PIPETUNE: TUNING PIPELINE PARALLELISM FOR EFFICIENT VISION-LANGUAGE MODEL TRAINING

Anonymous authors
Paper under double-blind review

ABSTRACT

Training vision-language models (VLMs) efficiently is crucial for advancing multimodal understanding, yet remains challenging due to the heterogeneity of training data. Variations in sequence lengths and modality composition significantly degrade the performance of pipeline parallelism (PP), leading to increased idle times and low hardware utilization.

We present PipeTune, a unified framework that systematically mitigates these inefficiencies by jointly optimizing micro-batch construction, ordering, size, and vision encoder computation. PipeTune adopts a computation-aware packing algorithm to balance workloads, dynamically adjusts micro-batch sizes based on sampled data, reorders execution to minimize stalls, and exploits idle times for encoder pre-computation. A lightweight simulator guides runtime decisions, enabling performance optimization without altering training semantics.

Across diverse model sizes, dataset mixtures, and hardware configurations, PipeTune consistently accelerates training, achieving up to 40.7% reduction in iteration time. Our evaluation demonstrates that each optimization component contributes complementary gains, and the overall overhead remains minimal. By holistically addressing data-induced inefficiencies, PipeTune enables more scalable and efficient training of VLMs.

1 INTRODUCTION

The emergence of vision-language models (VLMs) significantly expands the capabilities of large language models (LLMs). By augmenting the LLM backbone with a vision encoder, VLMs enable the joint understanding of both textual and visual information, thereby unlocking a wide range of multimodal applications such as visual question answering (Antol et al., 2015), multimodal dialogue (Team et al., 2024), and embodied AI (Driess et al., 2023). However, training VLMs efficiently still remains challenging.

A key difficulty in VLM training arises from data heterogeneity. Modern VLMs rely on mixtures of multimodal and textual datasets to achieve broad capabilities. This heterogeneity results in significant variation in sequence lengths and modality composition, both within and across training iterations. Such variability significantly complicates efficient training, especially under pipeline parallelism (PP), a widely adopted approach for scaling model across multiple devices.

In particular, heterogeneous inputs exacerbate three main sources of inefficiency: (1) micro-batch imbalance, where uneven computation across micro-batches leads to stragglers and stalls; (2) inter-iteration fluctuation, where shifting data distributions cause inconsistent pipeline utilization; and (3) modality composition variance, where uneven visual workloads make the vision encoding a bottleneck. Together, these issues increase pipeline bubbles, reduce hardware utilization, and ultimately degrade training throughput.

Existing approaches attempt to mitigate parts of these inefficiencies. For instance, WLB-LLM (Wang et al., 2025) introduces a variable-length packing method to balance computation across micro-batches, Optimus (Feng et al., 2025) exploits pipeline bubble times for encoder computation, and OrchMLLM (Zheng et al., 2025) dispatches batches to improve modality coherence. However, these solutions focus on specific aspects of the problem and fall short of providing a holistic solution.

In this paper, we introduce PipeTune, a unified framework that systematically optimizes pipeline parallelism for efficient VLM training. PipeTune jointly tunes micro-batch construction, ordering, size, and vision encoder scheduling to address the above inefficiencies.

Specifically, PipeTune adopts a computation-aware packing algorithm that balances workloads across micro-batches, mitigating stragglers caused by uneven sequences. It further dynamically adjusts micro-batch sizes according to the characteristics of sampled data, ensuring better pipeline filling and higher utilization. To reduce stalls, PipeTune reorders micro-batch execution and leverages idle periods for vision encoder pre-computation, overlapping computation that would otherwise delay downstream stages. Finally, a lightweight pipeline simulator guides these decisions at runtime, enabling optimization without affecting convergence behavior. Extensive experiments show that PipeTune consistently accelerates training across diverse model scales, data mixtures, and hardware setups, achieving up to 40.7% reduction in iteration time.

2 VISION-LANGUAGE MODEL TRAINING

Figure 1 illustrates the workflow of vision-language models (VLMs). A VLM typically builds upon a large language model (LLM) backbone. By integrating a vision encoder—commonly a Vision Transformer (ViT)—alongside a lightweight projection module, it enables the LLM to jointly reason over both textual and visual modalities. Given the input images and texts, the vision encoder first transforms the images into feature embeddings. These embeddings are then mapped by the projector into token space and interleaved with text tokens to form the *input sequences*. The backbone model then processes the *sequences* and generate the outputs.

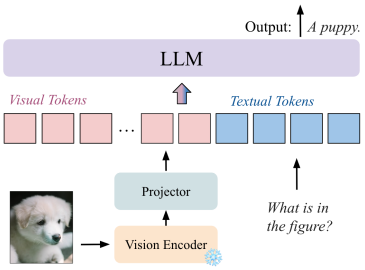


Figure 1: Overview of a vision-language model (VLM) workflow.

2.1 DATASETS

VLM training requires large-scale, diverse datasets comprising both visual and textual inputs. Many previous works (Lu et al., 2024; Wu et al., 2024; Wang et al., 2024; Lin et al., 2024; Deitke et al., 2025) have emphasized the critical role of data construction in determining VLM performance. The open-source datasets commonly used in these studies can be broadly categorized into two types:

- **Image-text pairs:** Each sample consists of one or more images paired with textual annotations or question–answer pairs. Representative datasets include COYO-700M (Byeon et al., 2022b), LAION (Schuhmann et al., 2022).
- **Interleaved image-text corpora:** Samples consist of natural language text interspersed with one or more images, often mimicking real-world multimedia documents. Examples include MMC4 (Zhu et al., 2023), Wiki (Burns et al., 2023).

To ensure broad coverage of real-world scenarios, datasets are typically mixed together during training (Lin et al., 2024). This means that at each training iteration, a VLM needs to simultaneously process the sequences from different datasets, which often vary in the length and the number of images.

In addition, previous works have also found that training solely on multimodal data may lead to catastrophic forgetting of language capabilities (Lin et al., 2024; Lu et al., 2024). To address this, a joint training strategy is often adopted, mixing multimodal data with **text-only corpora** (e.g., FLAN (Longpre et al., 2023)). For example, DeepSeek-VL (Lu et al., 2024) found that a multimodal-to-text mixing ratio of 7:3 yields a favorable trade-off between language retention and multimodal performance. They further introduced a warm-up strategy that gradually reduces the proportion of language-only data as training progresses, thereby facilitating a smooth transition.

