arXiv:2601.02311v1 [cs.DC] 5 Jan 2026

分布式深度学习的放置语义：分析并行策略的系统化框架

Deep Pankajbhai
Mehta Adobe Inc.

摘要

训练大语言模型需要在众多加速器之间分配计算，但由于缺乏能够预测其行为的统一系统化框架，实践者只能通过反复试错来选择并行策略（数据并行、张量并行、流水线并行、ZeRO）。我们提出放置语义：通过五种模式（复制、分片、分片并聚集、物化、卸载）描述每种策略如何在设备间放置四种训练状态（参数、优化器、梯度、激活值）。仅根据放置方式，无需了解实现细节，我们就能推导出内存消耗和通信量。我们的预测与已发表的结果完全一致：正如原始论文所报告，ZeRO-3 比数据并行少用 $8\times$ 内存，通信成本为 $1.5\times$ 。我们证明了两个条件（梯度完整性、状态一致性）是分布式训练获得与单设备结果相同效果的必要且充分条件，并提供了安全组合不同策略的组合规则。该框架将 ZeRO 第1至3阶段、全分片数据并行（FSDP）、张量并行和流水线并行统一为具有不同放置选择的实例。

1 引言

我们旨在弥合分布式深度学习实践与理论之间的差距。训练一个拥有 700 亿个参数的模型，仅模型状态就需要约 1120 GB 内存 [23], ，远超当前 GPU 的 80 GB 容量。实践者必须运用并行策略将这些状态分布到各个设备上：数据并行（DP） [14], ZeRO/全分片数据并行（FSDP）[19, 25], 张量并行（TP） [22], 流水线并行（PP） [9], 以及专家并行 [6]。

每种策略都通过其实现方式来描述：通信操作、数据结构布局和运行时优化。这种以实现为中心的视角，使得回答一些基本问题变得困难：

- ZeRO 第 2 阶段与第三阶段究竟有何区别?
- 给定新的配置，每台设备将使用多少内存?
- 何时可以安全地将张量并行与流水线并行结合起来?
- 分布式训练要与单设备结果一致，必须满足哪些性质?

我们的框架能够精确回答这些问题。例如，我们指出，ZeRO 第 2 阶段和第三阶段恰好只在一个放置选择上有所不同（参数：复制 vs. 分片并聚集）。仅凭这一差异，我们便推导出，第三阶段可将每台设备的内存占用从 1120 GB 降至 140 GB，减少 $8\times$，同时使通信量增加 $1.5\times$。这些预测与 ZeRO 原论文完全一致。

1.1 我们的贡献

我们提出了放置语义，这是一种能够回答这些问题的系统化框架。该框架基于三个理念：

训练状态是基本单位。 我们确定了每种训练配置都需要管理的四种状态：参数 Θ、优化器状态 Ω、梯度 G 和激活值 A。这些是分布式训练的基本对象。

放置是具体规范。 对于每种状态，我们定义其放置方式：哪些设备持有哪些部分。我们以精确的语义形式化定义了五种放置模式：复制（R）、分片（S）、分片并聚集（S^*）、实体化（M）和卸载（O）。这五种模式之所以出现，是因为分片有两种变体：纯分片，即每个设备仅使用其本地分片；以及分片并聚集，即为进行计算而暂时重新组装各个分片。这一区分至关重要：它将 ZeRO 第 2 阶段（梯度的纯分片）与第三阶段（参数的分片并聚集）区分开来。我们将范围限定为这五种模式，因为它们涵盖了当前实践中的所有策略；中间模式（例如针对 $1 < k < N$ 的 k 路复制）可以直接扩展得到。

成本源于放置。 给定放置规范，我们通过形式化规则推导内存和通信。这是我们的关键技术成果：资源预测无需依赖实现细节。

1.2 新内容

以往工作描述具体系统，而我们提出了系统性的基础，使跨系统推理成为可能：

1. 系统化的放置语义，并对模式作出精确定义（第3节） 2. 根据规格计算内存和通信的推导规则（第4节） 3. 正确性条件及其必要性与充分性的证明（第5节）

表 1：使用 Adam 优化器进行混合精度训练时，训练 700亿参数模型的内存需求。根据 ZeRO 论文中的计算方式 [19], ，我们计入 FP32 主权重。为便于推导，我们将主权重与优化器状态归为一组，得到 $|\Omega| = 12P$ 字节的总量。

状态	数量	精度	内存
参数 Θ	P	FP16	140 GB
主权重	P	FP32	280 GB
优化器 Ω (Adam m, v)	$2P$	FP32	560 GB
梯度 G	P	FP16	140 GB
模型状态总量			**1120 GB**

4. 用于组合策略的组合演算（第6节）

以往的工作描述了各种系统。我们提供了一个系统化框架，在该框架中，这些系统都是具体实例。这种关系类似于计算复杂性理论与具体算法之间的关系：计算复杂性理论提供了分析任意算法的工具，而算法论文则描述具体的解决方案。

验证。 我们使用 ZeRO 论文 [19] 中公布的结果来验证我们的框架。我们的推导规则预测出的内存缩减（$8\times$）和通信开销（$1.5\times$）与原作者报告的结果一致，从而证实放置规范能够反映真实的系统行为（第7节）。

2 背景

我们先确立符号约定，并回顾训练过程中有哪些内容会占用内存。我们采用标准术语：FP16 和 FP32 分别表示 16 位和 32 位浮点格式；SGD 表示随机梯度下降；NVMe 表示非易失性内存高速存储。

