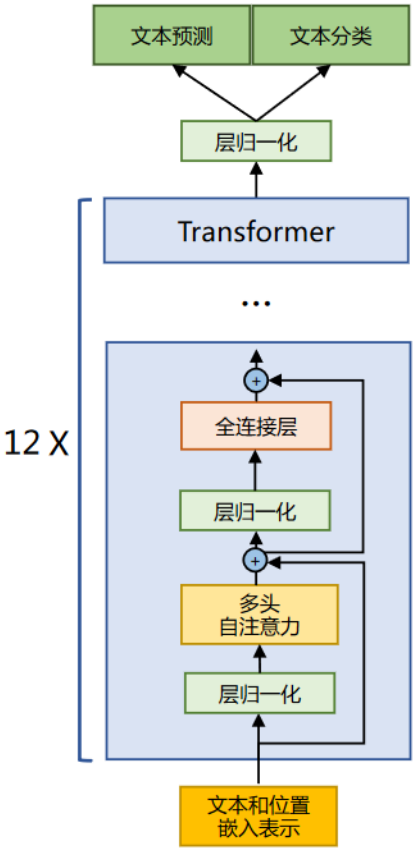
LLaMA 系列模型

1.LLama

1.1 简介

Open and Efficient Foundation Language Models (Open 但没完全 Open 的 LLaMA)

2023 年 2 月，Meta（原 Facebook）推出了 LLaMA 大模型，使用了 1.4T token 进行训练，虽然最大模型只有 65B，但在相关评测任务上的效果可以媲美甚至超过千亿级大模型，被认为是近期开源大模型百花齐放的开端之一，“羊驼”系列模型及其生态快速发展。



LLaMA 所采用的 Transformer 结构和细节，与标准的 Transformer 架构不同的地方包括采用了**前置层归一化（Pre-normalization）**并使用 **RMSNorm 归一化函数**（Normalizing Function）、激活函数更换为 **SwiGLU**，并使用了**旋转位置嵌入（RoP）**，整体 Transformer 架构与 GPT-2 类似。

1.2 RMSNorm 归一化函数

为了使得模型训练过程更加稳定，GPT-2 相较于 GPT 就引入了**前置层归一化方法**，将第一个层归一化移动到多头自注意力层之前，第二个层归一化也移动到了全连接层

之前，同时残差连接的位置也调整到了多头自注意力层与全连接层之后。层归一化中也采用了 **RMSNorm 归一化函数**。 针对输入向量 aRMSNorm 函数计算公式如下

$$RMS(a) = \sqrt{\frac{1}{n}\sum_{i=1}^{n} a_i^2}$$

$$\bar{a}_i = \frac{a_i}{RMS(\boldsymbol{a})}$$

此外，RMSNorm 还可以引入可学习的缩放因子 g_i 和偏移参数 b_i ，从而得到 $\bar{a}_i = \frac{a_i}{\mathrm{RMS}(\boldsymbol{a})} g_i + b_i$ 。