While data mixture and curriculum strategies may boost VLM performance, they also make the training data increasingly heterogeneous. Such *data heterogeneity* brings challenges to the training efficiency, which we will discuses later.

2.2 PARALLELISM

Since the language model constitutes the core component of a VLM, the parallelism strategies used in LLM training are also adopted in VLM training. These strategies include data parallelism (DP; Li et al. (2020)), tensor parallelism (TP; Shoeybi et al. (2019)), pipeline parallelism (PP; Narayanan et al. (2021)), and others (Li et al., 2021; Rajbhandari et al., 2020). Among these strategies, PP is commonly used when we need to support larger models and cross-node scaling.

PP partitions the model vertically into *pipeline stages*. Each stage typically contains a subset of the model layers. Input batches are processed sequentially through the pipeline stages in both forward and backward passes. To improve device utilization and throughput, batches are further divided into smaller *micro-batches* so that stages can process different micro-batches at the same time. One widely adopted scheduling strategy is 1F1B (one-forward–one-backward), in which each stage alternates between forward and backward computations (Narayanan et al., 2021). This method reduces peak memory usage and pipeline idle time (also known as bubble time).

In VLM training, sequences in input batches come from diverse datasets and vary significantly in length. Simply padding them to the same maximum length (denoted as *max_seq_len*) when constructing micro-batches leads to substantial wasted computation and memory. Instead, *packing* is often adopted, which concatenates sequences along the sequence dimension to form a longer sequence (Jiang et al., 2024). Attention masks are then applied to prevent tokens from attending across sample boundaries. Recent techniques such as FlashAttention-2 (Dao, 2023) further optimize the kernels for variable-length inputs, which eliminates redundant attention computations between unrelated tokens. While the micro-batch size (denoted as *mbs*) is usually predefined before training to fit within GPU memory constraints, with packing, each micro-batch now comprises a single long sequence of length $mbs \times max\_seq\_len$.

In addition, with visual components introduced in VLMs, their integration into the PP framework can be handled in different ways. Depending on the relative size of these components and the chosen strategy, they can either be assigned to standalone pipeline stages (Zhang et al., 2025) or co-located with existing LLM layers (Lu et al., 2024). In this paper, we adopt the latter approach, as the vision encoder we use is relatively small (0.4B) compared with the backbone models (3-13B). Figure 2 shows several examples of VLM 1F1B schedules, where the visual components (highlighted in green) are scheduled within the first PP stage.

To ensure the efficiency of pipeline parallelism, it's critical to make the computation on different stages overlap effectively over the timeline. This requires that the computation load is evenly distributed across micro-batches and stages. If there were workload imbalance (as illustrated in Figure 2a), the slowest stage may stall other stages and becomes the straggler (Lin et al., 2025). In addition, it is also essential to maintain a sufficient number of micro-batches in the schedule (Huang et al., 2019), otherwise, GPU utilization may remain low even when workloads are balanced. The datasets used for VLM training, however, pose severe challenges to these requirements.

3 CHALLENGES AND OUR SOLUTIONS

In this section, we discuss the efficiency challenges in vision-language model training and our corresponding solutions.

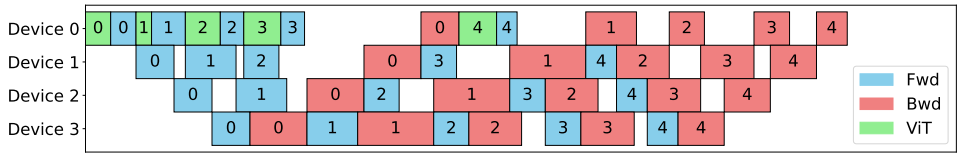
To illustrate, we construct a dataset by sampling from three representative datasets mentioned in Section 2.1: COYO-700M (Image-text pairs), MMC4-Core (Interleaved image-text corpora), and FLAN (Text-only corpora). Their characteristics are summarized in the table 1. Using this mixed dataset, we simulate and visualize some 4-stage pipelines in Figure 2.

3.1 MICRO-BATCH IMBALANCE

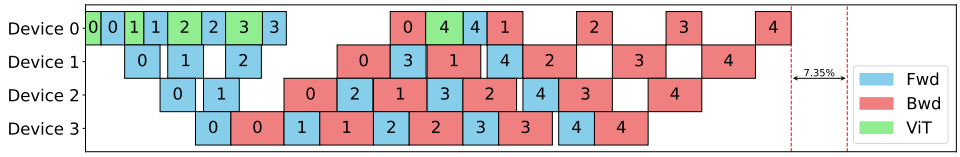
The first challenge stems from workload imbalance across micro-batches. As described in Section 2.2, at each iteration a data batch is sampled from the mixed dataset and then partitioned into micro-batches. With packing, each micro-batch comprises a long sequence containing the same number of tokens.

Dataset	# img / sample			# text tokens / sample		
	Mean	Median	Max	Mean	Median	Max
COYO-700M	1	1	1	17.24	13	1,523
MMC4-Core	4.08	3	15	408.94	362	16,470
FLAN	-	-	-	154.68	52	222,109

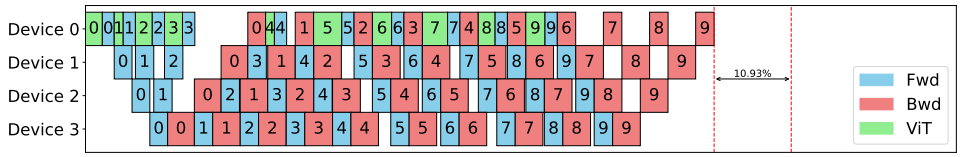
Table 1: Characteristics of three representative datasets used in VLM training.



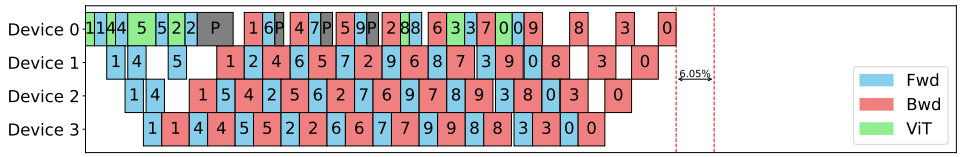
(a) Baseline with imbalanced micro-batches.



(b) Balanced micro-batches. More uniform workloads alleviate straggler effects.



(c) Balanced micro-batches with better micro-batch size. Adjusting the micro-batch size improves pipeline filling and GPU utilization.



