大模型技术及开发

提示工程

主讲人：陈小军

时间：2025.3.21

有监督微调

有监督微调（Supervised Finetuning，SFT）又称指令微调（Instruction Tuning），是指在已经训练好的语言模型的基础上，通过使用有标注的特定任务数据进行进一步的微调，使模型具备遵循指令的能力。

经过海量数据预训练后的语言模型虽然具备了大量的“知识”，但是由于其训练时的目标仅是进行下一个词的预测，因此不能够理解并遵循人类自然语言形式的指令。为了使模型具有理解并响应人类指令的能力，还需要使用指令数据对其进行微调。

如何构造指令数据，如何高效低成本地进行指令微调训练，以及如何在语言模型基础上进一步扩大上下文等问题，是大语言模型在有监督微调阶段的核心。

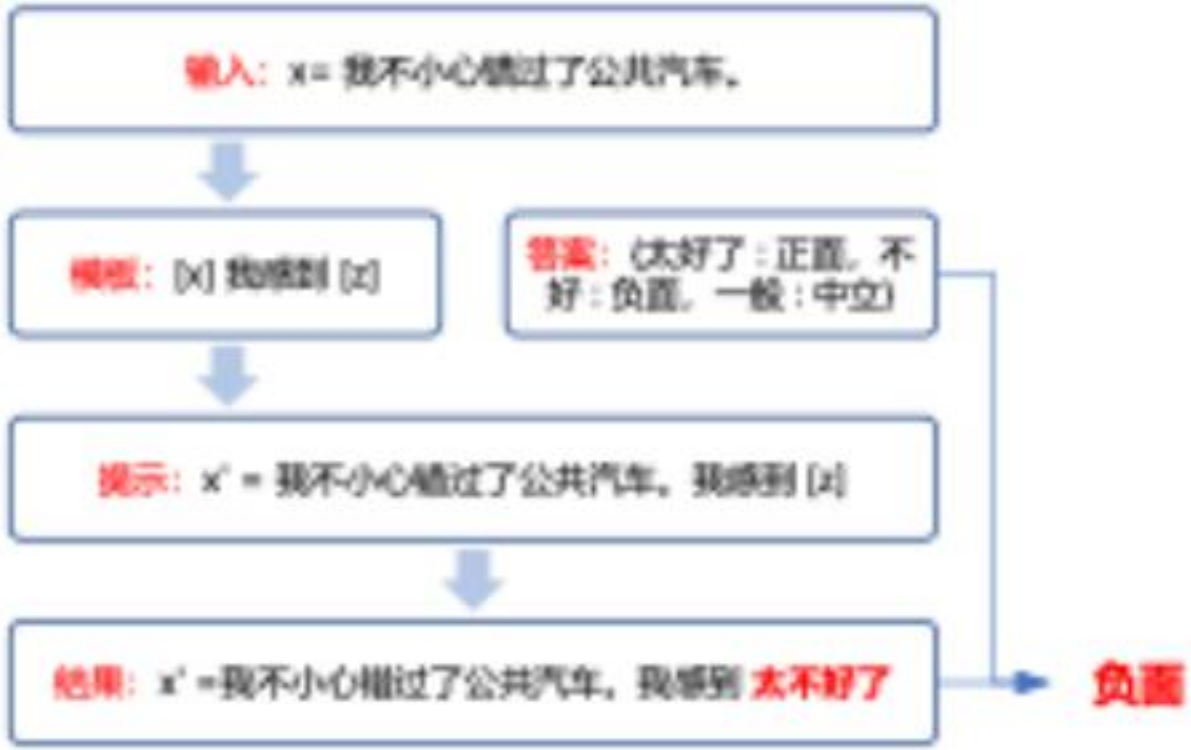
目录

contents


01
简介
202X

提示学习

提示学习（Prompt-based Learning）不同于传统的监督学习，它直接利用了在大量原始文本上进行预训练的语言模型，并通过定义一个新的提示函数，使该模型能够执行小样本甚至零样本学习，以适应仅有少量标注或没有标注数据的新场景。



原始输入x 经过一个模板，被修改成一个带有一些未填充槽的文本提示x'，然后将这段提示输入语言模型，语言模型即以概率的方式填充模板中待填充的信息，然后根据模型的输出导出最终的预测标签 y

提示学习

使用提示学习完成预测的整个过程可以描述为三个阶段：提示添加、答案搜索、答案映射。

（1）提示添加：在这一步骤中，需要借助特定的模板，将原始的文本和额外添加的提示拼接起来，一并输入到语言模型中。例如，在情感分类任务中，根据任务的特性，可以构建如下含有两个插槽的模板：

“[X] 我感到[Z]”

