩阵较为复杂，它结合了编码器的自注意力、解码器的掩码自注意力以及交叉注意力三种机制。编码器的自注意力矩阵与 Encoder-only 架构类似，用于生成输入序列的全面上下文表示，呈现“完全”的注意力。而解码器的掩码自注意力矩阵则呈现出“下三角”的注意力，确保在生成当前 Token 时，模型只关注之前生成的 Token。此外，交叉注意力机制允许解码器始



终能够动态地参考编码器生成的完整上下文表示，确保输出与输入序列高度相关且连贯。例如，在编码器将输入  $x_i$  转化为上下文向量时，可以利用从  $x_1 \sim x_n$  的所有输入信息；当解码器在生成 Token  $y_i$  的时候，可以参考由  $x_1 \sim x_n$  转化得到的上下文向量以及先前生成的 Token 序列  $y_1 \sim y_{i-1}$  的相关信息。

**Decoder-only 架构**中的注意力矩阵来自于掩码自注意力模块，其特点是呈现出“下三角”的注意力模式。这意味着在预测当前 Token 时，模型只能依赖于已经生成的历史 Token 信息，体现了**单向注意力机制**。例如，在生成 Token  $y_i$  的时候，模型只能考虑先前  $y_1 \sim y_{i-1}$  的信息，这样的设计确保了生成过程的顺序性和文本的连贯性。

### 2. 适用任务

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

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

**Encoder-Decode 架构**在 Encoder-only 架构的基础上添加了解码器，使模型能够基于编码器输出的上下文表示逐步生成输出序列。这种编码器和解码器的结合，使得模型可以有效地处理复杂的输入条件，并生成相关且连贯的高质量内容。因此，Encoder-Decoder 架构的模型非常适合于处理各种复杂的**有条件生成任务**，例如机器翻译、文本摘要和问答系统等需要同时理解输入并生成相应输出的场景。但新添加解码器也同样带来了模型规模以及计算量庞大的问题。

**Decoder-only 架构**进一步删除了 Encoder-Decoder 架构中的编码器部分，从而降低了模型本身的计算复杂度。这一架构的模型使用掩码（Mask）操作确保在每个时间步生成当前 Token 时只能访问先前的 Token，并通过自回归生成机制，从起始 Token 开始逐步生成文本。大规模预训练数据的加持使得 Decoder-only 架构的模型能够生成高质量、连贯的文本，在自动故事生成、新闻文章生成此类不依赖于特定的输入文本的**无条件文本生成任务**中表现出色。然而，在模型规模有限的情况下（例如 GPT-1 以及 GPT-2 等模型），由于缺乏编码器提供的双向上下文信息，Decoder-only 架构的模型在理解复杂输入数据时**存在一定局限性**，表现可能不如 Encoder-Decoder 架构。

在不同的历史阶段，三种模型架构分别展现了自身的优势。随着模型规模以及数据规模的显著增长，Decoder-only 架构的模型逐渐占据上风，以其**强大的任务泛化**性能展现出成为“大一统”的架构的潜力。当前，以 GPT-3、GPT-4 等为代表的大型 Decoder-only 语言模型，已经发展出了与人类媲美甚至超越人类的记忆、推理以及执行复杂任务的能力。

### 2.2.3 模型架构的历史演变

随着时间的流逝，我们见证了上述三种架构的演变和流行趋势的更替。在大语言模型的早期发展阶段（2018 年左右），BERT 和 GPT-1 分别作为 Encoder-only 和 Decoder-only 架构的代表几乎同时出现。但受限于当时的模型参数规模，BERT 强大的上下文理解能力比 GPT-1 初阶的文本生成能力更为亮眼。使得 Encoder-only 架构得到了更为广泛的探索和应用，而 Decoder-only 架构吸引得关注则相对关注较少。然而，随着用户对**机器翻译等生成任务需求的增加**，缺乏解码组件的 Encoder-only 架构逐渐无法满足直接生成的需求，因而被逐渐“冷落”。同时，2019 年末诞生了一众 Encoder-Decoder 架构的模型，由于其能够**有效处理序列到序列的生成任**

务，逐渐成为主流。到了 2019 年末，随着算力资源的急速发展，研究者开始寻求不断提升参数量来激发更强的生成能力。得益于其参数易扩展性，Decoder-only 架构下的模型参数量急剧扩充，文本生成能力大幅提升。自 2021 年之后，在 GPT-3 等模型的推动下，Decoder-only 架构开始占据主流，甚至逐渐主导了大语言模型的发展。尽管如此，Encoder-Decoder 架构也仍然活跃在开源社区中，不断被探索和改进。至于 Encoder-only 架构，在 BERT 带来最初的爆炸性增长之后，其关注度有所下降，但也仍然在部分判别任务中发挥着重要作用，例如文本分类、情感分析、命名实体识别等。总的来讲，大语言模型的主流架构经历了从 Encoder-only 到 Encoder-Decoder，再到 Decoder-only 的发展过程，而引发这种更迭趋势的原因可能是模型本身生成能力以及计算效率上的差异。

**Encoder-only 架构**专注于对输入数据进行深入的表示和理解，这使得它在理解型任务中表现出色。但当涉及到文本生成任务时，其生成能力则不尽如人意。由于缺少专门用于生成输出的组件，它需要通过迭代进行掩码预测来生成文本，这需要模型进行多次前向传播以逐步填充文本中的掩码部分，从而导致计算资源的大量消耗。在当今对人工智能内容生成（AIGC）需求不断增长的背景下，这种架构的应用受到了一定的限制。而 **Encoder-Decoder 架构**使用编码器来处理输入，并使用解码器来生成输出，能够处理复杂的序列到序列任务。但相较于后续崛起的 Decoder-only 架构，其模型参数规模更为庞大，随之带来的是显著增加的计算复杂度。特别是在处理长序列数据时，编码器和解码器均面临沉重的计算负担，这在一定程度上限制了模型的应用效率。相比之下，**Decoder-only 架构**通过仅使用解码器来优化计算流程，显著简化了模型结构。它采用自回归生成策略，逐步构建输出文本，每一步的生成过程仅需考虑已经生成的文本部分，而不是整个输入序列。这样的设计减少了模型的参数量和计算需求，提高了模型本身的可扩展性，因此非常适合大量文本生成的场景，在 AIGC 的应用场景下得到了广泛的应用。

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

## 2.3 基于 Encoder-only 架构的大语言模型

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

### 2.3.1 Encoder-only 架构

Encoder-only 架构的核心在于能够覆盖输入所有内容的**双向编码模型** (Bidirectional Encoder Model)。在处理输入序列时，双向编码模型融合了从左往右的正向注意力以及从右往左的反向注意力，能够充分捕捉每个 Token 的上下文信息，因此也被称为具有**全面的注意力机制**。得益于其上下文感知能力和动态表示的优势，双向编码器显著提升了自然语言处理任务的性能。

不同于先前常用的 Word2Vec 和 GloVe 此类，为每个词提供一个**固定向量表示**的**静态编码方式**，双向编码器为每个词生成**动态的上下文嵌入** (Contextual Embedding)，这种嵌入依赖于输入序列的具体上下文，使得模型能够更加精准地理解词与词之间的依赖性和语义信息，有效处理词语的多义性问题。这种动态表示使得双向编码器在**句子级别的任务上表现出色**，显著超过了静态词嵌入方法的性能 [20]。

Encoder-only 架构基于双向编码模型，选用了 Transformer 架构中的编码器部分。虽然 Encoder-only 模型不直接生成文本，但其生成的上下文嵌入对于深入理



解输入文本的结构和含义至关重要。这些模型在需要深度理解和复杂推理的自然语言处理任务中展现出卓越的能力，成为了自然语言处理领域的宝贵工具。当前，BERT[11] 及其变体，如 RoBERTa[23]、ALBERT[17] 等，都是基于 Encoder-only 架构的主流大语言模型。接下来将对这些模型进行介绍。

### 2.3.2 BERT 语言模型

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

#### 1. BERT 模型结构

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

#### 2. BERT 预训练方式

BERT 使用小说数据集 BookCorpus[46] (包含约 8 亿个 Token) 和英语维基百科数据集<sup>6</sup> (包含约 25 亿个 Token) 进行预训练，总计约 33 亿个 Token，总数据量达到了 15GB 左右。在预训练任务上，BERT 开创性地提出了掩码语言建模 (Masked

<sup>6</sup><https://dumps.wikimedia.org/>



Language Model, MLM) 和下文预测 (Next Sentence Prediction, NSP) 两种任务来学习生成上下文嵌入。其完整的预训练流程如图2.7所示。

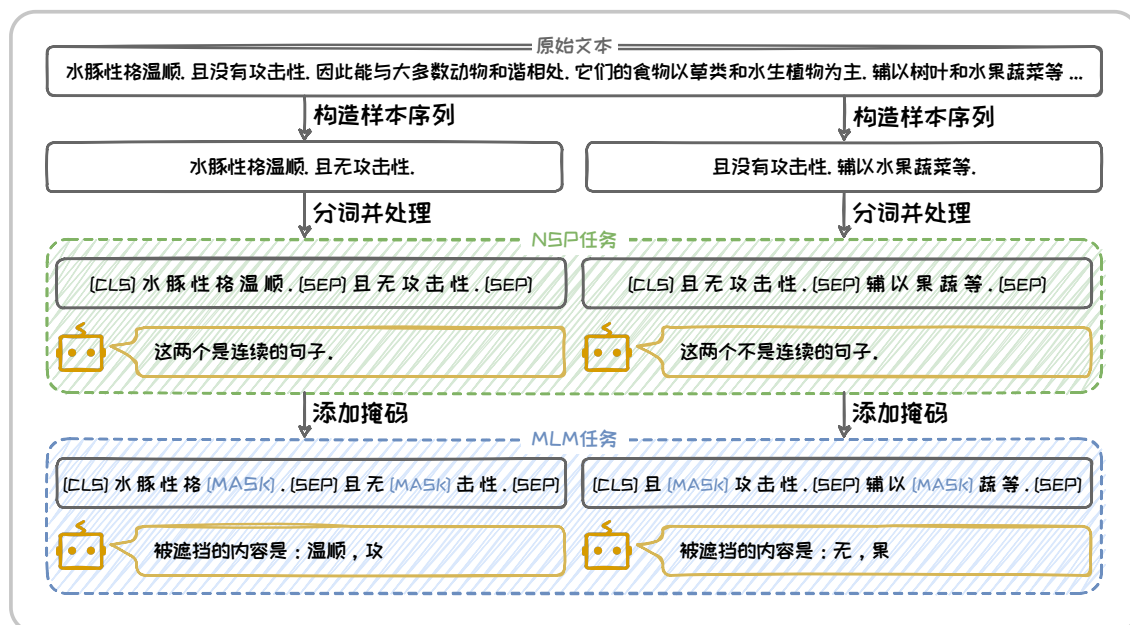


图 2.7: BERT 预训练任务。

具体而言, BERT 先基于给定的原始文本构造多个样本序列, 每个样本序列由原始文本中的两个句子组成, 这两个句子有 50% 的概率是来自原文的连续句子, 另外 50% 的概率是随机挑选的两个句子。随后, 对构造出来的样本序列进行分词, 并在序列的开头添加特殊标签 [CLS], 在每个句子的结尾添加特殊标签 [SEP]。其中 [CLS] 标签用于聚合整个序列的信息, 而 [SEP] 标签则明确句子之间的界限。

接着, BERT 利用处理后的序列进行下文预测任务, 利用模型判断样本序列中的两个句子**是否为连续的**。这一任务训练 BERT 识别和理解句子之间的关系, 捕捉句子层面的语义特征。这对于理解文本的逻辑流、句子之间的关联性有很大帮助, 特别是在问答和自然语言推理等需要理解文档层次结构的自然语言处理 (NLP) 任务中。

最后, BERT 随机选择样本序列中大约 15% 的 Token 进行遮掩, 将其替换为特殊标签 [MASK] 或者随机单词。模型需要**预测这些被替换的 Token 的原始内容**。这

这个过程类似于**完型填空**，要求模型根据周围的上下文信息来推断缺失的 Token。预测过程使用的交叉熵损失函数驱动了 BERT 模型中参数的优化，使其能够学习到文本的双向上下文表示。值得注意的是，在 MLM 任务的训练过程中，BERT 仅针对那些被随机替换的 Token 进行学习，即只计算这些 Token 的预测损失来更新模型参数。

通过这两种预训练任务的结合，使 BERT 在理解语言的深度和广度上都有显著提升。BERT 不仅能够捕捉到 Token 的细粒度特征，还能够把握长距离的依赖关系和句子间的复杂联系，为各种下游任务提供了坚实的语言理解基础。

### 3. BERT 下游任务

BERT 可以应用于各种自然语言处理任务，包括但不限于文本分类（如情感分析）、问答系统、文本匹配（如自然语言推断）和语义相似度计算等。然而，由于 BERT 的输出是输入中所有 Token 的向量表示，因此总长度不固定，无法直接应用于各类下游任务。为了解决这一问题，BERT 设计了 [CLS] 标签来提取**整个输入序列的聚合表示**。[CLS] 标签是专门为分类和汇总任务设计的特殊标记。其全称是“Classification Token”，即分类标记。通过注意力机制，[CLS] 标签汇总整个输入序列的信息，生成一个固定长度的向量表示，从而实现对所有 Token 序列信息的概括，便于处理各种下游任务。