2.1 训练状态

训练步骤使用一个数据批次，将参数从 Θ_t 变换为 Θ_{t+1} 。这需要维护四个状态张量。

参数 $\Theta \in \mathbb{R}^P$ 即模型权重。对于一个包含 L 层、隐藏维度为 H 的 Transformer，参数量约为 $P \approx 12LH^2$ [10].[1] 每个注意力层包含 $4H^2$ 个参数（查询、键、值和输出投影），每个前馈层包含 $8H^2$ 个参数（两个扩展倍数为 $4\times$ 的矩阵）。

[1]对于较大的 H ，这一近似成立，但未计入嵌入参数；嵌入参数约增加 $V \cdot H$ ，其中 V 为词汇表大小。对于具有典型词汇表的 700亿参数模型，嵌入参数约占总参数量的 1–2%。

优化器状态Ω 包含优化器维护的辅助值。Adam [11] 存储一阶矩 $m \in \mathbb{R}^P$ 和二阶矩 $v \in \mathbb{R}^P$，因此$|\Omega| = 2P$。为确保数值稳定性，这些值以 FP32 格式存储 [16]。

梯度 $G \in \mathbb{R}^P$ 是在反向传播过程中 $\nabla_\Theta \mathcal{L}$ 计算得到的导数。

激活值 A 是前向传播过程中产生的、梯度计算所需的中间值。其大小取决于批次大小 B、序列长度 S以及架构细节。

2.2 内存核算

表1 根据 ZeRO 论文的混合精度内存核算方式，展示了具体的内存需求 [19]。关键观察是，优化器状态占据主导地位：Adam 需要 $2P$ 个 FP32 值，即 $8P$ 字节，而 FP16 参数需要 $2P$ 字节。计入 FP32 主权重（混合精度训练稳定性所必需）后，每个参数总计需要 16 字节。

备注1（内存核算约定）。在本文中，我们采用 *ZeRO* 论文的约定，即每个参数占用 *16* 字节：*2* 字节（*FP16* 参数） $+$ *2* 字节（*FP16* 梯度） $+$ *4* 字节（*FP32* 主权重） $+$ *8* 字节（*FP32* 优化器状态）。当我们写作 $|\Theta|$, $|\Omega|$, $|G|$ 时，指的是以字节为单位的内存占用，而不是参数量。具体而言： $|\Theta| = 2P$ 字节、$|G| = 2P$字节，以及 $|\Omega| = 12P$ 字节（主权重 $+$ *Adam* 状态）。

2.3 通信原语

分布式训练使用集合通信操作。我们采用标准成本模型 [21]。对于 N 台设备和张量大小 $|T|$:

全归约 会在各设备间聚合（求和） T，并将结果分发给所有设备。使用环形算法时，每台设备发送和接收 $2 \cdot \frac{N-1}{N} \cdot |T|$字节。

归约散播 会聚合 T 并分发互不重叠的分片。设备 i接收总和的分片 i。成本：每台设备 $\frac{N-1}{N} \cdot |T|$ 字节。

全聚集 会收集分片，并将完整张量分发给所有设备。成本：每台设备 $\frac{N-1}{N} \cdot |T|$ 字节。

3 放置语义

现在我们介绍系统化框架。我们先进行直观理解，然后给出精确定义。

3.1 直观理解

考虑在 $N = 8$ 设备上进行训练。对于参数 Θ，我们有以下选择：

- 每个设备都存储一份完整副本（数据并行）

- 每个设备存储 $1/8$ 的参数（ZeRO 第三阶段）
- 没有设备会持久存储参数；需要时再聚集它们（采用激进分片的 FSDP）

每种选择对内存和通信都有不同影响。我们将这些选择形式化为放置模式。

3.2 放置模式

定义 1 (放置模式**)**。设 X 为一个大小为 $|X|$ 、分布在 N 个设备上的状态张量，这些设备的索引为 $\{0,\ldots,N-1\}$。放置模式 π 指定每个设备 i 持久存储 X 的哪一部分。我们定义了五种模式：

复制（R）：每个设备都存储完整的张量。

$$\pi_R(X,i) = X \quad \textit{for all } i \in \{0,\ldots,N-1\} \tag{1}$$

分片（S）：张量被划分为 N 个连续分片；设备 i 存储分片 i。

$$\pi_S(X,i) = X\left[\frac{i\cdot|X|}{N} : \frac{(i+1)\cdot|X|}{N}\right] \tag{2}$$

分片并聚集（S^*）：与 S 类似，但在每次使用之前，都会通过全聚集重建完整张量，使用后再丢弃非本地部分。这体现了 *ZeRO-3/FSDP* 的参数处理方式。

$$\pi_{S^*}(X,i) = X\left[\frac{i\cdot|X|}{N} : \frac{(i+1)\cdot|X|}{N}\right] \ \textit{(persistent)}, \quad X \ \textit{(transient during use)} \tag{3}$$

物化（M）：没有任何设备会持久存储 X 。需要 X 时，会根据其他状态重建，使用后再丢弃。此方式适用于可重新计算的中间值，例如激活值。

$$\pi_M(X,i) = \emptyset \quad \textit{(persistent storage)} \tag{4}$$

卸载（O）：张量存储在 *CPU* 内存或 *NVMe* 中，需要时再传输到 *GPU*。

$$\pi_O(X,i) = \emptyset \quad \textit{(GPU memory)} \tag{5}$$

示例 1（数据并行）。在数据并行（*DP*）中，所有模型状态都会被复制：$\pi_\Theta = R$、$\pi_\Omega = R$、$\pi_G = R$。每个设备都持有完整参数和完整优化器状态，并计算完整梯度。在本地完成梯度计算后，通过全归约在各设备间同步梯度。

