Efficient Scaling of LLM Training with Flexible Context Parallelism

Yifan Niu[1*] **Han Xiao**[1*] **Dongyi Liu**[1] **Wei Zhou**[2] **Jia Li**[1,3†]
[1]The Hong Kong University of Science and Technology (Guangzhou)
[2]Huawei Technologies Co., Ltd.
[3]The Hong Kong University of Science and Technology

Abstract

Scaling long-context capabilities is crucial for Large Language Models (LLMs). However, real-world data contain a large number of sequences with heterogeneous lengths. Existing training libraries for LLMs rely on static parallelism strategies, which suffer from severe load imbalance, redundant communication, and suboptimal hardware utilization under data heterogeneity. In this work, we propose **F**lexible **C**ontext **P**arallelism (FCP), an efficient parallelism strategy that adaptively reconfigures communication groups and context parallelism degrees during LLM training. We support non-power-of-two Ring-CP degrees and combine memory-aware grouping with polynomial-time degree allocation, whose solve takes tens of milliseconds in the evaluated configurations. FCP is able to maintain high hardware efficiency even under extreme data heterogeneity. Experimental results demonstrate that FCP significantly outperforms Megatron-LM and DeepSpeed in both LLM and MLLM training, improving throughput across heterogeneous workloads on clusters of up to 64 devices. For extremely unbalanced batches, FCP even achieves 2.24 $\times$ speedup. The code is available at https://github.com/ivanniu/FCP.

1 Introduction

Training language and vision-language models on long documents and videos requires efficient use of distributed compute. Real training corpora contain heterogeneous sequence lengths, so equal token counts do not necessarily imply equal attention costs. This makes per-batch resource allocation an important complement to model and kernel optimization.

A fundamental challenge in scaling LLMs to ultra-long contexts is the quadratic complexity $O(L^2)$ of self-attention regarding the number of tokens [15, 16, 24, 25]. Existing parallel training strategies are broadly divided into two categories: static parallelism and dynamic parallelism. Static parallelism (e.g., Megatron-LM [20], DeepSpeed [23]) typically employs predefined 4D parallelism: Data Parallelism (DP) [3, 13], Tensor Parallelism (TP) [26], Pipeline Parallelism (PP) [7, 18, 19], Sequence Parallelism (SP) [2, 14, 8] or Context Parallelism (CP) [2, 11] to distribute the training on massive clusters. However, static methods often suffer from workload imbalances and redundant communication when dealing with variable-length sequences or heterogeneous batching. To address the issue, recent research has introduced dynamic parallelism. ByteScale [6] uses heuristic scheduling, while FlexSP [28] optimizes flexible sequence-parallel allocations using a programming-based solver. FCP studies a different tradeoff: low-overhead allocation over integer Ring-CP degrees.

In this work, we follow the trajectory of dynamic parallelism and rethink the essential properties that define an optimal scheduling strategy. A robust dynamic parallelism should satisfy three core re-

*Equal contribution.
†Correspondence to: Jia Li (jialee@ust.hk).

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

quirements: (1) **Workload Balance**. The computational load across different communication groups should be balanced to minimize synchronization stalls and maximize hardware efficiency. (2) **Elastic Parallelism Degree**. The strategy should support flexible, non-power-of-two parallelism degrees to prevent short sequences from incurring redundant communication overhead, thereby adapting to highly skewed data distributions. (3) **Minimal Scheduling Overhead**. Scheduling should use modest CPU resources and expose little latency on the training critical path; the amount hidden depends on the duration of the concurrent training step.

Motivated by these principles, we propose Flexible Context Parallelism (FCP), an efficient and adaptive parallel strategy designed to accelerate LLM long-context training on heterogeneous data. FCP dynamically reconfigures communication groups and parallelism degrees for each micro-batch to handle the skewed distributions of sequence lengths. FCP is built on Ring-style Context Parallelism (CP), allowing for arbitrary integer parallelism degrees that better fit the varying sequence lengths. We use memory-aware heuristic grouping followed by a polynomial-time dynamic program that optimally allocates CP degrees conditional on the fixed groups and profiled cost model. We implement FCP as a non-intrusive, asynchronous module atop Megatron-LM, allowing scheduling to overlap with device computation in the evaluated configurations. Furthermore, FCP can also be adapted to the training of Multimodal Large Language Models (MLLMs) [4, 30, 1]. Experimental results demonstrate that FCP significantly outperforms Megatron-LM and DeepSpeed in both LLM and MLLM training, improving throughput across heterogeneous workloads on clusters of up to 64 devices. For extremely unbalanced batches, FCP even achieves 2.24 $\times$ speedup.

