主流开源大模型架构详细对比

一、架构类型分类

1. Decoder-Only 架构（自回归生成模型）

特点：

- 仅使用Transformer Decoder部分
- 自回归生成，使用因果掩码（Causal Mask）
- 适合文本生成、对话等任务

代表模型：

- GPT系列（GPT-2, GPT-3, GPT-3.5, GPT-4）
- LLaMA系列（LLaMA, LLaMA-2, LLaMA-3）
- Qwen系列（Qwen, Qwen-2）
- Baichuan系列（Baichuan-2, Baichuan-3）
- Mistral系列（Mistral-7B, Mixtral-8x7B）

2. Encoder-Only 架构（理解任务模型）

特点：

- 仅使用Transformer Encoder部分
- 双向上下文理解
- 适合文本分类、NER、情感分析等任务

代表模型：

- BERT系列（BERT, RoBERTa, ALBERT）

3. Encoder-Decoder 架构（序列到序列模型）

特点：

- 同时使用Encoder和Decoder
- Encoder理解输入，Decoder生成输出
- 适合翻译、摘要、问答等任务

代表模型：

- T5系列（T5, T5-XXL, Flan-T5）
- BART

4. Prefix-Decoder 架构（GLM架构）

特点：

- 结合Encoder和Decoder的优势
- 前缀部分双向理解，生成部分自回归

代表模型：

- ChatGLM系列（ChatGLM-2, ChatGLM-3, GLM-4）

二、关键技术详细对比

1. 位置编码（Position Encoding）

绝对位置编码（Absolute Position Encoding）

- **使用模型**：GPT-2, GPT-3, BERT
- **实现方式**：可学习的位置嵌入，直接加到token embedding上
- **优点**：简单直接，易于实现
- **缺点**：外推能力有限，难以处理训练时未见过的序列长度

```
# 伪代码示例
position_embedding = nn.Embedding(max_seq_len, hidden_size)
pos_emb = position_embedding(position_ids)
token_emb = token_embedding(input_ids)
final_emb = token_emb + pos_emb
```

RoPE（旋转位置编码，Rotary Position Embedding）

- **使用模型**：LLaMA, Qwen, ChatGLM, Baichuan, Mistral
- **实现方式**：通过旋转矩阵将位置信息编码到注意力计算中
- **优点**：
  - 相对位置编码，外推能力强
  - 可以处理比训练时更长的序列
  - 计算效率高
- **缺点**：实现相对复杂

```
# RoPE核心思想：通过旋转矩阵编码相对位置
# Q, K 在计算注意力前进行旋转
def apply_rotary_pos_emb(q, k, cos, sin):
    q_embed = (q * cos) + (rotate_half(q) * sin)
    k_embed = (k * cos) + (rotate_half(k) * sin)
    return q_embed, k_embed
```

相对位置编码（Relative Position Encoding）

- **使用模型**：T5
- **实现方式**：在注意力计算中加入相对位置偏置
- **优点**：泛化能力强
- **缺点**：需要额外的位置偏置矩阵

2. 归一化（Normalization）

Layer Normalization

- **使用模型**：BERT, GPT, ChatGLM
- **公式**：`LN(x) = γ * (x - μ) / (σ + ϵ) + β`
- **位置**：
  - Post-Norm：BERT使用，在残差连接之后
  - Pre-Norm：GPT使用，在残差连接之前（更稳定）

RMSNorm（Root Mean Square Layer Normalization）

- **使用模型**：LLaMA, Qwen, Baichuan, Mistral
- **公式**：`RMSNorm(x) = (x / RMS(x)) * γ`，其中 `RMS(x) = sqrt(mean(x²))`
- **优点**：
  - 计算更高效（不需要计算均值）
  - 训练更稳定
  - 性能与LayerNorm相当或更好

```
# RMSNorm实现
class RMSNorm(nn.Module):
    def __init__(self, dim):
        super().__init__()
        self.scale = nn.Parameter(torch.ones(dim))

    def forward(self, x):
        norm = x.norm(dim=-1, keepdim=True) * (x.shape[-1] ** -0.5)
        return self.scale * x / (norm + 1e-8)
```