示例 2（ZeRO 第三阶段）。*ZeRO* 第三阶段将所有内容分片：$\pi_\Theta = S^*$, $\pi_\Omega = S$, $\pi_G = S$。参数分片分布在各设备上；在每一层进行计算之前，通过全聚集重建完整参数，使用后即丢弃。优化器状态和梯度始终保持分片状态。

表2：常见并行策略的放置规范。对于张量并行和流水线并行，放置按层或阶段进行。 S^* 表示计算前分片并聚集。DP =数据并行，TP = 张量并行，PP = 流水线并行。

策略	π_Θ	π_Ω	π_G	π_A
数据并行（DP）	R	R	R	R
ZeRO 第 1 阶段	R	S	R	R
ZeRO 第 2 阶段	R	S	S	R
ZeRO 第三阶段 / FSDP	S^*	S	S	R
ZeRO 卸载	O	O	S	R
张量并行（TP，层内）	S	S	S	S
流水线并行（PP，层间）	S	S	S	R

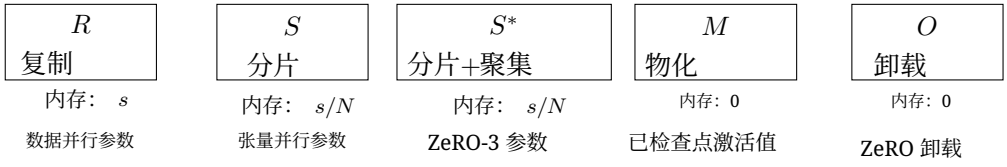

图1：五种放置模式。上：每个设备的 GPU 内存成本（s =张量大小， N = 设备数量）。下：常见策略中的示例用法。S^* （分片并聚集）是 ZeRO-3/FSDP 的关键创新：参数以分片形式存储，但会在计算时临时聚集。

3.3 放置规范

定义 2 （放置规范）。 放置规范是一个元组$\Pi = (\pi_\Theta, \pi_\Omega, \pi_G, \pi_A)$，其中每个 $\pi_X \in \{R, S, S^*, M, O\}$ 指定状态 X的放置模式。

并行策略由其放置规范完全确定。表 2 列出了已知策略的规范，图 1展示了五种模式及其内存成本。

备注 2。 物化模式（M）未出现在表 *2* 中，因为常见策略会持久存储所有状态。不过， M 可用于建模激活检查点，即重新计算激活值，而不是存储它们。

4 推导规则

现在，我们提出主要的技术贡献：一套从放置规范推导内存和通信的规则。

4.1 预备知识

定义 3（重构单元）。设 s_{unit} 表示在分片并聚集操作期间可独立重构的最小单元的大小。对于 *Transformer* 模型，这通常对应于一层：对于隐藏维度为 H 的标准 *Transformer* 层，每个参数占用 $s_{unit} = 12H^2 \cdot$ 字节。s_{unit} 的选择是实现层面的决策，需要在内存占用与通信粒度之间进行权衡。

4.2 内存推导

定理 1（由放置方式决定的内存）。设 $\Pi = (\pi_\Theta, \pi_\Omega, \pi_G, \pi_A)$ 为针对 N 个设备的放置规范。每个设备的 *GPU* 内存为：

$$M(\Pi) = \mu(\pi_\Theta, |\Theta|) + \mu(\pi_\Omega, |\Omega|) + \mu(\pi_G, |G|) + \mu(\pi_A, |A|) \tag{6}$$

其中 μ： $\{R, S, S^*, M, O\} \times \mathbb{R}^+ \to \mathbb{R}^+$ 定义如下：

$$\mu(R, s) = s \tag{7}$$
$$\mu(S, s) = s/N \tag{8}$$
$$\mu(S^*, s) = s/N + s_{unit} \tag{9}$$
$$\mu(M, s) = s_{unit} \tag{10}$$
$$\mu(O, s) = 0 \tag{11}$$

其中 s_{unit} 在定义 3中定义。

证明。我们根据定义逐一证明每种情况 1.

情况 R**：** 根据公式 (1)，对于所有 i，$\pi_R(X, i) = X$。每个设备都存储完整张量，因此每设备内存为 $|X| = s$。

情况 S**：** 根据公式 (2)，设备 i 存储 $X[i|X|/N : (i+1)|X|/N]$，其大小为 $|X|/N = s/N$。

情况 S^***：** 根据公式 (3)，设备 i 持久存储分片 i（大小为 s/N）。在计算期间，完整张量会被临时聚集。如果一次聚集一个单位（这是标准做法），则峰值内存为 $s/N + s_{\text{unit}}$。注意：将聚集与计算重叠的流水线实现可能需要 $2 \cdot s_{\text{unit}}$ 临时内存。

情况 M**：** 根据公式 (4)，持久存储为 $\emptyset$。然而，在计算期间，必须重构张量。如果一次重构一个单位（例如，一层），则峰值临时内存为 s_{unit}。

