

DIP: Efficient Large Multimodal Model Training with Dynamic Interleaved Pipeline

Zhenliang Xue

Institute of Parallel and Distributed
Systems
Shanghai Jiao Tong University
Shanghai, China
xuezhenliang@sjtu.edu.cn

Hanpeng Hu

StepFun
Shanghai, China
haanpeng@outlook.com

Xing Chen

StepFun
Shanghai, China
xchen382@asu.edu

Yimin Jiang

StepFun
Shanghai, China
jymthu@gmail.com

Yixin Song

Institute of Parallel and Distributed
Systems
Shanghai Jiao Tong University
Shanghai, China
yixinsong@sjtu.edu.cn

Zeyu Mi

Institute of Parallel and Distributed
Systems
Shanghai Jiao Tong University
Shanghai, China
yzmizeyu@sjtu.edu.cn

Yibo Zhu

StepFun
Shanghai, China
zhuyibo@stepfun.com

Daxin Jiang

StepFun
Shanghai, China
djiang@stepfun.com

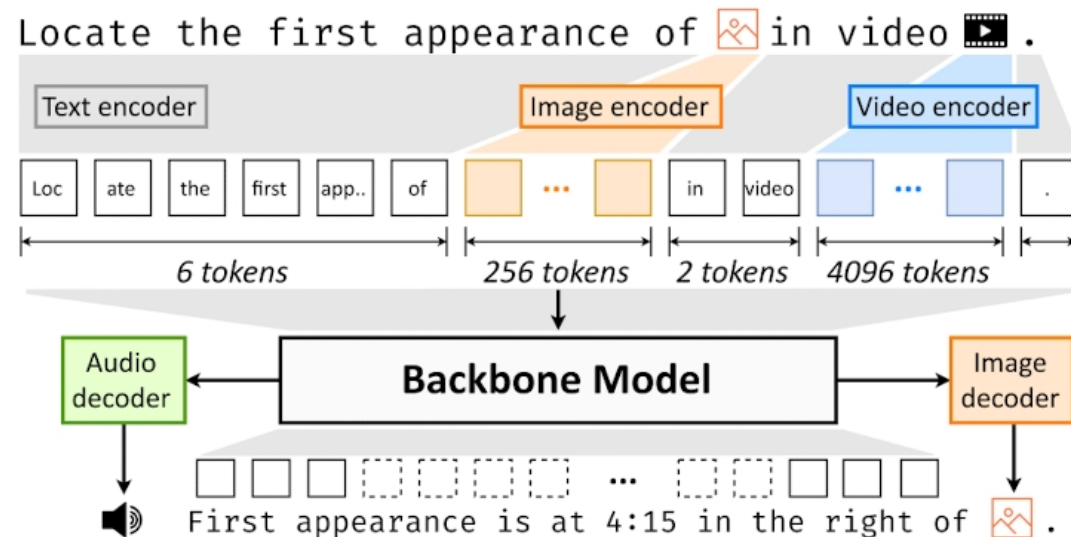
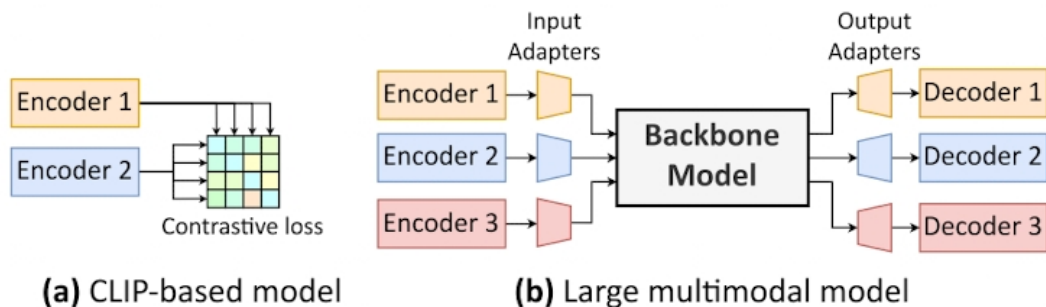
Yubin Xia

Institute of Parallel and Distributed
Systems
Shanghai Jiao Tong University
Shanghai, China
xiayubin@sjtu.edu.cn

Haibo Chen

Institute of Parallel and Distributed
Systems
Shanghai Jiao Tong University
Shanghai, China
haibochen@sjtu.edu.cn