在**文本分类任务**中，可以将输出中 [CLS] 标签对应的向量提取出来，传递给一个**全连接层**，从而用于分类，例如判断整个句子的情绪是积极、消极或是中立的。在**问答系统任务**中，需要输入问题以及一段相关的文本，即 “[CLS] 问题 [SEP] 文本 [SEP]”。最终同样提取出 [CLS] 标签的对应向量，并传递给**两个全连接层**，用于判断答案是否存在于相关文本中。如果存在，这两个全连接层分别用于输出答案的起始和结束位置。通过这种方式，BERT 能够从提供的段落中准确提取出问题的答案。在**语义相似度任务**中，需要计算两段或者多段文本之间的语义相似度。在

这一任务中，可以通过构造 “[CLS] 文本 1 [SEP] 文本 2 [SEP]” 的方式，结合一个**线性层**来直接输出两个文本之间的相似度；也可以**不添加额外的组件**，直接提取 [CLS] 标签对应的向量，再利用额外的相似度度量方法（例如余弦相似度）来计算多段文本之间的相似度。

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

### 2.3.3 BERT 衍生语言模型

BERT 的突破性成功还催生了一系列相关衍生模型，这些模型继承了 BERT 双向编码的核心特性，并在其基础上进行了改进和优化，以提高模型性能或者模型效率，在特定任务和应用场景中展现出了卓越的性能。较为代表性的衍生模型为 RoBERTa[23]、ALBERT[17] 以及 ELECTRA[9] 等，接下来将对这三种模型分别展开介绍。

#### 1. RoBERTa 语言模型

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

### (1) RoBERTa 模型结构

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

### (2) RoBERTa 预训练方式

在预训练语料库的选择上，RoBERTa 在 BERT 原有的小说数据集 BookCorpus[46]（包含约 8 亿个 Token）以及英语维基百科数据集<sup>7</sup>（包含约 25 亿个 Token）的基础上，添加了新闻数据集 CC-News<sup>8</sup>（包含约 76GB 的新闻文章）、网页开放数据集 OpenWebText<sup>9</sup>（包含约 38GB 的网页文本内容）以及故事数据集 Stories[41]（包含约 31GB 的故事文本），总数据量达到约 160GB。

至于具体的预训练任务，RoBERTa 移除了 BERT 中的下文预测任务，并将 BERT 原生的静态掩码语言建模任务更改为**动态掩码语言建模**。具体而言，BERT 在数据预处理期间对句子进行掩码，随后在每个训练 epoch 中，掩码位置不再变化。而 RoBERTa 则将训练数据复制成 10 个副本，分别进行掩码。在同样训练 40 个 epoch 的前提下，BERT 在其静态掩码后的文本上训练了 40 次，而 RoBERTa 将 10 个不同掩码后的副本分别训练了 4 次，从而增加模型训练的多样性，有助于模型学习到更丰富的上下文信息。

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

---

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

<sup>8</sup><http://web.archive.org/save/http://commoncrawl.org/2016/10/newsdataset-available>

<sup>9</sup><http://Skylion007.github.io/OpenWebTextCorpus>



## 2. ALBERT 语言模型

ALBERT (A Lite BERT) 是由 Google Research 团队于 2019 年 9 月提出的轻量级 BERT 模型，旨在通过参数共享和嵌入分解技术来减少模型的参数量和内存占用，从而**提高训练和推理效率**。BERT-Base 模型有着 1.1 亿个参数，而 BERT-Large 更是有着 3.4 亿个参数，这使得 BERT 不仅难以训练，推理时间也较长。ALBERT 在设计过程通过参数因子分解技术和跨层参数共享技术**显著减少了参数的数量**。接下来将对 ALBERT 模型的结构、预训练方式以及下游任务进行介绍。

### (1) ALBERT 模型结构

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

#### 参数因子分解

在 BERT 中，Embedding 层的输出向量维度  $E$  与隐藏层的向量维度  $H$  是一致的，这意味着 Embedding 层的输出直接用作后续编码模块的输入。具体来说，BERT-Base 模型对应的词表大小  $V$  为 3,0000 左右，并且其隐藏层的向量维度  $H$  设置为 768。因此，BERT 的 Embedding 层需要的参数数量是  $V \times H$ ，大约为 2,304 万。

相比之下，ALBERT 将 Embedding 层的矩阵先进行分解，将词表对应的独热编码向量通过一个低维的投影层下投影至维度  $E$ ，再将其上投影回隐藏状态的维度  $H$ 。具体来说，ALBERT 选择了一个较小的 Embedding 层维度，例如 128，并将参数数量拆解为  $V \times E + E \times H$ 。按照这个设计，ALBERT 的 Embedding 层大约需要 394 万个参数，大约是 BERT 参数数量的六分之一。对于具有更大隐藏层向量维度  $H$  的 Large 版本，ALBERT 节省参数空间的优势更加明显，能够将参数量压缩至 BERT 的八分之一左右。



### 跨层参数共享

以经典的 BERT-Base 模型为例，模型中共有 12 层相同架构的编码模块，所有 Transformer 块的参数都是独立训练的。ALBERT 为了降低模型的参数量，提出了跨层参数共享机制，只学习第一层编码模块的参数，并将其直接共享给其他所有层。该机制在一定程度上牺牲了模型性能，但显著提升了参数存储空间的压缩比，从而实现了更高效的资源利用。

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

### (2) ALBERT 预训练方式

ALBERT 使用与 BERT 完全一致的数据集来进行预训练，即小说数据集 Book-Corpus[46]（包含约 8 亿个 Token）以及英语维基百科数据集<sup>10</sup>（包含约 25 亿个 Token），总计 33 亿个 Token，约 15GB 数据量。另外，在预训练任务的选择上，ALBERT 保留了 BERT 中的掩码语言建模任务，并将下文预测任务替换为**句序预测**（Sentence Order Prediction, SOP），如图2.8所示。具体而言，ALBERT 从文本中选择连续的两个句子，将这两个句子直接拼接起来，或是先将这两个句子的顺序

<sup>10</sup><https://dumps.wikimedia.org>

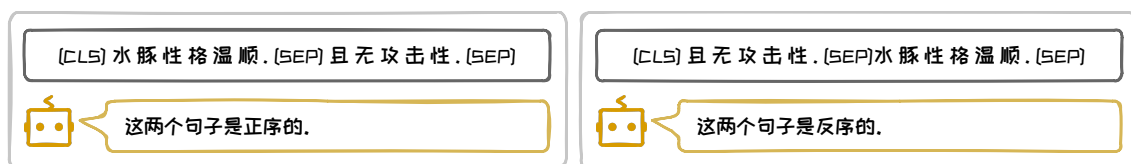


图 2.8: ALBERT 句序预测任务。

翻转后再进行拼接，并将拼接后的内容作为输入样本，而模型需要预测该样本中的两个句子是正序还是反序。

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

### 3. ELECTRA 语言模型

ELECTRA (Efficiently Learning an Encoder that Classifies Token Replacements Accurately) [9] 是由 Google Brain 和斯坦福大学的研究人员于 2020 年 3 月提出的另一种 BERT 变体，旨在解决大规模预训练语言模型中的效率和可扩展性问题。通过使用生成器-判别器架构，ELECTRA 能够更高效地利用预训练数据，提高了模型在下游任务中的表现。

#### (1) ELECTRA 预训练方式

在模型结构上，ELECTRA 在 BERT 原有的掩码语言建模基础上结合了生成对抗网络 (Generative Adversarial Network, GAN) 的思想，采用了一种生成器-判别器结构。具体来说，ELECTRA 模型包含一个生成器和一个判别器，其中生成器 (Generator) 是一个能进行掩码预测的模型 (例如 BERT 模型)，负责将掩码后的文本恢复原状。而判别器 (Discriminator) 则使用替换词检测 (Replaced Token Detection, RTD) 预训练任务，负责检测生成器输出的内容中的每个 Token 是否是原文中的内容。其完整的流程如图 2.9 所示。

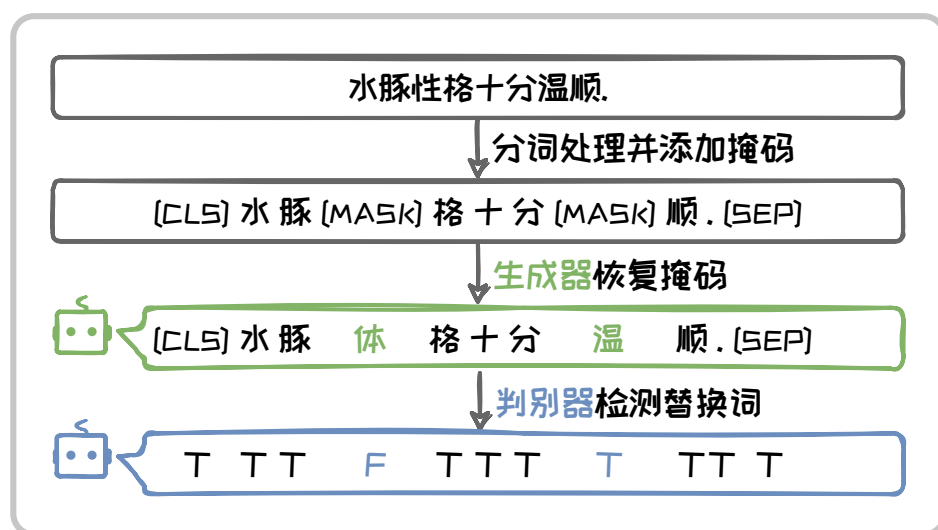


图 2.9: ELECTRA 预训练任务。

## (2) ELECTRA 模型结构

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

其中，ELECTRA-Small 和 ELECTRA-Base 使用与 BERT 一致的数据集来进行预训练，共包含 33 亿个 Token。而 ELECTRA-Large 则使用了更多样化的训练数据，包括大规模网页数据集 ClueWeb<sup>11</sup>、CommonCrawl<sup>12</sup>以及大型新闻文本数据集

<sup>11</sup><https://lemurproject.org/clueweb09.php>

<sup>12</sup><https://commoncrawl.org>

Gigaword<sup>13</sup>，最终将数据量扩充到了 330 亿个 Token，从而帮助模型学习更广泛的语言表示。

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

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

上述基于 Encoder-only 架构的大语言模型在文本分类、情感分析等多个自然语言处理任务中取得了良好效果。表 2.1 从模型参数量及预训练语料等方面对上述模型进行总结。可以看出这些经典模型参数大小止步于 6.6 亿，预训练任务也主要服务于自然语言理解。这些模型没有继续寻求参数量上的突破，并且通常只专注于判别任务，难以应对生成式任务，因此在当前愈发热门的生成式人工智能领域中可以发挥的作用相对有限。

表 2.1: Encoder-only 架构代表模型参数和语料大小表。

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

## 2.4 基于 Encoder-Decoder 架构的大语言模型

Encoder-Decoder 架构在 Encoder-only 架构的基础上引入 Decoder 组件，以完成机器翻译等序列到序列 (Sequence to Sequence, Seq2Seq) 任务。本节将对 Encoder-Decoder 架构及其代表性模型进行介绍。

<sup>13</sup><https://catalog.ldc.upenn.edu/LDC2011T07>

### 2.4.1 Encoder-Decoder 架构

Encoder-Decoder 架构主要包含编码器和解码器两部分。该架构的详细组成如图2.10所示。其中，编码器部分与 Encoder-only 架构中的编码器相同，由多个编码模块堆叠而成，每个编码模块包含一个自注意力模块以及一个全连接前馈模块。模型的输入序列在通过编码器部分后会被转变为固定大小的上下文向量，这个向量包含了输入序列的丰富语义信息。解码器同样由多个解码模块堆叠而成，每个解码模块由一个带掩码的自注意力模块、一个交叉注意力模块和一个全连接前馈模块组成。其中，带掩码的自注意力模块引入掩码机制防止未来信息的“泄露”，确保解码过程的自回归特性。交叉注意力模块则实现了解码器与编码器之间的信息交互，对于生成与输入序列高度相关的输出至关重要。

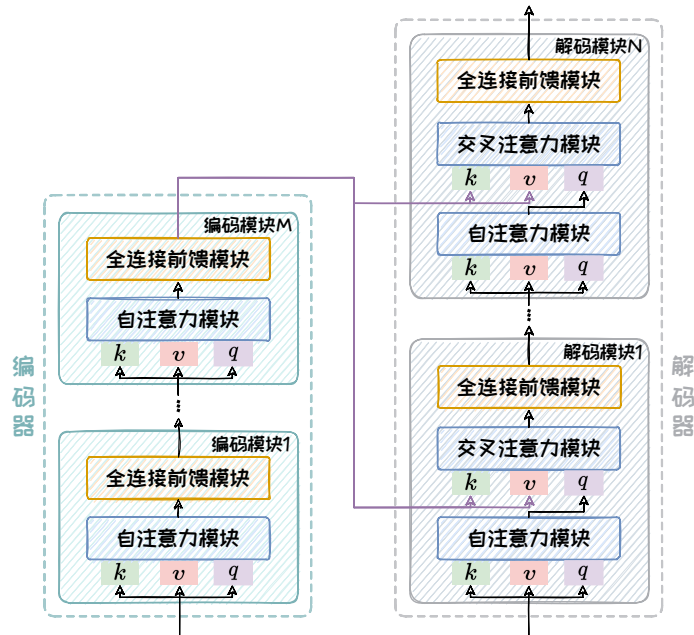


图 2.10: Encoder-Decoder 架构。

自注意模块在编码器和解码器中的注意力目标不同的。在编码器中，我们需要对输入序列的上下文进行“通盘考虑”，所以采用双向注意力机制以全面捕捉上下文信息。但在解码器中，自注意力机制则是单向的，仅以上文为条件来解码得到



