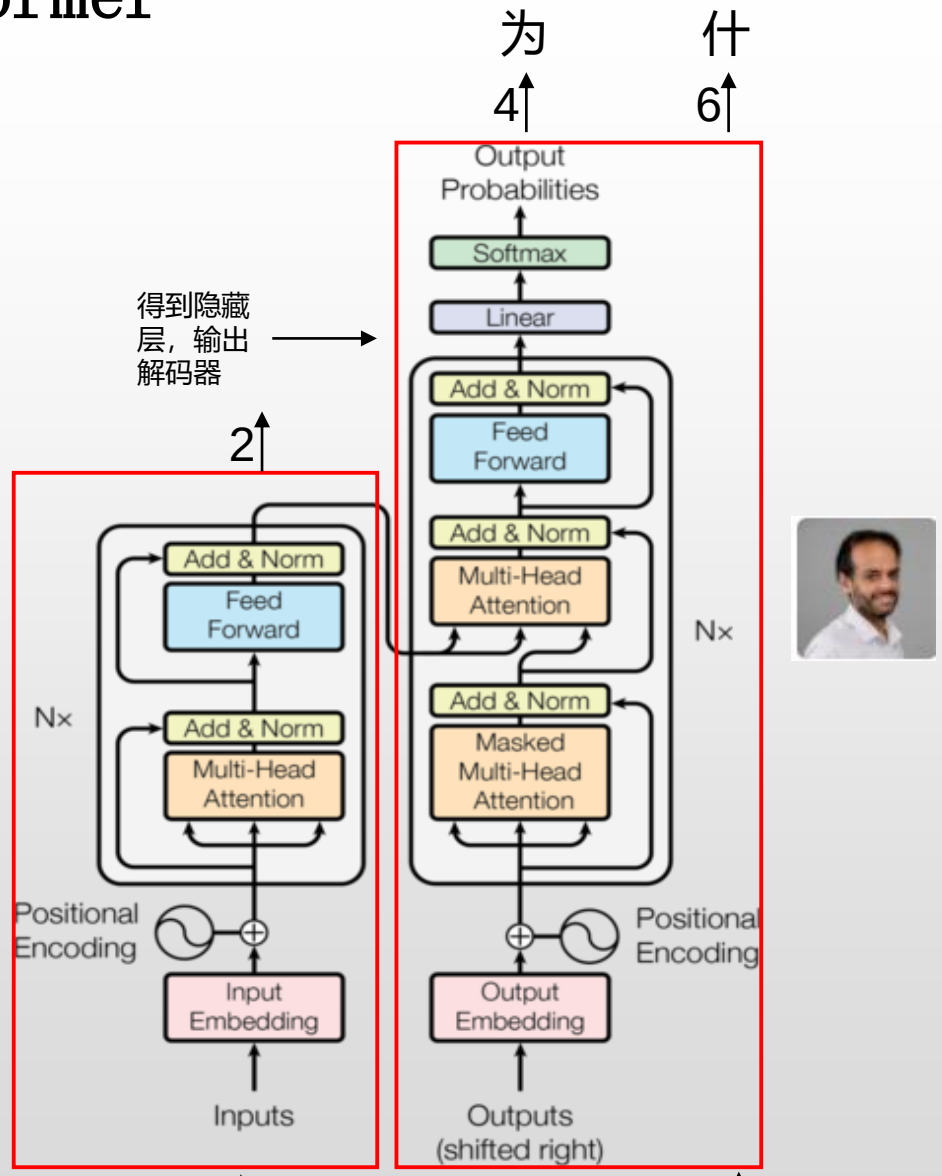
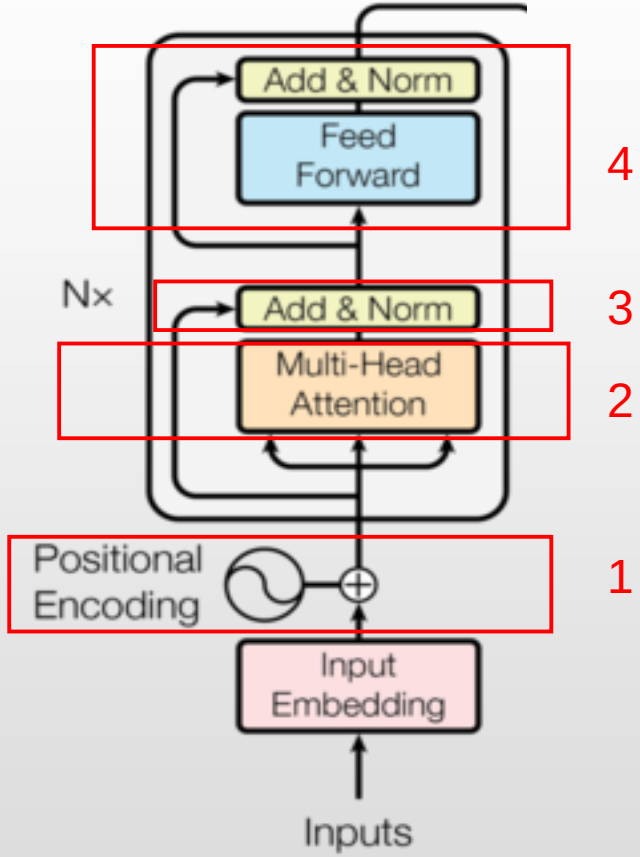
MindSpore

大规模分布式训练策略介绍

Transformer





http://jalammar.github.io/illustrated-transformer/

Transformer

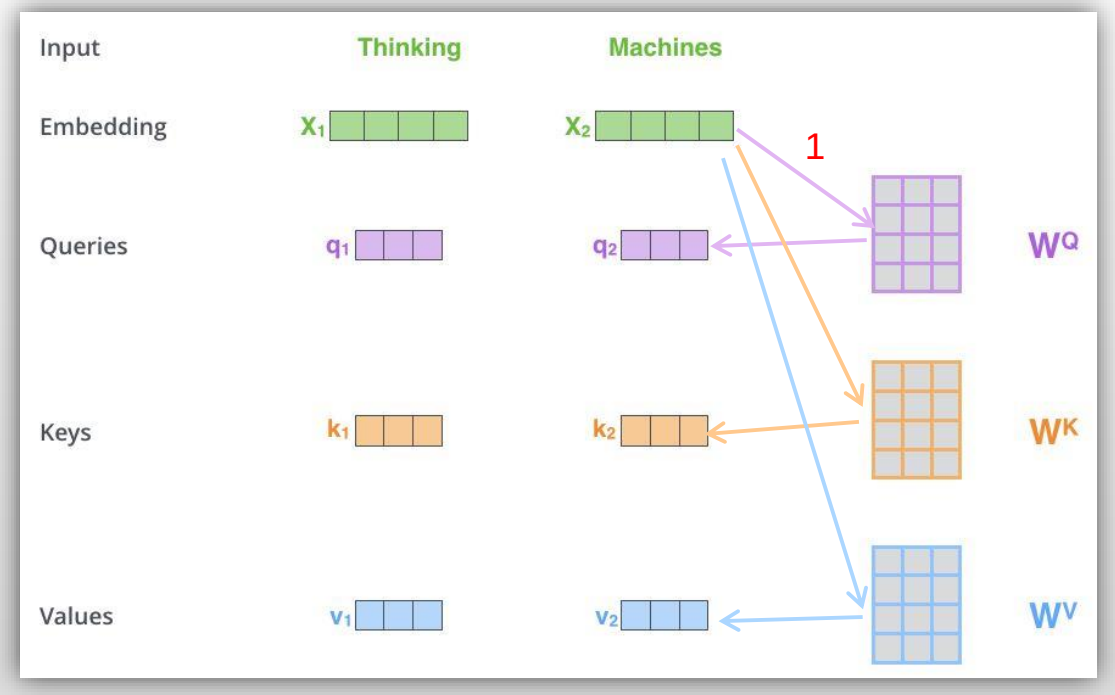
“我们应该从苦涩的教训中学到一点：通用方法非常强大，这类方法会随着算力的增加而继续扩展，搜索和学习似乎正是这样的方法”

