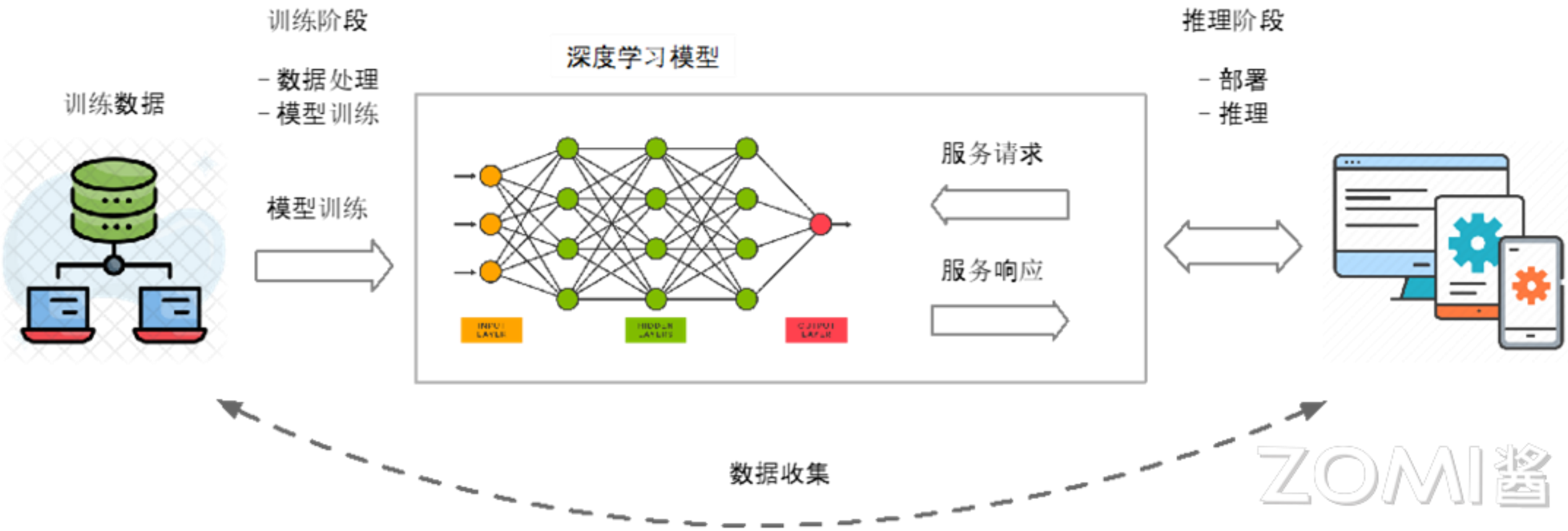
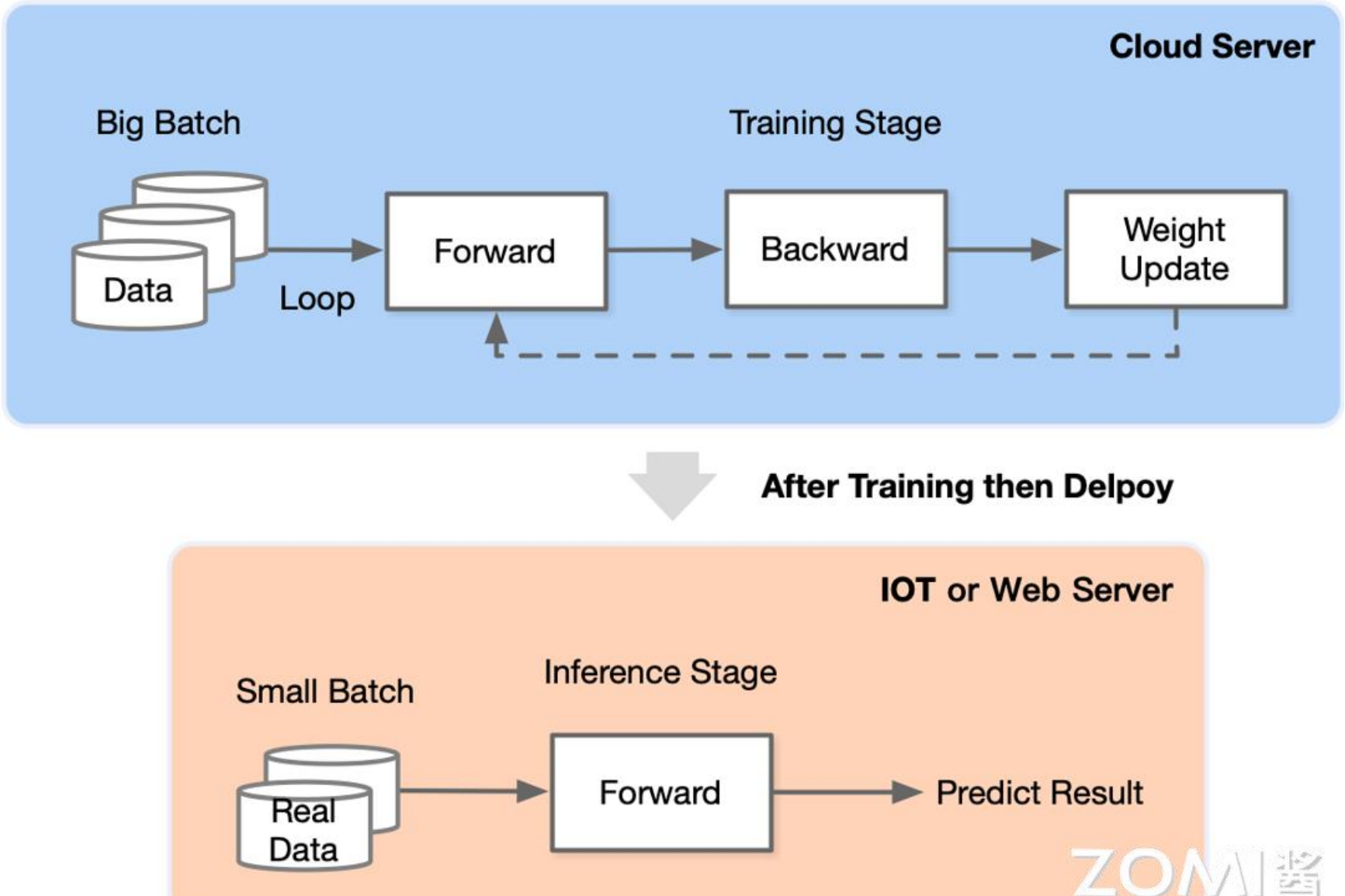
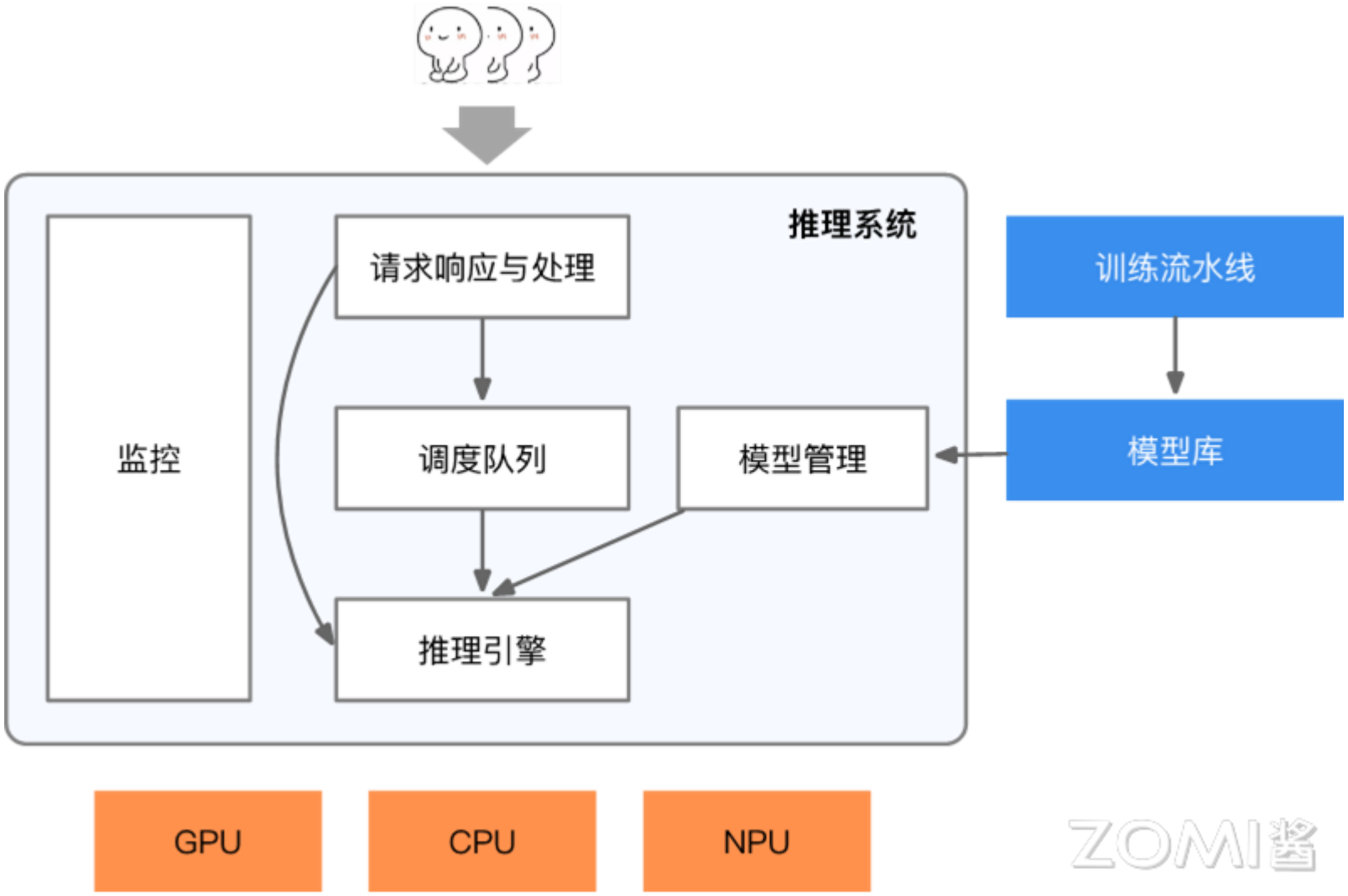
训练