下文，通过掩码操作避免解码器“窥视”未来的信息。交叉注意力通过将解码器的查询（query）与编码器的键（key）和值（value）相结合，实现了两个模块间的有效信息交流。

通过自注意力和交叉注意力机制的结合，Encoder-Decoder 架构能够**高效地编码输入信息并生成高质量的输出序列**。自注意力机制确保了输入序列和生成序列内部的一致性和连贯性，而交叉注意力机制则确保了解码器在生成每个输出 Token 时都能参考输入序列的全局上下文信息，从而生成与输入内容高度相关的结果。在这两个机制的共同作用下，Encoder-Decoder 架构不仅能够深入理解输入序列，还能够根据不同任务的需求灵活生成长度适宜的输出序列，在机器翻译、文本摘要、问答系统等任务中得到了广泛应用，并取得了显著成效。本节将介绍两种典型的基于 Encoder-Decoder 架构的代表性大语言模型：T5[29] 和 BART[18]。

## 2.4.2 T5 语言模型

自然语言处理涵盖了语言翻译、文本摘要、问答系统等多种任务。通常，每种任务都需要对训练数据、模型架构和训练策略进行定制化设计。这种定制化设计不仅耗时耗力，而且训练出的模型难以跨任务复用，导致开发者需要不断“重复造轮子”。为了解决这一问题，Google Research 团队在 2019 年 10 月提出了一种基于 Encoder-Decoder 架构的大型预训练语言模型 T5（Text-to-Text Transfer Transformer），其采用了统一的文本到文本的转换范式来处理多种任务。下面分别从模型结构、预训练方式以及下游任务三个方面对 T5 模型进行介绍。

### 1. T5 模型结构

T5 模型的核心思想是**将多种 NLP 任务统一到一个文本转文本的生成式框架中**。在此统一框架下，T5 通过不同的输入前缀来指示模型执行不同任务，然后生成相应的任务输出，正如图2.11所示。这种方法可以视为早期的提示（Prompt）技

术的运用，通过构造合理的输入前缀，T5 模型能够引导自身针对特定任务进行优化，而无需对模型架构进行根本性的改变。这种灵活性和任务泛化能力显著提高了模型的实用性，使其能够轻松地适应各类新的 NLP 任务。

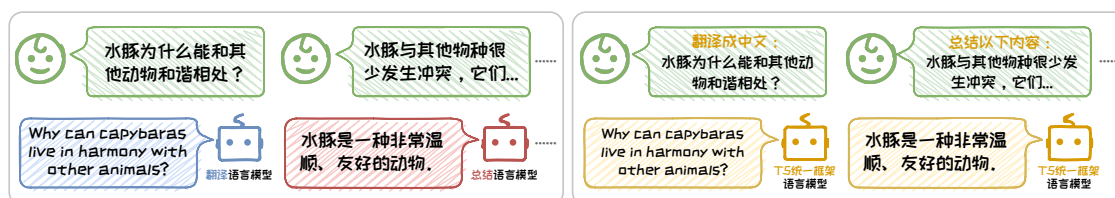


图 2.11: 传统语言模型和 T5 统一框架。

在模型架构方面，T5 与原始的包括一个编码器和一个解码器的 Transformer 架构相同。每个编码器和解码器又分别由多个编码模块和解码模块堆叠而成。T5 模型根据不同的具体参数，提供了五个不同的版本，分别是 T5-Small、T5-Base、T5-Large、T5-3B 以及 T5-11B。**T5-Small** 由 6 个编码模块和 6 个解码模块堆叠而成，其中隐藏层维度为 512，自注意力头的数量为 8，**总参数数量约为 6000 万**；**T5-Base** 与 BERT-Base 对标，由 12 个编码模块和 12 个解码模块堆叠而成，其中隐藏层维度为 768，自注意力头的数量为 12，**总参数数量约为 2.2 亿**；**T5-Large** 与 BERT-Large 对标，由 24 个编码模块和 24 个解码模块堆叠而成，隐藏层维度为 1024，自注意力头的数量为 16，**总参数数量约为 7.7 亿**；**T5-3B** 在 T5-Large 的基础上，把自注意力头的数量扩大到 32，另外还将全连接前馈网络的中间层维度扩大了 4 倍，**总参数数量约为 28 亿**；**T5-11B** 在 T5-3B 的基础上，进一步将自注意力头的数量扩大到 128，并又将全连接前馈网络的中间层维度扩大了 4 倍，**总参数数量约为 110 亿**。

### 2. T5 预训练方式

为了获取高质量、覆盖范围广泛的预训练数据集，Google Research 团队从大规模网页数据集 Common Crawl<sup>14</sup>中提取了大量的网页数据，并经过严格的清理和

<sup>14</sup><https://commoncrawl.org>

过滤，最终生成了 C4 数据集（Colossal Clean Crawled Corpus），其覆盖了各种网站和文本类型，总规模达到了约 750GB。

基于此数据集，T5 提出了名为 Span Corruption 的预训练任务。这一与训练任务从原始输入中选择 15% 的 Token 进行破坏，每次都选择连续三个 Token 作为一个小段（span）整体被掩码成 [MASK]。与 BERT 模型中采用的单个 Token 预测不同，T5 模型需要**对整个被遮挡的连续文本片段进行预测**。这些片段可能包括连续的短语或子句，它们在自然语言中构成了具有完整意义的语义单元。这一设计要求模型不仅等理解局部词汇的表面形式，还要可以捕捉更深层次句子结构和上下文之间的复杂依赖关系。Span Corruption 预训练任务显著提升了 T5 的性能表现，尤其是在文本摘要、问答系统和文本补全等需要生成连贯和逻辑性强的文本生成任务中。

### 3. T5 下游任务

基于预训练阶段学到的大量知识以及新提出的文本转文本的统一生成式框架，T5 模型可以在完全**零样本**（Zero-Shot）的情况下，利用 Prompt 工程技术直接适配到多种下游任务。同时，T5 模型也可以通过微调（Fine-Tuning）来适配到特定的任务。但是，微调过程需要针对下游任务收集带标签训练数据，同时也需要更多的计算资源和训练时间，因此通常只被应用于那些对精度要求极高且任务本身较为复杂的应用场景。

综上所述，T5 模型的文本转文本的统一生成式框架不仅简化了不同自然语言处理任务之间的转换流程，也为大语言模型的发展提供了新方向。如今，T5 模型已经衍生了许多变体，以进一步改善其性能。例如，mT5[43] 模型扩展了对 100 多种语言的支持，T0[31] 模型通过多任务训练增强了零样本学习（Zero-Shot Learning）能力，Flan-T5[8] 模型专注于通过指令微调，以实现进一步提升模型的灵活性和效率等等。

### 2.4.3 BART 语言模型

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

#### 1. BART 模型结构

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

#### 2. BART 预训练方式

在预训练数据上, BART 使用了与 RoBERTa[23] 相同的语料库, 包含小说数据集 BookCorpus[46]、英语维基百科数据集<sup>15</sup>、新闻数据集 CC-News<sup>16</sup>、网页开放数据集 OpenWebText<sup>17</sup>以及故事数据集 Stories, 总数据量达到约 160GB。

在预训练任务上, BART 以重建被破坏的文本为目标。其通过 Token 遮挡任务 (Token Masking)、Token 删除任务 (Token Deletion)、连续文本填空任务 (Text Infilling)、句子打乱任务 (Sentence Permutation) 以及文档旋转任务 (Document Rotation) 等五个任务来破坏文本, 然后训练模型对原始文本进行恢复。这种方式

<sup>15</sup><https://dumps.wikimedia.org>

<sup>16</sup><http://web.archive.org/save/http://commoncrawl.org/2016/10/newsdataset-available>

<sup>17</sup><http://Skylion007.github.io/OpenWebTextCorpus>



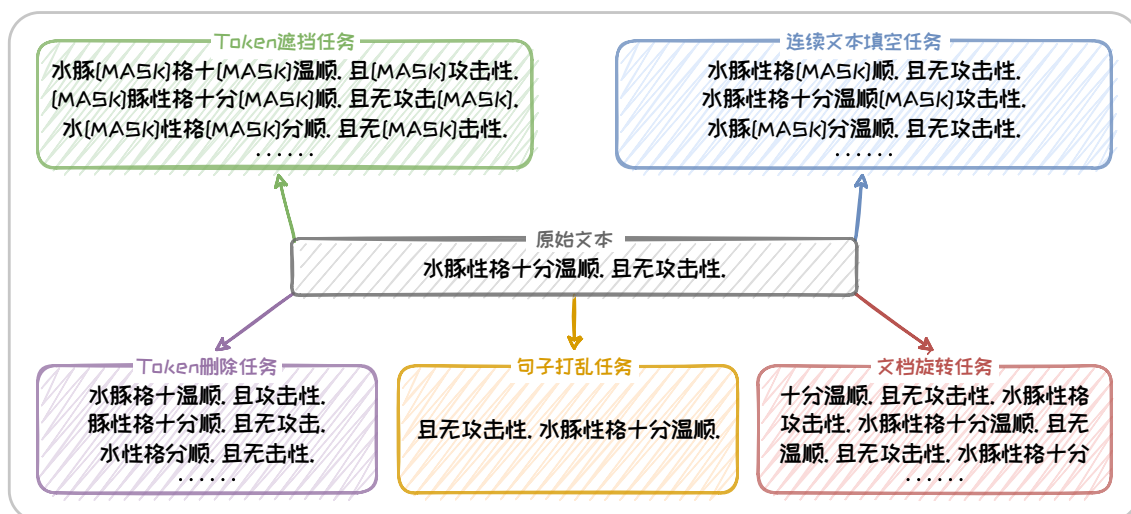


图 2.12: BART 预训练任务。

锻炼了模型对文本结构和语义的深入理解，增强了其在面对不完整或损坏信息时的鲁棒性。五个文本破坏任务的具体形式如下所述。

- **Token 遮挡任务** (Token Masking): 类似于 BERT 中的 MLM 任务，在原始文本中随机采样一部分 Token 并将其替换为 [MASK]，从而训练模型推断被删除的 Token 内容的能力。
- **Token 删除任务** (Token Deletion): 在原始文本中随机删除一部分 Token，从而训练模型推断被删除的 Token 位置以及内容的能力。
- **连续文本填空任务** (Text Infilling): 类似于 T5 的预训练任务，在原始文本中选择几段连续的 Token（每段作为一个 span），整体替换为 [MASK]。其中 span 的长度服从  $\lambda = 3$  的泊松分布，如果长度为 0 则直接插入一个 [MASK]。这一任务旨在训练模型推断一段 span 及其长度的能力。
- **句子打乱任务** (Sentence Permutation): 将给定文本拆分为多个句子，并随机打乱句子的顺序。旨在训练模型推理前后句关系的能力。
- **文档旋转任务** (Document Rotation): 从给定文本中随机选取一个 Token，作为文本新的开头进行旋转。旨在训练模型找到文本合理起始点的能力。



在图2.12中，以给定文本“水豚性格十分温顺. 且无攻击性.”为例，对这个五个任务进行演示。在预训练结束后，可以对 BART 进行微调使其能够将在预训练阶段学到的语言知识迁移到具体的应用场景中，适配多种下游任务。这种从预训练到微调的流程，使得 BART 不仅在文本生成任务上表现出色，也能够适应文本理解类任务的挑战。后续同样也出现了 BART 模型的各种变体，包括可处理跨语言文本生成任务的 mBART[22] 模型等。

综上所述，基于 Encoder-Decoder 架构的大语言模型，在生成任务中展示了良好的性能表现。表2.2从模型参数量和预训练语料规模的角度对本章提到的基于 Encoder-Decoder 架构的模型进行了总结。可以看出此时模型参数数量的上限已达 110 亿。在模型结构和参数规模的双重优势下，相较于基于 Encoder-only 架构的模型，这些模型在翻译、摘要、问答等任务中取得了更优的效果。

表 2.2: Encoder-Decoder 架构代表模型参数和语料大小表。

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

## 2.5 基于 Decoder-only 架构的大语言模型

在开放式 (Open-Ended) 生成任务中，通常输入序列较为简单，甚至没有具体明确的输入，因此维持一个完整的编码器来处理这些输入并不是必要的。对于这种任务，Encoder-Decoder 架构可能显得过于复杂且缺乏灵活性。在这种背景下，Decoder-only 架构表现得更为优异。本节将对 Decoder-only 架构及其代表性模型进行介绍。

