大语言模型算法演进综述

Review of Evolution of Large Language Model Algorithms

朱炫鹏/ZHU Xuanpeng，姚海东/YAO Haidong，刘隽/LIU Jun，熊先奎/XIONG Xiankui

（中兴通讯股份有限公司，中国 深圳 518057）
(ZTE Corporation, Shenzhen 518057, China)

DOI：10.12142/ZTETJ.202402003
网络出版地址：http://kns.cnki.net/kcms/detail/34.1228.TN.20240422.2005.004.html
网络出版日期：2024-04-23
收稿日期：2024-03-02

摘要：基于Transformer架构的大语言模型展现出强大的能力，是人类迈向通用人工智能（AGI）的一个重大进步。大语言模型架构和算法的演进分为提高推理效率、提高模型能力两条技术路线。介绍了两条技术路线主流的技术方案和思路。提高推理效率的方法有分布式推理、计算优化、访存优化、量化等；提高模型能力主要是引入新的架构，如混合专家（MoE）模型、状态空间模型（SSM）等。

关键词：大语言模型；Transformer；注意力

Abstract: The large language model based on the Transformer architecture shows powerful capabilities, and it is a major progress towards artificial general intelligence (AGI). The evolution of large language model architecture and algorithms is divided into two technical paths: improving the inference efficiency and model capability. The mainstream technical solutions and ideas for the two technical routes are described. Methods for improving inference efficiency include distributed inference, computing optimization, memory access optimization, and quantification. To improve model capabilities, new architectures such as mixture of experts (MoE) and state space model (SSM) are introduced.

Keywords: large language model; Transformer; attention

引用格式：朱炫鹏, 姚海东, 刘隽, 等. 大语言模型算法演进综述 [J]. 中兴通讯技术, 2024, 30(2): 9-20. DOI: 10.12142/ZTETJ.202402003
Citation: ZHU X P, YAO H D, LIU J, et al. Review of evolution of large language model algorithms [J]. ZTE technology journal, 2024, 30(2): 9-20. DOI: 10.12142/ZTETJ.202402003

1 大语言模型算法发展概况

OpenAI于2022年、2023年分别发布ChatGPT[1]和GPT-4[2]，其强大的会话能力、多模态能力震惊业界，是人类迈向通用人工智能（AGI）的一个重大进步。ChatGPT和GPT-4能力强大的原因有两个：一是Transformer[3]架构的自注意力机制，可获取任意距离间单词的相关信息；二是大模型、大数据、大算力，规模超过了一定阈值，则会产生涌现能力[4]。

目前各大公司都发布了自己的大语言模型（LLM）。本文中，我们主要介绍大语言模型在两条技术路线上的架构和算法的演进。

1.1 语言模型的发展历程

语言模型的发展经历了统计语言模型、神经语言模型、预训练语言模型和大语言模型4个阶段[5]。其结构从基于统计概率发展到基于神经网络，模型复杂度不断增加，能力也出现了质的提升。

1） 统计语言模型

最初的语言模型是基于统计概率的，即根据语料统计出在某个上下文出现某个词的概率，根据概率选择最合适的词。

2） 神经语言模型

文献[6]首次将神经网络引入语言模型。常见的模型结构有循环神经网络（RNN）[7]、长短期记忆网络（LSTM）[8]等。RNN用隐藏层保存逐个输入的词的信息，但由于梯度消失和梯度爆炸，只能保留短期信息。LSTM使用门控机制，可以选择性地保留长期信息。

3） 预训练语言模型

ELMo[9]用预训练的双向LSTM网络根据上下文动态生成词向量，解决了一词多义问题。双向LSTM网络可以在下游任务上微调，得到更好的效果。基于Transformer的双向编码器表征法（BERT）[10]也采用了预训练+下游任务微调的范式。

4） 大语言模型

预训练语言模型的性能随着规模的增大而提高，成幂律关系[11-12]。OpenAI设计了大型语言模型GPT-3[13]。该模型表现出强大的能力，性能和规模超越了幂律关系，出现了涌现

现象，如上下文学习、思维链推理。

1.2 大语言模型算法演进路线

大语言模型的发展主要有两条技术路线：一是提高推理效率，降低推理成本；二是提高模型能力，迈向AGI。

大语言模型能力强大，有广阔的应用前景，各厂商都在积极部署，提供服务。但是，由于模型规模巨大，算法对硬件不够友好，需要消耗大量的算力、存储、能源。因此，如何降低推理成本、推理延时，是一个亟待解决的问题。大语言模型主要的技术路线有分布式推理、减小模型计算量、减小模型访存量、提升硬件亲和性等。

大语言模型是迈向AGI的重大进步，而Transformer是其中的核心架构，发挥了重大作用。但Transformer也有一定的不足，如计算量大，通过提升规模来提升性能更加困难；上下文窗口长度有限，难以支持超长序列。研究人员通过引入新的结构，解决这些问题，取得了较好的效果。

2 大语言模型架构

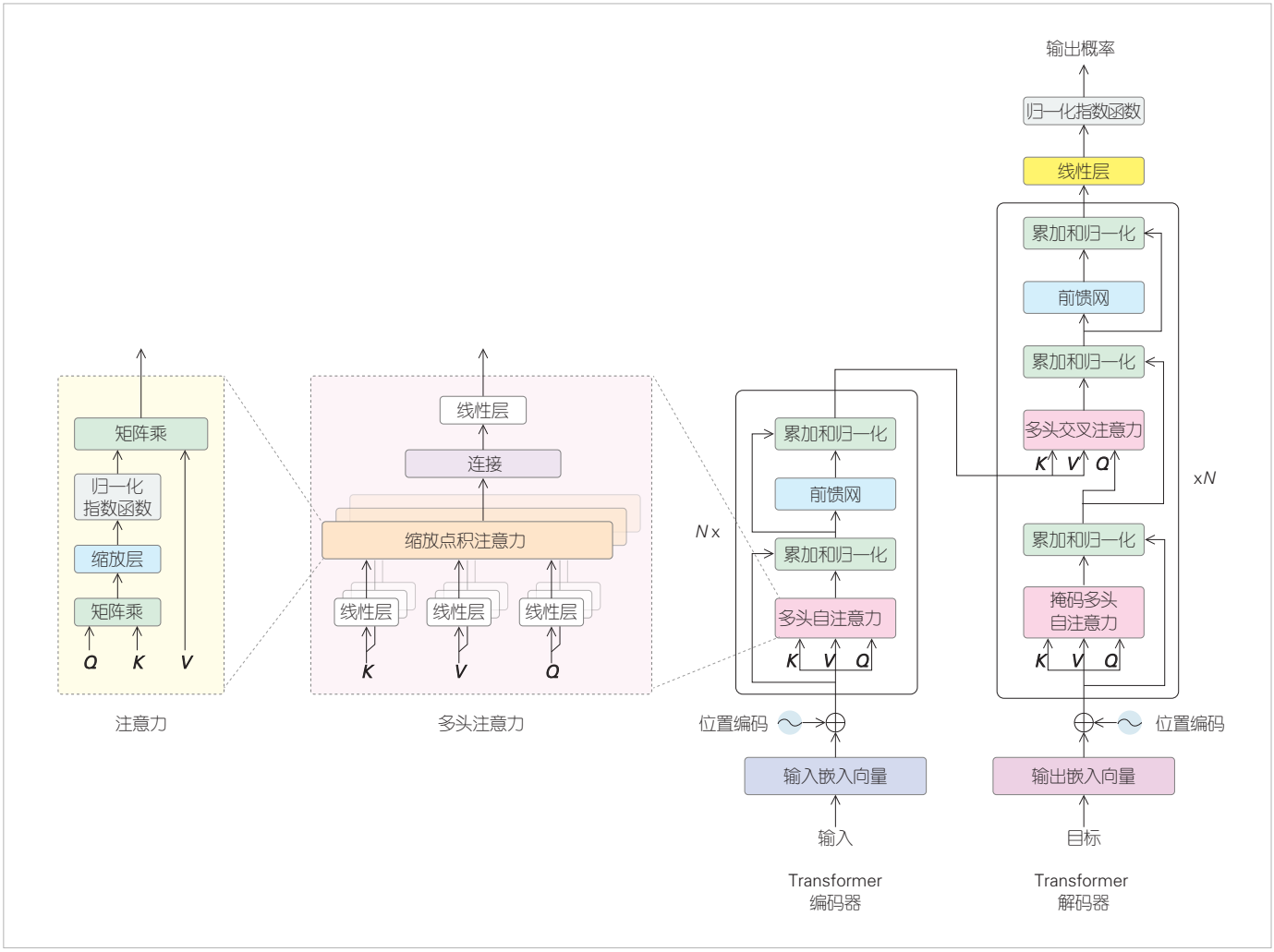
2.1 Transformer

Transformer模块是组成大语言模型的基础单元，由多头注意力、前馈网、Softmax、LayerNorm等部分组成，本节中我们主要介绍Transformer的结构和算法。