1.3 SwiGLU 计划函数

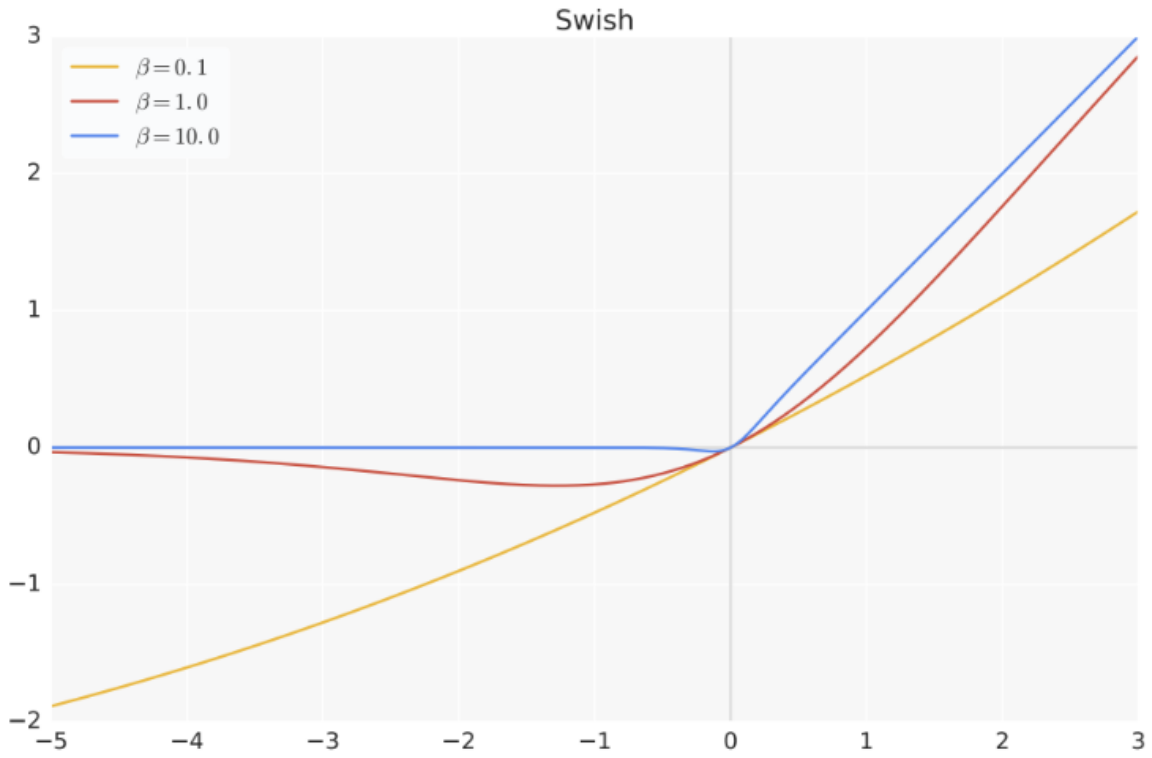
SwiGLU 激活函数是相较于 ReLU 函数在大部分评测中都有不少提升。在 LLaMA 中全连接层 使用带有 SwiGLU 激活函数的 FFN（Position-wise Feed-Forward Network）的计算公式如下：

$$\mathrm{FFN}_{\mathrm{SwiGLU}}\left(\boldsymbol{x}, \boldsymbol{W}, \boldsymbol{V}, \boldsymbol{W}_2\right) = \mathrm{SwiGLU}(\boldsymbol{x}, \boldsymbol{W}, \boldsymbol{V})\boldsymbol{W}_2$$