3. 激活函数（Activation Function）

ReLU

- **使用模型**：T5
- **公式**：`ReLU(x) = max(0, x)`
- **特点**：简单高效，但存在死神经元问题

GELU（Gaussian Error Linear Unit）

- **使用模型**：GPT, BERT
- **公式**：`GELU(x) = x * Φ(x)`，其中Φ是标准正态分布的CDF
- **特点**：平滑的ReLU变体，性能更好

SwiGLU（Swish-Gated Linear Unit）

- **使用模型**：LLaMA, Qwen, Baichuan, Mistral
- **公式**：`SwiGLU(x) = Swish(xW + b) ⊙ (xV + c)`，其中 `Swish(x) = x * sigmoid(x)`
- **特点**：
  - 门控机制，性能优于GELU
  - 需要更多参数（两个线性层）

```
# SwiGLU实现
class SwiGLU(nn.Module):
    def __init__(self, dim):
        super().__init__()
        self.gate_proj = nn.Linear(dim, dim, bias=False)
        self.up_proj = nn.Linear(dim, dim, bias=False)
        self.down_proj = nn.Linear(dim, dim, bias=False)

    def forward(self, x):
        gate = F.silu(self.gate_proj(x))  # SiLU = Swish
        up = self.up_proj(x)
        return self.down_proj(gate * up)
```

4. 注意力机制（Attention Mechanism）

MHA（Multi-Head Attention）

- **使用模型**：GPT, BERT, T5, Qwen, Baichuan
- **特点**：
  - Q、K、V的头数相同
  - 标准的多头注意力实现
  - 计算复杂度：$O(n^2d)$

MQA（Multi-Query Attention）

- **使用模型**：ChatGLM
- **特点**：
  - 多个Q头共享1个K头和1个V头
  - 大幅减少KV缓存（减少到1/num_heads）
  - 推理效率高，但可能略微损失性能

```
# MQA核心思想
# 假设有32个Q头，但只有1个K头和1个V头
Q = [Q1, Q2, ..., Q32]  # 32个Q头
K = [K1]  # 1个K头，所有Q头共享
V = [V1]  # 1个V头，所有Q头共享
```

GQA（Grouped Query Attention）

- **使用模型**：LLaMA-2, LLaMA-3, Mistral
- **特点**：
  - Q头数 > KV头数（例如：32个Q头，4个KV头）
  - 平衡了MHA的性能和MQA的效率
  - 推理时KV缓存减少到原来的1/8

```
# GQA示例：32个Q头，4个KV头
# 每8个Q头共享1个KV头
Q = [Q1-Q8, Q9-Q16, Q17-Q24, Q25-Q32]  # 4组，每组8个Q头
K = [K1, K2, K3, K4]  # 4个K头
V = [V1, V2, V3, V4]  # 4个V头
```

SWA（Sliding Window Attention）

- **使用模型**：Mistral
- **特点**：
  - 固定窗口大小（例如4096）
  - 每个token只关注窗口内的其他token
  - 计算复杂度从$O(n^2)$降低到$O(n \times w)$，其中w是窗口大小
  - 通过多层堆叠，仍能获得全局感受野

三、模型架构详细对比表

模型	架构类型	位置编码	归一化	激活函数	注意力机制	参数量	上下文长度
GPT-3.5	Decoder-Only	绝对位置编码	LayerNorm	GELU	MHA	175B	8K-32K
LLaMA-2	Decoder-Only	RoPE	RMSNorm	SwiGLU	GQA	7B/13B/70B	4K
LLaMA-3	Decoder-Only	RoPE	RMSNorm	SwiGLU	GQA	8B/70B/405B	128K
Qwen-2	Decoder-Only	RoPE	RMSNorm	SwiGLU	MHA/GQA	0.5B-72B	32K-128K
ChatGLM-3	Prefix-Decoder	RoPE (2D)	LayerNorm	GELU	MQA	6B	32K
Baichuan-2	Decoder-Only	RoPE	RMSNorm	SwiGLU	MHA	7B/13B	4K-32K
Mistral-7B	Decoder-Only	RoPE	RMSNorm	SwiGLU	SWA+GQA	7B	8K-32K