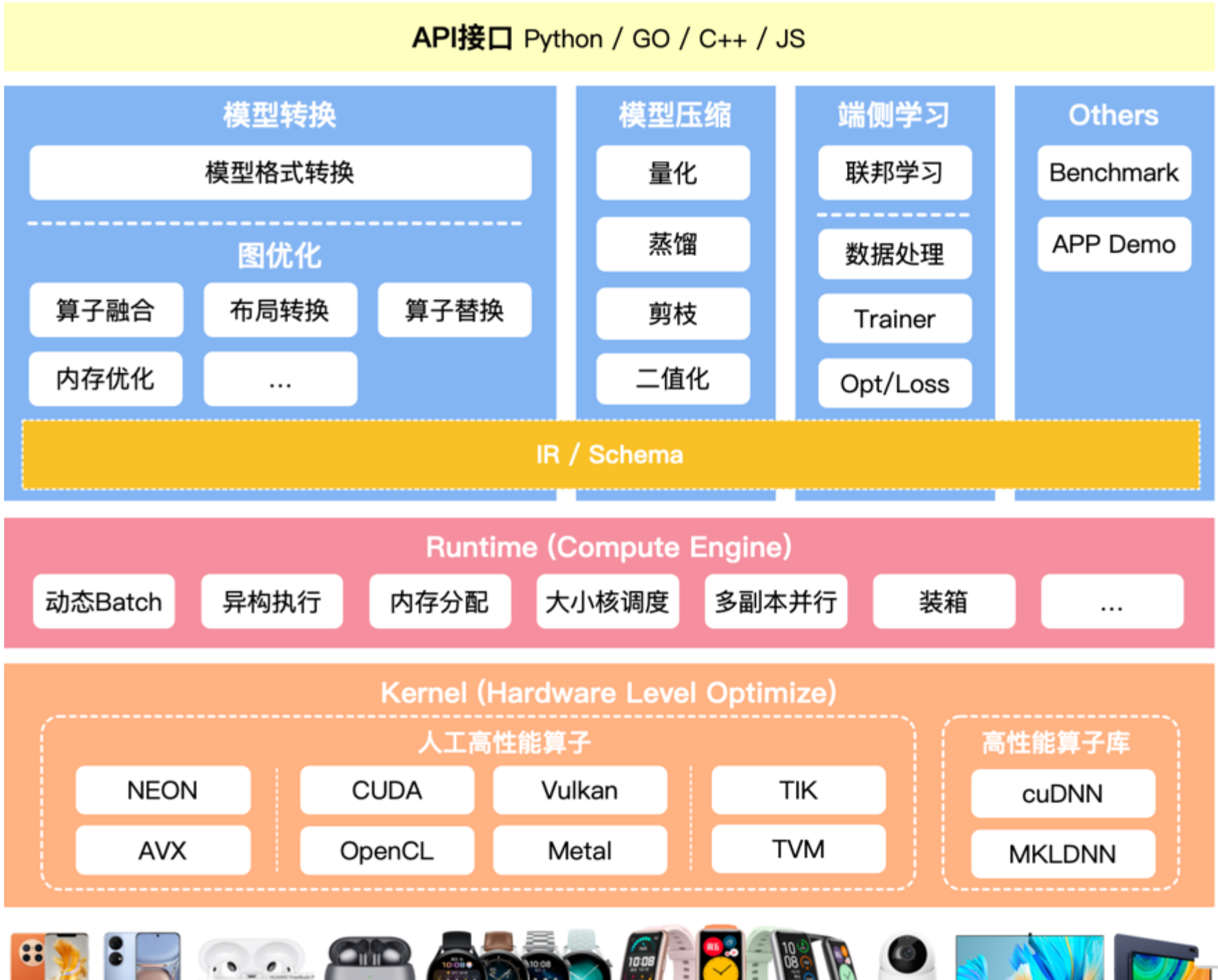


训练与推理


Cloud Server
Big Batch
Training Stage
Data
Loop
Forward
Backward
Weight Update
After Training then Delpoy
IOT or Web Server
Small Batch
Inference Stage
Real Data
Forward
Predict Result

推理系统




API接口 Python / GO / C++ / JS
模型转换
模型格式转换
图优化
算子融合
布局转换
算子替换
内存优化
...
模型压缩
量化
蒸馏
剪枝
二值化
端侧学习
联邦学习
数据处理
Trainer
Opt/Loss
Others
Benchmark
APP Demo
IR / Schema
Runtime (Compute Engine)
动态Batch
异构执行
内存分配
大小核调度
多副本并行
装箱
...
Kernel (Hardware Level Optimize)
人工高性能算子
NEON
AVX
CUDA
OpenCL
Vulkan
Metal
TIK
TVM
高性能算子库
cuDNN
MKLDNN

- **移植**：将模型从开发环境迁移到生产环境，这可能涉及不同的操作系统、硬件平台和软件框架。
- **压缩**：为了在有限的计算资源下运行，模型需要进行压缩。常见的方法包括模型剪枝、量化和蒸馏，这些技术可以减少模型的参数数量和计算复杂度。
- **加速**：使用专用的硬件（如 GPU、TPU）和优化的算法（如图计算库、并行计算）来加速模型的推理过程。
- **监控和维护**：部署后的模型需要持续监控其性能，并进行必要的维护和更新，以确保其在实际应用中始终保持高效和准确。

SGLang + Triton关系

反映了现代AI推理框架的**分层设计理念**。当你在SGLang中选择Triton作为backend时，本质上是在构建一个**分层的推理系统**：SGLang作为**顶层优化器**，**Triton**作为**底层**执行引擎。这种设计模式类似于用PyTorch写模型但底层用CUDA加速。具体来看：

一、为什么需要分层架构?

以SGLang + Triton的典型组合为例：

- **SGLang的定位**：专注**大语言模型推理优化**，特别是针对多轮对话、长上下文等场景的KV缓存复用、动态批处理等专项优化
- **Triton的定位**：提供**硬件级执行环境**，包括显存管理、多模型并发调度、GPU/CPU异构计算等基础设施
- **协作价值**：SGLang无需重复造轮子实现底层调度，可复用Triton的成熟基础设施，集中资源攻克LLM特有的加速难题

二、SGLang如何集成Triton?

- 通过一个具体案例说明（假设部署Llama3-70B模型）：

SGLang如何集成Triton?

- 通过一个具体案例说明（假设部署Llama3-70B模型）

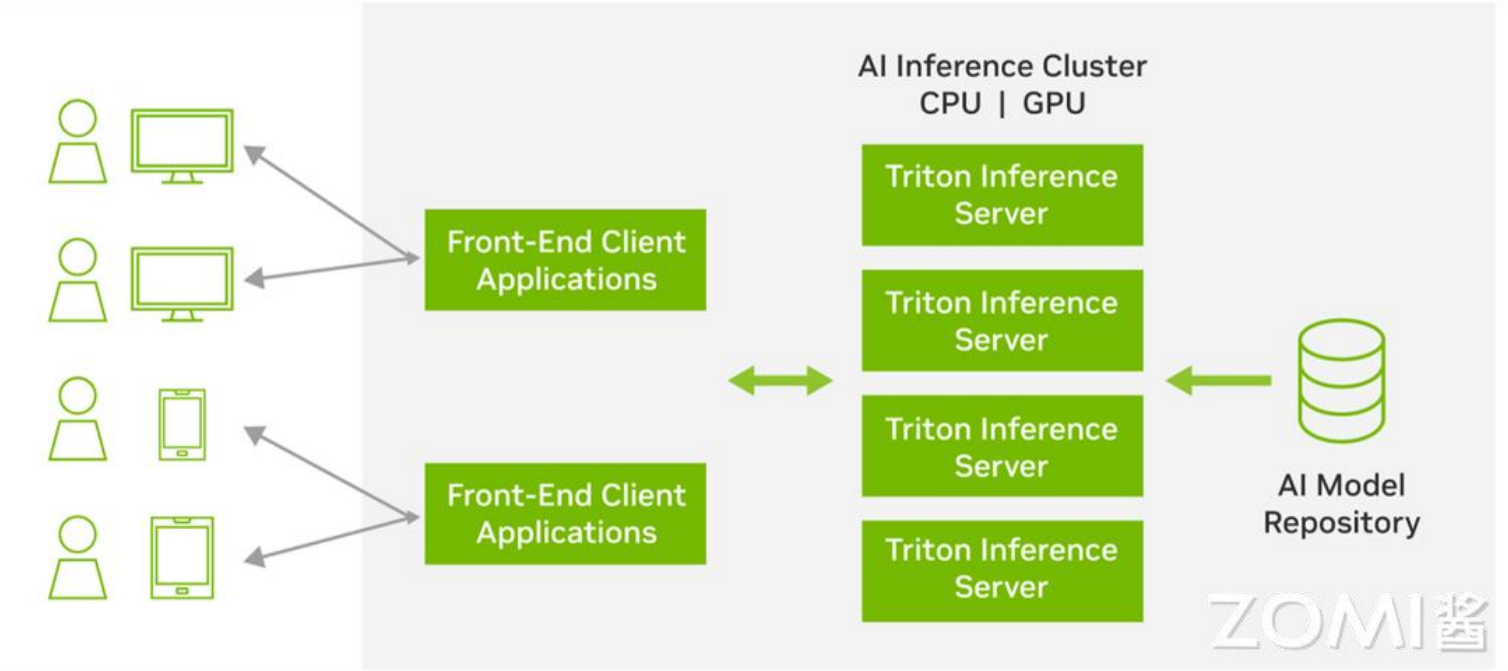
```
# SGLang的典型使用示例
from sglang import Runtime, endpoint
# 指定使用Triton后端
rt = Runtime(
    model_path="meta-llama/Llama-3-70b-chat-hf",
    backend="triton",  # ← 关键配置项
    tp_size=4          # 张量并行度
)
@endpoint
def generate(prompt):
    return rt.generate(prompt, max_tokens=512)

# 实际运行时：
# 1. SGLang将模型分解为计算图
# 2. Triton负责将子图分配到4块GPU执行
# 3. Triton管理中间结果的显存交换
```

[用户请求]
↓
SGLang层（处理逻辑：动态批处理/KV缓存复用/采样策略）
↓
Triton层（执行逻辑：GPU核函数优化/显存页表管理）
↓
硬件层（A100/H100 GPU集群）