其中，[X] 插槽中填入待分类的原始句子，[Z] 插槽中为需要语言模型生成的答案。

假如原始文本

x = “我不小心错过了公共汽车。”

通过此模板，整段提示将被拼接成

x' = “我不小心错过了公共汽车。我感到[Z]”


02
数据影响分析
202X

数据主要影响因素

大语言模型的训练需要巨大的计算资源，通常不可能多次迭代大语言模型预训练。千亿级参数量的大语言模型每次预训练的计算需要花费数百万元。因此，在训练大语言模型之前，构建一个准备充分的预训练语料库尤为重要。

本节将从**数据规模、数据质量和数据多样性**三个方面分析数据对大语言模型的性能影响。

数据规模对大语言模型性能的影响

数据质量对大语言模型性能的影响

数据多样性对大语言模型性能的影响

03

开源数据

2025

04

训练任务

2025

04

分布式训练

2025

分布式训练概述

大语言模型参数量和所使用的数据量都非常巨大，因此都采用了分布式训练架构完成训练。

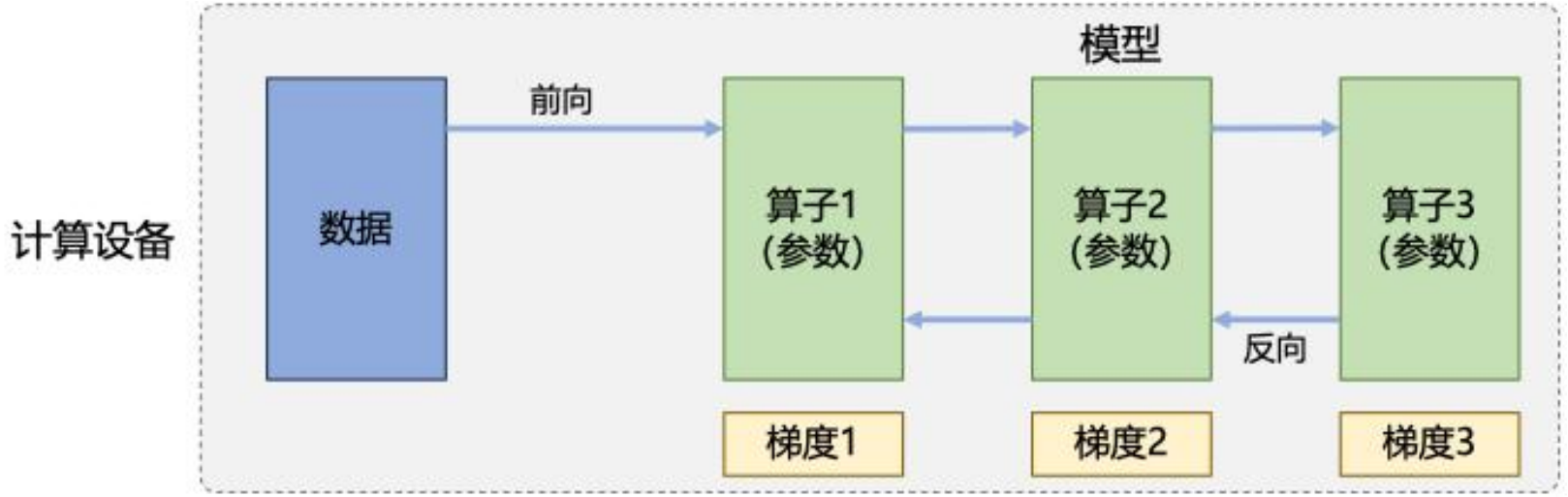
- **OPT模型**训练使用了992块NVIDIA A100 80G GPU，采用全分片数据并行（Fully Sharded Data Parallel）以及Megatron-LM张量并行（Tensor Parallelism），整体训练时间将近2个月。
- **BLOOM模型**的研究人员则公开了更多在硬件和所采用的系统架构方面的细节。该模型的训练一共花费3.5个月，使用48个计算节点。每个节点包含8块NVIDIA A100 80G GPU（总计384个GPU），并且使用4*NVLink用于节点内部GPU之间通信。节点之间采用四个Omni-Path 100 Gbps网卡构建的增强8维超立方体全局拓扑网络进行通信。
- **LLaMA模型**训练采用NVIDIA A100-80GB GPU，LLaMA-7B模型训练需要 82432 GPU小时，LLaMA-13B模型训练需要 135168 GPU小时， LLaMA-33B模型训练花费了 530432 GPU小时，而LLaMA-65B模型训练花费则高达 1022362 GPU小时。