2 Related Work

Static training systems such as Megatron-LM [20] and DeepSpeed [23] combine data, tensor, pipeline, and sequence/context parallelism. Dynamic systems additionally adapt to heterogeneous sequence lengths. ByteScale [6] combines data-aware sharding, dynamic communication, and heuristic scheduling. FlexSP [28] uses bucket construction and a programming-based allocator, with power-of-two degrees in its evaluated Ulysses-based configuration. FCP instead combines memory-aware grouping with an $O(K'N^2)$ degree allocator and Ring-CP execution. Support for non-power-of-two degrees alone is not unique to FCP. Moreover, a backend restriction of an evaluated system does not establish a fundamental limitation of its scheduling algorithm. Our in-framework comparisons in Section 6.5 therefore distinguish the reproduced scheduling strategies from complete original systems. FlexSP reports solver/computation overlap; when overlap is complete, scheduling quality and resource requirements, rather than solver latency alone, determine the remaining benefit.

3 Preliminaries

3.1 Large Language Models

The primary computational workload of Large Language Models (LLMs) resides in a stack of Transformer blocks. Each block is composed of two main components: a Multi-Head Self-Attention (MHA) module and a Feed-Forward Network (FFN). While the FFN exhibits linear complexity with respect to the sequence length L, the MHA module presents a fundamental scaling bottleneck. Formally, for an input hidden state $H \in \mathbb{R}^{L \times D}$, the self-attention operation is computed as:

$$\text{Attn}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{1}$$

where Q, K, V are linear projections of H. The computation of the affinity matrix QK^T requires $\mathcal{O}(L^2D)$ floating-point operations (FLOPs). As LLMs scale to ultra-long contexts, this quadratic complexity $\mathcal{O}(L^2)$ dominates the execution time per transformer layer.

3.2 Distributed Training Paradigms

Data Parallelism. Data Parallelism (DP) [13] partitions data along the batch dimension. Each device maintains a full replica of model parameters and synchronizes gradients via All-Reduce operations. To mitigate the redundancy of model states, Sharded Data Parallelism (SDP) (e.g., ZeRO [22],

FSDP [31]) partitions model parameters, gradients, and optimizer states across devices, trading increased communication for significantly reduced per-device memory consumption.

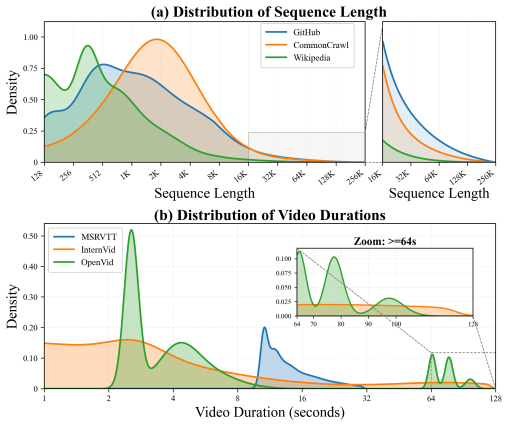
Model Parallelism. Model Parallelism decomposes model parameters across devices, categorized into Tensor Parallelism (TP) and Pipeline Parallelism (PP). TP [26] partitions individual operators, requiring frequent synchronization of activations. Therefore, TP is typically restricted to intra-node deployment to leverage high-bandwidth interconnects. PP [7, 18] partitions the model by layers and only transfers boundary activations between stages, making it suitable for cross-node scaling.

