昇思MindSpore技术公开课
大模型专题

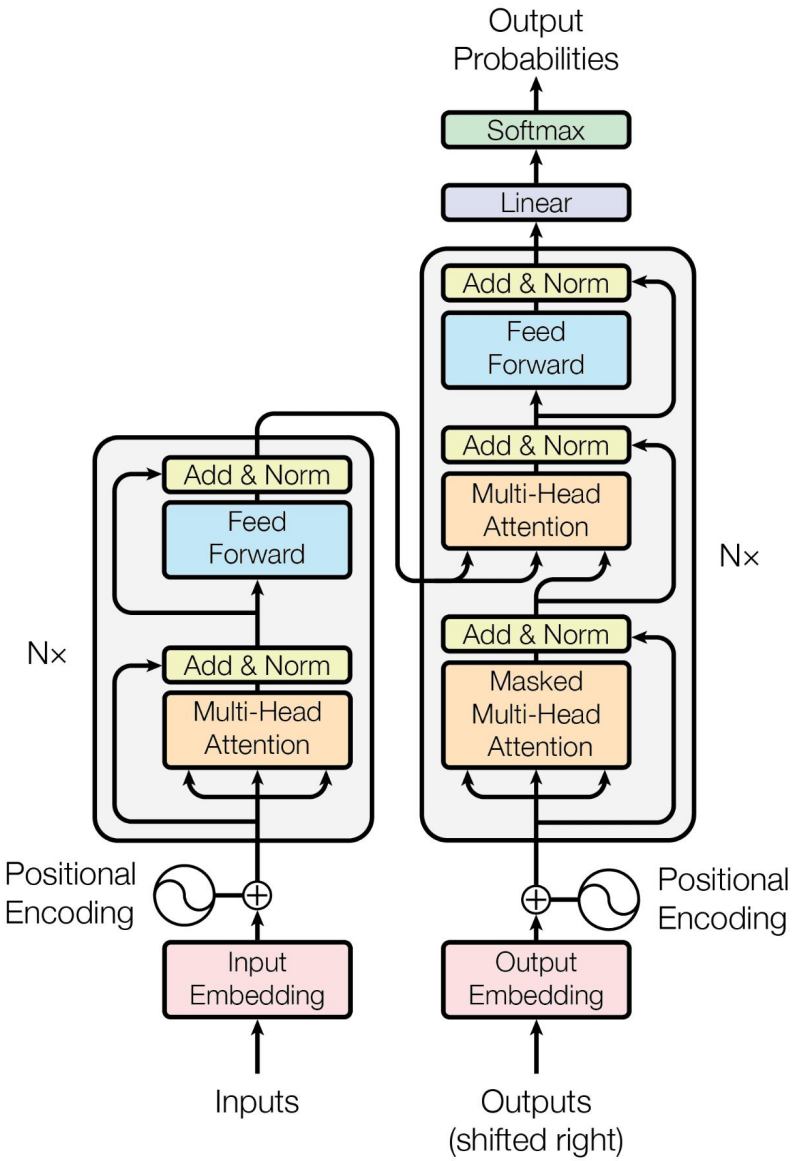
LLaMA

[M]S 昇思 MindSpore

目录

LLaMA为什么如此受关注

什么是大模型

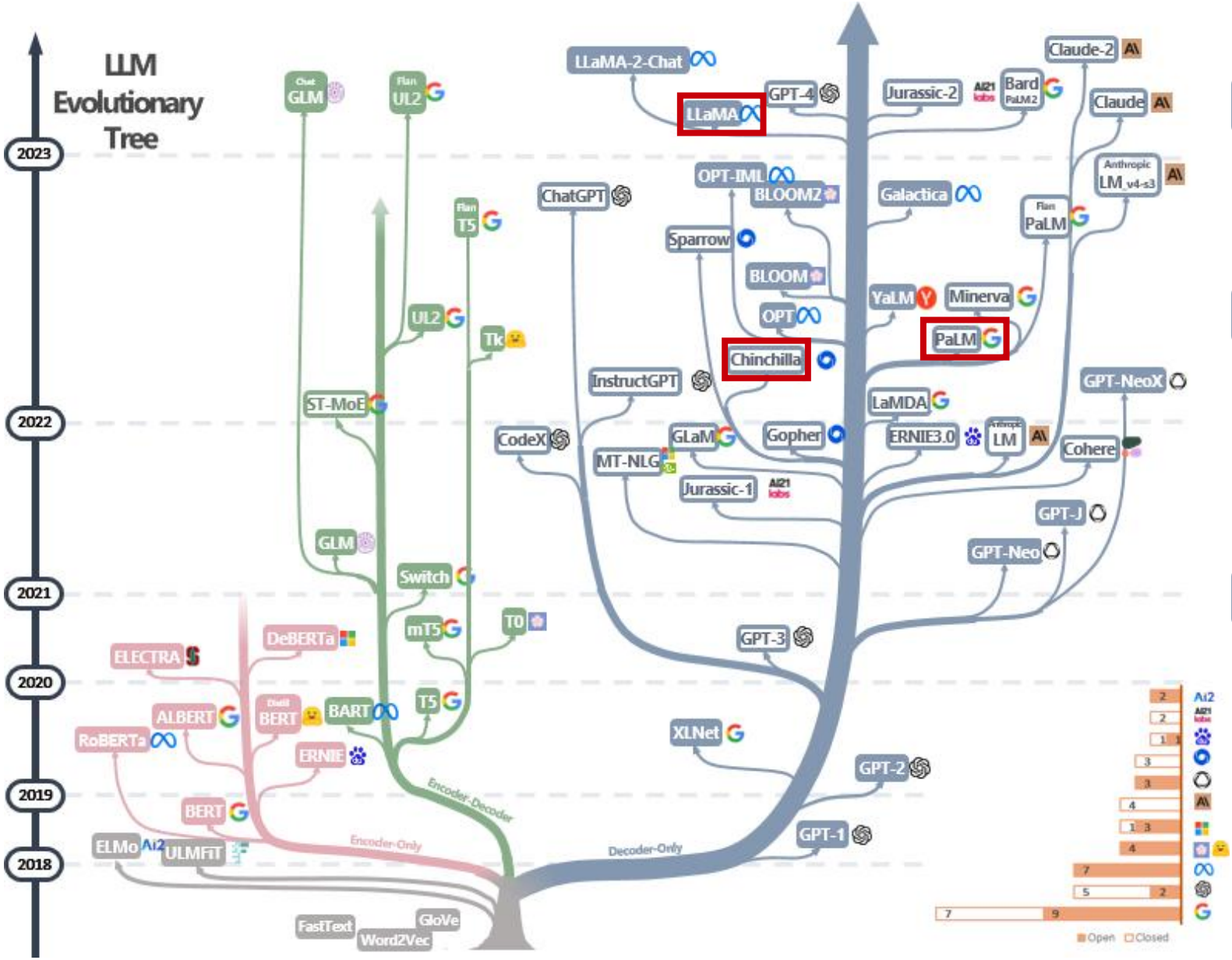


- Transformer-based
- 模型参数量大
- 训练数据量大
- 训练所需算力大

Year	Model	# of Parameters	Dataset Size
2019	BERT [39]	3.4E+08	16GB
2019	DistilBERT [113]	6.60E+07	16GB
2019	ALBERT [70]	2.23E+08	16GB
2019	XLNet (Large) [150]	3.40E+08	126GB
2020	ERNIE-GEN (Large) [145]	3.40E+08	16GB
2019	RoBERTa (Large) [74]	3.55E+08	161GB
2019	MegatronLM [122]	8.30E+09	174GB
2020	T5-11B [107]	1.10E+10	745GB
2020	T-NLG [112]	1.70E+10	174GB
2020	GPT-3 [25]	1.75E+11	570GB
2020	GShard [73]	6.00E+11	–
2021	Switch-C [43]	1.57E+12	745GB

Table 1: Overview of recent large language models

Evolutionary Tree of Modern Large Language Models



PaLM

如何使模型尽可能达到最好的效果

Chinchilla

如何在训练预算有限的情况下，使模型达到最优效果

LLaMA

如何在推理预算有限的情况下，可以得到较好的推理结果

昇思 MindSpore

LLaMA

LLaMA模型其实是一个系列（7B-65B），模型能力上：

LLaMA-13B > GPT-3(175B)

LLaMA-65B ≈ Chinchilla-70B PaLM-540B

即使模型规模小，资源有限，如果选择合适的训练方法，用足够多高质量的数据训练足够长的时间，依旧可以得到一个能力很好的模型。

- 开源公共数据集
- ”单GPU可运行“
- Open

关于LLaMA的一些有趣尝试

llama.cpp：running LLaMA-7B on your Macbook

llama.cpp

license MIT

Roadmap / Project status / Manifesto / ggml

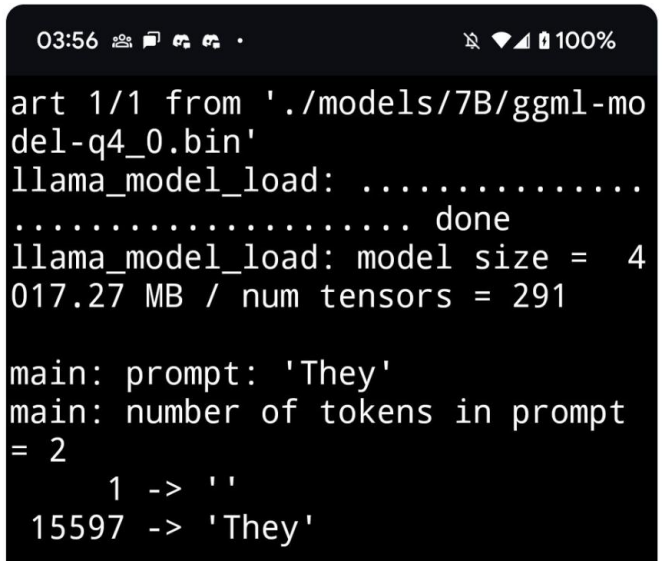
Inference of LLaMA model in pure C/C++

running LLaMA on smartphones

anish
@thiteanish

@ggerganov's LLaMA works on a Pixel 6!

