

FlowTrain: Flow-Based Decoupled Training for Industrial-Grade Vision-Language Models

Zhida Jiang¹, Zhaolong Xing^{1*}, Yang Pei¹, Xiaolong Chen², Yuanhang Xiao², Chengzhi Huang¹, Xiyu Liu¹, Haopeng Liu², Qingyuan Sang¹, Lingfeng Zhou¹, Jiaxing Wang¹, Zicheng Zhang¹, Wenzhe Wang², Xinyu Liu², Yan Li¹, Zhen Chen^{1*}, Ke Zhang¹

¹JD.com

²Huawei

{xingzhaolong1, chenzhen48}@jd.com

Abstract

Industrial-grade distributed training of vision-language models (VLMs) remains far less efficient than that of unimodal LLMs. Existing solutions either follow a monolithic design that assigns uniform parallelism to heterogeneous modules or adopt a disaggregated deployment that separates modules while executing them as a batch-synchronized pipeline. In this paper, we highlight that the above solutions are still not sufficient, and VLM training can be further *decoupled*. To this end, we present FlowTrain, a flow-based decoupled training framework that reformulates VLM training as a producer-consumer dataflow coordinated through a *unified memory pool*. The encoder and backbone can progress independently over a global virtual address space. Since this execution decoupling fundamentally changes the optimization objective of allocation and scheduling, FlowTrain further introduces a *heterogeneous parallel allocator* that assigns module-specific parallelism strategies by solving a throughput matching problem. The *dynamic packing scheduler* is used to construct balanced microbatches at runtime according to the actual LLM-side computation cost. Extensive experiments on real-world workloads show that FlowTrain achieves over 50% MFU and up to $1.7\times$ throughput improvement, narrowing the efficiency gap to LLM-only training.

1 Introduction

Vision-language models (VLMs) extend the reasoning capabilities of LLMs by connecting visual content with its linguistic context (Wang et al., 2025a; Zhu et al., 2025; Laskar et al., 2025). Such multimodal learning capabilities have achieved remarkable progress in industrial applications, such as autonomous driving (Li et al., 2025a), embodied agents (Lu et al., 2026), and visual question answering (Wang et al., 2025c). Although model

capabilities have advanced rapidly, training efficiency remains a major bottleneck that limits industrial deployment of VLMs. VLM training achieves much lower Model FLOPs Utilization (MFU) than unimodal LLM training, even with massive investments in AI accelerators (Xue et al., 2026a).

Distributed training paradigms for multimodal models including VLMs can be broadly classified into two categories. (i) Monolithic frameworks, exemplified by Megatron-LM (Shoeybi et al., 2019) and its multimodal extensions, treat a VLM as a single network with heterogeneous layers and impose a uniform tensor/pipeline parallelism (TP/PP) strategy across all modules. They ignore the fact that VLMs are structurally heterogeneous, including modality encoders, projectors, and a large LLM backbone (Feng et al., 2025). These modules exhibit drastically different arithmetic intensities and preferred parallelism configurations. The monolithic design thus leads to poor resource utilization and prolongs training duration (Zhang et al., 2025).

(ii) Recent disaggregated frameworks represented by DistTrain (Zhang et al., 2025) recognize the above architectural asymmetry and separate different modules into independent device groups with customized parallelism strategies. However, we identify that disaggregated training is still regarded as a rigid pipeline, where all the data parallelism (DP) replicas of encoders must materialize the entire global batch before the LLM backbone can start computation. The encoder outputs are synchronized with the LLM backbone via *point-to-point* transfer. Such batch-level synchronization barriers are exacerbated by dynamic variability of multimodal inputs, ultimately limiting end-to-end training efficiency (Xue et al., 2026b).

Our key insight is that disaggregation is necessary but insufficient to unlock training efficiency. Instead of a separated-but-synchronized pipeline, an efficient training framework should achieve *both spatial disaggregated deployment and temporal de-*

*Corresponding Authors.

coupled execution, so that the encoders and the backbone progress as independent producers and consumers rather than as upstream and downstream pipeline stages. Motivated by this, we propose FlowTrain, a flow-based decoupled VLM training framework with three core designs. As the foundation of FlowTrain, the *unified memory pool* unifies heterogeneous memory resources (HBM and DRAM) across all nodes through 64-bit global virtual address (GVA). All frozen encoders write the embeddings into the pool and immediately proceed to the next input. Based on the GVA and page tables, the backbone consumes embeddings from the pool through transparent memory access. VLM training is thus converted from a rigid pipeline into a continuous producer-consumer workflow.

Once execution is decoupled through the unified memory pool, resource allocation and batch scheduling of VLM training should be reconsidered. On the one hand, the optimization objective for resource allocation is no longer minimizing the worst pipeline stage, but balancing the throughput between different modules. FlowTrain introduces a *heterogeneous parallel allocator* that determines module-specific parallelism strategies and resource partitions using a flow matching formulation. On the other hand, since the unified memory pool absorbs runtime variance in encoder latency, batch scheduling should be decided by the actual computation cost after encoding, instead of raw sample size. To construct balanced microbatches, FlowTrain designs a *dynamic packing scheduler* that minimizes padding waste through segment tree-based best-fit packing and jointly balances loads across PP and DP dimensions via zig-zag dispatch. The main contributions of this work are as follows:

- We identify disaggregation alone as insufficient due to batch-level synchronization barriers, and propose a decoupled FlowTrain framework that rethinks VLM training from rigid pipeline execution to continuous producer-consumer dataflow.
- We design three coordinated components to realize decoupled execution, i.e., unified memory pool for asynchronous exchange, heterogeneous parallel allocator for flow matching, dynamic packing scheduler for workload balancing.
- We implement FlowTrain on Ray and evaluate it on real-world workloads across three VLM scales. FlowTrain consistently achieves over 50% MFU and up to $1.7\times$ throughput improvement, closing the efficiency gap to LLM-only training.

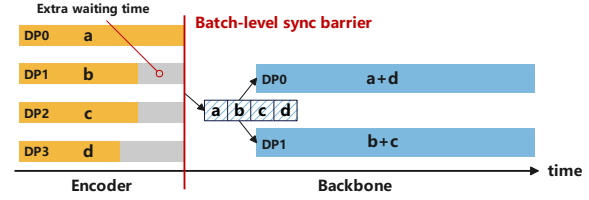


Figure 1: Batch-level synchronization barriers under disaggregated training.