Sequence-Dimension Partitioning (CP & SP). Both Sequence Parallelism (SP) and Context Parallelism (CP) distribute data along the sequence dimension to overcome memory bottlenecks. Sequence Parallelism (SP), such as DeepSpeed-Ulysses [8], uses all-to-all primitives to redistribute data so that each device computes attention for a subset of heads over the full sequence. In a standard equal-head-sharding implementation, the SP degree must divide the applicable local attention-head counts, including key/value heads where required by the backend. This often yields power-of-two candidates for the evaluated models, but powers of two are not a universal mathematical requirement of Ulysses. Context Parallelism (CP) (e.g., Ring-Attention [2, 14, 12]) decomposes the attention computation itself. CP often employs ring-based point-to-point communication to exchange Key/Value blocks, allowing the overlap of communication with computation. Thus, CP usually supports longer context without being constrained by head-count divisibility.

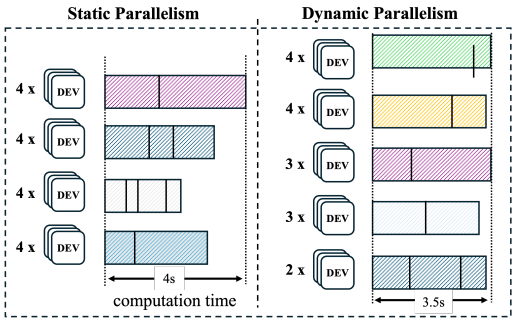
4 Method

4.1 Motivation and Observation

The length distributions of real-world data are skewed, exhibiting significant variations across different datasets and modalities. As shown in Figure 1 (a), most sequences in common text datasets (e.g., GitHub [9], CommonCrawl [21], and Wikipedia [5]) and multimodal datasets (e.g., MSRVTT [29], InternVid [27], and OpenVid [17]) are relatively short, but a few samples are several times longer than the average. Furthermore, the length distributions of these datasets also show significant differences in mean, central location, and long-tail patterns.



(a) Data length distributions.



(b) Static parallelism vs. dynamic parallelism.

Figure 1: (a) Length distribution of text and multimodal datasets. (b) Compared to static parallelism, dynamic parallelism facilitates load balancing against heterogeneous data.

When training on a mix of short and long contexts, static methods often partition resources by equal token counts or fixed parallelism degree, but this does not account for the non-linear computation induced by attention. A rank processing a single long sequence L performs $O(L^2)$ computation, whereas a rank processing multiple packed short sequences $\{l_i\}$ only executes $\sum O(l_i^2)$. Since $L^2 \gg \sum l_i^2$, ranks assigned long text sequences or long video-derived visual contexts become stragglers, forcing other ranks to idle at synchronization barriers, as illustrated in Figure 1 (b). Conversely, dynamic parallelism adaptively reconfigures communication groups and sequence assignment. By avoiding over-packing of short sequences and balancing the computational workload, it minimizes hardware idle time and reduces the total computation time.

4.2 Problem Formulation

Notation. In this work, we focus on the dynamic construction of CP groups, as the ring attention overlaps the communication cost well for long sequences. By dynamically assigning each sequence s_k in a micro-batch $\mathcal{B}$ to a specific CP group C_p with an appropriate parallelism degree d_p, our formulation implicitly incorporates DP. We exclude Model Parallelism (TP & PP) from our dynamic optimization space, as the overhead of reconfiguring TP/PP groups (e.g., weight reshuffling) during runtime is unacceptable. Consequently, we treat TP and PP as predefined static configurations. In our discussion, a single "rank" r_n represents a complete model replica (i.e., TP $\times$ PP physical devices), and N denotes the total number of such replicas available in the cluster.

Formally, let $\mathcal{R}$ be the set of N available computing units in the cluster. We aim to dynamically partition $\mathcal{R}$ into disjoint CP groups $\mathcal{C} = \{C_p\}^P$ with degree d_p, and $\bigcup C_p = \mathcal{R}$. Given a micro-batch of K sequences $\{s_k\}$ with heterogeneous lengths and a per-rank memory budget E, we aim to determine: (1) the total number of CP groups, (2) the parallelism degree of each group, and (3) the assignment of each sequence to a specific CP group.