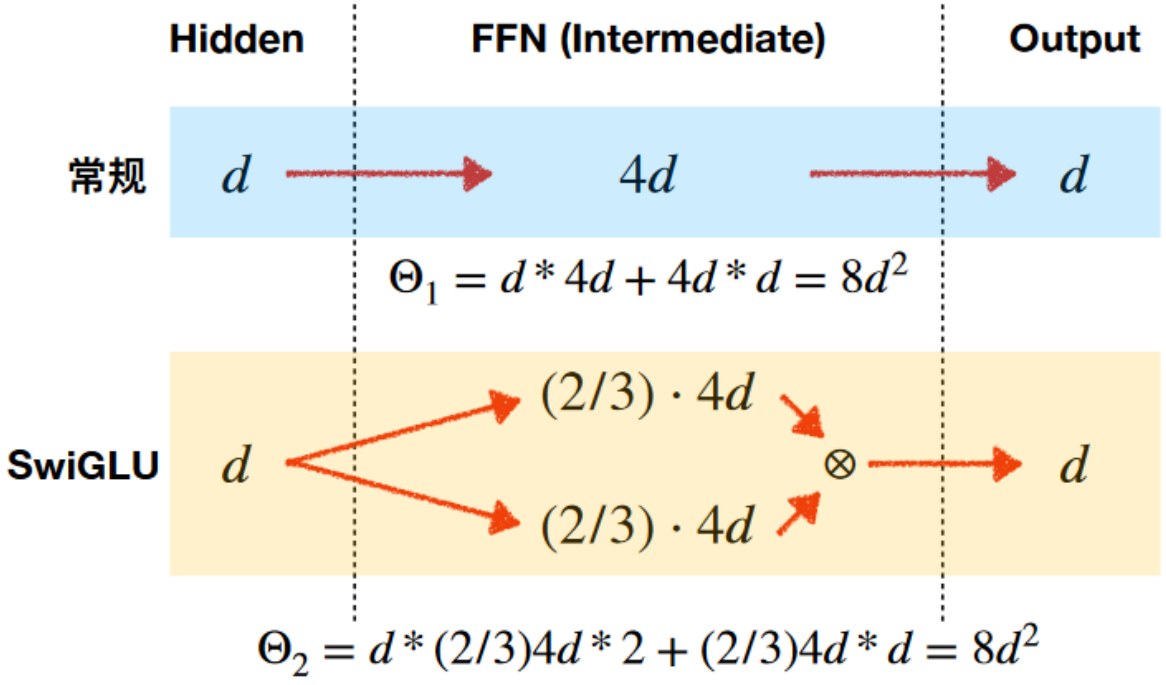
$$\mathrm{SwiGLU}(\boldsymbol{x}, \boldsymbol{W}, \boldsymbol{V}) = \mathrm{Swish}_\beta(x\boldsymbol{W}) \otimes \boldsymbol{x}\boldsymbol{V}$$

$$\mathrm{Swish}_\beta(\boldsymbol{x}) = \boldsymbol{x}\sigma(\boldsymbol{\beta}\boldsymbol{x})$$

其中， $\sigma(x)$ 是 Sigmoid 函数。下图给出了 Swish 激活函数在参数 β 不同取值下的形状。可以看 到当 β 趋近于 0 时，Swish 函数趋近于线性函数 $y = x$ ，当 β 趋近于无穷大时，Swish 函数趋近于 ReLU 函数， β 取值为 1 时，Swish 函数是光滑且非单调。在 HuggingFace 的 Transformer 库中 Swish1 函数使用 silu 函数代替。



LLaMA 中直接将 FFN 中的 ReLU 替换为 SwiGLU，并将维度放缩为 $(2/3) \cdot 4d$



1.4 旋转位置嵌入（RoPE）

在位置编码上，使用旋转位置嵌入（Rotary Positional Embeddings，RoPE）代替原有的绝 对位置编码。RoPE 借助了**复数的思想**，出发点是**通过绝对位置编码的方式实现相对位置编码**。其目标是通过下述运算来给 `q`，`k` 添加绝对位置信息：

$$\tilde{\boldsymbol{q}}_m = f(\boldsymbol{q}, m), \tilde{\boldsymbol{k}}_n = f(\boldsymbol{k}, n)$$

经过上述操作后， $\tilde{\boldsymbol{q}}_m$ 和 $\tilde{\boldsymbol{k}}_n$ 就带有位置 m 和 n 的绝对位置信息。

