The Llama 3.1 Herd of Models

Paper Reading session

week 1

Cs Route

Basics and Building Blocks of llama

philosophiocal

week 2

Bench marks,Training recipes and Impact on LLM space

This session Hopefully answers these question from our POV

- llama is another Decoder only Model (ever wondered why companies use decoder only models but they exist encoder and decoder based models ?)
- What did they do new to talk about them except for the fact of being open source
- What are interesting finds on this Models except data collecting, data mining (more on collection of data on next session) ?
- What are the guidelines of training a LLMs in this era efficiently and parallely

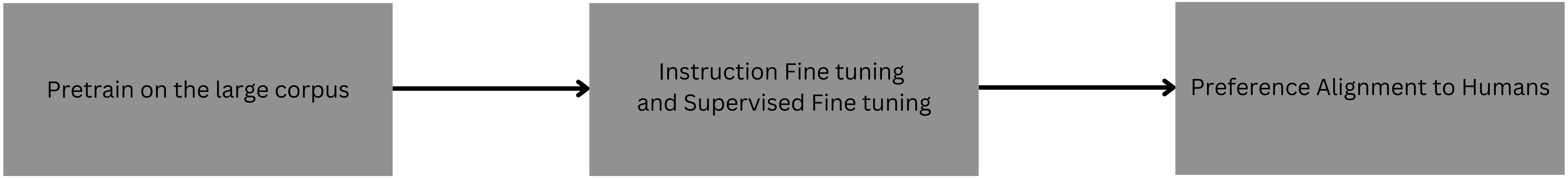
Bit into transformers decoder only (skip if people are familiar)

Lets answer the first question of this session ?

- Why decoder only models used in Multimodal (leave llama) in general ?
- why not encoder decoder models ? doesnt make sense right ? or does it arguable convos be encoder decoder models such as T5

explanation in Whiteboard

General Pipe Line of LLM



Model architecture details on the paper

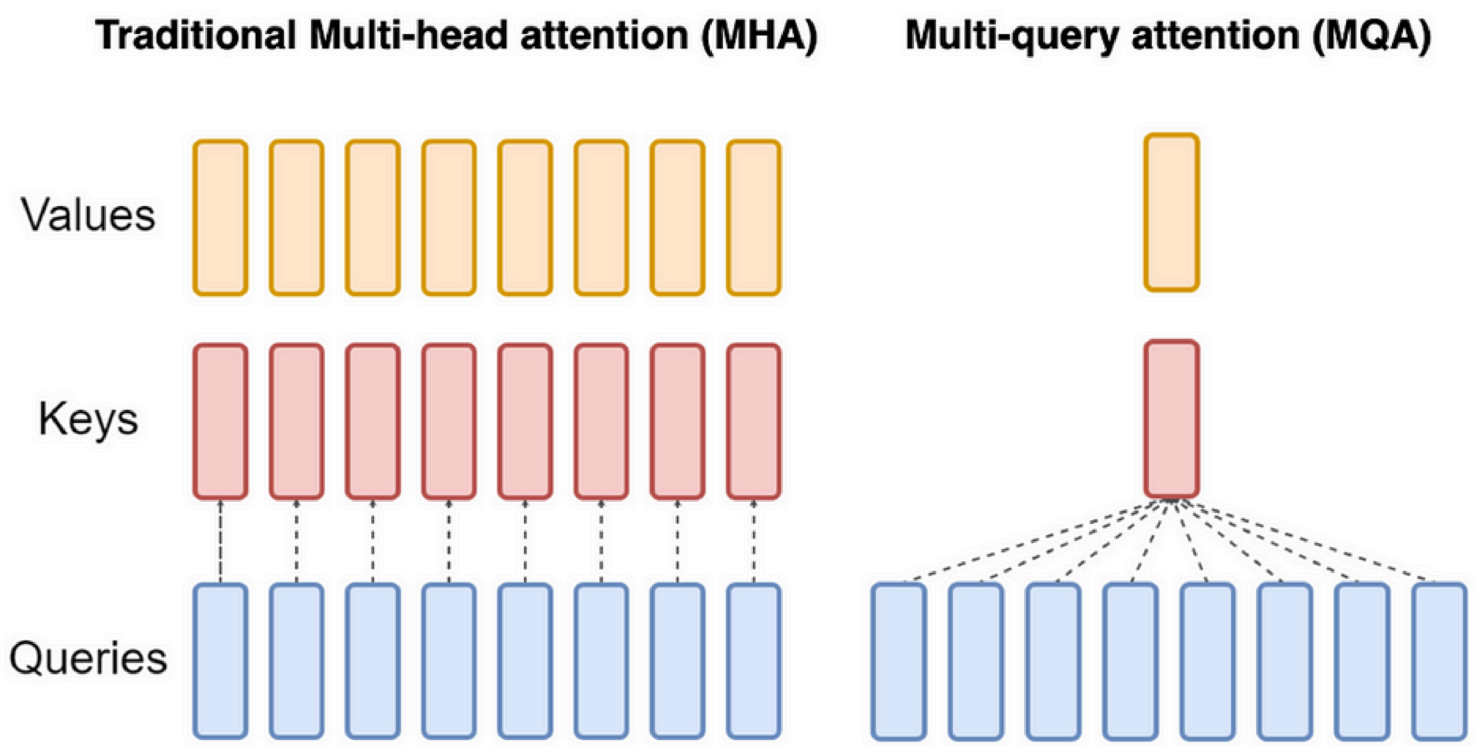
	8B	70B	405B
Layers	32	80	126
Model Dimension	4,096	8192	16,384
FFN Dimension	14,336	28,672	53,248
Attention Heads	32	64	128
Key/Value Heads	8	8	8
Peak Learning Rate	3×10^{-4}	1.5×10^{-4}	8×10^{-5}
Activation Function	SwiGLU		
Vocabulary Size	128,000		
Positional Embeddings	RoPE ($\theta = 500,000$)		