Background: 为什么 LMM 训练更复杂



- 左图 Figure 1: 传统 CLIP-based model 更像两个 encoder 做表征对齐。
- 右图 Figure 2: 现代 LMM 会把 text encoder、image/video encoder、backbone model、decoder 串起来。
- 关键变化: 模型不再是均匀的一串 Transformer 层, 而是由不同模态模块组成的异构流水线。

先理解 Pipeline Bubble 是什么

Pipeline parallelism 的目标，是把模型切成多个 stage，让不同 GPU 像流水线一样连续处理 microbatch。理想情况下，每个 stage 用时差不多，GPU 就不会闲着。

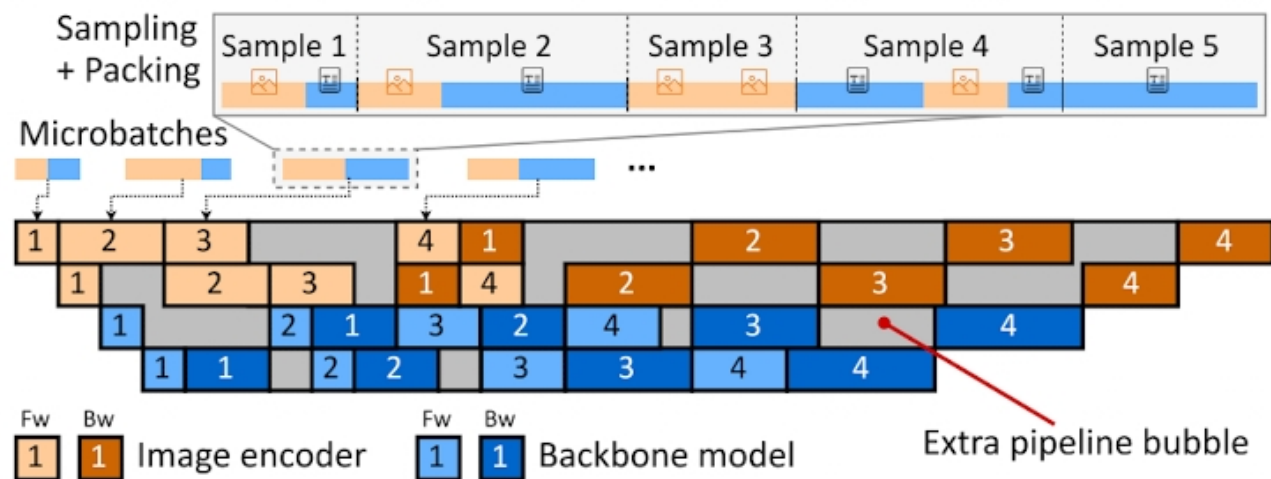


Figure 3. Illustration of the impact of dynamic imbalance.

- Figure 3 里，橙色代表 image encoder，蓝色代表 backbone model。
- 当某些 microbatch 的图片更多时，image encoder 部分会变慢。
- 红色箭头标出的 Extra pipeline bubble，就是快的 stage 等慢的 stage 产生的空闲时间。

Problem 1: 模型模块本身就不均衡

LMM 里不同模块的计算方式不一样。比如 ViT encoder 处理图片 patch，LM backbone 处理文本 token，DiT 处理视频扩散任务。它们不是同一种层，也很难被均匀切成 pipeline stage。

- 论文举例：即使用穷举搜索找 37B VLM 的最优 stage 切分，stage latency 仍然有 16.7% 的差异。

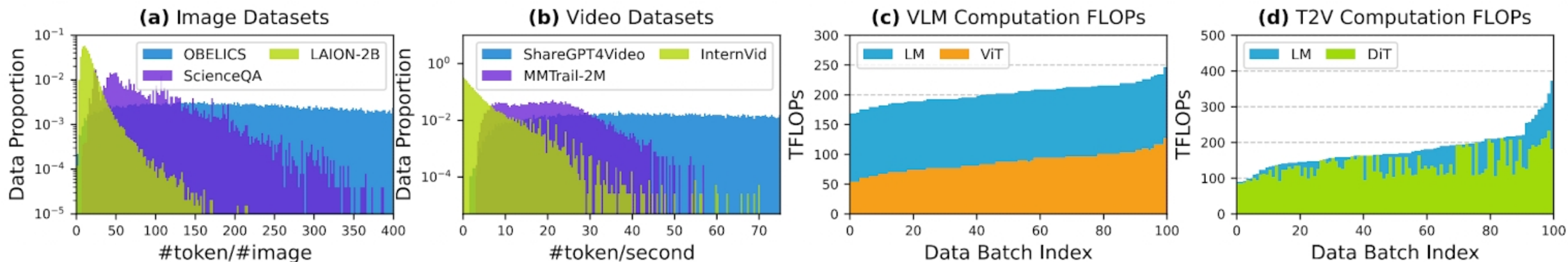
- Table 1 里，同样 7B 规模，静态 VLM 已经比纯 LM 慢；说明模型结构异构本身就会带来开销。

