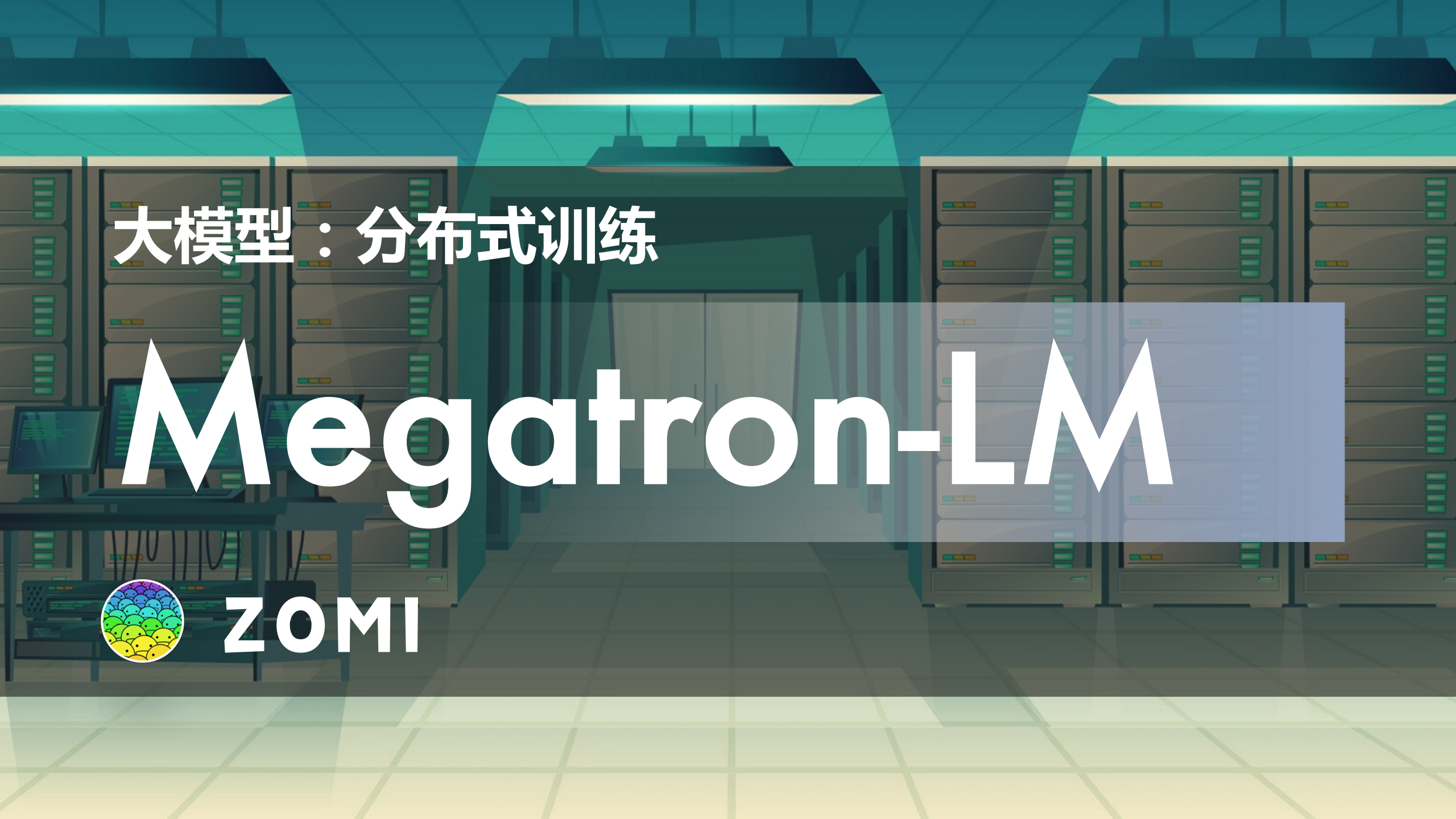

大模型：分布式训练
Megatron-LM
ZOMI

大模型业务全流程

数据 & 模型算法

2. 数据处理

- 开源数据
- 数据预处理
- 向量数据库

3. 模型算法

- LLM模型架构
- 多模态一切皆Tokens

模型训练 & 微调

4. 模型训练

- 混合精度
- 梯度检查
- 梯度累积
- …
- 5. 分布式并行
- 训练集群稳定性

6. 模型微调

- 全参微调
- 低参微调
- 指令微调

模型验证 & 推理部署

7. 模型验证

- 下游任务
- 测评标准
- bench mark

8. 推理与智能体

- 量化压缩
- 推理加速
- 9. Agent 智能体

集群算力准备

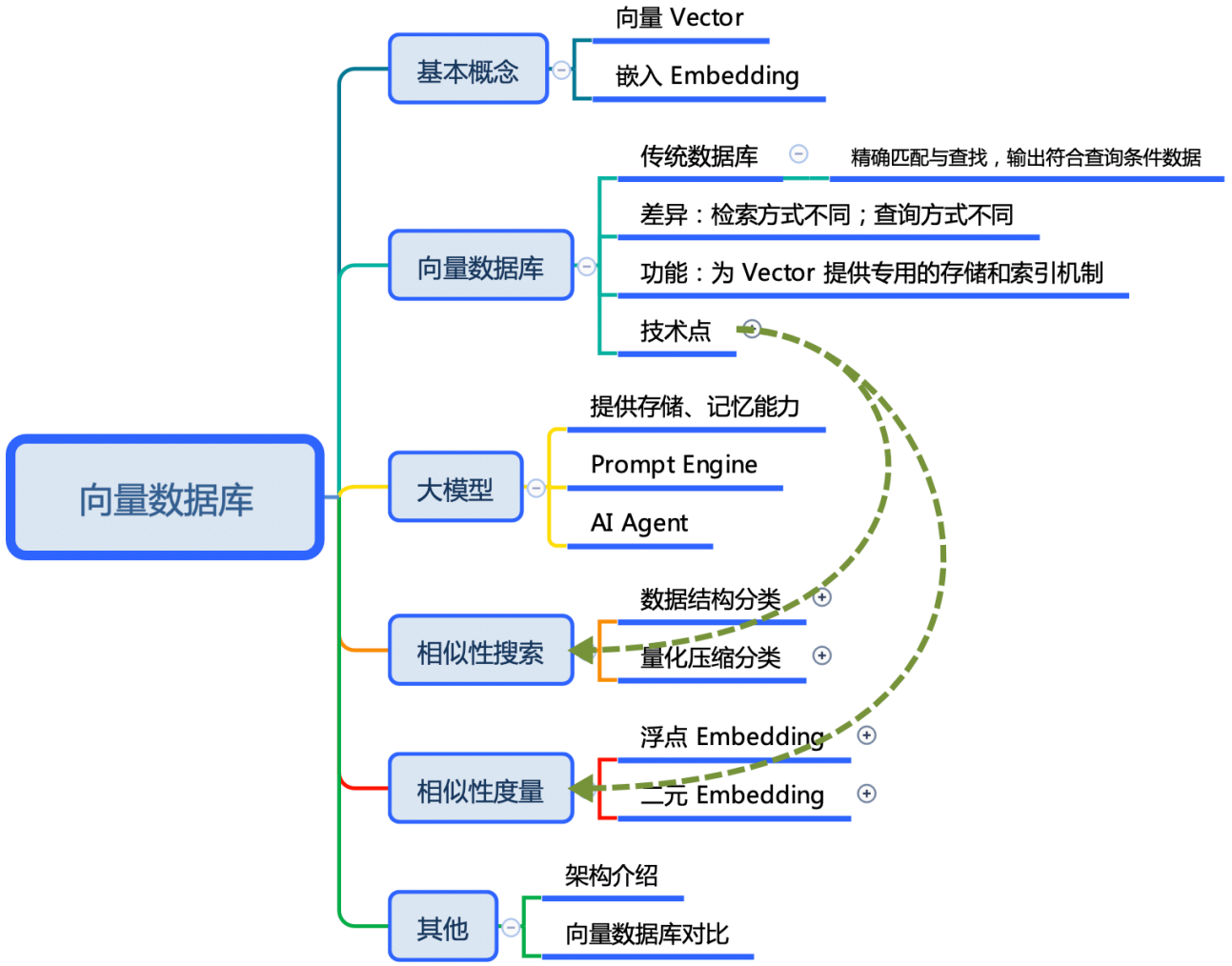
1. AI 集群建设

- 计算、存储、网络
- AI 集群机房建设
- AI 集群上线与运维

大模型系列 – 分布式训练加速

- **具体内容**
  - **分布式加速库：**业界常用分布式加速库 & 作用
  - **DeepSpeed 特性：**基本概念 - 整体框架 – Zero-1/2/3 – ZeRO-Offload – ZeRO-Infinity