案例 O**：** 根据公式（5），GPU 内存为 $\emptyset$。该张量位于 CPU/NVMe 中，对 GPU 内存的贡献为 0。 □

示例3（内存计算）。对于一个 *700*亿参数模型（$P = 70 \times 10^9$），使用$N = 8$ 个设备，并按照每个参数*16*字节计算（见备注*1*）：

数据并行（R，R，R，R）：

$$\begin{aligned} M_{DP} &= \mu(R, |\Theta|) + \mu(R, |\Omega|) + \mu(R, |G|) + \mu(R, |A|/N) \\ &= 2P + 12P + 2P + |A|/8 = 16P + |A|/8 \end{aligned}$$

以字节计：每台设备 $16 \times 70 \times 10^9 = 1120\,GB$（不包括激活值）。

ZeRO 第三阶段 $(S^*,\ S,\ S,\ R)$：

$$\begin{aligned} M_{Z3} &= \mu(S^*, 2P) + \mu(S, 12P) + \mu(S, 2P) + \mu(R, |A|/N) \\ &= 2P/N + 12P/N + 2P/N + |A|/8 = 16P/N + |A|/8 \end{aligned}$$

$N = 8$ 的字节数：每台设备 $16P/8 = 2P = 140\,GB$，属于一种 $8\times$ 归约。

4.3 通信推导

定理 2 (由布局决定的通信)。设 Π 是一种放置规范。每个训练步骤的通信量由前向-反向-更新周期所需的状态转换决定：

1. 如果在本地计算 $\pi_G = R$ 和梯度，同步需要全归约：

$$C^R_{sync} = 2 \cdot \frac{N-1}{N} \cdot |G| \tag{12}$$

2. 如果 $\pi_G = S$，同步使用归约散播：

$$C^S_{sync} = \frac{N-1}{N} \cdot |G| \tag{13}$$

3. 如果 $\pi_\Theta = S^*$，则必须在使用前聚集参数。对于前向和反向传播：

$$C_{gather} = 2 \cdot \frac{N-1}{N} \cdot |\Theta| \tag{14}$$

证明。 **第 1 部分：** 对于 $\pi_G = R$，每个设备根据其数据分片计算局部梯度 G_i 。为确保正确性（定理 3），最终梯度必须是 $G = \frac{1}{N}\sum_i G_i$。全归约计算该总和，并将其分发到所有设备。环形全归约算法要求每个设备在散播阶段发送（$N-1$）$/N \cdot |G|$ 字节，并在聚集阶段发送（$N-1$）$/N \cdot |G|$ 字节，总计 2（$N-1$）$/N \cdot |G|$。

第 2 部分： 对于 $\pi_G = S$，我们需要总和，但每个设备只需要其分片。归约散播计算总和，并将分片 i分发到设备 i。这只需要环形全归约的散播阶段：（$N-1$）$/N \cdot |G|$ 字节。

第 3 部分： 采用 $\pi_\Theta = S^*$ 时，参数以分片形式存储，但计算时必须完整。每层前向传播之前，全聚集会从分片中重构参数。反向传播时也需要进行相同的重构，以计算梯度。每次全聚集的开销为 $(N-1)/N \cdot |\Theta|$，需要进行两次，因此总计为 $2(N-1)/N \cdot |\Theta|$。 □

示例 4 （通信量计算）。对于 $P = 70 \times 10^9$ 个参数和 $N = 8$ 个设备（梯度采用 *FP16* 格式，因此每个占用 $|G| = 2P$ 字节）：

数据并行：仅进行梯度同步。

$$C_{DP} = 2 \cdot \frac{7}{8} \cdot 2P = 3.5P \approx 245\ GB\ per\ device$$

ZeRO 第三阶段：梯度同步（归约散播） $+$ 参数聚集（$2\times$全聚集）。

$$C_{Z3} = \frac{7}{8} \cdot 2P + 2 \cdot \frac{7}{8} \cdot 2P = 5.25P \approx 368 \ GB \ per \ device$$

ZeRO 第三阶段的通信量***1.5***$\times$ 多于数据并行，但使用 ***8***$\times$更少的内存。

4.4 基本权衡

推论 1 （内存-通信权衡）。对于使用模式$\{R, S, S^*\}$的策略，内存缩减与通信开销之间的关系取决于分片的是哪种状态：

1. 对优化器状态进行分片（$R \to S$）可缩减内存，且不会增加通信量（更新在本地进行）。*2.* 对梯度进行分片（$R \to S$）可缩减内存并减少通信量（归约散播相较于全归约）。*3.* 对参数进行分片（$R \to S^*$）可缩减内存，但会增加 $2 \cdot \frac{N-1}{N} \cdot |\Theta|$ 通信量（每一步执行两次全聚集）。

——

证明。**第一部分：**优化器状态仅在更新步骤中访问。采用分片后，每个设备都使用本地梯度分片更新其本地分片。优化器本身不需要跨设备通信。

第二部分：梯度同步从全归约（开销为 $2 \cdot \frac{N-1}{N} \cdot |G|$）变为归约散播（开销为 $\frac{N-1}{N} \cdot |G|$），减少了 $2\times$。

第三部分：采用 $\pi_\Theta = S^*$ 时，必须在前向和反向传播之前聚集参数。每次全聚集的开销为 $\frac{N-1}{N} \cdot |\Theta|$，每个步骤需要进行两次。 □

这解释了 ZeRO 各阶段的设计：第一阶段对优化器状态进行分片（可减少内存，且无额外通信开销）；第二阶段还对梯度进行分片（进一步减少内存，并带来通信方面的优势）；第三阶段则对参数进行分片（最大限度地减少内存，但会增加通信开销）。