2 Related Work

Compared with unimodal LLM training, optimizing VLM training is more difficult due to its inherent heterogeneity and dynamics. Driven by continuous scaling of industrial VLMs combined with trillion-level datasets, distributed training on large-scale clusters has become imperative. Early systems such as DistMM (Huang et al., 2024) can handle traditional multimodal training efficiently, but do not support LLM training naively. Although frameworks like Megatron-LM (Shoeybi et al., 2019; Rasley et al., 2020; Zhao et al., 2023) excel at LLM workloads, they treat VLMs as monolithic models with heterogeneous layers, resulting in suboptimal resource allocation.

Other studies (Feng et al., 2025; Wang et al., 2025b) adopt fine-grained bubble exploitation or pipeline scheduling algorithms to improve resource utilization. Following this idea, LongCat-Flash-Omni (Team, 2025a) and MegaScale-Omni (Xue et al., 2026a) further reduce engineering complexity. However, colocated schemes may suffer from the long-tail effect, resulting in resource conflicts and additional onload/offload overhead. Disaggregated systems represented by DistTrain (Zhang et al., 2025) deploy heterogeneous modules on separate resource groups, addressing model and data heterogeneity.

Limitation of Existing Methods. However, the above frameworks still retain batch-level synchronization barriers between heterogeneous modules, failing to break the *rigid pipeline execution* paradigm. Even disaggregated training separates the modules spatially but does not decouple their execution temporally. All encoder DP replicas have to fully materialize their output and align at batch boundaries, as illustrated in Figure 1. Only after the slowest encoder replica completes its global batch can the embeddings be synchronously point-to-point transferred to the backbone (Lin et al., 2025). Strict data dependency produces idle periods on both sides whenever their latencies diverge.

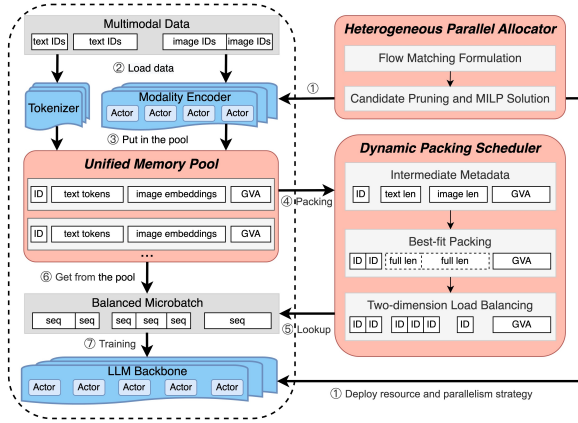


Figure 2: Overview of FlowTrain.

More importantly, industrial multimodal samples exhibit obvious variability in resolution and sequence length. Such variance further magnifies the performance penalty of batch-level synchronization barriers and induces a severe workload imbalance between distributed accelerators. Existing sample scheduling only considers raw input characteristics (e.g., image resolution) rather than actual computational costs after encoding, which also depend on parallelism configuration and runtime conditions (Zhang et al., 2025; Xue et al., 2026b).

3 Methodology

3.1 Overview of FlowTrain

To overcome the limitations of monolithic and disaggregated frameworks, we propose a flow-based VLM training framework, called FlowTrain. The design principle is disaggregated in deployment and decoupled in execution. Instead of treating heterogeneous modules as pipeline stages synchronized at batch boundaries, FlowTrain reformulates training as a continuous producer-consumer dataflow, enabling each component to progress independently at its native throughput.

As illustrated in Figure 2, FlowTrain contains three coordinated components. The end-to-end workflow is as follows. Before training starts, the *heterogeneous parallel allocator* (§3.3) partitions the cluster into separate actor pools (encoder/LLM) that execute with independent parallelism configurations. During training, the frozen encoders process multimodal input asynchronously and place embeddings into the *unified memory pool* (§3.2), while text tokens are inserted directly into the same pool. The lightweight metadata is also stored to track cross-modal pairing information. After sufficient paired embeddings accumulate in the pool,

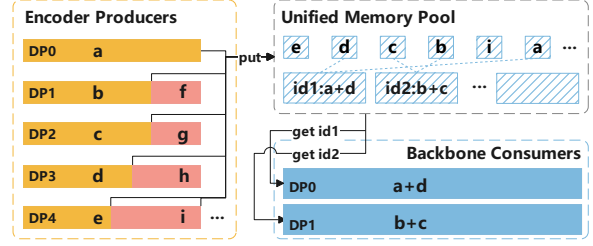


Figure 3: Decoupled producer-consumer dataflow in FlowTrain.

the *dynamic packing scheduler* (§3.4) leverages the metadata to construct computationally balanced microbatches. The LLM backbone continuously consumes microbatches based on the GVA without being aware of their actual storage location, improving resource utilization and training throughput.

3.2 Unified Memory Pool

As shown in Figure 3, the unified memory pool serves as the asynchronous coordination layer between encoder producers and backbone consumers. As the foundation of FlowTrain, it breaks strict data dependency by converting point-to-point batch transfer into globally addressable buffered exchange, which logically unifies heterogeneous memory resources (HBM and DRAM) across all nodes. Each memory location, regardless of its physical location or type, is assigned a unique 64-bit GVA. As illustrated in Figure 4, the GVA space reserves two disjoint contiguous regions corresponding to HBM and DRAM, respectively. Each region provides a GVA interval $[start, start + world_size * max_size]$, where $start$ is a globally agreed address base, $world_size$ is the number of total processes, and max_size is the contributed maximum memory capacity. The address space is linearly divided among all processes and visible to all processes. The GVA segment owned by i -th process is $[start + i * max_size, start + (i + 1) * max_size]$. This abstraction provides a single, consistent memory view to all training processes, allowing transparent access to remote memory as if it were local.