分布式训练挑战

- **计算墙：**单个计算设备所能提供的计算能力与大语言模型所需的总计算量之间存在巨大差异。2022年3月发布的NVIDIA H100 SXM的单卡FP16算力也只有 2000 TFLOPs，而 GPT-3 则需要 314 ZFLOPs 的总计算量，两者相差了 8 个数量级。
- **显存墙：**单个计算设备无法完整存储一个大语言模型的参数。GPT-3包含1750亿参数，在推理阶段如果采用FP32格式进行存储，需要700GB的计算设备内存空间，而 NVIDIA H100 GPU只有80GB显存。
- **通信墙：**分布式训练系统中各计算设备之间需要频繁地进行参数传输和同步。由于通信的延迟和带宽限制，这可能成为训练过程的瓶颈。GPT-3训练过程中，如果分布式系统中存在128个模型副本，那么在每次迭代过程中至少需要传输 89.6TB的梯度数据。而截止2023年8月，单个InfiniBand链路仅能够提供不超过800Gb/s 带宽。

分布式训练的并行策略

分布式训练系统目标就是将单节点模型训练转换成等价的分布式并行模型训练。对于大语言模型来说，训练过程就是根据数据和损失函数，利用优化算法对神经网络模型参数进行更新的过程。



单节点模型训练系统结构如上图所示，主要由数据和模型两个部分组成。

分布式训练的并行策略

根据单个设备模型训练系统的流程，可以看到，如果进行并行加速，可以从**数据和模型**两个维度进行考虑。

- 数据进行切分（Partition），并将同一个模型复制到多个设备上，并行执行不同的数据分片，这种方式通常被称为**数据并行（Data Parallelism，DP）**。
- 模型进行划分，将模型中的算子分发到多个设备分别完成，这种方式通常被称为**模型并行（Model Parallelism，MP）**。
- 当训练大语言模型时，往往需要同时对数据和模型进行切分，从而实现更高程度的并行，这种方式通常被称为**混合并行（Hybrid Parallelism，HP）**。

数据并行

在数据并行系统中，每个计算设备都存
网络模型的完整副本（Model Replic
迭代时，每个计算设备只分配了一个批
本的子集，并根据该批次样本子集的数
络模型的前向计算。

假设一个批次的训练样本数为N，使用
设备并行计算，每个计算设备会分配到
本。前向计算完成后，每个计算设备都
地样本计算损失误差得到梯度G_i（i 为
号），并将本地梯度G_i 进行广播。所
需要聚合其他加速度卡给出的梯度值，
平均梯度　　对模型进行更新，完成该
批次训练。