1.?

2.?

1) Grouped Query attention

- Grouped query attention is used in llama where we separate the head dimension from Batch, num_heads,seq,Dim into Batch,num_heads// num_key_value, num_key_value,dim
- Perform attention operation as it is on the newly 5 dimensional data



Pesudo code of GQA with causal Mask

```
from einops import rearrange,reduce,repeat,einsum
def gqa(query,key,value,group,num_head,dropout=0.):

  scale=query.size(-1)**0.5

  query=query/scale

  b,hq,t,d=query.shape
  b,hk,s,d=key.shape

  num_head_groups= num_head//group


  query=rearrange(query,"b (h g) n d -> b g h n d",g=num_head_groups)

  attn=einsum(query, key, "b g h n d,b h s d -> b g h n s")

  mask=torch.ones((b,t,s),device=query.device,dtype=torch.bool).tril_()

  if mask is not None:
    if mask.ndim == 2:
        mask = rearrange(mask, "b s -> b () () () s")
    elif mask.ndim == 3:
        mask = rearrange(mask, "b n s -> b () () n s")

  attn.masked_fill_(~mask,torch.finfo(attn.dtype).min)
  attn=F.softmax(attn,dim=-1)

  attn=F.dropout(attn,p=dropout)

  attn=einsum(attn,value,"b g h t s, b h s d -> b g h t d")
  attn=rearrange(attn,"b g h t d -> b (g h) t d")

  return attn
```

Absolute position embedding to relative

In standard transformer

$$Q = W_Q(x_Q + PE)$$

$$K = W_K(x_K + PE)$$

$$Q^T K = (x_Q^T W_Q^T + PE_Q^T W_Q^T)(W_K x_K + W_K PE_K)$$

$$= x^T W_Q^T W_K x_K + x_Q^T W_Q^T W_K \underline{PE_K} + \underline{PE_Q^T} \overbrace{W_Q^T}^{U^T} W_K x_K$$

$$+ PE_Q^T \underbrace{W_Q^T}_{V^T} W_K \underline{PE_K} \rightarrow \text{Relative} \quad \text{—(1)}$$

U^T, V^T make it learnable thats how Inception on Relative Position Embedding

2) Relative position encoding Starting

- Relative position embedding where we would add position vectors on each key and query value this converts the equation into 2set of different attention equation which are needed to be added.
- K is a hyper parameter where it clips the distance matrix between query and key/ value
- more on that in the next slides

Pesudo Code

```
class Relative_position_embedding(nn.Module):

  def __init__(self,head_dim,k,device):
    super().__init__()
    self.head_dim=head_dim
    self.k=k

    self.device=device
    self.position=nn.EmbeddingLayer(self.k*2+1,self.head_dim)

  def forward(self,q_len,k_len):
    vec_q=torch.arange(q_len)
    vec_k=torch.arange(k_len)
    distance_mat= vec_k[None,:] - vec_q[:,None]
    clipped_distance=torch.clamp(distance_mat,-self.k,self.k)
    final_mat= clipped_distance + self.k
    final_mat = torch.LongTensor(final_mat).to(self.device)
    embeddings=self.position[final_mat].to(self.device)

    return embeddings
```

Pesudo code on attention

```
"""Relational Positional embedding"""
if self.relative_pos:
  len_q=query.shape[2]
  len_k=key.shape[2]
  len_v=value.shape[2]

  relative_positionk=self.relative_position_k(len_q,len_k)
  relative_positionv=self.relative_position_v(len_q,len_v)



if self.relative_pos:
  r_q=rearrange(tensor=query,pattern='B N T D -> T (B N) D')

  attn2=(r_q@relative_positionk.transpose(1,2)).transpose(0,1)

  attn2=rearrange(tensor=attn2,pattern="(B N) Q K -> B N Q K",N=self.num_heads)

  mask=torch.ones((len_q,len_k),device=query.device,dtype=torch.bool).tril_() #casual Mask

  attn2=attn2.masked_fill(mask[:len_q,:len_k]==0,float('-inf')) #if causal

  scale=(self.dim//self.num_heads)**-0.5
  attn2=attn2*scale
  attn2=F.softmax(attn2,dim=-1)
  attn2=rearrange(tensor=attn2,pattern="B N Q K -> Q (B N) K")
  attn2=(attn2@relative_positionv).transpose(0,1)
  attn2=rearrange(tensor=attn2,pattern="(B N) Q D -> B N Q D",N=self.num_heads)

  attn2=F.dropout(attn2,p=self.attention_dropout)

  attn = attn1 + attn2
```