A logical GVA is insufficient for direct data movement because training processes ultimately access memory through a physical address. Similar to virtual memory management in modern operating systems, our pool maintains the page tables that map GVA used by processes to physical addresses in HBM and DRAM. The key difference is that the virtual address exposed to each process is no longer a small local VA region, but a globally consis-

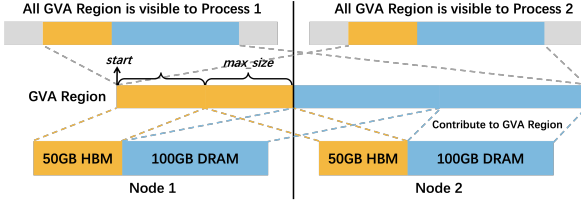


Figure 4: GVA region in the unified memory pool.

tent GVA region spanning pooled memory across nodes and memory tiers. During initialization of the memory pool before training, each process reserves a contiguous GVA region from the segment $[start + i * max_size, start + (i + 1) * max_size)$, allocates physical memory from the local operating system, and binds it to the reserved GVA region, which may be less than max_size . Each process exports its memory mapping information to all other processes, which update their page table accordingly. Then, each process has local access to the GVA and the page tables used for virtual address translation, eliminating distributed address resolution protocols on the hot path.

During training, the vision embeddings and text tokens are written into the GVA segment belonging to their own training process. LLM consumes the constructed microbatches directly from GVAs without being aware of their actual storage location. In the implementation layer, FlowTrain resolves the GVA to the corresponding physical address through the collected page tables. The available transport engine is selected based on the source and destination location and memory types. If the request crosses memory tiers within a node, FlowTrain invokes intra-node transfer engine like NVLink and HCCS. For cross-node destination, FlowTrain establishes (or reuses) transport endpoints and dispatches the transfer through an RDMA-class or other high-speed interconnect backend. The upper layers that operate on the GVA region are decoupled from hardware-specific addressing and transport details, enabling transparent memory access.

3.3 Heterogeneous Parallel Allocator

Existing resource allocation schemes (Zhang et al., 2025) aim to optimize the worst pipeline stages. After integrating the unified memory pool, our optimization objective has been changed to optimize the achievable throughput while avoiding producer-consumer imbalance under a fixed resource budget R . Let $\mathcal{E}$ denote the set of encoder modules and b denote the backbone. For each module $i \in \mathcal{E} \cup \{b\}$,

$\mathcal{C}_i$ defines its set of feasible parallelism configurations. Each configuration $c \in \mathcal{C}_i$ has a throughput $\tau_i(c)$ and resource cost $r_i(c)$. We use the binary variable $x_{i,c} \in \{0, 1\}$ to indicate whether c is selected. The aggregated encoder production rate and backbone consumption rate can be defined as:

$$T_{\text{enc}} = \sum_{e \in \mathcal{E}} \sum_{c \in \mathcal{C}_e} x_{e,c} \tau_e(c), \quad T_{\text{bb}} = \sum_{c \in \mathcal{C}_b} x_{b,c} \tau_b(c) \quad (1)$$

Then, flow matching problem is formalized as:

$$\begin{aligned} \text{lexmax} \quad & (G, -F) \\ \text{s.t.} \quad & G = \min(T_{\text{enc}}, T_{\text{bb}}), \\ & F = |T_{\text{enc}} - T_{\text{bb}}|, \\ & \sum_{c \in \mathcal{C}_i} x_{i,c} = 1, \quad \forall i \in \mathcal{E} \cup \{b\}, \\ & \sum_{i \in \mathcal{E} \cup \{b\}} \sum_{c \in \mathcal{C}_i} x_{i,c} r_i(c) \leq R. \end{aligned} \quad (2)$$

The objectives are optimized in lexicographic order. We first maximize the global throughput G , and then minimize flow imbalance F between producers and consumers. Over-allocating resources to the encoder incurs redundant buffering in the pool, while under-allocating starves the backbone.

To efficiently solve the problem in Eq. (2), we first prune the candidate space $\mathcal{C}_i$ while retaining practically deployable configurations. Since $\tau_i(c)$ depends on the combined effect of operator efficiency and communication overhead, we estimate a surrogate of $\tau_i(c)$ through lightweight profiling, which is the established methodology in parallelism planning (Um et al., 2024; Miao et al., 2022). Substituting the surrogates into Eq. (2) yields a Mixed-Integer Linear Programming (MILP) problem, which can be efficiently solved by off-the-shelf solvers (e.g., Gurobi) in milliseconds. Then, the obtained resource partition and module-specific parallelism strategies are materialized by Ray (Moritz et al., 2018) as a set of actors.

3.4 Dynamic Packing Scheduler

Similar to the allocator, the decoupled execution paradigm also reshapes the design space and scheduling objective. Regardless of monolithic or disaggregated design, microbatch composition is usually performed before multimodal encoding and only relies on raw sample size. In comparison, our unified memory pool absorbs runtime variance, preventing it from propagating into the backbone. The dynamic packing scheduler can flexibly assemble balanced microbatches from available embeddings in the pool based on the actual computational cost, rather than committing to fixed microbatch compositions before training begins. As summarized

Algorithm 1: Dynamic Packing Algorithm

```

/* Best-fit Packing */
1 Initialize segment tree  $\mathcal{T}$  over  $[0, L]$ ;
2 for Each sequence  $s$  with length  $\ell(s)$  do
3    $j \leftarrow \text{BESTFITQUERY}(\mathcal{T}, \ell(s))$ ;
4   if  $j \neq \text{null}$  then
5     Append  $s$  to microbatch  $B_j$ ;
6      $P_j \leftarrow P_j - \ell(s)$  and update  $\mathcal{T}$ ;
7   else
8     Create  $B_j$  with  $P_j \leftarrow L - \ell(s)$ ;
9     Insert into  $\mathcal{T}$ ;
10  if  $P_j < \ell_{\min}$  then
11    Remove  $B_j$  from  $\mathcal{T}$ ;

/* Two-dimension Load Balancing */
12 for Each packed microbatch  $B_j$  do
13   Compute  $\text{Cost}(B_j)$ ;
14   Assign  $B_j$  to bucket  $k$  by clustering  $\text{Cost}(B_j)$ ;
15 for Each bucket  $k$  do
16   Obtain  $\text{list}$  by sorting microbatches in
    descending order of  $\text{Cost}(B_j)$ ;
17   Initialize  $a \leftarrow 0$ , Forward  $\leftarrow \text{True}$ ;
18   while  $a < \text{len}(\text{list})$  do
19     if Forward then
20       for  $n = 0$  to  $N - 1$  do
21         Assign  $\text{list}[a]$  to  $\text{Rank}_n$ ;
22          $a \leftarrow a + 1$ ;
23     else
24       for  $n = N - 1$  to  $0$  do
25         Assign  $\text{list}[a]$  to  $\text{Rank}_n$ ;
26          $a \leftarrow a + 1$ ;
27   Forward  $\leftarrow \text{not Forward}$ ;

```

in Algorithm 1, the scheduler minimizes padding waste through best-fit packing, and then relieves PP and DP imbalance via zig-zag dispatch.