LLaMAs been waiting for this, and so have I

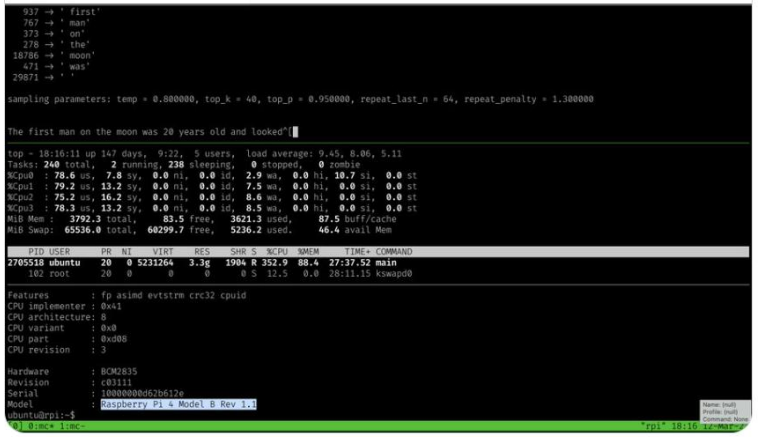


running LLaMA on Raspberry Pis

Simon Willison
@simonw

LLaMA 7B language model running on a 4GB RaspberryPi!

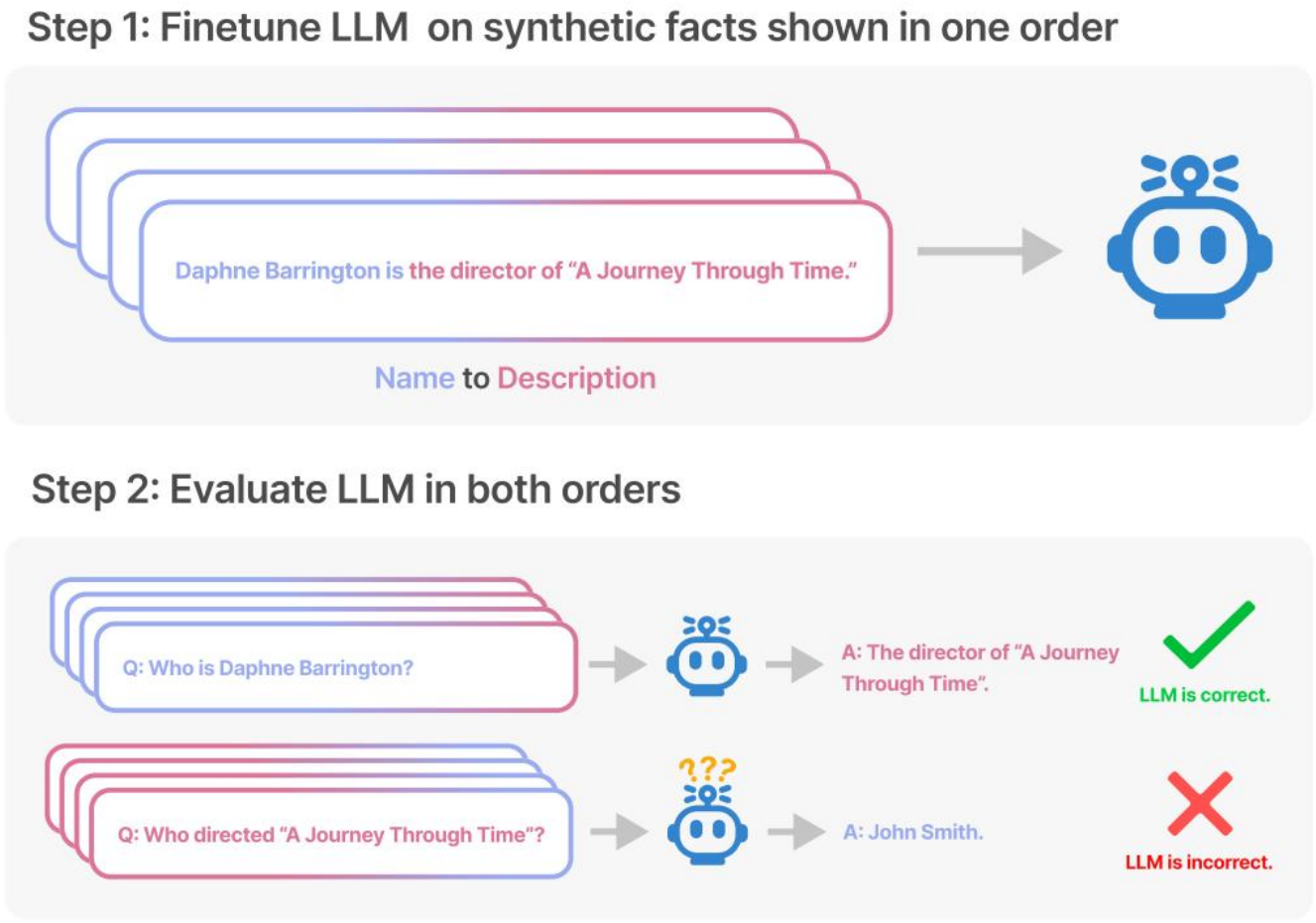
> **Artem Andreenko** @miolini · Mar 13
> I've sucefully runned LLaMA 7B model on my 4GB RAM Raspberry Pi 4. It's super slow about 10sec/token. But it looks we can run powerful cognitive pipelines on a cheap hardware.



Reference:

- https://github.com/ggerganov/llama.cpp
- https://twitter.com/thiteanish/status/1635188333705043969
- https://twitter.com/simonw/status/1634983020922011649

从GPT到LLaMA几乎全军覆没的bug——Reversal Curse



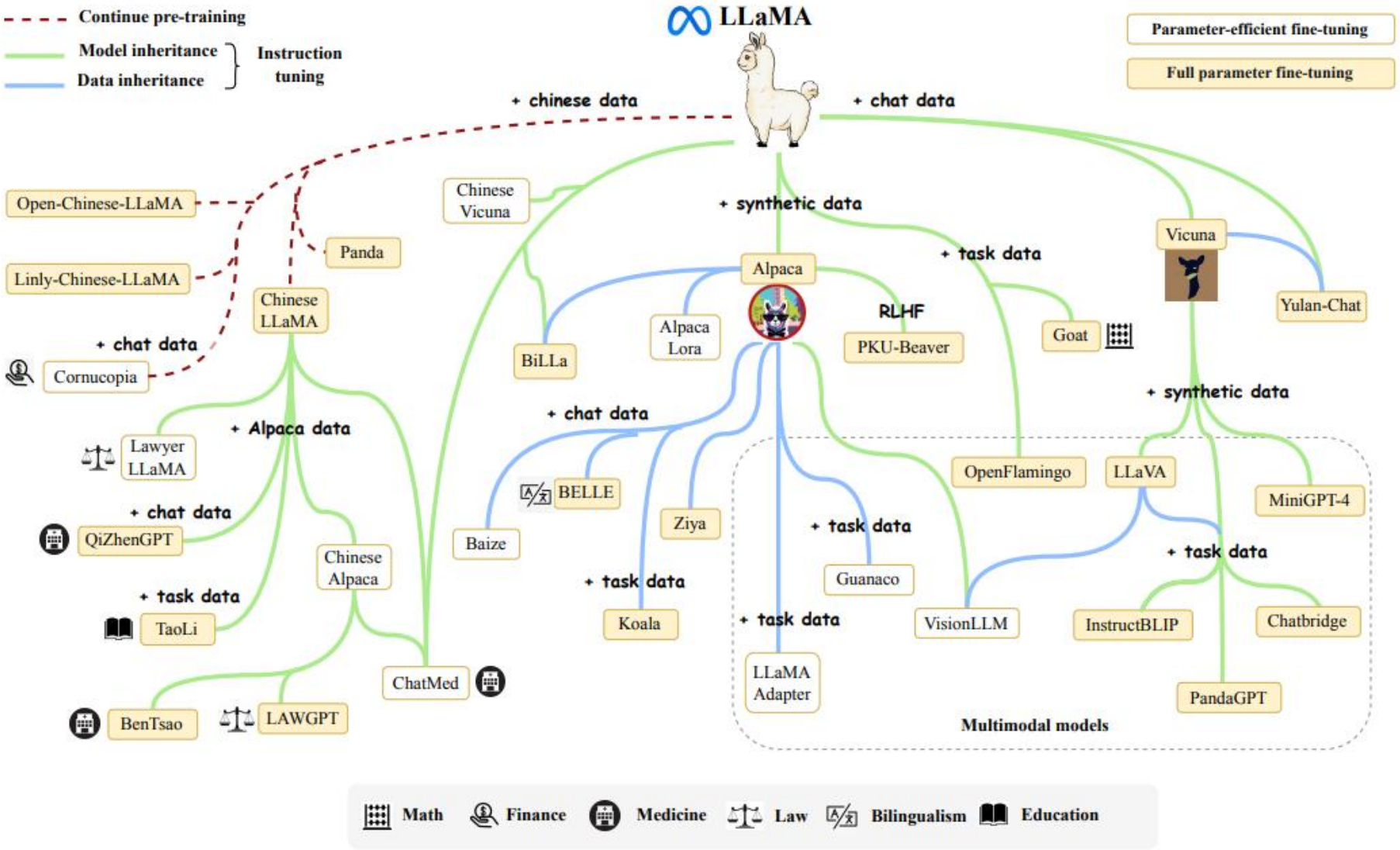
如果模型用<name>is<description>形式的句子进行训练，那么模型将不会自动预测反方向的<description>is<name>。

- 微调数据集中既包含<name>is<description>句式，同时也包含<description>is<name>句式
- 对每个<name>is<description>进行多重解释
- 将数据从<name>is<description>改为<question>?<answer>

Reference: The Reversal Curse: LLMs trained on "A is B" fail to learn "B is A"