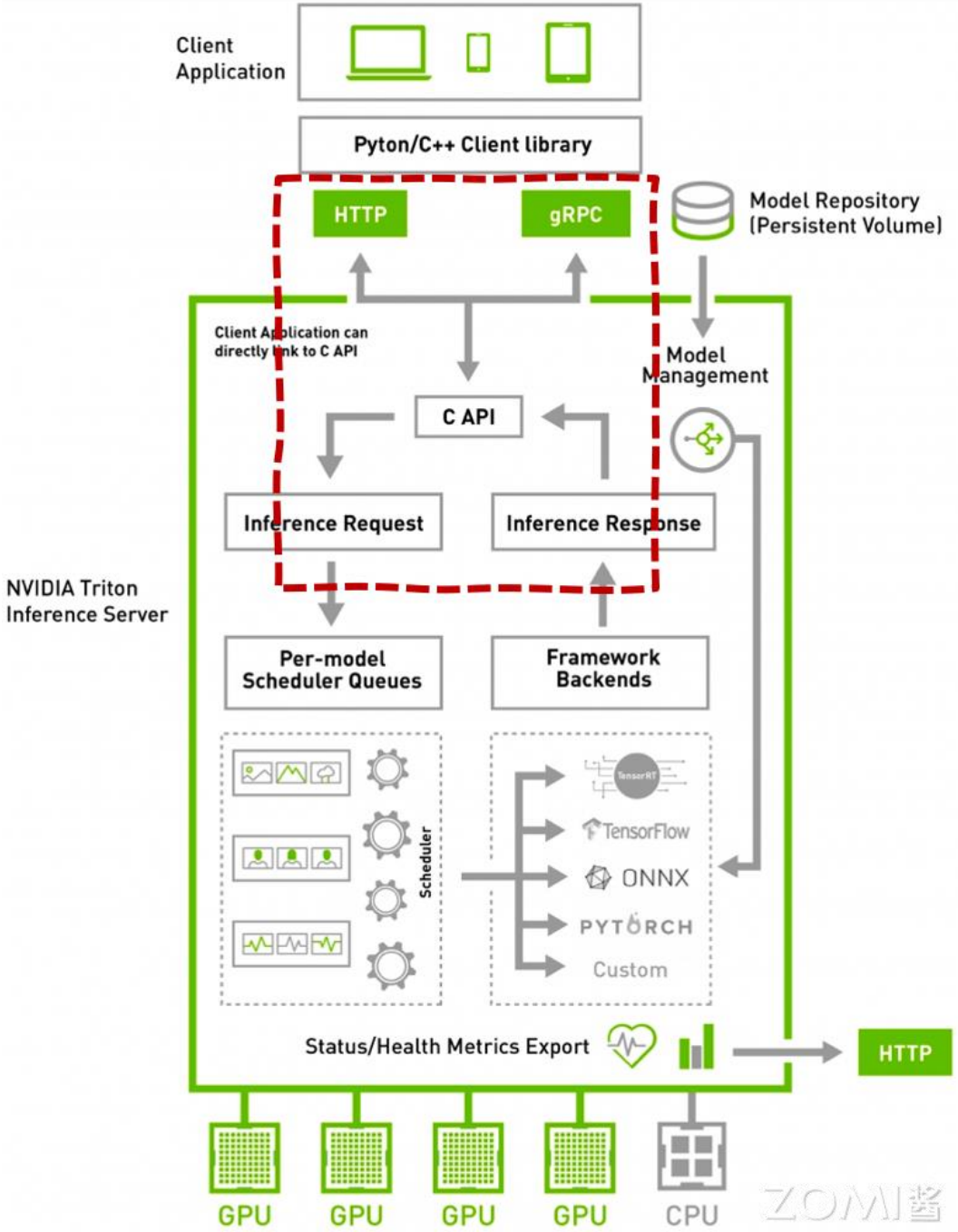
后端	吞吐量（tokens/s）	首token延迟（ms）	显存占用（GB）
纯Triton	1120	145	82
SGLang+PyTorch	890	210	78
SGLang+Triton	**1860**	**92**	**67**

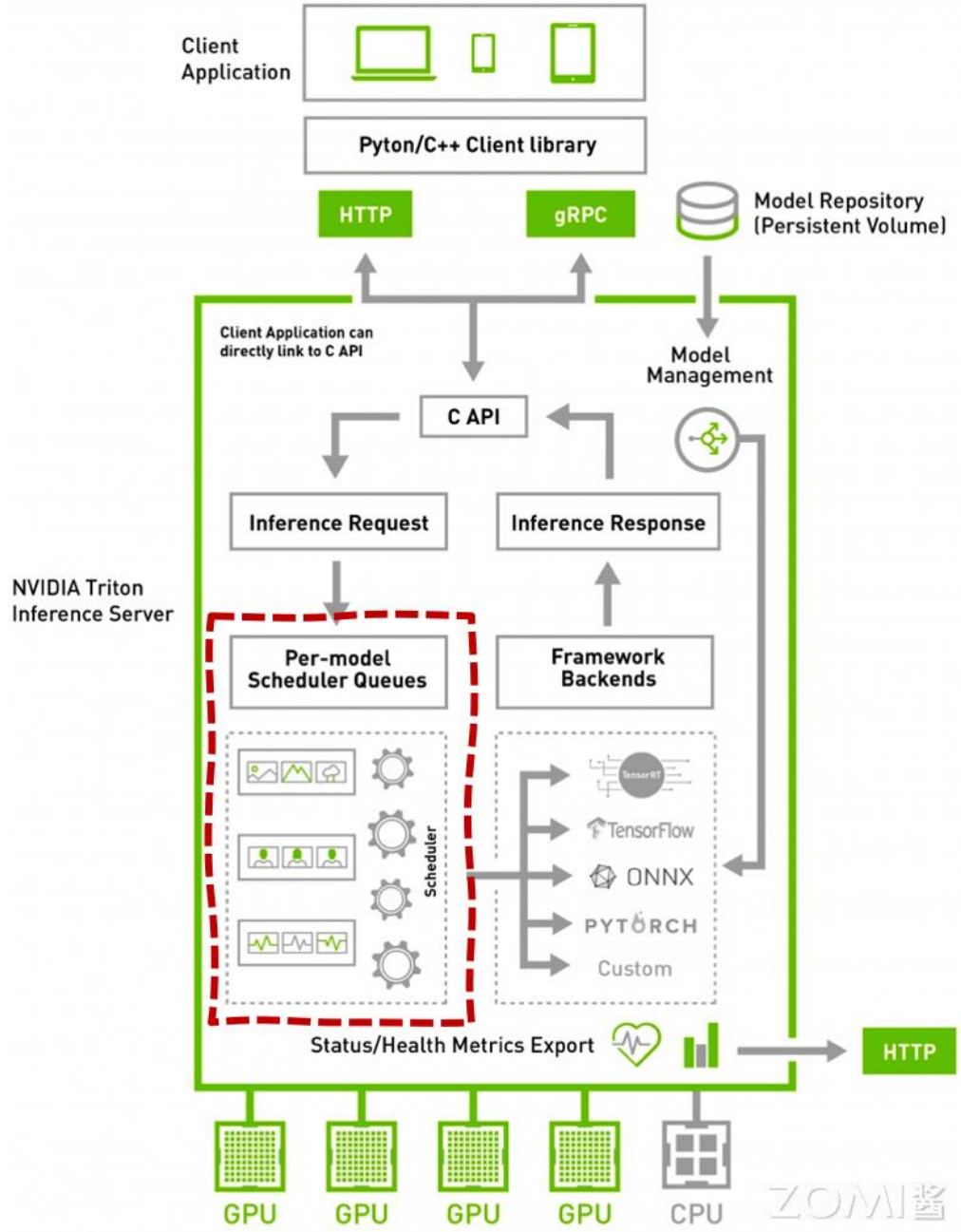
关键结论：
1. **吞吐量提升67%**：Triton的CUDA内核优化 + SGLang的KV缓存策略产生叠加优势
2. **显存占用下降18%**：Triton的页式内存管理 与 SGLang的缓存复用形成互补
3. **延迟降低50%**：Triton的异步执行流水线 有效隐藏了SGLang图优化的开销



Triton 的主要特点包括:

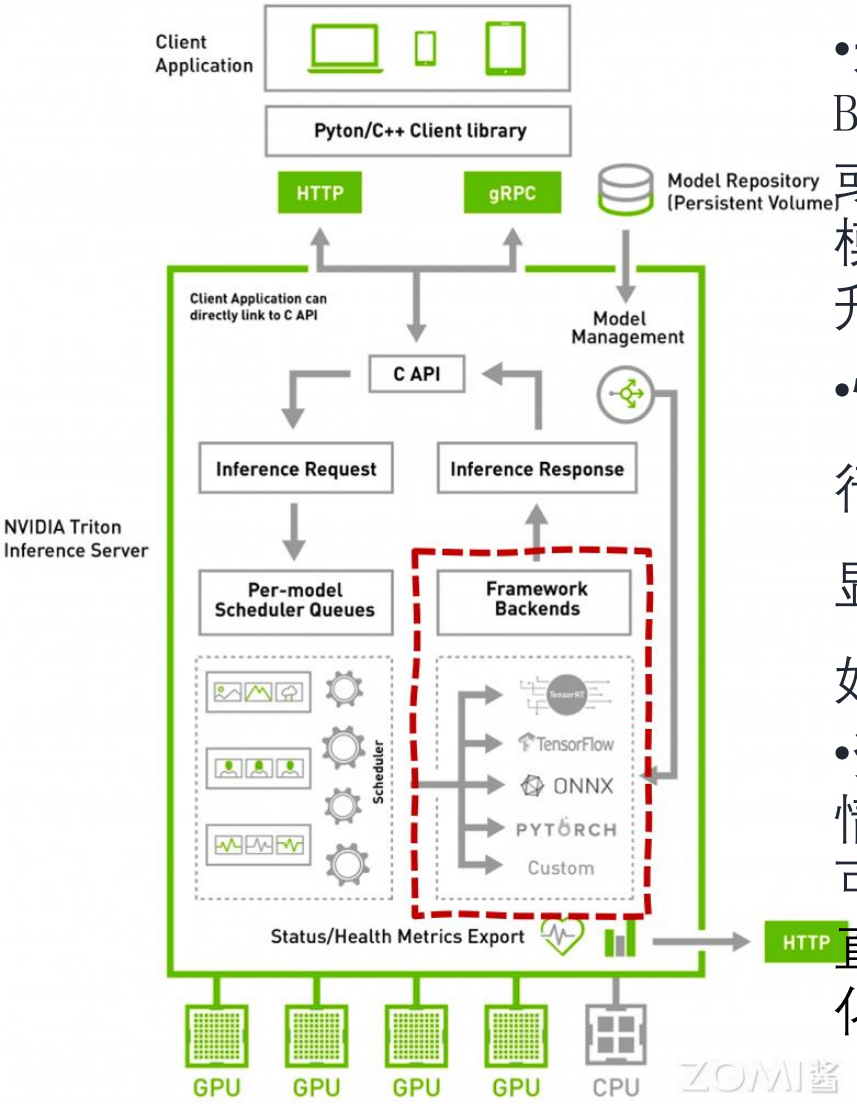
- **高性能**: 通过优化模型加载、执行和卸载的流程，Triton 显著提高了推理性能。
- **可扩展性**: 支持水平扩展和垂直扩展，能够适应不同的计算资源和负载需求。
- **多框架支持**: 兼容 TensorFlow、PyTorch、ONNX 等主流 AI 框架。
- **模型优化**: 集成 TensorRT 等优化工具，进一步提升模型推理性能。
- **灵活性**: 提供灵活的部署选项，支持公有云、私有云和边缘设备。
- **安全性**: 支持安全传输和访问控制，保障推理服务的安全性。


Client Application
Pyton/C++ Client library
HTTP
gRPC
Model Repository (Persistent Volume)
Client Application can directly link to C API
Model Management
C API
Inference Request
Inference Response
NVIDIA Triton Inference Server
Per-model Scheduler Queues
Framework Backends
Scheduler
TensorRT
TensorFlow
ONNX
PYTORCH
Custom
Status/Health Metrics Export
HTTP
GPU
GPU
GPU
GPU
CPU