最终可以得到二维情况下用复数表示的 RoPE：

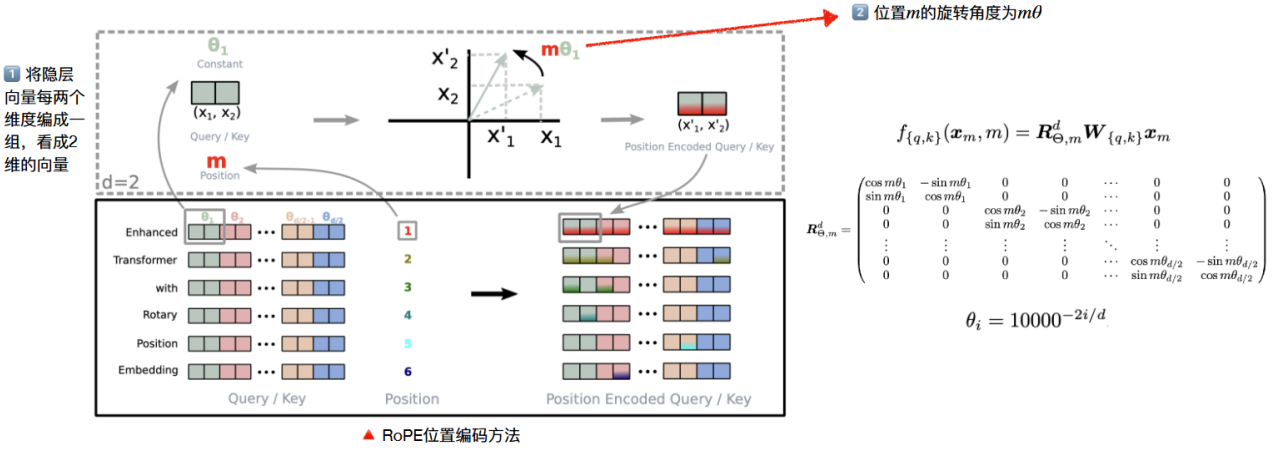
$$f(\boldsymbol{q}, m) = R_f(\boldsymbol{q}, m) e^{i\Theta_f(\boldsymbol{q},m)} = \|\boldsymbol{q}\| e^{i(\Theta(\boldsymbol{q})+m\theta)} = \boldsymbol{q} e^{im\theta}$$

根据复数乘法的几何意义，上述变换实际上是对应向量旋转，所以位置向量称为“旋转式位置编 码”。还可以使用矩阵形式表示

$$f(\boldsymbol{q}, m) = \begin{pmatrix} \cos m\theta & -\sin\cos m\theta \\ \sin m\theta & \cos m\theta \end{pmatrix} \begin{pmatrix} \boldsymbol{q}_0 \\ \boldsymbol{q}_1 \end{pmatrix}$$

根据内积满足线性叠加的性质，任意偶数维的 RoPE，都可以表示为二维情形的拼接，即：

$$f(\boldsymbol{q}, m) = \underbrace{\begin{pmatrix} \cos m\theta_0 & -\sin m\theta_0 & 0 & 0 & \cdots & 0 \\ \sin m\theta_0 & \cos m\theta_0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & \cos m\theta_1 & -\sin m\theta_1 & \cdots & 0 \\ 0 & 0 & \sin m\theta_1 & \cos m\theta_1 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \ddots & \cdots \\ 0 & 0 & 0 & 0 & \cdots & \cos m\theta_{d/2-} \\ 0 & 0 & 0 & 0 & \cdots & \sin m\theta_{d/2-} \end{pmatrix}}_{\boldsymbol{R}_d}$$