  - **Megatron 特性：**总体介绍 – 整体流程 – 并行配置 – DP – TP – PP
  - **DeepSpeed 的昇腾迁移：**原生使用，不需要单独处理
  - **Megatron-LM 的昇腾迁移：**使用 Patch 方式进行替换
  - **AscendSpeed 前移 Qwen 72B**

大模型系列 – 分布式训练加速



分布式并行意义

- **深度学习训练耗时：**

训练耗时 = 训练数据规模 x 单步计算量 / 计算速率

模型相关，相对固定　　可变因素

- **计算速率：**

计算速率 = 单设备计算速率 x 设备数 x 多设备并行效率（加速比）

Moore定律+算法优化　　可变因素

分布式并行意义

- **深度学习训练耗时：**

训练耗时 = 训练数据规模 x 单步计算量 / 计算速率

模型相关，相对固定　　可变因素

- **计算速率：**

计算速率 = 单设备计算速率 x 设备数 x 多设备并行效率（加速比）

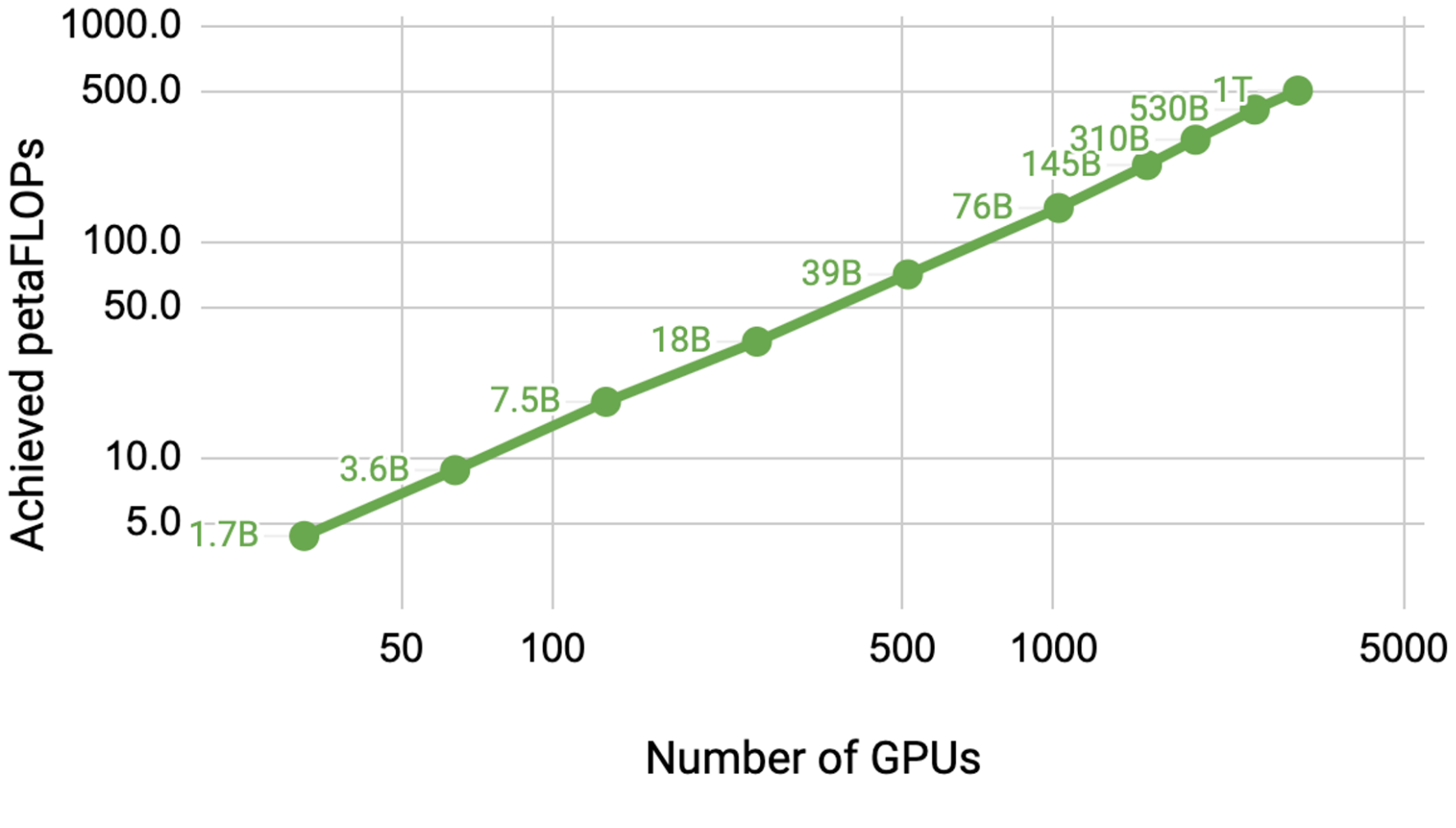
混合精度	服务器架构	数据并行
算子融合	通信拓扑优化	模型并行
梯度累加		流水并行