We utilize the segment tree-based algorithm to pack variable-length sequences in fixed-capacity microbatches, thereby reducing memory fragmentation (Ding et al., 2024). A segment tree covers the integer interval $[0, L]$, where L is the maximum microbatch sequence length (e.g., 8192). Each leaf corresponds to the residual capacity of the candidate microbatches, and each internal node stores the maximum residual capacity in its subtree (Line 1). For each incoming sequence s with length $\ell(s)$, the scheduler first examines the left subtree, which

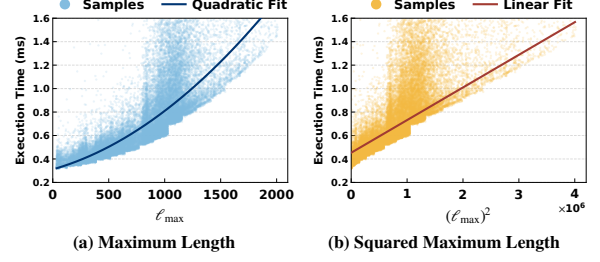


Figure 5: Execution cost modeling with maximum sub-sequence length $\ell_{\max}$.

represents smaller residual capacities. If its residual capacity is $\geq \ell(s)$, the search recursively descends into the left subtree. Otherwise, it proceeds to the right subtree. This process locates the open microbatch whose residual capacity is $\geq \ell(s)$ and minimal among all candidates by $O(\log L)$ best-fit queries (Lines 2-3). If a suitable microbatch B_j is found, append incoming sequence s to B_j and update its residual capacity to $P_j \leftarrow P_j - \ell(s)$ (Lines 4-6). Otherwise, create a new microbatch with residual capacity $L - \ell(s)$ and insert it into the tree (Lines 7-9). When a microbatch’s residual capacity falls below the minimum sequence length in the current distribution, it is removed from the segment tree (Lines 10-11).

Then, packed microbatches are assigned to PP stages and DP ranks to relieve imbalance across both dimensions. To characterize execution costs, we profile packed microbatches with different sequence compositions. We observe from Figure 5 that the execution time is approximately linear in $(\ell_{\max})^2$, where $\ell_{\max} = \max_{s \in B_j} \ell(s)$ is the maximum sub-sequence length within a packed microbatch B_j . This observation is consistent with the quadratic scaling of the attention kernel (Dao, 2024). Meanwhile, the execution time of MLP layers scales approximately linearly with the total number of valid tokens. Based on these observations, we estimate the execution cost of B_j after maximum microbatch length (L) normalization:

$$\text{Cost}(B_j) = \frac{1}{L} \sum_{s \in B_j} \ell(s) + \frac{1}{L^2} (\ell_{\max})^2 \quad (3)$$

Based on $\text{Cost}(B_j)$ capturing both linear and quadratic computations, we cluster packed microbatches with similar computational intensities into a discrete bucket, ensuring consistent execution latencies across PP stages (Lines 12-14). Within each bucket, the microbatches are sorted in descending order of $\text{Cost}(B_j)$ (Lines 15-16). The sorted micro-

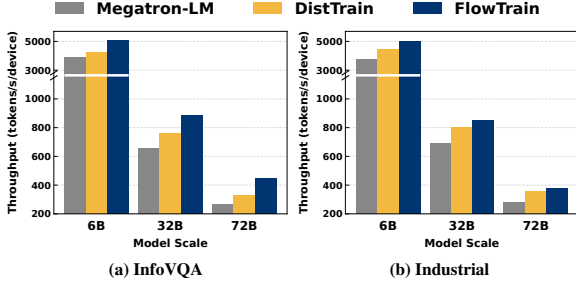


Figure 6: Training throughput across different methods on varying model sizes.

batches are dispatched in order $\text{Rank}_0 \rightarrow \text{Rank}_{N-1}$ (Lines 17-22), and then the direction of the traversal is reversed to $\text{Rank}_{N-1} \rightarrow \text{Rank}_0$ (Lines 23-27). This zig-zag dispatch ensures that high and low-cost microbatches are paired on DP ranks, forcing the total cost to converge across the cluster (Abdelhamid et al., 2020).

4 Experiments

4.1 Experimental Setup

Models and Datasets. We adopt a representative VLM architecture, where the vision encoder adopts the ViT architecture (Dosovitskiy et al., 2021), and the LLM backbone is based on Qwen3 (Team, 2025b). We train three model scales spanning small, medium, and large configurations, i.e., VLM-6B, VLM-32B, and VLM-72B. We evaluate FlowTrain on *InfoVQA* (Mathew et al., 2022), an open-source visual question answering dataset, and *Industrial* dataset collected from a real-world e-commerce platform.

Training Configurations. All experiments are conducted on a production cluster with 256 Ascend NPUs (64GB) (Zhou et al., 2025). FlowTrain uses Ray to orchestrate heterogeneous actors and enable independent execution between the encoder and the LLM backbone. The image and text subsequences are interleaved to form sequences of up to 8192, and the global batch size is set to 256.

Baselines. (1) *Megatron-LM* (Shoeybi et al., 2019) provides mature optimizations for LLM training and treats the multimodal model as a monolithic network. (2) *DistTrain* (Zhang et al., 2025) adopts disaggregated model orchestration and data preprocessing, which deploy heterogeneous modules with independent resources and parallelism configurations. We also report the LLM-only performance as a reference upper bound for unimodal training efficiency. For fair comparison, all meth-

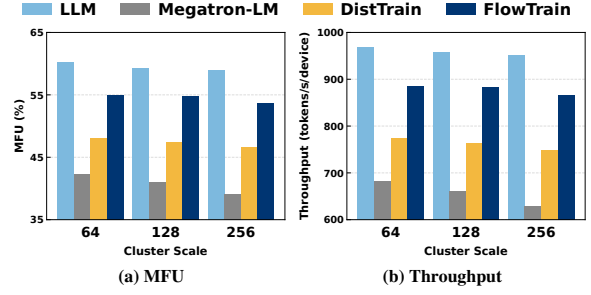


Figure 7: The efficiency gap compared to LLM-only training at different cluster scales.

ods are evaluated under the same software environment, optimizer configuration, and data preprocessing pipeline.