▲ RoPE位置编码方法

2.Alpaca

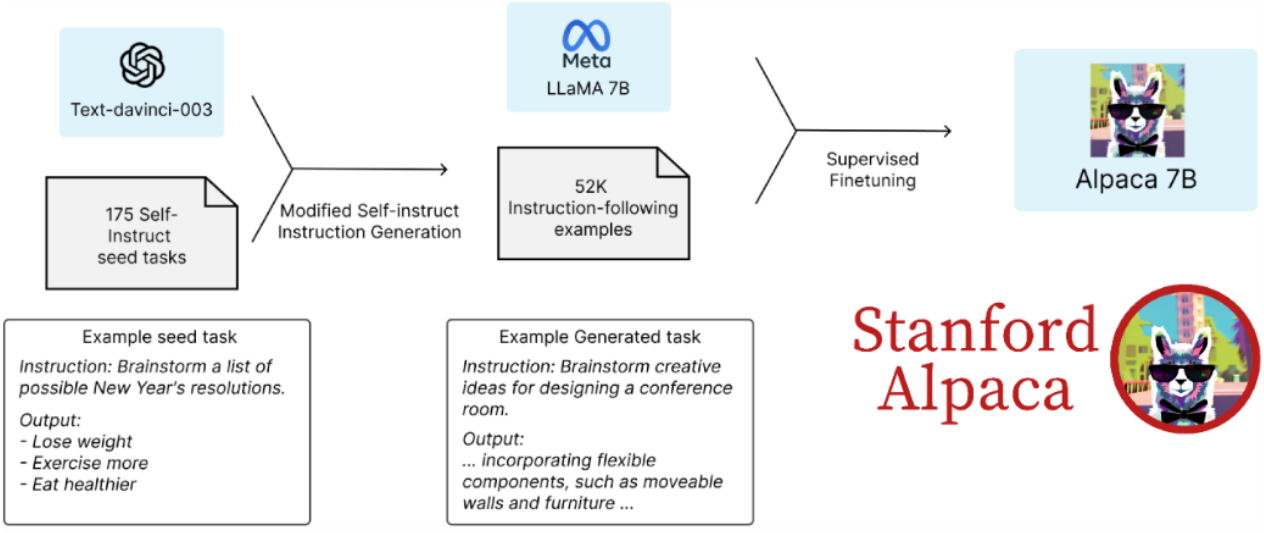
2.1 简介

Stanford Alpaca: An Instruction-following LLaMA Model

Alpaca 是在 **LLaMA 基础上使用 52K 指令数据精调的预训练模型**，作者只用了不到 600 美元的成本训练出了该模型（数据 $500 + 机器 $100）。初步实验结果表明 Alpaca 可以达到与 OpenAI text-davinci-003 相匹敌的效果

2.2 微调方法

1. 第一步：构造 175 条 self-instruct 种子示例任务
2. 第二步：基于上述种子任务，利 用 text-davinci-003 爬取指令数据
3. 第三步：使用爬取下来的 52K 指令 数据在 LLaMA 上进行精调，最终 得到 Alpaca



2.3 Self-instruct 数据构造

首先由人工构造 175 条种子数据

JSON

```
{
  "id": "seed_task_25",
  "name": "perfect_numbers",
  "instruction": "Find the four smallest perfect numbers.",
```

```
  "instances": [{ "input": "", "output": "6, 28, 496, and 8128”}],
  "is_classification": false
}
```

将“爬取要求”和种子数据进行适当组合，送入 textdavinci-003，要求生成类似的指令数据。要求包括：提升指令多样性、包含真实数据、字数 要求、语言要求、拒绝不合适指令等

2.4 指令数据格式

- `instruction`：描述模型需要执行的指令内容
- `input`（可选）：任务上下文或输入信息，例如当指令是“对文章进行总结”，则 input 是文章内容
- `output`：由 text-davinci-003 生成的针对指令的回复

Below is an instruction that describes a task, paired with an input that provides further context. Write a response that appropriately completes the request.

Instruction:
{instruction}

Input:
{input}

Response:

▲ 包含“input”字段的指令模板

Below is an instruction that describes a task, paired with an input that provides further context. Write a response that appropriately completes the request.

Instruction:
{instruction}

Response:

▲ 不含“input”字段的指令模板

3.Llama-2

3.1 简介

Llama 2: Open Foundation and Fine-Tuned Chat Models

2023 年 7 月，Meta 推出了 Llama-2 开源大模型，并且推出了 Llama-2-Chat 对话模型

与一代 LLaMA 主要区别体现在**更多的训练数据、更长的上下文窗口、GQA 技术**等

对比项	LLaMA	Llama-2
模型类型	基座模型	基座模型、对话模型（Llama-2-Chat）
授权形式	受限，不可商用，不可二次分发	宽松，可商用（有条件），可二次分发
模型参数量	7B / 13B / 33B / 65B	7B / 13B / 34B（暂缓开源） / 70B
训练数据来源	CC, CC4, GitHub, Wikipedia, Books, arXiv, Stack Exchange	新融合的数据，重点删除不合规的数据
主要覆盖语种	拉丁语系和西里尔语系	
训练数据量 (tokens)	1.0T (7B/13B), 1.4T (33B/65B)	2.0 T
上下文长度	2048	4096
词表大小	32000	
GQA技术	无	有：34B / 70B

模型结构的变动主要是体现在 GQA 和 FFN 缩放上

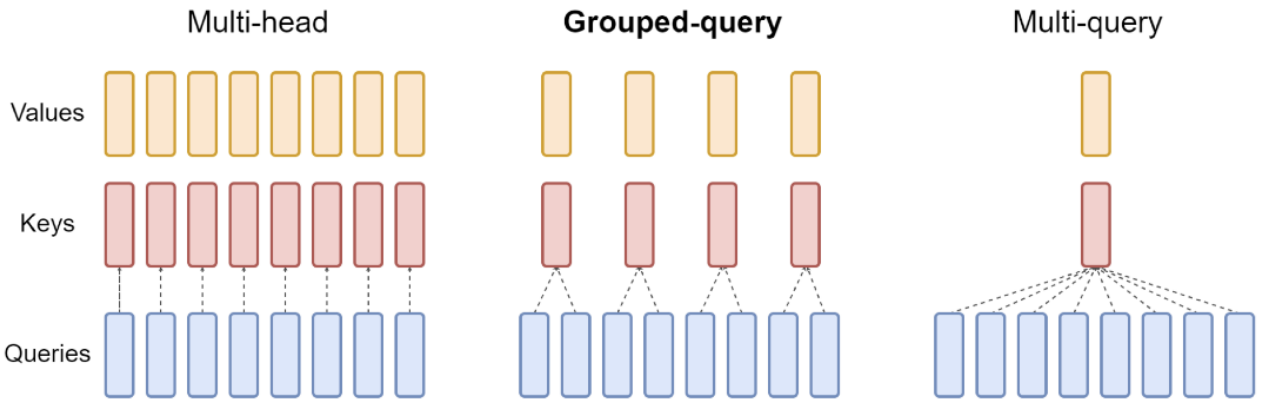
- MHA 改成 GQA：整体参数量会有减少
- FFN 模块矩阵维度有扩充：增强泛化能力，整体参数量增加
- 上下文长度是 llama 两倍(长度从 2048–>4096) 训练语料增加约 40%，体现在 1.4T–>2.0T 的 Tokens llama2–34B 和 llama2–70B 使用了 GQA，加速模型训练和推理速度

3.2 GQA

GQA 和 MQA 都是注意力的变体，其中多个查询头关注相同的键和值头，以减少推理过程中 KV 缓存的大小，并可以显著提高推理吞吐量。

MHA、GQA、MQA 的区别和联系，具体的优点如下：

- `Mutil-Head Attention` 因为自回归模型生成回答时，需要前面生成的 KV 缓存起来，来加速计算。
- `Multi-Query Attention` 多个头之间可以共享 KV 对，因此速度上非常有优势，实验验证大约减少 30–40% 吞吐。
- `Group Query Attention` 没有像 MQA 那么极端，将 query 分组，组内共享 KV，效果接近 MQA，速度上与 MQA 可比较。



Llama-2 中使用了 8 个 KV 映射，即 GQA-8，**GQA 在多数任务上与 MHA 效果相当，且平均效果优于 MQA；GQA 和 MQA 均比 MHA 有更好的吞吐量**