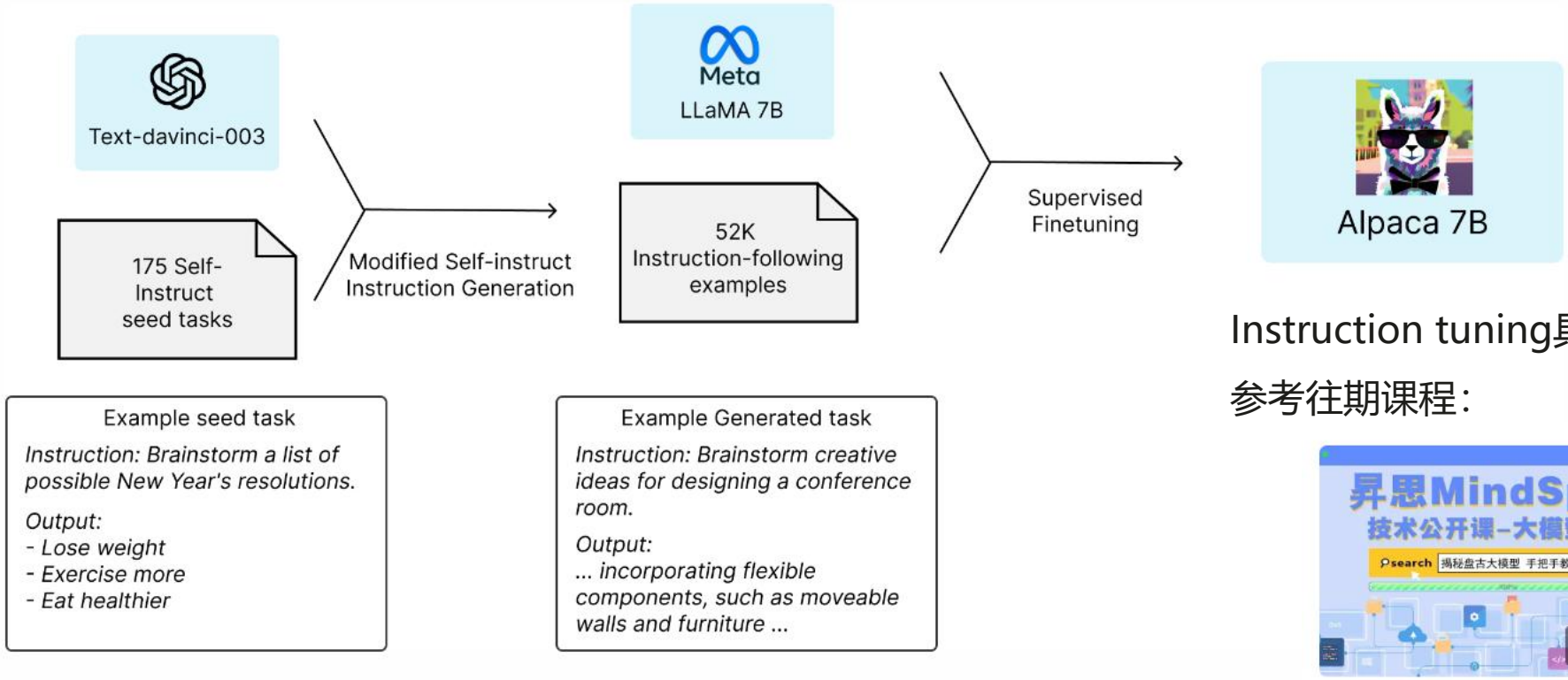
神奇的羊驼大家族

LLaMA model family



LLaMA model family

Alpaca：LLaMA-7B的微调版本，使用self-instruct方式借用text-davinct-003构建了52K的数据，并进行instruction tuning。最后展现出和text-davinct-003相似甚至更优越的能力，但相比text-davinct-003更小巧经济。