We consider positive-integer Ring-CP degrees that satisfy the backend's memory, padding, and kernel constraints [2]. Compared with restricting the candidate degrees to powers of two, this allows finer-grained allocation. We evaluate the candidate-set effect separately from the scheduling-policy effect. We define the sequence assignment matrix $\boldsymbol{A} \in \{0,1\}^{K\times P}$, where $A_{k,p} = 1$ represents that s_k is assigned to C_p. The more flexible CP group and sequence assignment problem can be formulated as follows:

$$\arg\min_{\boldsymbol{A},\mathcal{C}} \max \mathcal{T}(C_p), \tag{2}$$

$$\text{s.t.} \quad \mathcal{M}(C_p) \le E \cdot d_p, \ \forall p \in [1, P] \tag{3}$$

$$\sum\nolimits_k A_{k,p} \ge 1, \ \forall p \in [1, P] \tag{4}$$

$$\sum\nolimits_p A_{k,p} = 1, \ \forall k \in [1, K] \tag{5}$$

$$\sum\nolimits_p d_p = N \tag{6}$$

where $\mathcal{T}(\cdot)$ and $\mathcal{M}(\cdot)$ denotes the execution time and the memory cost in CP group C_p. The optimization objective is to minimize the makespan, defined as the maximum execution time across all CP groups. The constraints are defined as follows: Cond. (3) enforces the hardware memory limit on each rank; Cond. (4) excludes empty groups, and Cond. (5) assigns each sequence to exactly one group; finally, Cond. (6) requires all N ranks to belong to the disjoint CP groups, consistent with the terminal DP state used below. This is a model-based allocation problem: each group's predicted time depends on its sequences and degree; physical placement and concurrent-group interference are not additional optimization variables. Note that this optimization problem is *NP-hard*.

4.3 Cost Estimation

Memory Estimation. The memory consumption consists of two primary components: model states and activations. For model states, the model state memory M_{ms} remains constant per rank. For activations, CP partitions each sequence s_k evenly into d_p chunks along the sequence dimension. Thus, for a CP group C_p, the memory consumption is estimated as:

$$\mathcal{M}(C_p) = \sum_k A_{k,p}|s_k| \cdot \mathrm{M}_{token} + d_p \mathrm{M}_{ms}, \tag{7}$$

where M_{token} is activation memory per token and $\mathcal{M}(C_p)$ is aggregate group memory. Equivalently, estimated memory per rank is $\mathrm{M}_{ms} + \sum_k A_{k,p}|s_k|\mathrm{M}_{token}/d_p$. Workspace and padding costs must be accounted for in the profiling budget; this model is not a proof of runtime OOM freedom.

Time Estimation. (1) **Computation Cost.** Computation cost should take MHA and FFN into consideration. To achieve a unified estimation for both LLM and MLLM workloads, we generalize the computation cost model to accommodate diverse attention mask. While standard LLMs primarily employ causal attention, MLLMs often incorporate full attention for specific tokens. We introduce a mask efficiency factor η_k to represent the additional computational overhead introduced by the

full attention. For standard causal attention, η_k is set to 0, whereas for full attention, η_k reflects the additional computations required. Thus, the computation cost for group C_p is represented as:

$$\mathcal{T}_{cp}(C_p, d_p) = \sum_k A_{k,p}(\alpha_1(d_p)(1+\eta_k)|s_k|^2 + \alpha_2(d_p)|s_k|) + \beta_1(d_p), \tag{8}$$

where the coefficients are calibrated for the candidate CP degree d_p and fixed model/backend. Their degree dependence captures sharding and measured execution efficiency; η_k depends on the attention mask.

(2) **Communication Cost.** Ring Attention [2] utilizes point-to-point communication in a circular topology to exchange Key-Value (KV) blocks. The total communication cost is proportional to the sequence length s_k. The total communication cost is estimated as:

$$\mathcal{T}_{cm}(C_p, d_p) = \frac{1}{v_p} \sum_k A_{k,p}\alpha_3(d_p)|s_k| + \beta_2(d_p), \tag{9}$$

where v_p is the profiled bandwidth for the group configuration. The degree-dependent coefficients absorb communication rounds and per-rank message sizes; a different backend requires recalibration.