2.1.1 注意力机制

注意力机制是针对一个文本序列，计算每个token（符号）与其他tokens之间的相关系数，找出相关度高的tokens，用于生成特征。例如，“这是一只猫，它很可爱。”在这句话里，“猫”与“这”“它”的相关度会比较高。

注意力机制是基于查询-键-值（QKV）计算的。具体算法为：输入$\boldsymbol{Q}$、$\boldsymbol{K}$、$\boldsymbol{V}$，用$\boldsymbol{Q}$和$\boldsymbol{K}^T$做矩阵乘，除以$\sqrt{D_k}$，做Softmax，得到注意力矩阵$\boldsymbol{A}$；$\boldsymbol{A}$和$\boldsymbol{V}$做矩阵乘，得到输出特征。具体的公式如下，结构见图1。



▲图1 Transformer架构[3]

$$\text{Attention}(\boldsymbol{Q},\boldsymbol{K},\boldsymbol{V})=\text{Softmax}(\frac{\boldsymbol{Q}\boldsymbol{K}^{\mathrm{T}}}{\sqrt{D_k}})\boldsymbol{V}=\boldsymbol{A}\boldsymbol{V}, \tag{1}$$

其中，$\boldsymbol{Q}\in\mathrm{R}^{N\times D_k}$，$\boldsymbol{K}\in\mathrm{R}^{M\times D_k}$，$\boldsymbol{V}\in\mathrm{R}^{M\times D_v}$，$N$是$\boldsymbol{Q}$的长度，$M$是$\boldsymbol{K}$的长度，$D_k$是$\boldsymbol{K}$的向量维度，$D_v$是$\boldsymbol{V}$的向量维度。

注意力机制能够并行计算所有tokens间的相关信息，没有距离的限制，与RNN、LSTM相比更具有优势。

2.1.2 多头注意力

多头注意力（MHA）是将$\boldsymbol{Q}$、$\boldsymbol{K}$、$\boldsymbol{V}$转换成多份，每份单独计算注意力，结果合并在一起。每一份称为一个头，多个头可以计算不同领域中的相关关系，增加模型的信息容量和能力，具体如公式（2），结构见图1。

$$\text{MultiHeadAttn}(\boldsymbol{Q},\boldsymbol{K},\boldsymbol{V})=\text{Concat}(\boldsymbol{head}_1,\cdots,\boldsymbol{head}_H)\boldsymbol{W}^O,$$

$$\boldsymbol{head}_i=\text{Attention}(\boldsymbol{Q}\boldsymbol{W}_i^Q,\boldsymbol{K}\boldsymbol{W}_i^K,\boldsymbol{V}\boldsymbol{W}_i^V), \tag{2}$$

其中，$\boldsymbol{Q}$、$\boldsymbol{K}$、$\boldsymbol{V}$的向量维度都是D_m，转换后每一份的维度分别为D_k、D_k、D_v，合并后维度又恢复成D_m。

在Transformer中多头注意力有3种不同的形式：

1）自注意力。多头注意力公式中取$\boldsymbol{Q}=\boldsymbol{K}=\boldsymbol{V}=\boldsymbol{X}$，$\boldsymbol{X}$是Transformer的输入特征，即计算$\boldsymbol{X}$与自己的注意力。

2）掩码自注意力。在自注意力公式中，将注意力矩阵$\boldsymbol{A}$中的某些值改为$-\infty$，避免一些tokens间的关注。

3）交叉注意力。在编码器和解码器之间计算注意力，$\boldsymbol{K}$、$\boldsymbol{V}$来自编码器，$\boldsymbol{Q}$来自解码器。

2.1.3 前馈网

前馈网（FFN）由两个全连接层组成。经过第1个全连接层，特征维度由D_m扩大到D_f；经过第2个全连接层，特征维度由D_f恢复到D_m，具体见公式（3）：

$$\text{FFN}(\boldsymbol{H}')=\text{ReLU}(\boldsymbol{H}'\boldsymbol{W}^1+\boldsymbol{b}^1)\boldsymbol{W}^2+\boldsymbol{b}^2, \tag{3}$$

其中，$\boldsymbol{H}'$是本层输入，$\boldsymbol{W}^1\in\mathrm{R}^{D_m\times D_f}$，$\boldsymbol{W}^2\in\mathrm{R}^{D_f\times D_m}$，$\boldsymbol{b}^1\in\mathrm{R}^{D_f}$，$\boldsymbol{b}^2\in\mathrm{R}^{D_m}$。

2.1.4 残差连接和归一化层

残差连接能够防止梯度消失，归一化层使特征数值维持在均值0、方差1。这样多个Transformer组合成深层网络，可以保持前向、反向数值的稳定，具体见公式（4）—（5）：

$$\boldsymbol{H}'=\text{LayerNorm}(\text{SelfAttention}(\boldsymbol{X})+\boldsymbol{X}), \tag{4}$$

$$\boldsymbol{H}=\text{LayerNorm}(\text{FFN}(\boldsymbol{H}')+\boldsymbol{H}')。 \tag{5}$$

2.1.5 位置编码

Transformer一次性输入序列的所有tokens，不像RNN那样可以根据输入顺序表示token的前后关系。因此，Transformer需要在每个token上累加一个位置编码，来表示token在序列中的位置。

2.1.6 Transformer整体架构

完整的Transformer由编码器和解码器两部分组成，结构如图1。编码器包括多头自注意力、前馈网和其他辅助层。解码器包括掩码多头自注意力、多头交叉注意力、前馈网和其他辅助层。

在设计模型时，我们根据模型的不同功能，可以选择编码器和解码器的不同组合。

1）使用编码器–解码器。使用Transformer的完整结构，输入、输出都是序列。此结构一般用于序列到序列任务，如文本翻译。

2）只使用编码器。输入为序列，输出是序列的表示。此结构一般用于文本分类、序列标记任务，如BERT模型使用的是编码器。

3）只使用解码器。因为只有解码器，没有编码器，要移除解码器中与编码器关联的多头交叉注意力。此结构的输入为序列，输出是一个token（该token再作为输入，继续输出下一个token，直到输出结束）。此结构一般用于序列生成任务。GPT模型使用的是解码器。

2.2 ChatGPT系列模型架构

OpenAI从2018年发布GPT–1[14]到2023年发布GPT–4，模型的能力产生了质的飞跃。模型虽然一直保持Transformer Decoder的总体架构不变，但具体模块有所调整，如GPT–2[15]将LayerNorm移到解码器的输入，而GPT–4引入了混合专家（MoE）层[16]。另外，模型的规模也在数量级地增加，参数量从1.17×10^8增加到1×10^{12}以上。这样的量变引起质变，模型产生了涌现能力。

ChatGPT系列模型的主要创新点和架构如表1所示。

3 大语言模型高效推理

3.1 大语言模型的计算特性

Transformer的结构与CNN有较大差别。CNN的卷积计算数据复用率高。Transformer中的矩阵乘法、矩阵乘向量，数据复用率较低；非线性算子Softmax、LayerNorm计算时要多次遍历数据[17]。这些特点造成Transformer无法充分利用计

▼表1 ChatGPT系列模型的主要创新点和架构

模型名	主要创新点	发布时间/年	上下文序列长度（token）	Transformer层数	多头数量	参数量
GPT-1	• 基于Transformer解码器的单向语言模型 • 无监督预训练+有监督微调模式	2018	512	12	12	1.17×10^{8}
GPT-2	• 多任务预训练，取消微调 • 将LayerNorm移到解码器的输入	2019	1 024	48	48	1.5×10^{9}
GPT-3	• 模型层数增加 • 上下文学习 • 少样本学习、单样本学习、零样本学习	2020	2 048	96	96	1.75×10^{11}
ChatGPT（GPT-3.5）	• RLHF • PPO	2022	4 096	–	–	$>1.75\times10^{11}$
GPT-4	• 引入MoE • 多模态	2023	8 000	–	–	$>1\times10^{12}$