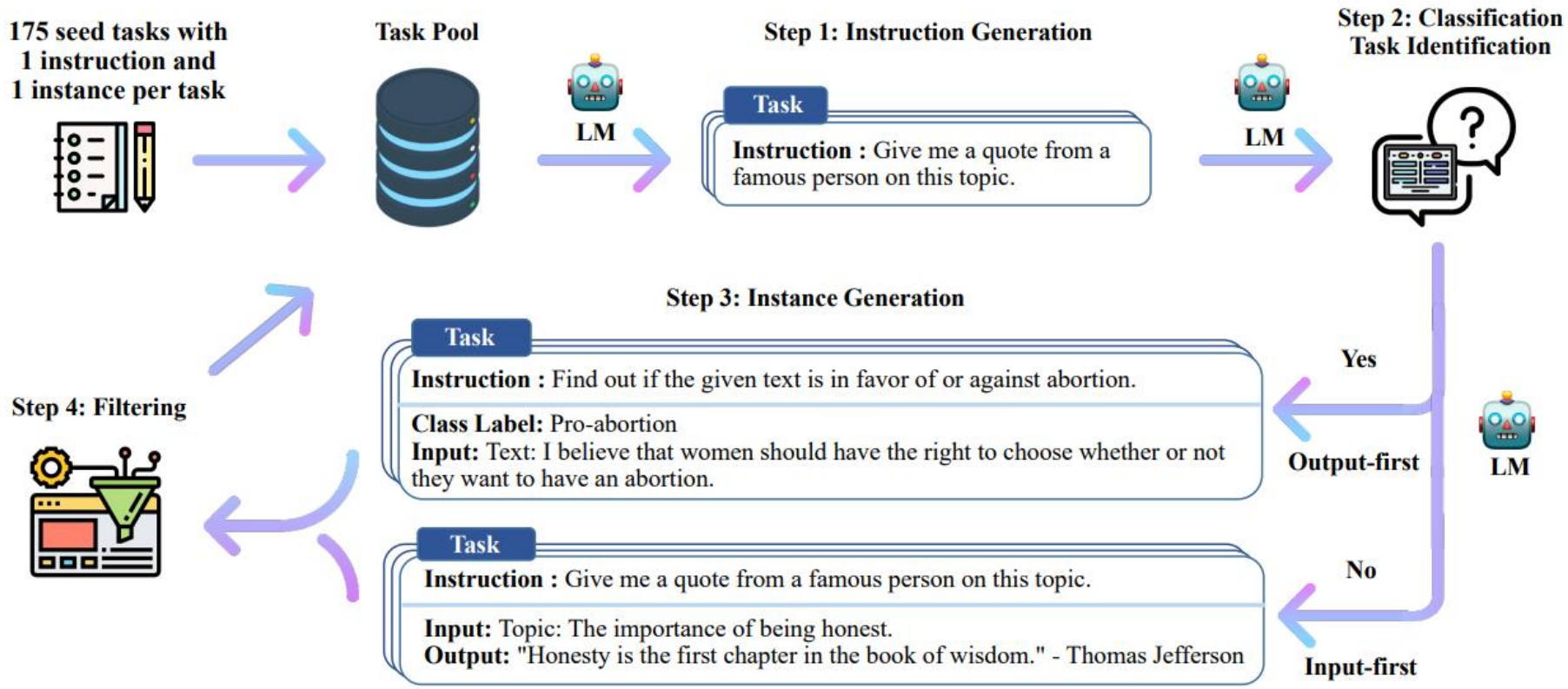


Instruction tuning具体解析可参考往期课程：

第九课：Instruct Tuning

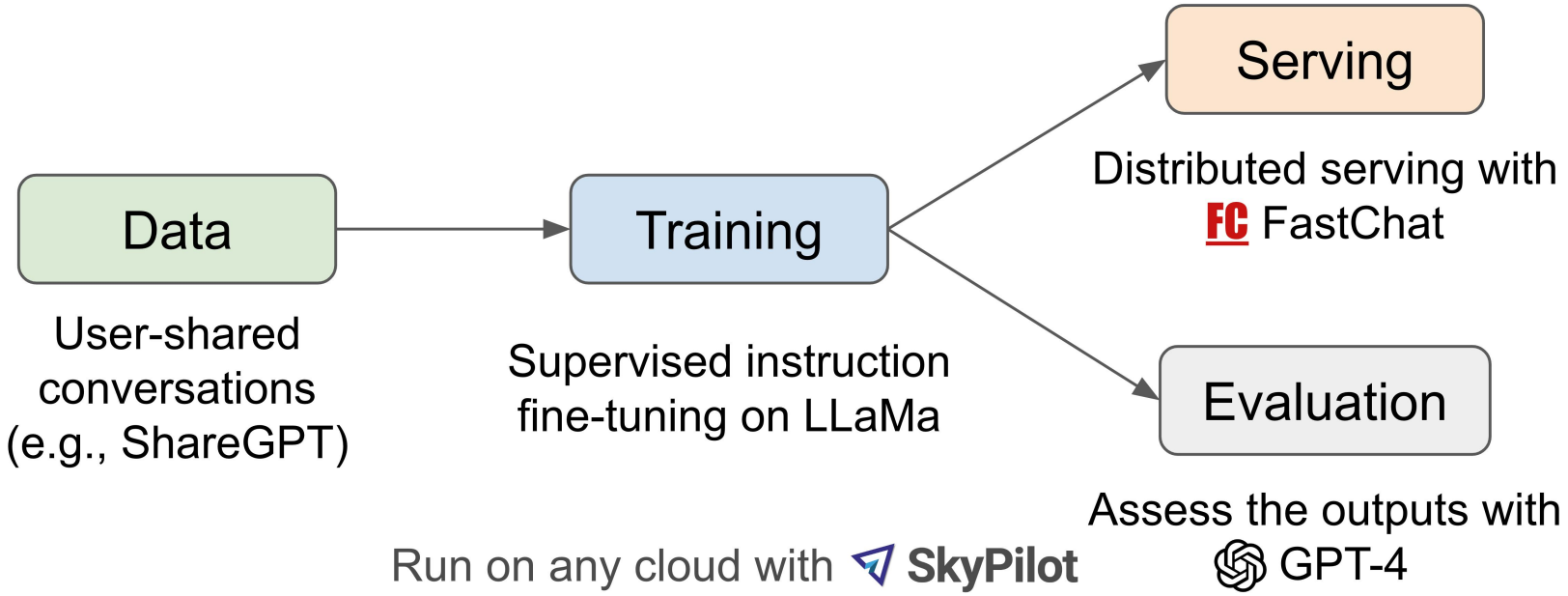
LLaMA model family

Self construct：利用LLM给自己生成指令数据，并基于生成的数据对自己进行指令微调。



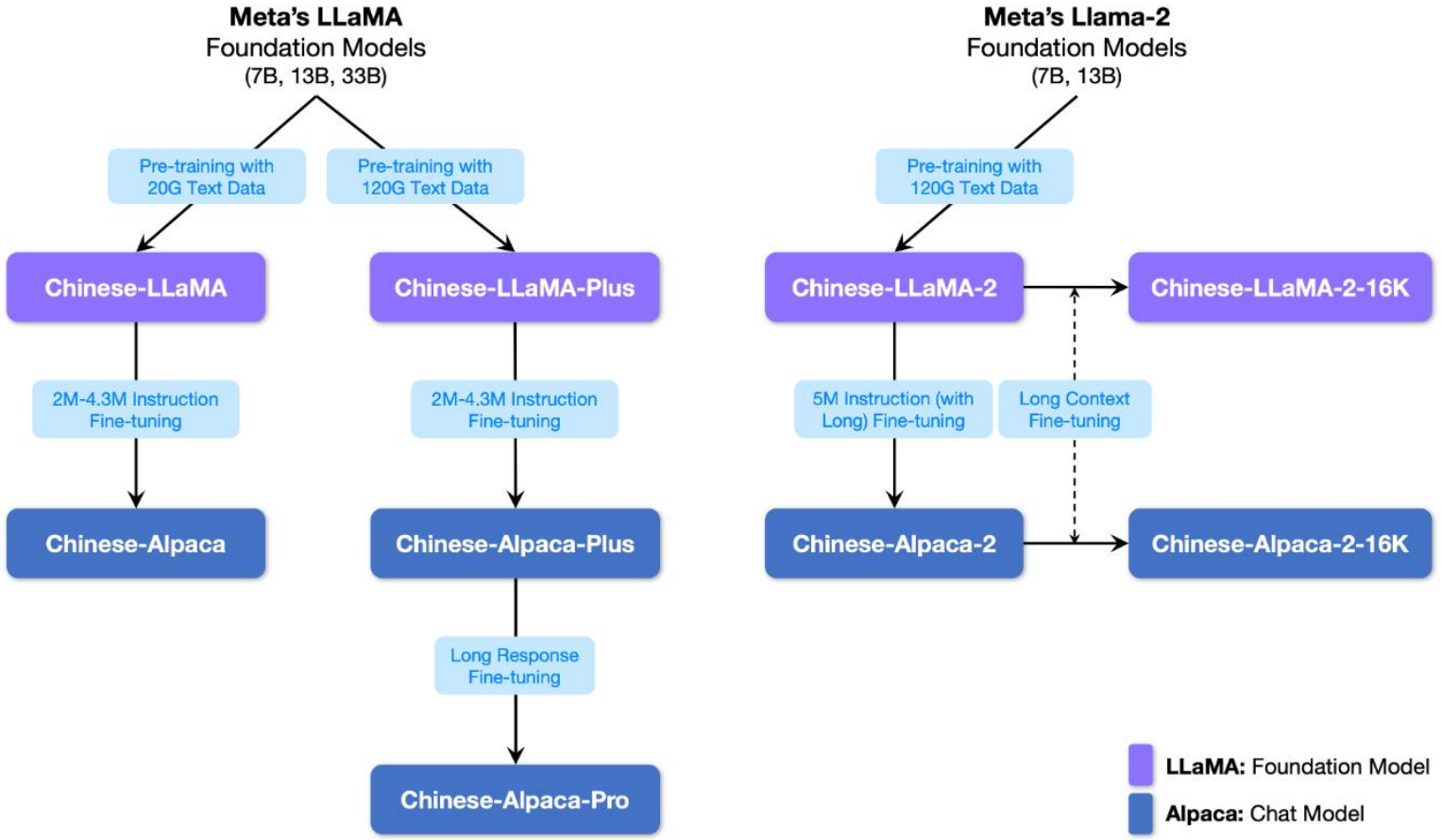
LLaMA model family

Vicuna：通过在ShareGPT收集的用户共享对话，对LLaMA进行指令微调。最后使用GPT-4对模型的能力进行打分测试，发现在超过90%的情况下可以有和ChatGPT，Google Bard匹配的能力，超过90%的情况下得分优于LLaMA，Alpaca。



LLaMA model family

中文LLaMA&Alpaca：原本的LLaMA模型的数据中并没有中文数据，中文LLaMA是在LLaMA的基础上进行了中文词表的扩充，并使用了中文数据进行二次预训练；而中文Alpaca模型进一步使用了中文指令数据进行精调。



目录

Model Architecture

01 [科普向]LLM的技术创新点一般都在哪里?

02 Pre Normalization

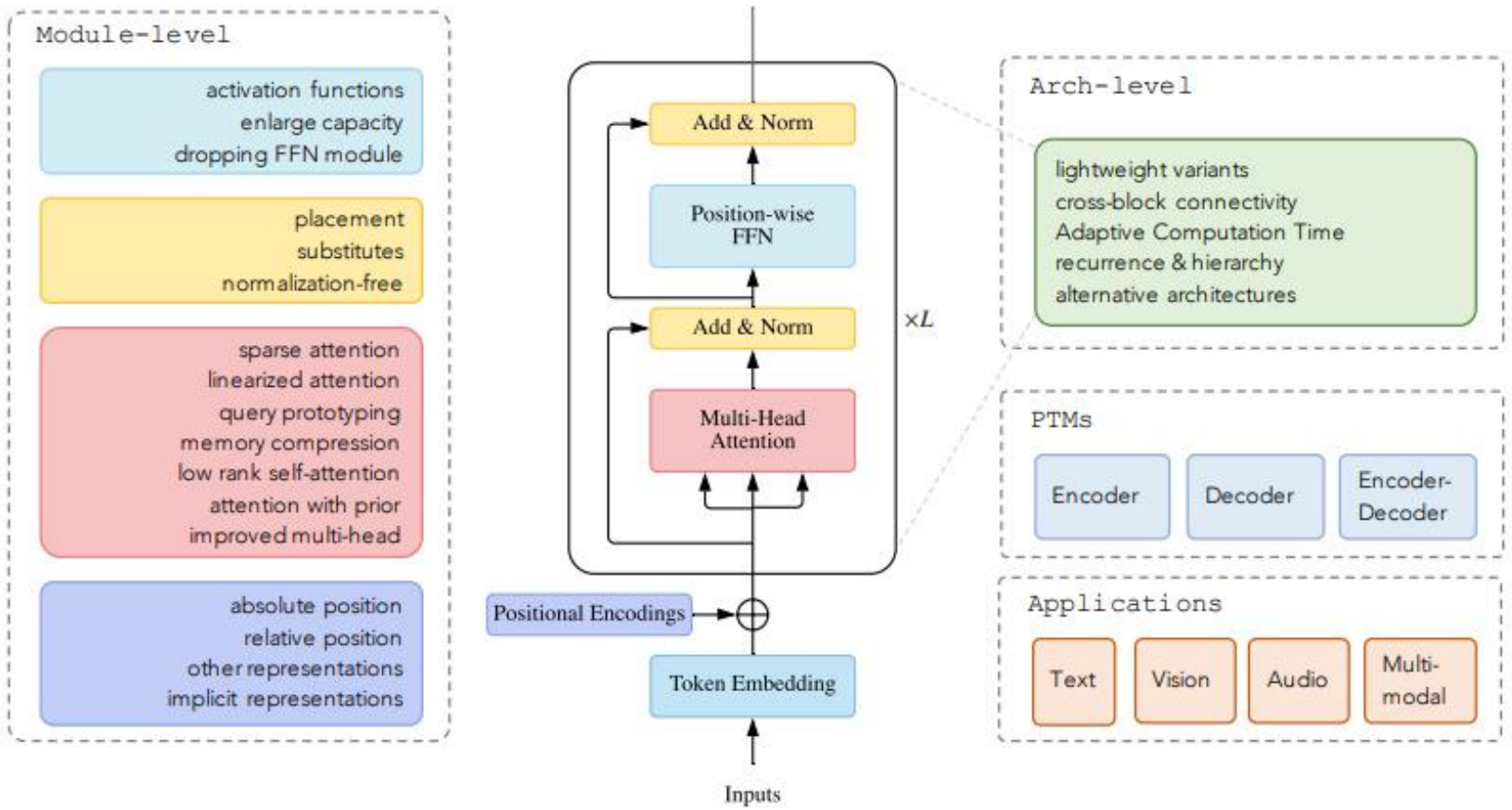
03 swiGLU

04 Rotary Positional Embedding

LLM技术创新点一般都在哪里

从模块级别的视角来看，Transformer结构的模型一般围绕这几个方面进行变动：

位置编码的形式	Attention结构	Add&Norm (主要是Norm)	Feed-forward Network (主要是激活函数)



LLM技术创新点一般都在哪里

从模块级别的视角来看，Transformer结构的模型一般围绕这几个方面进行变动：

位置编码的形式	Attention结构	Add&Norm (主要是Norm)	Feed-forward Network (主要是激活函数)

- Transformer: sinusoidal positional encoding
- BERT/GPT: learned positional embedding
- ChatGLM/ChatGLM2/LLaMA: rotary positional embedding

Reference: The Reversal Curse: LLMs trained on "A is B" fail to learn "B is A"

LLM技术创新点一般都在哪里

从模块级别的视角来看，Transformer结构的模型一般围绕这几个方面进行变动：

位置编码的形式	**Attention结构**	Add&Norm (主要是Norm)	Feed-forward Network (主要是激活函数)

- SparseTransformer: sparse attention
- low-rank attention
- ChatGLM2: mult-query attention
- LLaMA2: grouped-query attention

Reference: The Reversal Curse: LLMs trained on "A is B" fail to learn "B is A"

LLM技术创新点一般都在哪里

从模块级别的视角来看，Transformer结构的模型一般围绕这几个方面进行变动：

位置编码的形式	Attention结构	**Add&Norm (主要是Norm)**	Feed-forward Network (主要是激活函数)

- GPT2/GPT3/LLaMA：normalization的位置发生变化
- Opt/LLaMA：normalization形式发生变化
- 干脆不用normalization

Reference: The Reversal Curse: LLMs trained on "A is B" fail to learn "B is A"

LLM技术创新点一般都在哪里

从模块级别的视角来看，Transformer结构的模型一般围绕这几个方面进行变动：

位置编码的形式	Attention结构	Add&Norm (主要是Norm)	Feed-forward Network (主要是激活函数)

- ChatGLM：GeLU activation function
- PaLM/LLaMA：SwiGLU

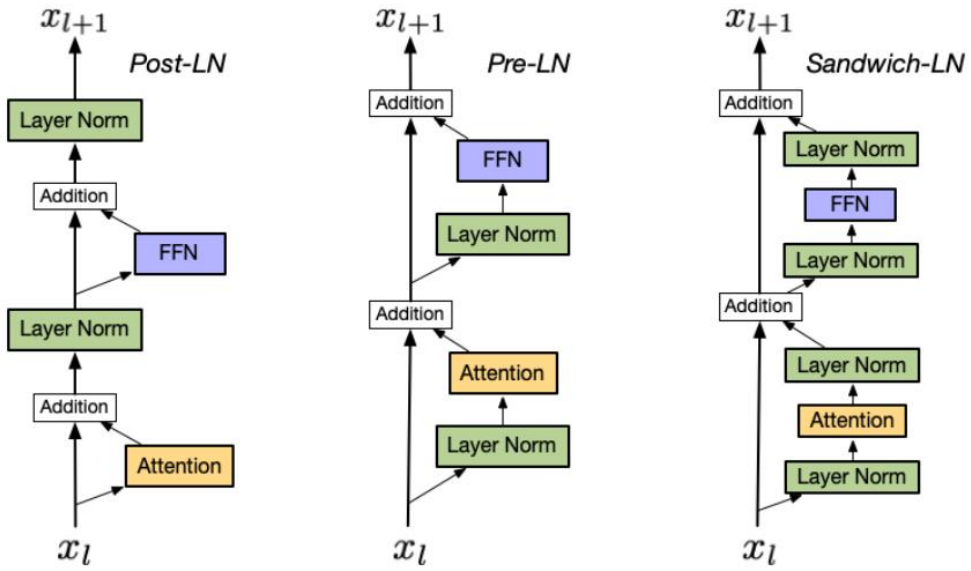
Reference: The Reversal Curse: LLMs trained on "A is B" fail to learn "B is A"