4.2 End-to-end Effectiveness

We evaluate the end-to-end training performance of FlowTrain with three model scales. The results in Figure 6 show that FlowTrain consistently achieves the highest throughput among all training systems. For the *InfoVQA* dataset, FlowTrain achieves $1.3\times$ average and $1.7\times$ maximum throughput improvement over both baselines. For the *Industrial* dataset, FlowTrain achieves $1.2\times$ average and $1.4\times$ maximum throughput improvement. Notably, the performance gap between FlowTrain and baselines increases as the models grow larger. FlowTrain outperforms Megatron-LM by 30.4% for VLM-6B, 33.7% for VLM-32B, and 69.1% for VLM-72B on the *InfoVQA* dataset. This is because the baselines suffer from batch-level synchronization barriers, leading to poor hardware utilization. In comparison, FlowTrain allows execution decoupling and coordinates heterogeneous modules through a shared memory abstraction, which substantially improves the training efficiency of large-scale VLMs.

4.3 Scaling Performance

We further evaluate scalability by varying the cluster scales. Figure 7 reports the results of training VLM-32B on the *InfoVQA* dataset, together with LLM-only training as a reference upper bound. FlowTrain achieves above 53% MFU and $1.1\text{--}1.4\times$ throughput improvement across all evaluation scales and exhibits the most robust scaling behavior, while the training efficiency of other methods is far lower than ours. Another noteworthy observation is that FlowTrain is closer to LLM-only training performance than baselines. The average MFU gaps are 18.7, 12.1, and 5.0 percentage points for Megatron-LM, DistTrain, and FlowTrain, respec-

Component	VLM-6B		VLM-32B		VLM-72B	
	MFU	Throughput	MFU	Throughput	MFU	Throughput
w/o Pool	21.6	2118	44.4	717	47.0	399
w/o Allocator	39.5	3879	50.1	809	48.9	415
w/o Scheduler	46.1	4531	49.1	793	47.2	400
FlowTrain	52.1	5125	54.8	884	52.9	448

Table 1: Ablation study of three core components on MFU (%) and throughput (tokens/s/device).

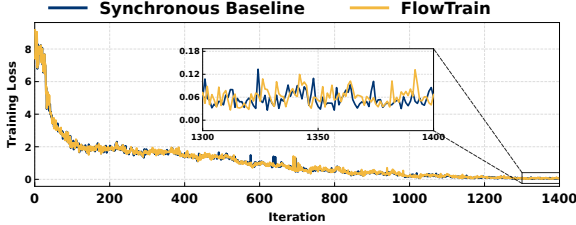


Figure 8: Training loss curves of FlowTrain and synchronous baseline.

tively. These results confirm that our design can significantly narrow the gap to the efficiency regime associated with unimodal training workloads.

4.4 Ablation Study

We conduct ablation experiments to quantify the individual contribution of three components in FlowTrain. As summarized in Table 1, removing the unified memory pool causes the largest degradation, reducing MFU and throughput by 15.6 percentage points and 29.5% on average, respectively. The system falls back to point-to-point batch synchronization, which reintroduces idle time and weakens the main benefit of decoupled execution. Without the heterogeneous parallel allocator, the training throughput decreases by up to 24.3%. Due to throughput mismatch between producers and consumers, the backbone is either starved of embeddings or overwhelmed by excessive production in the pool. Disabling the scheduler produces consistent performance loss. The MFU drops to 46.1%, 49.1%, and 47.2%, while the throughput decreases by 10.3-11.6%. The microbatches are no longer packed and dispatched according to actual computation cost, leading to padding waste and load imbalance. The ablation results demonstrate that three core components are essential and their combination delivers the maximum efficiency gain.

4.5 Training Consistency

We evaluate whether FlowTrain preserves the optimization behavior of synchronous training. Figure 8 shows that FlowTrain closely follows synchronous training throughout the entire training

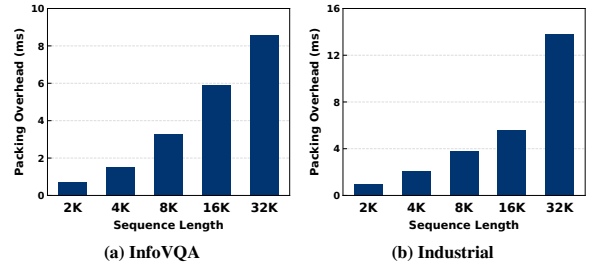


Figure 9: The average overhead of dynamic packing algorithm under different sequence lengths.

process on the InfoVQA dataset. Over the final 100 iterations, FlowTrain and baseline achieve average training losses of 0.0615 and 0.0553, respectively, corresponding to an absolute difference of only 0.0062. The reason is that FlowTrain only changes the embedding execution and scheduling of frozen encoders but does not introduce stale activations or cross-version gradients. These results indicate that FlowTrain preserves synchronous updates and does not materially alter the observed optimization trajectory.

4.6 Algorithm Overhead

The additional overhead introduced by dynamic packing algorithm is a major concern in practical industrial deployments. Figure 9 records that the average packing overhead for InfoVQA and Industrial datasets increases from 0.7-1.0ms to 8.6-13.8ms with sequence length varying from 2K to 32K. This upward trend is expected because longer sequences enlarge the candidate packing space and increase the cost of residual-capacity queries and load balancing. In particular, the Industrial dataset incurs larger algorithm overhead than the InfoVQA dataset due to its more severe data imbalance. Nevertheless, the overhead remains small compared with the forward and backward computation time. Even at 32K length, the packing overhead accounts for less than 0.1% of one training iteration.

5 Conclusion

In this paper, we propose FlowTrain, a disaggregated and decoupled VLM training framework that breaks batch-level synchronization barriers through three core designs: a unified memory pool that enables flexible producer-consumer collaboration, a heterogeneous parallel allocator for throughput matching, and a dynamic packing scheduler constructing balanced microbatches. Empirical evaluations on different models and scales demonstrate the efficiency of FlowTrain.