MoE：混合专家　　PPO：近端策略优化　　RLHF：人类反馈强化学习

算硬件的能力，在现有图形处理器（GPU）、加速器上计算效率较低。

3.1.1 模型规模大

大语言模型存在计算量大、存储量大的特点。以GPT-3为例，GPT-3包含96层Transformer，每层有96个注意力头，词向量深度为12 288。整个模型参数量达到1 750亿个，按照INT8数据格式计算，总大小达到175 GB。推理生成一个token需要的计算量达到3 240亿次。

英伟达A100型号GPU的INT8算力为624万亿次运算每秒（TOPS），算力利用率小于10%。A100的显存为80 GB，显然无法装下整个GPT-3模型。因此，对于GPT-3这类大语言模型，推理必须引入分布式技术，将一个模型拆分到多个GPU上，提升推理速度，减小推理延时。

3.1.2 计算强度低

考虑GPU计算深度学习模型的效率，我们需要引入算术强度的概念。算术强度的含义是模型的计算量与内存读写量的比值，具体如公式（6）[17]：

$$\text{Arithmetic Intensity}=\frac{\text{FLOPs}}{\text{MOPs}}, \tag{6}$$

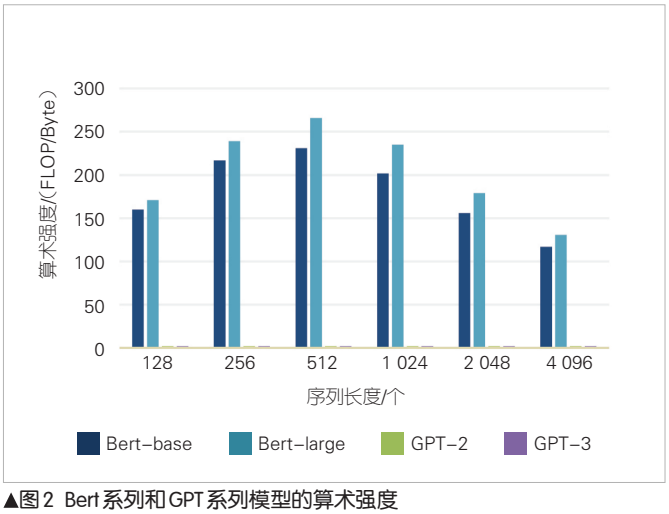
其中，FLOPs表示浮点计算次数，MOPs表示内存访问次数，以一个字节数据为单位统计。算术强度越高，说明模型的计算密集度越高，反之则说明模型的访存密集度越高。对于特定的AI加速器，算术强度有一个最佳值，模型达到这个最佳值，就能够同时最大效率地利用存储带宽与计算资源。如果高于最佳值，模型受限于计算资源，造成存储带宽的浪费；反之，模型受限于存储带宽，则会造成计算资源的浪费。

图2展示了Bert-base、Bert-large、GPT-2、GPT-3在不同序列长度下的推理算术强度。可以看出，由编码器组成的Bert系列模型算术强度较高，达到数百，且随着序列长度变化而变化；由解码器组成的GPT系列模型算术强度很低，只有2，且不随序列长度变化而变化。算术强度为2，意味着读取一个数据只计算2次，计算器件的大量时间浪费在等待数据上，无法充分发挥算力。

表2是Llama2模型[18]（数据类型FP16）在不同Batch（批量）下各层的推理算术强度。可以看出，Llama2的算术强度随着Batch的增大而增大。当Batch为1时，算术强度为1，推理效率很低；当Batch增加到512时，算术强度增加到10.18。再继续增加Batch大小，算术强度基本不变，因为此时激活的大小已超过权重大小，在访存量中占主导地位。

3.1.3 非线性层计算效率低

Transformer中包含LayerNorm、Softmax等非线性运算，



▲图2 Bert系列和GPT系列模型的算术强度

▼表2 Llama2模型各层算术强度

Batch大小	算子	计算量/TFLOPs	访存量/TMOPs	算术强度
1	MHA(线性投影)	42.95	42.46	1.01
	MHA(矩阵乘)	10.74	10.74	1.00
	FFN(线性投影)	57.98	57.99	1.00
	其他	0.02	0.02	1.18
	总计	111.69	111.21	1.00
128	MHA(线性投影)	5 497.56	43.79	125.53
	MHA(矩阵乘)	1 374.39	1 374.56	1.00
	FFN(线性投影)	7 421.70	59.22	125.32
	其他	2.84	2.41	1.18
	总计	14 296.49	1 479.99	9.66
512	MHA(线性投影)	21 990.23	47.82	459.85
	MHA(矩阵乘)	5 497.56	5 498.23	1.00
	FFN(线性投影)	29 686.81	62.95	471.61
	其他	11.37	9.65	1.18
	总计	57 185.98	5 618.65	10.18
4 096	MHA(线性投影)	175 921.86	85.40	2 059.93
	MHA(矩阵乘)	43 980.47	43 985.83	1.00
	FFN(线性投影)	237 494.51	97.71	2 430.59
	其他	91.00	77.19	1.18
	总计	457 487.84	44 246.13	10.34

FFN：前馈网　　TFLOPs：万亿次浮点运算
MHA：多头注意力　　TMOPs：万亿次内存访问

它们也会降低模型的计算效率。Softmax的计算公式为：

$$m = \max(\boldsymbol{X}),\ y_i = \frac{e^{x_i - m}}{\sum_{j=0}^{L-1} e^{x_j - m}}, \tag{7}$$

其中，$\boldsymbol{X}$表示输入向量，L表示输入向量长度。从公式（7）可以看出，Softmax计算会带来这些挑战：计算过程多次完整访问向量$\boldsymbol{X}$，这导致数据长距离共享；每次计算一行向量，这和通常矩阵运算访问数据的方式不一致，这会导致算子融合困难；算术强度低，访存需求大，序列较长时需要反复从片外内存读取数据，这降低了效率[19]。

LayerNorm层包含激活的均值、方差统计计算，也需要多次访问向量$\boldsymbol{X}$，进行非线性计算。这会造成计算效率的降低，和Softmax存在的问题类似。

3.2 大语言模型推理效率提升方法

大语言模型推理效率的评价指标有：首token生成延时、每秒生成token数、每token消耗的芯片毫秒数[20]。提升推理效率的思路主要有3条：

• 增加计算硬件：进行分布式推理，将模型拆分到多个GPU上，提升推理速度，减小推理延时；

• 减小计算量：删除模型中不必要的计算；

• 减小访存量：减少对外部存储器的访问，充分利用缓存；降低数据精度，减小数据大小。

3.2.1 分布式推理

当大语言模型计算量、参数量超过单个GPU能力时，就必须做分布式训练[21]和分布式推理。分布式推理使用的两种典型并行方式为：Tensor并行和Pipeline并行，一般节点内采用Tensor并行[22-24]，节点间采用Pipeline并行[23]。

Tensor并行将每个Transformer层的计算量、存储量平均分布到多个GPU上，能有效降低推理延时，但会增加GPU间数据交互的负担。因此选择Tensor并行度时，需要对比并行计算收益和通信负担，综合评估效率指标。

Pipeline并行将模型的各个Transformer层分配到不同的节点，每个节点负责不同的层。在推理时，各节点上的层进行串行计算。因此，Pipeline并行不能减少推理总延时，只能减小节点存储量。在满足延时指标的前提下，Pipeline并行度不宜过大，只要节点显存足够支持本节点的Transformer层即可。

3.2.2 计算优化

在Transformer的自注意力中，矩阵乘的计算复杂度为序列长度的平方。当序列较长时，计算量会大幅增加。自注意力的非线性算子虽然计算量不大，但计算效率低。这两点都会使延时增大。为了缩短延时，我们可以使用多种方案减少计算步骤，降低计算量。

3.2.2.1 KV Cache

大语言模型由解码器组成，是生成式模型。在推理时，输入一段提示，能够生成一段回答。生成的回答并不是一次生成所有tokens，而是每次推理生成一个token，这个token再和输入的所有tokens拼接在一起，作为下一次推理的输入，生成下一个token。这样反复进行，直到模型输出结束符（EOP）为止。