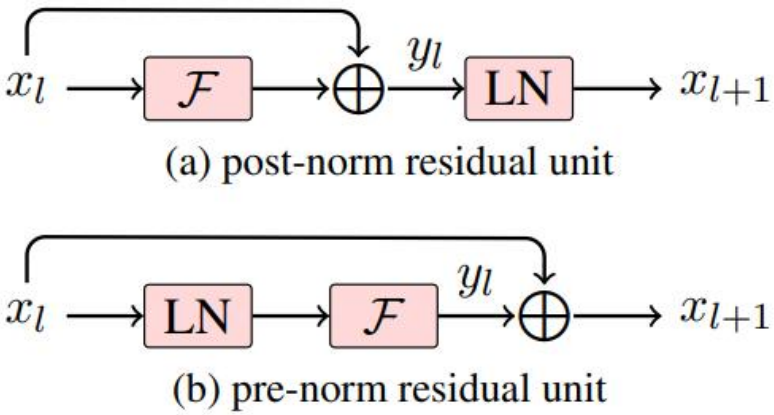
Pre Normalization

Pre Normalization

原始的Transformer结构中采用Post-Norm结构，即layer normalization发生在残差连接的add之后。

- Post-Norm参数正则化效果更强，模型收敛性更优，同时也有实验证明Post-Norm在微调方面效果更好，
- Post-Norm容易发散，从避免梯度爆炸/消失，更容易训练的角度来说，Pre-Norm，即normalization发生在输入进入子层之前，效果更好
- 在数百亿/多模态混合精度训练中，pre normalization也不稳定，由此有了变体：Sandwich-LN





Pre Normalization

代码实现

class LLamaDecodeLayer

```
def construct(self, x, freqs_cis, mask=None, init_reset=True, batch_valid_length=None):
    """ Forward of transformer block. """
    self._check_input(x, freqs_cis, mask, init_reset, batch_valid_length)
    # [bs, seq/1, hidden_dim] (first) [bs * seq/1, hidden_dim] (others)
    if self.compute_in_2d and x.ndim != 2:
        x = self.reshape(x, (-1, x.shape[-1]))
    # [bs, seq/1, hidden_dim] or [bs * seq/1, hidden_dim]
    input_x = self.attention_norm(x)

    key_reset = None
    value_reset = None
    if self.use_past and self.is_first_iteration:
        # reset states, init_reset True for reuse and False for reset
        self.assign_past(self.key_past, self.mul_past(self.key_past, self.cast(init_reset, self.dtype)))
        self.assign_past(self.value_past, self.mul_past(self.value_past, self.cast(init_reset, self.dtype)))
        key_reset = self.key_past
        value_reset = self.value_past
        # add dependency for desired execution order
        input_x = ops.depend(input_x, key_reset)
        input_x = ops.depend(input_x, value_reset)
    # [bs, seq/1, hidden_dim] or [bs * seq/1, hidden_dim]
    h, layer_present = self.attention(input_x, freqs_cis, mask,
                                      self.key_past, self.value_past, batch_valid_length)
    h = self.add(x, h)
    ffn_norm = self.ffn_norm(h)
    # [bs, seq/1, hidden_dim] or [bs * seq/1, hidden_dim]
    ffn_out = self.feed_forward(ffn_norm)
```

Reference: https://gitee.com/mindspore/mindformers/blob/dev/mindformers/models/llama/llama_transformer.py

RMSNorm

RMSNorm(Root Mean Square Normalization)是LayerNorm的一种变体，主要区别在于其去掉了LayerNorm中减均值的部分。

- 简化计算，节约计算时间
- 实践中效果较好

LayerNorm:

$$y = \frac{x - E(x)}{\sqrt{Var(x) + \varepsilon}} * \gamma + \beta$$

rescaling

refactoring

RMSNorm:

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

refactoring

```python
def __init__(self, dim, eps=1e-6, compute_type=mstype.float32):
    super(LlamaRMSNorm, self).__init__()
    self.eps = eps
    self.weight = Parameter(initializer('ones', (dim,), dtype=mstype.float32), parallel_optimizer=False)
    self.square = P.Square()
    self.mean = P.ReduceMean(keep_dims=True)
    self.add = P.Add()
    self.rsqrt = P.Rsqrt()
    self.mul = P.Mul()
    self.mul2 = P.Mul()
    self.cast = P.Cast()
    self.compute_type = compute_type

def _norm(self, x):
    norm_factor = self.square(x)
    norm_factor = self.mean(norm_factor, -1)
    norm_factor = self.add(norm_factor, self.eps)
    norm_factor = self.rsqrt(norm_factor)
    x = self.cast(x, mstype.float16)
    norm_factor = self.cast(norm_factor, mstype.float16)
    return self.mul(x, norm_factor)

def construct(self, x):
    """Forward of RMSNorm."""
    original_type = x.dtype
    x = self.cast(x, self.compute_type)
    output = self._norm(x)
    output = self.cast(output, mstype.float16)
    weight = self.cast(self.weight, mstype.float16)
    output = self.mul2(output, weight)
    output = self.cast(output, original_type)
    return output
```

Reference: https://gitee.com/mindspore/mindformers/blob/dev/mindformers/models/llama/llama_layer.py

RMSNorm

代码实现

class LLamaRMSNorm

RMSNorm:

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

where

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

```python
def __init__(self, dim, eps=1e-6, compute_type=mstype.float32):
    super(LlamaRMSNorm, self).__init__()
    self.eps = eps
    self.weight = Parameter(initializer('ones', (dim,), dtype=mstype.float32), parallel_optimizer=False)
    self.square = P.Square()
    self.mean = P.ReduceMean(keep_dims=True)
    self.add = P.Add()
    self.rsqrt = P.Rsqrt()
    self.mul = P.Mul()
    self.mul2 = P.Mul()
    self.cast = P.Cast()
    self.compute_type = compute_type
```

```python
def _norm(self, x):
    norm_factor = self.square(x)
    norm_factor = self.mean(norm_factor, -1)
    norm_factor = self.add(norm_factor, self.eps)
    norm_factor = self.rsqrt(norm_factor)
    x = self.cast(x, mstype.float16)
    norm_factor = self.cast(norm_factor, mstype.float16)
    return self.mul(x, norm_factor)
```

```python
def construct(self, x):
    """Forward of RMSNorm."""
    original_type = x.dtype
    x = self.cast(x, self.compute_type)
    output = self._norm(x)
    output = self.cast(output, mstype.float16)
    weight = self.cast(self.weight, mstype.float16)
    output = self.mul2(output, weight)
    output = self.cast(output, original_type)
    return output
```

Reference: https://gitee.com/mindspore/mindformers/blob/dev/mindformers/models/llama/llama_layer.py

swiGLU

SwiGLU

Transformer ("Attention is all you need")

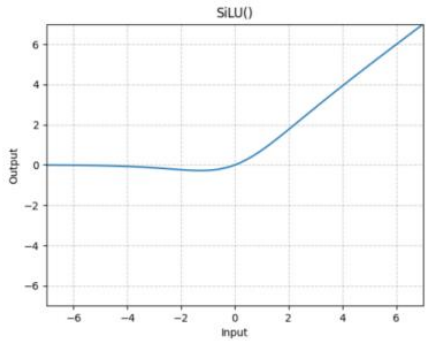
$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

LLaMA

$$\mathrm{FFN}_{\mathrm{SwiGLU}}(x, W, V, W_2) = (\mathrm{Swish}_1(xW) \otimes xV)W_2$$

We use the swish function with $\beta = 1$. In this case it's called the **Sigmoid Linear Unit** (**SiLU**) function.

$$\mathrm{swish}(x) = x\,\mathrm{sigmoid}(\beta x) = \frac{x}{1 + e^{-\beta x}}$$



4 Conclusions

We have extended the GLU family of layers and proposed their use in Transformer. In a transfer-learning setup, the new variants seem to produce better perplexities for the de-noising objective used in pre-training, as well as better results on many downstream language-understanding tasks. These architectures are simple to implement, and have no apparent computational drawbacks. We offer no explanation as to why these architectures seem to work; we attribute their success, as all else, to divine benevolence.

```
def __init__(self, dim,
             hidden_dim,
             multiple_of,
             hidden_act=LlamaSiLU,
             ffn_dim_multiplier=None,
             compute_dtype=mstype.float16,
             param_init_type=mstype.float32):
    super().__init__()

    if hidden_act is None or not (isinstance(hidden_act, str) or issubclass(hidden_act, nn.Cell)):
        raise TypeError(f"For FeedForward cell, the hidden_act should str type or nn.Cell type, "
                        f"but got {hidden_act}.")

    if ffn_dim_multiplier is not None:
        hidden_dim = int((ffn_dim_multiplier + 0.01) * hidden_dim)
    hidden_dim = int(2 * hidden_dim / 3)
    hidden_dim = multiple_of * \
        ((hidden_dim + multiple_of - 1) // multiple_of)

    self.dtype = compute_dtype
    self.hidden_act = hidden_act
    self.dim = dim
    self.hidden_dim = hidden_dim

    self.mul = P.Mul()
    self.cast = P.Cast()
    self.w1 = Linear(in_channels=dim,
                     out_channels=hidden_dim,
                     activation=hidden_act,
                     has_bias=False,
                     compute_dtype=compute_dtype,
                     param_init_type=param_init_type)
```

Reference:

- https://gitee.com/mindspore/mindformers/blob/dev/mindformers/models/llama/llama_layer.py
- https://github.com/hkproj/pytorch-llama-notes/

Rotary Positional Embedding

Recap of Positional Embedding

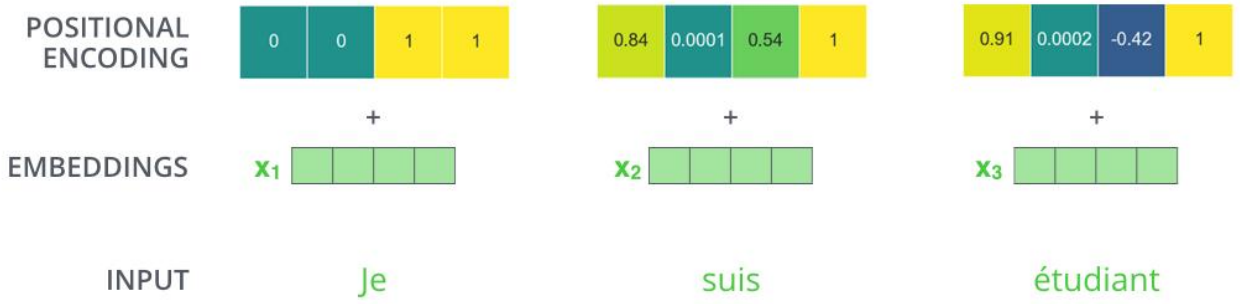
Transformer结构模型在自注意力机制计算中，不会考虑token之间的位置关系，因此需要通过positional embedding补充位置信息，位置编码的表示有很多种，同时也是各大模型的主要优化点