2. Megatron-LM

https://github.com/NVIDIA/Megatron-LM

1. Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism
2. Efficient Large-Scale Language Model Training on GPU Clusters Using Megatron-LM
3. Reducing Activation Recomputation in Large Transformer Models

Megatron-LM Achieve



Megatron-LM Achieve

Number of parameters (billion)	Attention heads	Hidden size	Number of layers	Tensor model-parallel size	Pipeline model-parallel size	Number of GPUs	Batch size	Achieved teraFlOP/s per GPU	Percentage of theoretical peak FLOP/s	Achieved aggregate petaFLOP/s
1.7	24	2304	24	1	1	32	512	137	44%	4.4
3.6	32	3072	30	2	1	64	512	138	44%	8.8
7.5	32	4096	36	4	1	128	512	142	46%	18.2
18.4	48	6144	40	8	1	256	1024	135	43%	34.6
39.1	64	8192	48	8	2	512	1536	138	44%	70.8
76.1	80	10240	60	8	4	1024	1792	140	45%	143.8
145.6	96	12288	80	8	8	1536	2304	148	47%	227.1
310.1	128	16384	96	8	16	1920	2160	155	50%	297.4
529.6	128	20480	105	8	35	2520	2520	163	52%	410.2
1008.0	160	25600	128	8	64	3072	3072	163	52%	502.0

Table 1: Weak-scaling throughput for GPT models ranging from 1 billion to 1 trillion parameters.

Megatron-LM Achieve

Scheme	Number of parameters (billion)	Model-parallel size	Batch size	Number of GPUs	Microbatch size	Achieved teraFlOP/s per GPU	Training time for 300B tokens (days)
ZeRO-3 without Model Parallelism	174.6	1	1536	384	4	144	90
				768	2	88	74
				1536	1	44	74
	529.6	1	2560*	640	4	138	169
			2240	1120	2	98	137
				2240	1	48	140
PTD Parallelism	174.6	96	1536	384	1	153	84
				768	1	149	43
				1536	1	141	23
	529.6	280	2240	560	1	171	156
				1120	1	167	80
				2240	1	159	42

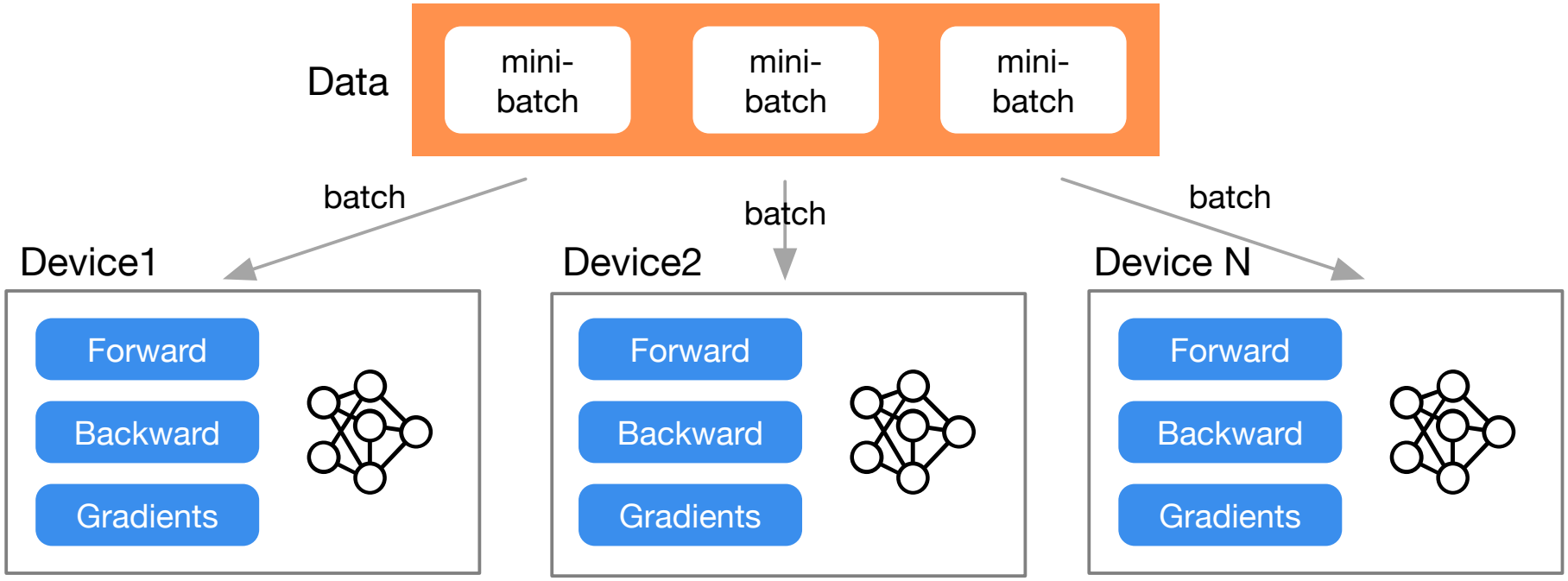
Table 2: Comparison of PTD Parallelism to ZeRO-3 (without model paralllelism). The 530-billion-parameter GPT model did not fit on 560 GPUs when using a microbatch size of 4 with ZeRO-3, so we increased the number of GPUs used to 640 and global batch size to 2560 to provide a throughput estimate (relevant row marked in table with a *).