Client Application
Pyton/C++ Client library
HTTP
gRPC
Model Repository (Persistent Volume)
Client Application can directly link to C API
Model Management
C API
Inference Request
Inference Response
NVIDIA Triton Inference Server
Per-model Scheduler Queues
Framework Backends
Scheduler
TensorRT
TensorFlow
ONNX
PYTORCH
Custom
Status/Health Metrics Export
HTTP
GPU
GPU
GPU
GPU
CPU

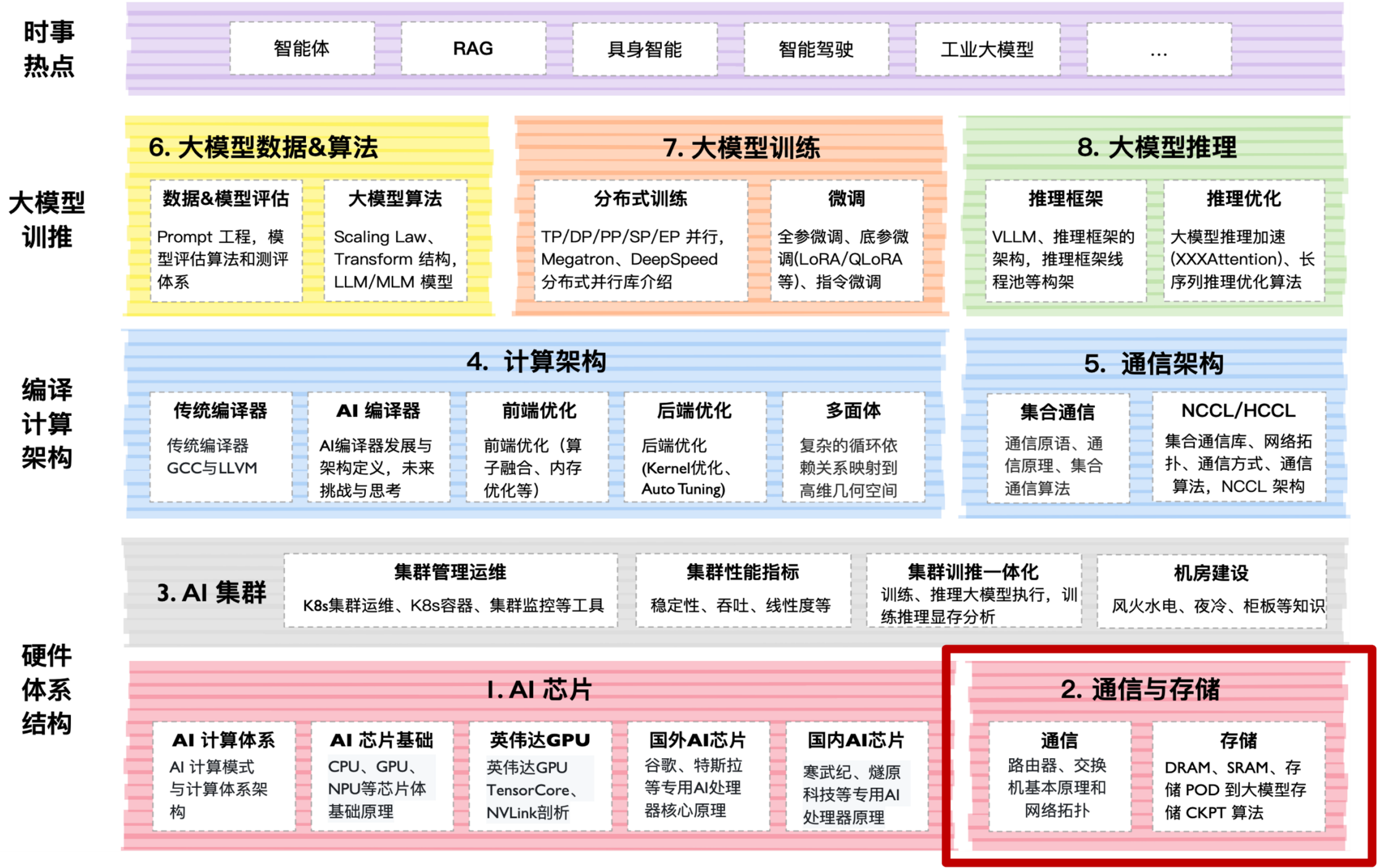
Triton 推理引擎

Triton 的一大亮点在于其高度灵活且强大的推理引擎支持体系，将 TensorFlow、TensorRT、PyTorch、ONNX Runtime 等主流框架统一整合为“Backends”。这一设计极大地促进了神经网络模型部署的标准化和效率，使得开发者能够在一个统一的平台上轻松管理多样化的模型，而无需关注底层实现细节，获得具有多后端架构的优势。



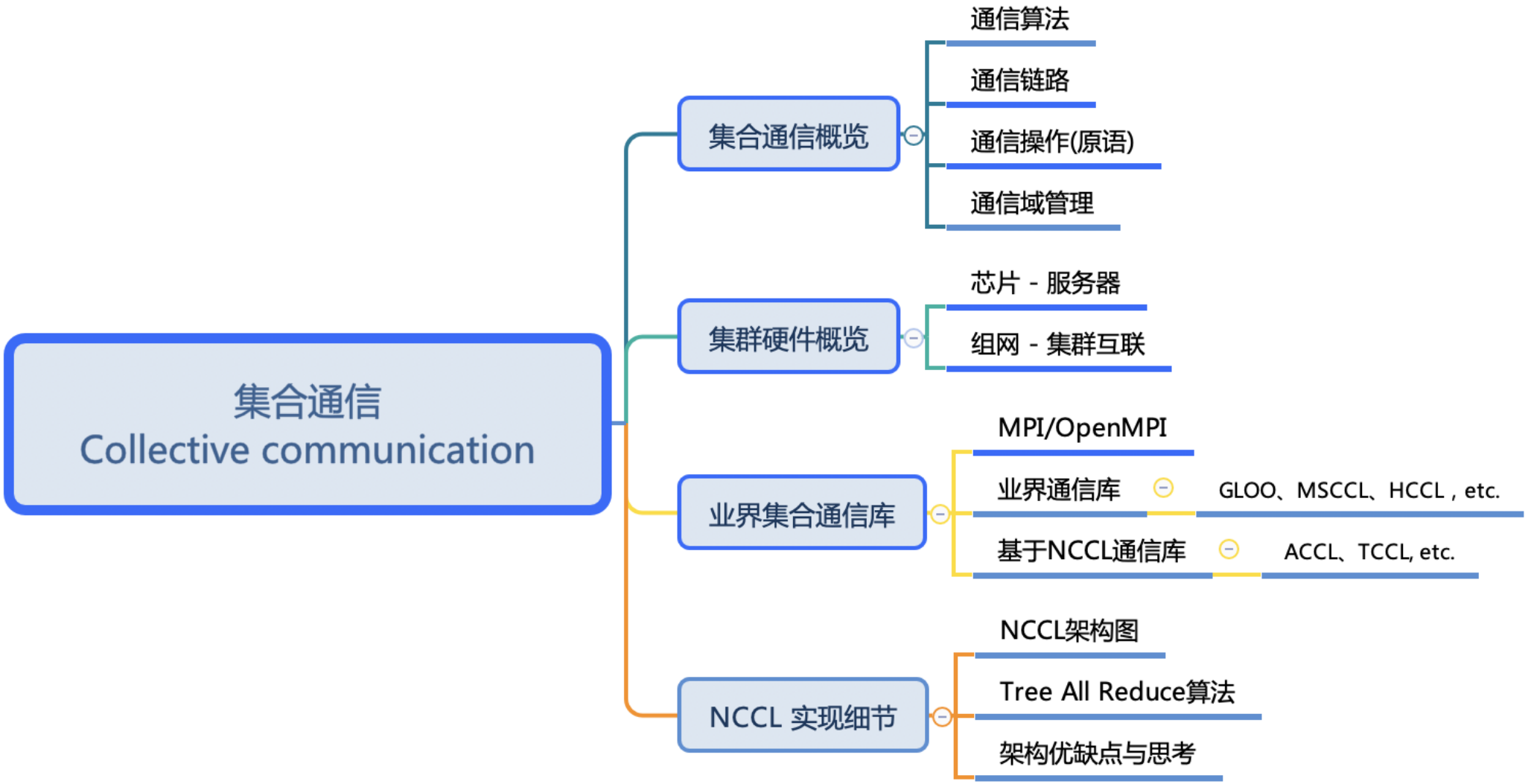
- **无缝迁移与混合部署**：通过将不同框架的模型推理能力抽象为统一的 Backend 接口，Triton 允许用户在不修改模型代码的情况下，自由选择或切换推理引擎。这意味着，开发者可以在 TensorFlow 模型和 PyTorch 模型之间轻松迁移，甚至在同一服务中混合部署多种框架的模型，极大提升了开发效率和灵活性。
- **性能优化与硬件加速**：Triton 集成的每个 Backend 都针对特定框架进行了优化，尤其是 TensorRT，作为英伟达开发的高性能推理加速库，能显著提高 GPU 上的推理速度。此外，Triton 还能自动利用硬件加速特性，如 FP16、INT8 量化，进一步提升吞吐量和降低延迟。
- **资源高效利用**：多后端架构使得 Triton 能够根据模型特性和硬件资源情况智能选择最合适的推理引擎。例如，对于某些模型，使用 TensorRT 可能比原生 TensorFlow 提供更好的性能；而对于复杂的 PyTorch 模型，直接利用 PyTorch Backend 可能更为合适。这种动态适配策略有助于最大化资源利用率。

通信框架


时事热点
智能体
RAG
具身智能
智能驾驶
工业大模型
...
大模型训推
6. 大模型数据&算法
数据&模型评估
Prompt 工程，模型评估算法和测评体系
大模型算法
Scaling Law、Transform 结构，LLM/MLM 模型
7. 大模型训练
分布式训练
TP/DP/PP/SP/EP 并行，Megatron、DeepSpeed 分布式并行库介绍
微调
全参微调、底参微调(LoRA/QLoRA 等)、指令微调
8. 大模型推理
推理框架
VLLM、推理框架的架构，推理框架线程池等构架
推理优化
大模型推理加速(XXXAttention)、长序列推理优化算法
编译计算架构
4. 计算架构
传统编译器
传统编译器 GCC与LLVM
AI 编译器
AI编译器发展与架构定义，未来挑战与思考
前端优化
前端优化（算子融合、内存优化等）
后端优化
后端优化(Kernel优化、Auto Tuning)
多面体
复杂的循环依赖关系映射到高维几何空间
5. 通信架构
集合通信
通信原语、通信原理、集合通信算法
NCCL/HCCL
集合通信库、网络拓扑、通信方式、通信算法，NCCL 架构
硬件体系结构
3. AI 集群
集群管理运维
K8s集群运维、K8s容器、集群监控等工具
集群性能指标
稳定性、吞吐、线性度等
集群训推一体化
训练、推理大模型执行，训练推理显存分析
机房建设
风火水电、夜冷、柜板等知识
1. AI 芯片
AI 计算体系
AI 计算模式与计算体系架构
AI 芯片基础
CPU、GPU、NPU等芯片体基础原理
英伟达GPU
英伟达GPU TensorCore、NVLink剖析
国外AI芯片
谷歌、特斯拉等专用AI处理器核心原理
国内AI芯片
寒武纪、燧原科技等专用AI处理器原理
2. 通信与存储
通信
路由器、交换机基本原理和网络拓扑
存储
DRAM、SRAM、存储 POD 到大模型存储 CKPT 算法