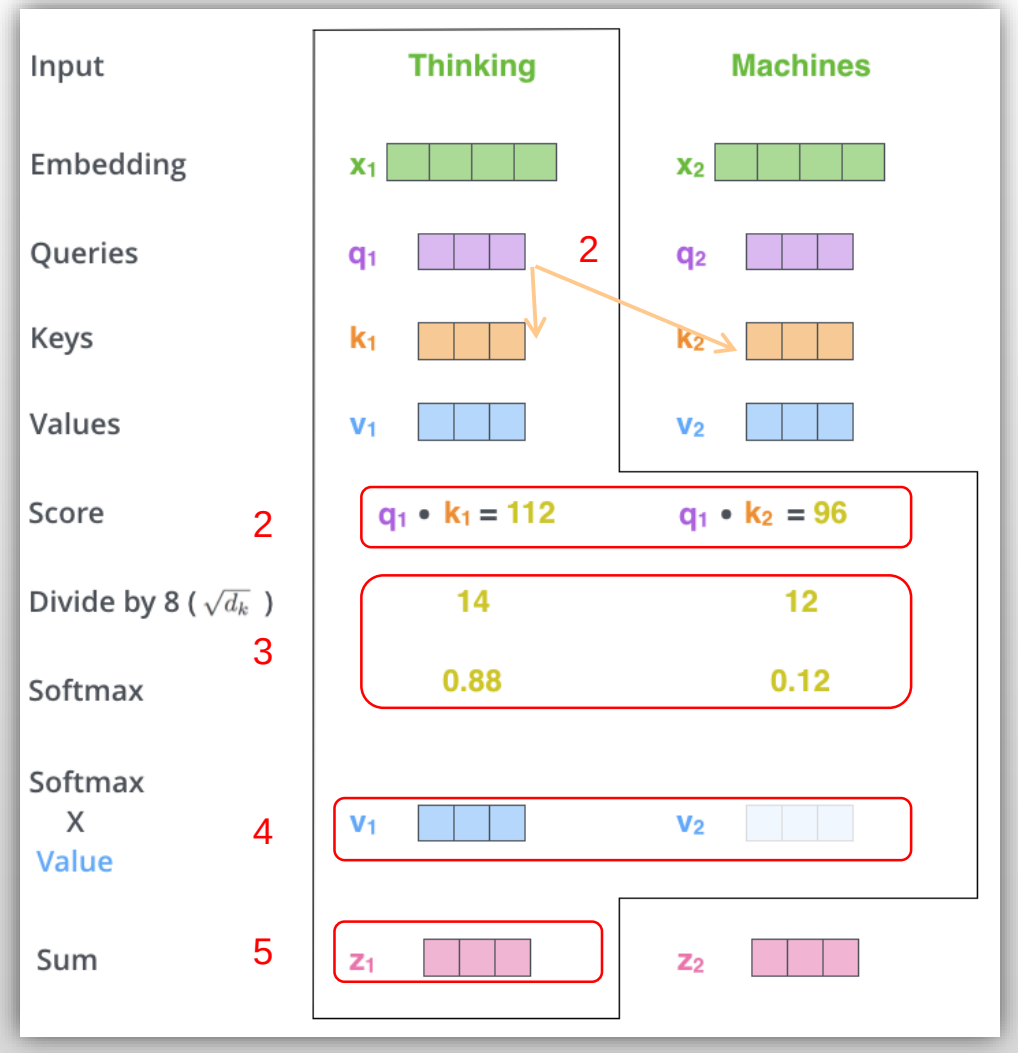
----Richard Sutton

A lot of the subsequent research work saw the architecture shed either the encoder or decoder, and use just one stack of transformer blocks – stacking them up as high as practically possible, feeding them massive amounts of training text, and throwing vast amounts of compute at them (hundreds of thousands of dollars to train some of these language models, likely millions in the case of AlphaStar).

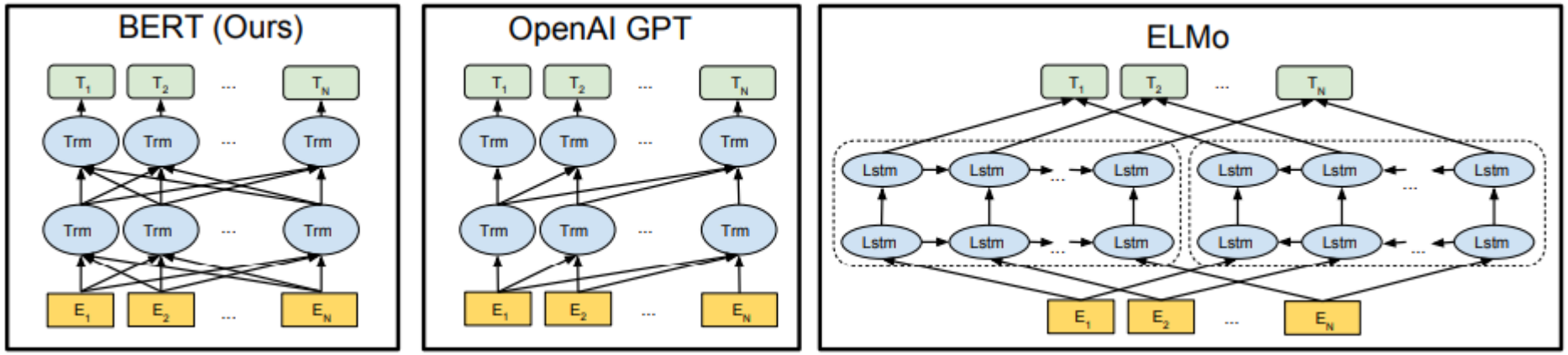
——Jay Alammar

Attention机制





盘古 α



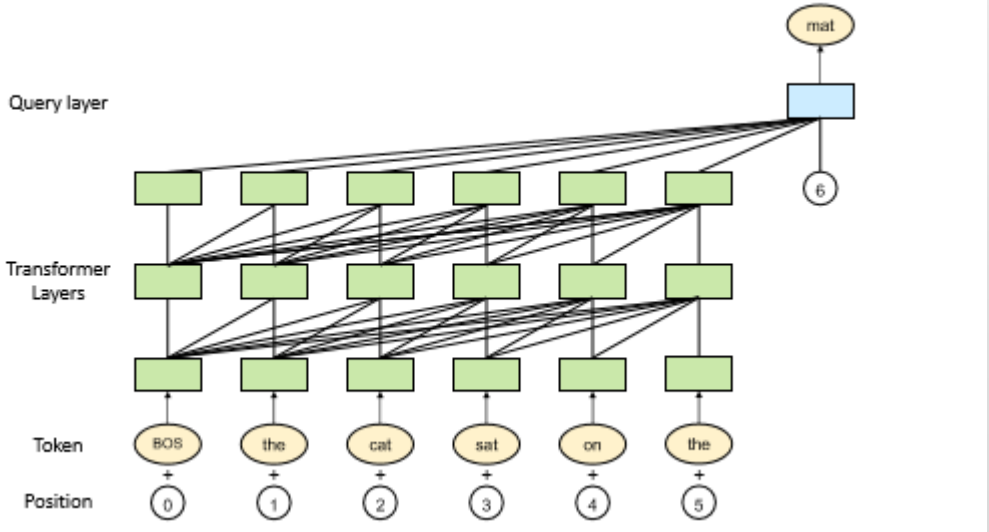


Table 1: Model sizes and hyperparameters of PanGu-α models.

Model	#Parameters	#Layers (L)	Hidden size (d)	FFN size (d_{ff})	#Heads (N_h)
PanGu-α 2.6B	2.6B	32	2560	10240	40
PanGu-α 13B	13.1B	40	5120	20480	40
PanGu-α 200B	207.0B	64	16384	65536	128