(d) Encoder pre-computation and micro-batch reordering further mitigate encoder bottleneck and shorten end-to-end latency.

Figure 2: Pipeline parallelism simulation using the mixed dataset sampled from three datasets in Table 1. The four panels illustrate progressive optimizations addressing inefficiency. Together, these optimizations achieve a 24.33% performance gain..

However, as shown in Table 1, all three datasets exhibit a long-tail distribution in sequence length. This means that when sequences are packed, some micro-batches may contain fewer but longer sequences, while others may include many shorter sequences. As illustrated in Figure 3, due to the quadratic complexity of self-attention, micro-batches with longer sequences require disproportionately more computation. These "long-sequence" micro-batches can therefore become stragglers in pipeline execution. For instance, in Figure 2a, we simulate a scenario in which micro-batches are imbalanced: Micro-batch 1 requires significantly more time for its forward and backward passes. This delay stalls subsequent computations and creates pipeline bubbles.

Our Solution. To address this, we implement a computation-aware packing algorithm that balances workloads across micro-batches. Our approach is inspired by the method proposed in Wang et al. (2025), but instead of relying on a latency estimator, we directly estimate the computational cost in FLOPs. This yields more reliable estimates, particularly for short sequences, where latency predictors tend to be inaccurate. Check the appendix A.3 for more details.

As shown in Algorithm 1, the algorithm takes a batch of input sequences and their lengths, and incrementally constructs packed micro-batches under the maximum sequence length constraint $L_{\max}$. For each new sequence, the algorithm first tries to place it into the micro-batch that yields the lowest

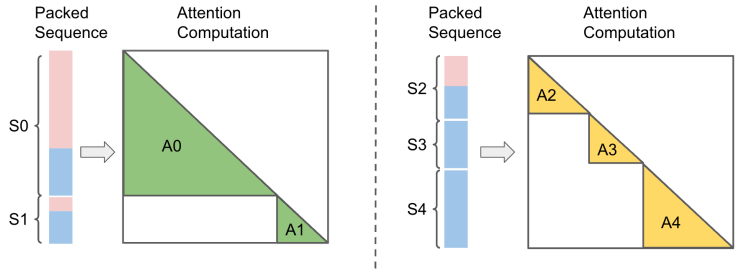


Figure 3: Illustration of workload imbalance across micro-batches caused by the quadratic complexity of attention computation. Pink blocks represent visual tokens, while blue blocks indicate textual tokens in the packed sequences.

computation cost. If no placement satisfies the length constraint, it falls back to the micro-batch with the smallest current length. This greedy two-stage strategy effectively alleviates the workload imbalance. For the micro-batches that are still slightly imbalanced after packing, we adopt micro-batch reordering to further reduce the pipeline bubble, which is covered in the later section. As shown in Figure 2b, applying the balance algorithm reduces pipeline bubbles and achieves a 7.35% reduction in end-to-end training time.

Algorithm 1 Balance Packing Algorithm

Input: Sequences $\{s_i\}_{i=1}^{n}$ with lengths $\{l_i\}$, max length $L_{\max}$, model config C
Output: Packed micro-batches B

Sort sequences by length in descending order $B \leftarrow [\,]$
foreach *sequence* (s, l) **do**
  target $\leftarrow \varnothing$ minF $\leftarrow +\infty$
  foreach *micro-batch* $b \in B$ **do**
    if *len*$(b) + l \le L_{\max}$ **then**
      $F \leftarrow$ `compute_flops`$(b \cup \{l\}, C)$ **if** $F <$ *minF* **then**
        minF $\leftarrow F$ target $\leftarrow b$
  if *target* $= \varnothing$ **then**
    target $\leftarrow \arg\min_{b:\,\text{len}(b)+l \le L_{\max}} \text{len}(b)$
  if *target* $= \varnothing$ **then**
    Create new micro-batch with (s)
  else
    Append s to target
return B

3.2 Inter-iteration Fluctuation

Data heterogeneity can also cause fluctuation across iterations. As mentioned in Section 2.2, the micro-batch size (mbs) is typically defined during initialization and remains fixed throughout training. However, this static configuration may not be optimal. Consider the MMC4 and FLAN datasets listed in Table 1. If each image is transformed into 576 visual tokens by the encoder, then the average sequence length of MMC4 samples is 17.83 times longer than that of FLAN samples. Consequently, when using the same micro-batch size, FLAN samples yield far fewer tokens per step and therefore fewer micro-batches. When sampling from such heterogeneous datasets, the average sequence length and the number of micro-batches after packing may fluctuate across iterations, leading to under-utilization of pipeline parallelism. While the micro-batches have been balanced in the pipeline shown in Figure 2b, the pipeline is still insufficiently filled. In contrast, as shown in Figure 2c, simply reducing the micro-batch size by half accelerates the pipeline by 10.93%.

Our Solution. Instead of relying on the static configuration, at each iteration, we use a pipeline simulator to estimate the latency and search for the optimal micro-batch size based on the data sampled.

3.3 Modality Composition Variance

Another source of inefficiency arises from variance in modality composition, meaning that the proportion of visual and textual tokens can differ significantly across micro-batches. As illustrated in Figure 3, the first packed sequence contains a much higher proportion of visual tokens than the second. This issue can persist even with balance algorithm, as a sequence comprises purely textual tokens and another sequence with purely visual tokens can have identical computation workload for the LLM backbone. However, the processing time required by the vision encoder can vary substantially. Thus, micro-batches with a large proportion of images can cause the vision encoder to become a performance bottleneck. As shown in Figure 2c, both micro-batch 5 and 7 suffer from encoder-induced delays, which propagate through the pipeline.

Our Solution. We identify two opportunities to mitigate this issue. First, following Feng et al. (2025), idle device times can be exploited for encoder pre-computation. During such periods, we pre-fetch images from subsequent micro-batches and perform encoder computation in advance, as highlighted in gray in Figure 2d.

Second, we can permute the execution order of micro-batches within the pipeline. Micro-batch reordering can not only reduce the bubbles caused by variance across micro-batches, but also search for a better scheduling for encoder pre-computation. For example, injecting micro-batch 5 earlier in the schedule reduces the delay caused by its higher encoding overhead. Because gradients from different micro-batches are accumulated throughout training, this reordering preserves the original convergence semantics (we further explain this in the appendix A.2). With these two techniques applied, we observe a further 6.05% improvement in training efficiency.