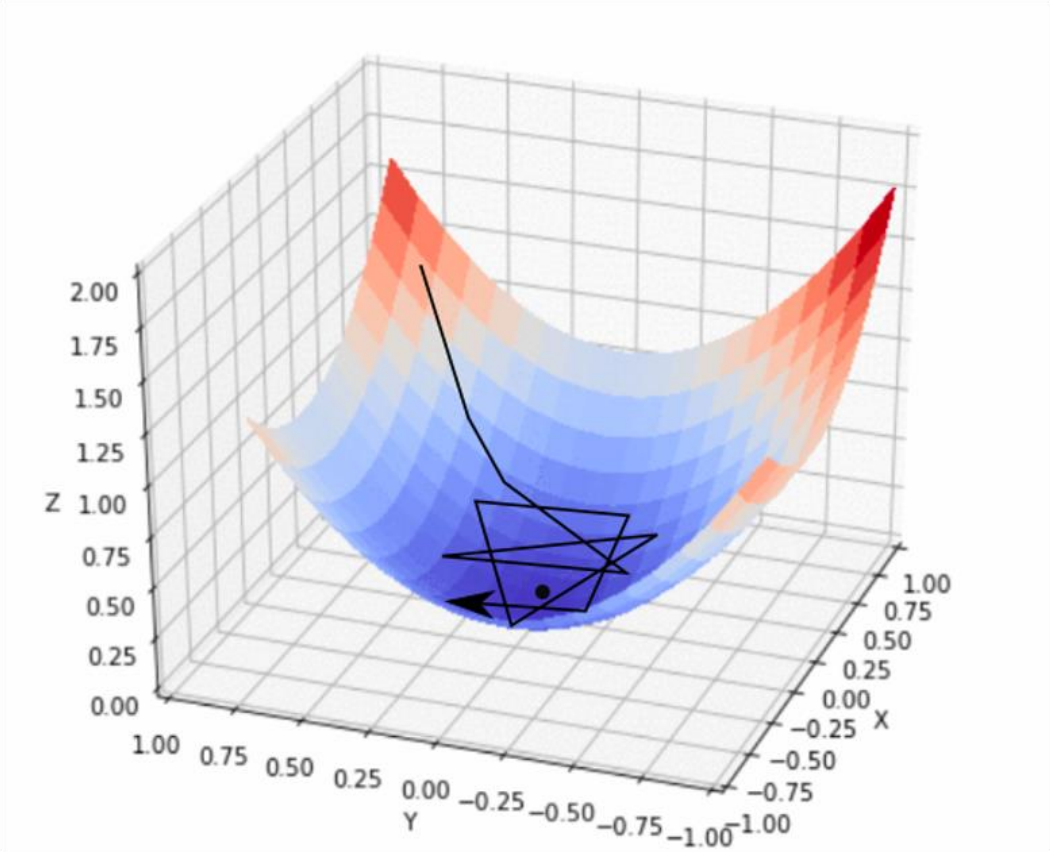
通信特性	HCCL	NCCL
通信算法	ring/mesh + ring/Hav-Doub/Pair-Wise， etc.	ring + Tree ring， etc.
通信链路	NIC / 灵渠总线 / PCIE	NIC / NVLink / NVSwitch / GPU-Direct / PCIE / CMC
通信操作	allreduce、broadcast、reduce、reduce scatter、allgather、all2all、send、recv	allreduce、broadcast、reduce、reduce scatter、allgather、all2all、send、recv
通信域管理	全局通信域、子通信域、基于全局/子通信域配置算法	全局通信域、子通信域、自定义通信域配置算法

思维导图 XMind

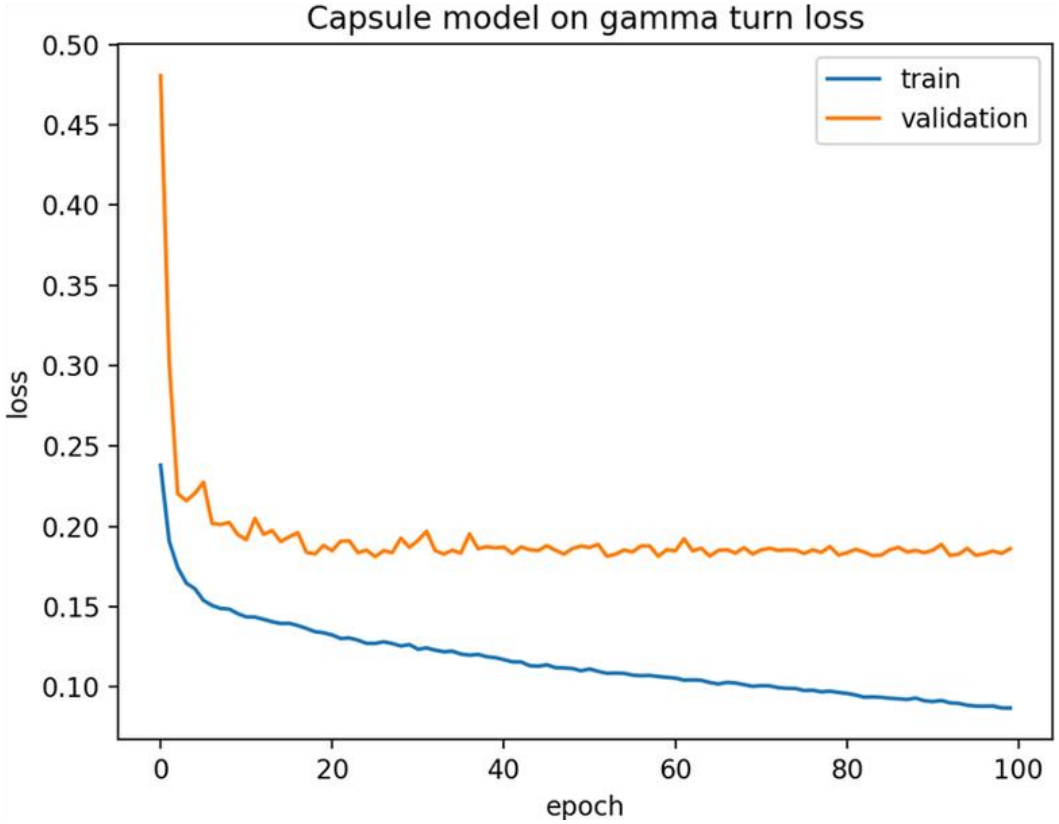

集合通信
Collective communication
集合通信概览
通信算法
通信链路
通信操作(原语)
通信域管理
集群硬件概览
芯片 - 服务器
组网 - 集群互联
业界集合通信库
MPI/OpenMPI
业界通信库
GLOO、MSCCL、HCCL , etc.
基于NCCL通信库
ACCL、TCCL, etc.
NCCL 实现细节
NCCL架构图
Tree All Reduce算法
架构优缺点与思考