Megatron-LM Achieve

- 要实现在 AI 集群**高吞吐量**，需要在多个方面进行创新和精心设计：
  1. 高效计算核（kernel）实现，基于计算操作 compute-bound 而非内存绑定 memory-bound
  2. 对网络模型进行多维并行 PTD，以减少网络发送的字节数，提升模型利用率 MFU
  3. 特定通信域优化和高速硬件互联

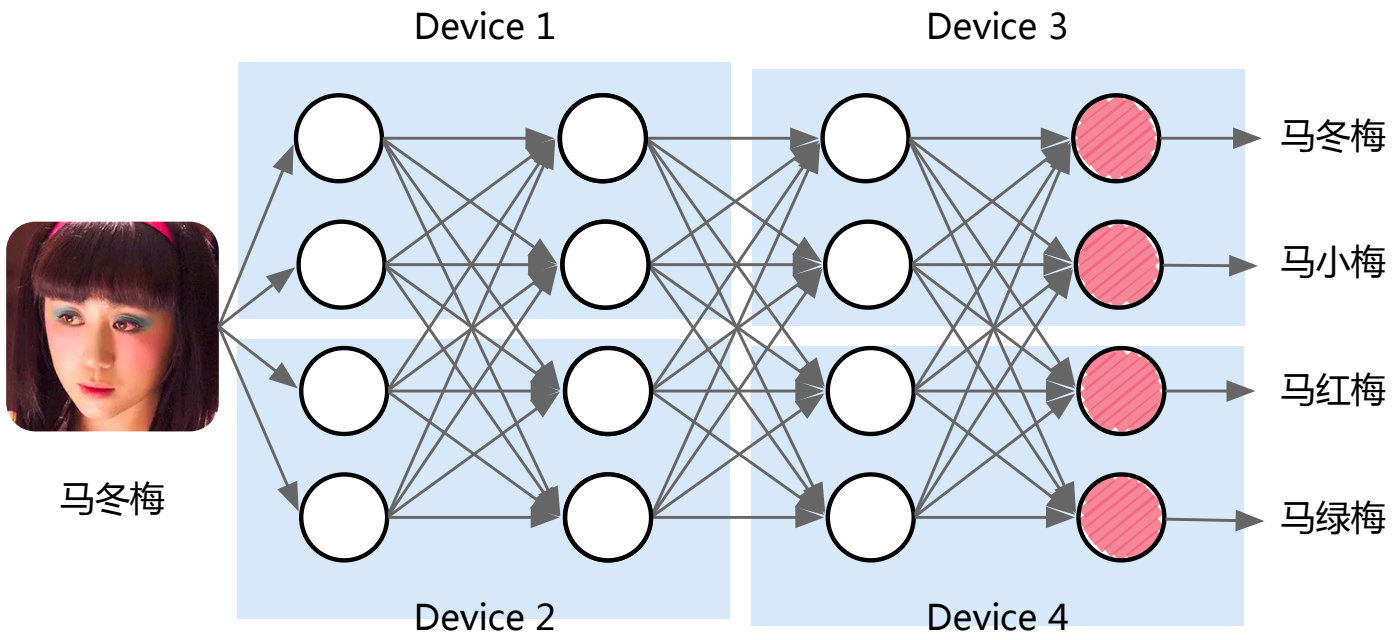
PTD : Data parallelism

1. Data parallelism, DP
2. Distribution Data Parallel, DDP
3. Fully Sharded Data Parallel, FSDP



PTD：Data parallelism

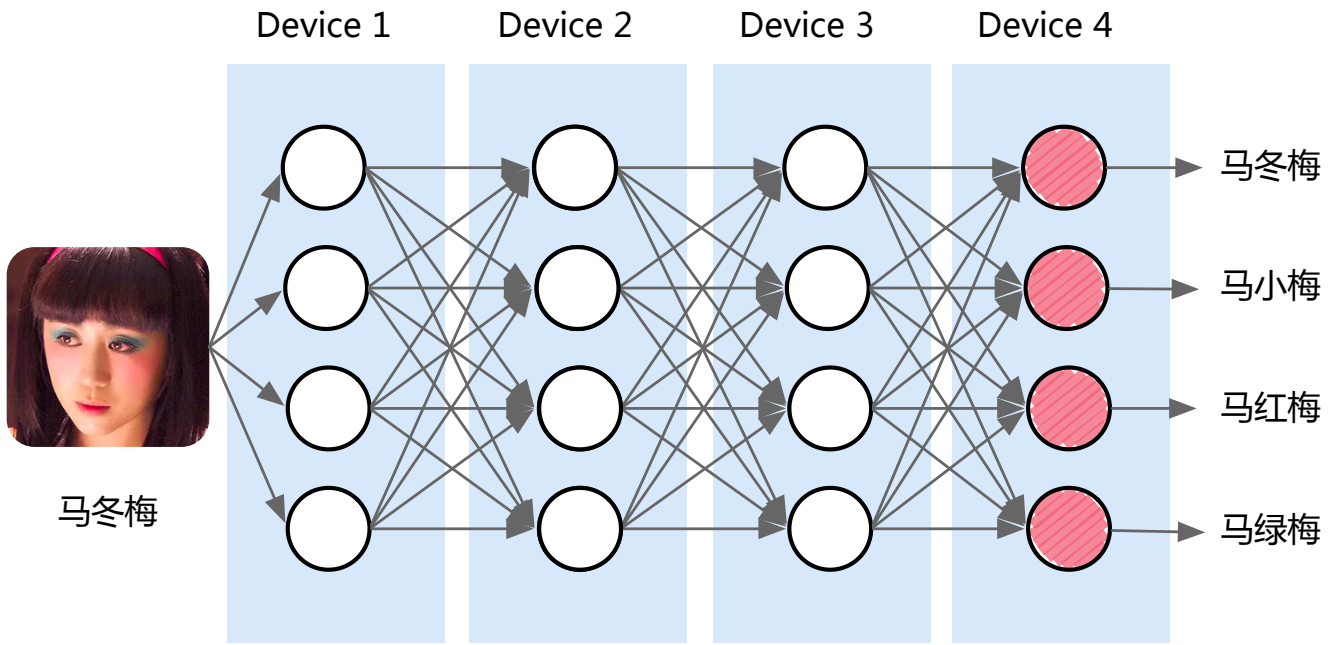
- Divide parameters in the layer into different devices, which we called tensor model parallelism
  - 张量并行：将计算图中的层内的参数切分到不同设备，即层内并行



张量并行（层内并行）

PTD：Data parallelism

- Model divided layers into different devices, which we called pipeline parallelism
  - 流水并行：按模型layer层切分到不同设备，即层间并行