4 PipeTune

Integrating the aforementioned solutions, we present **PipeTune**, a framework designed to systematically enhance the efficiency of pipeline parallelism in vision-language model training. PipeTune revises the packing strategy and adopts the balance algorithm to evenly distribute workloads across micro-batches. It then adaptively tunes pipeline parallelism along three key dimensions: (i) *micro-batch order*, which determines the execution sequence of micro-batches within the pipeline; (ii) *micro-batch size*, which controls the number of micro-batches and affects pipeline utilization; and (iii) *encoder computation*, which governs the scheduling of visual encoder processing. These dimensions are jointly optimized through a simulator-driven approach, enabling PipeTune to evaluate candidate configurations and select those that minimize pipeline latency at runtime.

To efficiently explore the configuration space, PipeTune first performs offline profiling to characterize the relationship between computation time and input sequence length for both the backbone and encoder models. Based on these profiling results, and given the input sequences in each iteration, it estimates each micro-batch's stage latency. It then employs a directed acyclic graph (DAG)-based simulator to approximate the overall pipeline latency. Specifically, the computation of micro-batch i at stage j is abstracted as a node in the DAG, and dependencies between nodes are determined by the 1F1B schedule. With m micro-batches in an n-stage pipeline, the DAG contains mn nodes, and topological sorting can be performed in $O(mn)$ time. Using this simulator, PipeTune searches for the optimal micro-batch sizes and ordering that minimize latency, while rescheduling encoder computation to better exploit idle times. We will discuss the search overhead in Section 5.2.

5 Evaluation

We develop PipeTune atop the open-sourced, PyTorch-native framework TorchTitan (Liang et al., 2024) and evaluate its training performance across various VLMs, with backbone language model sizes of 3B, 7B, and 13B. The backbone models adopt architectures similar to the LLaMA series models (Touvron et al., 2023). The configurations are detailed in the table 4.

Model	# of Layers	Embed Dim	FFN Hidden Dim	# of Attn. Heads
3B	24	2560	6912	32
7B	32	4096	11008	32
13B	40	5120	13824	40

Figure 4: Backbone model specifications used in the evaluations.

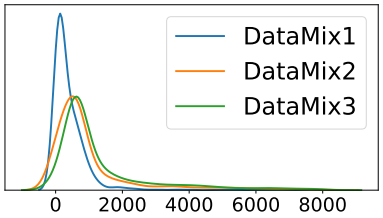


Figure 5: Sequence length distributions of different data mixtures.

For the ViT-based image encoder, we use the `Siglip-so400m` model and connect it to the backbone via a two-layer MLP projector. In all experiments, input images are rescaled to a resolution of 336 pixels, resulting in each image being encoded into 576 patch tokens by the vision encoder.

Testbed: We conduct our experiments on two super-computing clusters to evaluate PipeTune under diverse hardware environments:

- **Cluster A**: Each node contains 4 NVIDIA A100-80GB GPUs, connected via high-bandwidth NVLinks, and a single 64-core AMD CPU with 256GB DRAM. For inter-node communication, each node is equipped with 4 NICs delivering 200Gbps of bandwidth via PCIe 4.0.
- **Cluster B**: Each node contains a single NVIDIA H200 GPU (96GB HBM3) and a 72-core Grace CPU with 120GB DRAM. Nodes are connected via NVIDIA InfiniBand, providing up to 400Gbps of internode bandwidth.

All the experiments are conducted using pipeline parallelism across 4 nodes (PP = 4). On Cluster A, we additionally enable tensor parallelism (TP = 4) within each node to maximize NVLink bandwidth utilization. The global batch size (i.e., total number of samples per iteration) is fixed at 128. We set max_seq_len to 4,096 for PP = 4 experiments. For experiments with TP = 4 and PP = 4, max_seq_len is increased to 8,192. Micro-batch sizes (`mbs`) are tuned per configuration to fully utilize available GPU memory.

Datasets: Following prior work (Lin et al., 2024), we adopt a combination of interleaved image–text corpora (MMC4-Core (Zhu et al., 2023)), image–text pairs (COYO-700M (Byeon et al., 2022a)), and text-only corpora (FLAN (Chung et al., 2024)). To evaluate PipeTune under different data heterogeneity conditions, we construct three mixed datasets with varying sampling ratios:

- **DataMix1:** MMC4 : COYO : FLAN = 5 : 5 : 90
- **DataMix2:** MMC4 : COYO : FLAN = 30 : 30 : 40
- **DataMix3:** MMC4 : COYO : FLAN = 45 : 45 : 10

The sequence length distributions of these three mixtures are shown in Figure 5. DataMix1 simulates a language-dominant setting, with text-only data comprising 90% of the training corpus. In contrast, DataMix2 and DataMix3 progressively increase the proportion of multimodal data to 60% and 90%, respectively. As the multimodal content increases, both the average sequence length and its variance grow substantially, intensifying the heterogeneity challenges addressed by PipeTune.

Baselines: We compare PipeTune against two baselines — Original and Balanced — both of which fix the micro-batch size to the maximum value permitted by GPU memory throughout training. Neither baseline employs micro-batch reordering nor encoder pre-computation. We use training iteration time (excluding data loading) as the performance metric. All configurations are implemented within the same framework to ensure a fair comparison.

- **Original:** Samples are randomly packed into micro-batches without any workload balancing.
- **Balanced:** Samples are packed using the balance algorithm introduced in Section 3.1.

5.1 Overall Performance

We first compare PipeTune with the baselines in terms of average iteration time. As shown in Figure 6, PipeTune improves training throughput by 18.40%–40.72% compared with the original setting, with consistent gains across model sizes and dataset mixtures. Improvement over the bal-