5 正确性条件

我们将形式化说明分布式训练在何种条件下能够产生正确结果。

定义 4 （语义等价）。 如果分布式训练配置产生与单设备训练相同的参数更新序列 $\Theta_0, \Theta_1, \Theta_2, \ldots$（由归约顺序引起的浮点差异除外），则二者在语义上等价。

定理 3（梯度完整性）。 若要实现语义等价，在步骤 t 用于参数更新的梯度必须等于：

$$G_t = \frac{1}{B}\sum_{j=1}^{B} \nabla_\Theta \mathcal{L}(x_j, \Theta_t) \tag{15}$$

其中 B 是全局批次大小，$\{x_j\}_{j=1}^{B}$ 是训练样本。

证明。 单设备随机梯度下降严格按照公式 (16) 计算，并更新 $\Theta_{t+1} = \Theta_t - \eta \cdot$ 优化器(G_t, Ω_t)。如果分布式配置产生了不同的梯度 $G'_t \neq G_t$，那么 $\Theta'_{t+1} \neq \Theta_{t+1}$，这违反了语义等价性。

反之，如果 G_t 满足公式 (16)，则该更新与单设备训练一致。 □

梯度完整性违规：

- 样本缺失：设备未能贡献其梯度
- 样本重复：同一样本由多个设备处理
- 归一化错误：除以本地批次大小，而不是全局批次大小

定理 4 (状态一致性**)**。 为了实现语义等价，每当访问或通信状态张量时，所有参与设备都必须持有按位完全相同的值（允许浮点结合性造成的差异），并使用相同的数据类型。

证明。 假设各设备持有的值不一致。以参数为例：如果设备 0 上的参数为 $\Theta^{(0)}$，而设备 1 上的参数为 $\Theta^{(1)} \neq \Theta^{(0)}$，那么它们会针对相同的输入计算出不同的梯度，从而破坏梯度完整性。

对于数据类型：如果设备 0 使用 FP32 进行归约，而设备 1 使用 FP16 进行归约，舍入结果就会不同，从而产生不一致的结果。 □

状态一致性违规：

- 过时参数：更新后，设备使用了过期副本
- 类型不匹配：不同设备使用不同精度
- 归约顺序依赖：未进行适当处理的非确定性归约

定理 5（必要性与充分性）。 在以下假设下，梯度完整性和状态一致性对于语义等价而言是联合必要且充分的：

1. 确定性操作：给定相同的输入，所有算术运算都会产生相同的结果。 *2.* 一致初始化：所有设备均以相同的 Θ_0 和 Ω_0 开始。

3. 同步执行：所有设备完成当前步骤后，才会有设备开始下一步。

证明。**必要性：**定理3 和4已证明这一点。违反任一条件都会导致参数轨迹偏离单设备训练。

充分性：我们通过对训练步数进行归纳法来证明。

基础情形：根据假设 (2)，在 $t = 0$ 时，所有设备都具有相同的 Θ_0 和 Ω_0 。状态一致性成立。

归纳步骤：假设在步骤 t时，所有设备的 Θ_t 和 Ω_t一致。我们证明步骤 $t+1$ 会产生一致的 Θ_{t+1}。

1. 每个设备 i 都在大小为$b = B/N$的数据分片上计算局部梯度 $G_t^{(i)}$ 。局部梯度是局部样本的均值： $G_t^{(i)} = \frac{1}{b}\sum_{j\in\text{分片}_i} \nabla_\Theta \mathcal{L}(x_j, \Theta_t)$。根据假设 (1)，相同的参数和输入会产生相同的梯度。

2. 通过全归约进行梯度同步，计算得到 $\sum_i G_t^{(i)}$ 。由于每个局部梯度已经是 b 局部样本的均值，且有 N个设备各自贡献这样的均值，因此将全归约的求和结果除以 N，即可得到正确的全局平均梯度 $G_t = \frac{1}{B}\sum_{j=1}^{B} \nabla_\Theta \mathcal{L}(x_j, \Theta_t)$。根据假设 (3)，所有设备都会在继续执行之前完成此操作。

3. 根据梯度完整性，同步结果等于公式 (16)。

4. 根据状态一致性，所有设备都看到相同的 G_t, Θ_t, Ω_t。

5. 根据假设 (1)，优化器更新 $\Theta_{t+1} = f(\Theta_t, G_t, \Omega_t)$ 在所有设备上产生相同的结果。

因此， Θ_{t+1} 保持一致，并与单设备训练相匹配。 □

备注 3. 非确定性 *GPU* 操作（例如归约操作中的原子操作）可能会违反假设 *(1)*。框架提供了满足此假设的确定性模式，但会带来一定的性能损失。

6 组合演算

大规模训练结合了多种策略。我们将有效的组合形式化。例如，节点内的张量并行可以与节点间的数据并行相结合：张量并行负责层内分布，而数据并行负责梯度平均。

定义 5 (组合)。 设 Π_1 和 Π_2 是针对设备组 D_1 和 D_2的放置规范。组合 $\Pi_1 \otimes \Pi_2$ 在 D_1 的每个子集内应用 Π_1 ，并在各子集之间应用 Π_2 。