单卡训练神经网络模型

梯度下降算法寻找数据鞍点

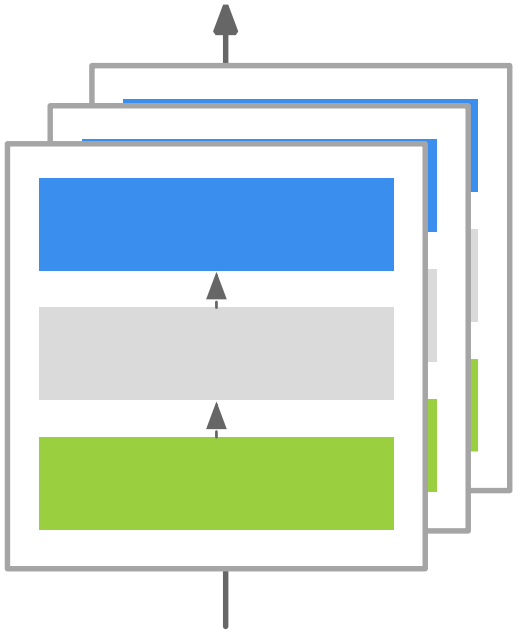


损失值在下降



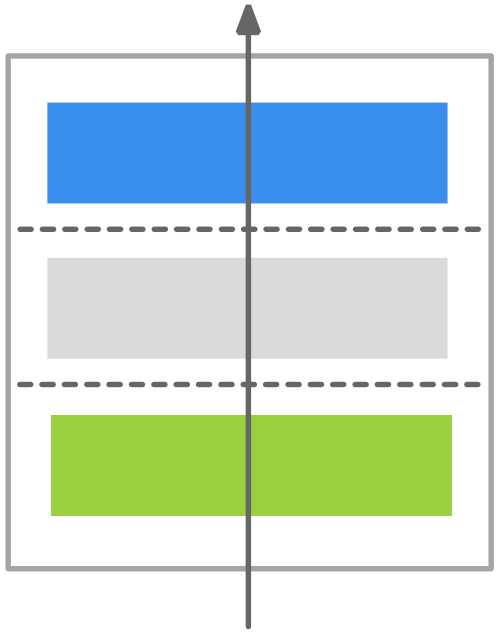
分布式训练

Data Parallelism



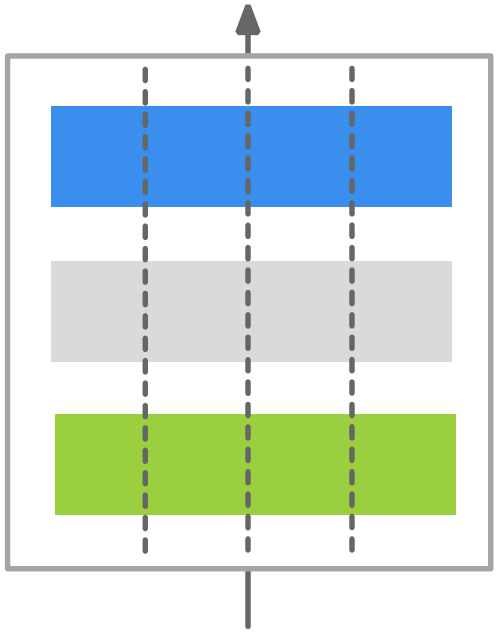
将数据分为若干份，分别映射到不同的 NPU

Pipeline Parallelism



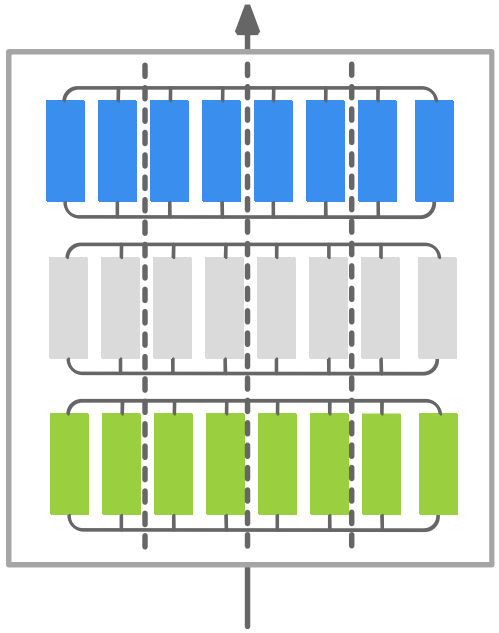
将一个网络拆分为多个流水 Stage 在不同 NPU

Tensor Parallelism



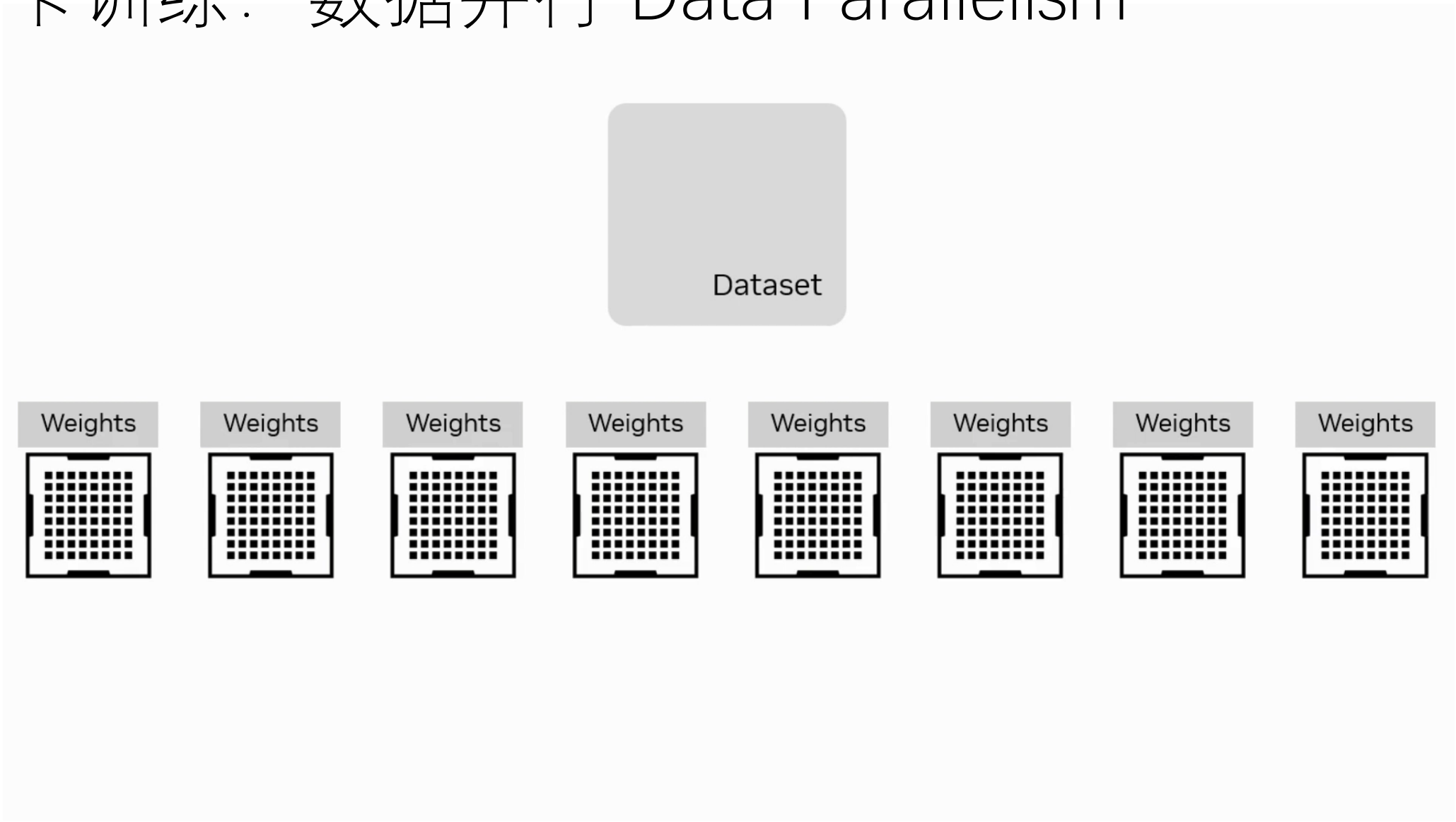
将模型层内的参数切分到不同 NPU

Expert Parallelism

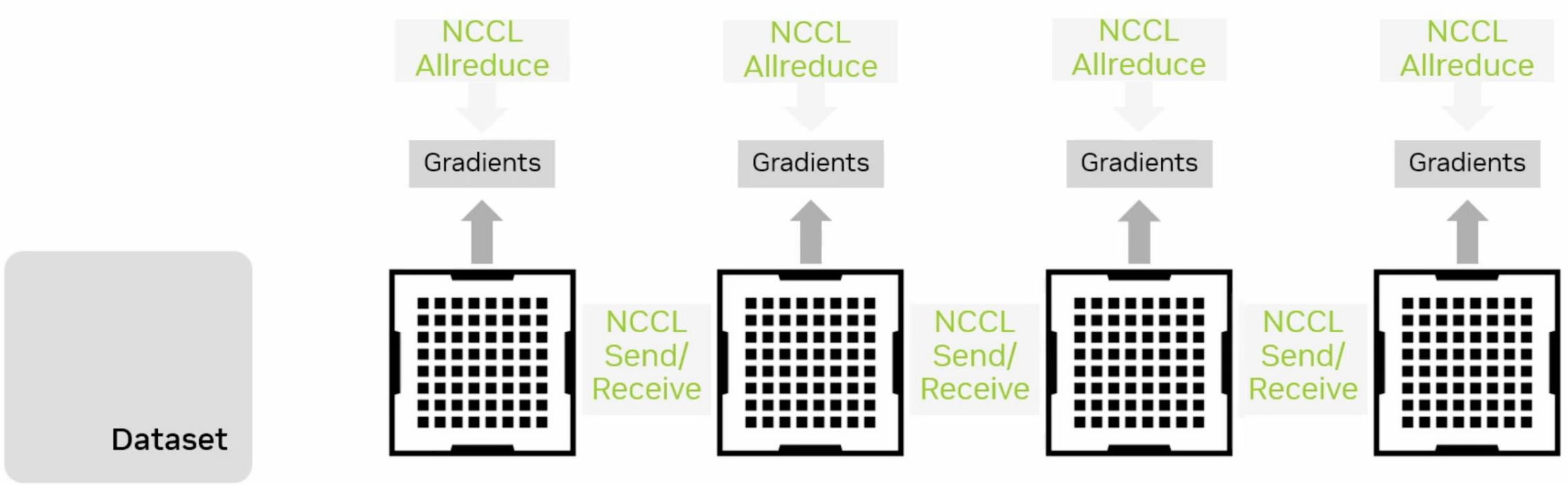