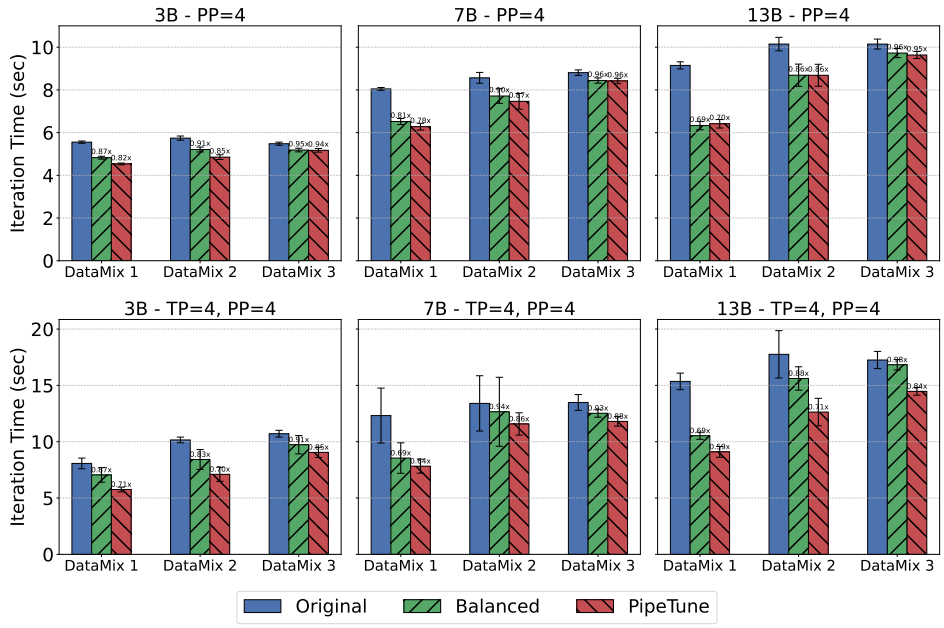


Figure 6: Average iteration times on three data mixtures across different model sizes and parallelism settings.

anced setting can also reach 19.32%, underscoring the effectiveness of PipeTune's joint optimization strategy.

Performance improvements are more pronounced with larger GPU configurations (TP = 4, PP = 4). Tensor parallelism allows for larger micro-batch sizes within GPU memory limits, thereby expanding the search space for better configurations. PipeTune exploits this flexibility by dynamically identifying optimal settings across different iterations.

To further analyze these gains, Table 2 presents a breakdown of results for the 13B model on Cluster A, where we incrementally integrate three key components (balance algorithm, micro-batch reordering & encoder pre-computation, and adaptive micro-batch sizing) over the original setting. This breakdown highlights the distinct contributions of each component and shows how their cumulative effect leads to the end-to-end performance gains.

For DataMix1, the balance algorithm provides the largest initial gain (31.46%). Because 90% of this dataset consists of text-only samples, the resulting sequences are relatively short, producing fewer micro-batches per iteration. In this regime, pipeline throughput is highly sensitive to straggler micro-batches. Eliminating variance through balancing has a direct impact on end-to-end efficiency. While techniques like reordering and pre-computation yield only marginal improvements here.

For DataMix2 and DataMix3, which include more multimodal data, sequence lengths are longer and more heterogeneous. In these settings, balancing alone provides moderate benefits but does not fully address the imbalance across modality and iterations. Here, adaptive micro-batch sizing emerges as the dominant contributor, yielding improvements of 11.8% and 9.1% respectively. Moreover, with a larger number of micro-batches in the pipeline, reordering them and leveraging idle time within the pipeline for encoder pre-computation also bring more benefits.

5.2 Ablation and Analysis

Effects of adaptive micro-batch size. We further investigate the impact of adaptive micro-batch size. We conduct experiments using the 3B model with TP = 4, as this configuration has the largest search space — the maximum micro-batch size can be set to 8. As shown in Figure 7, fixing the micro-batch size at 4 achieves the best static performance across all three data mixtures. For DataMix1 and DataMix2, PipeTune matches this optimal static performance after searching. However, in DataMix3, the higher variance in sequence lengths causes more significant inter-iteration

Settings	DataMix1	DataMix2	DataMix3
Original	15.35	17.76	17.25
+ Balance algorithm	**10.52 (↓31.5%)**	15.61 (↓12.1%)	16.82 (↓2.5%)
+ Micro-batch reordering & Encoder pre-computation	10.34 (↓32.6%)	14.73 (↓17.1%)	16.04 (↓7.0%)
+ Adaptive micro-batch sizing	9.10 (↓40.7%)	**12.63 (↓28.9%)**	**14.47 (↓16.1%)**

Table 2: Breakdown of PipeTune's optimizations on the 13B model (TP = 4). Values denote average iteration time in seconds, with relative improvements over the original shown in parentheses.

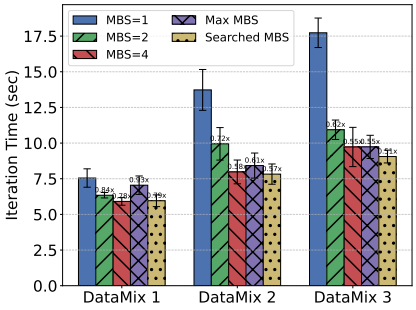


Figure 7: Effects of adaptive micro-batch size on the 3B model (TP = 4, PP = 4).

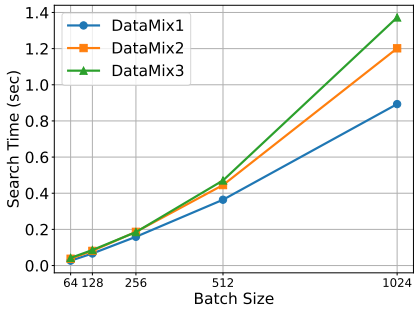


Figure 8: PipeTune's search overhead across different global batch sizes.

fluctuations. By adaptively selecting the optimal micro-batch size at each step, PipeTune achieves a further 9.27% improvement compared to static settings.

Overhead of PipeTune optimization. The primary source of PipeTune overhead comes from finding the optimal micro-batch order, as this involves permutation and simulating the corresponding latency. Exhaustive permutation becomes infeasible as the number of micro-batches increases. Thus, following Jiang et al. (2024), we cluster micro-batches based on their estimated latency and only permute the order of these clusters. We observe that applying the balance algorithm (§3.1) significantly reduces computation variance across micro-batches, enabling us to keep the number of clusters below six. Figure 8 reports the search overhead as we scale the global batch size. In the previous experiments, the overhead remains under 0.2 seconds. Even when scaling the global batch size to 1,024, PipeTune requires less than 1.5 seconds to tune the pipeline parallelism, demonstrating the efficiency and scalability of our method.

6 Related Works and Discussion

Model Heterogeneity. In this work, we primarily focus on mitigating the training inefficiency introduced by data heterogeneity. As discussed in §2.2, the vision encoder used in our experiments is relatively lightweight compared to the LLM backbone — a common setup in many open-sourced VLMs (Wang et al., 2024; Lu et al., 2024). However, as the vision encoder scales, model heterogeneity can also become a non-negligible challenge. Prior studies (Huang et al., 2024; Zhang et al., 2025; Jang et al., 2025) have explored strategies for efficiently scheduling heterogeneous modules during VLM training.