- 这个问题叫 pipeline stage imbalance。

Model Setup	Time (s)	PFLOPs	MFU
LM 7B	4.068	12.8	0.400
ViT 2B + LM 5B (static data)	4.567	12.7	0.351
ViT 2B + LM 5B (dynamic data)	6.789	12.8	0.239

直观理解：流水线每段工作量不一样，短的段就会等长的段。

Problem 2: 每个 batch 的模态比例也在变



- Figure 4(a)(b): 不同数据集中文本/图片、文本/视频的比例分布差异很大。
- Figure 4(c)(d): 即使经过 data packing, VLM 和 T2V 的计算量仍然一批一批波动。
- 原因是 packing 主要平衡总长度, 但不能保证 image encoder、backbone、DiT 各自的计算量都平衡。

这就是 training data dynamicity: 训练数据本身让每一轮的最优调度都不一样。

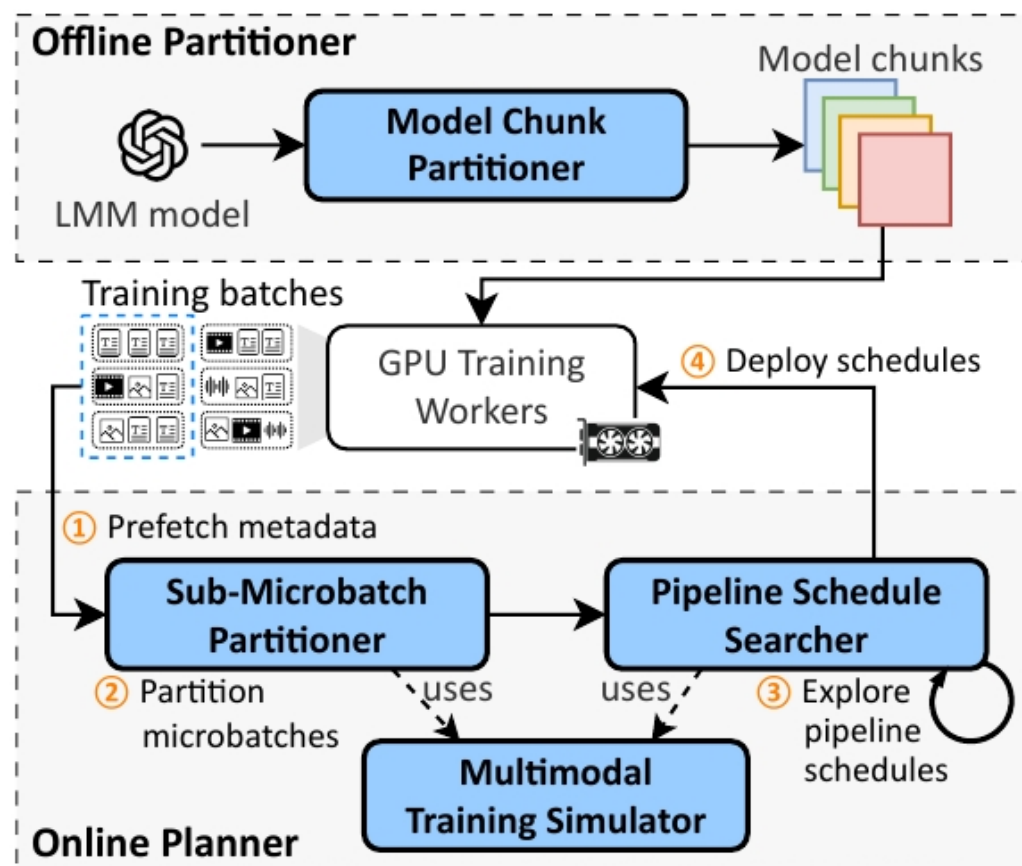
为什么已有方法不够用

传统 LLM 训练系统通常假设：虽然不同 microbatch 的长度可能不同，但这种变化会比较均匀地影响所有 Transformer 层。

- 但在 LMM 中，图片变多主要影响 image encoder，文本变多主要影响 backbone，视频变长可能影响 DiT。
- 因此，单纯做 data packing 或 microbatch reordering，不能解决“某个模态模块被单独压垮”的问题。
- Megatron-LM、nnScaler、Optimus 等系统要么使用静态 schedule，要么主要针对固定任务/固定结构，难以适应每个 batch 都变化的 LMM。