专家放置在不同 NPU，处理不同 Batch 样本

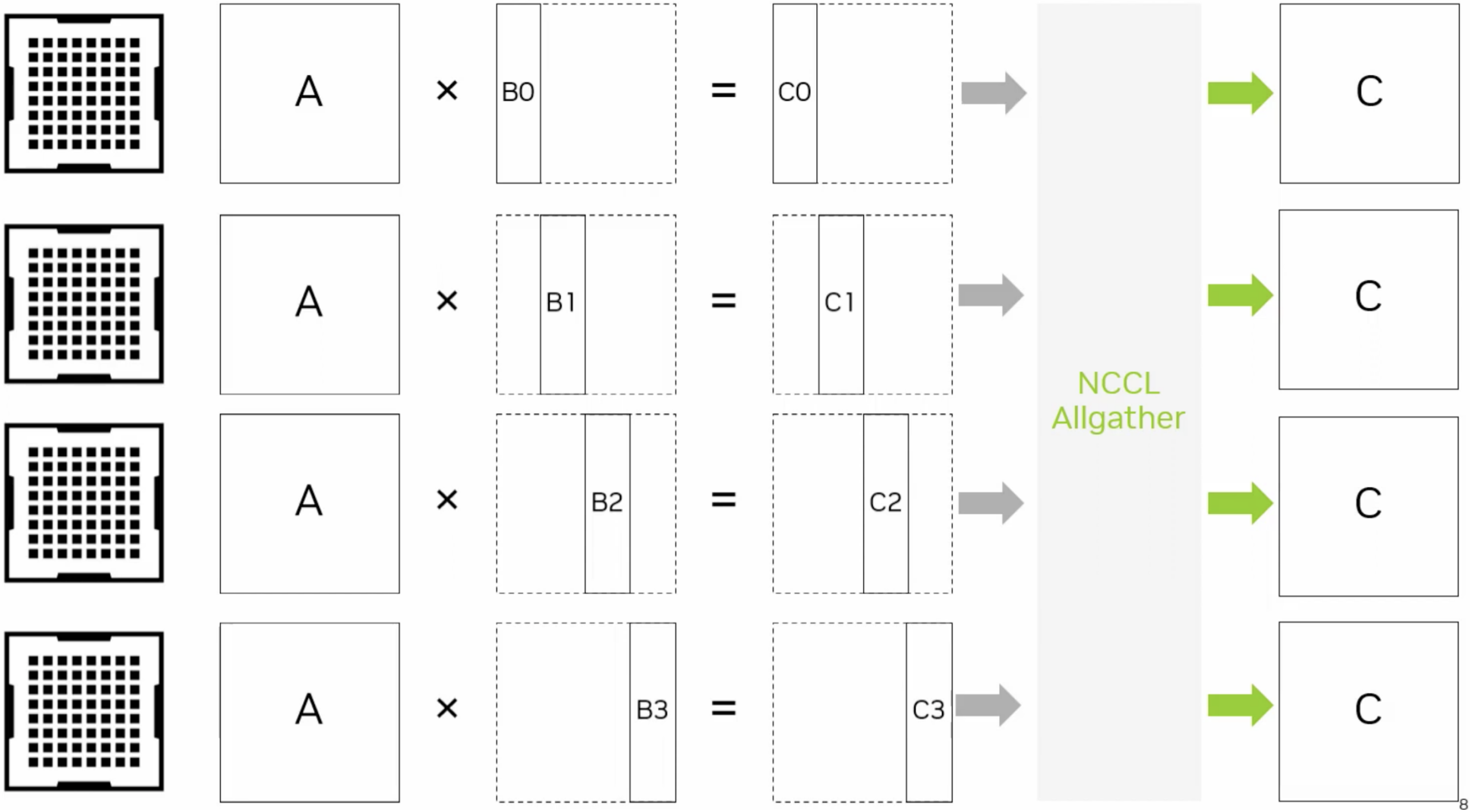
多卡训练：数据并行 Data Parallelism



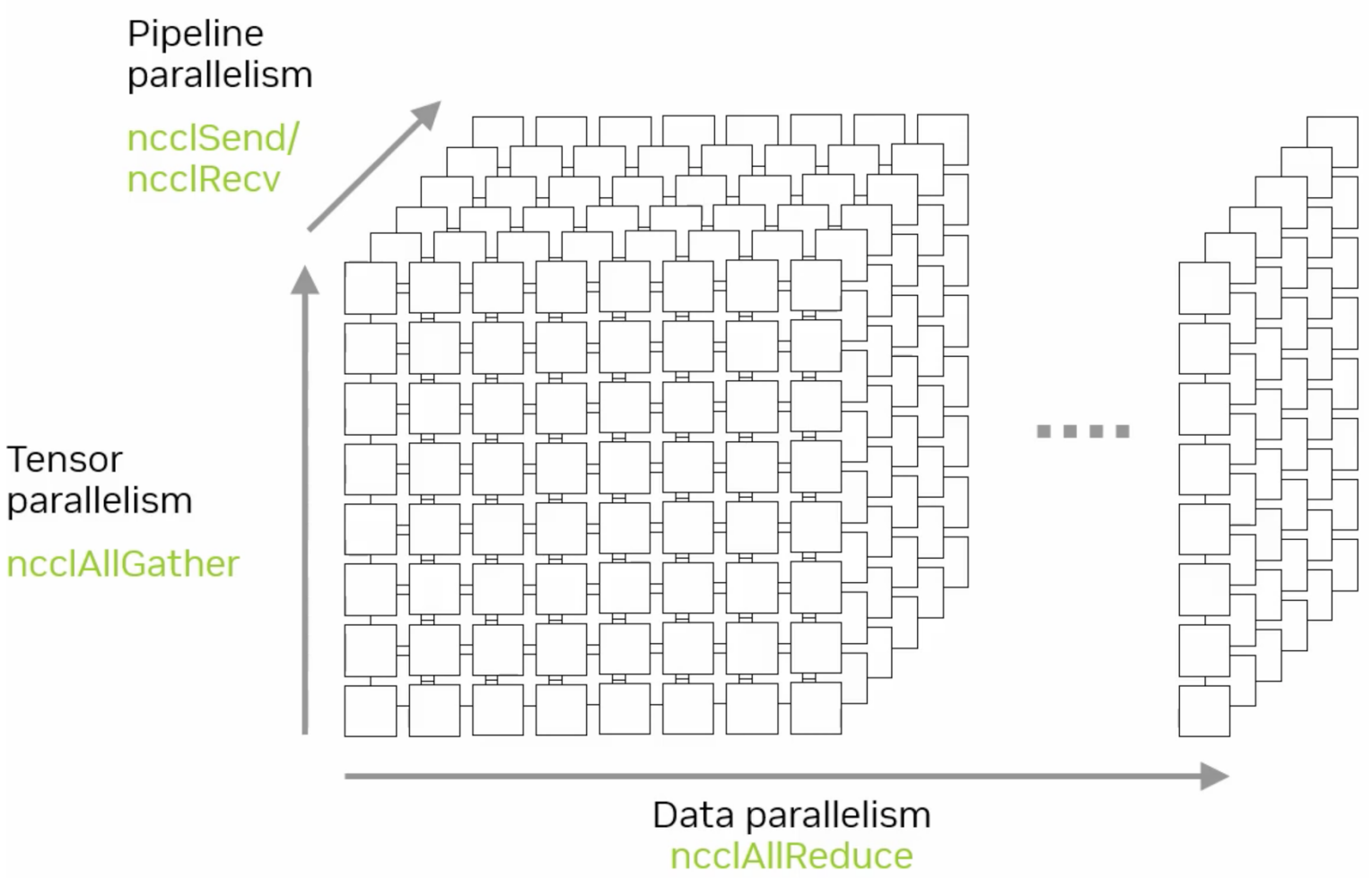
多卡训练：流水并行 Pipeline Parallelism



多卡训练：张量并行 Tensor Parallelism



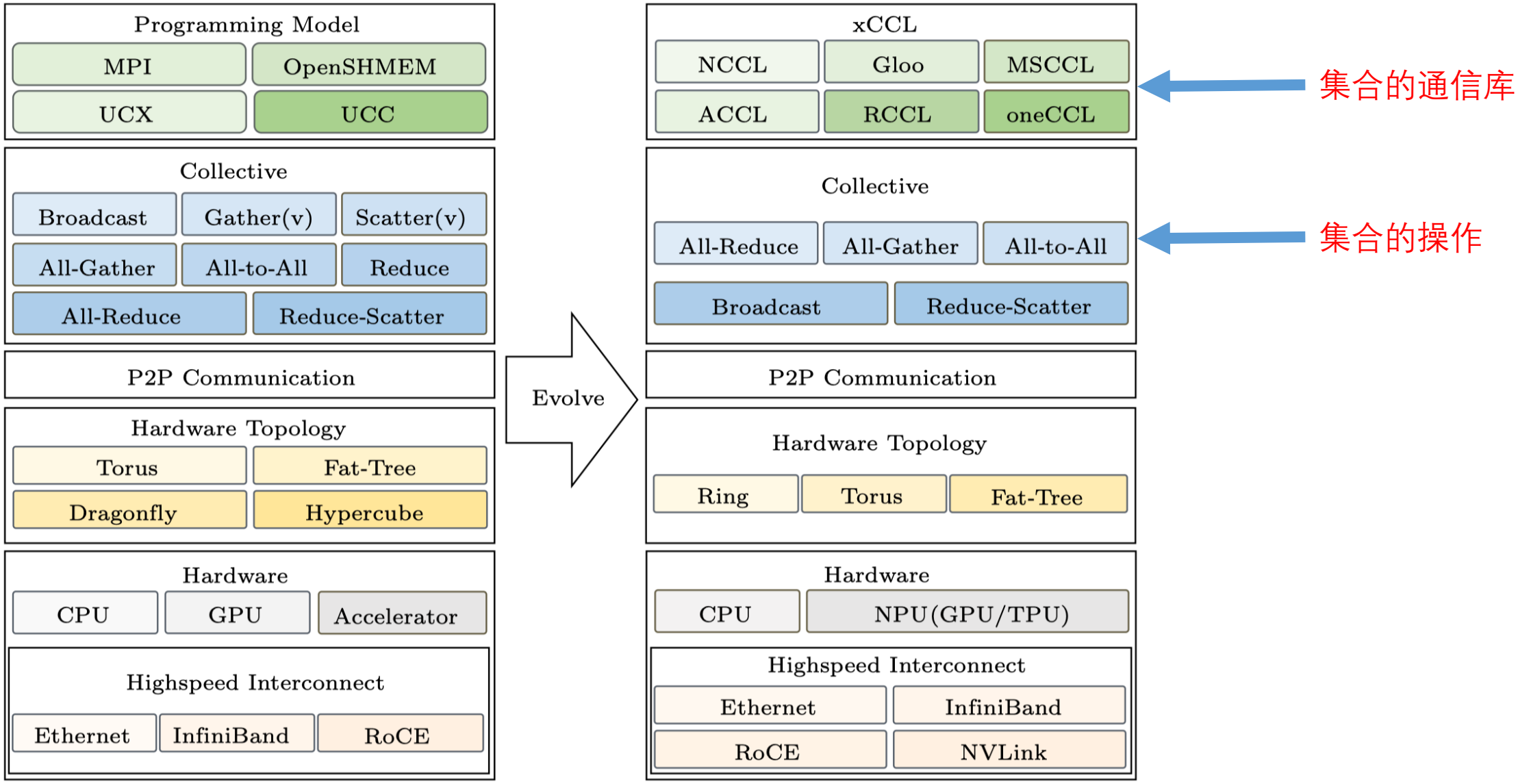
多卡训练：多维并行 Multi Parallelism



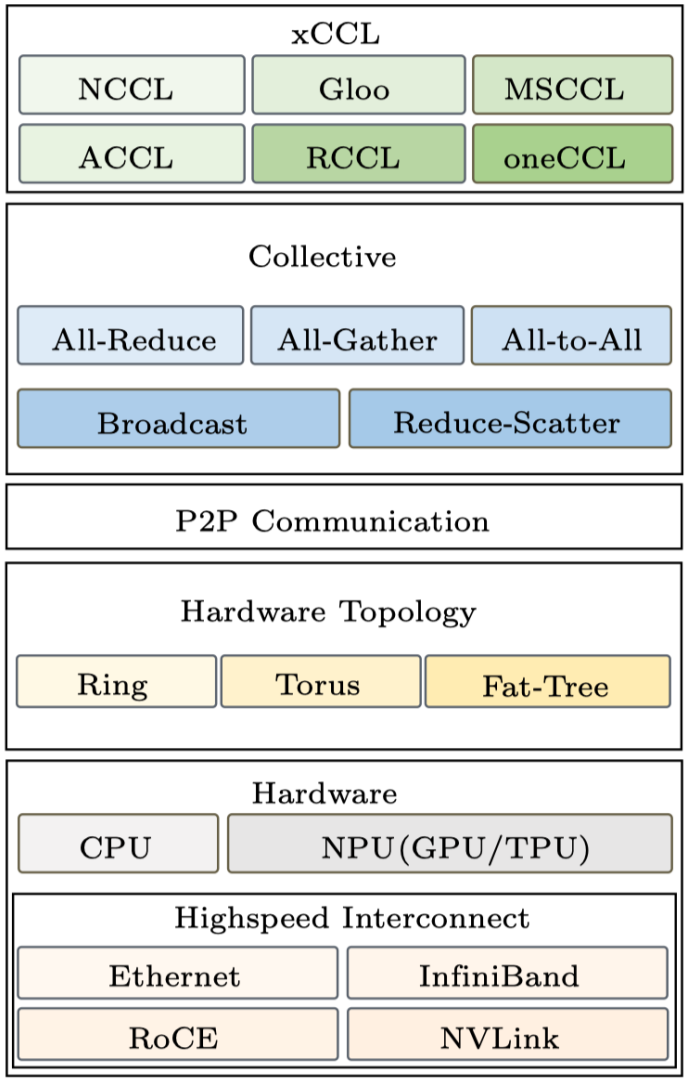
- MoE(Mixture of Experts)
  - Send/Recv(all2all)
- FSDP(Full Sharded DP)
  - AllGather
- Long Sequence
  - AllGather/AllReduce