思考

1. self-attention计算量
   o(n^2)　　attn = Softmax(QK^T)V

2. 位置编码
   绝对位置编码
   相对位置编码

3. Pre/Post-LayerNorm

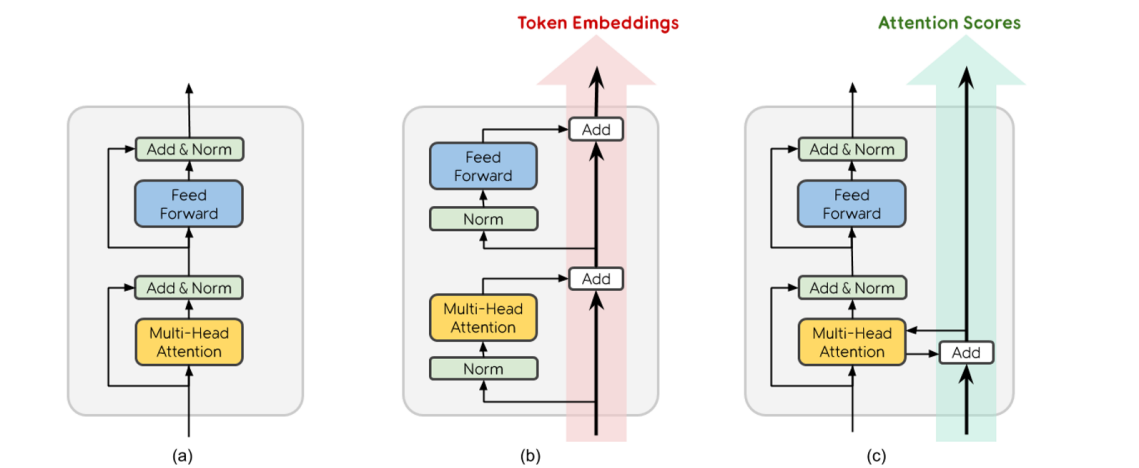
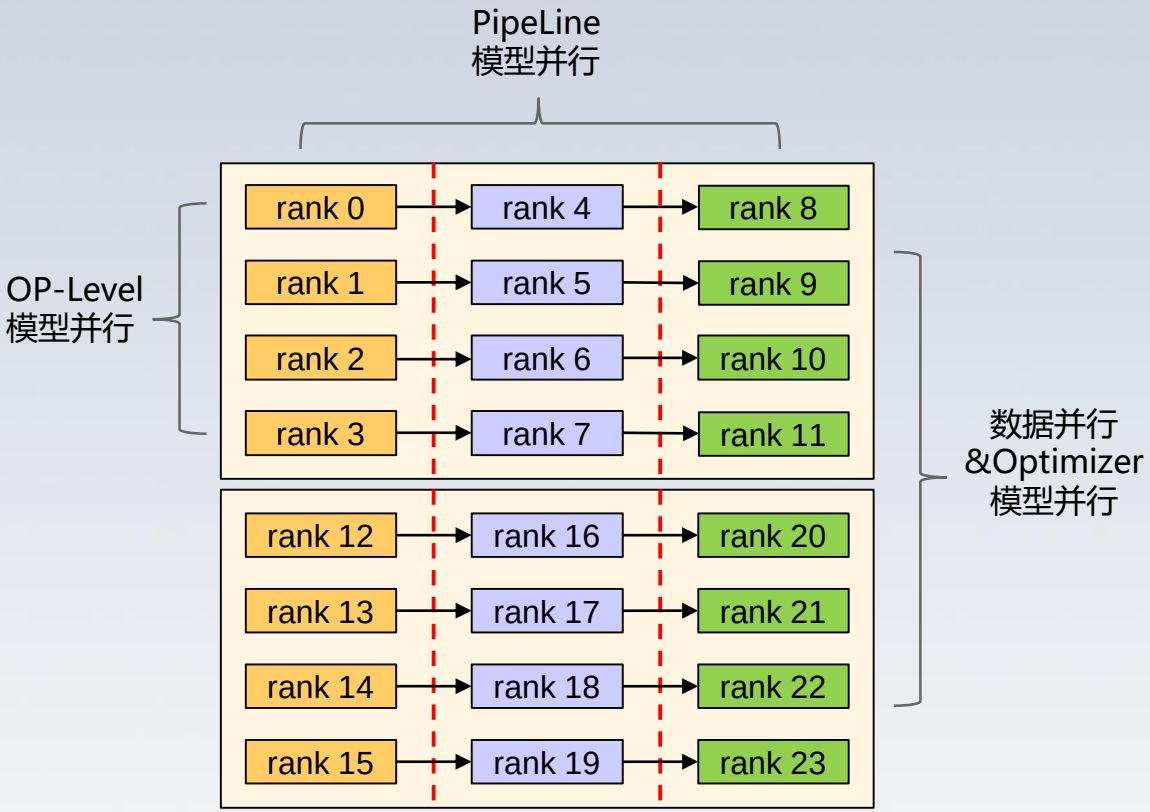


Figure 1: Comparison of different Transformer layers: (a) The prevalent Post-LN layer used by (*e.g.*) BERT; (b) Pre-LN layer used by (*e.g.*) GPT-2 that creates a "direct" path to propagate token embeddings; (c) Our RealFormer layer that creates a "direct" path to propagate attention scores (by adding a simple skip edge on top of (a)).

分布式并行训练



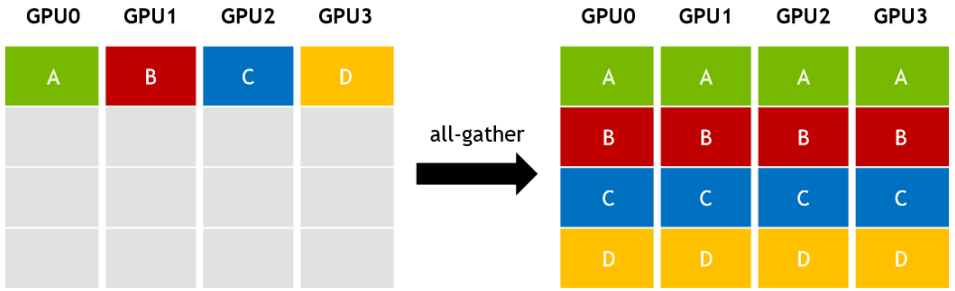
多维度自动并行，最大化计算通信比

- 算子级混合并行、Optimizer模型并行、算子间PipeLine模型并行：多种并行模式融合，实现计算通信比最优；
- 拓扑感知调度：自动根据网络带宽服务内>服务器>机柜间的集群拓扑，把通信量大的调度到服务期内或机柜内，通信量小的调度到机柜间，充分利用网络带宽；
- 重计算Rematerialization：在反向计算过程重计算部分正向子网，降低正向输出缓存；
- Host-Device异构并行：参数梯度和Optimzier State 存储到Host内存，降低卡上内存开销，提升单卡可运行参数规模；
- 高效内存复用：通过Tensor全生命周期的整图内存复用规划，实现接近理论值的内存复用率；