生成式模型推理的弊端是很明显的。每次输入的所有tokens，都要参与注意力计算。除了最新token，前面的其他tokens的计算与前次推理相同，是重复计算。为了解决这个问题，可使用KV Cache。它可以将每次推理计算出的tokens的特征缓存在显存中，下次推理直接取用，无须重复计算。这样每次推理，输入只有一个最新token，自注意力的计算量可以大幅下降。

大语言模型推理过程属于带宽受限型，KV Cache虽然

减少了大量计算，但也带来了存储和带宽的压力。例如，GPT-3模型参数占用显存大小为350 GB，假设Batch大小为64，输入序列长为512，输出序列长度为32，则KV cache占用显存为164 GB，大约是模型参数占用显存的1/2。KV Cache的规模较大，且对存储带宽要求较高，并会遭遇内存墙问题。因此，又出现了多查询注意力（MQA）[25]和分组查询注意力（GQA）[26]。

3.2.2.2 共享关注头

在原始Transformer的MHA中，QKV分别包含相同数量的头，且一一对应。每个头的QKV内部进行计算，再将结果拼接在一起。

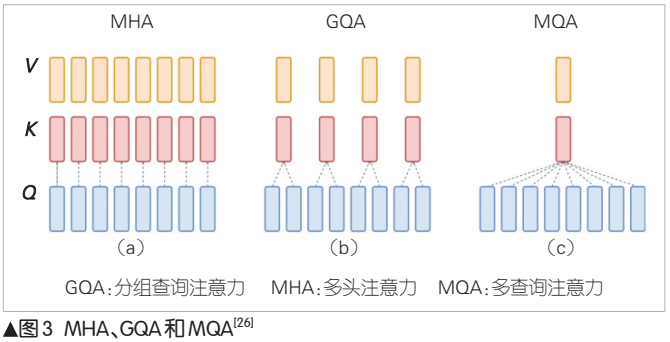
MQA的$\boldsymbol{Q}$仍然保持原来的头数，但$\boldsymbol{K}$和$\boldsymbol{V}$只有一个头，共享给所有$\boldsymbol{Q}$头使用，如图3（c）所示。MQA免除了多个$\boldsymbol{KV}$头的计算和存储，大大减少了存储和访存带宽压力，推理吞吐量可提高30%～40%，而模型性能只有少量损失。

GQA是MHA和MQA的折衷，该方法减少了模型性能损失，获得MQA带来的推理加速好处。具体方法是，不是所有$\boldsymbol{Q}$头共享一组$\boldsymbol{KV}$，而是$\boldsymbol{Q}$头分成多组，每组共享一组$\boldsymbol{KV}$，例如图3（b）是每两个$\boldsymbol{Q}$头共享一组$\boldsymbol{KV}$。

3.2.2.3 推测解码

推测解码[27-28]是2023年新兴的大语言模型推理加速技术，通过增加推理的并行度来提高计算效率，降低延时。其具体方法是为大语言模型配备一个小语言模型，推理时，先由小语言模型“推测”生成几个tokens，然后将这几个tokens放入大语言模型中进行推理验证。如果验证正确，则小语言模型继续“推测”后续token；如果验证错误，大语言模型修正已有token，小语言模型接受修正，继续“推测”后续token。

推测解码之所以能降低延时，一是因为小模型计算速度远超大模型，有数量级的提升；二是因为大模型并行推理几个tokens，只需读取一次参数，计算强度提高，平均每个token的计算延时大幅降低。



▲图3 MHA、GQA和MQA[26]

3.2.2.4 精简Transformer

文献[29]介绍了Transformer的简化，并以信号传播理论及实证研究结果为基础，证明了Transformer中许多组件，如残差连接、Value、投影和LayerNorm，可以在不牺牲训练速度的情况下被删除。在纯自回归解码器和纯BERT编码器模型上的实验表明，简化后Transformer实现了与标准Transformer相当的训练速度和性能，同时训练吞吐量提高了15%，使用的参数减少了15%。

3.2.3 访存优化

3.2.3.1 FlashAttention

FlashAttention[30]通过分块计算Softmax和核函数融合，来降低对显存的访问。

计算Softmax需要遍历两遍全体数据。FlashAttention修改了Softmax算法，将数据分成多个小块，无须遍历即可逐块计算出Softmax的中间结果，当所有块计算完成后，再对中间结果进行一次校正，就可得最终结果。

FlashAttention还将Softmax和前后的矩阵乘等算子融合成一个统一计算设备架构（CUDA）核函数。每块数据在GPU上完成核函数计算后，结果被输出到显存。这个过程充分利用了数据局部性，中间激活保存在GPU缓存，避免了反复读写显存，这可以使计算速度提升7.6倍。

FlashAttention-2[31]是对FlashAttention的改进，它消除了原先频繁的系数更新，减少了对加速设备不擅长的非矩阵乘法运算的需求，提出了在序列长度上的并行化，获得并行加速优势。结合GPU运行特点，在一个注意力计算块内，我们将工作分配在一个CUDA线程block的不同warp上，以减少通信和共享内存读写，提高模型效率。

3.2.3.2 Flash-Decoding

Flash-Decoding[32-33]借鉴了FlashAttention的思路，将并行化维度扩展到KV序列长度。KV序列被分成多个小块，每块内部完成QKV注意力计算，多块之间可并行计算，无须等待Softmax统计全局最大值。

Flash-Decoding不是在运行时实时统计数据最大值，而是在模型设计时离线统计数据的分布，在分布区间内取一个较大的值Φ作为最大值。运行时直接用Φ计算Softmax，可以应付90%以上的情况。如果实际的最大值超过了Φ，就会

暂停Flash-Decoding，回退到原始算法。

3.2.3.3 PagedAttention

KV Cache显著减小了模型的计算量，但也存在一些缺点：一是显存占用大，达到数GB以上；二是大小动态变化，随着序列长度的不同，大小可相差数千倍，且不可预测。这给有效管理KV Cache带来了很大的挑战。研究发现，由于碎片化和过度保留，现有系统浪费了60%～80%的显存。受操作系统中虚拟内存和分页经典思想启发，PagedAttention[34]允许在非连续的内存空间中存储连续的KV Cache，有效提高内存的利用率。同时PagedAttention带来另一个关键优势：高效的内存共享。例如，在并行采样中，多个输出序列是由同一个提示生成的，提示的KV Cache页面可以在输出序列中共享。

3.2.4 量化

量化是深度学习模型通用的压缩方法，将模型的权重/激活数据格式转换为INT8、INT4等整数，以降低计算量和存储量。大语言模型量化按阶段分有3种：训练感知量化（QAT）、训练后量化（PTQ）、微调感知量化（QAFT）[35]。由于大语言模型训练困难，因此QAT用得较少，PTQ、QAFT用得比较多，本文中我们主要讨论PTQ。

3.2.4.1 仅量化权重

大语言模型训练完成后，权重是已知的，而激活则要等到推理时才能知道，且取值范围与输入的序列相关。因此，权重比较容易量化，误差较小；而激活的量化较难，如果遇到离群值，误差会比较大。基于这样的特点，我们可以对模型做混合精度量化，方法有LLM.int8()[36]、GPT量化（GPTQ）[37]、不相干处理量化（QuIP）[38]、激活感知的权重量化（AWQ）[39]、离群值感知的权重量化（OWQ）[40]、稀疏量化表示（SpQR）[41]、细粒度权重量化（FineQuant）[42]。

LLM.int8()认为，激活中的离群值很重要，因此在计算矩阵乘法时，需要对离群值和正常值做不同的处理：

- 取出激活中异常值所在的列以及权重中对应的行，保持FP16格式，计算点乘；
- 剩下的激活正常值，与对应的权重量化到INT8，计算点乘，再反量化成FP16；
- 两者的结果累加，得最终结果。

3.2.4.2 权重和激活都量化

ZeroQuant[43]对权重做按组量化，对激活做按token量化，并逐层使用知识蒸馏缓解精度损失。此方法在BERT和GPT-3模型上精度较高。

针对激活的离群值，SmoothQuant[44]的量化方案是：激活有离群值，权重无离群值，如果把两者平均一下，让它们的数值都落入正常范围，就容易量化了。具体做法是，按通道统计激活的取值范围，如果发现有离群值，则把该通道的数据都除以系数a，权重中对应通道数据都乘以a，这样最终的计算结果不变。