定理 6 (张量-数据组合)。 当满足以下条件时，张量并行（度数为 T）可与数据并行（度数为 D）在 $N = T \times D$ 个设备上组合：

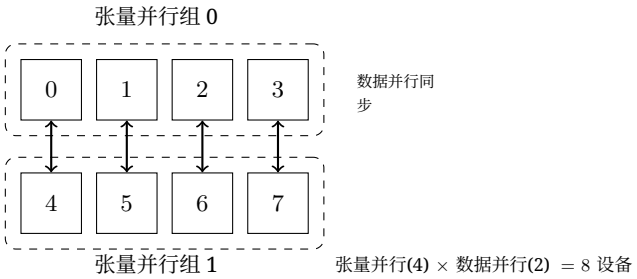

图 2：张量并行（行内）与数据并行（跨行）的组合。张量并行通信发生在虚线框内；数据并行通信沿垂直箭头进行。

1. 张量并行组由设备 $\{iT, iT+1, \ldots, (i+1)\,T-1\}$ 组成，适用于 $i \in \{0, \ldots, D-1\}$*2.* 张量并行通信（逐层）在数据并行梯度同步之前完成 *3.* 数据并行梯度同步在张量并行组之间聚合，而非组内

证明。 我们验证梯度完整性和状态一致性。

梯度完整性： 在每个张量并行组内，设备持有不同的参数分片，但处理相同的数据并计算部分梯度。张量并行通信（对激活值执行全归约）确保前向和反向计算正确。在不同的张量并行组之间，处理不同的数据分片。数据并行梯度同步（跨组执行全归约）对梯度求平均，满足公式（16）。

状态一致性： 张量并行组通过同步更新来维持分片参数的一致性。数据并行组通过梯度同步来维持复制状态的一致性。这种分离方式（组内采用张量并行、组间采用数据并行）确保不会发生冲突。 □

定理 7（流水线与数据并行组合）。 当满足以下条件时，流水线并行（K 个阶段）可在 $N = K \times D$ 台设备上与数据并行（D 个副本）组合：

1. 每个阶段 k 在设备 $\{kD, kD+1, \ldots,$ 上复制 D 次（$k+1$）$D-1\}$*2.* 梯度同步按阶段进行：仅在同一阶段的副本之间执行全归约 *3.* 激活值传输发生在同一流水线中对应的阶段之间

证明。 **梯度完整性：** 每个阶段 k 都有参数 $\Theta^{(k)}$。阶段 k 的副本处理不同的数据分片，并计算局部梯度。每个阶段执行全归约以对这些梯度求平均，从而满足 $\Theta^{(k)}$ 的梯度完整性。

算法 1 基于放置语义的示例性策略选择 要求：模型大小 P、设备内存 M_d、设备数量 N、互连类型 确保：放置规范 Π1： $M_{\text{model}} \leftarrow 16P$ {参数 + 优化器 + 梯度（单位：字节）}2：如果 $M_{\text{model}} < 0.7 \cdot M_d$ 则 3：返回 (R, R, R, R) { 数据并行 }4：结束如果 5：如果 $M_{\text{model}}/N < 0.7 \cdot M_d$ 则 6：返回 (S^*, S, S, R) {ZeRO-3 / FSDP}7：结束如果 8：如果单层 $> 0.3 \cdot M_d$ 且互连速度快，则 9：在节点内添加张量并行 10：结束如果 11：返回组合规范

状态一致性： 在流水线内部，各阶段持有互不重叠的参数。在数据并行组内部，各副本持有相同的参数（通过全归约进行同步）。无需任何设备协调相互冲突的放置。

□

6.1 无效组合

命题 1 (慢速互连上的张量并行)。 跨设备的张量并行在互连延迟为 α 时，会产生每步延迟开销 $O(L \cdot \alpha)$ 其中 L 为层数。

证明。 对于 TP，每个层至少需要一次同步集合通信（全归约或全聚集）。对于 L 层，这会向关键路径增加 L 个延迟项。如果 α 很大（例如，跨节点以太网相较于节点内 NVLink），其影响将超过计算时间。

□

这解释了为什么在实践中 TP 仅限于节点内通信。

备注 4 （三向组合）。 生产系统通常采用 *TP*$\otimes$ *PP* $\otimes$ *DP*（三维并行）。当满足以下条件时，这种组合是有效的：*(1) TP* 位于最内层（节点内），*(2) PP* 位于中间层（流水线内的节点间），*(3) DP* 位于最外层（跨流水线副本）。其正确性可通过分层应用定理*6* 和 *7*得出。

7 应用：策略选择

我们将展示该框架如何指导实际决策。

注意：算法中的阈值（0.7、0.3）1 仅为示例性启发式方法，旨在为激活值和运行时分配留出余量。实践者应根据测得的激活值大小和框架开销进行调整。

7.1 与已发表结果的验证

我们使用 ZeRO 论文中已发表的数据验证推导规则 [19]。

内存验证。 ZeRO 论文采用混合精度核算：2 字节（FP16 参数） + 2 字节（FP16 梯度） + 4 字节（FP32 主权重） + 8 字节（FP32 优化器状态） = 16 每个参数的字节数。我们的框架采用相同的核算方式（见表 1 和备注1）。

对于数据并行，ZeRO 报告每个设备 $16P$ 字节。我们的框架：$\mu(R,\ 2P) + \mu(R,\ 2P) + \mu(R,\ 12P) = 16P$字节，**完全匹配**。