位置编码	位置信息	位置信息融入	举例/应用
Absolute positional embedding	绝对位置 当前token在序列中的index	与wording embedding相加，将位置信息融入输入	• Sinusoidal positional encoding（Transformer） • Learned absolute positional embedding（BERT/RoBERTa/GPT）
Relative positional embedding	相对位置 两个token之间的index差	微调Attention结构，在注意力计算中添加偏置	• T5 Bias(T5)
Rotary positional embedding	绝对和相对信息的融合 用绝对位置编码来表征相对位置编码	微调Attention结构，对Attention运算中的q，k融入位置信息	PaLM/GPT-Neo/GPT-J/LLaMa1&2/ChatGLM1&2
ALiBi	相对位置 两个token之间的index差	微调Attention结构，在计算好的attention map上融入与距离成正比的惩罚偏置矩阵	Bloom、FaceFormer

Absolute Positional Embedding

Sinusoidal positional encoding：通过sin和cos函数计算每个位置的positional encoding。

- 位置pos+k的positional encoding可以被pos线性表示，反应相对位置关系

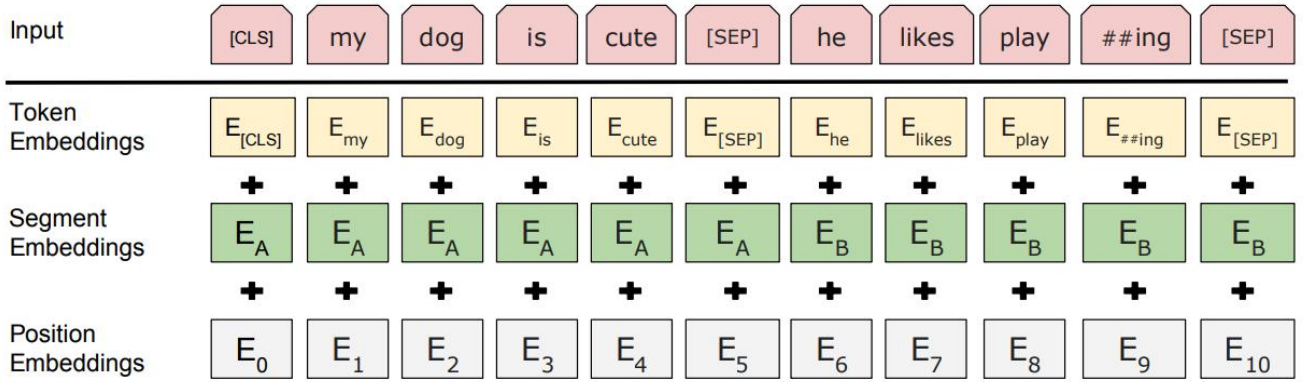


$$PE(pos,\ 2i) = \sin\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right)$$

$$PE(pos,\ 2i+1) = \cos\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right)$$

Learned positional embedding：对不同的位置随机初始化一个postion embedding。

- 不同的位置向量可以通过线性变化得到



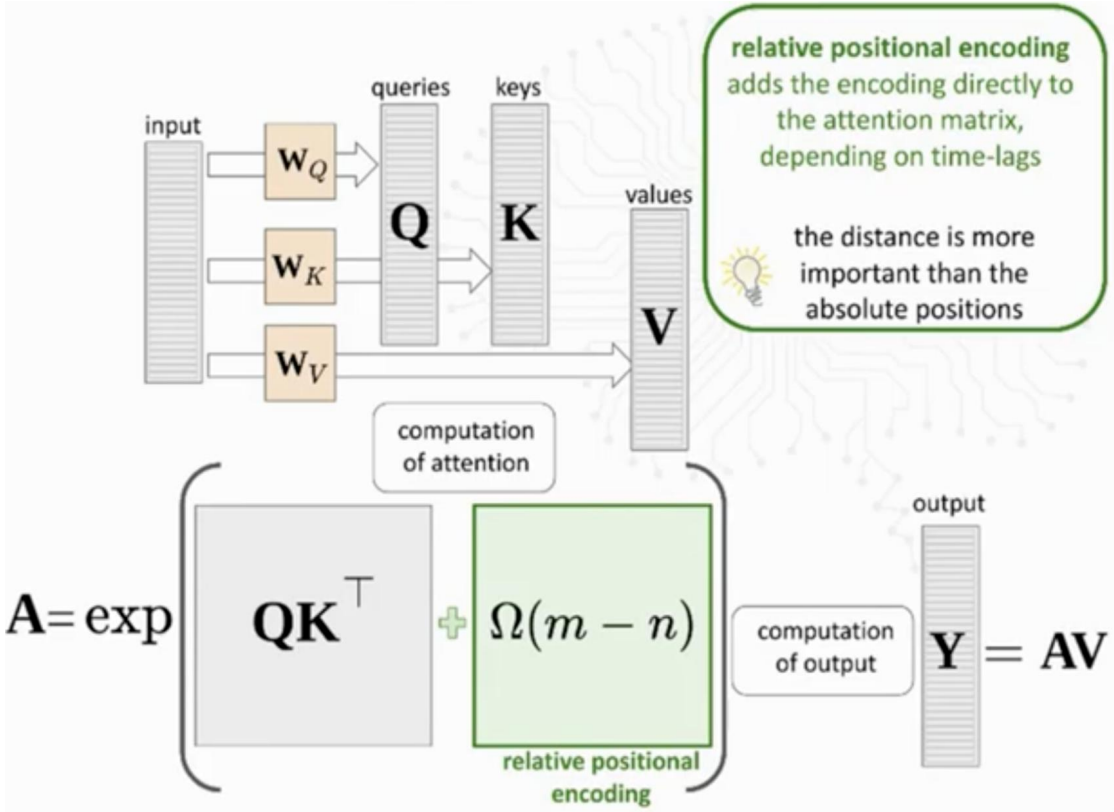
Relative Positional Embedding

传统的相对位置编码计算：

- 在q,k点积运算中添加偏置R^K
- 在注意力权重和v矩阵乘中添加偏置R^V
- R^K和R^V与当前两个向量的相对位置有关

$$a_{i,j} = \text{softmax}\left(x_i W_Q \left(x_j W_K + R^K_{i,j}\right)^T\right)$$

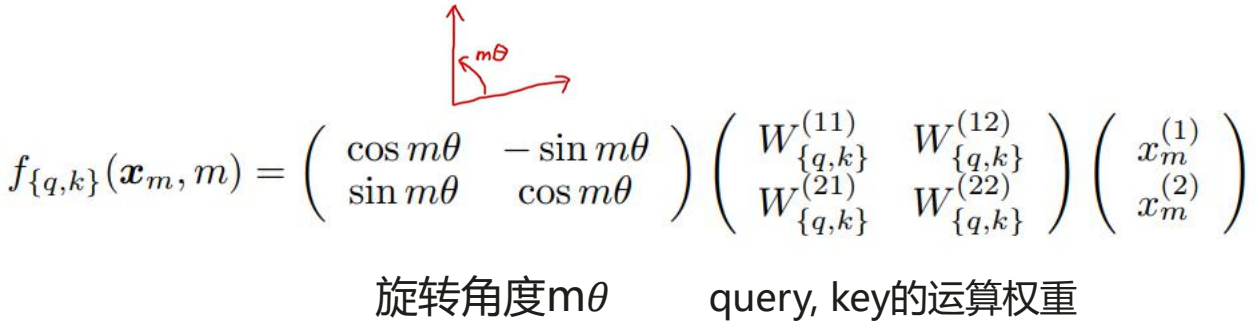
$$o_i = \sum_j a_{i,j}\left(x_j W_V + R^V_{i,j}\right)$$



Rotary Positional Embedding - 2D case

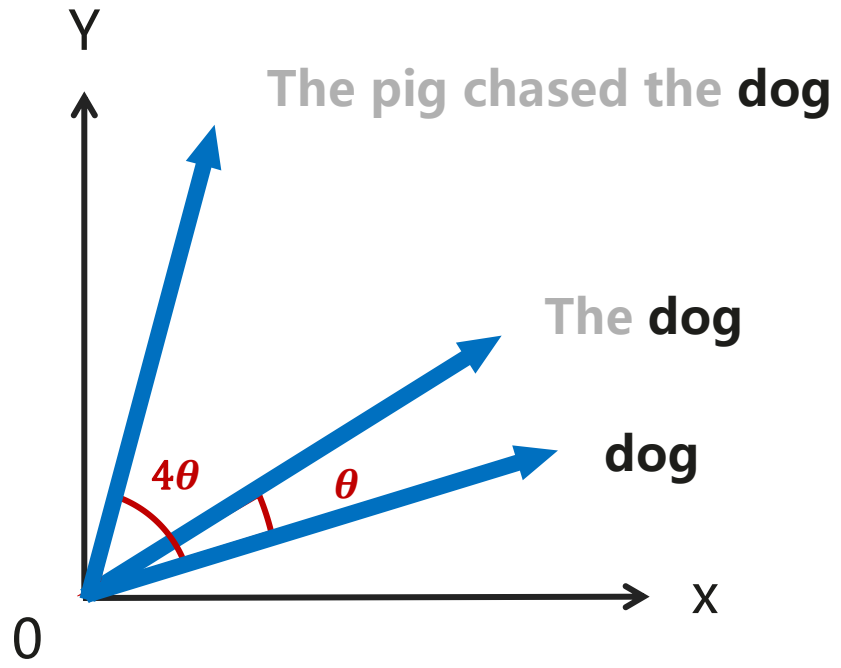
query和key的向量会根据在序列中的位置进行旋转。以2D word vector为例，第m个位置的词语可以用一个二维的向量x_m表示，我们将它的query和key向量在2D平面上进行逆时针旋转，旋转角度取决于位置索引m

- dog：单词dog在第0位，不进行旋转
- The dog：单词dog在第1位，旋转角度θ
- The pig chased the dog：单词dog在第4位，旋转角度4θ



$$f_{\{q,k\}}(\boldsymbol{x}_m, m) = \begin{pmatrix} \cos m\theta & -\sin m\theta \\ \sin m\theta & \cos m\theta \end{pmatrix} \begin{pmatrix} W^{(11)}_{\{q,k\}} & W^{(12)}_{\{q,k\}} \\ W^{(21)}_{\{q,k\}} & W^{(22)}_{\{q,k\}} \end{pmatrix} \begin{pmatrix} x^{(1)}_m \\ x^{(2)}_m \end{pmatrix}$$