集合通信

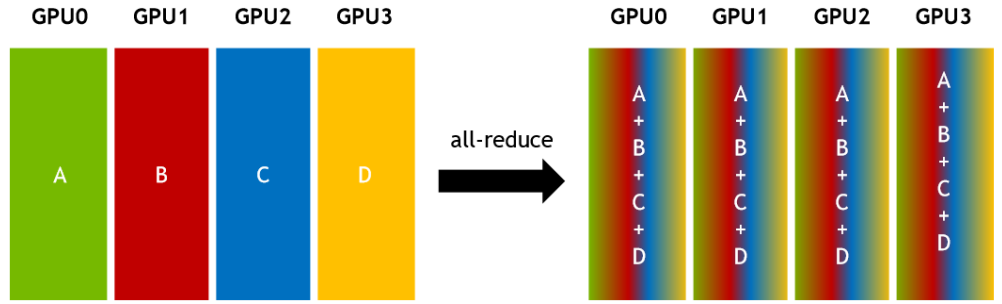
ALL-GATHER

Gather messages from all; deliver gathered data to all participants



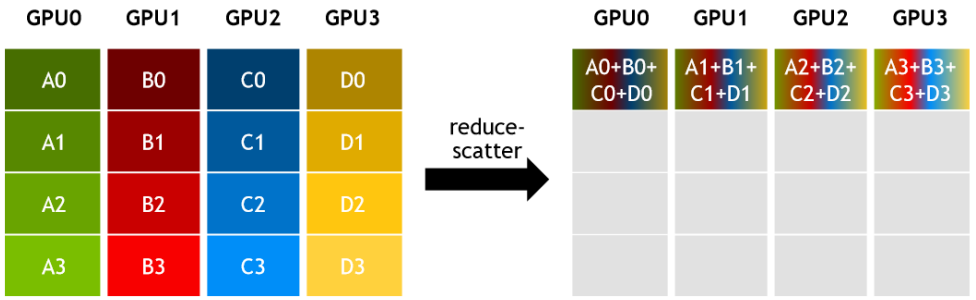
ALL-REDUCE

Combine data from all senders; deliver the result to all participants



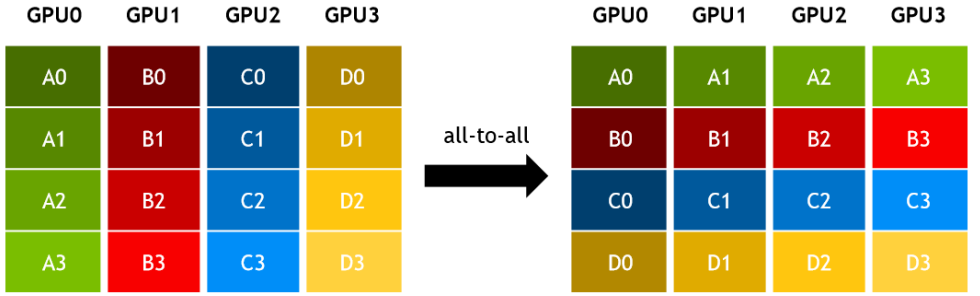
REDUCE-SCATTER

Combine data from all senders; distribute result across participants

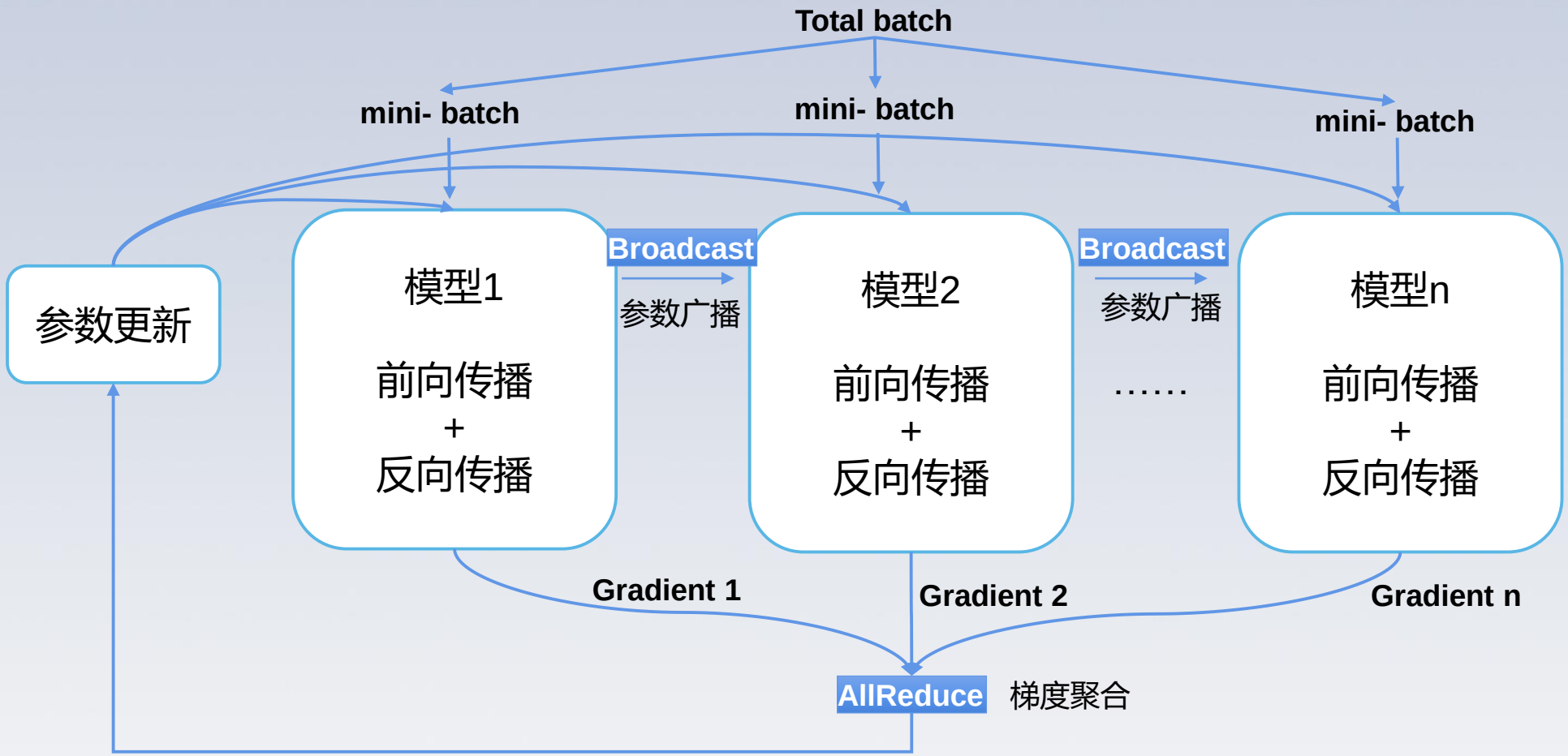


ALL-TO-ALL

Scatter/Gather distinct messages from each participant to every other



数据并行



每卡跑同样的模型，处理不同的样本数据

context.set_auto_parallel_context(parallel_mode=ParallelMode.DATA_PARALLEL)

数据并行转模型并行样例

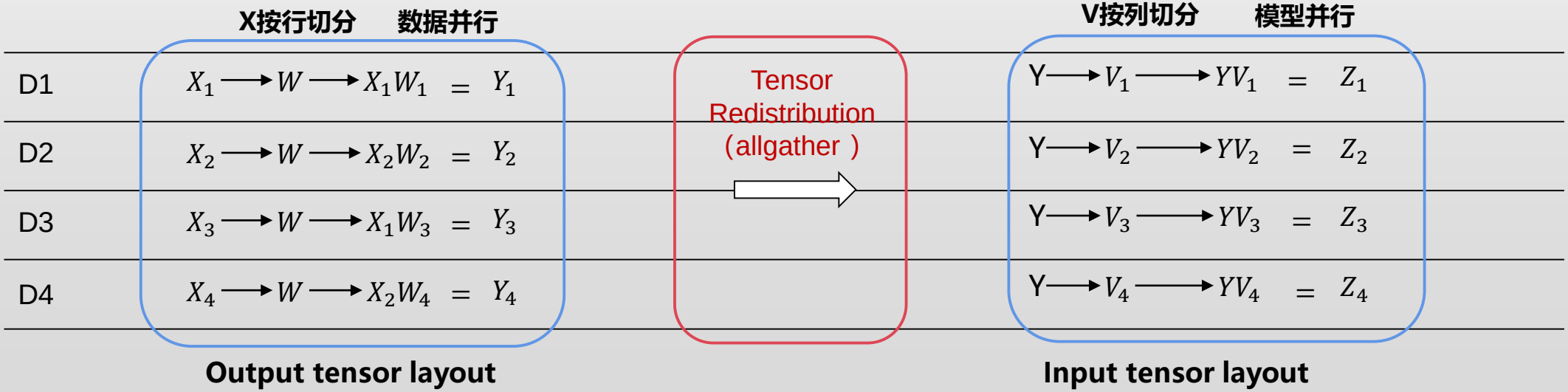
典型场景：ReID

$$Z = (X \times W) \times V$$

1. $Y = \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \end{bmatrix} \times W = \begin{bmatrix} Y_1 \\ Y_2 \\ Y_3 \\ Y_4 \end{bmatrix}$

2. $Z = \mathrm{Y} \times [\mathrm{V}_1 \quad V_2 \quad V_3 \quad V_4] = [Z_1 \quad Z_2 \quad Z_3 \quad Z_4]$

```
class DenseMatMulNet(nn.Cell):
    def __init__(self):
        super(DenseMutMulNet, self).__init__()
        self.matmul1 = ops.MatMul.set_strategy({[4, 1], [1, 1]})
        self.matmul2 = ops.MatMul.set_strategy({[1, 1], [1, 4]})
    def construct(self, x, w, v):
        y = self.matmul1(x, w)
        z = self.matmul2(y, v)
        return s
```



模型并行

A		B		C	
1 2 3 4	@	5 6 7 8	=	19 22 43 50	1

A1 A2		B		C1 C2		C	
1 2	@	5 6 7 8	=	19 22	AllGather	19 22 43 50	
3 4	@	5 6 7 8	=	43 50			2

A		B1 B2		C1 C2		C	
1 2 3 4	@	5 7	=	19 43	AllGather	19 22 43 50	
1 2 3 4	@	6 8	=	22 50			3

A1 A2		B1 B2		C1 C2		C	
1 3	@	5 6	=	5 6 15 18	AllReduce	19 22 43 50	
2 4	@	7 8	=	14 16 20 32			4