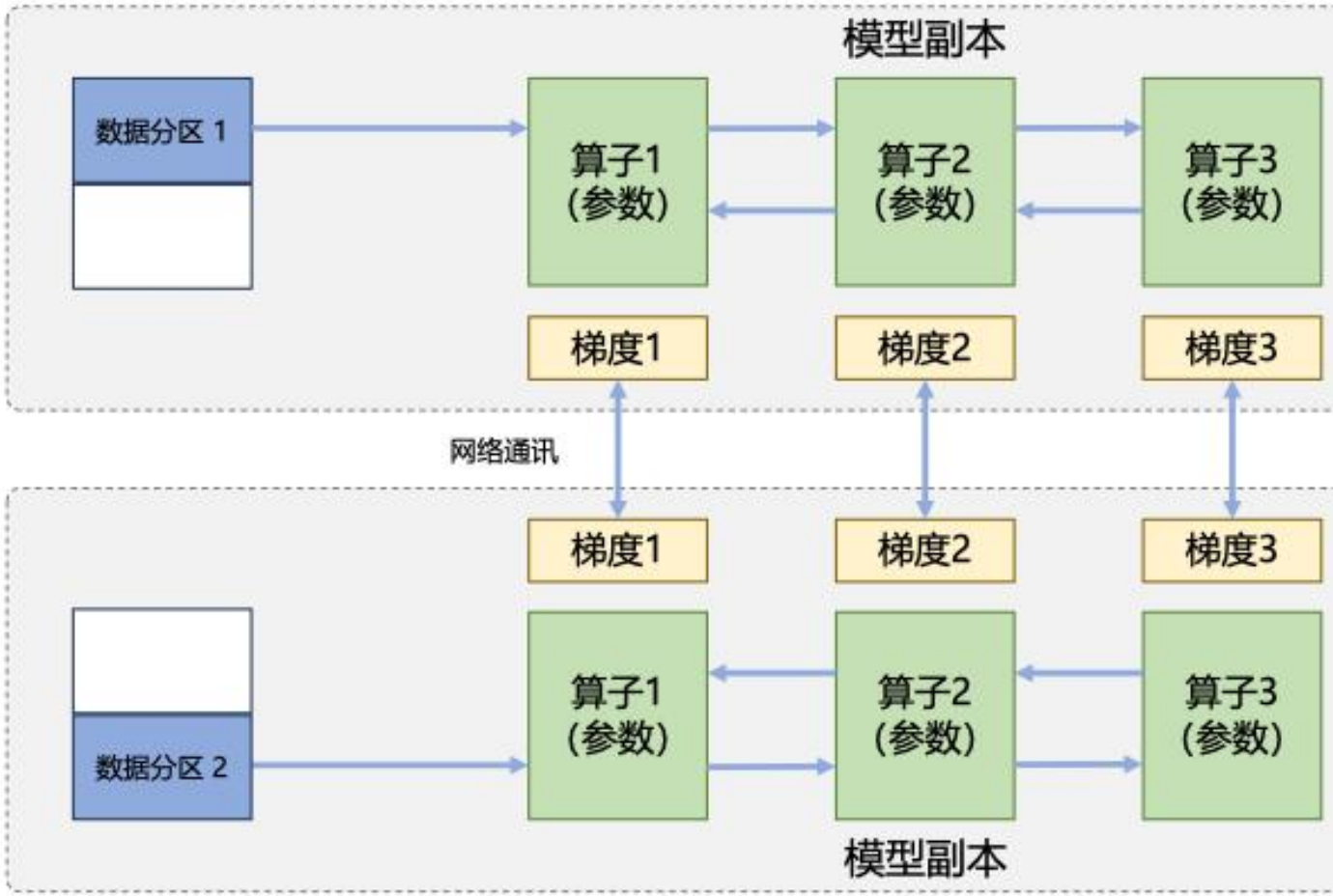


图4.4 由两个计算设备组成的数据并行训练系统样例

数据并行

使用PyTorch DistributedDataParallel 实现单个服务器多加速卡训练的代码如下

```python
class DistributedSampler(Sampler):
    def __init__(self, dataset, num_replicas=None, rank=None, shuffle=True, seed=0):
        if num_replicas is None:
            if not dist.is_available():
                raise RuntimeError("Requires distributed package to be available")
            num_replicas = dist.get_world_size()
        if rank is None:
            if not dist.is_available():
                raise RuntimeError("Requires distributed package to be available")
            rank = dist.get_rank()
        self.dataset = dataset  # 数据集
        self.num_replicas = num_replicas  # 进程个数，默认等于world_size(GPU个数)
        self.rank = rank   # 当前属于哪个进程/哪块GPU
        self.epoch = 0
        self.num_samples = int(math.ceil(len(self.dataset) * 1.0 / self.num_replicas))
                                        # 每个进程的样本个数
        self.total_size = self.num_samples * self.num_replicas  # 数据集总样本的个数
        self.shuffle = shuffle  # 是否要打乱数据集
        self.seed = seed

    def __iter__(self):
        # 1. Shuffle处理：打乱数据集顺序
        if self.shuffle:
            # 根据epoch和种子进行混淆
            g = torch.Generator()
            # 这里self.seed是一个定值，通过set_epoch改变self.epoch可以改变我们的初始化种子
            # 这就可以让每一个epoch中数据集的打乱顺序不同，使每一个epoch中
            # 每一块GPU得到的数据都不一样，这样有利于更好的训练
            g.manual_seed(self.seed + self.epoch)
            indices = torch.randperm(len(self.dataset), generator=g).tolist()
        else:
            indices = list(range(len(self.dataset)))

        # 数据补充
        indices += indices[:(self.total_size - len(indices))]
        assert len(indices) == self.total_size

        # 分配数据
        indices = indices[self.rank:self.total_size:self.num_replicas]
        assert len(indices) == self.num_samples
        return iter(indices)

    def __len__(self):
        return self.num_samples

    def set_epoch(self, epoch):
        r"""
        设置此采样器的训练轮数epoch
        当:attr:`shuffle=True`时，确保所有副本在每个轮数使用不同的随机顺序
        否则，此采样器的下一次迭代将产生相同的顺序

        Arguments:
            epoch (int): 训练轮数
        """
        self.epoch = epoch
```