流水线并行（层间并行）

Megatron-LM Code

Megatron-LM / megatron /

Add file

Shanmugam Ramasamy Merge branch 'quick_start_guide' into 'main' 4379202 · 4 days ago History

Name	Last commit message	Last commit date
..		
core	Merge branch 'quick_start_guide' into 'main'	4 days ago
data	Dataloader fixes from `training-nov2023` branch	last week
deploy	[OMNIML-614] AMMO ptq + TensorRT-LLM export examples f...	2 weeks ago
fp16_deprecated	Clean up licensing.	2 years ago
fused_kernels	Merges various github PRs	8 months ago
model	Put embedding layers in separate buckets to make sure embe...	last month
mpu/tests	Attempt to fix warnings by using the latest APIs	4 months ago

2.1 Megatron-LM

整体代码架构

Step1：启动

I. 分布式启动

- 启动脚本在 examples/pretrain_bert_distributed.sh，其利用了 torch.distributed.launch 来启动多个进程。具体业务代码是 pretrain_gpt.py。
- 因为 GPUS_PER_NODE 是 8，所以 nproc_per_node 是 8，这样，在本机上就启动了8个进程，每个进程之中含有**模型的一部分**。进程的 rank 是被 torch.distributed.launch 调用 elastic 自动分配。

Step1：启动

- nnodes：节点的数量，通常一个节点对应一个主机，方便记忆，直接表述为主机
- node_rank：节点的序号，从0开始
- nproc_per_node：一个节点中显卡的数量
- master_addr：master节点的ip地址，也就是0号主机的IP地址，该参数是为了让 其他节点 知道0号节点的位，来将自己训练的参数传送过去处理
- master_port：master节点的port号，在不同的节点上master_addr和master_port的设置是一样的，用来进行通信

Step1：启动

I. 构造基础

- **获取模型**
  - model_provider返回模型普通版本（vanilla version）。所谓vanilla，我们指的是一个简单的cpu模型，没有 fp16或 ddp，但是已经被 Megatron 改造为并行的版本。
- **获取数据集**
  - train_valid_test_datasets_provider 接受train/valid/test数据集的大小，返回 “train，valid，test” 数据集。
- **步进函数**
  - forward_step 接受“数据迭代器”和“模型”，并返回“loss”标量，该标量带有一个字典，其中key:value是希望在训练期间监视信息，例如“lm loss:value”。

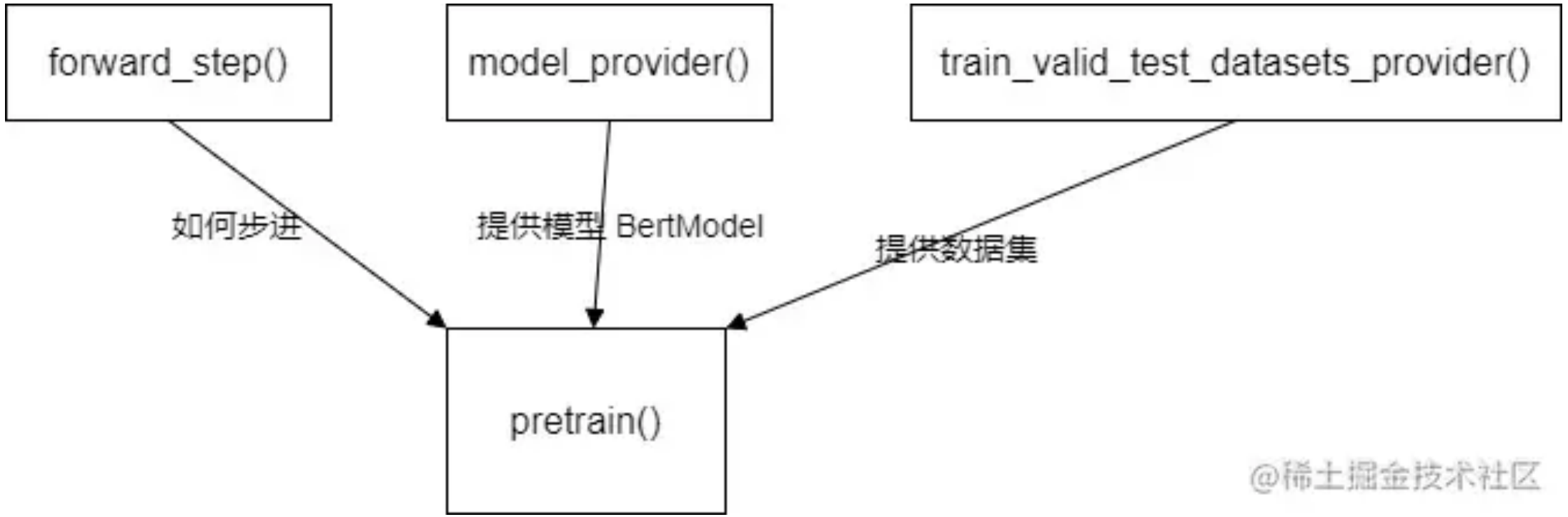
Step1：启动

I. 构造基础

- **广播数据**
  - forward_step 会调用 get_batch 获取batch 数据，其内部会从迭代器获取数据，然后使用broadcast_data函数把输入数据从 rank 0 广播到所有tensor-model-parallel 其他 ranks之上。

Step1：启动

- 逻辑图具体如下，三个不同的函数分别为预训练提供不同的功能输入，做到了解耦。



Step2： Pretrain 函数

1. 初始化Megatron。
2. 使用model_provider设置模型、优化器和lr计划。
3. 调用train_val_test_data_provider以获取train/val/test数据集。
4. 使用forward_step_func训练模型。

Step3： 初始化

1. initialize_megatron

- initialize_megatron 设置全局变量，初始化分布式环境等。

Step3： 初始化

I. 初始化分布式环境：

- _initialize_distributed() 位于 megatron/initialize.py：
  - 调用 torch.distributed.init_process_group 初始化分布式环境
  - 调用 mpu.initialize_model_parallel 来设置模型并行，数据并行等各种进程组
- 创建完worker进程之后，程序需要知道哪些进程在训练同一个模型，torch.distributed.init_process_group 就实现了这个功能。
- torch.distributed.init_process_group 会生成一个进程组，同组内进程训练同一个模型，也能确定用什么方式进行通信。
- 进程组会给组内每个进程一个序号，就是gloabl rank，如果是多机并行，每个机器创建的进程之间也有一个序号，就是 local rank。如果是单机多卡并行，local rank 和 global rank是一致的。