类似的方法还有OliVe[45]、基于重排列的训练后量化（RPTQ）[46]、量化大语言模型（QLLM）[47]、Outlier Suppression+[48]。

4 大语言模型的性能提升

将大语言模型加入新的结构，也可以提升模型性能。典型的结构有MoE和状态空间模型（SSM）[49]。

MoE采用稀疏化策略，即在模型中放置大量专家，每个专家代表一个领域，推理时每次只启用少量专家。这样的话，模型可以在整体上关注更多的领域，而推理时，单个token只关注特定领域，避免了在无关领域上浪费算力。这样既减小了计算量，又提高了模型性能。

SSM不受上下文窗口长度的限制，能够处理超长序列，选择性记录上下文信息，可以在音频、文本等方面展现出良好的性能。SSM在结构上综合了CNN与RNN的特点，计算量、存储量比Transformer更小。

4.1 MoE

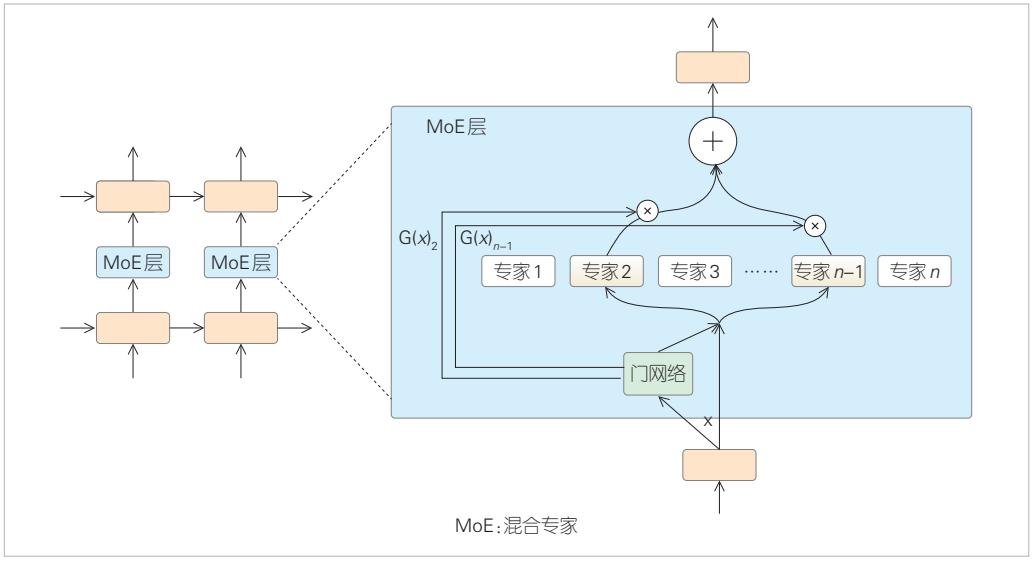
文献[16]首先提出了MoE的概念，认为用单个模型去适应多个场景的样本，会受到很多干扰，因此会导致学习很慢、泛化困难。使用多个模型，且每个模型学习一个场景，就可以得到比较好的效果。在多个模型之前增加一个门网络，就可以决定每个数据应该被哪个模型处理。

文献[50]提出将MoE引入语言模型，并做了两点创新：

- Sparsely-Gated：门网络每次只选择两个专家进行计算，显著降低了计算量；
- token-level：对一个序列中的每个token，各自独立地分别选择专家；

文献[51]的MoE架构如图4，推理的计算步骤如下：

- 输入$\boldsymbol{x}$进入门网络，经过一个全连接层，计算出每个专家的概率；
- 选出概率最高的两个专家，将$\boldsymbol{x}$输入到这两个专家中；
- 两个专家计算的结果，与前面的概率相乘累加，得最终输出。



▲图4 MoE架构[51]

GShard[51]和Switch Transformers[52]在大语言模型中加入MoE，取得了较好效果，下面我们将详细介绍。

4.1.1 GShard

文献[51]认为，增加大语言模型的深度和宽度，计算复杂度则会超线性增加，一旦超过了$O(n)$，大规模训练难以实现。在Transformer中加入MoE，计算复杂度会明显下降到小于$O(n)$，大规模训练则可以实现。因此，文献[51]提出了GShard，并在分布式训练、推理中使用了专家并行，将多个专家分布在不同的计算设备上，降低了每个设备的存储量。

Transformer中加入MoE的方法，将FFN替换为MoE。MoE中有多个专家，每个专家都是一个单独的FFN。具体如公式（8）—（10）：

$$\boldsymbol{G}_{s,E} = \mathrm{GATE}(\boldsymbol{x}_s), \tag{8}$$

$$\mathrm{FFN}_e(\boldsymbol{x}_s) = \boldsymbol{wo}_e \cdot \mathrm{ReLU}(\boldsymbol{wi}_e \cdot \boldsymbol{x}_s), \tag{9}$$

$$\boldsymbol{y}_s = \sum_{e=1}^{E} G_{s,e} \cdot \mathrm{FFN}_e(\boldsymbol{x}_s), \tag{10}$$

其中，$\boldsymbol{x}_s$是MoE的输入，向量$\boldsymbol{G}_{s,E}$是门控网络GATE计算的每个专家的选择概率，其元素为$G_{s,e}, e \in [1, E]$，选择策略是取概率最大的两个专家进行实际计算，其他专家的概率设为0。

MoE中有E个专家FFN_1，…，FFN_E，第e个专家内部的计算为输入全连接层（权重为$\boldsymbol{wi}_e$）、线性整流函数（ReLU）、输出全连接层（权重为$\boldsymbol{wo}_e$）。

$\boldsymbol{y}_s$是MoE的输出，由选中的两个专家的计算结果与概率相乘累加而成。

GShard专家并行如图5所示。图5中分别为标准Transformer编码器、MoE Transformer编码器和多设备分布式的MoE Transformer编码器。MoE Transformer编码器与Transformer编码器的差别是FFN替换成了MoE，MoE放置在单计算设备上，设备需要存储所有专家的权重。多设备分布式的MoE Transformer编码器进一步实现了专家并行，E个专家分别放置在E个计算设备上，每个设备放置一个专家，因此只需存储一个专家的权重。在推理时，每个token通过门网络选择两个专家。如果专家就在自己所在的设备上，则直接进行计算；否则，将token通过All–to–All集合通信接口发送到专家所在的设备，进行计算，结果再通过All–to–All集合通信接口发回到原设备。