关键矛盾：LMM 的 workload 是按模态变化的，但传统 pipeline schedule 往往不是按模态感知的。

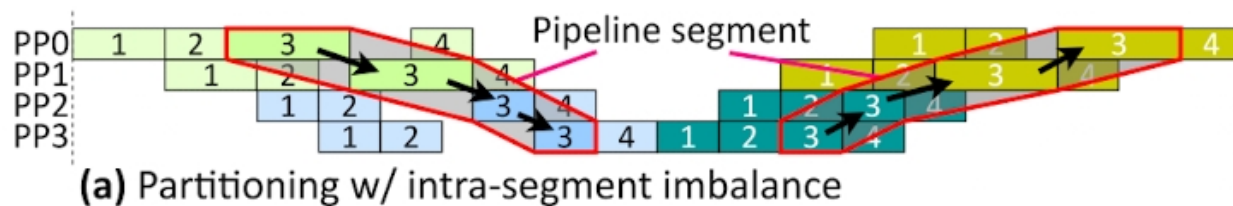
DIP 总体流程：一个训练时的“调度员”



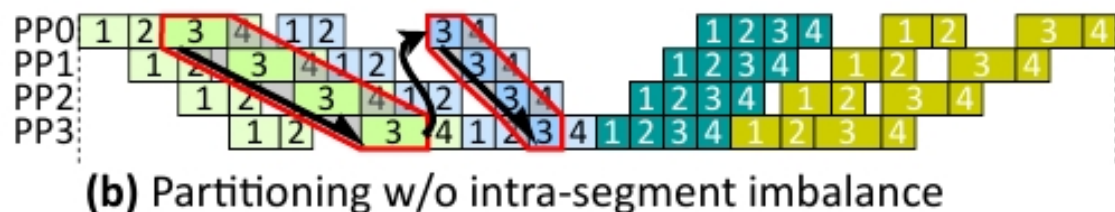
- Offline Partitioner: 训练前先把 LMM 模型切成多个 model chunks，放到不同 GPU 上。
- Metadata Prefetch: 训练时提前看下一批数据，例如图片数、token 数、视频长度。
- Sub-Microbatch Partitioner: 根据当前 batch 的模态组成，把数据切成更细的模态小批次。
- Pipeline Schedule Searcher: 用 simulator 估计每个 stage 的时间和显存，再搜索 schedule。
- Deploy schedules: 把最终 schedule 下发给 GPU workers 执行。

方法 1: Separated Partitioning

第一步是解决 intra-segment imbalance: 不要把不同模态的计算混在同一个 pipeline segment 里。



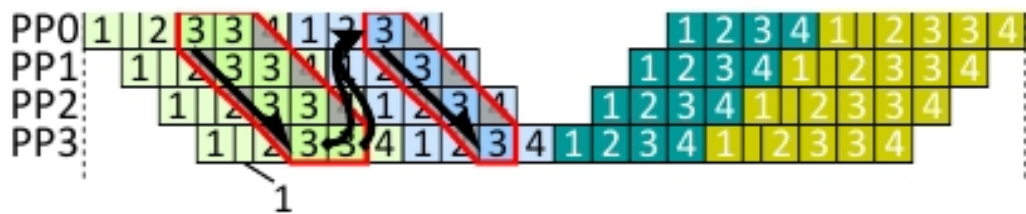
上图 Figure 5(a): 不同模态被混在一起。某些 rank 做图像编码器，某些 rank 做 backbone，它们用时不同，所以等待不可避免。