DistributedSampler 类，将数据集的样本随机打乱并分配到不同计算设备

模型并行

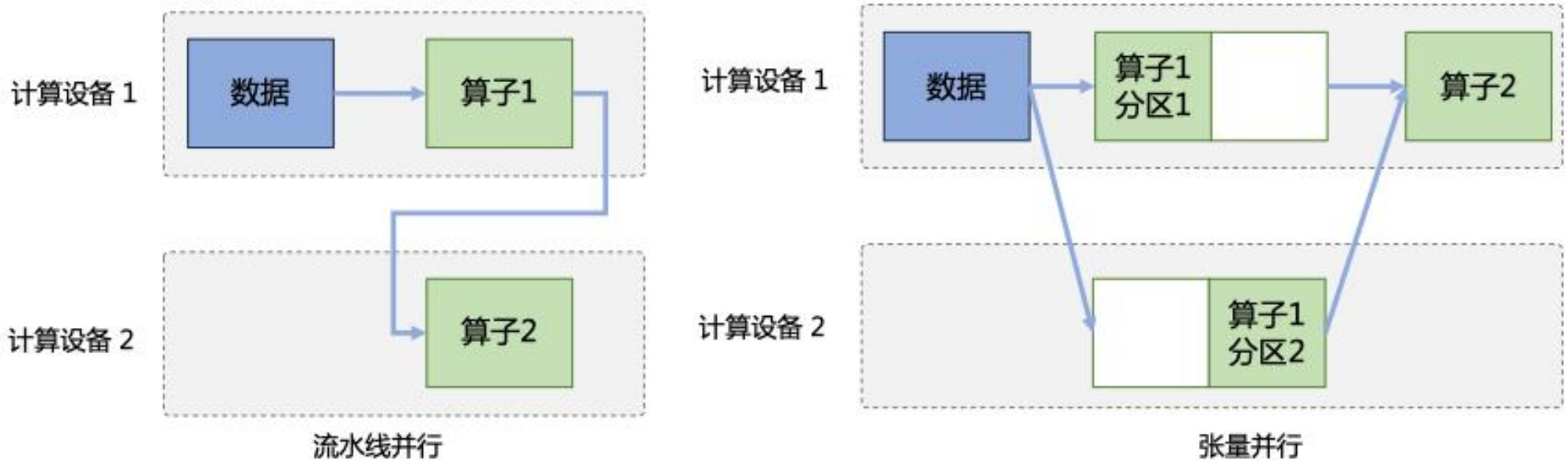


图4.5 两节点模型并行训练系统样例

模型并行往往用于解决单节点内存不足的问题。可以从计算图角度，用以下两种形式进行切分：

1. 按模型的层切分到不同设备，即层间并行或算子间并行（Inter-operator Parallelism），也称之为**流水线并行（Pipeline Parallelism，PP）**。
2. 将计算图层内的参数切分到不同设备，即层内并行或算子内并行（Intra-operator Parallelism），也称之为**张量并行（Tensor Parallelism，TP）**。

模型并行-流水线并行

流水线并行是一种并行计算策略，将模型的各个层分段处理，并将每个段分布在不同的计算设备上，使得前后阶段能够流水式、分批进行工作。流水线并行通常应用于大语言模型的并行系统中，以有效解决单个计算设备内存不足的问题。

图4.6 流水线并行样例

图4.6 给出了一个由四个计算设备组成的流水线并行系统，包含了前向计算和后向计算。其中F1、F2、F3、F4 分别代表四个前向路径，位于不同的设备上；而B4、B3、B2、B1 则代表逆序的后向路径，也分别位于四个不同的设备上。

模型并行-流水线并行

流水线并行是一种并行计算策略，将模型的各个层分段处理，并将每个段分布在不同的计算设备上，使得前后阶段能够流水式、分批进行工作。流水线并行通常应用于大语言模型的并行系统中，以有效解决单个计算设备内存不足的问题。

图4.6 流水线并行样例

图4.6 给出了一个由四个计算设备组成的流水线并行系统，包含了前向计算和后向计算。其中F1、F2、F3、F4 分别代表四个前向路径，位于不同的设备上；而B4、B3、B2、B1 则代表逆序的后向路径，也分别位于四个不同的设备上。