## 2.5.1 Decoder-only 架构

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

值得一提的是，Decoder-only 架构模型的概念最早可以追溯到 2018 年发布的 GPT-1[27] 模型。但在当时，由于以 BERT 为代表的 Encoder-only 架构模型在各项任务中展现出的卓越性能，Decoder-only 架构并没有受到足够的关注。直到 2020 年，GPT-3[5] 的突破性成功，使得 Decoder-only 架构开始被广泛应用于各种大语言模型中，其中最为流行的有 OpenAI 提出的 GPT 系列、Meta 提出的 LLaMA 系列等。其中，GPT 系列是起步最早的 Decoder-only 架构，在性能上也成为了时代的标杆。但从第三代开始，GPT 系列**逐渐走向了闭源**。而 LLaMA 系列虽然起步较晚，但凭借着同样出色的性能以及**始终坚持的开源道路**，也在 Decoder-only 架构领域占据了一席之地。接下来将对这两种系列的模型进行介绍。

## 2.5.2 GPT 系列语言模型

GPT (Generative Pre-trained Transformer) 系列模型是由 OpenAI 开发的一系列基于 Decoder-only 架构的大语言模型。自从 2018 年问世以来，GPT 系列模型经历了快速的发展，其在模型规模、预训练范式上不断演进，取得了万众瞩目的效果，引领了本轮大语言模型发展的浪潮。其演进历程可以划分为五个阶段，表2.3对这五个阶段的模型参数规模和预料规模进行了总结。从表中可以明显看出，GPT 系列模型**参数规模与预训练语料规模呈现出激增的趋势**。然而，自 ChatGPT 版本起，

GPT 系列模型转向了闭源模式，其具体的参数量和预训练数据集的详细信息已不再公开。尽管如此，根据扩展法则，有理由猜测 ChatGPT 及其后续版本在参数规模与预训练语料规模上都有所增长。下面将对这五个发展阶段的模型分别进行介绍。

表 2.3: GPT 系列模型参数和语料大小表。

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

### 1. 初出茅庐：GPT-1 模型

OpenAI 的前首席科学家 Ilya Sutskever 在采访中<sup>18</sup>透露，OpenAI 自成立初期就开始探索如何通过下一词预测解决无监督学习的问题。但当时所用的 RNN 模型无法很好解决长距离依赖问题，上述问题没有得到很好的解决。直到 2017 年，Transformer 的出现为这一问题提供了新的解决方案了，为 OpenAI 的发展指明了方向。随后，OpenAI 开始步入正轨。2018 年 6 月，OpenAI 发布了第一个版本的 GPT (Generative Pre-Training) 模型，被称为 GPT-1[27]。GPT-1 开创了 Decoder-only 架构下，通过下一词预测解决无监督文本生成的先河，为自然语言处理领域带来了革命性的影响。下面将分别对 GPT-1 的模型结构、预训练、下游任务等方面展开介绍。

#### (1) GPT-1 模型结构

在模型架构方面，GPT-1 使用了 Transformer 架构中的 Decoder 部分，省略了 Encoder 部分以及交叉注意力模块。其模型由 12 个解码块堆叠而成，每个解码块

<sup>18</sup><https://hackernoon.com/an-interview-with-ilya-sutskever-co-founder-of-openai>

包含一个带掩码的自注意力模块和一个全连接前馈模块。其中隐藏层维度为 768，自注意力头的数量为 12，模型的最终参数数量约为 1.17 亿。

图2.13对比了 BERT-Base 以及 GPT-1 的模型结构。从图中可以看出，GPT-1 在结构上与 BERT-Base 高度类似，两者都包含 12 个编码或解码模块，每个模块也同样由一个自注意力模块和一个全连接前馈模块组成。两者之间的本质区别在于 BERT-Base 中的自注意力模块是双向的自注意力机制，而 GPT-1 中的自注意力模块则是带有掩码的单向自注意力机制。

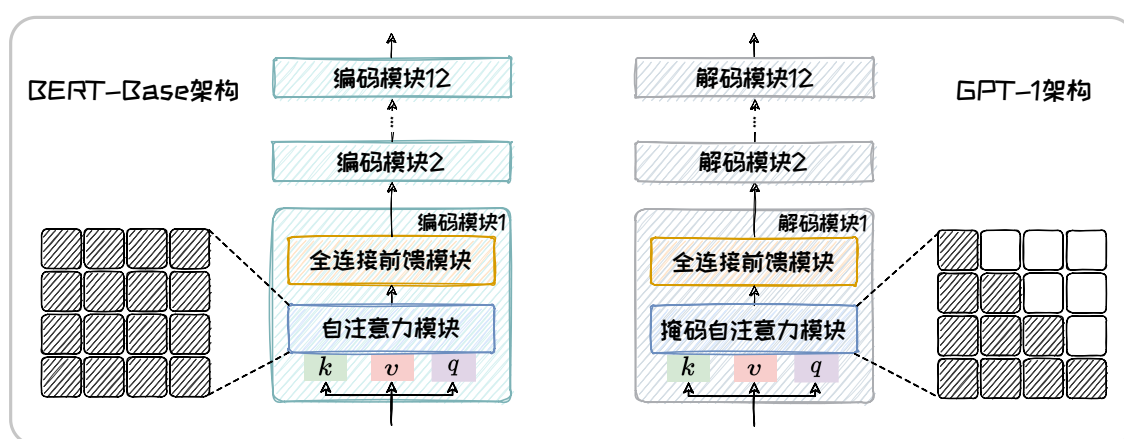


图 2.13: BERT-Base 和 GPT-1 模型。

## (2) GPT-1 预训练方法

GPT-1 使用小说数据集 BookCorpus[46] 来进行预训练，该数据集包含约 8 亿个 Token，总数据量接近 5GB。在预训练方法上，GPT-1 采用下一词预测任务，即基于给定的上文预测下一个可能出现的 Token。以自回归的方法不断完成下一词预测任务，模型可以有效地完成文本生成任务，如图2.14所示。通过这种预训练策略，模型可以在不需要人为构造大量带标签数据的前提下，学习到大量语言的“常识”，学会生成连贯且上下文相关的文本。这不仅提高了模型的泛化能力，而且减少了对标注数据的依赖，为自然语言处理领域带来了新的研究方向和应用前景。

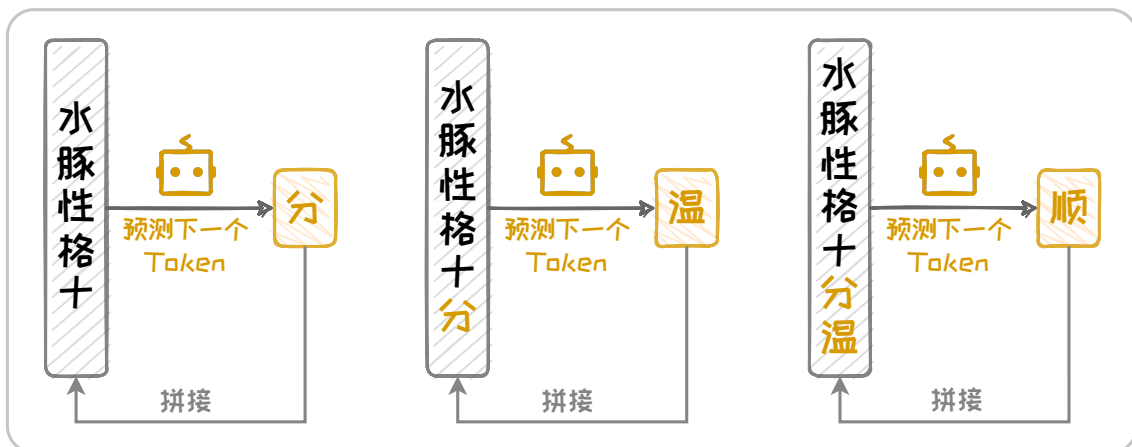


图 2.14: GPT-1 语言建模预训练任务。

### (3) GPT-1 下游任务

尽管 GPT-1 模型在预训练后展现出了一定的潜力，但其任务泛化能力仍受限于当时的训练数据量和模型参数数量。为了提升模型在特定下游任务上的表现，通常需要进行进一步的有监督微调。微调过程涉及使用针对特定任务的标注数据来优化模型的参数，其中模型的输入和输出均以文本序列的形式呈现。例如，在以下任务中，我们需要构建针对特定应用场景的微调策略：

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

GPT-1 具备原生的文本生成能力。但受限于训练数据量和模型参数数量，其生成能力还不足以用于解决实际问题。此外，由于其单向注意力机制的限制，其全面



理解上下文的能力也有所欠缺。四个月後，具有双向上下文理解能力的 BERT 被提出，并以其强大的上下文嵌入能力迅速吸引了业界的广泛关注，遮盖了 GPT-1 的锋芒。尽管 GPT-1 当时在实用性上可能不及 BERT，但它作为 Decoder-only 架构的开端，为后续大语言模型的惊艳表现拉开了序幕。

## 2. 小有所成：GPT-2 模型

虽然 GPT-1 锋芒未露，OpenAI 并没有改变其 Decoder-only 的技术路线，而是选择在这一路线上继续深耕。在 2019 年 2 月，OpenAI 发布了 GPT 系列的第二代产品 GPT-2<sup>19</sup>。相较于 GPT-1，GPT-2 在模型规模和预训练样本的质量上都进行了显著的提升，显著增强了模型的任务泛化能力。以下将对 GPT-2 的模型结构、预训练语料和下游任务适配情况进行介绍。

### (1) GPT-2 模型结构

GPT-2 模型延续了 GPT-1 的 Decoder-only 架构，并在此基础上进一步加大了参数数量。GPT-2 一共发布了四个版本，分别是 GPT-2 Small、GPT-2 Medium、GPT-2 Large 以及 GPT-2 XL。其中 **GPT-2 Small** 在模型规模上接近 GPT-1 以及 BERT-Base，由 12 个编码块堆叠而成，隐藏层维度为 768，自注意力头的数量为 12，**总参数数量约为 1.24 亿**；**GPT-2 Medium** 在模型规模上接近 BERT-Large，由 24 个解码块堆叠而成，隐藏层维度为 1024，自注意力头的数量为 16，**总参数数量约为 3.55 亿**；**GPT-2 Large** 由 36 个解码块堆叠而成，隐藏层维度为 1280，自注意力头的数量为 20，**总参数数量约为 7.74 亿**；**GPT-2 XL** 是最大规模版本，由 48 个解码块堆叠而成，其中隐藏层维度为 1600，自注意力头的数量为 25，**总参数数量约为 15 亿**。

### (2) GPT-2 预训练方法

在预训练中，GPT-2 继续采用下一词预测任务，但进一步提升了预训练数据的数量和质量。其采用了全新的 WebText 数据集，该数据集由 40GB 经过精心筛选和

<sup>19</sup><https://openai.com/index/gpt-2-1-5b-release>

清洗的网络文本组成。通过使用 WebText 数据集进行预训练，GPT-2 的语言理解能力得到了显著增强，接触到了更多样化的语言使用场景，还学习到了更复杂的语言表达方式。这使得 GPT-2 在捕捉语言细微差别和构建语言模型方面更为精准，从而在执行各种自然语言处理任务时能够生成更准确、更连贯的文本。

### (3) GPT-2 下游任务

GPT-2 的任务泛化能力得到了改善，在某些任务上可以不进行微调，直接进行零样本学习。这种能力大大增加了 GPT-2 在处理下游任务时的灵活性，降低了下游任务适配所需的成本。这为 Decoder-only 架构博得了更多关注。

## 3. 崭露头角：GPT-3 模型

为了进一步提升任务泛化能力，OpenAI 于 2020 年 6 月推出了第三代模型 GPT-3[5]。与前两代模型相比，GPT-3 在模型规模和预训练语料上进一步提升，并涌现出了优良的上下文学习（In-Context Learning, ICL）能力。在上下文学习能力的加持下，GPT-3 可以在不进行微调的情况下，仅通过任务描述或少量示例即可完成多样化的下游任务。关于上下文学习详细介绍参见本书 3.2 章节。以下将对 GPT-3 的模型结构、预训练语料和下游任务适配情况进行介绍。

### (1) GPT-3 模型架构

在模型架构上，GPT-3 继承并扩展了前两代的架构，显著增加了解码块的数量、隐藏层的维度和自注意力头的数量，参数量最高达到 1750 亿。庞大的参数量使得 GPT-3 能够捕获更加细微和复杂的语言模式，显著提升了模型的文本生成能力。GPT-3 设计了多个不同参数规模的版本，以满足不同应用场景的需求，详细参数细节见表 2.4。

### (2) GPT-3 预训练方法

延续了前两代的预训练方法，GPT-3 继续采用下一词预测作为预训练任务。其使用了更大规模和更多样化的互联网文本数据集，数据量接近 1TB，涵盖了 Com-

表 2.4: GPT-1 至 GPT-3 模型具体参数表。

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

mon Crawl<sup>20</sup>、WebText、BookCorpus[46]、Wikipedia<sup>21</sup>等多个来源，包括书籍、网站、论坛帖子等各类文本形式。所有数据都经过了严格的筛选和清洗流程，以确保数据的质量和多样性。基于这些数据，GPT-3 学习到了更加丰富和多元的语言知识和世界知识。

### (3) GPT-3 下游任务

GPT-3 模型涌现出了良好的上下文学习能力，使其可以在无需微调的情况下，仅通过在输入文本中明确任务描述和提供少量示例，便能够执行多种下游任务。上下文学习能力极大地增强了 GPT-3 的任务泛化能力，使其能够快速适应不同的应用场景。GPT-3 开始在文本生成、问答系统、语言翻译等众多自然语言处理任务中崭露头角。