下图 Figure 5(b): DIP 把不同模态拆到不同 pipeline segment。这样每个 segment 内部更同质，先去掉段内不均衡。

方法 2: Modality-aware Sub-microbatch

第二步是解决 inter-segment imbalance: 不同模态 segment 之间仍然可能一快一慢, 所以 DIP 继续把数据按模态切得更细。



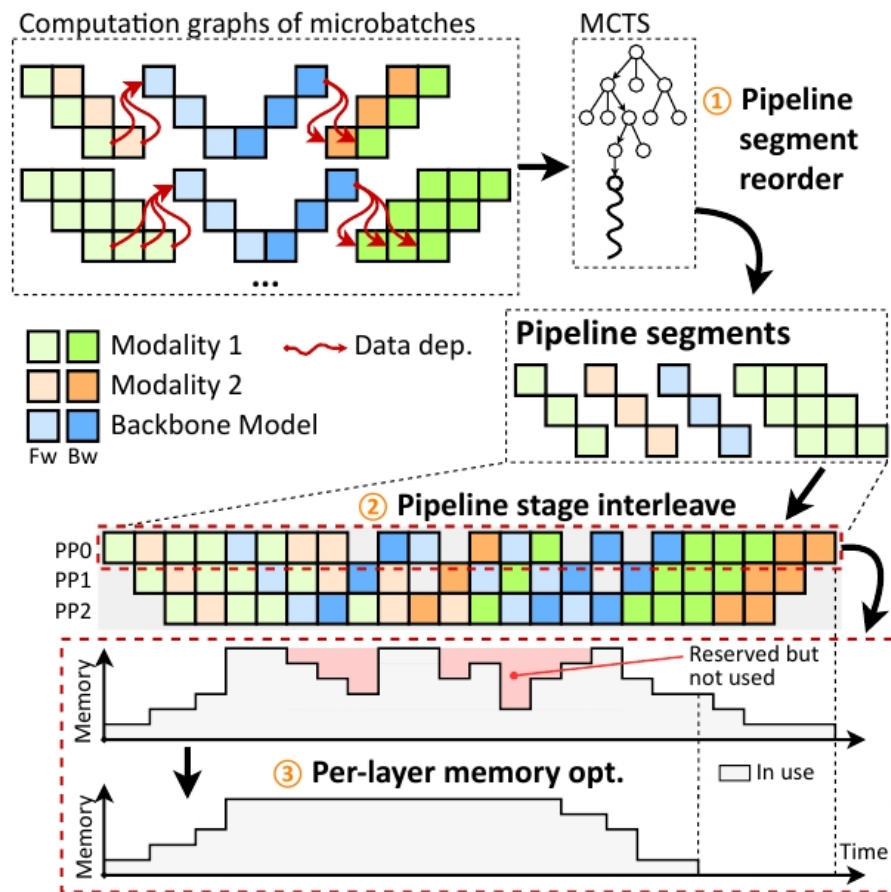
(c) Modality-aware partitioning

Figure 5(c) 的意思是: 如果这一批图片很多, 图像编码器会很慢。DIP 不让它一次处理全部图片, 而是拆成多个图像小批次。

每个小块更短, 调度器就能把图像部分和语言模型部分穿插起来, 减少 stage 之间互相等待。

通俗说: 把“大块慢活”切成“小块可穿插的活”。

方法 3: Pipeline Schedule Searcher



- ① Pipeline segment reorder: 先决定这些 segment 谁先执行。DIP 用 MCTS 搜索较好的顺序。
- ② Pipeline stage interleave: 顺序确定后, 把 forward 和 backward stage 尽量交错排紧, 减少空泡。
- ③ Per-layer memory optimization: 在不超过显存上限的前提下, 选择 checkpointing/offloading 等策略。

重点: DIP 不是暴力求全局最优, 而是在一次训练迭代能容忍的时间里找到足够好的 schedule。

DIP 如何在线搜索而不拖慢训练

如果每个 batch 都要重新搜索 schedule，最直接的问题是：搜索会不会比训练还慢？

DIP 的做法是把搜索放到 CPU 上异步执行。

- GPU 正在训练当前 batch 时，CPU 同时为下一个 batch 搜 schedule。
- 搜索过程依赖 simulator，而不是每个候选 schedule 都真实跑一遍 GPU。
- 复杂搜索被拆成三个阶段：segment reorder、stage interleave、memory optimization。
- 这样可以把搜索时间隐藏在训练过程中，尽量不进入关键路径。

可以理解成：GPU 在干活，CPU 在提前给下一批任务排队。

Implementation & Evaluation Setup

Table 2. Model specifications used in the evaluation.