模型并行-张量并行

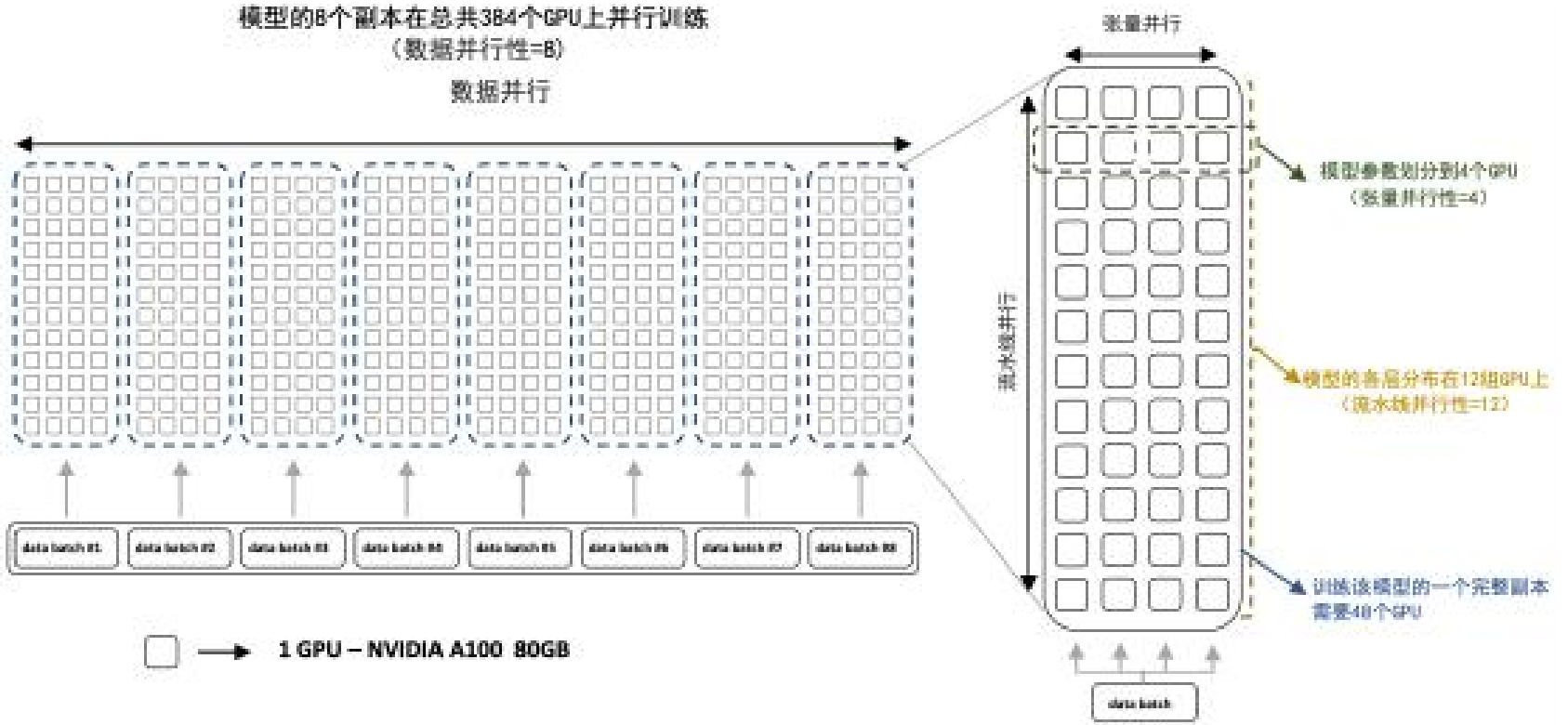
张量并行需要根据模型的具体结构和算子类型，解决如何将参数切分到不同设备，以及如何保证切分后数学一致性这两个问题。

大语言模型都是以Transformer 结构为基础，Transformer 结构主要由嵌入式表示（Embedding）、矩阵乘（MatMul）和交叉熵损失（Cross Entropy Loss）计算构成。