## 4. 蓄势待发：InstructGPT 等模型

在 GPT-3 的基础上，OpenAI 进一步推出了一系列衍生模型。这些模型在 GPT-3 网络结构之上通过特定的训练方法，各个“身怀绝技”。例如，采用十亿行代码继续

<sup>20</sup><https://commoncrawl.org>

<sup>21</sup><http://web.archive.org/save/http://commoncrawl.org/2016/10/newsdataset-available>

预训练 (Continual Pre-Training) 的 Codex[6] 模型, 可以有效地处理代码生成任务; 采用用户偏好对齐 (User Intent Alignment) 的 InstructGPT[25] 模型, 具备良好的指令跟随能力。其中, 最具启发意义的是 InstructGPT, 其也是 ChatGPT 的前身。它通过引入了**人类反馈强化学习** (Reinforcement Learning from Human Feedback, RLHF), 显著提升了模型对用户指令的响应能力。

人类反馈强化学习旨在缓解模型在遵循用户指令时可能出现的不准确性和不可靠性, 以使模型生成的内容更符合人类的要求。在人类反馈强化学习中, 人类评估者首先提供关于模型输出质量的反馈, 然后使用这些反馈来微调模型。具体过程如图2.15所示, 整体可以分为以下三个步骤: 1) **有监督微调**: 收集大量“问题-人类回答”对作为训练样本, 对大语言模型进行微调。2) **训练奖励模型**: 针对每个输入, 让模型生成多个候选输出, 并由人工对其进行质量评估和排名, 构成偏好数据集。用此偏好数据集**训练一个奖励模型**, 使其可以对输出是否符合人类偏好进行打分。3) **强化学习微调**: 基于上一步中得到的奖励模型, **使用强化学习方法优化第一步中的语言模型**, 即在语言模型生成输出后, 奖励模型对其进行评分, 强化学习算法根据这些评分调整模型参数, 以提升高质量输出的概率。

经过 RLHF 训练得到的 InstructGPT 的性能通常优于 GPT-3, 尤其是在需要精确遵循用户指令的场景中。它生成的回答更加贴合用户的查询意图, 有效减少了不相关或误导性内容的生成。InstructGPT 的研究为构建更智能、更可靠的 AI 系统提供了新的思路, 展示了人类智慧和机器学习算法结合的巨大潜力。但是, RLHF 的计算成本十分高昂, 主要是由于: 1) **奖励模型**的训练过程复杂且耗时。2) 除了需要单独训练语言模型和奖励模型外, 还需要协调这两个模型进行**多模型联合训练**, 这一过程同样复杂且消耗大量资源。

为了克服 RLHF 在计算效率上的缺陷, 斯坦福大学在 2023 年在其基础上, 提出了一种新的算法**直接偏好优化 (Direct Preference Optimization, DPO)** [28]。DPO

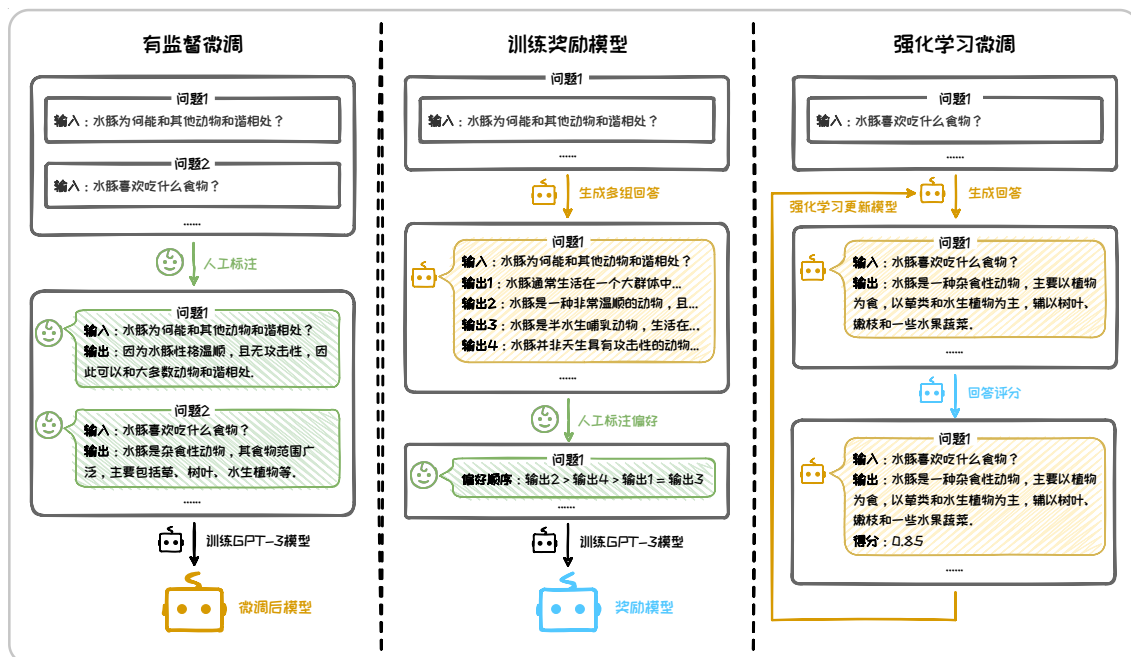


图 2.15: 人类反馈强化学习 (RLHF) 过程。

算法直接利用人类偏好数据来训练模型，省略了单独构建奖励模型以及应用复杂强化学习算法的步骤。该方法首先收集包含多个响应的人类偏好数据，并从中标记出最优和次优响应。然后微调模型以提高模型选择最优响应的概率，同时降低选择次优响应的概率。这种方法显著简化了人类反馈对齐的流程，提高了训练效率和模型稳定性。尽管在处理复杂的人类偏好时可能略逊于 RLHF，但 DPO 在计算效率上的优势使其在多个领域得到了广泛应用。

## 5. 一鸣惊人：ChatGPT 以及 GPT-4 等模型

OpenAI 于 2022 年 11 月推出了聊天机器人 (ChatGPT Chat Generative Pre-trained Transformer)<sup>22</sup>。ChatGPT “一鸣惊人”，以强大的对话能力展示出令人惊讶的智能，一度燃起了 ChatGPT 是否可以通过“图灵测试”的讨论 [3]。此外，用户可以通过 OpenAI 提供的网页端或 API 轻松使用预训练后的 ChatGPT 模型，而无需在本地部署，标志着一种新的服务模式 LLMaaS (LLM as a Service) 的出现。但

<sup>22</sup><https://openai.com/blog/chatgpt>



是，从 ChatGPT 起，GPT 系列模型走向闭源，我们无从窥探 ChatGPT 及后续模型的技术细节。

四个月后，OpenAI 于 2023 年 3 月继续发布了 **GPT-4**<sup>23</sup> 模型。相较于 ChatGPT，GPT-4 在理解复杂语境、捕捉语言细微差别、生成连贯文本等任务上进一步提升，并且能够更有效地处理数学问题、编程挑战等高级认知任务。此外，GPT-4 还引入了对图文双模态的支持，扩展了其在图像描述和视觉问题解答等应用领域的可能性。

一年后，为了进一步提升模型性能以及用户体验，OpenAI 于 2024 年 5 月提出了 **GPT-4o**<sup>24</sup>。GPT-4o 模型在前代 GPT-4 的基础上，大幅提升了响应速度，显著降低了延迟，并且还增强了多模态处理能力以及多语言支持能力。其在客户支持、内容创作和数据分析等领域表现亮眼。GPT-4o 的推出标志着 AIGC 的应用日趋成熟。

GPT 系列模型的演进过程是人工智能发展史中一个激动人心的篇章。从 GPT-1 的“初出茅庐”到 GPT-4o 的“一鸣惊人”，短短 6 年时间，GPT 系列模型便带来了革命性的突破。诸多“科幻”变成现实，众多产业将被重塑。但是，随着 GPT 系列模型走向闭源，用户仅可以使用其功能，却无法参与到模型的共同创造和改进过程中。

### 2.5.3 LLaMA 系列语言模型

LLaMA (Large Language Model Meta AI) 是由 Meta AI 开发的一系列大语言模型，其模型权重在非商业许可证下向学术界开放，推动了大语言模型的“共创”和知识共享。在模型架构上，LLaMA 借鉴了 GPT 系列的设计理念，同时在技术细节上进行了创新和优化。LLaMA 与 GPT 系列的主要区别在于：GPT 系列的升级主

---

<sup>23</sup><https://openai.com/index/gpt-4-research>

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

线聚焦于模型规模与预训练语料的同步提升，而 LLaMA 则在模型规模上保持相对稳定，更专注于提升预训练数据的规模。表2.5展示了不同版本 LLaMA 模型对应的发布时间、参数量以及语料规模。当前，Meta AI 共推出三个版本的 LLaMA 模型。在这些模型的基础上，大量衍生模型陆续被推出。原始的 LLaMA 模型和其衍生模型一起构成了 LLaMA 生态系统。接下来将对 LLaMA 的三个版本及其部分衍生模型进行介绍。

表 2.5: LLaMA 系列模型参数和语料大小表。

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

## 1. LLaMA1 模型

LLaMA1[40] 是 Meta AI 于 2023 年 2 月推出的首个大语言模型。其在 Chin-chilla[15] 扩展法则的指引下，实践“小模型 + 大数据”的理念，旨在以大规模的优质数据训练相对较小的模型。相对较小的参数规模可以赋能更快的推理速度，使其可以更好的应对计算资源有限的场景。

在预训练语料方面，LLaMA1 的预训练数据涵盖了大规模网页数据集 Common Crawl<sup>25</sup>、T5[29] 提出的 C4 数据集，以及来自 Github、Wikipedia、Gutenberg、Books3、Arxiv 以及 StackExchange 等多种来源的数据，总数据量高达 5TB。

在模型架构方面，LLaMA1 采用了与 GPT 系列同样的网络架构。但是，其在 Transformer 原始词嵌入模块、注意力模块和全连接前馈模块上进行了优化。在词嵌入模块上，为了提高词嵌入质量，LLaMA1 参考了 GPTNeo[4] 的做法，使用**旋转位置编码** (Rotary Positional Embeddings, RoPE) [36] 替代了原有的绝对位置编码，从而增强位置编码的表达能力，增强了模型对序列顺序的理解。在注意力模

<sup>25</sup><https://commoncrawl.org>

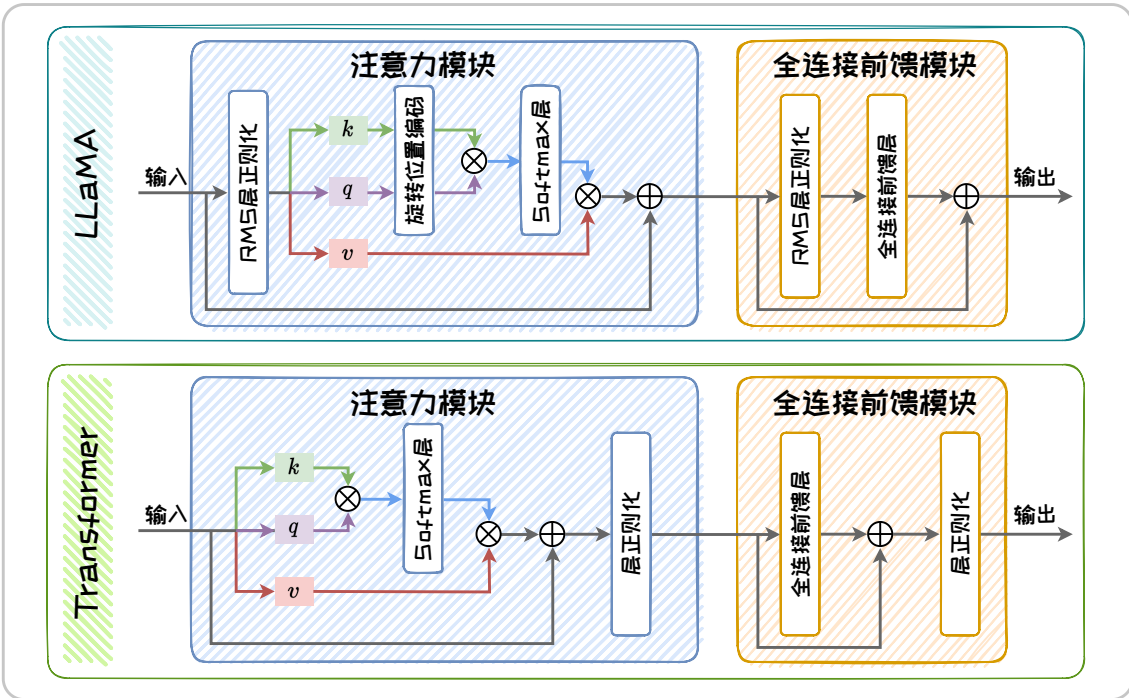


图 2.16: LLaMA 解码块架构与标准 Transformer 解码器架构对比。

块上，LLaMA1 参考了 PaLM[7] 的做法，将 Transformer 中的 RELU 激活函数改为 SwiGLU 激活函数 [34]。并且，LLaMA1 在进行自注意力操作之前对查询（query）以及键（key）添加旋转位置编码。在全连接前馈模块上，LLaMA1 借鉴了 GPT-3[5] 中的 **Pre-Norm** 层正则化策略，将正则化应用于自注意力和前馈网络的输入。在注意力模块和全连接前馈模块上的改进如图2.16所示。