Imbalance across DP groups. Previous work (Zhang et al., 2025; Zheng et al., 2025) has also investigated methods for re-balancing workloads along the data parallel dimension to eliminate potential bottlenecks. While our method focuses on optimizing pipeline parallelism, these techniques are generally complementary and have the potential to further optimize the training efficiency.

7 CONCLUSION

In this paper, we presented PipeTune, a framework that accelerates VLM training by tuning pipeline parallelism. PipeTune balances micro-batch construction and adaptively optimizes micro-batch order, micro-batch size, and encoder computation. Through extensive experiments across diverse model scales, dataset compositions, and hardware configurations, PipeTune achieves up to 40.7% reduction in training latency. Our analysis further shows that each component contributes complementary performance gains, while the optimization overhead remains negligible., highlighting the effectiveness and scalability of our approach.

REFERENCES

Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C Lawrence Zitnick, and Devi Parikh. Vqa: Visual question answering. In *Proceedings of the IEEE international conference on computer vision*, pp. 2425–2433, 2015.

Siddharth Barman and Sanath Kumar Krishnamurthy. Approximation algorithms for maximin fair division. *ACM Transactions on Economics and Computation (TEAC)*, 8(1):1–28, 2020.

Andrea Burns, Krishna Srinivasan, Joshua Ainslie, Geoff Brown, Bryan A Plummer, Kate Saenko, Jianmo Ni, and Mandy Guo. A suite of generative tasks for multi-level multimodal webpage understanding. *arXiv preprint arXiv:2305.03668*, 2023.

Minwoo Byeon, Beomhee Park, Haecheon Kim, Sungjun Lee, Woonhyuk Baek, and Saehoon Kim. Coyo-700m: Image-text pair dataset. `https://github.com/kakaobrain/coyo-dataset`, 2022a. Accessed: 2025-05-09.

Minwoo Byeon, Beomhee Park, Haecheon Kim, Sungjun Lee, Woonhyuk Baek, and Saehoon Kim. Coyo-700m: Image-text pair dataset. `https://github.com/kakaobrain/coyo-dataset`, 2022b.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. Scaling instruction-finetuned language models. *Journal of Machine Learning Research*, 25(70):1–53, 2024.

Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*, 2023.

Matt Deitke, Christopher Clark, Sangho Lee, Rohun Tripathi, Yue Yang, Jae Sung Park, Mohammadreza Salehi, Niklas Muennighoff, Kyle Lo, Luca Soldaini, et al. Molmo and pixmo: Open weights and open data for state-of-the-art vision-language models. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 91–104, 2025.

Danny Driess, Fei Xia, Mehdi SM Sajjadi, Corey Lynch, Aakanksha Chowdhery, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, Wenlong Huang, et al. Palm-e: An embodied multimodal language model. 2023.

Weiqi Feng, Yangrui Chen, Shaoyu Wang, Yanghua Peng, Haibin Lin, and Minlan Yu. Optimus: Accelerating {Large-Scale}{Multi-Modal}{LLM} training by bubble exploitation. In *2025 USENIX Annual Technical Conference (USENIX ATC 25)*, pp. 161–177, 2025.

Ronald L. Graham. Bounds on multiprocessing timing anomalies. *SIAM journal on Applied Mathematics*, 17(2):416–429, 1969.

Jun Huang, Zhen Zhang, Shuai Zheng, Feng Qin, and Yida Wang. {DISTMM}: Accelerating distributed multimodal model training. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pp. 1157–1171, 2024.

Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *Advances in neural information processing systems*, 32, 2019.

Insu Jang, Runyu Lu, Nikhil Bansal, Ang Chen, and Mosharaf Chowdhury. Cornstarch: Distributed multimodal training must be multimodality-aware. *arXiv preprint arXiv:2503.11367*, 2025.

Chenyu Jiang, Zhen Jia, Shuai Zheng, Yida Wang, and Chuan Wu. Dynapipe: Optimizing multi-task training through dynamic pipelines. In *Proceedings of the Nineteenth European Conference on Computer Systems*, pp. 542–559, 2024.

Richard E Korf. Multi-way number partitioning. In *IJCAI*, volume 9, pp. 538–543, 2009.

Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, et al. Pytorch distributed: Experiences on accelerating data parallel training. *arXiv preprint arXiv:2006.15704*, 2020.

Shenggui Li, Fuzhao Xue, Chaitanya Baranwal, Yongbin Li, and Yang You. Sequence parallelism: Long sequence training from system perspective. *arXiv preprint arXiv:2105.13120*, 2021.

Wanchao Liang, Tianyu Liu, Less Wright, Will Constable, Andrew Gu, Chien-Chin Huang, Iris Zhang, Wei Feng, Howard Huang, Junjie Wang, et al. Torchtitan: One-stop pytorch native solution for production ready llm pre-training. *arXiv preprint arXiv:2410.06511*, 2024.

Ji Lin, Hongxu Yin, Wei Ping, Pavlo Molchanov, Mohammad Shoeybi, and Song Han. Vila: On pre-training for visual language models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 26689–26699, 2024.

Jinkun Lin, Ziheng Jiang, Zuquan Song, Sida Zhao, Menghan Yu, Zhanghan Wang, Chenyuan Wang, Zuocheng Shi, Xiang Shi, Wei Jia, et al. Understanding stragglers in large model training using what-if analysis. *arXiv preprint arXiv:2505.05713*, 2025.

Shayne Longpre, Le Hou, Tu Vu, Albert Webson, Hyung Won Chung, Yi Tay, Denny Zhou, Quoc V Le, Barret Zoph, Jason Wei, et al. The flan collection: Designing data and methods for effective instruction tuning. In *International Conference on Machine Learning*, pp. 22631–22648. PMLR, 2023.

Haoyu Lu, Wen Liu, Bo Zhang, Bingxuan Wang, Kai Dong, Bo Liu, Jingxiang Sun, Tongzheng Ren, Zhuoshu Li, Hao Yang, et al. Deepseek-vl: towards real-world vision-language understanding. *arXiv preprint arXiv:2403.05525*, 2024.

Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, et al. Efficient large-scale language model training on gpu clusters using megatron-lm. In *Proceedings of the international conference for high performance computing, networking, storage and analysis*, pp. 1–15, 2021.

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–16. IEEE, 2020.