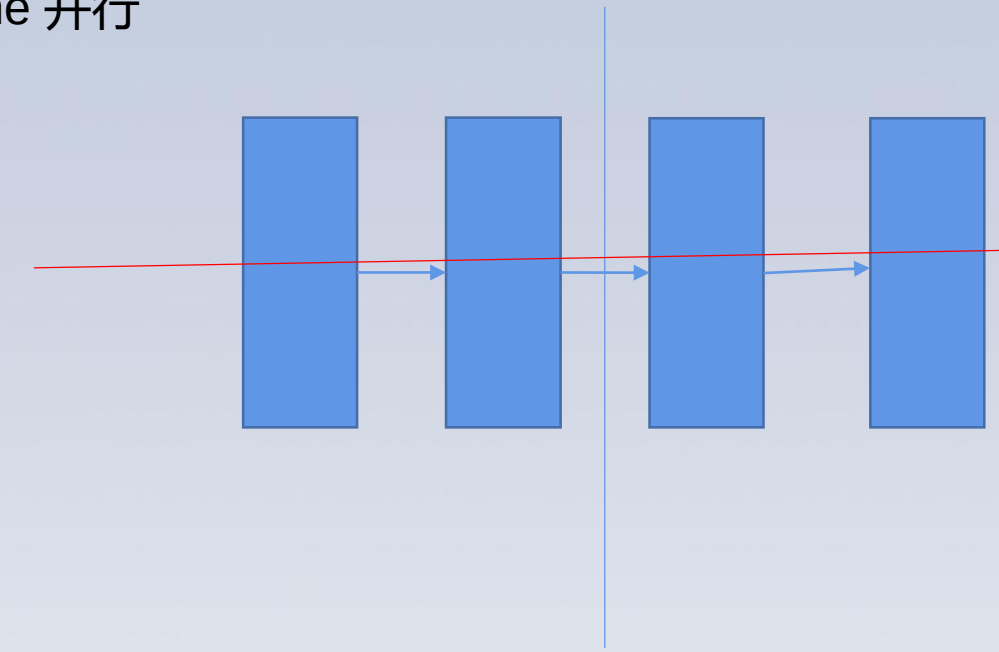
```
1  self.matmul=ops.MatMul()
2  self.matmul=ops.MatMul().shard(((2,1),(1,1)))
3  self.matmul=ops.MatMul().shard(((1,1),(1,2)))
4  self.matmul=ops.MatMul().shard(((1,2),(2,1)))
```

```
c = self.matmul(a,b)
```

```
context.set_auto_parallel_context(parallel_mode=ParallelMode.SEMI_AUTO_PARALLEL)
```

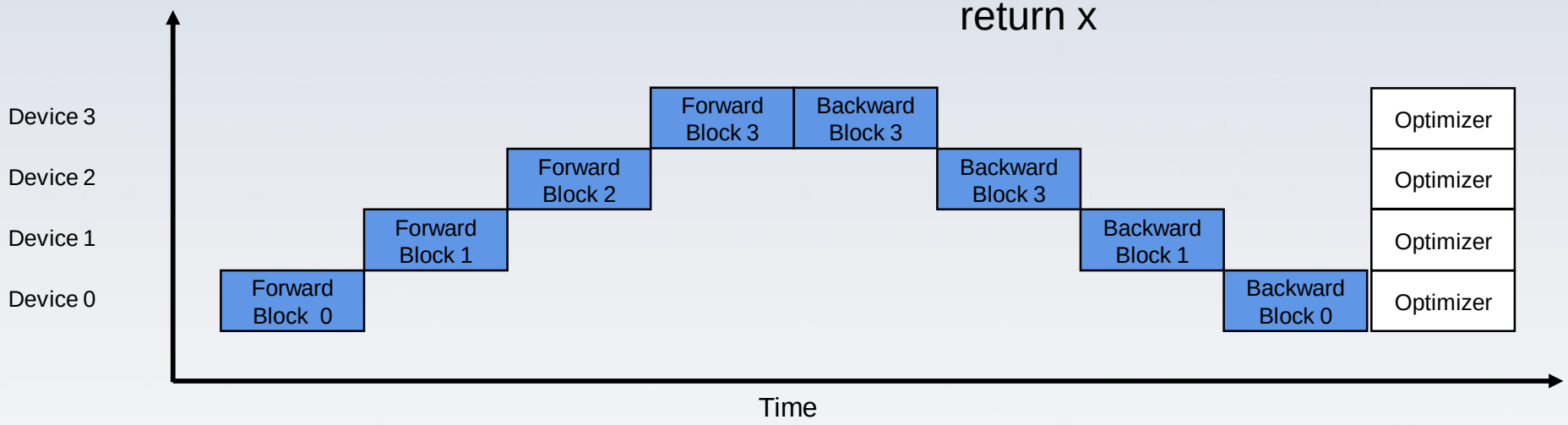
```
context.set_auto_parallel_context(parallel_mode=ParallelMode. AUTO_PARALLEL)
```

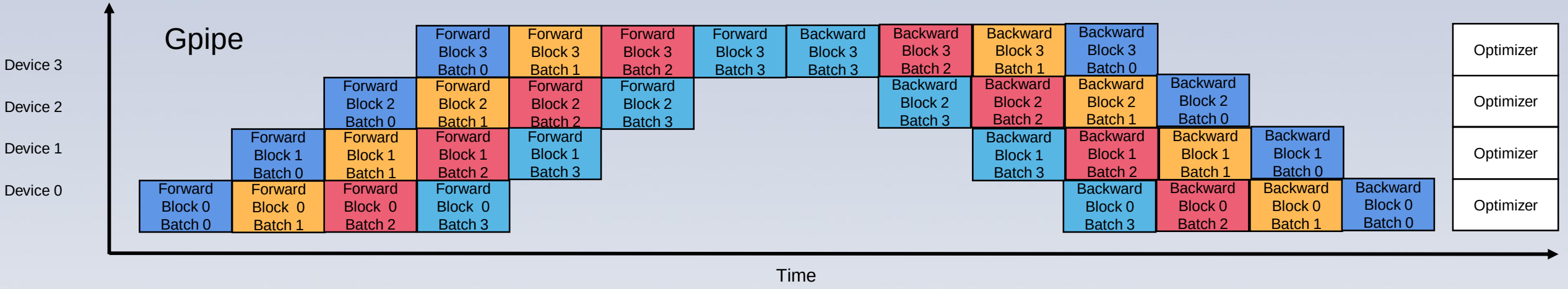
Pipeline 并行



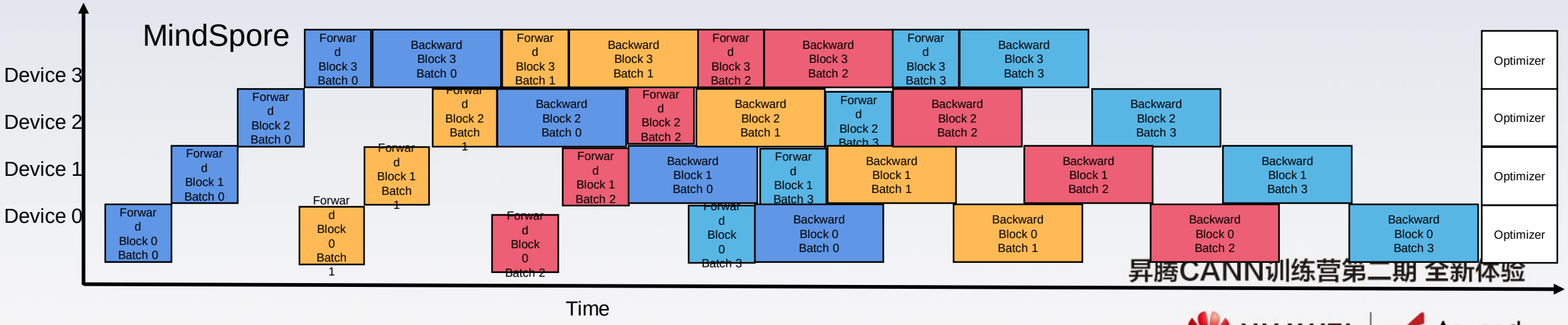
```
context.set_auto_parallel_context(pipeline_stages=2)

class nnn(nn.Cell):
  def __init__(self):
    super(nnn, self).__init__()
    self.subnet1 = subnet().pipeline_stage(0)
    self.subnet2 = subnet().pipeline_stage(1)
  def construct(self,x):
    x = self.subnet1(x)
    x = self.subnet2(x)
    return x
```