RoPE (Rotational Position Embedding)

$$
\begin{aligned}
f_q(\boldsymbol{x}_m, m) &= (\boldsymbol{W}_q \boldsymbol{x}_m) e^{im\theta} \\
f_k(\boldsymbol{x}_n, n) &= (\boldsymbol{W}_k \boldsymbol{x}_n) e^{in\theta} \\
g(\boldsymbol{x}_m, \boldsymbol{x}_n, m-n) &= \mathrm{Re}[(\boldsymbol{W}_q \boldsymbol{x}_m)(\boldsymbol{W}_k \boldsymbol{x}_n)^* e^{i(m-n)\theta}]
\end{aligned}
\tag{12}
$$

where $\mathrm{Re}[\cdot]$ is the real part of a complex number and $(\boldsymbol{W}_k \boldsymbol{x}_n)^*$ represents the conjugate complex number of $(\boldsymbol{W}_k \boldsymbol{x}_n)$. $\theta \in \mathbb{R}$ is a preset non-zero constant. We can further write $f_{\{q,k\}}$ in a multiplication matrix:

$$
f_{\{q,k\}}(\boldsymbol{x}_m, m) = \begin{pmatrix} \cos m\theta & -\sin m\theta \\ \sin m\theta & \cos m\theta \end{pmatrix} \begin{pmatrix} W^{(11)}_{\{q,k\}} & W^{(12)}_{\{q,k\}} \\ W^{(21)}_{\{q,k\}} & W^{(22)}_{\{q,k\}} \end{pmatrix} \begin{pmatrix} x^{(1)}_m \\ x^{(2)}_m \end{pmatrix}
\tag{13}
$$

3.2.2 General form

In order to generalize our results in 2D to any $\boldsymbol{x}_i \in \mathbb{R}^d$ where d is even, we divide the d-dimension space into $d/2$ sub-spaces and combine them in the merit of the linearity of the inner product, turning $f_{\{q,k\}}$ into:

$$f_{\{q,k\}}(\boldsymbol{x}_m, m) = \boldsymbol{R}^d_{\Theta,m} \boldsymbol{W}_{\{q,k\}} \boldsymbol{x}_m \tag{14}$$

where

$$\boldsymbol{R}^d_{\Theta,m} = \begin{pmatrix} \cos m\theta_1 & -\sin m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ \sin m\theta_1 & \cos m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cos m\theta_2 & -\sin m\theta_2 & \cdots & 0 & 0 \\ 0 & 0 & \sin m\theta_2 & \cos m\theta_2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \cos m\theta_{d/2} & -\sin m\theta_{d/2} \\ 0 & 0 & 0 & 0 & \cdots & \sin m\theta_{d/2} & \cos m\theta_{d/2} \end{pmatrix} \tag{15}$$

is the rotary matrix with pre-defined parameters $\Theta = \{\theta_i = 10000^{-2(i-1)/d}, i \in [1, 2, ..., d/2]\}$. A graphic illustration of RoPE is shown in Figure (1). Applying our RoPE to self-attention in Equation (2), we obtain:

$$\boldsymbol{q}^\intercal_m \boldsymbol{k}_n = (\boldsymbol{R}^d_{\Theta,m} \boldsymbol{W}_q \boldsymbol{x}_m)^\intercal (\boldsymbol{R}^d_{\Theta,n} \boldsymbol{W}_k \boldsymbol{x}_n) = \boldsymbol{x}^\intercal \boldsymbol{W}_q R^d_{\Theta,n-m} \boldsymbol{W}_k \boldsymbol{x}_n \tag{16}$$

3.4.2 Computational efficient realization of rotary matrix multiplication

Taking the advantage of the sparsity of $\boldsymbol{R}^d_{\Theta,m}$ in Equation (15), a more computational efficient realization of a multiplication of R^d_Θ and $\boldsymbol{x} \in \mathbb{R}^d$ is:

$$\boldsymbol{R}^d_{\Theta,m}\boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \vdots \\ x_{d-1} \\ x_d \end{pmatrix} \otimes \begin{pmatrix} \cos m\theta_1 \\ \cos m\theta_1 \\ \cos m\theta_2 \\ \cos m\theta_2 \\ \vdots \\ \cos m\theta_{d/2} \\ \cos m\theta_{d/2} \end{pmatrix} + \begin{pmatrix} -x_2 \\ x_1 \\ -x_4 \\ x_3 \\ \vdots \\ -x_d \\ x_{d-1} \end{pmatrix} \otimes \begin{pmatrix} \sin m\theta_1 \\ \sin m\theta_1 \\ \sin m\theta_2 \\ \sin m\theta_2 \\ \vdots \\ \sin m\theta_{d/2} \\ \sin m\theta_{d/2} \end{pmatrix} \tag{34}$$