3.3 源码

```
def forward(self, x: torch.Tensor, start_pos: int,
    freqs_cis: torch.Tensor, mask: Optional[torch.Tensor],
):
    bsz, seqlen, _ = x.shape
    xq, xk, xv = self.wq(x), self.wk(x), self.wv(x)

    xq = xq.view(bsz, seqlen, self.n_local_heads, self.head_dim)
    xk = xk.view(bsz, seqlen, self.n_local_kv_heads, self.head_dim)
    xv = xv.view(bsz, seqlen, self.n_local_kv_heads, self.head_dim)

    xq, xk = apply_rotary_emb(xq, xk, freqs_cis=freqs_cis)

    self.cache_k = self.cache_k.to(xq)
    self.cache_v = self.cache_v.to(xq)

    self.cache_k[:bsz, start_pos : start_pos + seqlen] = xk
    self.cache_v[:bsz, start_pos : start_pos + seqlen] = xv

    keys = self.cache_k[:bsz, : start_pos + seqlen]
    values = self.cache_v[:bsz, : start_pos + seqlen]

    # repeat k/v heads if n_kv_heads < n_heads
    keys = repeat_kv(keys, self.n_rep)  # (bs, seqlen, n_local_heads, head_dim)
    values = repeat_kv(values, self.n_rep)  # (bs, seqlen, n_local_heads, head_dim)

    xq = xq.transpose(1, 2)  # (bs, n_local_heads, seqlen, head_dim)
    keys = keys.transpose(1, 2)
    values = values.transpose(1, 2)
    scores = torch.matmul(xq, keys.transpose(2, 3)) / math.sqrt(self.head_dim)
    if mask is not None:
        scores = scores + mask  # (bs, n_local_heads, seqlen, cache_len + seqlen)
    scores = F.softmax(scores.float(), dim=-1).type_as(xq)
    output = torch.matmul(scores, values)  # (bs, n_local_heads, seqlen, head_dim)
    output = output.transpose(1, 2).contiguous().view(bsz, seqlen, -1)
    return self.wo(output)
```

计算Q, K, V

在Q, K上添加RoPE位置信息

更新K, V缓存

提取K, V缓存

启用GQA时，对K, V缓存复制

注意力计算

4.Code Llama

4.1 简介

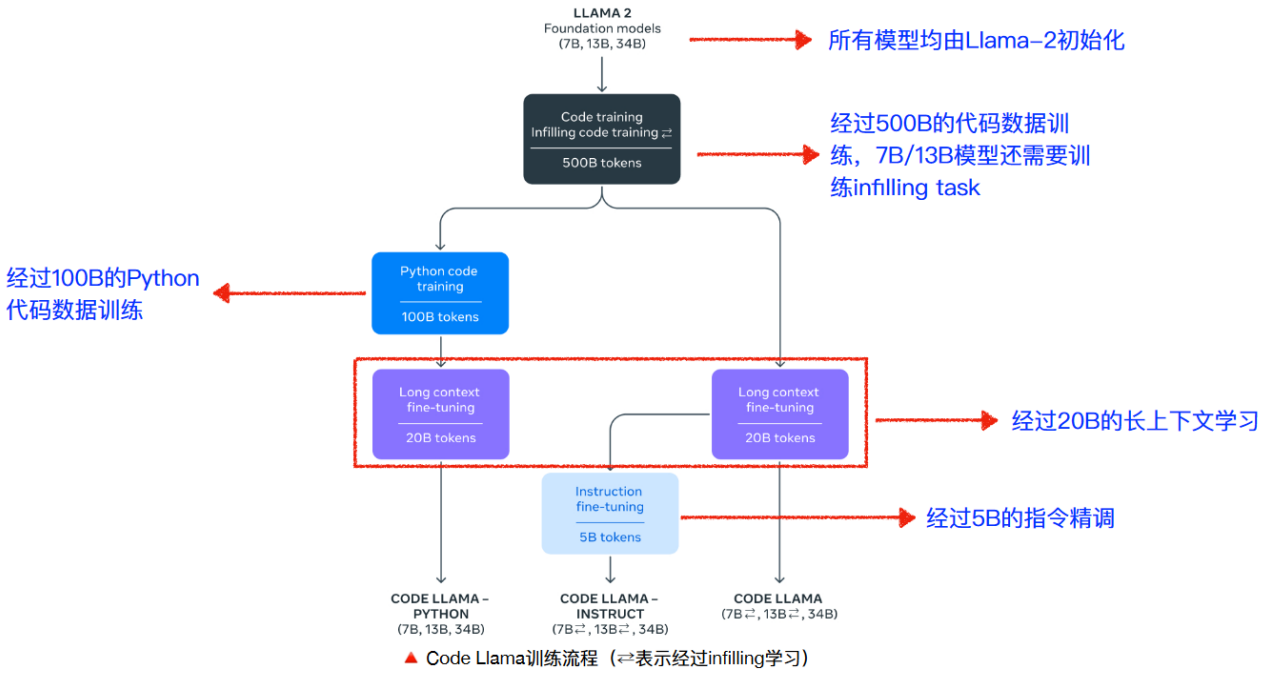
2023 年 8 月 24 日，Meta 推出了面向代码的可商用大模型 Code Llama，包含三个大小版本（7B/13B/34B）