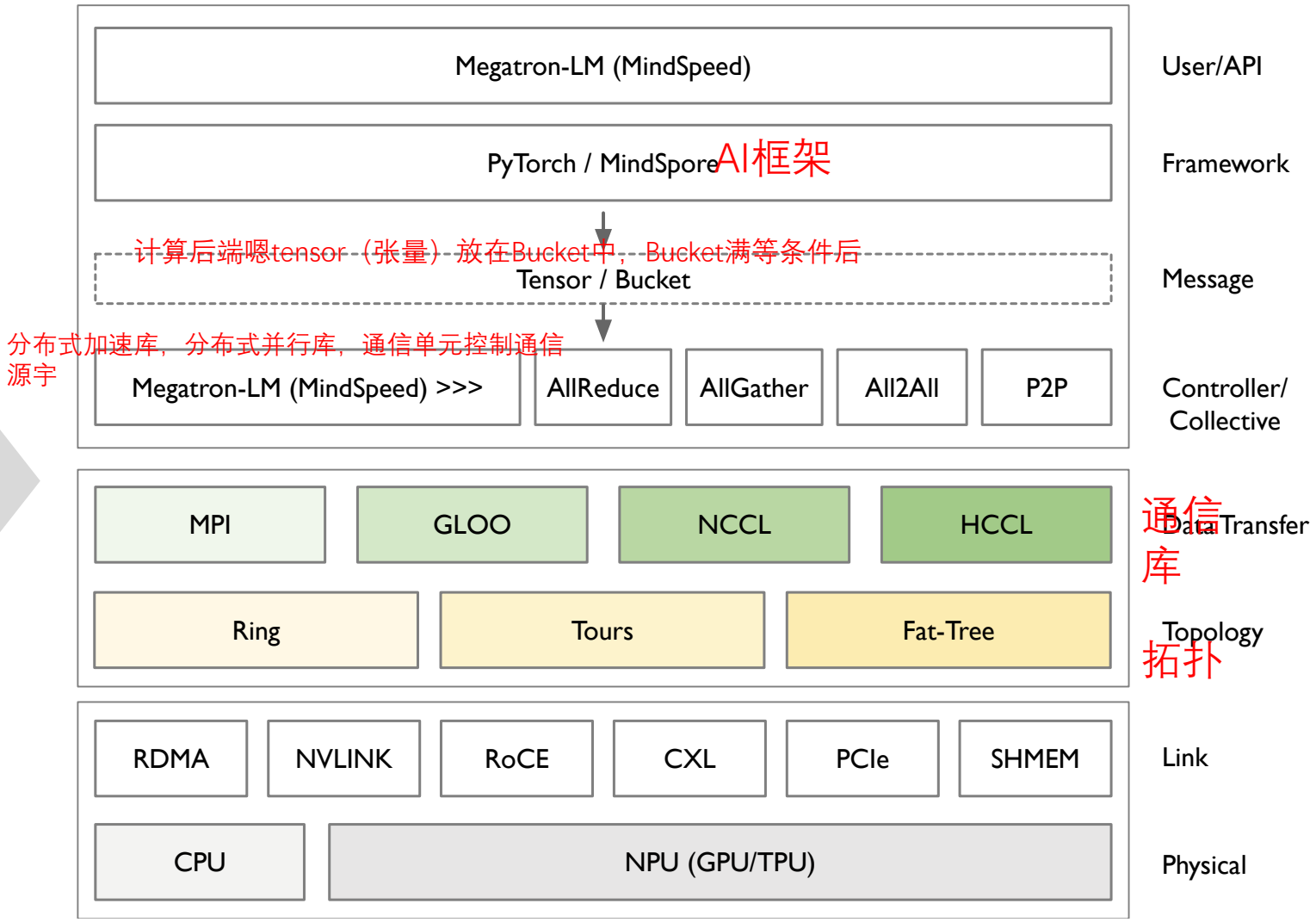
计算与通信解耦

- 将计算与通信解耦，计算的归计算，通信的归通信，通过性能优化策略减少通信的次数：
  - 提升集群训练性能（模型利用率 MFU/算力利用率 MFU）；
  - 避免通信与计算假死锁（计算耗时长，通信长期等待）；

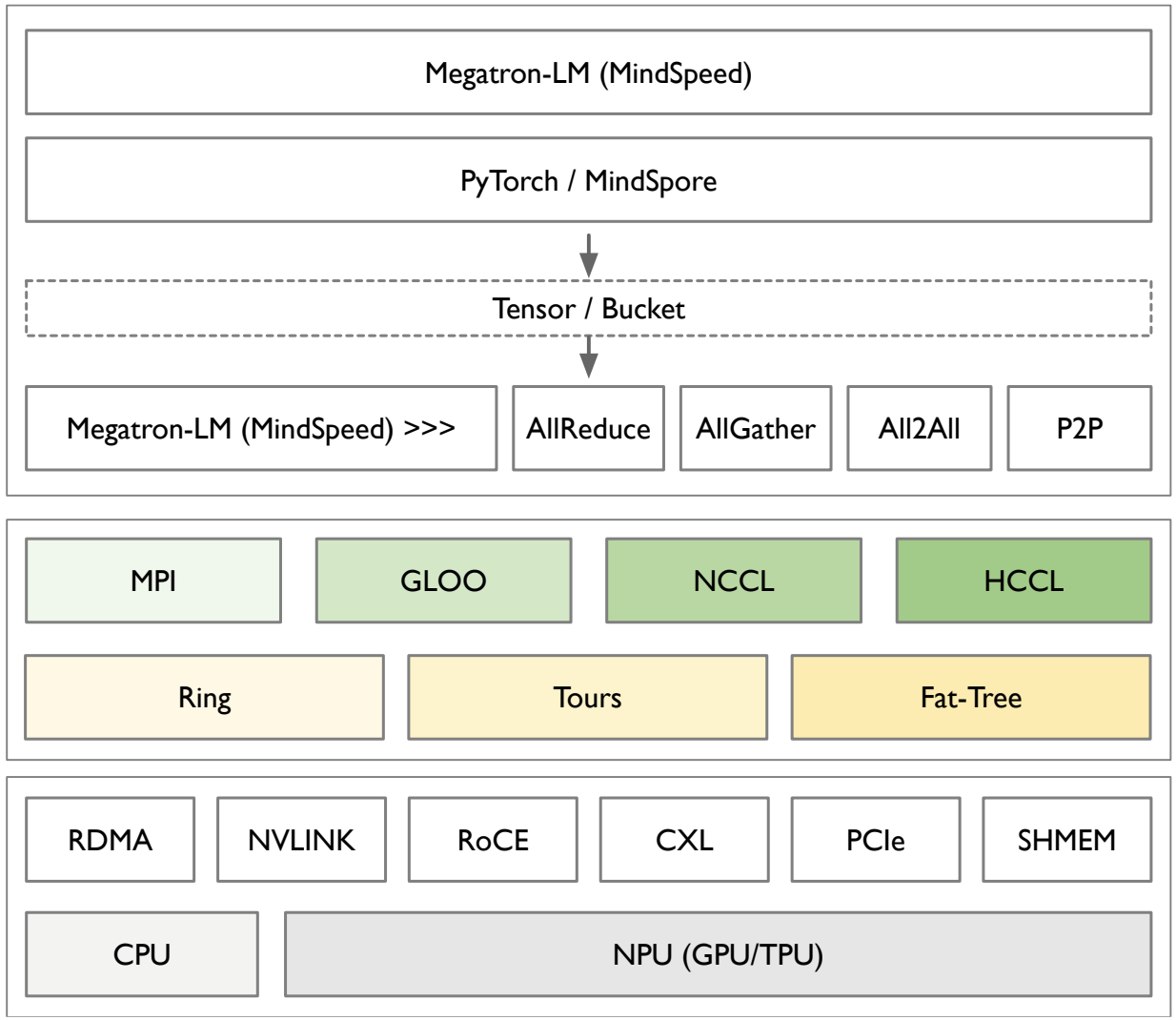

Programming Model
MPI
OpenSHMEM
UCX
UCC
Collective
Broadcast
Gather(v)
Scatter(v)
All-Gather
All-to-All
Reduce
All-Reduce
Reduce-Scatter
P2P Communication
Hardware Topology
Torus
Fat-Tree
Dragonfly
Hypercube
Hardware
CPU
GPU
Accelerator
Highspeed Interconnect
Ethernet
InfiniBand
RoCE
Evolve
xCCL
NCCL
Gloo
MSCCL
ACCL
RCCL
oneCCL
Collective
All-Reduce
All-Gather
All-to-All
Broadcast
Reduce-Scatter
P2P Communication
Hardware Topology
Ring
Torus
Fat-Tree
Hardware
CPU
NPU(GPU/TPU)
Highspeed Interconnect
Ethernet
InfiniBand
RoCE
NVLink
集合的通信库
集合的操作
(a)
(b)

XCCL 在 AI 系统中的位置





XCCL 在 AI 系统中的位置



- Megatron-LM/MindSpeed 分布式加速框架解耦了计算与通信：
  1. 计算主要通过 PyTorch 等AI框架执行；
  2. 通信通过 XCCL 通信库来执行；

- **做一个maxxx的分发，通过原语分发**

- 将框架计算出来 Tensor 记录到 Bucket 中
- 通过控制层在后台启动 loop 线程
- 周期性的从 Bucket 中读取 Tensor
- 控制层在节点之间协商一致后，进行消息分发到具体 NPU 上执行通信

类型	函数名	含义
通信	MPI_Bcast	一对多广播相同消息
	MPI_Gather	多对一收集各进程消息
	MPI_Gatherv	MPI_Gather 一般化
	MPI_Allgather	全局收集
	MPI_Allgatherv	MPI_Allgather 一般化
	MPI_Scatter	一对多散播不同消息
	MPI_Scatterv	MPI_Scatter 一般化
	MPI_Alltoall	多对多全局交换信息
	MPI_Alltoallv	MPI_Alltoall 一般化
规约	MPI_Reduce	多对一规约
	MPI_Allreduce	MPI_Reduce 一般化
	MPI_Reduce_scatter	MPI_Reduce 一般化
	MPI_Scan	前缀和
同步	MPI_Barrier	路障同步

- MPI（message passing interface）
- 当 MPI 进程启动后，每个 Process 会分配唯一序号 Rank。集合通信需要指定一个协调者（e.g. Rank 0 Process，一般称为 ROOT）。
- 算子：深度学习算法由一个个计算单元组成，我们称这些计算单元为算子（Operator，简称OP）。在网络模型中，算子对应层中的计算逻辑，例如：卷积层（Convolution Layer）是一个算子；全连接层（Fully-connected Layer， FC layer）中的权值求和过程，是一个算子。
- 梯度是一个与函数相切的向量，指向此函数最大增量的方向。
- 权重也称为参数，是与神经网络中神经元或单元之间的连接相关的值。
- 另一方面，偏差是机器学习模型中的附加参数，允许微调和改变预测。 偏差使模型能够解释无法仅通过输入特征捕获的因素。 它们可以被认为是偏移量或截距，即使所有输入特征为零，也可以帮助模型进行预测。 在神经网络中，偏差通常表示为网络中单独的神经元，具有固定的输入值1 和相关的权重。 即使输入特征没有提供足够的信息，偏置神经元也能确保模型能够做出预测。