Name	# of Layers	Embed Dim	FFN Hidden Dim	# of Attn. Heads	# of Attn. Groups
ViT 5B [12]	63	1792	15360	16	16
ViT 22B [12]	48	6144	24576	48	48
Llama3 8B [17]	32	4096	14336	32	8
Qwen2 32B [51]	64	5120	27648	40	8
Qwen2 72B [51]	80	8192	29568	64	8
DiT 5B [35]	28	3584	10240	28	28
DiT 30B [35]	48	6144	24576	48	48

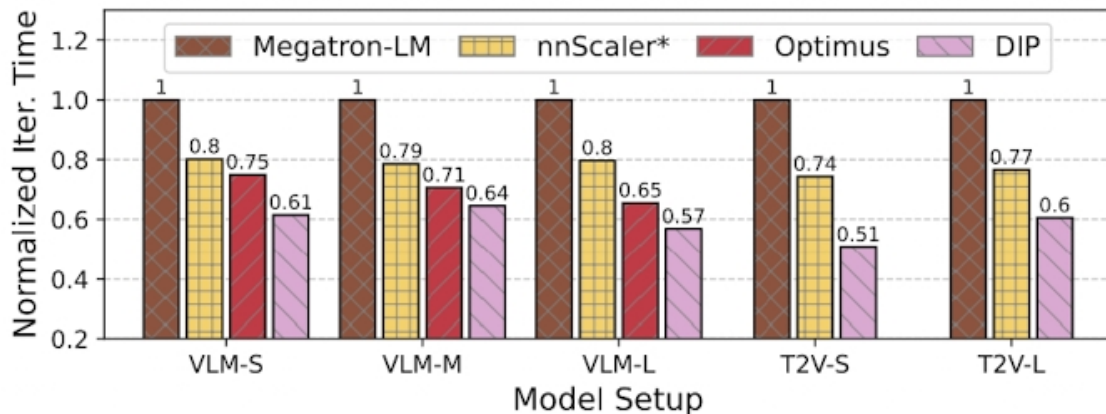
- DIP 基于 Megatron-LM 实现，扩展了 schedules module，让 worker 可以执行动态 action list。

- Simulator 用 operator-level cost model 估计计算、访存和通信耗时。

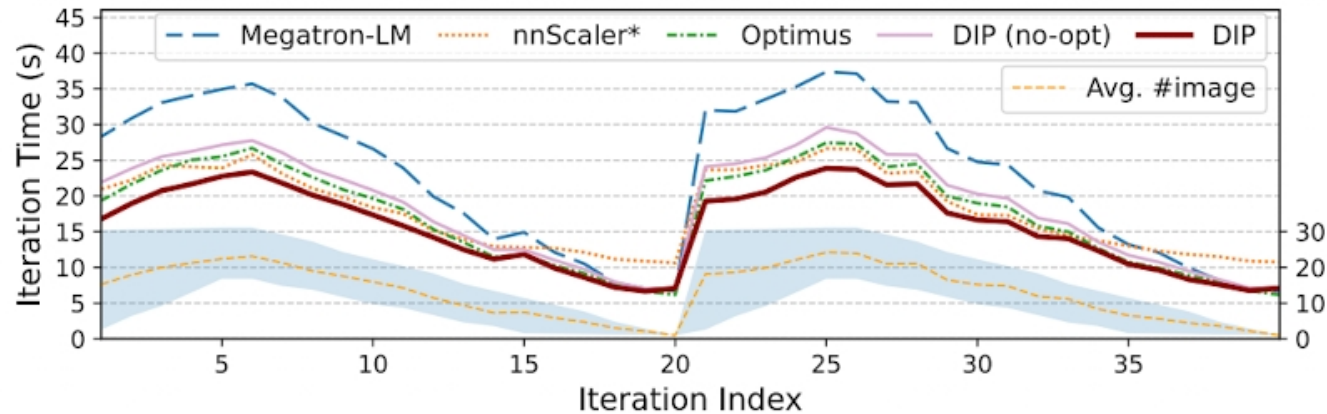
- 评估覆盖 VLM 和 T2V 两类模型，规模从 12B 到 94B。

- 真实实验使用 H800/H20 集群；超大 H100 场景使用校准后的模拟器。

End-to-End Performance: 整体训练效果



(a) Average end-to-end performance



(b) Dynamic workloads

Table 4. VLM-S end-to-end performance of FSDP, Megatron-LM and DIP on 16 H20 GPUs.

	FSDP [55]	Megatron-LM [39]	DIP
Iteration time (s)	40.270	39.053	28.606
Relative time	1.03	1.00	0.73

- Figure 8(a): 五组模型上，DIP 的 normalized iteration time 都最低。
- Figure 8(b): 图片数量上升时，Megatron-LM 明显变慢；DIP 曲线更低也更稳。
- Table 4: 16 张 H20 上，DIP 把 VLM-S iteration time 从 39.053s 降到 28.606s。

Ablation: 每个优化分别贡献了什么

Table 5. Quantitative impact of DIP's optimizations.