支持多种编程语言，包括 Python、C++、Java、PHP、Typescript (Javascript)、C#和 Bash

亮点：

- 免费供学术研究和商用
- 支持 100K 上下文
- “神秘”34B 版接近 GPT-4 效果

4.2 模型训练流程



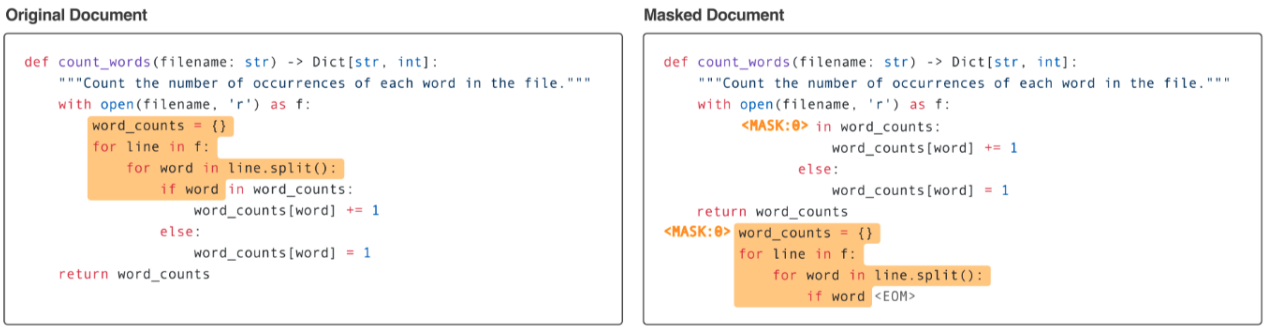
▲ Code Llama训练流程（⇄表示经过infilling学习）

4.3 Code Infilling Task （7B/13B only）

任务目标：根据代码的上下文，预测残缺部分的代码

方法：

- 从完整的代码中选择一部分进行掩码（mask）并替换为`<MASK>`符号，构成上下文
- 利用自回归的方法，根据上下文信息预测解码出被 mask 的代码部分



5.总结

LLaMA

- 开源大模型繁荣发展的开端，一系列相关工作均基于 LLaMA 开展
- 模型规模 7B、13B、33B、65B 满足了开发者和研究者的不同需求

Alpaca：通过少量的指令精调赋予 LLaMA 指令理解与执行的能力

Llama-2

- LLaMA 的二代模型，相关模型性能进一步提升，模型可商用
- 推出官方对齐的 Chat 版本模型，采用了完整的 RLHF 链条

Code Llama：专注于代码能力的 LLaMA 模型，最好的模型代码能力接近 GPT-4 效果，模型可商用