Limitations

While FlowTrain demonstrates consistent efficiency gains for VLM training, several limitations remain. FlowTrain targets training stages in which the modality encoder is frozen, while the projector and LLM backbone remain trainable. We emphasize that the frozen-encoder setting is the dominant practice for industrial VLM deployment. FlowTrain can be extended to trainable encoder settings. However, it prevents encoder run-ahead across optimizer steps and thus provides a smaller decoupled opportunity than the frozen-encoder setting. In the future, we plan to extend FlowTrain to support additional training settings, thereby broadening its industrial applicability.

References

- Ahmed S. Abdelhamid, Ahmed R. Mahmood, Anas Daghistani, and Walid G. Aref. 2020. [Prompt: Dynamic data-partitioning for distributed micro-batch stream processing systems](#). In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*, pages 2455–2469. ACM.
- Tri Dao. 2024. [Flashattention-2: Faster attention with better parallelism and work partitioning](#). In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.
- Hantian Ding, Zijian Wang, Giovanni Paolini, Varun Kumar, Anoop Deoras, Dan Roth, and Stefano Soatto. 2024. [Fewer truncations improve language modeling](#). In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, volume 235 of *Proceedings of Machine Learning Research*, pages 11030–11048. PMLR / OpenReview.net.
- Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2021. [An image is worth 16x16 words: Transformers for image recognition at scale](#). In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.
- Weiqi Feng, Yangrui Chen, Shaoyu Wang, Yanghua Peng, Haibin Lin, and Minlan Yu. 2025. [Optimus: Accelerating large-scale multi-modal LLM training by bubble exploitation](#). In *Proceedings of the 2025 USENIX Annual Technical Conference, USENIX ATC 2025, Boston, MA, USA, July 7-9, 2025*, pages 161–177. USENIX Association.
- Jun Huang, Zhen Zhang, Shuai Zheng, Feng Qin, and Yida Wang. 2024. [DISTMM: accelerating distributed multimodal model training](#). In *21st USENIX Symposium on Networked Systems Design and Implementation, NSDI 2024, Santa Clara, CA, April 15-17, 2024*. USENIX Association.
- Md. Tahmid Rahman Laskar, Mohammed Saidul Islam, Ridwan Mahbub, Mizanur Rahman, Amran Bhuiyan, Israt Jahan, Mir Tafseer Nayeem, Shafiq Joty, Enamul Hoque, and Jimmy Huang. 2025. [Deploying tiny LVLM judges for real-world evaluation of chart models: Lessons learned and best practices](#). In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, EMNLP 2025 - Industry Track, Suzhou, China, November 4-9, 2025*, pages 1906–1918. Association for Computational Linguistics.
- Peizheng Li, Zhenghao Zhang, David Holtz, Hang Yu, Yutong Yang, Yuzhi Lai, Rui Song, Andreas Geiger, and Andreas Zell. 2025a. [Spacedrive: Infusing spatial awareness into vlm-based autonomous driving](#). *CoRR*, abs/2512.10719.
- Zongxia Li, Xiyang Wu, Hongyang Du, Fuxiao Liu, Huy Nghiem, and Guangyao Shi. 2025b. [A survey of state of the art large vision language models: Benchmark evaluations and challenges](#). In *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops 2025, Nashville, TN, USA, June 11-15, 2025*, pages 1587–1606. Computer Vision Foundation / IEEE.
- Jinkun Lin, Ziheng Jiang, Zuquan Song, Sida Zhao, Menghan Yu, Zhanghan Wang, Chenyuan Wang, Zuocheng Shi, Xiang Shi, Wei Jia, Zherui Liu, Shuguang Wang, Haibin Lin, Xin Liu, Aurojit Panda, and Jinyang Li. 2025. [Understanding stragglers in large model training using what-if analysis](#). In *19th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2025, Boston, MA, USA, July 7-9, 2025*, pages 483–498. USENIX Association.
- Xiaoya Lu, Zeren Chen, Xuhao Hu, Yijin Zhou, Weichen Zhang, Dongrui Liu, Lu Sheng, and Jing Shao. 2026. [Is-bench: Evaluating interactive safety of vlm-driven embodied agents in daily household tasks](#). In *Fortieth AAAI Conference on Artificial Intelligence, Thirty-Eighth Conference on Innovative Applications of Artificial Intelligence, Sixteenth Symposium on Educational Advances in Artificial Intelligence, AAAI 2026, Singapore, January 20-27, 2026*, pages 35680–35688. AAAI Press.
- Minesh Mathew, Viraj Bagal, Rubèn Tito, Dimosthenis Karatzas, Ernest Valveny, and C. V. Jawahar. 2022. [Infographicvqa](#). In *IEEE/CVF Winter Conference on Applications of Computer Vision, WACV 2022, Waikoloa, HI, USA, January 3-8, 2022*, pages 2582–2591. IEEE.
- Xupeng Miao, Yujie Wang, Youhe Jiang, Chunan Shi, Xiaonan Nie, Hailin Zhang, and Bin Cui. 2022. [Galatron: Efficient transformer training over multiple](#)