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

表 2.6: LLaMA1 模型具体参数表。

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

## 2. LLaMA2 模型

2023 年 7 月，Meta AI 发布了 LLaMA 系列的第二代模型 LLaMA2[39]。秉承“小模型 + 大数据”的设计理念，LLaMA2 在 LLaMA1 的基础上进一步优化和扩充了训练数据，将语料库的规模扩展至约 7TB，实现了对更丰富语言和领域资源的覆盖。此外，在预训练阶段之后，LLaMA2 采纳了人类反馈强化学习的方法，进一步提升了模型的性能。首先，其使用了大规模且公开的指令微调数据集 [8] 对模型进行有监督的微调。然后，LLaMA2 还训练了 RLHF 奖励模型，并基于**近似策略优化**（Proximal Policy Optimization, PPO）[33] 以及**拒绝采样**（Rejection Sampling）进行强化学习对模型进行更新。

在模型架构上，LLaMA2 继承了 LLaMA1 的架构。LLaMA2 共推出了四个版本的模型，不同版本对应的具体参数如表 2.7 所示。其中，LLaMA2-34B 和 LLaMA2-70B 还额外增加了**分组查询注意力**（Grouped Query Attention, GQA）[1]，以提升计算效率。在分组查询注意力机制下，键（key）以及值（value）不再与查询（query）一一对应，而是一组查询共享相同的键和值，从而有效降低内存占用并减少模型总参数量。

表 2.7: LLaMA2 模型具体参数表。

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

## 3. LLaMA3 模型

2024 年 4 月，Meta AI 于 2024 年 4 月进一步推出了第三代模型 LLaMA3<sup>26</sup>。LLaMA3 挑选了规模高达 50TB 的预训练语料，是 LLaMA2 的 7 倍之多。这一语料

<sup>26</sup><https://ai.meta.com/blog/meta-llama-3>

库不仅包含丰富的代码数据以增强模型的逻辑推理能力，还涵盖了超过 5% 的非英文数据，覆盖 30 多种语言，显著扩展了模型的跨语言处理能力。此外，LLaMA3 还进行了与 LLaMA2 一样的人类反馈强化学习，这一策略已被证明能显著提升模型性能。最终 LLaMA3 的性能在多个评测指标上全面超越了前代模型。即便是参数量相对较少的 80 亿参数版本，也展现出了超越 LLaMA2 700 亿参数版本的卓越性能。而 700 亿参数版本的 LLaMA3，在多项任务上的性能更是超越了业界标杆 GPT-4 模型。

在模型架构上，LLaMA3 与前一代 LLaMA2 几乎完全相同，只是在分词 (tokenizer) 阶段，**将字典长度扩大了三倍**，极大提升了推理效率。这一改进减少了中文字符等语言元素被拆分为多个 Token 的情况，有效降低了总体 Token 数量，从而提高了模型处理语言的连贯性和准确性。另一方面，扩大的字典有助于减少对具有完整意义的语义单元进行分割，使模型在处理文本时可以更准确的捕捉词义和上下文，提高生成文本的流畅性和连贯性。LLaMA3 在其推出的 80 亿参数以及 700 亿参数版本上，均采用了分组查询注意力机制。这两个版本的模型参数与 LLaMA2 的对应版本保持高度一致，但在性能上实现了质的飞跃，充分证明了数据的力量。

### 4. LLaMA 衍生模型

LLaMA 模型的开源共享吸引了众多研究者在其基础上继续创作。研究者们或继续改进 LLaMA 模型的性能，或将 LLaMA 模型适配到垂直领域，或将 LLaMA 模型支持的数据模态进行扩充。众智众创为 LLaMA 营造出了一个多样的、充满活力的研究生态。下面对三类主流的 LLaMA 衍生模型进行简要介绍。

- **性能改进类模型**：一些研究者专注于通过微调继续提升 LLaMA 模型性能。

例如，Alpaca<sup>27</sup> 基于 GPT-3.5 生成的指令遵循样例数据对 LLaMA1 进行微调，以较小的模型规模实现了与 GPT-3.5 相媲美的性能。Vicuna<sup>28</sup> 模型则另辟蹊

<sup>27</sup><https://crfm.stanford.edu/2023/03/13/alpaca.html>

<sup>28</sup><https://lmsys.org/blog/2023-03-30-vicuna>



径，利用 ShareGPT 平台上累积的日常对话数据微调 LLaMA1 模型，进一步提升了它的对话能力。Guanaco[10] 模型则通过在微调 LLaMA1 的过程中引入 QLoRA 技术，显著降低了微调的时间成本，提高了微调效率。

- **垂域任务类模型**：尽管 LLaMA 在通用任务上表现出色，但在特定领域的应用潜力仍待挖掘。因此，大量研究者们针对垂直领域对 LLaMA 进行微调，以改善其在垂直领域上的表现。例如，CodeLLaMA[30] 模型在 LLaMA2 的基础上，利用大量公开代码数据进行微调，使其能更好地适应自动化代码生成、错误检测、以及代码优化等任务。LawGPT[24] 模型通过 30 万条法律问答对 LLaMA1 模型进行指令微调，显著增强了其对法律内容的处理能力。GOAT[21] 模型通过 Python 脚本生成的数学题库对 LLaMA1 模型进行微调，提高其解决各类数学题的准确率。Cornucopia<sup>29</sup>模型则利用金融问答数据进行微调，增强了金融问答的效果。
- **多模态任务类模型**：通过整合视觉模态编码器和跨模态对齐组件，研究者们将 LLaMA 模型扩展到多模态任务上。例如，LLaVA[19] 在 Vicuna 的基础上利用 CLIP 提取图像特征并利用一个线性投影层实现图片和文本之间的对齐。MiniGPT4[45] 在 Vicuna 的基础上使用 ViT-G/14 以及 Q-Former 作为图像编码器，并同样使用线性投影层来实现图片和文本之间的对齐，展现了多模态任务处理能力。

这些衍生模型不仅丰富了 LLaMA 模型的应用场景，也为自然语言处理领域的研究提供了新的方向和可能性。LLaMA 系列以其开源开放的态度，吸引全球研究者参与共创。我们有理由相信 LLaMA 系列模型携其衍生模型将绽放出满天繁星，照亮大语言模型前行之路。

---

<sup>29</sup><https://github.com/jerry1993-tech/Cornucopia-LLaMA-Fin-Chinese>

表 2.8: GPT 系列和 LLaMA 系列模型参数和语料大小表。

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

基于 Decoder-only 架构的大语言模型，凭借其卓越的生成能力，引领了新一轮生成式人工智能的浪潮。表2.8展示了 GPT 系列和 LLaMA 系列不同版本的具体参数，从中可以发现基于 Decoder-only 架构的模型在参数数量和预训练语料规模上急速增长。随着算力资源和数据资源的进一步丰富，基于 Decoder-only 架构的大语言模型必将释放出更为璀璨的光芒。

## 2.6 非 Transformer 架构

Transformer 结构是当前大语言模型的主流模型架构，其具备构建灵活、易并行、易扩展等优势。但是，Transformer 也并非完美。其并行输入的机制会导致模型规模随输入序列长度平方增长，导致其在处理长序列时面临计算瓶颈。为了提高计算效率和性能，解决 Transformer 在长序列处理中的瓶颈问题，可以选择基于 RNN 的语言模型。RNN 在生成输出时，只考虑之前的隐藏状态和当前输入，理论上可以处理无限长的序列。然而，传统的 RNN 模型（如 GRU、LSTM 等）在处理长序列时可能难以捕捉到长期依赖关系，且面临着梯度消失或爆炸问题。为了克服这些问题，近年来，研究者提出了两类现代 RNN 变体，分别为**状态空间模型**（State Space Model, SSM）和**测试时训练**（Test-Time Training, TTT）。这两种范式都可

以实现关于序列长度的线性时间复杂度，且避免了传统 RNN 中存在的问题。本节将对这两种范式及对应的代表性模型进行简要介绍。

## 2.6.1 状态空间模型 SSM

**状态空间模型**（State Space Model, SSM）[13] 范式可以有效处理长文本中存在的长程依赖性（Long-Range Dependencies, LRDs）问题，并且可以有效降低语言模型的计算和内存开销。本小节将首先介绍 SSM 范式，再分别介绍两种基于 SSM 范式的代表性模型：RWKV 和 Mamba。

### 1. SSM

SSM 的思想源自于控制理论中的动力系统。其通过利用一组状态变量来捕捉系统状态随时间的连续变化，这种连续时间的表示方法天然地适用于描述长时间范围内的依赖关系。此外，SSM 还具有递归和卷积的离散化表示形式，既能在推理时通过递归更新高效处理序列数据，又能在训练时通过卷积操作捕捉全局依赖关系。

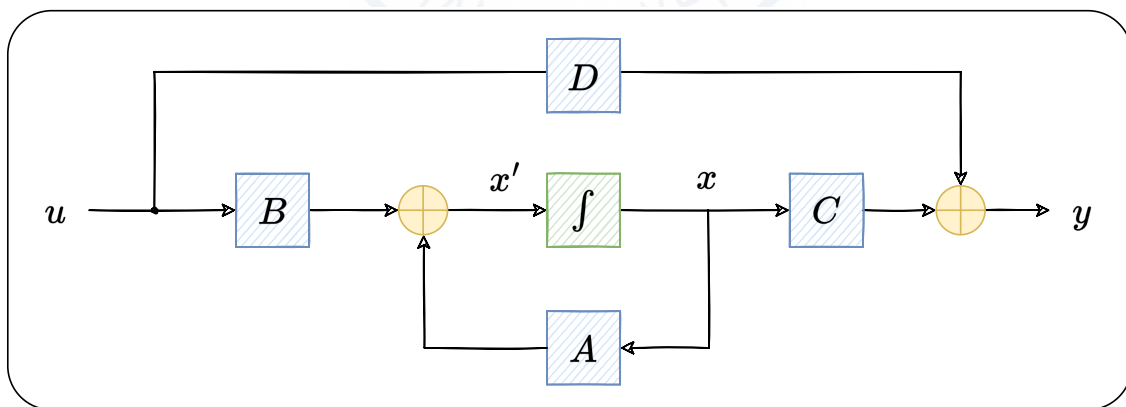


图 2.17: SSM 范式。

如图 2.17，SSM 在三个随时间  $t$  变化的变量和四个可学习的矩阵的基础上构造而成。三个变量分别为： $x(t) \in \mathbb{C}^n$  表示  $n$  个状态变量， $u(t) \in \mathbb{C}^m$  表示  $m$  个状

态输入,  $y(t) \in \mathbb{C}^p$  表示  $p$  个输出。四个矩阵分别为: 状态矩阵  $\mathbf{A} \in \mathbb{C}^{n \times n}$ , 控制矩阵  $\mathbf{B} \in \mathbb{C}^{n \times m}$ , 输出矩阵  $\mathbf{C} \in \mathbb{C}^{p \times n}$  和命令矩阵  $\mathbf{D} \in \mathbb{C}^{p \times m}$ 。SSM 的系统方程为:

$$\begin{aligned} x'(t) &= \mathbf{A}x(t) + \mathbf{B}u(t) \\ y(t) &= \mathbf{C}x(t) + \mathbf{D}u(t), \end{aligned} \quad (2.5)$$

其中,  $x'(t) = \mathbf{A}x(t) + \mathbf{B}u(t)$  为状态方程, 描述了系统状态如何基于输入和前一个状态变化, 其计算出的是状态关于时间的导数  $x'(t)$ , 为了得到状态  $x(t)$ , 还需对其进行积分操作。  $y(t) = \mathbf{C}x(t) + \mathbf{D}u(t)$  为输出方程, 描述了系统状态如何转化为输出, 其中的  $x(t)$  是通过状态方程更新且积分后的值。在深度学习中,  $\mathbf{D}u(t)$  项表示残差连接, 可被忽略。

该方程可视作 SSM 系统方程的连续形式, 适用于对连续数据 (例如音频信号、时间序列) 的处理, 但是在训练和推理都非常慢。为了提高对 SSM 的处理效率, 需要对该方程进行离散化操作。**离散化 (Discretization)** 是 SSM 中最为关键的步骤, 能够将系统方程从连续形式转换为递归形式和卷积形式, 从而提升整个 SSM 架构的效率。将连续形式的 SSM 系统方程离散化时, 可以使用梯形法代替连续形式中的积分操作, 其原理是将定义在特定区间上的函数曲线下的区域视为梯形, 并利用梯形面积公式计算其面积。由此, 可以得出离散化后递归形式下的系统方程:

$$\begin{aligned} x_k &= \bar{\mathbf{A}}x_{k-1} + \bar{\mathbf{B}}u_k \\ y_k &= \bar{\mathbf{C}}x_k. \end{aligned} \quad (2.6)$$