对于使用 N 设备的 ZeRO 第三阶段，ZeRO 论文报告每个设备 $16P/N$ 字节。我们的框架预测： $16P/N$ 字节（加上聚合的参数所需的临时内存），**再次匹配**。

通信验证。 ZeRO 论文指出，ZeRO 第三阶段所需的通信量是数据并行的 1.5× 倍。我们的框架计算结果为：

- 数据并行： $C_{\text{数据并行}} = 2 \cdot \underbrace{\frac{N-1}{N} \cdot |G|}_{} \approx 2|G|$ 适用于大型 N
- ZeRO 第三阶段： $C_{\text{Z3}} = \underbrace{\frac{N-1}{N} \cdot |G| + 2 \cdot \frac{N-1}{N}}_{} \cdot |\Theta| \approx |G| + 2|\Theta| = 3|G|$ （因为$|\Theta| = |G|$ 采用 FP16）

比值 $3|G|/2|G| = 1.5$，**与已发表的 1.5× 开销相符**。

需要指出的是，此验证将解析预测与已发表的解析结果进行比较，而非运行时测量。这种吻合表明，我们的框架采用了与 ZeRO 作者相同的成本模型。使用性能分析工具进行实证验证将有助于增强这些结果，但这超出了我们的理论研究范围。

验证协议： 给定一个配置，按以下步骤验证正确性：

1. **梯度完整性检查：** 在 1 个设备和 N 个设备上运行相同的批次。比较梯度范数： $\|G_1 - G_N\|/\|G_1\| < 10^{-5}$。2. **状态一致性检查：** 每次集合通信后，验证所有设备的校验和是否相同。3. **训练轨迹检查：** 使用相同的随机种子，在 1 个设备和 N 个设备上训练 100 步。最终损失差异应为 $< 10^{-4}$。

8 相关工作

并行系统。 Li 等人 [14]对数据并行进行了系统化，Goyal 等人 [7]则将其扩展至大规模。ZeRO [19] 引入了状态分片，并在 DeepSpeed [20]中实现。FSDP [25]提供了原生支持分片的 PyTorch。Megatron-LM[22]确立了张量并行模式；Korthikanti 等人 [12] 将其扩展至序列并行。GPipe [9] 引入了同步流水线；

表3：我们的贡献与先前工作的比较。

能力	先前工作	本文
描述特定系统	✓	
系统化的放置定义		✓
从规范推导成本		✓
证明正确性条件		✓
带证明的组合演算		✓
涵盖任意新策略		✓

PipeDream [17]探索了异步变体；Narayanan等人 [18] 开发了高效的调度方案。GShard [13] 和 Switch Transformer [6]确立了专家并行。

自动并行。 Alpa [26] 将并行性选择表述为一个优化问题，并针对内存和通信建立成本模型；它侧重于搜索算法，而非语义基础。Galvatron [15]同样利用基于性能分析的成本模型优化并行配置。Unity [24] 通过经形式化验证的图替换，联合优化代数变换与并行化；它使用定理证明器验证各项变换的正确性，而我们的工作则证明整个训练过程满足正确性条件。我们的工作提供了一种声明式框架，通过放置来指定 策略，而非通过搜索发现策略；这两种方法相辅相成。

内存优化。 混合精度训练 [16] 通过采用低精度降低内存占用。激活检查点 [2] 以计算换取内存。FlashAttention [4, 5] 通过重计算优化注意力内存。

扩展研究。 扩展定律 [10, 8]指导容量分配。关于 GPT-3 [1], PaLM [3], 和 LLaMA [23]的训练报告介绍了实用的配置。

与我们工作的区别。 表3总结了主要差异。以往工作描述了特定系统、实证发现或基于搜索的优化。我们提供了一个包含定义、推导规则和证明的系统化框架。ZeRO描述了一种实现；我们提供了语义，在该语义中，ZeRO阶段1、2和3是仅在放置规范上有所不同的实例。这使得人们能够系统地推理单篇系统论文都未曾涉及的性质。

9 局限性

我们的框架假设采用同步训练。异步方法（例如，PipeDream 的权重暂存）会引入陈旧性，因此需要进一步形式化。

我们假设使用同构设备。异构系统（混用不同类型的 GPU）需要逐设备能力建模，而我们当前的框架尚未

涵盖这一点。

推导规则给出了渐近成本。实现常数（内核启动开销、内存分配器行为）会影响实际性能，但不在我们的讨论范围内。我们对通信量进行建模，而非通信时间；通信与计算之间的重叠属于实现优化，未被我们的框架捕获。

我们对内存和通信进行建模，但不涉及计算时间。完整的资源模型需要进行操作级分析。

专家并行（混合专家）需要扩展框架，以处理条件路由，即不同输入会激活不同的参数子集。这一扩展属于未来工作。

序列并行 [12] 符合我们的框架，因为它将 $\pi_A = S$ 用于激活值，并通过激活分片产生相应的通信。上下文并行（环形注意力）需要对注意力算子内部的通信模式进行建模，这一扩展留待未来工作。