Techniques	Iter. Time (s)	$\Delta\%$
Vanilla Megatron-LM	26.13	0.0%
+ Modality-aware partitioner (§4)	22.27	17.3%
+ Pipeline stage interleaving (§5.2)	18.81	38.9%
+ Pipeline segment reordering (§5.1)	17.61	48.3%
+ Pre-layer memory optimization (§5.3)	16.05	62.8%

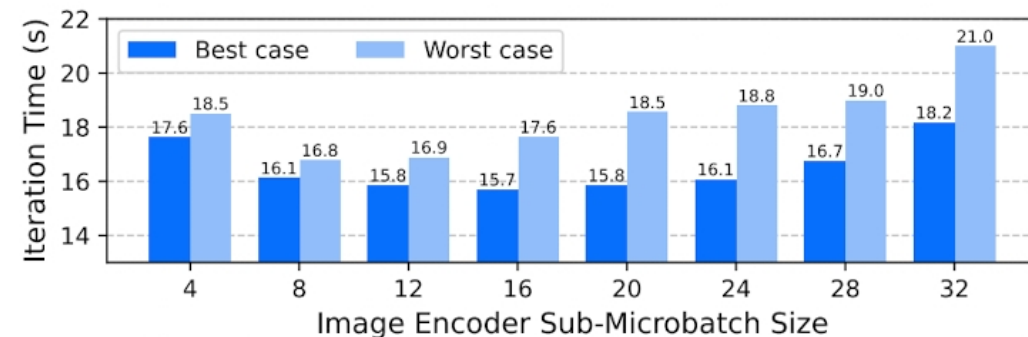
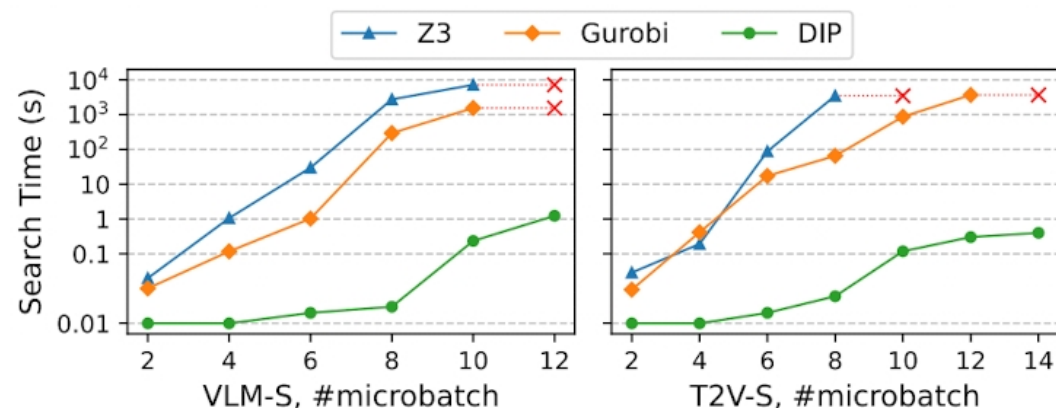
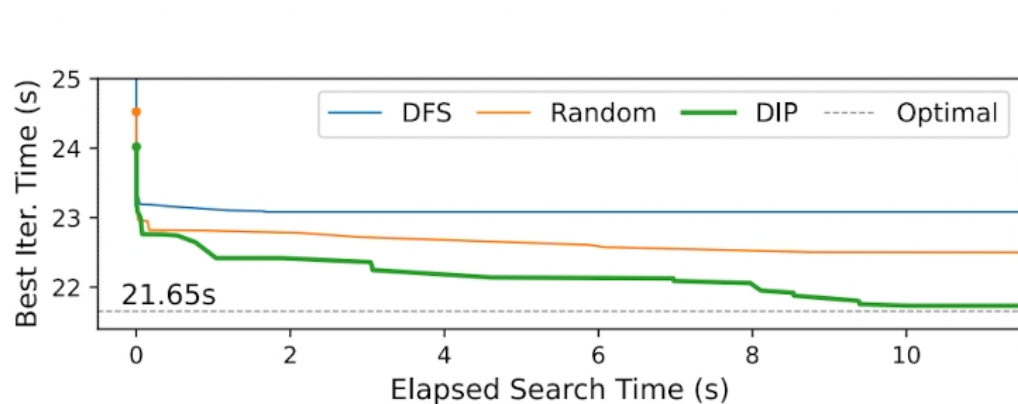