这三种类型的算子有较大的差异，都需要设计对应的张量并行策略[135] 才可以实现将参数切分到不同的设备。

模型并行-混合并行

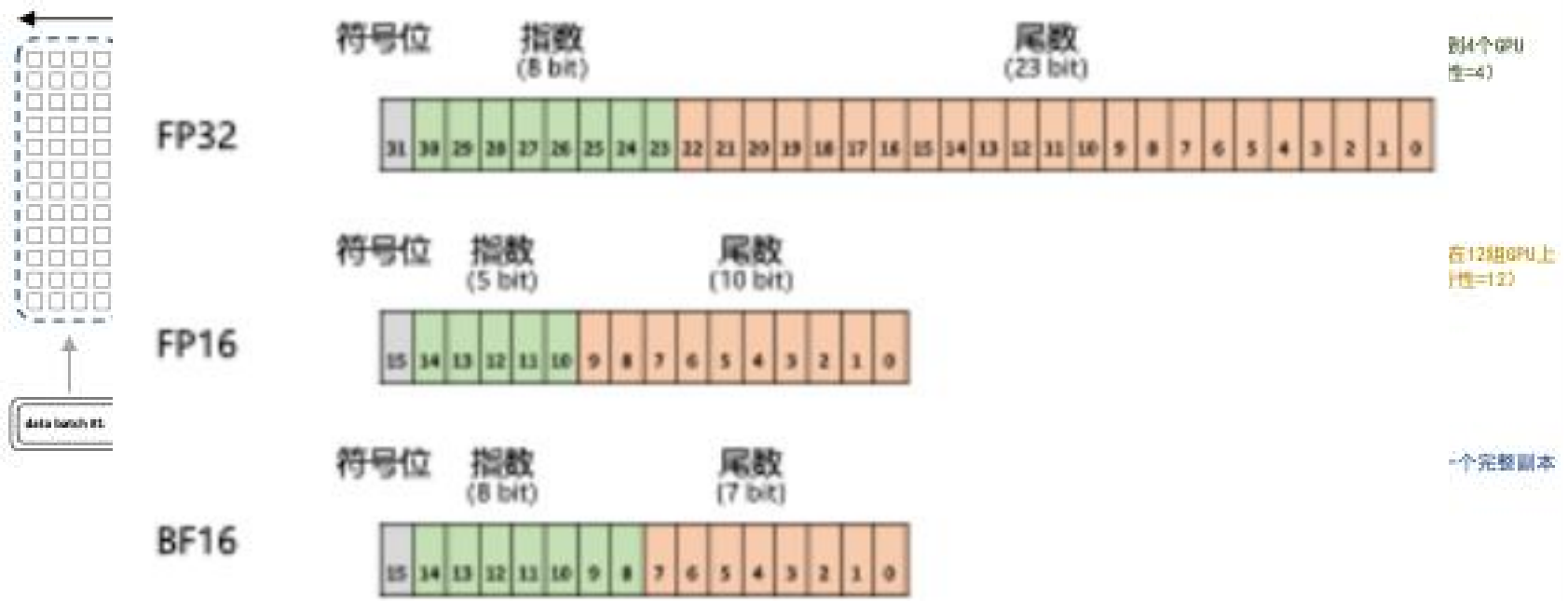
BLOOM 使用Megatron-DeepSpeed[108] 框架进行训练，主要包含两个部分：Megatron-LM提供张量并行能力和数据加载原语；DeepSpeed[138] 提供ZeRO 优化器、模型流水线及常规的分布式训练组件。通过这种方式可以实现数据、张量和流水线三维并行。