- gpus using automatic parallelism. *Proc. VLDB Endow.*, 16(3):470–479.
- Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I. Jordan, and Ion Stoica. 2018. [Ray: A distributed framework for emerging AI applications](#). In *13th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2018, Carlsbad, CA, USA, October 8-10, 2018*, pages 561–577. USENIX Association.
- Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. 2020. [Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters](#). In *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020*, pages 3505–3506. ACM.
- Gaurav Shinde, Anuradha Ravi, Emon Dey, Shadman Sakib, Milind Rampure, and Nirmalya Roy. 2025. [A survey on efficient vision-language models](#). *WIREs Data. Mining. Knowl. Discov.*, 15(3).
- Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2019. [Megatron-lm: Training multi-billion parameter language models using model parallelism](#). *CoRR*, abs/1909.08053.
- Meituan LongCat Team. 2025a. [Longcat-flash-omni technical report](#). *CoRR*, abs/2511.00279.
- Qwen Team. 2025b. [Qwen3 technical report](#). *CoRR*, abs/2505.09388.
- Taegeon Um, Byungsoo Oh, Minyoung Kang, Woo-Yeon Lee, Goeun Kim, Dongseob Kim, Youngtaek Kim, Mohd Muzzammil, and Myeongjae Jeon. 2024. [Metis: Fast automatic distributed training on heterogeneous gpus](#). In *Proceedings of the 2024 USENIX Annual Technical Conference, USENIX ATC 2024, Santa Clara, CA, USA, July 10-12, 2024*, pages 563–578. USENIX Association.
- Xiao Wang, Ibrahim Alabdulmohsin, Daniel Salz, Zhe Li, Keran Rong, and Xiaohua Zhai. 2025a. [Scaling pre-training to one hundred billion data for vision language models](#). *CoRR*, abs/2502.07617.
- Yujie Wang, Shenhan Zhu, Fangcheng Fu, Xupeng Miao, Jie Zhang, Juan Zhu, Fan Hong, Yong Li, and Bin Cui. 2025b. [Spindle: Efficient distributed training of multi-task large models via wavefront scheduling](#). In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2, ASPLOS 2025, Rotterdam, Netherlands, 30 March 2025 - 3 April 2025*, pages 1139–1155. ACM.
- Zining Wang, Tongkun Guan, Pei Fu, Chen Duan, Qianyi Jiang, Zhentao Guo, Shan Guo, Junfeng Luo, Wei Shen, and Xiaokang Yang. 2025c. [Marten: Visual question answering with mask generation for multi-modal document understanding](#). In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2025, Nashville, TN, USA, June 11-15, 2025*, pages 14460–14471. Computer Vision Foundation / IEEE.
- Chunyu Xue, Yangrui Chen, Jianyu Jiang, Ningxin Zheng, Junda Feng, Jingji Chen, Shixiong Zhao, Shen Yan, Yi Lin, Lei Shi, Zhanbo Wang, Lishu Luo, Faming Wu, Haibin Lin, Yanghua Peng, Xin Liu, and Quan Chen. 2026a. [Megascale-omni: A hyper-scale, workload-resilient system for multimodal LLM training in production](#). In *Proceedings of the 21st European Conference on Computer Systems, EuroSys 2026, McEwan Hall/The University of Edinburgh, Edinburgh, Scotland, UK, April 27-30, 2026*, pages 675–692. ACM.
- Zhenliang Xue, Hanpeng Hu, Xing Chen, Yimin Jiang, Yixin Song, Zeyu Mi, Yibo Zhu, Daxin Jiang, Yubin Xia, and Haibo Chen. 2026b. [DIP: efficient large multimodal model training with dynamic interleaved pipeline](#). In *Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2, ASPLOS 2026, Pittsburgh, PA, USA, March 22-26, 2026*, pages 618–632. ACM.
- Zili Zhang, Yinmin Zhong, Yimin Jiang, Hanpeng Hu, Jianjian Sun, Zheng Ge, Yibo Zhu, Daxin Jiang, and Xin Jin. 2025. [Disttrain: Addressing model and data heterogeneity with disaggregated training for multimodal large language models](#). In *Proceedings of the ACM SIGCOMM 2025 Conference, SIGCOMM 2025, São Francisco Convent, Coimbra, Portugal, September 8-11, 2025*, pages 24–38. ACM.
- Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Less Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, Alban Desmaison, Can Balioglu, Pritam Damania, Bernard Nguyen, Geeta Chauhan, Yuchen Hao, Ajit Mathews, and Shen Li. 2023. [Pytorch FSDP: experiences on scaling fully sharded data parallel](#). *Proc. VLDB Endow.*, 16(12):3848–3860.
- Yuhang Zhou, Zibo Wang, Zhibin Wang, Ruyi Zhang, Chen Tian, Xiaoliang Wang, Wanchun Dou, Guihai Chen, Bingqiang Wang, Yonghong Tian, Yan Zhang, Hui Wang, Fuchun Wei, Boquan Sun, Jingyi Zhang, Bin She, Teng Su, Yifan Yao, Chunsheng Li, and 4 others. 2025. [Accelerating model training on ascend chips: An industrial system for profiling, analysis and optimization](#). In *Proceedings of the 2025 USENIX Annual Technical Conference, USENIX ATC 2025, Boston, MA, USA, July 7-9, 2025*, pages 1387–1408. USENIX Association.
- He Zhu, Junyou Su, Minxin Chen, Wen Wang, Yijie Deng, Guanhua Chen, and Wenjia Zhang. 2025. [Plangpt-vl: Enhancing urban planning with domain-specific vision-language models](#). In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, EMNLP 2025 - Industry Track, Suzhou, China, November 4-9, 2025*, pages 2461–2483. Association for Computational Linguistics.

A Methodological Details

A.1 Motivation

VLMs align vision and text in a shared representation space, enabling joint processing of multimodal data (Li et al., 2025b; Shinde et al., 2025). Their architectural disparity leads to divergent operator arithmetic intensities and thus divergent responses to distributed parallelism strategies. The core issue is not only where heterogeneous modules are placed, but also whether they can progress independently at their native throughputs. During disaggregated training, the encoders and the LLM backbone remain temporally bound by the underlying pipeline execution order. Execution coupling still introduces batch-level synchronization barriers, resulting in significant resource waste.

Furthermore, dynamic multimodal inputs induce a severe workload imbalance between distributed accelerators. For example, mixed 480P-1080P images with various lengths could belong to the same training batch. Due to data dependence, such variance propagates across the batch boundary and amplifies pipeline bubbles, thereby limiting the end-to-end throughput of VLM training. The above observations motivate us to rethink a different training paradigm that decouples encoder and backbone execution, enabling each module to progress independently via continuous dataflow.

A.2 Backpressure Mechanism

The unified memory pool operates as a bounded buffer. Encoder outputs are preferentially allocated in HBM and fall back to DRAM when HBM cannot satisfy the incoming allocation. If neither tier has sufficient free space, the corresponding encoder actors are blocked until the backbone releases consumed embeddings. LLM actors do not have to be aware of their actual storage location when consuming the microbatches from GVAs. Each GVA is associated with readiness and completion metadata. The backbone can access an embedding only after its write has completed, and the underlying memory cannot be reclaimed until consumption finishes. FlowTrain does not evict unconsumed embeddings, as doing so would alter the training data stream. After an embedding is consumed, its pages are returned to the corresponding free lists for reuse. Page-granular allocation allows logically contiguous GVA objects to map to non-contiguous physical pages, thereby reducing physical memory fragmentation without memory compaction.

Parallelism	Vision Encoder			LLM Backbone		
	DP	TP2	TP4	TP8	TP4PP2	TP8PP2
MFU (%)	45.0	34.2	25.7	60.1	56.7	55.4

Table 2: MFU of the vision encoder and LLM backbone under different parallelism strategies.