模型	架构类型	位置编码	归一化	激活函数	注意力机制	参数量	上下文长度
Mixtral-8x7B	Decoder-Only (MoE)	RoPE	RMSNorm	SwiGLU	SWA+GQA	47B (激活13B)	32K
BERT	Encoder-Only	绝对位置编码	LayerNorm	GELU	双向MHA	110M-340M	512
T5	Encoder-Decoder	相对位置编码	LayerNorm	ReLU	MHA	220M-11B	512

四、MoE（Mixture of Experts）架构详解

核心思想

MoE架构将模型分为多个专家网络（Expert），每个token只激活部分专家，从而在保持大模型容量的同时降低计算成本。

关键组件

1. **专家网络（Experts）**：多个独立的FFN网络
2. **路由网络（Router）**：决定每个token激活哪些专家
3. **负载均衡（Load Balancing）**：确保专家利用均匀

代表模型

Mixtral-8x7B

- **专家数量**：8个专家，每个7B参数
- **激活数量**：每次激活2个专家
- **总参数量**：47B（8×7B - 共享部分）
- **激活参数量**：约13B（2×7B）
- **优势**：推理时只计算激活的专家，大幅降低计算量

Switch Transformer

- **专家数量**：多个专家
- **激活数量**：每次激活1个专家
- **优势**：计算量最小，但可能损失性能

MoE架构优势

1. **参数量大但激活参数少**：可以训练超大模型，但推理时只使用部分参数
2. **训练和推理效率高**：只计算激活的专家
3. **可扩展性强**：可以轻松扩展到更多专家

MoE架构挑战

1. **负载均衡**：需要确保所有专家都被充分利用
2. **通信开销**：多GPU训练时需要专家间的通信
3. **训练稳定性**：需要特殊的训练技巧

五、架构演进趋势

1. 位置编码演进

- **早期**：绝对位置编码（GPT-2, BERT）
- **现在**：RoPE成为主流（LLaMA, Qwen等）
- **未来**：可能向更高效的位置编码发展

2. 归一化演进

- **早期**：Layer Normalization（BERT, GPT）
- **现在**：RMSNorm成为主流（LLaMA, Qwen等）
- **原因**：计算更高效，训练更稳定

3. 激活函数演进

- **早期**：ReLU（T5）
- **中期**：GELU（GPT, BERT）
- **现在**：SwiGLU成为主流（LLaMA, Qwen等）
- **原因**：性能更好，虽然参数更多

4. 注意力机制演进

- **早期**：标准MHA
- **现在**：GQA、MQA、SWA等变体
- **原因**：平衡性能和效率，减少KV缓存

5. 架构类型演进

- **早期**：单一架构（Encoder-Only或Decoder-Only）
- **现在**：MoE架构兴起（Mixtral, DeepSeek-V3）
- **未来**：可能向更高效的架构发展

6. 上下文长度演进

- **早期**：512-2K tokens（BERT, GPT-2）
- **现在**：32K-128K tokens（LLaMA-3, Qwen-2）
- **未来**：可能支持更长的上下文

六、选择

根据任务选择架构

1. **文本生成/对话**：Decoder-Only（GPT, LLaMA, Qwen）
2. **文本理解/分类**：Encoder-Only（BERT）
3. **翻译/摘要**：Encoder-Decoder（T5, BART）
4. **代码生成**：Prefix-Decoder with 2D RoPE（ChatGLM）

根据资源选择模型

1. **资源充足**：LLaMA-3 70B, Qwen-2 72B
2. **资源中等**：LLaMA-2 13B, Qwen-2 7B
3. **资源有限**：LLaMA-2 7B, Qwen-2 1.5B
4. **追求效率**：Mixtral-8x7B（MoE架构）

根据语言选择模型

1. **中文任务**：Qwen, ChatGLM, Baichuan
2. **英文任务**：LLaMA, GPT, Mistral
3. **多语言任务**：Qwen, LLaMA

七、总结

主流开源大模型的架构设计在不断演进，主要趋势包括：

1. **位置编码**：从绝对位置编码向RoPE演进
2. **归一化**：从LayerNorm向RMSNorm演进
3. **激活函数**：从ReLU/GELU向SwiGLU演进
4. **注意力机制**：从MHA向GQA/MQA/SWA演进
5. **架构类型**：从单一架构向MoE架构演进
6. **上下文长度**：从512 tokens向128K+ tokens演进

这些演进都是为了在保持或提升模型性能的同时，提高训练和推理效率，降低计算成本。