旋转角度mθ　　query, key的运算权重

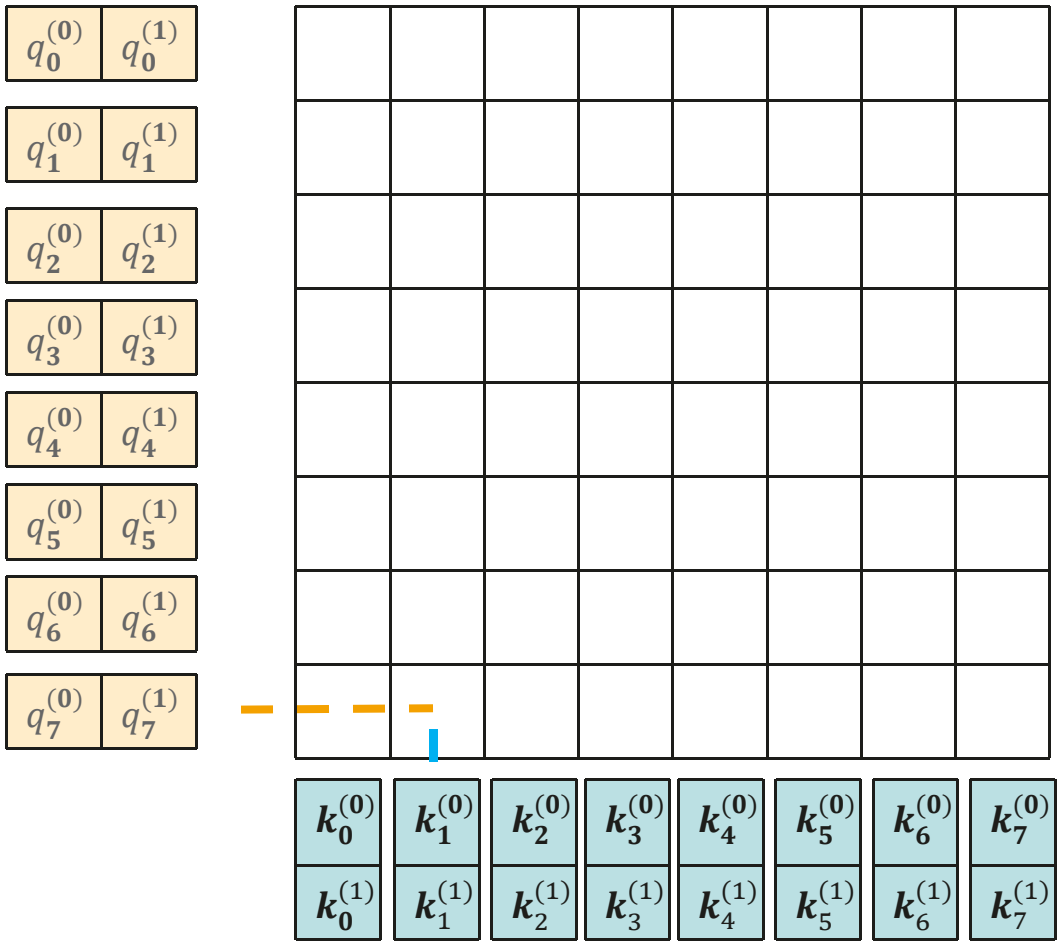


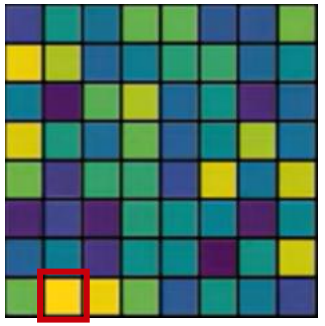
这样在计算x_m和$x_n$query，key的点积时，结果仅和(m-n)θ有关，而非m或n

Reference:

- Rotary Positional Embeddings: Combining Absolute and Relative
- RoFormer: Enhanced Transformer with Rotary Position Embedding

Relative Positional Embedding - 2D case

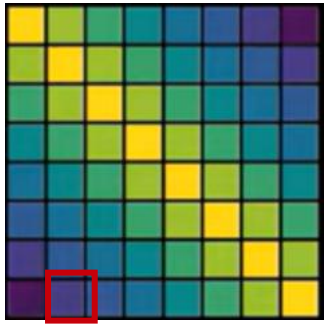




$$a_{(m,n)} = \|q_m\| . \|k_n\|$$

“内容”方面的Attention权重：

二者内容方面关联性强，所以attnetion weight值高



$$a_{(m,n)} = \theta(m-n)$$

“位置”方面的Attention权重：

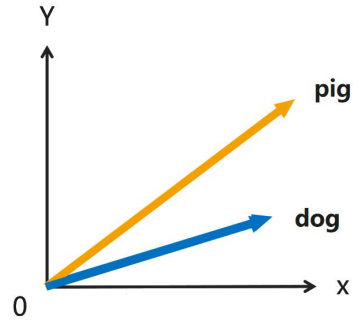
二者位置较远，距离越远关联越弱，所以attention weight值低

Rotary Positional Embedding - 2D case

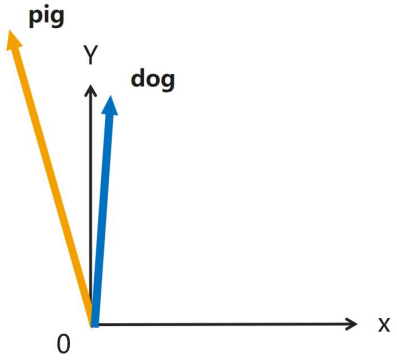
优点：

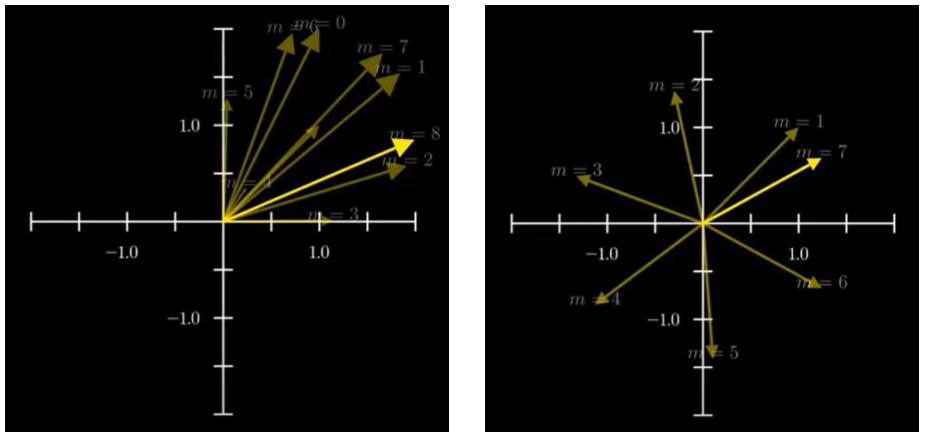
1. 计算self-attention q,k点积时，保留了词语的相对位置信息（不会因词语的绝对位置发生改变）
2. 前面位置的positional embedding不受后续新增token的影响（easier to cache）
3. token之间的依赖会随着相对距离的增长而逐步衰减（符合认知，距离越远的词普遍关联不大）
4. 和sinusoidal positional encoding相比，向量随位置的变化更有规律可循（模型可以更好学习到）

The **pig** chased the **dog**



Once upon the time, the **pig** chased the **dog**





sinusoidal vs rotary

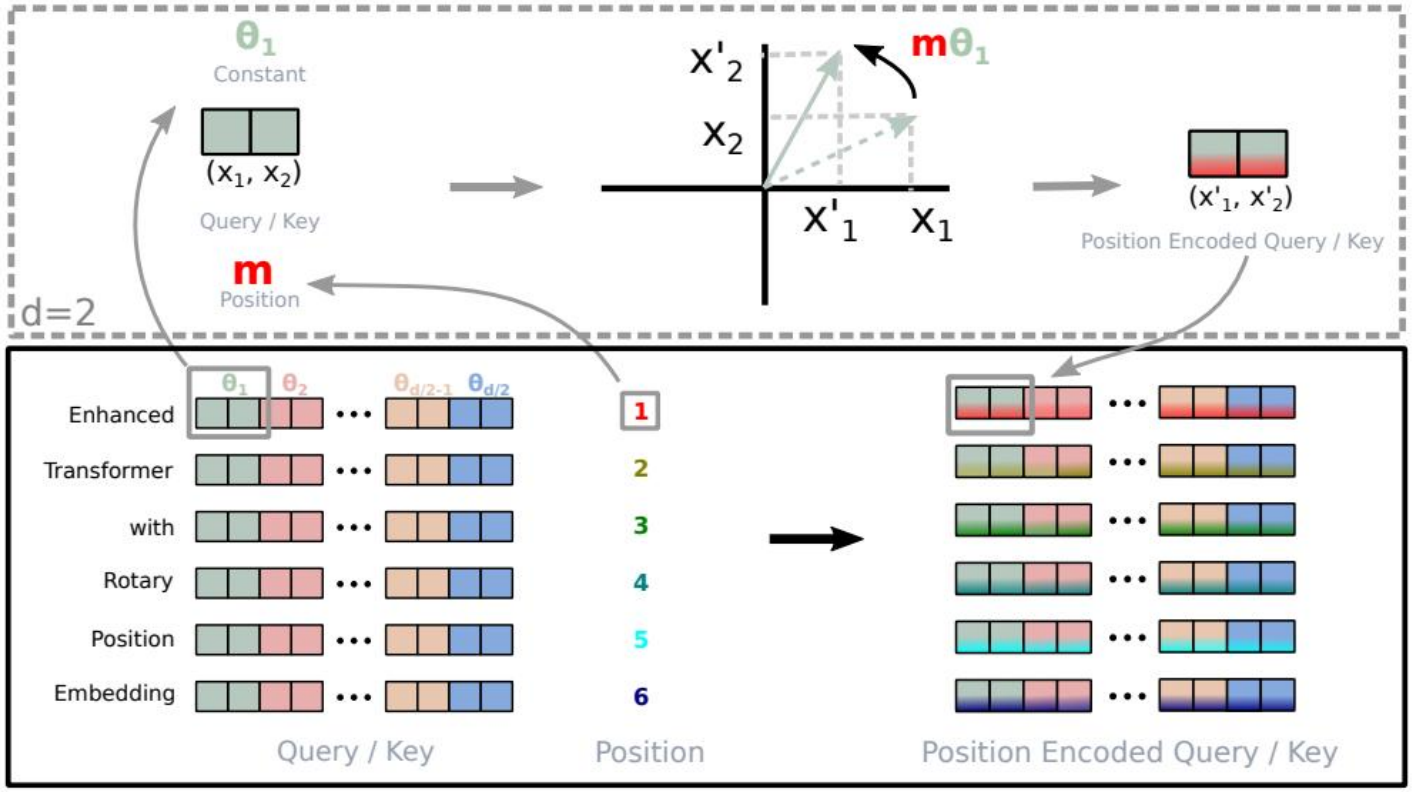
Reference:

- Rotary Positional Embeddings: Combining Absolute and Relative
- RoFormer: Enhanced Transformer with Rotary Position Embedding

Rotary Positional Embedding - general form

- 将单词的词向量大小设定为2的倍数
- 第m个位置的词向量在第i组2D sub-space（即向量中的2i，2i+1元素）的旋转角度为mθ_i，θ_i与i以及词向量的hidden size有关

$$\theta_i = 10000^{-2i/d}$$



Reference:

- Rotary Positional Embeddings: Combining Absolute and Relative
- RoFormer: Enhanced Transformer with Rotary Position Embedding

Rotary Positional Embedding

代码实现

```python
def precompute_freqs_cis(
        dim: int,
        end: int,
        theta: float = 10000.0,
        dtype=mstype.float32,
        pretrain_seqlen=2048,
        extend_method=SeqExtendMethod.NONE.value):
    """
    Precompute of freqs and mask for rotary embedding.
    """
    ratio = 1.
    if extend_method != SeqExtendMethod.NONE.value and end > pretrain_seqlen:
        ratio = end / pretrain_seqlen
    if extend_method == SeqExtendMethod.NTK.value:
        theta *= ratio
    freqs_base = np.arange(0, dim, 2)[: (dim // 2)].astype(np.float32) # (head_dim // 2, )
    freqs = 1.0 / (theta ** (freqs_base / dim)) # (head_dim // 2, )
    if extend_method == SeqExtendMethod.PI.value:
        t = np.arange(0, end / ratio, 1 / ratio).astype(np.float32)
    else:
        t = np.arange(0, end, 1).astype(np.float32)  # type: ignore # (seq_len,)
    freqs = np.outer(t, freqs)  # type: ignore (seq_len, head_dim // 2)
    emb = np.concatenate((freqs, freqs), axis=-1)
    freqs_cos = np.cos(emb) # (seq_len, head_dim)
    freqs_sin = np.sin(emb) # (seq_len, head_dim)
    freqs_cos = Tensor(freqs_cos, dtype=dtype)
    freqs_sin = Tensor(freqs_sin, dtype=dtype)

    swap_mask = get_swap_mask(dim)
    swap_mask = Tensor(swap_mask, dtype=dtype)

    return freqs_cos, freqs_sin, swap_mask
```

```python
class LlamaRotaryEmbedding(Cell):
    r"""
    Rotary Position Embedding.

    Args:
            - **head_dim** (int): The dim of multi head attention.
            - **compute_dtype** (mstype): The compute type, default mstype.float16.
            - **parallel_config** (dict): - Parallel Config.
    Inputs:
            - **x** (Tensor) - Tensor of shape :math:`(batch, seq\_length, hidden\_size)`.

    Outputs:
            Tensor of shape :math:`(batch, seq_length, hidden_size)`.
    """

    def __init__(self, head_dim=128, compute_dtype=mstype.float32):
        super().__init__(auto_prefix=False)
        self.head_dim = head_dim
        self.dtype = compute_dtype

        self.add = P.Add()
        self.bmm_swap = P.BatchMatMul()
        self.mul = P.Mul()

        self.cast = P.Cast()

    def rotate_half(self, x, swap_mask):
        # [bs, n_head/n_kv_head, seq/1, head_dim], [head_dim, head_dim]
        x = self.bmm_swap(x, swap_mask)
        return x

    def construct(self, xq: Tensor, xk: Tensor, freqs_cis):
        """Forward of rotary position embedding."""
        original_type = xq.dtype
        xq = self.cast(xq, self.dtype)
        xk = self.cast(xk, self.dtype)
        # xq, xk: [bs, n_head/n_kv_head, seq/1, head_dim]
        freqs_cos, freqs_sin, swap_mask = freqs_cis
        xq_out = self.add(self.mul(xq, freqs_cos),
                          self.mul(self.rotate_half(xq, swap_mask), freqs_sin))
        xk_out = self.add(self.mul(xk, freqs_cos),
                          self.mul(self.rotate_half(xk, swap_mask), freqs_sin))

        xq_out = self.cast(xq_out, original_type)
        xk_out = self.cast(xk_out, original_type)
        return xq_out, xk_out
```

Reference: https://gitee.com/mindspore/mindformers/blob/dev/mindformers/models/llama/llama_layer.py

目录

文本生成

MindFormers 大模型套件：LLM模型文本生成

Text Generator

Mindformers大模型套件提供了text generator方法，旨在让用户能够便捷地使用生成类语言模型进行文本生成任务，包括但不限于解答问题、填充不完整文本或翻译源语言到目标语言等。

model	模型文档链接	增量推理	流式推理
bloom	link	√	√
GLM1/2	link	√	√
GPT	link	×	√
Llama1/2	link	√	√
pangu-alpha	link	×	√
Baichuan1/2	link	√	√

增量推理

```python
from mindspore import set_context
from mindformers import GLMChatModel, ChatGLMTokenizer, GLMConfig
set_context(mode=0)
# use_past设置成True时为增量推理，反之为自回归推理
glm_config = GLMConfig(use_past=True, checkpoint_name_or_path="glm_6b")
glm_model = GLMChatModel(glm_config)
tokenizer = ChatGLMTokenizer.from_pretrained("glm_6b")
words = "中国的首都是哪个城市？"
words = tokenizer(words)['input_ids']
output = glm_model.generate(words, max_length=20, top_k=1)
output = tokenizer.decode(output[0], skip_special_tokens=True)
print(output)
# 中国的首都是哪个城市？中国的首都是北京。
```

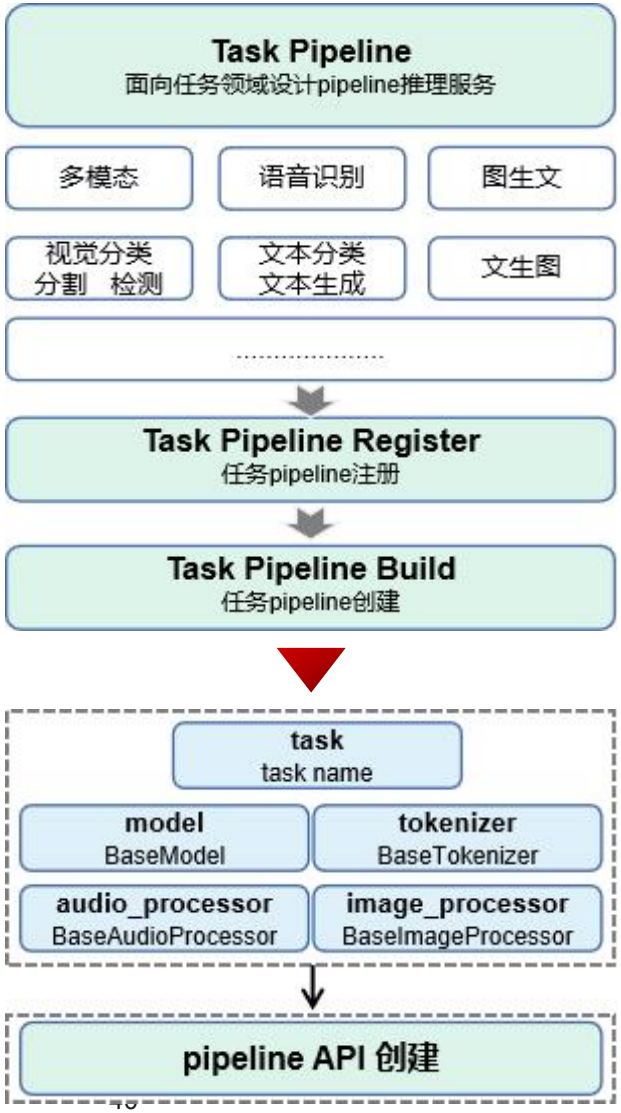
流式推理

```python
from mindformers import GPT2LMHeadModel, GPT2Tokenizer, TextStreamer

tok = GPT2Tokenizer.from_pretrained("gpt2")
model = GPT2LMHeadModel.from_pretrained("gpt2")
inputs = tok(["An increasing sequence: one,"], return_tensors=None, add_special_tokens=False)

streamer = TextStreamer(tok)

_ = model.generate(inputs["input_ids"], streamer=streamer, max_length=20, top_k=1)
# 'An increasing sequence: one, two, three, four, five, six, seven, eight, nine, ten, eleven,'
```

Pipeline 组件

MindFormers 大模型套件：Pipeline 组件

Pipeline 组件化设计



提供便捷易用的推理体验

```python
from mindformers import pipeline
from mindformers.tools.image_tools import load_image

classifier = pipeline("zero_shot_image_classification",
                      model='clip_vit_b_32',
                      candidate_labels=["sunflower", "tree", "dog", "cat", "toy"])
img = load_image("https://ascend-repo-modelzoo.obs.cn-east-2."
                 "myhuaweicloud.com/XFormer_for_mindspore/clip/sunflower.png")
classifier(img)
# result
# [[{'score': 0.99995565, 'label': 'sunflower'},
#  {'score': 2.5318595e-05, 'label': 'toy'},
#  {'score': 9.903885e-06, 'label': 'dog'},
#  {'score': 6.75336e-06, 'label': 'tree'},
#  {'score': 2.396818e-06, 'label': 'cat'}]]
```

```python
from mindformers.pipeline import pipeline
from mindformers.tools.image_tools import load_image


pipeline_task = pipeline("image_classification", model='swin_base_p4w7')
img = load_image("https://ascend-repo-modelzoo.obs.cn-east-2."
          "myhuaweicloud.com/XFormer_for_mindspore/clip/sunflower.png")
pipeline_result = pipeline_task(img, top_k=3)
print(pipeline_result)
# 输出
# [[{'score': 0.89573187, 'label': 'daisy'}, {'score': 0.005366202, 'label': 'bee'},
# {'score': 0.0013296203, 'label': 'fly'}]]
```

```python
from mindformers.pipeline import pipeline
pipeline_task = pipeline("translation", model='t5_small')
pipeline_result = pipeline_task("translate the English to Romain: a good boy!", top_k=3)
print(pipeline_result)
#[{'translation_text': ['Romain ist ein guter Junge!']}]
```

Thank you.

把数字世界带入每个人、每个家庭、
每个组织，构建万物互联的智能世界。

Bring digital to every person, home, and organization for a fully connected, intelligent world.

HUAWEI