4.1.2 Switch Transformers

Switch Transformers也是将FFN替换为MoE，不同的是修改了门网络，每次只选择一个专家计算，称为Switch层。这样有3个好处：

- 减少了路由的计算量；
- 每个专家的批量大小（专家容量）至少可以减半；
- 路由执行简化，通信成本降低。

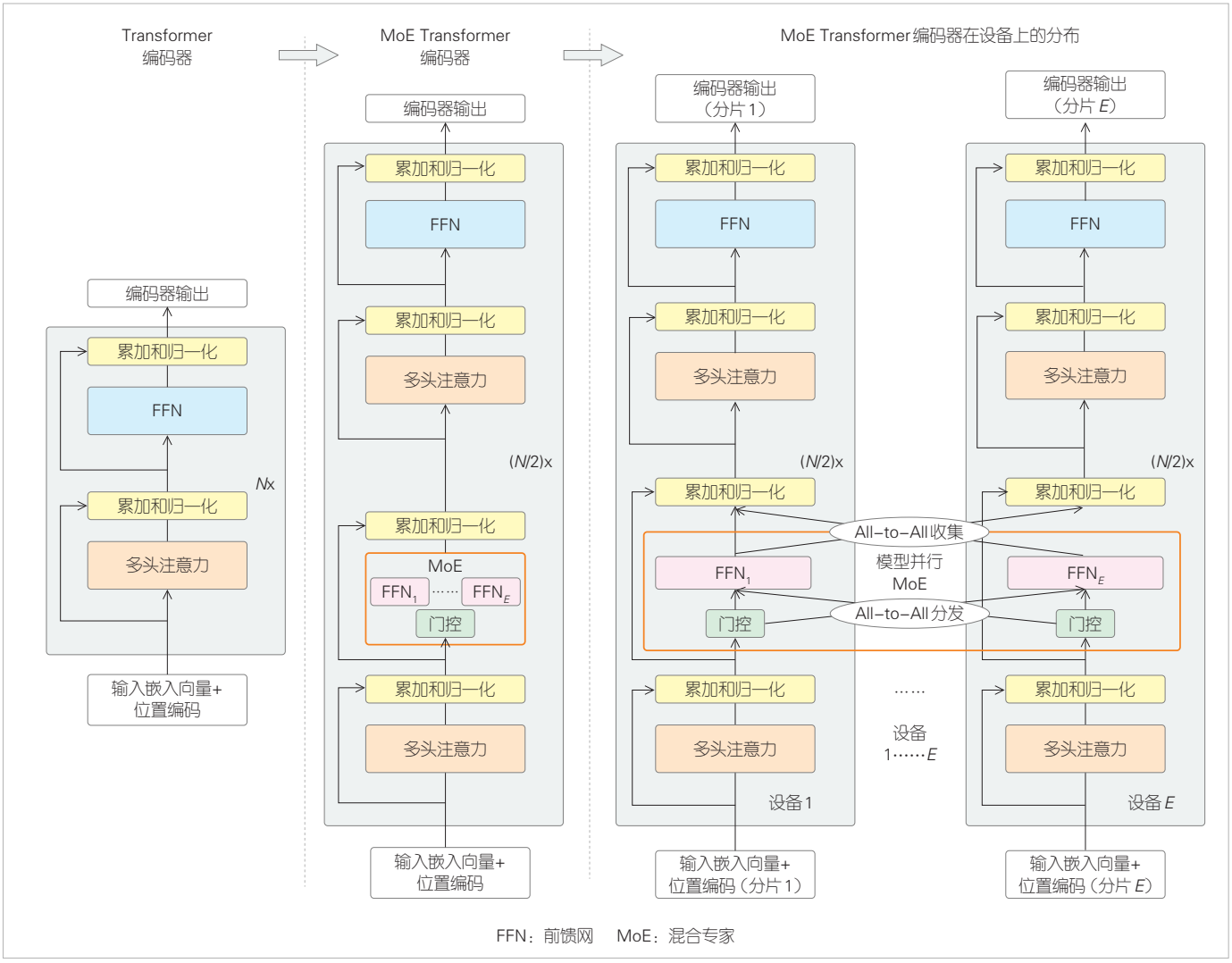
虽然Switch层选择的专家数减小了，但模型的性能却比普通MoE模型更高。这说明在大语言模型场景，设备内存相对稀缺。如果专家接收过多的tokens，会因内存不足而丢弃token，这样会造成模型性能下降。

Switch Transformers使用数据并行、模型并行、专家并行，在较大的集群上进行分布式训练，得到了参数量为3.95×10^{11}、1.571×10^{12}的大语言模型。

4.2 状态空间模型

Transformer的优点是能够并行计算一个序列所有tokens间的相关信息；缺点是序列的长度有限，对于超过上下文窗口长度的序列（如音频、视频）难以计算。SSM可以解决此问题。

SSM是控制论里的概念，其作用是对一个输入连续时间



▲图5 单设备混合专家与多设备GShard 专家并行[51]

信号进行处理，得到一个输出连续时间信号，信号的长度没有限制。调整SSM的参数，可以使输出信号成为输入信号的特征。音频、视频数据本身就是连续时间信号，适合用SSM处理。但文本序列是离散的，且信息密度大，不能直接用SSM处理，因此需要对SSM做离散化及一些调整。

语言模型加入调整后的SSM，有些取得了较好的效果，目前常见的SSM语言模型有线性注意力[53]、S4[54]、H3[55]、Hyena[56]、RetNet[57]、接受权重键值（RWKV）[58]、Mamba[59]等。本文中，我们主要介绍S4和Mamba。

4.2.1 S4模型

S4模型的连续SSM公式为：

$$\boldsymbol{h}'(t) = \boldsymbol{A}\boldsymbol{h}(t) + \boldsymbol{B}\boldsymbol{x}(t), \tag{11}$$

$$\boldsymbol{y}(t) = \boldsymbol{C}\boldsymbol{h}(t), \tag{12}$$

其中，$\boldsymbol{x}(t)$是输入信号，$\boldsymbol{h}(t)$是隐藏状态，$\boldsymbol{h}'(t)$是$\boldsymbol{h}(t)$的变化率，$\boldsymbol{y}(t)$是输出信号，$\boldsymbol{A}$、$\boldsymbol{B}$、$\boldsymbol{C}$是参数矩阵。

SSM离散化后才能用于处理文本序列，具体做法是根据步长Δ将$\boldsymbol{A}$、$\boldsymbol{B}$离散化为$\bar{\boldsymbol{A}}$、$\bar{\boldsymbol{B}}$，得离散化SSM公式为：

$$\boldsymbol{h}_t = \bar{\boldsymbol{A}}\boldsymbol{h}_{t-1} + \bar{\boldsymbol{B}}\boldsymbol{x}_t, \tag{13}$$

$$\boldsymbol{y}_t = \boldsymbol{C}\boldsymbol{h}_t, \tag{14}$$

其中，$\boldsymbol{x}_t$是输入序列中的第t个token，$\boldsymbol{h}_t$是第t个隐藏状态，$\boldsymbol{y}_t$是第t个输出。

以上公式是递归的，序列长度不受限制。如果已知序列长度，可以进行展开，得：

$$\boldsymbol{y}_k = \boldsymbol{C}\bar{\boldsymbol{A}}^k\bar{\boldsymbol{B}}\boldsymbol{x}_0 + \boldsymbol{C}\bar{\boldsymbol{A}}^{k-1}\bar{\boldsymbol{B}}\boldsymbol{x}_1 + \cdots + \boldsymbol{C}\overline{\boldsymbol{AB}}\,\boldsymbol{x}_{k-1} + \boldsymbol{C}\bar{\boldsymbol{B}}\boldsymbol{x}_k。 \tag{15}$$

此式可转换为卷积形式：

$$\bar{K} = (C\bar{B}, C\overline{AB}, ..., C\bar{A}^{k}\bar{B}),$$

$$y = x * \bar{K}。 \quad (16)$$

因此，S4模型在训练时可以使用卷积形式计算SSM，在推理时可以使用递归形式计算SSM。

4.2.2 Mamba模型

Mamba在S4的基础上做了3点改进：

1）对输入信息有选择的处理

序列建模的一个基本问题是把上下文压缩成更小的状态。从这个角度来看，注意力机制虽然效果好，但效率不是很高。因为它不压缩上下文，而是全部存储（也就是KV Cache），这直接导致训练和推理消耗算力大。RNN压缩全部上下文，但压缩率随着序列长度增加而增大，最终会丢失长期信息，因此推理和训练效率高，但性能较差。Mamba的解决办法是，让模型对上下文有选择的处理，丢弃不重要信息，对重要信息进行压缩保留，在计算效率和保留信息两方面取得平衡[59]。

S4的参数矩阵B、C和步长Δ，是所有tokens共享的，每个token计算使用相同的一套B、C、Δ。Mamba的改进点是，用输入序列x（长度为L）经过3个全连接层，生成L套不同的B、C、Δ，每个token计算使用一套，从而实现对信息的选择性处理，关注或忽略特定的内容。