Model architecture details on the paper

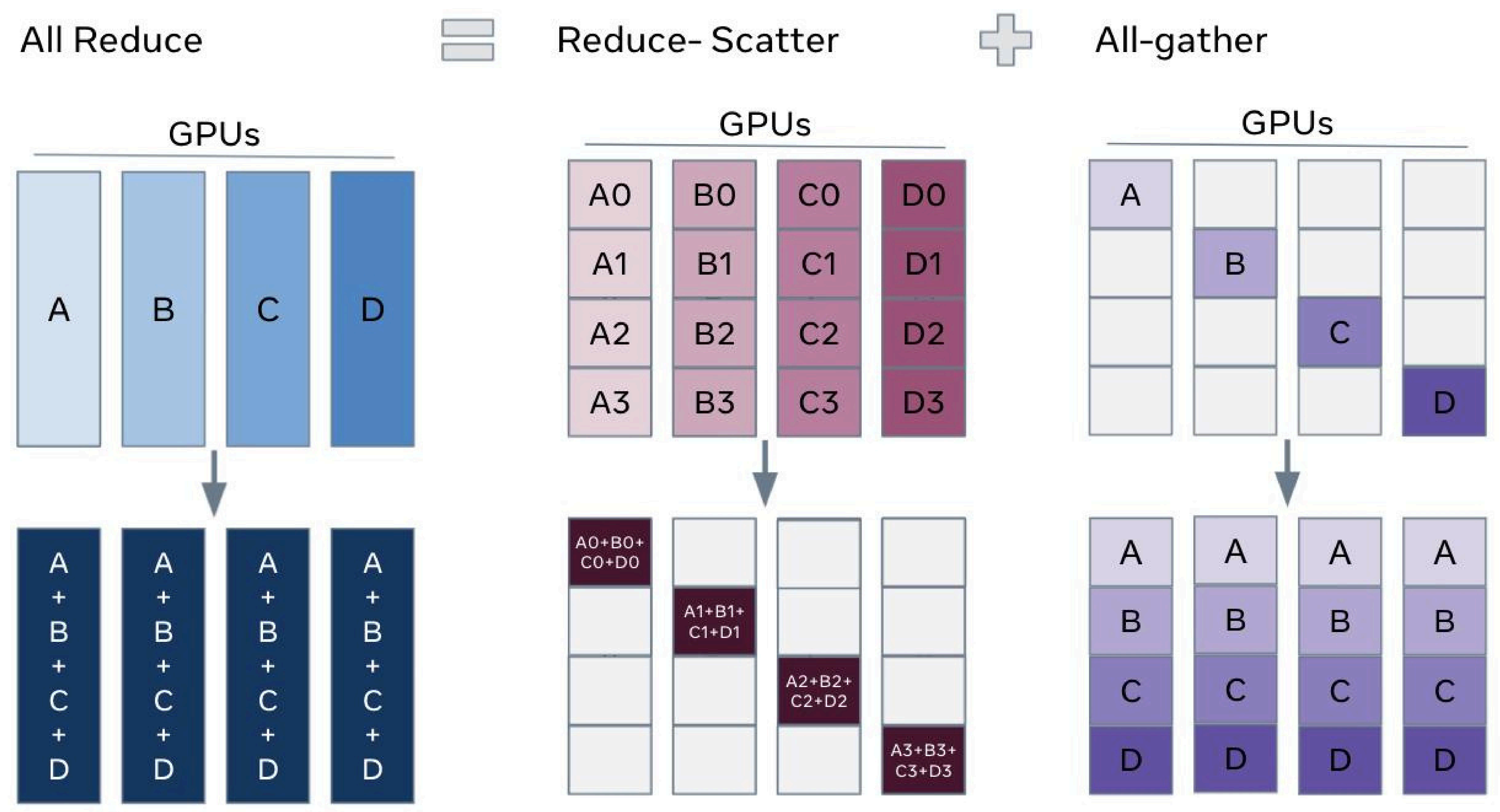
	8B	70B	405B
Layers	32	80	126
Model Dimension	4,096	8192	16,384
FFN Dimension	14,336	28,672	53,248
Attention Heads	32	64	128
Key/Value Heads	8	8	8
Peak Learning Rate	3×10^{-4}	1.5×10^{-4}	8×10^{-5}
Activation Function		SwiGLU	
Vocabulary Size		128,000	
Positional Embeddings		RoPE ($\theta = 500,000$)	

!