Step3： 初始化

I. 初始化进程组全局变量：

- _initialize_distributed() 位于 megatron/initialize.py：
  - 调用 torch.distributed.init_process_group 初始化分布式环境
  - 调用 mpu.initialize_model_parallel 来设置模型并行，数据并行等各种进程组
- 创建完worker进程之后，程序需要知道哪些进程在训练同一个模型，torch.distributed.init_process_group就实现了这个功能。
- torch.distributed.init_process_group 会生成一个进程组，同组内进程训练同一个模型，也能确定用什么方式进行通信。
- 进程组会给组内每个进程一个序号，就是gloabl rank，如果是多机并行，每个机器创建的进程之间也有一个序号，就是 local rank。如果是单机多卡并行，local rank 和 global rank是一致的

Step3： 初始化

I. 初始化进程组全局变量：

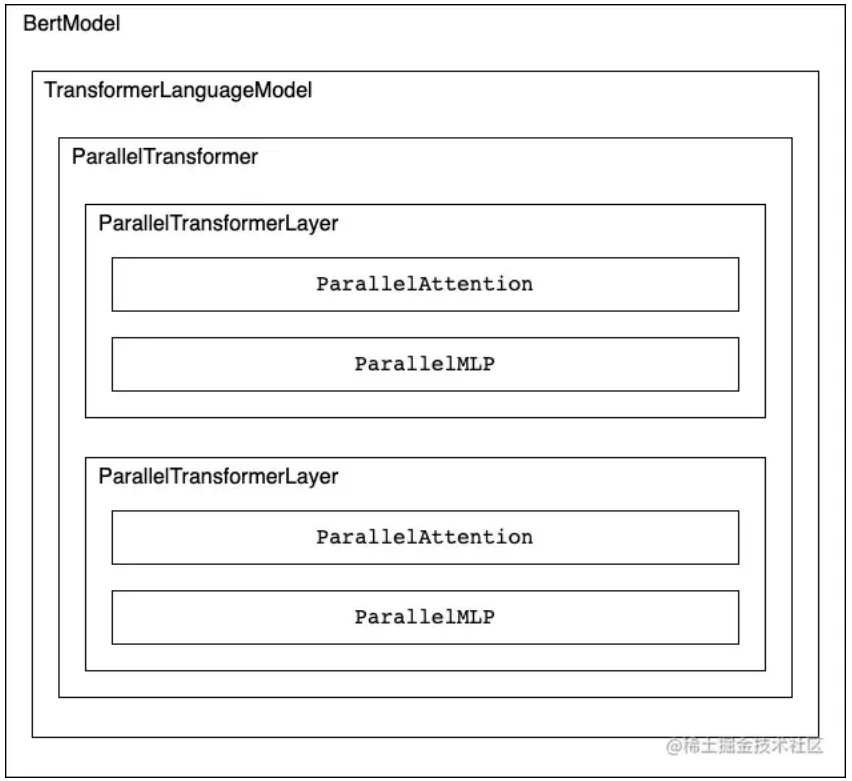
- 调用 mpu.initialize_model_parallel 设置MP、DP等进程组，每个 rank 对应进程都有自己全局变量
  - _TENSOR_MODEL_PARALLEL_GROUP ：当前 rank 所属的 Intra-layer model parallel group，TP 进程组。
  - _PIPELINE_MODEL_PARALLEL_GROUP ：当前 rank 所属的Intra-layer model parallel group，PP进程组。
  - _MODEL_PARALLEL_GROUP ：当前 rank 所属于MP进程组，包括了 TP 和 PP。
  - _EMBEDDING_GROUP ： Embedding 并行对应进程组。
  - _DATA_PARALLEL_GROUP ：当前 rank 所属的 DP 进程组。

Step4：设置模型

1. setup_model_and_optimizer：设置模型和优化器，其中重点是get_model
2. GPTModel：其主要调用了 get_language_model
3. 语言模型： get_language_model 会获取一个 TransformerLanguageModel
   - TransformerLanguageModel 就是具体的语言模型，其中重要的是 ParallelTransformer。这里会依据传入的配置来进行生成。
   - 如果是第一层，即有 pre_process，则会加入 embedding layer。
   - 如果是中间层，则会根据 encoder 还是 decoder 来生成对应的 ParallelTransformer。
   - 如果是最后一层，即有 post_process，则会加入 Pooler，在外层 BertModel 也会有对应处理。

Step4：设置模型

- ParallelTransformer：会调用 ParallelTransformerLayer 生成具体的 Transformer层， ParallelTransformer 包括多个 Transformer，其中每层 Transformer 是一个 ParallelTransformerLayer。



Step4：设置模型

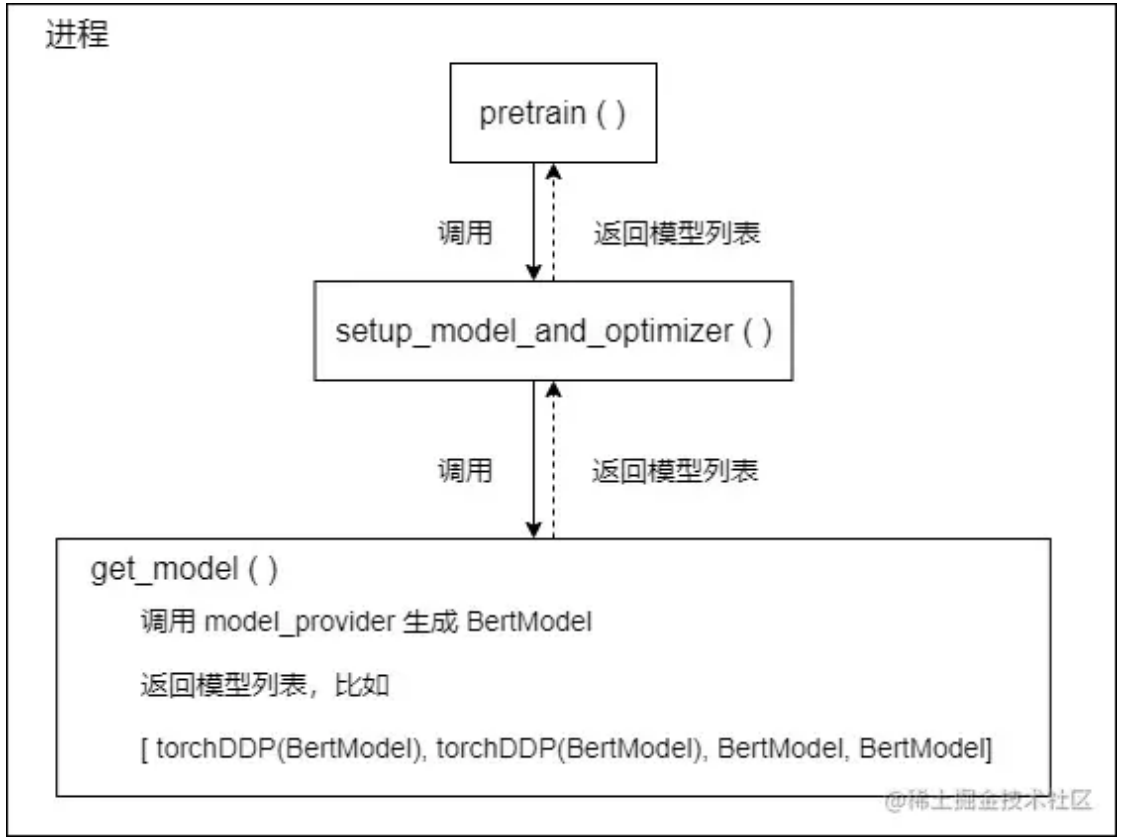
- ParallelTransformer：会调用 ParallelTransformerLayer 生成具体的 Transformer层， ParallelTransformer 包括多个 Transformer，其中每层 Transformer 是一个 ParallelTransformerLayer。
  - get_num_layers：获取层数，即获取本模型在并行处理状况下，应该拥有多少层
  - get_pipeline_model_parallel_world_size：获取本流水线组world size数目，就是流水线深度
  - 前向传播： ParallelTransformerLayer 的 forward 方法