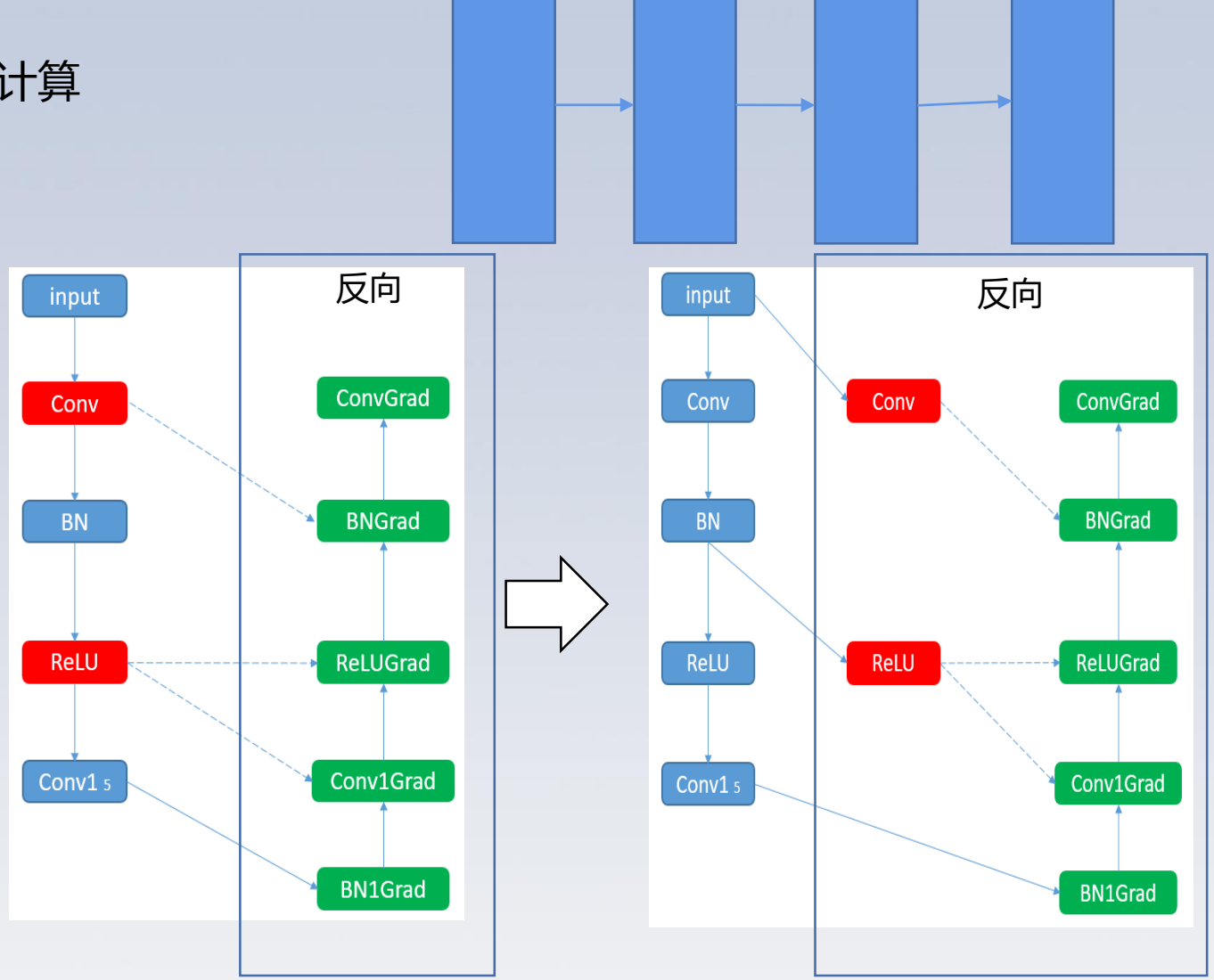




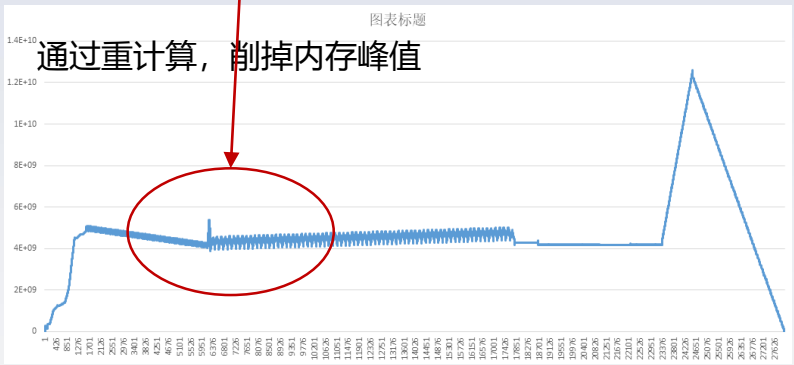
反向提前，降低activation存储时间，stage 0最多存储stage数量个micro batch



重计算



正向activation累积，出现内存峰值



正向activation不存，反向计算时，重新计算正向activation

Algorithm 1 The LAMB optimizer

Data: the number of iterations T; the number of layers L; the learning rate η; (the users only need to input η, default setting: weight decay rate λ=0.01, β_1=0.9, β_2=0.999, ϵ=1e-6)

Result: the trained model w

initialization

$t = 0$

while $t < T$ **do**

****** within each iteration ******

Get a batch of data x_t

$l = 0$

while $l < L$ **do**

****** within each layer ******

$g_t^l = \nabla L(w_{t-1}^l, x_t)$

$m_t^l = \beta_1 m_{t-1}^l + (1-\beta_1) g_t^l$

$v_t^l = \beta_2 v_{t-1}^l + (1-\beta_2) g_t^l \odot g_t^l$

$\hat{m}_t^l = m_t^l / (1-\beta_1^t)$

$\hat{v}_t^l = v_t^l / (1-\beta_2^t)$

$r_1 = \|w_{t-1}^l\|_2$

$r_2 = \| \frac{\hat{m}_t^l}{\sqrt{\hat{v}_t^l + \epsilon}} + \lambda w_{t-1}^l \|_2$

$r = r_1 / r_2$

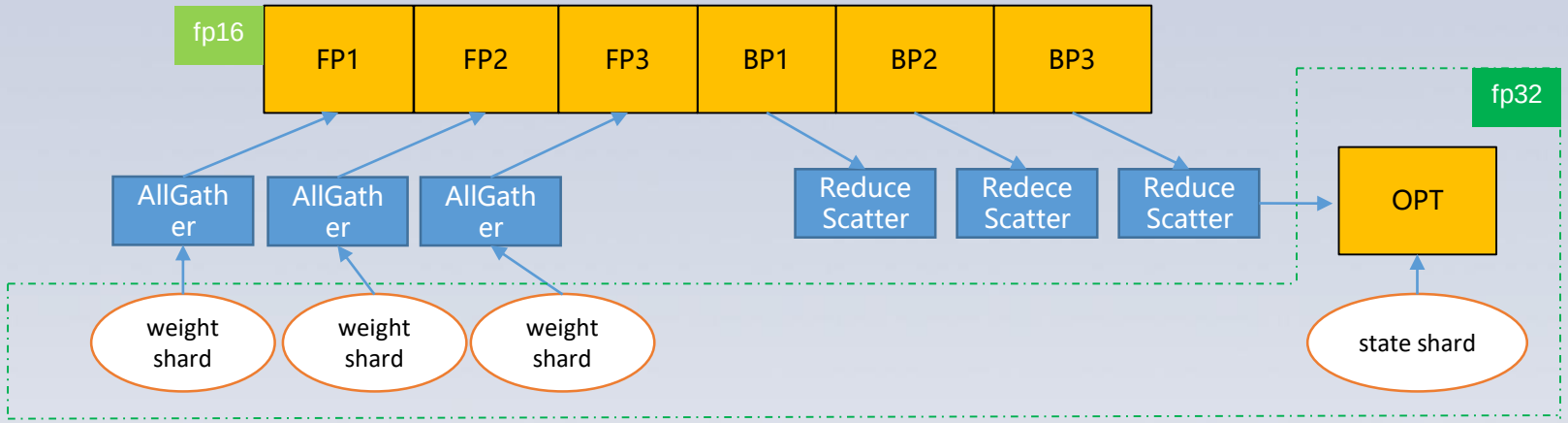
$\eta^l = r \times \eta$

$w_t^l = w_{t-1}^l - \eta^l \times (\frac{\hat{m}_t^l}{\sqrt{\hat{v}_t^l + \epsilon}} + \lambda w_{t-1}^l)$

end

end

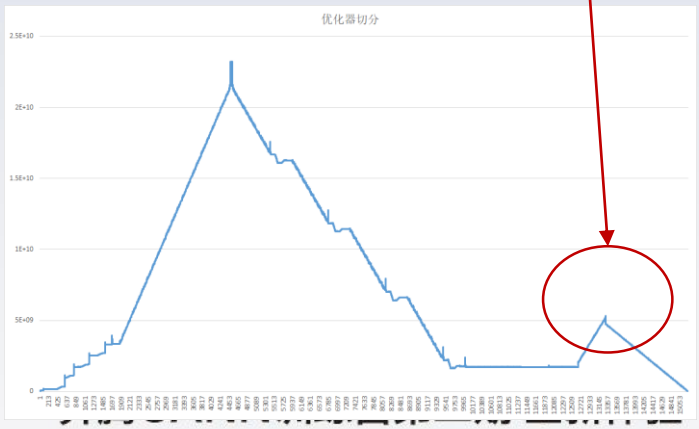
优化器并行 – tensor sharding



内存开销

Optimizer切分前

Optimizer切分后



Feature:

- inner-layer切分：在参数、动量、梯度最高维切分。
- 通信分组并行：allgather和reducecatter分别和正、反向运算并行。
- 混合精度：正反向传播和通信采用fp16运算，优化器及参数采用fp32。

Usage:

使能优化器并行：

context.set_auto_parallel_context(enable_parallel_optimizer=True, parallel_mode=ParallelMode.AUTO_PARALLEL)

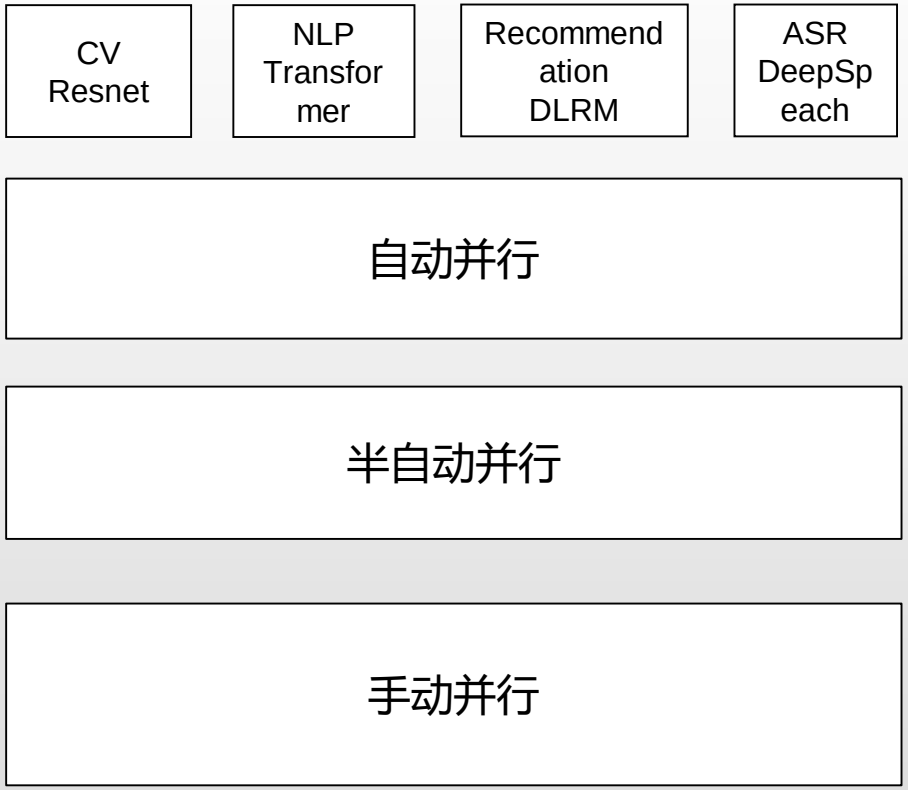
指定分组融合：

```
for i in range(config.num_layers):
        self.blocks.append(Block(config, i+1)).set_comm_fusion((int(i/10))+2)
```

以gpt3为例，每10层transformer融合到一起。

HUAWEI | Ascend

自动并行三层架构



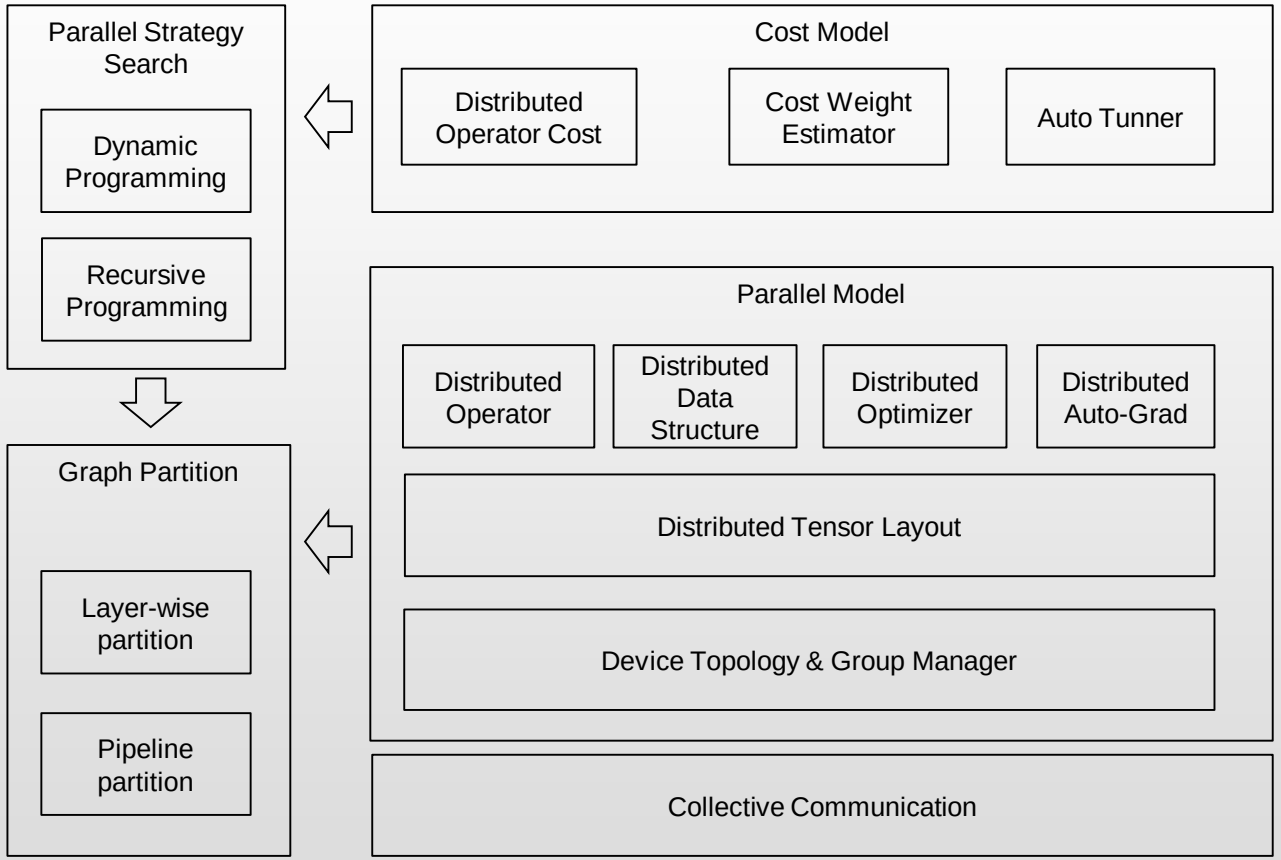
自动并行分三个层实现，提供三层API，上层都是基于底层功能的进一步并行化：

1. 手动并行：直接提供通信原语，用户可以基于通信原语通过编码，手动把模型切分到多个节点上并行，用户需要感知图切分、算子切分、集群拓扑，才能实现最优性能；

2. 半自动并行：并行逻辑和算法逻辑解耦，用户还是按单卡串行的方式写算法代码，并行逻辑变成一些配置，用户只需要配置并行策略，就不需要再通过编码实现算子切分，图切分以及和集群拓扑相关的调度；

3. 自动并行：单卡串行API，用户写单卡串行算法，由自动并行通过搜索算法，自动生成最优的切分策略，即半自动中的并行策略配置配置；

自动并行逻辑视图


Parallel Strategy Search
Dynamic Programming
Recursive Programming
Graph Partition
Layer-wise partition
Pipeline partition
Cost Model
Distributed Operator Cost
Cost Weight Estimator
Auto Tunner
Parallel Model
Distributed Operator
Distributed Data Structure
Distributed Optimizer
Distributed Auto-Grad
Distributed Tensor Layout
Device Topology & Group Manager
Collective Communication

以数据为中心的Cost Model

1. 该Cost Model是为了构建相对性能建模，不是为了构建绝对性能建模

$$T = \alpha \times D_{cal} + \beta_1 \times D_{mp\_inner\_op\_comm} + \beta_2 \times D_{mp\_inter\_op\_comm} + \gamma \times D_{dp\_comm}$$