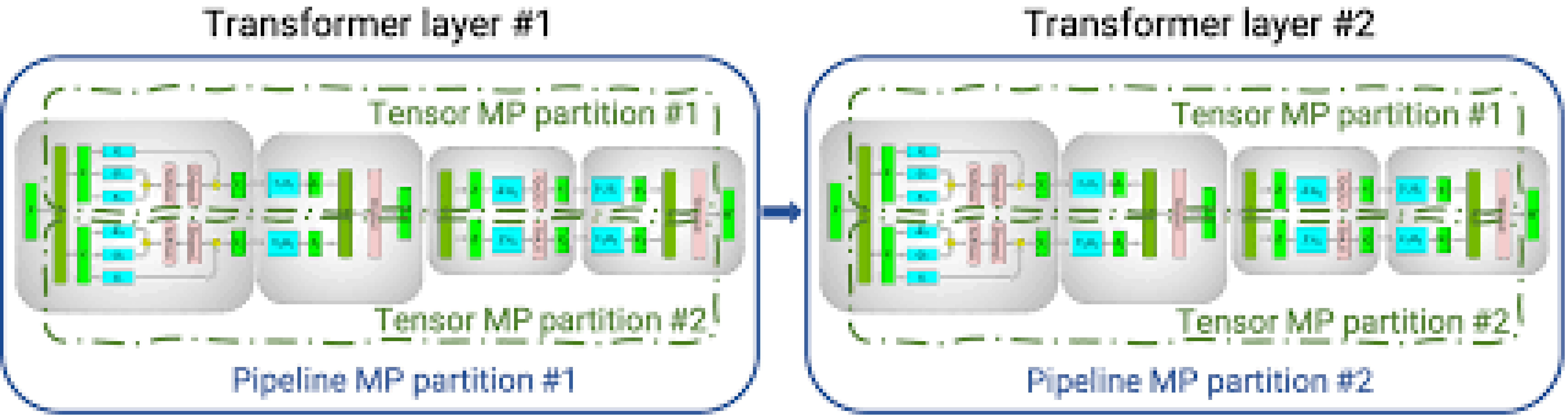
!