A.3 Solving Flow Matching Problem

Once the execution is decoupled, the optimization objective for resource allocation is essentially changed. The parallelism strategy can no longer be optimized as a stage-latency minimization problem. The allocator formulates a flow matching optimization problem that independently determines the resource partition and parallelism strategy for each module under a fixed hardware budget. However, it is intractable to directly solve the optimization problem in Eq. (2) for two reasons. On the one hand, the search space grows significantly with the cluster scale. For a cluster with 256 devices, the number of possible strategy combinations may exceed 10^4 . On the other hand, the steady-state throughput $\tau_i(c)$ depends on the combined effect of operator efficiency and communication overhead, neither of which can be explicitly formulated as a closed-form function. To this end, the allocator exploits a three-step solution that prunes the search space, builds a throughput surrogate model, and solves the MILP problem.

Firstly, we reduce the candidate space $\mathcal{C}_i, \forall i \in \mathcal{E} \cup \{b\}$. As the encoder is relatively small, we only consider DP and TP for the encoder side. Therefore, each candidate configuration occupies $\text{DP} \times \text{TP}$ devices. For the substantially larger backbone side, we additionally include PP, requiring $\text{DP} \times \text{TP} \times \text{PP}$ devices per configuration. We prune invalid and low-efficiency configurations where the global batch size cannot be divisible by the resulting DP degree, and TP/PP degrees cannot factorize the per-module resource budget and cause cross-node tensor parallelism. *Secondly*, we perform throughput profiling by executing a small number of warm-up iterations under a fixed packed sequence length. This lightweight profiling estimates a surrogate of $\tau_i(c)$ and is only performed once before VLM training begins, resulting in acceptable overhead compared to end-to-end training (Um et al., 2024; Miao et al., 2022). *Thirdly*, substituting the surrogates into Eq. (2) yields the MILP problem, which can be efficiently solved by off-the-shelf solvers.

Method	VLM-6B		VLM-32B		VLM-72B	
	MFU	Throughput	MFU	Throughput	MFU	Throughput
Megatron-LM	40.0	3929	40.9	661	31.2	265
DistTrain	43.3	4254	47.4	764	43.9	331
FlowTrain	52.1	5125	54.8	884	52.9	448

Table 3: Complete results of MFU (%) and throughput (tokens/s/device) on the InfoVQA dataset.

Method	VLM-6B		VLM-32B		VLM-72B	
	MFU	Throughput	MFU	Throughput	MFU	Throughput
Megatron-LM	38.7	3801	43.0	694	36.9	282
DistTrain	45.4	4470	49.7	803	47.9	361
FlowTrain	50.8	4990	52.8	852	50.0	382

Table 4: Complete results of MFU (%) and throughput (tokens/s/device) on the Industrial dataset.

B Evaluation Details

B.1 Setting

Models. VLM-6B, VLM-32B, and VLM-72B differ primarily in encoder depth, projector dimension, and LLM capacity. The vision encoder uses 12 to 28 layers with a hidden size of 1536, whereas the LLM backbone ranges from 24 to 80 layers with hidden sizes from 4096 to 8192.

Metrics. (1) *MFU* measures the fraction of hardware FLOPs converted into effective model computation, which directly quantifies resource efficiency independent of hardware count. (2) *Throughput* is reported as the number of tokens processed per second per device (tokens/s/device), reflecting training speed. (3) *Training loss* tracks the model’s convergence trajectory throughout the training process, which is used to evaluate consistency. (4) *Algorithm overhead* quantifies the average additional latency introduced by dynamic packing scheduling.

B.2 Different Parallelism Preferences

The encoder and the LLM backbone in VLMs differ fundamentally in architecture, scale, and computational characteristics, making a single parallelism strategy suboptimal for both. The vision encoder is built from narrow transformer layers with moderate hidden dimensions, while the LLM backbone consists of wide transformer layers with large hidden dimensions. We quantify this divergence in Table 2. When modifying the parallelism strategy from DP to TP2 and TP4, the MFU of the vision encoder decreases from 45.0% to 34.2% and 25.7% due to additional communication overhead and low operator computation efficiency. In contrast, the LLM backbone requires TP8 to satisfy its memory and computation demands and achieve the highest

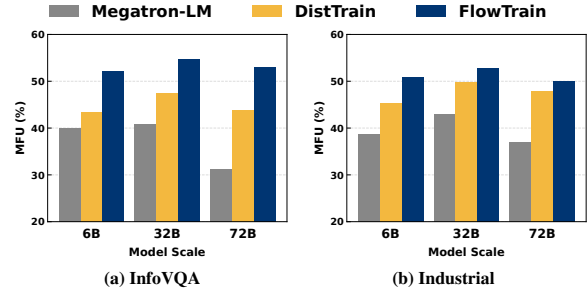


Figure 10: MFU across different methods on varying model scales.

MFU. This asymmetry reveals that different modules have incompatible parallelism preferences.

B.3 Resource Efficiency

Tables 3 and 4 present the complete results of MFU and throughput on the InfoVQA and Industrial datasets, respectively. We further plot the MFU of different methods with three VLM scales in Figure 10. Two key observations are as follows. Firstly, FlowTrain outperforms Megatron-LM by a large margin across all configurations. For example, FlowTrain achieves the MFU of 52.9% and 50.0% when training VLM-72B on the InfoVQA dataset and the Industrial dataset, while Megatron-LM only reaches 31.2% and 36.9%. This gap is rooted in the monolithic design of Megatron-LM, which forces all components to share the same parallelism strategy. The situation further worsens as the models grow larger. The MFU of Megatron-LM drops from 40.0% for VLM-6B to 31.2% for VLM-72B on the InfoVQA dataset, whereas FlowTrain remains stable above 52% through flow matching optimization between producers and consumers.

Secondly, DistTrain partially alleviates the parallelism mismatch through disaggregating orchestration, but its MFU is still lower than that of FlowTrain. This is because DistTrain retains batch-level synchronization barriers and dispatches microbatches without explicit load balancing. In comparison, FlowTrain allows execution decoupling and constructs balanced microbatches. The backbone no longer waits for the entire batch of encoder output before proceeding, thus improving resource utilization. The above results indicate that our disaggregated and decoupled design can substantially improve resource efficiency.