Christoph Schuhmann, Romain Beaumont, Richard Vencu, Cade Gordon, Ross Wightman, Mehdi Cherti, Theo Coombes, Aarush Katta, Clayton Mullis, Mitchell Wortsman, et al. Laion-5b: An open large-scale dataset for training next generation image-text models. *Advances in neural information processing systems*, 35:25278–25294, 2022.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

Gemini Team, Petko Georgiev, Ving Ian Lei, Ryan Burnell, Libin Bai, Anmol Gulati, Garrett Tanzer, Damien Vincent, Zhufeng Pan, Shibo Wang, et al. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*, 2024.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

Peng Wang, Shuai Bai, Sinan Tan, Shijie Wang, Zhihao Fan, Jinze Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, et al. Qwen2-vl: Enhancing vision-language model's perception of the world at any resolution. *arXiv preprint arXiv:2409.12191*, 2024.

Zheng Wang, Anna Cai, Xinfeng Xie, Zaifeng Pan, Yue Guan, Weiwei Chu, Jie Wang, Shikai Li, Jianyu Huang, Chris Cai, et al. Wlb-llm: Workload-balanced 4d parallelism for large language model training. *arXiv preprint arXiv:2503.17924*, 2025.

Zhiyu Wu, Xiaokang Chen, Zizheng Pan, Xingchao Liu, Wen Liu, Damai Dai, Huazuo Gao, Yiyang Ma, Chengyue Wu, Bingxuan Wang, et al. Deepseek-vl2: Mixture-of-experts vision-language models for advanced multimodal understanding. *arXiv preprint arXiv:2412.10302*, 2024.

Zili Zhang, Yinmin Zhong, Yimin Jiang, Hanpeng Hu, Jianjian Sun, Zheng Ge, Yibo Zhu, Daxin Jiang, and Xin Jin. Disttrain: Addressing model and data heterogeneity with disaggregated training for multimodal large language models. In *Proceedings of the ACM SIGCOMM 2025 Conference*, pp. 24–38, 2025.

Yijie Zheng, Bangjun Xiao, Lei Shi, Xiaoyang Li, Faming Wu, Tianyu Li, Xuefeng Xiao, Yang Zhang, Yuxuan Wang, and Shouda Liu. Orchestrate multimodal data with batch post-balancing to accelerate multimodal large language model training. *arXiv preprint arXiv:2503.23830*, 2025.

Wanrong Zhu, Jack Hessel, Anas Awadalla, Samir Yitzhak Gadre, Jesse Dodge, Alex Fang, Youngjae Yu, Ludwig Schmidt, William Yang Wang, and Yejin Choi. Multimodal c4: An open, billion-scale corpus of images interleaved with text. *Advances in Neural Information Processing Systems*, 36:8958–8974, 2023.

A Appendix

A.1 The Use of Large Language Models (LLMs)

In preparing this work, we made use of large language models (LLMs) in the following ways:

Writing Assistance: We used the LLM to improve the clarity and fluency of language. But all content was reviewed, verified, and where necessary, rewritten by the authors to ensure accuracy and originality.

Technical Support: During the implementation and experiments, we used the LLM for code and data analysis suggestions. But all the final implementation and experiments were designed and validated by the authors.

Limitations of Use: At no stage was the LLM used to generate novel research results or survey related work to make substantive claims.

A.2 Training Convergence Equivalence

In this section, we explain why our method does not alter the training semantics. PipeTune improves training efficiency by altering (1) micro-batch construction, (2) micro-batch ordering, (3) micro-batch sizes, and (4) vision encoder pre-computation. Vision encoder pre-computation is scheduled to utilize the idle times and always produces the same outputs, since the encoder parameters are not updated.

Now let θ denote the backbone model parameters and let $B = \{z_1, ..., z_N\}$ be a global batch of training examples, where $z_i = (x_i, y_i)$, For a per-example loss $\ell(\theta; z_i)$, the global batch loss is

$$L(\theta; B) = \frac{1}{N} \sum_{i=1}^{N} \ell(\theta; z_i) \tag{1}$$

Now with pipeline parallelism, the batch indices $\{1, ..., N\}$ are partitioned into disjoint micro-batches $S_1, ..., S_M$ where the sets $S_m, m \in \{1, ..., M\}$ may have arbitrary sizes. The batch loss

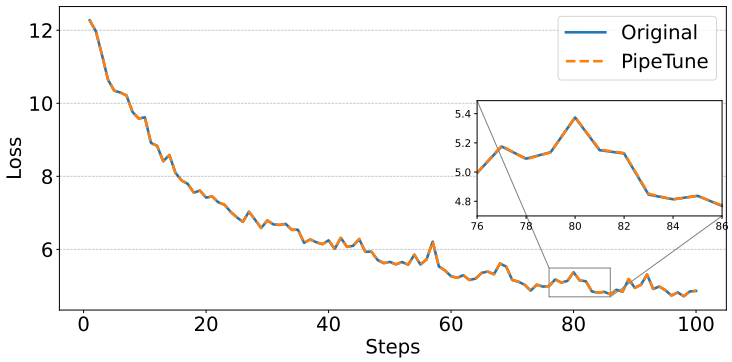


Figure 9: Loss curve of training the 3B model on DataMix1

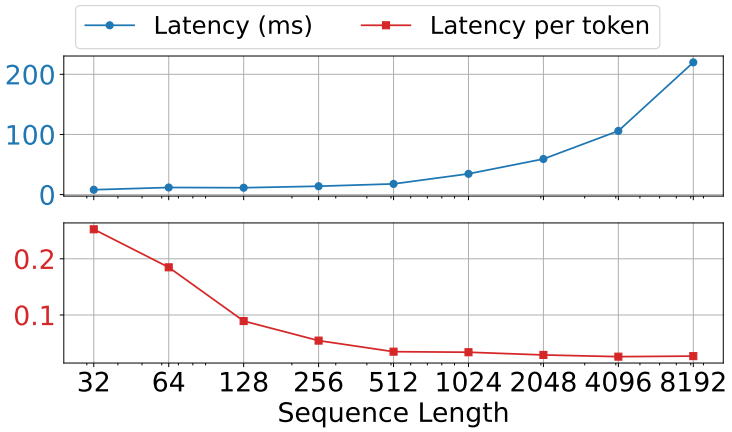


Figure 10: Computation time and normalized latency for a single LLaMA-7B model layer with tensor parallel size = 4. The red line shows the latency normalized by sequence length, which serves as a proxy for GPU utilization (lower values indicate higher utilization).

can be rewritten as

$$L(\theta; B) = \frac{1}{N} \sum_{m=1}^{M} \sum_{i \in S_m} \ell(\theta; z_i) \tag{2}$$

This expression depends only on the collection $\{z_i\}_{i=1}^{N}$, not on how these examples are grouped and the order of the micro-batches. Note that we can also rewrite the expression as

$$L(\theta; B) = \sum_{m=1}^{M} \frac{1}{N} \sum_{i \in S_m} \ell(\theta; z_i) \tag{3}$$

Thus, for each micro-batch in the 1F1B schedule, we only need to divide the sum of the per-example losses within that micro-batch by the global batch size N. The resulting accumulated gradient over the global batch is identical, regardless of the specific micro-batch partitioning, ordering, or the number of gradient accumulation steps.