Parallelism

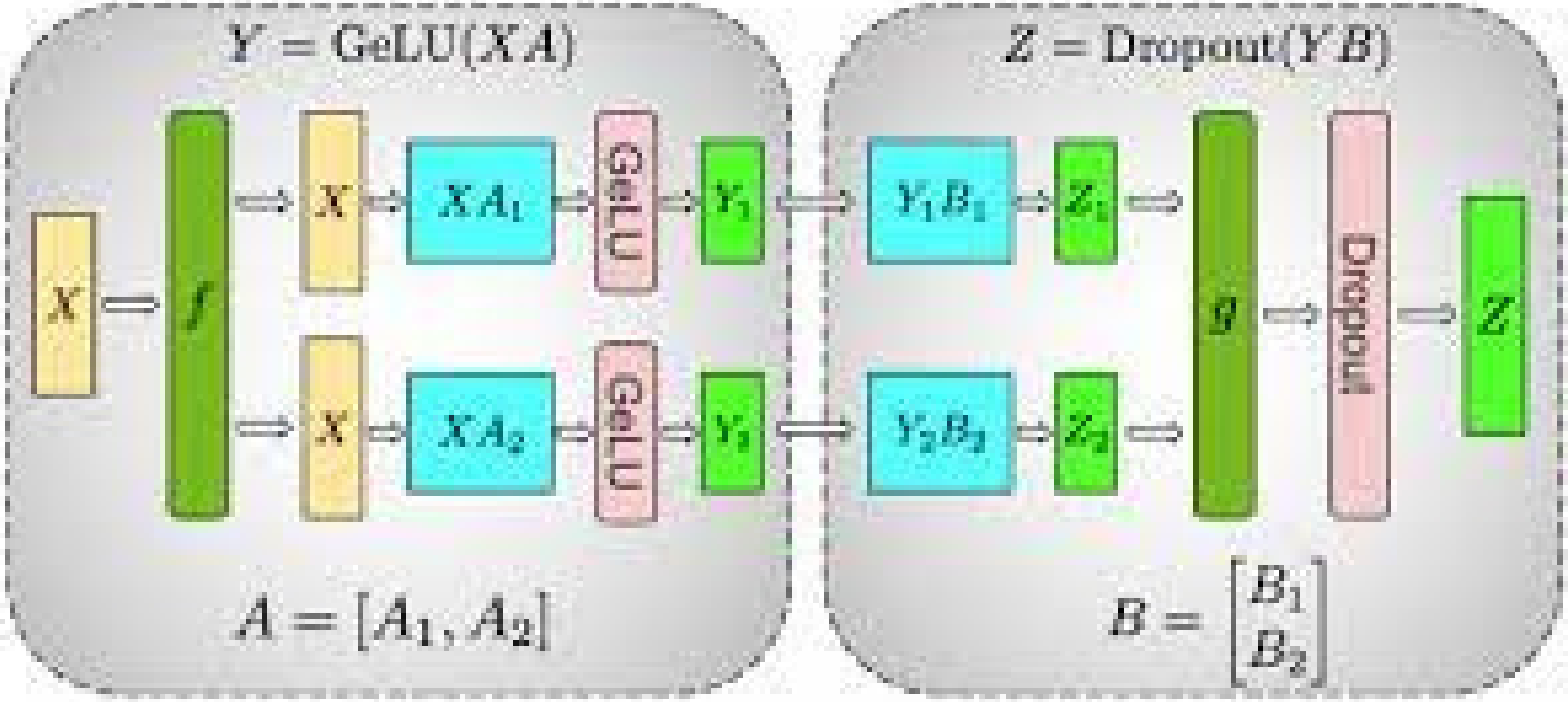
operations on distributed envirornment



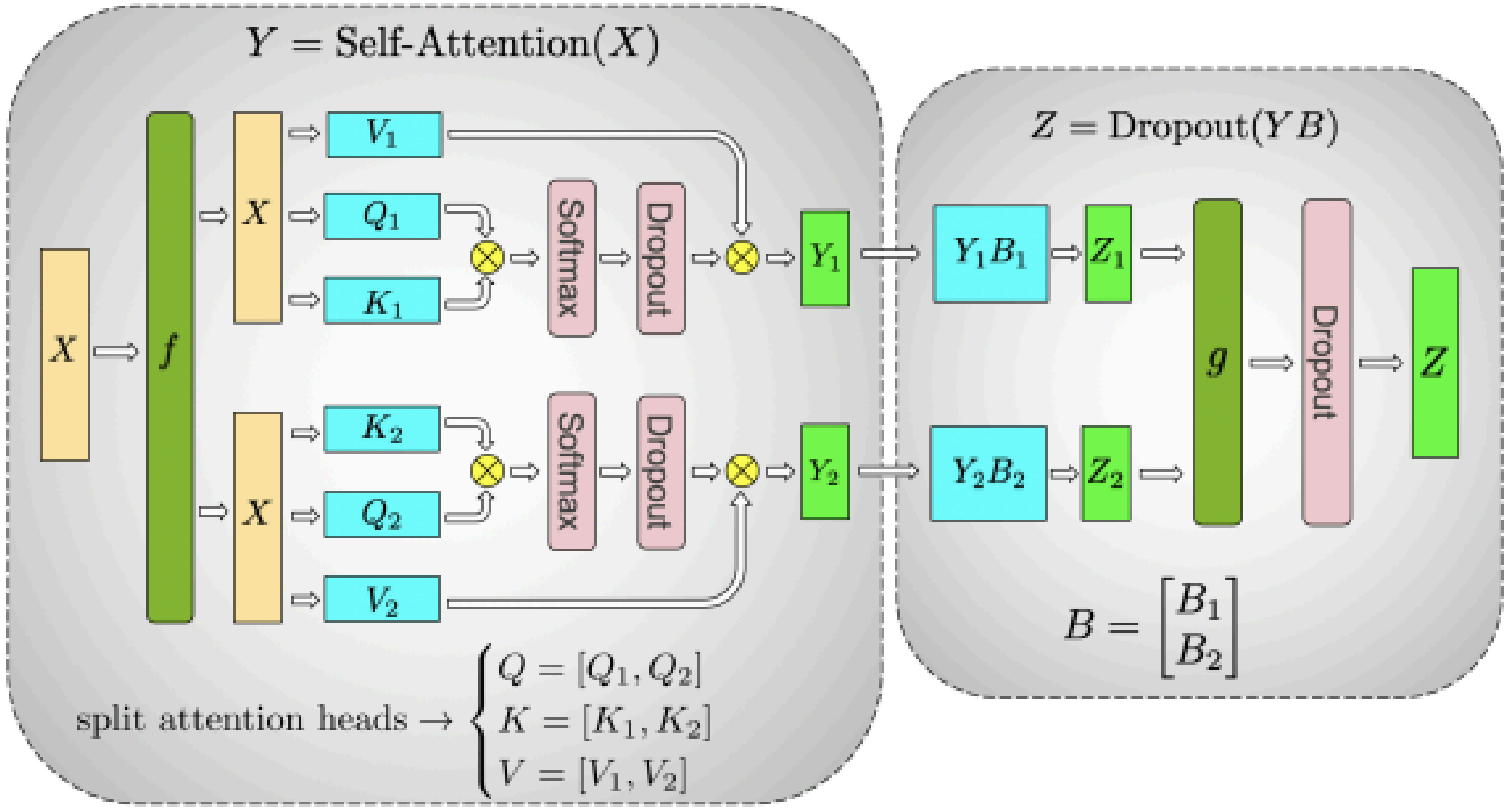
Data parallel