2）硬件感知算法

Mamba在GPU上对SSM算法做了优化，利用扫描而不是卷积来递归计算模型，尽量减少不同级别的GPU内存层次结构之间的IO访问。

3）更简单的架构

Mamba架构是SSM与Transformer的多层感知机（MLP）块的结合，两者相互融合，而不是重叠，形成了一个更加简单的结构，如图6所示。

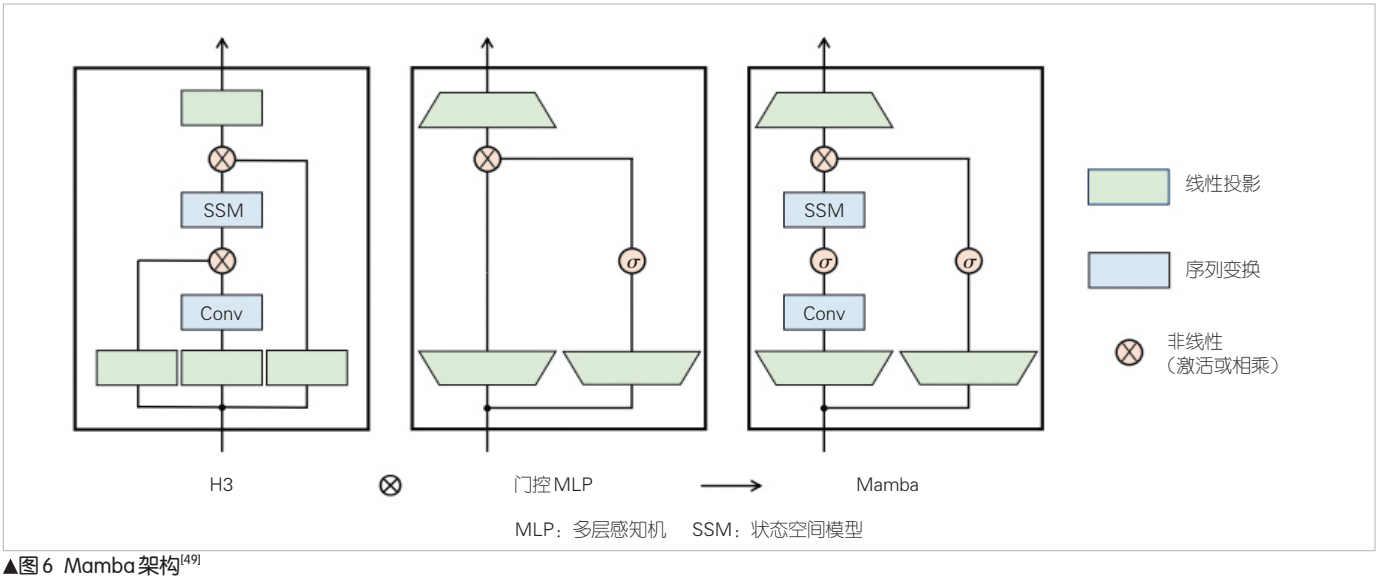
Mamba在合成、音频、基因、语言等多个领域的任务上都表现出良好的性能，并能处理超过1 M长度的序列。

5 结束语

大语言模型是神经网络模型长期发展的成果，特别是Transformer结构的注意力机制与大算力结合，产生了涌现能力，其文本理解、生成、对话能力已接近人类。但Transformer的一些固有特点也导致了其性能的不足，如生成式模式、自注意力造成消耗算力过大，上下文长度有限，硬件亲和性差造成推理性能低等。目前，大语言模型算法的演进是渐进的优化，对Transformer结构做局部的修改，以改善这些不足。未来的演进可能仍然是渐进的，也可能会产生革命性的创新，即提出全新的模式，代替生成式模式和自注意力，增强逻辑推理能力和世界模型认知，更加接近AGI。

参考文献

[1] OpenAI. Introducing ChatGPT [EB/OL]. [2024-03-10]. https://openai.com/blog/chatgpt/
[2] OpenAI. GPT-4 technical report [EB/OL]. [2024-03-10]. DOI: 10.48550/ARXIV.2303.08774
[3] VASWANI A , SHAZEER N, PARMAR N, et al. Attention is all you need [EB/OL]. (2017-06-12) [2023-08-02]. https://arxiv.org/abs/1706.03762
[4] 车万翔, 刘挺. 自然语言处理新范式：基于预训练模型的方法 [J]. 中兴通讯技术, 2022, 27(2): 3-9. DOI: 10.12142/ZTETJ.202202002



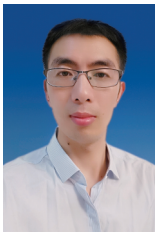
▲图6 Mamba架构[49]

[5] 王海宁. 自然语言处理技术发展 [J]. 中兴通讯技术, 2022, 27(2): 59–64. DOI: 10.12142/ZTETJ.202202009
[6] BENGIL Y, REJEAN D, PASCAL V. A neural probabilistic language model [EB/OL]. (2003–03–01) [2024–03–10]. https://dl. acm. org/doi/10.5555/944919.944966
[7] MIKOLOV T, KARAFIAT M, BURGET L, et al. Khudanpur. Recurrent neural network based language model [EB/OL]. [2024–03–10]. https://www.fit.vut.cz/research/publication/9362/.en
[8] HOCHREITER S, SCHMIDHUBER J. Long short–term memory [J]. Neural computation, 1997, 9(8): 1735–1780. DOI: 10.1162/neco.1997.9.8.1735
[9] PETERS M, NEUMANN M, IYYER M, et al. Deep contextualized word representations [C]//Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics. Association for Computational Linguistics, 2018: 2227–2237. DOI: 10.18653/v1/n18–1202
[10] DEVLIN J, CHANG M, LEE K, et al. BERT: pre–training of deep bidirectional transformers for language understanding [EB/OL]. (2018–11–11)[2024–03–10]. https://arxiv.org/abs/1810.04805
[11] KAPLAN J, MCCANDLISH S, HENIGHAN T, et al. Scaling laws for neural language models [EB/OL]. [2024–03–10]. https://arxiv.org/abs/2001.08361
[12] HOFFMANN J, BORGEAUD S, MENSCH A, et al. Training compute–optimal large language models [EB/OL]. [2024–03–10]. https://arxiv.org/abs/2001.08361
[13] BROWN T B, MANN B, RYDER N, et al. Language models are few–shot learners [C]//Proceedings of the 34th International Conference on Neural Information Processing Systems. ACM, 2020: 1877–1901. DOI: 10.5555/3495724.3495883
[14] RADFOR A, NARASIMHAN K, SALIMANS T, et al. Improving language understanding by generative pre–training [EB/OL]. [2024–03–10]. https://cdn. openai. com/research–covers/language–unsupervised/language_understanding_paper.pdf
[15] RADFORD A, WU J, CHILD R, et al. Language models are unsupervised multitask learners [EB/OL]. (2020–05–28) [2024–03–10]. https://arxiv.org/abs/2005.14165
[16] JACOBS R A, JORDAN M I, NOWLAN S J, et al. Adaptive mixtures of local experts [J]. Neural computation, 1991, 3(1): 79–87. DOI: 10.1162/neco.1991.3.1.79
[17] KIM S, HOOPER C, WATTANAWONG T, et al. Full stack optimization of transformer inference: a survey [EB/OL]. (2023–02–27)[2024–03–10]. https://arxiv.org/abs/2302.14017
[18] TOUVRON H, MARTIN L, STONE K, et al. Llama 2: open foundation and fine–tuned chat models [EB/OL]. (2023–05–18) [2024–03–10]. https://arxiv.org/abs/2307.09288
[19] CHOI J, LI H, KIM B, et al. Accelerating transformer networks through recomposing softmax layers [EB/OL]. [2024–03–10]. https://ieeexplore.ieee.org/document/9975410
[20] POPE R, DOUGLAS H, CHOWDHERY A, et al. Efficiently scaling Transformer inference [EB/OL]. (2022–11–09) [2024–03–10]. https://arxiv.org/abs/2211.05102
[21] 马子轩,翟季冬,韩文弢,等,郑纬民. 高效训练百万亿参数预训练模型的系统挑战和对策 [J]. 中兴通讯技术, 2022, 27(2): 51–58. DOI: 10.12142/ZTETJ.202202008
[22] SHOEYBI M, PATWARY M, PURI R, et al. Megatron–LM training multi–billion parameter language models using model parallelism [EB/OL]. (2019–09–17) [2024–03–13]. https://arxiv.org/abs/1909.08053
[23] NARAYANAN D, SHOEYBI M, CASPER J, et al. Efficient large–scale language model training on GPU Clusters using megatron–LM [EB/OL]. (2021–04–09) [2024–03–10]. https://arxiv. org/abs/2104.04473
[24] KORTHIKANTI V, CASPER J, LYM S, et al. Reducing activation recomputation in large transformer models [EB/OL]. (2022–05–10)[2024–03–10]. https://arxiv.org/abs/2205.05198
[25] SHAZEER N. Fast transformer decoding: one write–head is all you need [EB/OL]. (2019–11–06)[2024–03–10]. https://arxiv.org/abs/1911.02150
[26] AINSLIE J, LEE–THORP J, DE JONG M, et al. GQA: training generalized multi–query transformer models from multi–head checkpoints [C]//Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing. Association for Computational Linguistics, 2023: 4895–4901. DOI: 10.18653/v1/2023.emnlp–main.298
[27] STERN M, SHAZEER N M, USZKOREIT J, et al. Blockwise parallel decoding for deep autoregressive models [EB/OL]. [2023–03–10]. https://arxiv.org/pdf/1811.03115.pdf
[28] LEVIATHAN Y, KALMAN M, MATIAS Y. Fast inference from transformers via speculative decoding [C]//Proceedings of the 40th International Conference on Machine Learning. ACM, 2023: 19274–19286. DOI: 10.5555/3618408.3619203
[29] HE B, HOFMANN T. Simplifying transformer blocks [EB/OL]. (2023–11–03)[2023–03–10]. https://arxiv.org/abs/2311.01906
[30] DAO T, FU Y D, FU Y , et al. Flashattention: fast and memory–efficientexact attention with io–awareness. [EB/OL]. [2023–03–10].https://arxiv. abs/2205.14135
[31] DAO T. Flashattention–2: Faster attention with better parallelism and work partitioning [EB/OL]. (2023–06–17) [2024–03–10]. https://arxiv.org/abs/2307.08691
[32] DAO T, HAZIZA D, MASSA F, et al. Flash–decoding for long–context inference [EB/OL]. (2023–06–17) [2024–03–10]. https://pytorch.org/blog/flash–decoding/
[33] HONG K, DAI G, XU J, et al. FlashDecoding++ : faster large language model inference on GPUs [EB/OL]. (2023–11–02) [2024–03–10]. https://arxiv.org/abs/2311.01282
[34] KWON W, LI Z H, ZHUANG S Y, et al. Efficient memory management for large language model serving with PagedAttention [EB/OL]. (2023–09–12) [2024–03–10]. https://arxiv.org/abs/2309.06180
[35] WAN Z, WANG X, LIU C, et al. Efficient large language models: a survey [EB/OL]. (2023–12–06) [2024–03–10]. https://arxiv.org/abs/2312.03863
[36] DETTERS T, LEWIS M, BELKADA Y. LLM. int8(): 8–bit matrix multiplication for transformers at scale [EB/OL]. (2022–08–15) [2024–03–10]. https://arxiv.org/abs/2208.07339
[37] FRANTAR E, ASHKBOOS S, HOEFLER T, et al. GPTQ: accurate post–training quantization for generative pre–trained transformers [EB/OL]. (2022–10–31) [2024–03–10]. https://arxiv.org/abs/2210.17323
[38] CHEE J, CAI Y, KULESHOV V, et al. QuIP: 2–bit quantization of large language models with guarantees [EB/OL]. (2022–06–25) [2024–03–10]. https://arxiv.org/abs/2307.13304
[39] LIN J, TANG J, TANG H, et al. AWQ: activation–aware weight quantization for LLM compression and acceleration [EB/OL]. (2023–10–13)[2024–03–10]. https://arxiv.org/abs/2306.00978
[40] LEE C, JIN J, KIM T, et al. OWQ: Lessons learned from activation outliers for weight quantization in large language models [EB/OL]. (2023–06–04) [2024–03–10]. https://arxiv. org/abs/2306.02272
[41] DETTERS T, SVIRSCHEVSKI R, EGIAZARIAN V, et al. SpQR: a sparse–quantized representation for near–lossless LLM weight compression [EB/OL]. (2023–06–05) [2024–03–10]. https://arxiv.org/abs/2306.03078
[42] KIM J Y, HENRY R, FAHIM R, et al. FineQuant: unlocking efficiency with fine–grained weight–only quantization for LLMs [EB/OL]. (2023–08–16) [2024–03–10]. https://arxiv. org/abs/2308.09723
[43] YAO Z, AMINABADI Y R, ZHANG M, et al. ZeroQuant: efficient