在该方程中, 状态方程由前一步的状态和当前输入计算当前状态, 体现了递归的思想。其中,  $\bar{\mathbf{A}}, \bar{\mathbf{B}}, \bar{\mathbf{C}}$  为离散形式下的矩阵, 其与连续形式下矩阵  $\mathbf{A}, \mathbf{B}, \mathbf{C}$  的关系分别表示为:  $\bar{\mathbf{A}} = (\mathbf{I} - \frac{\Delta}{2}\mathbf{A})^{-1}(\mathbf{I} + \frac{\Delta}{2}\mathbf{A})$ ,  $\bar{\mathbf{B}} = (\mathbf{I} - \frac{\Delta}{2}\mathbf{A})^{-1}\Delta\mathbf{B}$ ,  $\bar{\mathbf{C}} = \mathbf{C}$ , 其中  $\Delta = t_{n+1} - t_n$ 。

递归形式的 SSM 类似于 RNN, 具有 RNN 的优缺点。其适用于顺序数据的处理, 能够实现与序列长度呈线性复杂度的高效推理, 但是无法并行训练, 当面临长

序列时存在梯度消失或爆炸问题。将系统方程的递归形式进行迭代，可以得到卷积形式。这里省略了推导过程，直接给出迭代后  $x_k$  和  $y_k$  的结果：

$$\begin{aligned} x_k &= \bar{\mathbf{A}}^k \bar{\mathbf{B}} u_0 + \bar{\mathbf{A}}^{k-1} \bar{\mathbf{B}} u_1 + \cdots + \bar{\mathbf{B}} u_k \\ y_k &= \bar{\mathbf{C}} x_k = \bar{\mathbf{C}} \bar{\mathbf{A}}^k \bar{\mathbf{B}} u_0 + \bar{\mathbf{C}} \bar{\mathbf{A}}^{k-1} \bar{\mathbf{B}} u_1 + \cdots + \bar{\mathbf{C}} \bar{\mathbf{B}} u_k. \end{aligned} \quad (2.7)$$

可以观察到，将系统方程的递归形式迭代展开后，输出  $y_k$  是状态输入  $u_k$  的卷积结果，其卷积核为：

$$\bar{\mathbf{K}}_k = (\bar{\mathbf{C}} \bar{\mathbf{B}}, \bar{\mathbf{C}} \bar{\mathbf{A}} \bar{\mathbf{B}}, \dots, \bar{\mathbf{C}} \bar{\mathbf{A}}^k \bar{\mathbf{B}}). \quad (2.8)$$

因此，SSM 系统方程的卷积形式为：

$$y_k = \bar{\mathbf{K}}_k * u_k, \quad (2.9)$$

其中，卷积核是由 SSM 中的矩阵参数决定的，由于这些参数在整个序列的处理过程中是固定的，被称为**时不变性**。时不变性使得 SSM 能够一致地处理不同时间步长的数据，进行高效的并行化训练。但由于上下文长度固定，卷积形式的 SSM 在进行自回归任务时延迟长且计算消耗大。结合离散化后 SSM 的递归形式和卷积形式的优缺点，可以选择在**训练时使用卷积形式，推理时使用递归形式**。

综上，SSM 架构的系统方程具有三种形式，分别为连续形式、离散化的递归形式以及离散化的卷积形式，可应用于文本、视觉、音频和时间序列等任务，在应用时，需要根据具体情况选择合适的表示形式。SSM 的优势在于能够处理非常长的序列，虽然比其它模型参数更少，但在处理长序列时仍然可以保持较快的速度。

当前，各种现有 SSM 架构之间的主要区别在于基本 SSM 方程的离散化方式或  $\mathbf{A}$  矩阵的定义。例如，S4[14] (Structured State Space Model) 是一种 SSM 变体，其关键创新是使用 HiPPO 矩阵来初始化  $\mathbf{A}$  矩阵，在处理长序列数据时表现优异。此外，RWKV 和 Mamba 是两种基于 SSM 范式的经典架构，下面将分别介绍这两种架构。



2. RWKV

RWKV (Receptance Weighted Key Value) [26] 是基于 SSM 范式的创新架构, 其核心机制 WKV 的计算可以看作是两个 SSM 的比。RWKV 的设计结合了 RNNs 和 Transformers 的优点, 既保留了推理阶段的高效性, 又实现了训练阶段的并行化。(注: 这里讨论的是 RWKV-v4)

RWKV 模型的核心模块有两个: **时间混合模块**和 **通道混合模块**。时间混合模块主要处理序列中不同时间步之间的关系, 通道混合模块则关注同一时间步内不同特征通道<sup>30</sup>之间的交互。时间混合模块和通道混合模块的设计基于四个基本元素: **接收向量  $R$** 、**键向量  $K$** 、**值向量  $V$**  和 **权重  $W$** , 下面将根据图 2.18, 介绍这些元素在两个模块中的计算流程和作用。

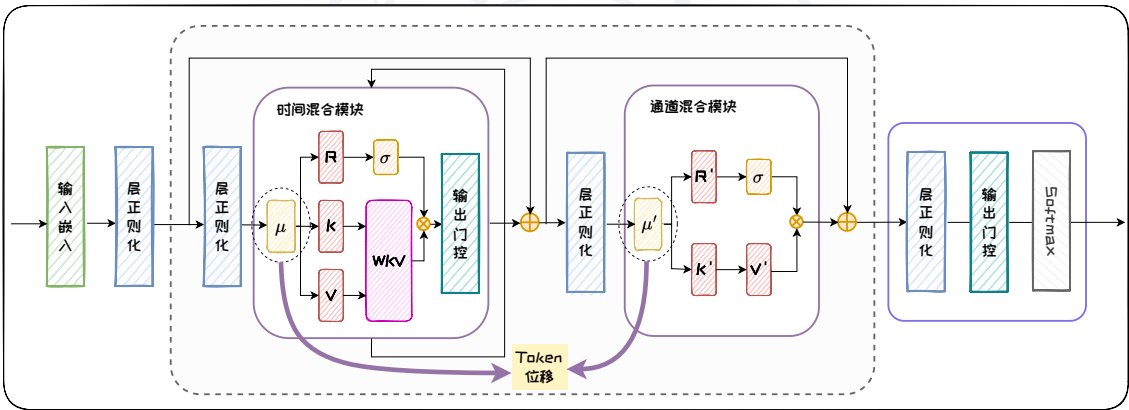


图 2.18: RWKV 架构。

时间混合模块和通道混合模块中共有的操作是 Token 位移, 该步通过对当前时间步和前一时间步的输入进行线性插值来实现, 从而确保了模型对序列中时间变化的敏感性。在时间混合模块中, 接受向量  $R$  负责接收并整合来自序列历史的信息, 权重  $W$  表示位置权重衰减, 键向量  $K$  和值向量  $V$  类似传统注意力机制中的键和值, 分别用于匹配和携带信息。时间混合模块首先将当前步长和前一步长的

<sup>30</sup>在这里, **通道**指的是在神经网络中同一时间步内的不同特征维度。例如, 在处理自然语言处理任务时, 每个时间步的输入可能是一段文本的一个词, 而每个词会通过嵌入层转化为一个向量, 这个向量的每个元素就代表一个特征通道, 向量维数就是通道数量。

输入进行线性组合，通过线性投影得到  $R$ 、 $K$ 、 $V$  向量；随后，通过 **WKV** 机制来确保每个通道的权重随时间推移逐步衰减；最后，将表示过去信息的  $\sigma(R)$  和表示当前信息的 **WKV** 向量通过输出门控进行整合，传递给通道混合模块。

时间混合模块中的 **WKV 机制** 是 **RWKV** 的核心部分。在 **RWKV** 中，权重  $W$  是一个与通道相关的时间衰减向量，该向量可以表示为： $w_{t,i} = -(t-i)w$ ， $i$  表示从当前时间步长  $t$  向后追溯的某个时间步长， $w$  是一个非负向量，其长度为通道数。通过这种方式，模型能够捕捉时间序列中不同时间步长之间的依赖关系，实现每个通道权重随时间向后逐步衰减的效果。**WKV** 机制的关键在于利用线性时不变递归来更新隐藏状态，这可以看作是两个 **SSM** 之比。具体公式为：

$$wkv_t = \frac{\sum_{i=1}^{t-1} e^{-(t-1-i)w+k_i} \odot v_i + e^{u+k_t} \odot v_t}{\sum_{i=1}^{t-1} e^{-(t-1-i)w+k_i} + e^{u+k_t}}, \quad (2.10)$$

其中， $u$  是一个单独关注当前 Token 的向量。由于分子和分母都表现出状态更新和输出的过程，与 **SSM** 思想类似，因此该公式可以看作是两个 **SSM** 之比。通过权重  $W$  和 **WKV** 机制，模块能够有效地处理长时间序列数据，并减少梯度消失问题。

在通道混合模块中， $R'$ 、 $K'$ 、 $V'$  的作用与时间混合模块类似， $R'$  和  $K'$  同样由输入的线性投影得到， $V'$  的更新则额外依赖于  $K'$ 。之后，将  $\sigma(R')$  和  $V'$  整合，以实现不同通道之间的信息交互和融合。

此外，**RWKV** 架构还采用了时间依赖的 **Softmax** 操作，提高数值稳定性和梯度传播效率，以及采用层归一化来稳定梯度，防止梯度消失和爆炸。为了进一步提升性能，**RWKV** 还采用了自定义 **CUDA** 内核、小值初始化嵌入以及自定义初始化等优化措施。

**RWKV** 在模型规模、计算效率和模型性能方面都表现可观。在**模型规模**方面，**RWKV** 模型参数扩展到了 14B，是第一个可扩展到数百亿参数的非 Transformer 架构。在**计算效率**方面，**RWKV** 允许模型被表示为 Transformer 或 RNN，从而使得模型在训练时可以并行化计算，在推理时保持恒定的计算和内存复杂度。在**模型性**

能方面, RWKV 在 NLP 任务上的性能与类似规模的 Transformer 相当, 在长上下文基准测试中的性能仅次于 S4。

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

### 3. Mamba

时不变性使得 SSM 能够一致地处理不同时间步长的数据, 进行高效的并行化训练, 但是同时也导致其处理信息密集的数据 (如文本) 的能力较弱。为了弥补这一不足, Mamba [12] 基于 SSM 架构, 提出了**选择机制** (Selection Mechanism) 和**硬件感知算法** (Hardware-aware Algorithm), 前者使模型执行基于内容的推理, 后者实现了在 GPU 上的高效计算, 从而同时保证了**快速训练和推理**、**高质量数据生成**以及**长序列处理能力**。

Mamba 的**选择机制**通过动态调整模型参数来选择需要关注的信息, 使模型参数能够根据输入数据动态变化。具体来说, Mamba 将离散化 SSM 中的参数  $\mathbf{B}, \mathbf{C}, \Delta$  分别转变成以下函数:  $s_{\mathbf{B}}(x) = \text{Linear}_N(x)$ 、 $s_{\mathbf{C}}(x) = \text{Linear}_N(x)$ 、 $s_{\Delta}(x) = \text{Broadcast}_D(\text{Linear}_1(x))$ , 并采用非线性激活函数  $\tau_{\Delta} = \text{softplus}$  来调节参数  $\Delta$ 。其中  $\text{Linear}_d$  是对特征维数  $d$  的参数化投影,  $s_{\Delta}$  和  $\tau_{\Delta}$  函数的选择与 RNN 的门控机制相关联 [38]。此外, Mamba 还对张量形状进行相应调整, 使模型参数具有时间维度, 这意味着模型参数矩阵在每个时间步都有不同的值, 从时间不变转变为时间变化的。

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

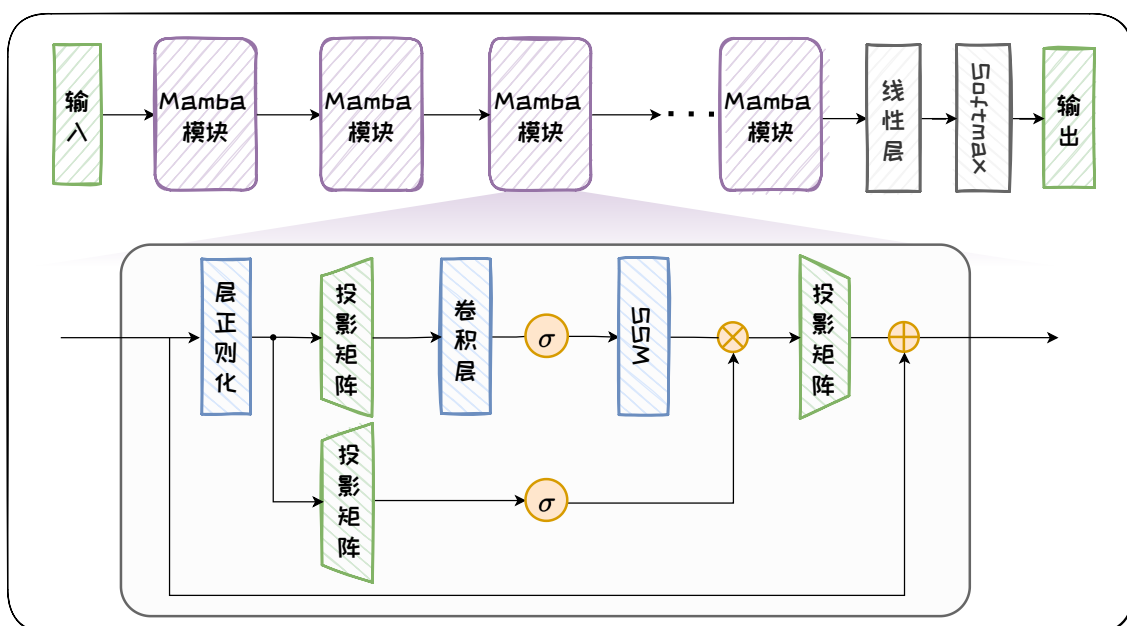


图 2.19: Mamba 架构。

并行扫描和重计算三方面内容。**内核融合**通过减少内存 I/O 操作来提高速度。**并行扫描**利用并行化算法提高效率。**重计算**则在反向传播时重新计算中间状态，以减少内存需求。