To further validate that our method preserves training semantics, Figure 9 compares the loss curves of the Original and PipeTune settings when training the 3B model on the DataMix1 dataset; the curves match with each other perfectly, with a maximum absolute deviation of less than 0.01.

A.3 PACKING ALGORITHM

The goal of the packing algorithm introduced in Section 3.1 is to partition sequences into micro-batches that are computationally balanced. Formally, given a set of sequences with lengths $l_1, \ldots, l_N$, a per-micro-batch maximum capacity $L_{\max}$, and a workload estimation model $F(\cdot)$, the objective is to partition the sequences into micro-batches $B_1, \ldots, B_m$ such that:

1. Each micro-batch satisfies the capacity constraint: $\text{len}(B_i) \leq L_{\max}$.
2. The micro-batches have as balanced a computational workload as possible, which is equivalent to minimizing the maximum workload $\max_j F(B_j)$.

This problem can be mapped to the multiway number partitioning problem (Korf, 2009), which has been proved to be NP-hard. To obtain an efficient solution, we adopt a greedy algorithm. Prior work (Barman & Krishnamurthy, 2020; Graham, 1969) shows that such algorithms can yield near-optimal results with an approximation factor of at most $4/3$ of the optimum.

For the workload estimator $F(\cdot)$, we use FLOPs to approximate the workload instead of directly estimating latency, as we find that latency estimates tend to be inaccurate, especially for short sequences. To illustrate this, in Figure 10 we show the computation time of a single LLaMA-7B layer for inputs of different lengths. As expected, longer inputs generally incur longer computation times. However, if we normalize the computation time by the sequence length (shown by the red line), we observe that the per-token time is higher for short sequences, then decreases with increasing input length, and eventually converges to a stable value. Since we pack balanced micro-batches incrementally, in the early stages when the sequences are short, the observed latency can be similar even for inputs of different lengths due to low hardware utilization. Thus, we choose FLOPs for workload estimation.

A.4 SIMULATOR PROFILING

In order to efficiently estimate the stage latency of different configurations during runtime, we perform offline profiling to characterize the relationship between computation time and input size for both the backbone and encoder models.

For the vision encoder, images are extracted from packed micro-batches at runtime. Since all images are rescaled to the same resolution, they are concatenated along the batch dimension and fed into the vision encoder. Consequently, to profile the computation time of the vision encoder, we only need to sweep over all possible batch sizes and record the corresponding latencies.

For the backbone model, we exploit the homogeneity of transformer layers: because computation and communication workloads are identical across layers, it suffices to capture the latency characteristics of a single layer. We further decompose the single-layer workload into two categories: attention computation and all other operations. Given a packed micro-batch input $\{s_1, s_2, \ldots, s_n\}$, FlashAttention computes attention for each sequence s_i independently, avoiding wasted computation. Since no inter-device communication is involved in attention computation, we can profile attention latency as a function of sequence length on a single GPU.

The remaining operations, such as GEMM computation and collective communication (e.g., All-Gather and ReduceScatter), process the packed sequence as a whole. We obtain their time cost by measuring the end-to-end latency for different overall sequence lengths and subtracting the corresponding attention latency. This methodology naturally extends to hybrid parallelism (i.e., TP + PP). For each pipeline stage, we just measure the end-to-end latency under the corresponding parallelism configuration. At runtime, we estimate the time for non-attention operations based on the overall packed sequence length, and estimate attention cost based on the individual sequence lengths within each packed micro-batch.

A.5 SIMULATOR-GUIDE SEARCH

To estimate the overall pipeline latency, we use a directed acyclic graph (DAG)-based simulator. The computation of micro-batch i at stage j is abstracted as a node in the DAG, and dependencies between nodes are determined by the 1F1B schedule. In this way, we can simulate the overall pipeline latency via topological sorting, while capturing the effects of stragglers and encoder-induced delays.

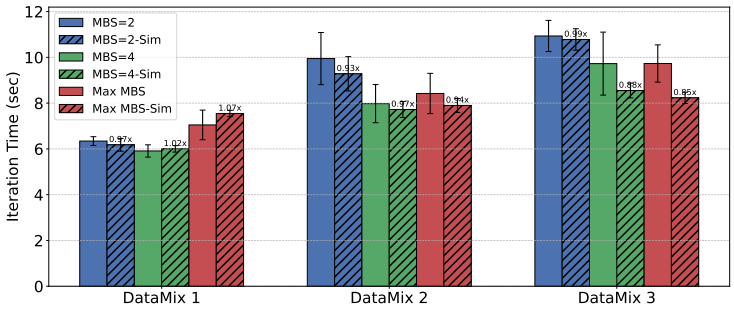


Figure 11: Comparison between actual and simulated latency on the 3B model (TP = 4, PP = 4).

Utilizing this simulator, PipeTune searches for the optimal configuration. PipeTune first iterates over the micro-batch sizes that satisfy the memory constraints. Given a micro-batch size, the balance algorithm repacks the samples into new micro-batches. We then estimate the stage latency of each micro-batch based on profiling results. Next, we cluster the micro-batches by their estimated latency and permute the order of the clusters. For each micro-batch ordering, we determine whether encoder pre-computation can be scheduled and compute the corresponding estimated overall latency. Simulations of different configurations are conducted in parallel to reduce search overhead. Finally, the simulator selects the micro-batch size, ordering, and encoder pre-computation plan that achieve the lowest estimated latency.

In Figure 7 we show the iteration times of 3B model with different micro-batch sizes on the three datasets. We compare the simulated latencies with the corresponding actual latencies in the Figure 11 to validate the effectiveness of the simulator. As we can see, the relative ordering of different configurations is consistent between the actual and simulated latencies, indicating that the simulator can effectively identify better configurations for efficient training.