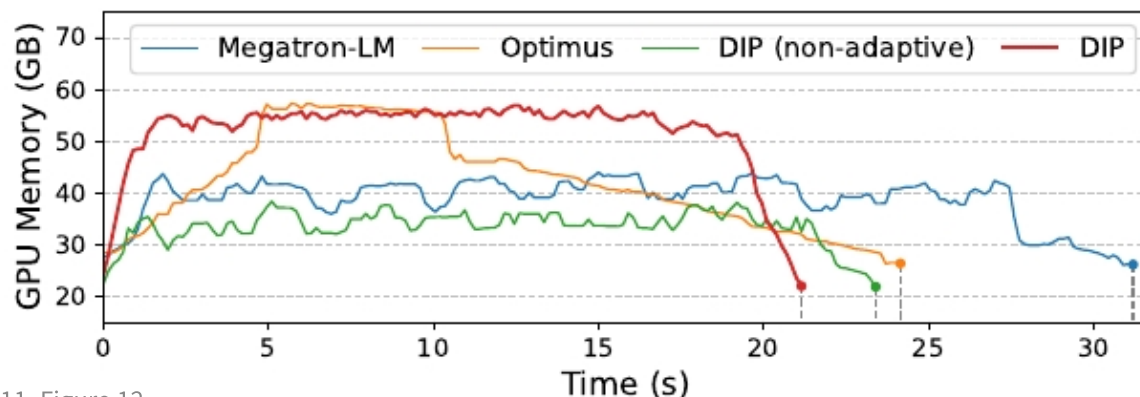
Figure 9. Impact of sub-microbatch sizes.

- 只加入 modality-aware partitioner, iteration time 从 26.13s 降到 22.27s, 说明按模态拆分非常关键。
- 加入 pipeline stage interleaving 后降到 18.81s, 说明搜索执行顺序也很重要。
- Figure 9 说明 sub-microbatch 不是越小越好: 太小会让 GPU 利用率下降, 所以需要折中。

Planner Evaluation: 搜索本身是否可用



左上：DIP 的 MCTS 比 random/DFS 更快找到好 schedule。右上：Z3/Gurobi 随 microbatch 数增大很快超时，而 DIP 仍保持可用。下方：DIP 的内存曲线更平稳，说明细粒度调度也帮助控制显存波动。



Original paper: Figure 10, Figure 11, Figure 12

Conclusion:

Table 6. Model combinations used in the large-scale simulation.

Name	Model Setup	DP	TP	PP	#GPU
VLM-XL-8k	ViT 22B + GPT 175B	128	8	8	8192
VLM-XL-16k	ViT 22B + GPT 175B	128	8	16	16384
T2V-XL-3k	Qwen2 72B + DiT 30B	96	8	4	3072
T2V-XL-6k	Qwen2 72B + DiT 30B	96	8	8	6144

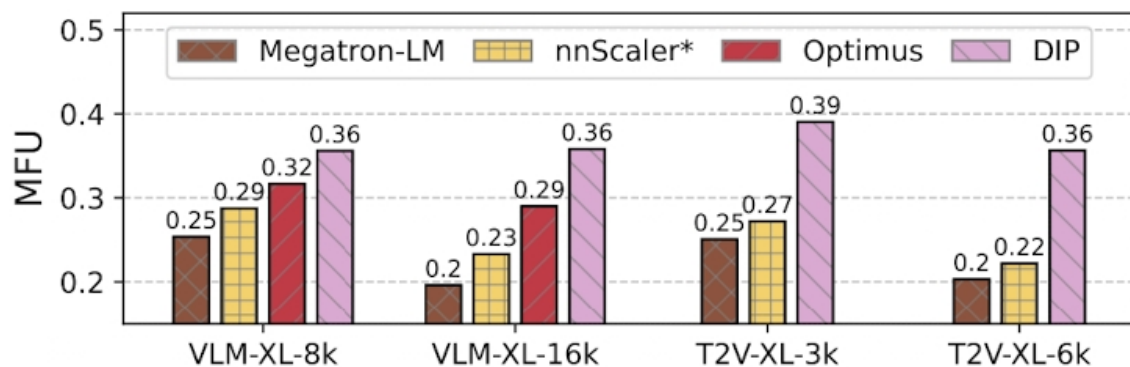


Figure 14. Large scale simulations on H100 clusters.

DIP 的核心贡献不是提出新的多模态模型，而是提出了一套面向 LMM 训练**动态负载**的 pipeline **调度系统**。它把多模态训练中的问题从“如何静态切模型”转化为“如何根据当前 batch 的模态组成动态排流水线”，从而有效减少 pipeline bubble，提高大规模多模态模型训练效率。