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

Mamba 通过将带有选择机制的 SSM 模块与 Transformer 的前馈层相结合，形成了一个简单且同质的架构设计。如图 2.19 所示，Mamba 架构是由完全相同的 Mamba 模块组成的递归模型，每个 Mamba 模块都在前馈层中插入了卷积层和带有选择机制的 SSM 模块，其中激活函数  $\sigma$  选用 SiLU / Swish 激活。

通过引入选择机制和硬件感知算法，Mamba 在实际应用中展示了卓越的性能和效率，包括：（1）**快速训练和推理**：训练时，计算和内存需求随着序列长度线性增长，而推理时，每一步只需常数时间，不需要保存之前的所有信息。通过硬件

感知算法，Mamba 不仅在理论上实现了序列长度的线性扩展，而且在 A100 GPU 上，其推理吞吐量比类似规模的 Transformer 提高了 5 倍。(2) **高质量数据生成**：在语言建模、基因组学、音频、合成任务等多个模态和设置上，Mamba 均表现出色。在语言建模方面，Mamba-3B 模型在预训练和后续评估中性能超过了两倍参数量的 Transformer 模型性能。(3) **长序列处理能力**：Mamba 能够处理长达百万级别的序列长度，展示了处理长上下文时的优越性。

虽然 Mamba 在硬件依赖性和模型复杂度上存在一定的局限性，但是它通过引入选择机制和硬件感知算法显著提高了处理长序列和信息密集数据的效率，展示了在多个领域应用的巨大潜力。Mamba 在多种应用上的出色表现，使其成为一种理想的通用基础模型。

### 2.6.2 训练时更新 TTT

在处理长上下文序列时，上述基于 SSM 范式的架构（例如 RWKV 和 Mamba）通过将上下文信息压缩到固定长度的隐藏状态中，成功将计算复杂度降低至线性级别，有效扩展了模型处理长上下文的能力。然而，随着上下文长度的持续增长，基于 SSM 范式的模型可能会过早出现性能饱和。例如，Mamba 在上下文长度超过 16k 时，困惑度基本不再下降 [37]。出现这一现象的原因可能是固定长度的隐藏状态限制了模型的表达能力，同时在压缩过程中可能会导致关键信息的遗忘。

为了解决这一限制，测试时训练 (Test-Time Training, TTT) [37] 范式提供了一种有效的解决方案。TTT 利用模型本身的参数来存储隐藏状态、记忆上文；并在每一步推理中，对模型参数进行梯度更新，已实现上文的不断循环流入，如图 2.20 所示。这个过程不同于传统的机器学习范式中模型在完成训练后的推理阶段通常保持静态的方式，TTT 在推理阶段会针对每一条测试数据一边循环训练一边推理。为了实现这种测试时训练的机制，TTT 在预训练和推理阶段均进行了独特的设计。



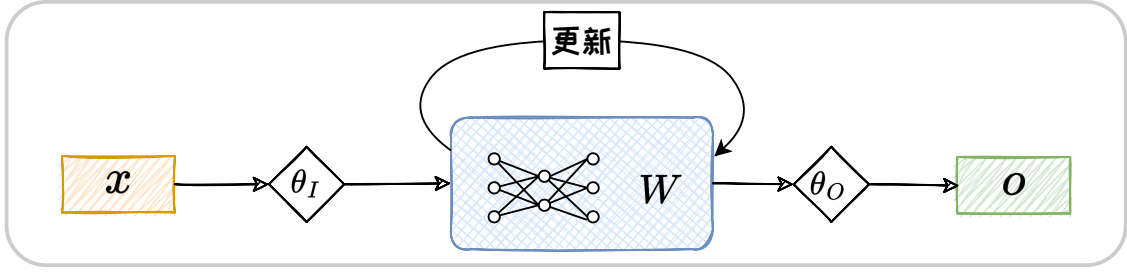


图 2.20: TTT 范式下的推理流程。

在 TTT 范式的预训练阶段，训练过程包含内部循环以及外部循环两个部分。其中外部循环遵循传统的下词预测任务，通过自回归方式优化模型全局权重参数。内部循环则是基于自监督的方式来优化隐藏状态。具体来说，模型需要在每个时间步动态地更新隐藏状态，使其能够不断适应新的输入数据。这种动态更新的机制类似于一个独立的机器学习模型在每个时间步对输入进行训练和优化。给定当前时间步输入  $x_t$  以及先前历史上下文  $x_1, x_2, \dots, x_{t-1}$  对应的隐藏状态  $W_{t-1}$ ，模型计算当前时间步的重构损失：

$$\ell(W_{t-1}; x_t) = \|f(\theta_K x_t; W_{t-1}) - \theta_V x_t\|^2, \quad (2.11)$$

其中  $\theta_K$  和  $\theta_V$  是通过外部循环学习到的参数。接着，模型以学习率  $\eta$  利用该损失进行梯度下降，更新隐藏状态：

$$W_t = W_{t-1} - \eta \nabla \ell(W_{t-1}; x_t). \quad (2.12)$$

最终，模型基于更新后的隐藏状态和当前输入生成输出：

$$z_t = f(x_t; W_t). \quad (2.13)$$

在推理阶段，无需执行外部循环任务。因此，模型只进行内部循环来对隐藏状态进行更新，使模型更好地适应新的数据分布，从而提升预测性能。

与 Transformer 相比，基于 TTT 范式的模型具有线性时间复杂度，这对于处理长序列数据至关重要。相较于基于 SSM 的 RWKV 和 Mamba 架构，TTT 通过模型

参数来保存上下文信息，能够更有效地捕捉超长上下文中的语义联系和结构信息。因此，TTT 在长上下文建模任务中展现出卓越的性能，特别是在需要处理超长上下文的应用场景中。未来，TTT 范式有望在超长序列处理任务中发挥重要作用。

## 参考文献

- [1] Joshua Ainslie et al. “Gqa: Training generalized multi-query transformer models from multi-head checkpoints”. In: *arXiv preprint arXiv:2305.13245* (2023).
- [2] Rohan Anil et al. “Palm 2 technical report”. In: *arXiv preprint arXiv:2305.10403* (2023).
- [3] Tim Bayne and Iwan Williams. “The Turing test is not a good benchmark for thought in LLMs”. In: *Nature Human Behaviour* 7.11 (2023), pp. 1806–1807.
- [4] Sid Black et al. “Gpt-neox-20b: An open-source autoregressive language model”. In: *arXiv preprint arXiv:2204.06745* (2022).
- [5] Tom Brown et al. “Language models are few-shot learners”. In: *NeurIPS*. 2020.
- [6] Mark Chen et al. “Evaluating large language models trained on code”. In: *arXiv preprint arXiv:2107.03374* (2021).
- [7] Aakanksha Chowdhery et al. “Palm: Scaling language modeling with pathways”. In: *Journal of Machine Learning Research* 24.240 (2023), pp. 1–113.
- [8] Hyung Won Chung et al. “Scaling instruction-finetuned language models”. In: *Journal of Machine Learning Research* 25.70 (2024), pp. 1–53.
- [9] Kevin Clark et al. “Electra: Pre-training text encoders as discriminators rather than generators”. In: *arXiv preprint arXiv:2003.10555* (2020).
- [10] Tim Dettmers et al. “Qlora: Efficient finetuning of quantized llms”. In: *NeurIPS*. 2024.
- [11] Jacob Devlin et al. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. In: *NAACL*. 2019.
- [12] Albert Gu and Tri Dao. “Mamba: Linear-Time Sequence Modeling with Selective State Spaces”. In: *arXiv preprint arXiv:2312.00752* (2023).
- [13] Albert Gu, Karan Goel, and Christopher Ré. “Efficiently modeling long sequences with structured state spaces”. In: *arXiv preprint arXiv:2111.00396* (2021).

- [14] Albert Gu et al. “On the Parameterization and Initialization of Diagonal State Space Models”. In: *NeurIPS*. 2022.
- [15] Jordan Hoffmann et al. “Training compute-optimal large language models”. In: *arXiv preprint arXiv:2203.15556* (2022).
- [16] Jared Kaplan et al. “Scaling laws for neural language models”. In: *arXiv preprint arXiv:2001.08361* (2020).
- [17] Zhenzhong Lan et al. “Albert: A lite bert for self-supervised learning of language representations”. In: *arXiv preprint arXiv:1909.11942* (2019).
- [18] Mike Lewis et al. “Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension”. In: *ACL*. 2020.
- [19] Haotian Liu et al. “Visual instruction tuning”. In: *NeurIPS*. 2024.
- [20] Qi Liu, Matt J Kusner, and Phil Blunsom. “A survey on contextual embeddings”. In: *arXiv preprint arXiv:2003.07278* (2020).
- [21] Tiedong Liu and Bryan Kian Hsiang Low. “Goat: Fine-tuned llama outperforms gpt-4 on arithmetic tasks”. In: *arXiv preprint arXiv:2305.14201* (2023).
- [22] Yinhan Liu et al. “Multilingual denoising pre-training for neural machine translation”. In: *Transactions of the Association for Computational Linguistics* 8 (2020), pp. 726–742.
- [23] Yinhan Liu et al. “Roberta: A robustly optimized bert pretraining approach”. In: *arXiv preprint arXiv:1907.11692* (2019).
- [24] Ha-Thanh Nguyen. “A brief report on lawgpt 1.0: A virtual legal assistant based on gpt-3”. In: *arXiv preprint arXiv:2302.05729* (2023).
- [25] Long Ouyang et al. “Training language models to follow instructions with human feedback”. In: *NeurIPS*. 2022.
- [26] Bo Peng et al. “RWKV: Reinventing RNNs for the Transformer Era”. In: *EMNLP*. 2023.
- [27] Alec Radford et al. “Improving language understanding by generative pre-training”. In: (2018).
- [28] Rafael Rafailov et al. “Direct preference optimization: Your language model is secretly a reward model”. In: *NeurIPS*. 2024.

- [29] Colin Raffel et al. “Exploring the limits of transfer learning with a unified text-to-text transformer”. In: *Journal of machine learning research* 21.140 (2020), pp. 1–67.
- [30] Baptiste Roziere et al. “Code llama: Open foundation models for code”. In: *arXiv preprint arXiv:2308.12950* (2023).
- [31] Victor Sanh et al. “Multitask prompted training enables zero-shot task generalization”. In: *arXiv preprint arXiv:2110.08207* (2021).
- [32] Rylan Schaeffer, Brando Miranda, and Sanmi Koyejo. “Are emergent abilities of large language models a mirage?” In: *NeurIPS*. 2024.
- [33] John Schulman et al. “Proximal policy optimization algorithms”. In: *arXiv preprint arXiv:1707.06347* (2017).
- [34] Noam Shazeer. “Glu variants improve transformer”. In: *arXiv preprint arXiv:2002.05202* (2020).
- [35] Shaden Smith et al. “Using deepspeed and megatron to train megatron-turing nlG 530b, a large-scale generative language model”. In: *arXiv preprint arXiv:2201.11990* (2022).
- [36] Jianlin Su et al. “Roformer: Enhanced transformer with rotary position embedding”. In: *arXiv preprint arXiv:2104.09864* (2021).
- [37] Yu Sun et al. “Learning to (Learn at Test Time): RNNs with Expressive Hidden States”. In: *arXiv preprint arXiv:2407.04620* (2024).
- [38] Corentin Tallec and Yann Ollivier. “Can recurrent neural networks warp time?” In: *ICLR*. 2018.
- [39] Hugo Touvron et al. “Llama 2: Open foundation and fine-tuned chat models”. In: *arXiv preprint arXiv:2307.09288* (2023).
- [40] Hugo Touvron et al. “Llama: Open and efficient foundation language models”. In: *arXiv preprint arXiv:2302.13971* (2023).
- [41] Trieu H Trinh and Quoc V Le. “A simple method for commonsense reasoning”. In: *arXiv preprint arXiv:1806.02847* (2018).
- [42] Ashish Vaswani et al. “Attention is all you need”. In: *NeurIPS*. 2017.
- [43] Linting Xue et al. “mT5: A massively multilingual pre-trained text-to-text transformer”. In: *NAACL*. 2021.

- [44] Aiyuan Yang et al. “Baichuan 2: Open large-scale language models”. In: *arXiv preprint arXiv:2309.10305* (2023).
- [45] Deyao Zhu et al. “Minigpt-4: Enhancing vision-language understanding with advanced large language models”. In: *arXiv preprint arXiv:2304.10592* (2023).
- [46] Yukun Zhu et al. “Aligning books and movies: Towards story-like visual explanations by watching movies and reading books”. In: *ICCV*. 2015.