Tensor Parallel

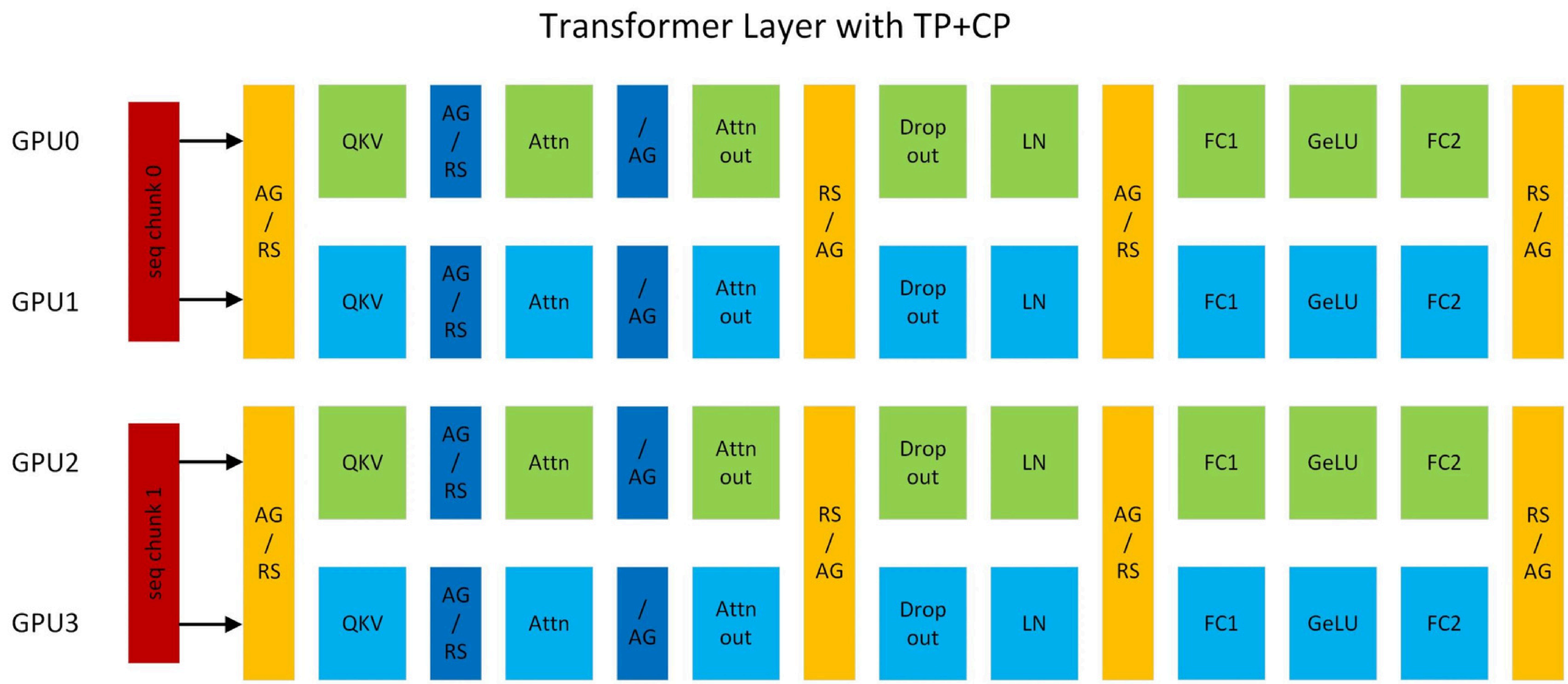


(a) MLP

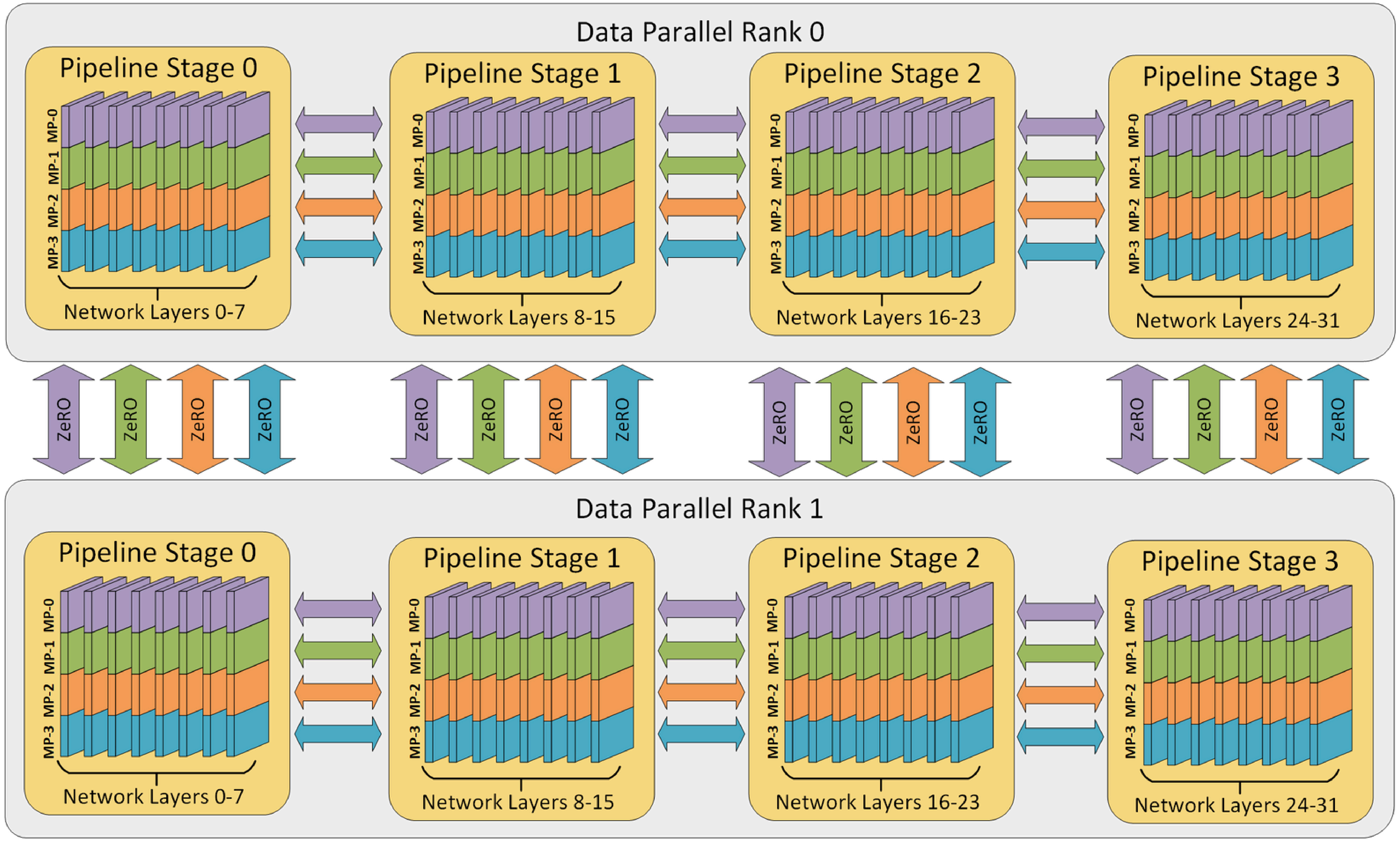


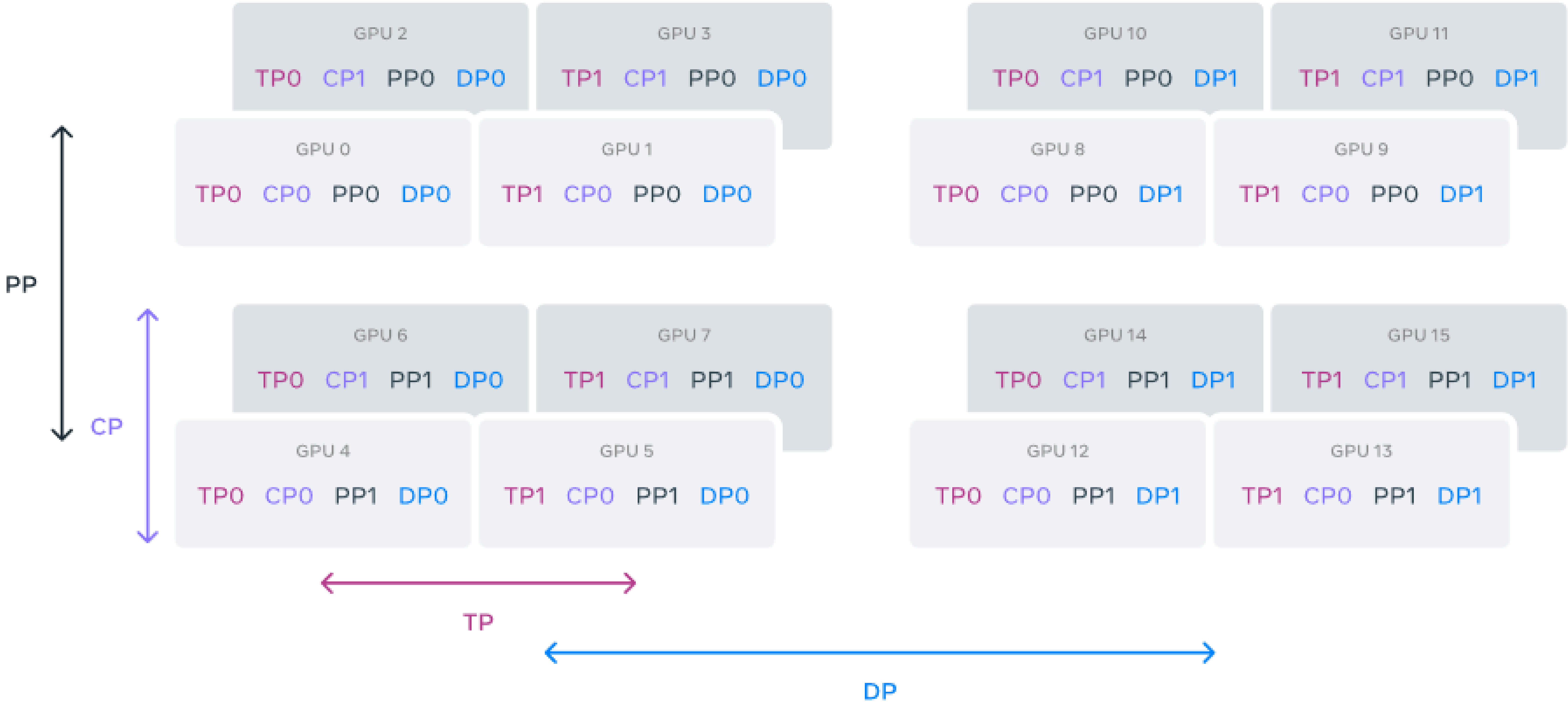
(b) Self-Attention

Context parallelism



Pipeline parallelism





Now on Meta’s Lamma paper

Any doubts ?

Feel free to ping me on discord @maddy

or

Linkedin - Madhava Prasath