激活检查点 [2] 与我们的框架正交：它通过重计算减少 $|A|$，但不会改变激活值的放置模式。对于减少后的 $|A|$，该框架仍可直接适用。

梯度累积（在同步之前处理多个微批次）是一种直接的扩展：通信成本分摊到各个累积步骤中，从而将有效通信量降低到原来的累积步骤数分之一。

10 结论

我们提出了放置语义，这是一种用于分布式训练的系统化框架。该框架定义了五种放置模式，根据规范推导内存和通信开销，证明正确性条件，并提供组合规则。

通过将分布式训练形式化，我们实现了：（1）依据规范精确比较策略；（2）无需实现即可预测资源需求；（3）通过明确的条件验证正确性；（4）以有原则的方式组合策略。

我们希望这一框架能帮助实践者理解现有系统，并帮助研究人员设计具有形式化保证的新型并行策略。

致谢

感谢 PyTorch、DeepSpeed 和 Megatron-LM 的开发者，他们开源的实现为本分析提供了参考。

参考文献

[1] Tom Brown、Benjamin Mann、Nick Ryder 等。语言模型是少样本学习器。载于*NeurIPS*，第 1877–1901 页，2020 年。[2] Tianqi Chen、Bing Xu、Chiyuan Zhang 和 Carlos Guestrin。以次线性内存开销训练深度神经网络。*arXiv:1604.06174*，2016 年。[3] Aakanksha Chowdhery、Sharan Narang、Jacob Devlin 等。PaLM：借助 Pathways 扩展语言建模。*JMLR*，24(240):1–113，2023 年。[4]Tri Dao、Dan Fu、Stefano Ermon、Atri Rudra 和 Christopher R´e。FlashAttention：具备 IO 感知能力的快速且内存高效的精确注意力机制。载于*NeurIPS*，2022 年。[5] Tri Dao。FlashAttention-2：更快的注意力机制，以及更优的并行性和工作划分。载于*ICLR*，2024 年。[6] William Fedus、Barret Zoph 和 Noam Shazeer。Switch Transformers：通过简单高效的稀疏性扩展至万亿参数模型。*JMLR*，23(120):1–39，2022 年。[7] Priya Goyal、Piotr Doll´ar、Ross Girshick 等。准确的大批量随机梯度下降：1 小时内训练 ImageNet。*arXiv:1706.02677*，2017 年。[8] Jordan Hoffmann、Sebastian Borgeaud、Arthur Mensch 等。训练计算最优大型语言模型。载于*NeurIPS*，2022 年。[9] Yanping Huang、Youlong Cheng、Ankur Bapna 等。GPipe：使用流水线并行高效训练巨型神经网络。载于*NeurIPS*，第 103–112 页，2019 年。[10] Jared Kaplan、Sam McCandlish、Tom Henighan 等。神经语言模型的扩展定律。*arXiv:2001.08361*，2020 年。[11] Diederik P. Kingma 和 Jimmy Ba。Adam：一种随机优化方法。载于*ICLR*，2015 年。[12] Vijay Korthikanti、Jared Casper、Sangkug Lym 等。减少大型 Transformer 模型中的激活值重计算。载于*MLSys*，2023 年。[13] Dmitry Lepikhin、HyoukJoong Lee、Yuanzhong Xu 等。GShard：通过条件计算和自动分片扩展巨型模型。载于*ICLR*，2021 年。[14] Shen Li、Yanli Zhao、Rohan Varma 等。PyTorch Distributed：加速数据并行训练的实践经验。*PVLDB*，13(12):3005–3018，2020 年。[15] Xupeng Miao、Yujie Wang、Youhe Jiang 等。Galvatron：使用自动并行在多个 GPU 上高效训练 Transformer。载于*VLDB*，2023 年。

[16] Paulius Micikevicius、Sharan Narang、Jonah Alben 等。混合精度训练。发表于*ICLR*，2018 年。[17] Deepak Narayanan、Aaron Harlap、Amar Phanishayee 等。PipeDream：面向深度神经网络训练的通用流水线并行。发表于*SOSP*，第 1–15 页，2019 年。[18] Deepak Narayanan、Mohammad Shoeybi、Jared Casper 等。使用 Megatron-LM 在 GPU 集群上进行高效的大规模语言模型训练。发表于*SC*，2021 年。[19]Samyam Rajbhandari、Jeff Rasley、Olatunji Ruwase 和 Yuxiong He。ZeRO：面向万亿参数模型训练的内存优化。发表于*SC*，2020 年。[20]Jeff Rasley、Samyam Rajbhandari、Olatunji Ruwase 和 Yuxiong He。DeepSpeed：系统优化使参数超过 1000 亿的深度学习模型训练成为可能。发表于*KDD*，第 3505–3506 页，2020 年。[21] Alexander Sergeev 和 Mike Del Balso。Horovod：在 TensorFlow 中实现快速、简便的分布式深度学习。*arXiv:1802.05799*，2018 年。[22] Mohammad Shoeybi、Mostofa Patwary、Raul Puri 等。Megatron-LM：使用模型并行训练数十亿参数的语言模型。*arXiv:1909.08053*，2019 年。[23] Hugo Touvron、Louis Martin、Kevin Stone 等。Llama 2：开放基础模型与微调聊天模型。*arXiv:2307.09288*，2023 年。[24] Colin Unger、Zhihao Jia、Wei Wu 等。Unity：通过联合优化代数变换与并行化加速深度神经网络训练。发表于*OSDI*，第 267–284 页，2022 年。[25] Yanli Zhao、Andrew Gu、Rohan Varma 等。PyTorch FSDP：扩展全分片数据并行的实践经验。*PVLDB*，16(12)：3848–3860，2023 年。[26] Lianmin Zheng、Zhuohan Li、Hao Zhang 等。Alpa：自动化分布式深度学习的算子间与算子内并行。发表于*OSDI*，第 559–578 页，2022 年。