25

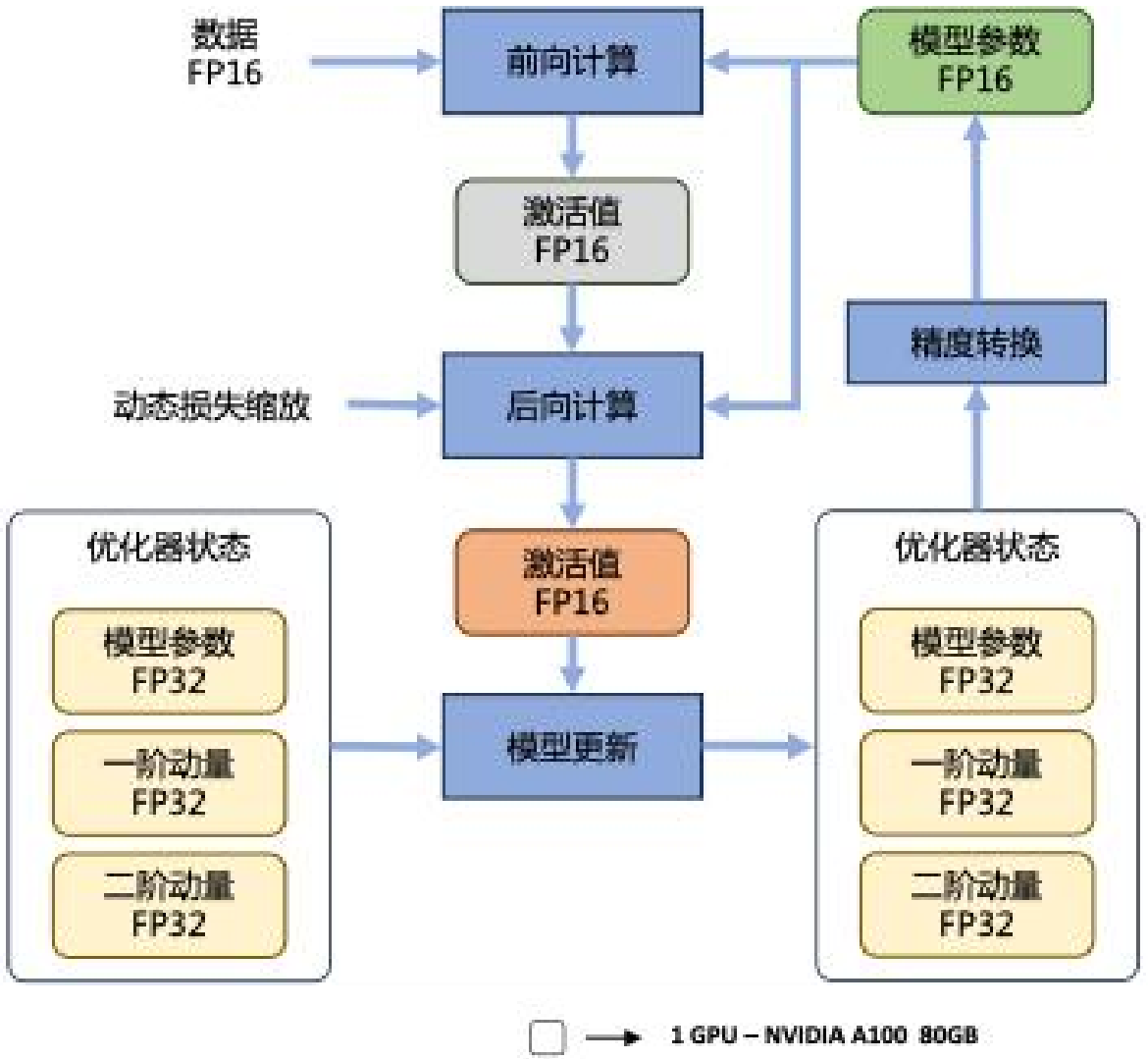
内存优化

当前，大语言模型训练通常采用Adam优化算法，除了需要每个参数梯度，还需要一阶动量（Momentum）和二阶动量（Variance）。虽然Adam 优化算法相较SGD算法效果更好也更稳定，但是对计算设备内存的占用显著增大。为了降低内存占用，大多数系统采用了混合精度训练（Mixed Precision Training）方式，即同时存在FP16（16 位浮点数）或者BF16（Bfloat16）和FP32（32 位浮点数）两种格式的数值。



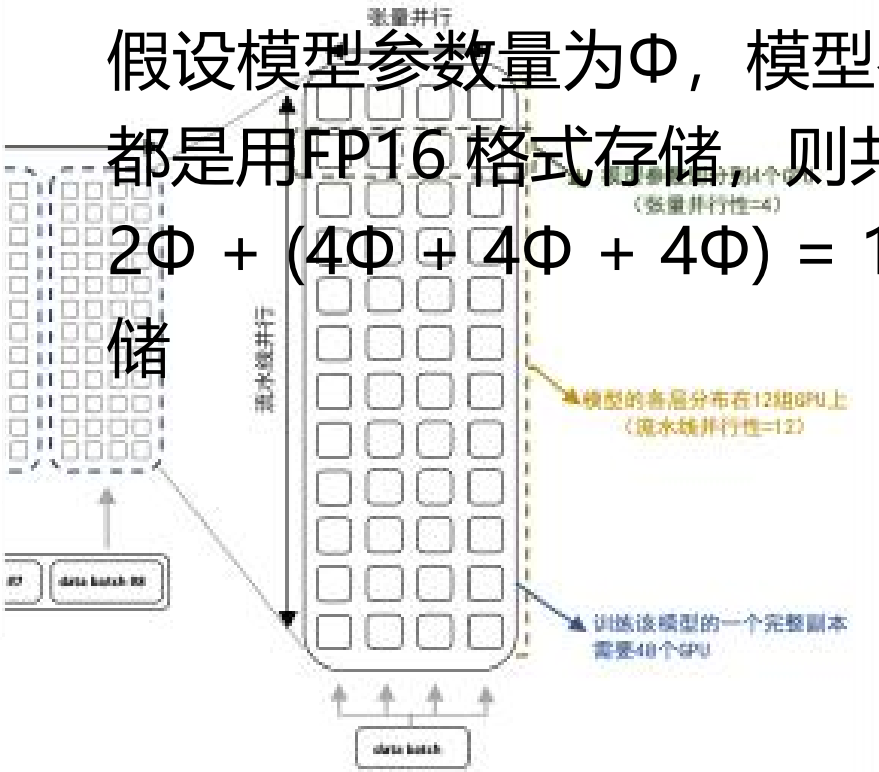
26

内存优化



混合精度优化的过程如图4.16 所示。Adam优化器状态包括模型参数备份、一阶动量和二阶动量都采用FP32保存式存储。

假设模型参数量为Φ，模型参数和梯度都是用FP16 格式存储，则共需要2Φ + 2Φ + (4Φ + 4Φ + 4Φ) = 16Φ 字节存储



分布式训练的集群架构

分布式训练需要使用由多台服务器组成的计算集群（Computing Cluster）完成。而集群的架构也需要根据分布式系统、大语言模型结构、优化算法等综合因素进行设计。分布式训练集群属于高性能计算集群（High Performance Computing Cluster，HPC），其目标是提供海量的计算能力。在由高速网络组成的高性能计算上构建分布式训练系统，主要有两种常见架构：

- 参数服务器架构
- 去中心化架构

DeepSeek实践

2025

谢谢大家

时间：202X.X