Step4：设置模型

- get_model： GPT 中含多层 transformer，直接按照层数切分，每层相同 Transformer layer。分布式并行启动了 N 个进程，每个进程里面有一个子模型，即原始 GPT 模型部分层。但怎么知道每个子模型包含了多少层？ → 通过 initialize_megatron() 建立的进程组，get_model() 会依据目前进程组情况进行处理。单个进程内模型获取如下：
  - 如果有 virtual 设置，遍历 virtual size，生成对应数模型 GPTModel
  - 否则是 encoder_and_decoder，则对 split 进行配置
  - 设置 tensor model parallel 属性
  - 把模型放置到 GPU 之上
  - 如果需要数据并行，则配置DDP

Step4：设置模型

- get_model： 分布式并行启动了 N 个进程，每个进程里面有一个子模型，即原始 GPT 模型部分层。通过 initialize_megatron() 建立的进程组，get_model() 会依据目前进程组情况进行处理。

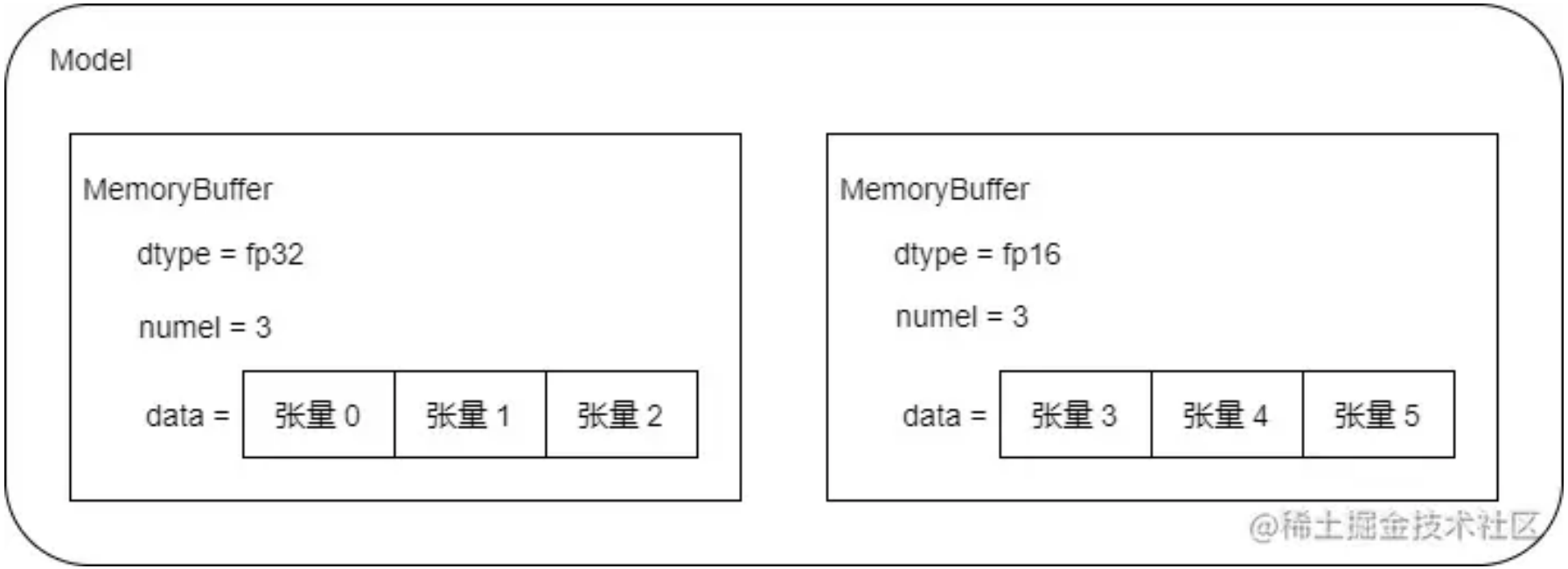


Step5：数据并行

1. 设置数据： build_train_valid_test_data_iterators() 对数据进行处理，提供 train()/valid()/test() 不同数据集
2. 分布式数据DDP： Megatron-LM 中单独实现分布式数据并行 DistributedDataParallel()
   - 使用连续内存来存储和累积梯度，每一种类型的张量属于一个统一的内存，可以统一执行 allreduce
   - __init__() 初始化目的是把同类型梯度连续存储
   - MemoryBuffer() 是内存抽象
   - _make_param_hook() 用于拷贝梯度
   - zero_grad_buffer() 用于将buffer清零