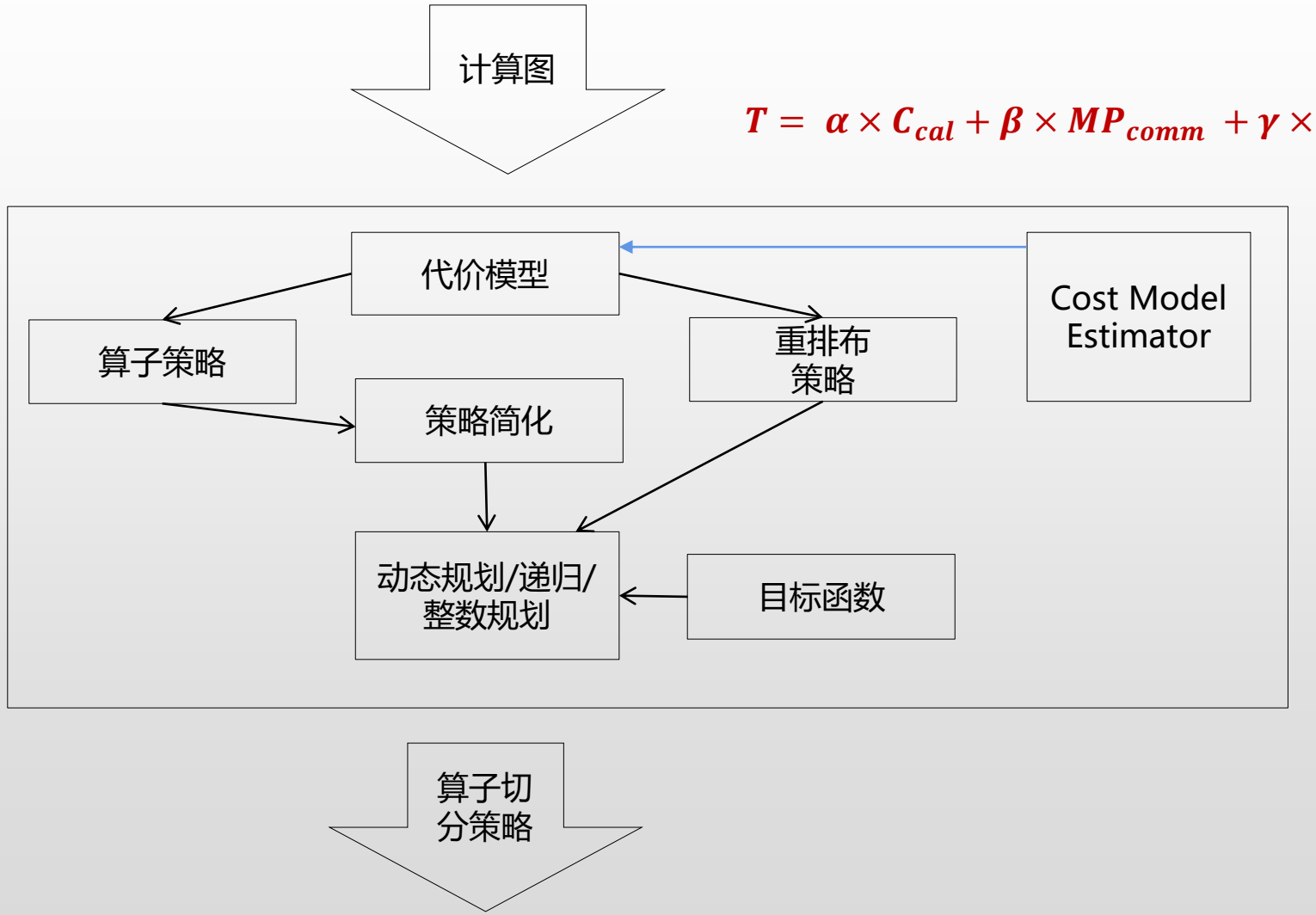
硬件无关Cost Model

1. Cube类算子计算数据量
2. Vector算子计算数据量
3. Inner-op模型并行通信数据量
4. Inter-op模型并行通信数据量
5. 数据并行通信数据量
6. 设备数

硬件相关Cost Model

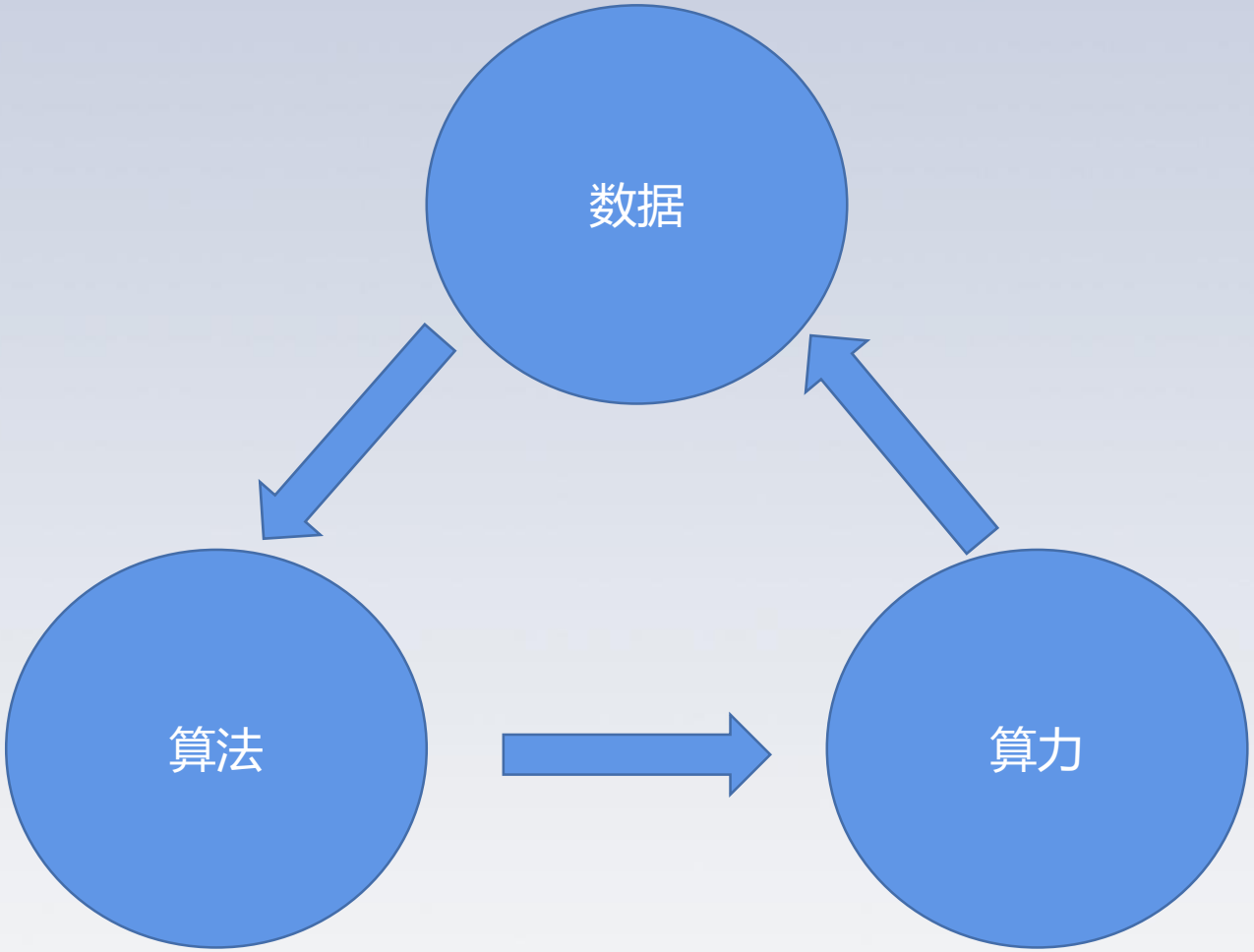
α：计算系数，Ascend/GPU/CPU不同
β：模型并行通信系数，和通信带宽和通信拓扑相关
γ：数据并行通信系数，和通信带宽拓扑相关

策略搜索与Cost Model



$$T = \alpha \times C_{cal} + \beta \times MP_{comm} + \gamma \times DP_{comm}$$

- Cost Model Estimator用来计算Cost Model系数，可以用强化学习；
- 目标函数是在内存上限下，求计算通信比最大的切分策略；


数据
算法
算力