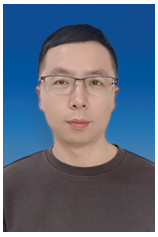
and affordable post-training quantization for large-scale transformers [EB/OL]. (2022-06-04)[2024-03-10]. https://arxiv.org/abs/2206.01861
[44] XIAO G X, LIN J, SEZNEC M, et al. SmoothQuant: accurate and efficient post-training quantization for large language models [EB/OL]. (2022-11-18) [2023-03-10]. https://arxiv. org/abs/2211.10438
[45] GUO C, TANG J, HU W M, et al. OliVe: accelerating large language models via hardware-friendly outlier-victim pair quantization [EB/OL]. (2022-11-18) [2024-03-15]. https://arxiv.org/abs/2304.07493
[46] YUAN Z H, NIU L, LIU J W, et al. RPTQ: reorder-based post-training quantization for large language models [EB/OL]. (2023-04-03)[2024-03-15]. https://arxiv.org/abs/2304.01089
[47] LIU J, GONG RH, WEI X Y, et al. QLLM: accurate and efficient low-bitwidth quantization for large language models [EB/OL]. (2023-12-12)[2024-03-12]. https://arxiv.org/abs/2310.08041
[48] WEI X Y, ZHANG Y C, LI Y H, et al. Outlier suppression+ : accurate quantization of large language models by equivalent and optimal shifting and scaling [EB/OL]. (2023-04-18) [2024-03-12]. https://arxiv.org/abs/2304.09145
[49] GU A, DAO T, ERMON S, et al. Hippo: recurrent memory with optimal polynomial projections [EB/OL]. (2020-08-17) [2024-03-10]. https://arxiv.org/abs/2008.07669
[50] SHAZEER M N, MIRHOSEINO A, ZAZIARZ M, et al. Outrageously large neural networks: the sparsely-gated mixture-of-experts layer [EB/OL]. (2017-01-23) [2024-03-10]. https://arxiv.org/abs/1701.06538
[51] LEPIKHIN D, LEE H J, XU Y Z, et al. GShard: scaling giant models with conditional computation and automatic sharding [EB/OL]. (2020-01-30) [2024-03-10]. https://arxiv. org/abs/2006.16668
[52] FEDUS W, ZOPH B, NOAM M. Switch Transformers: scaling to trillion parameter models with simple and efficient sparsity [EB/OL]. (2021-01-11) [2024-03-10]. https://arxiv. org/abs/2101.03961
[53] KATHAROPOULOS A, VYAS A, PAPPAS N, et al. Transformers are RNNs: fast autoregressive transformers with linear attention [EB/OL]. (2020-06-29) [2024-03-11]. https://arxiv. org/abs/2006.16236
[54] GU A, GOEL K, RE C. Efficiently modeling long sequences with structured state spaces [EB/OL]. (2021-10-31) [2024-03-12]. https://arxiv.org/abs/2111.00396
[55] DAO T, FU Y D, SAAB K K, et al. Hungry hungry hippos: towards language modeling with state space models [EB/OL]. (2022-12-28)[2024-03-12]. https://arxiv.org/abs/2212.14052
[56] POLI M, MASSAROLI S, NGUYEN E, et al. Hyena hierarchy: towards larger convolutional language models [EB/OL]. (2023-02-21)[2024-03-12]. https://arxiv.org/abs/2302.10866
[57] SUN Y T, DONG L, HUANG S H, et al. Retentive network: a successor to transformer for large language models [EB/OL]. (2023-07-17)[2024-03-12]. https://arxiv.org/abs/2307.08621
[58] PENG B, ALCAIDE E, ANTHONY Q, et al. RWKV: reinventing RNNs for the transformer era [EB/OL]. [2024-03-12]. https://aclanthology.org/2023.findings-emnlp.936/
[59] GU A, DAO T. Mamba: linear-time sequence modeling with selective state spaces [EB/OL]. (2023-12-01) [2024-03-12]. https://arxiv.org/abs/2312.00752

作 者 简 介

朱炫鹏，中兴通讯股份有限公司无线资深专家；主要研究方向为深度学习算法、计算机视觉、大语言模型。



姚海东，中兴通讯股份有限公司无线资深专家；主要从事深度学习、大模型网络架构及编译转换技术研究和设计。



刘隽，中兴通讯股份有限公司无线资深专家；主要研究方向包括深度学习算法、AI编译器、AI加速器架构设计和模拟仿真等。

熊先奎，中兴通讯股份有限公司无线首席架构师、“智算”技术委员会前瞻组组长；长期从事计算系统和体系结构、先进计算范式以及异构计算加速器研究工作；曾主导过中兴通讯ATCA先进电信计算平台、服务器存储平台、智能网卡和AI加速器等系统架构设计。