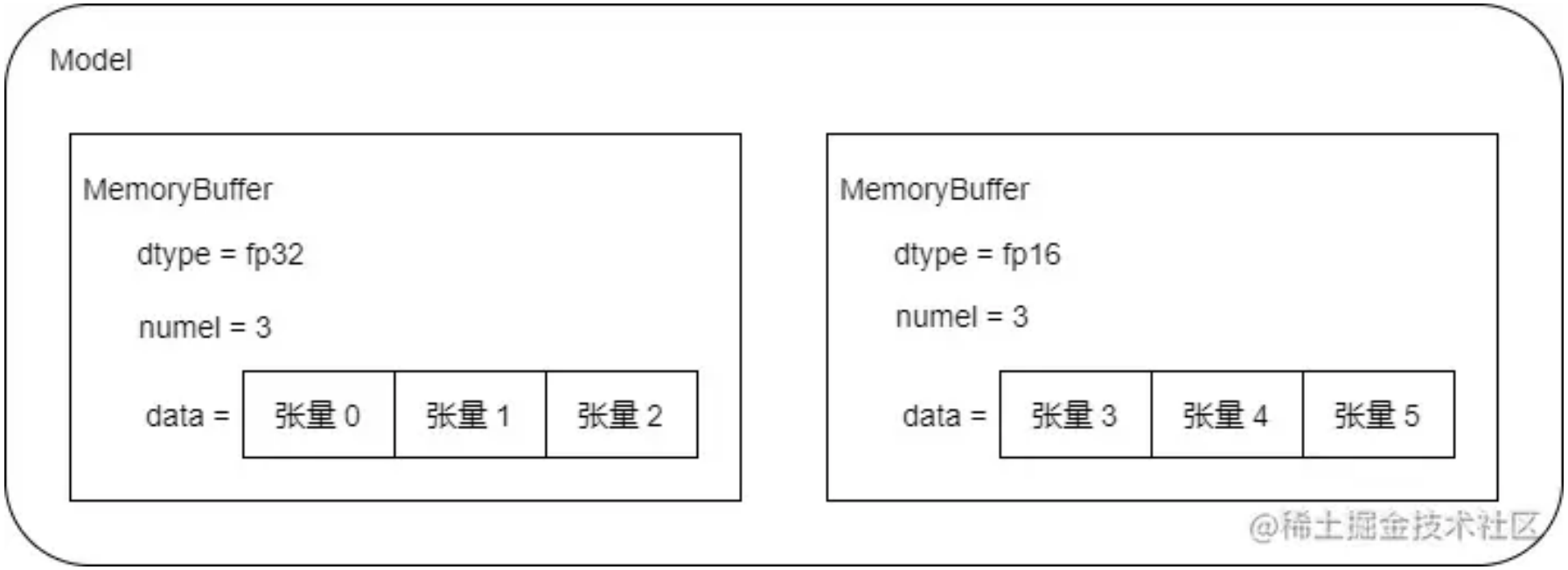
Step5：数据并行

1. 分布式数据DDP： Megatron-LM 中单独实现分布式数据并行 DistributedDataParallel()
   - e.g. 假设模型有6个参数，3个 fp32，3 个 fp16，被组合成两个连续内存 MemoryBuffer



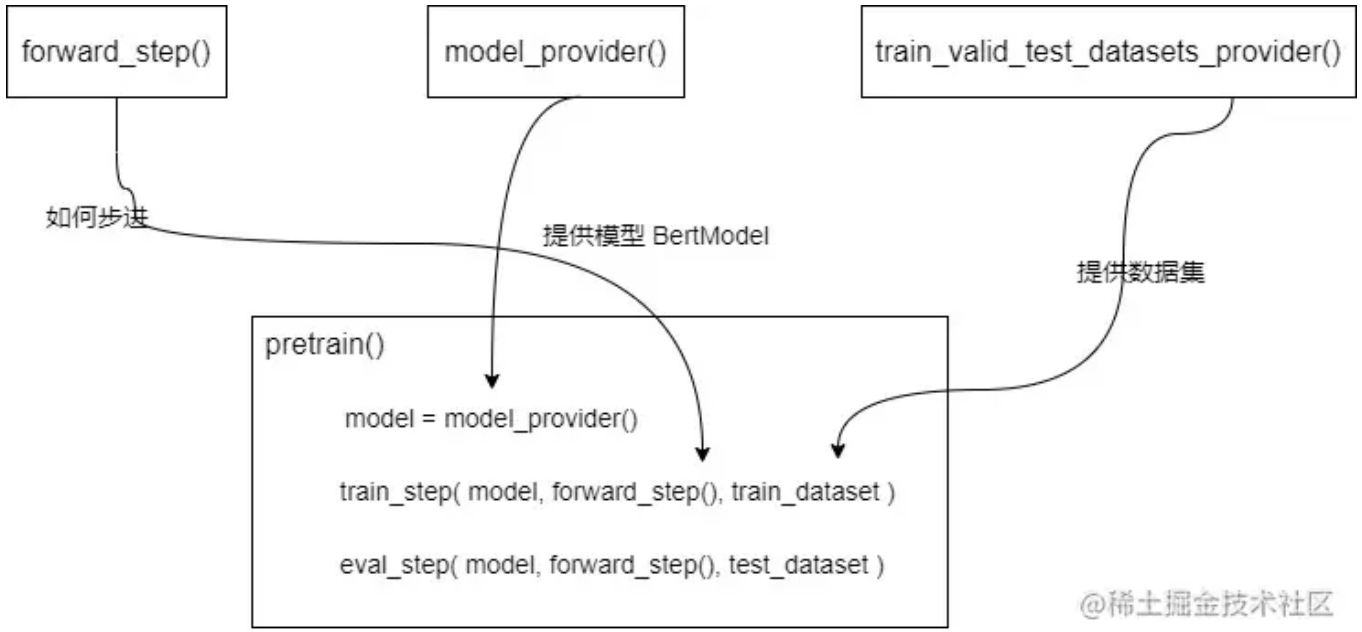
Step5：数据并行

1. 分布式数据DDP： Megatron-LM 中单独实现分布式数据并行 DistributedDataParallel()
   - e.g. 假设模型有6个参数，3个 fp32，3 个 fp16，被组合成两个连续内存 MemoryBuffer



Step6：模型训练

1. **训练step**：train_step() 获取 get_forward_backward_func() 得到 schedule，因为是流水线并行，所以需要 schedule 如何具体训练
2. **获取schedule**： get_forward_backward_func 获取 pipeline 的schedule，这里分为 flush 和 interleaving 两种。



Thank you

把AI系统带入每个开发者、每个家庭、每个组织，构建万物互联的智能世界

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

ZOMI

Course chenzomi12.github.io

GitHub github.com/chenzomi12/DeepLearningSystem