(3) **Total Time Estimation.** As the ring attention allows the communication cost and computation cost of attention to overlap, the communication time of attention needs to be excluded here. For simplicity, we do not detail the formula of the attention computation time $\mathcal{T}_{cpa}$ and communication time $\mathcal{T}_{cma}$, which has the similar form as Eq. (8) and (9). Then, the overall execution time is:

$$\mathcal{T}(C_p) = \mathcal{T}_{cp} + \mathcal{T}_{cm} - \min(\mathcal{T}_{cpa}, \mathcal{T}_{cma}). \tag{10}$$

4.4 Polynomial-Time Problem Solving

To efficiently solve the NP-hard problem defined in Eq. (2), we propose a two-stage approximation algorithm: Memory-aware Sequence Packing followed by Dynamic Programming Resource Allocation. The grouping step is heuristic; the degree-allocation step is exact conditional on those groups, feasible candidates, and the profiled model. The resulting joint schedule has no general approximation-ratio guarantee. Section 6.5 reports its gap to an exact reference that additionally permits idle ranks.

Stage 1: Atomic Sequence Grouping via Best-Fit Decreasing (BFD). To mitigate memory fragmentation and reduce the number of decision variables, we first group heterogeneous sequences into *atomic sequence groups*. We sort all sequences in $\mathcal{B}$ in descending order of their memory requirements. Assuming $E > \mathrm{M}_{ms}$, for each long sequence we determine its activation-memory lower bound on the CP degree $d_{\min,k'} = \lceil |s_k|\mathrm{M}_{token}/(E - \mathrm{M}_{ms}) \rceil$, round this bound up to an admissible backend degree, and initialize a "bin" with activation capacity $d_{\min,k'}(E - \mathrm{M}_{ms})$. We then employ a BFD strategy to greedily pack shorter sequences into the remaining memory headroom of these established bins. This transforms the original set of K sequences into K' atomic groups $\{\mathcal{G}_1, \ldots, \mathcal{G}_{K'}\}$, where $K' \leq K$. Each group $\mathcal{G}_k$ is treated as a single scheduling unit that requires at least $d_{\min,k'}$ ranks to satisfy memory constraint. Note that this successfully avoids communication redundancy caused by packing massive short sequences.

Stage 2: Optimal Resource Assignment via 2D-Dynamic Programming. Given the atomic groups, we propose a 2D dynamic programming approach to select the CP degree for each fixed group; the sequence-to-group assignment is already determined by Stage 1. Let $DP[i][j]$ denote the minimum achievable *makespan* (i.e., the maximum execution time among all groups) for the first i atomic groups utilizing a total of j ranks. The state transition is defined as:

$$DP[i,j] = \min_{\substack{d \in \mathcal{D}_i \\ d \leq j}} \max\{DP[i-1, j-d], \mathcal{T}(\mathcal{G}_i, d)\}, \tag{11}$$

where $\mathcal{D}_i$ contains all feasible degrees for group i, after memory and backend constraints. Initialize $DP[0,0] = 0$ and every other entry to $+\infty$. Infeasible predecessors are skipped. If $DP[K', N]$ is infinite, no allocation of the fixed groups uses exactly N ranks and the routine returns infeasible. Otherwise, backtracking starts at (K', N) and returns one degree per group. The allocation costs $O(K'N^2)$ time and $O(K'N)$ space in the worst case. The result is optimal for this fixed-group subproblem, not for joint grouping and allocation. Appendix B gives complete pseudocode.

5 Overall Workflow and Implementation

Workflow. Figure 2 shows the scheduler and executor. Given a global batch from the data loader, the scheduler's **micro-batch planner** divides it into memory-feasible micro-batches for gradient accumulation. For each micro-batch, the scheduler performs **memory-aware sequence grouping**, uses the **dynamic programming allocator** to select CP degrees, and creates the corresponding communication groups. The executor then dispatches data and runs the resulting plan. Micro-batch planning is part of the scheduler; the grouping and degree-allocation stages determine the adaptive CP plan.

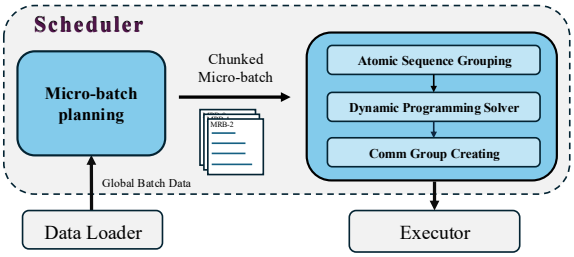


Figure 2: Overall workflow of FCP, including micro-batch planning within the scheduler.

Implementation Details. Our implementation is based on the Megatron-LM framework. For seamless integration with Megatron, all FCP components are implemented as add-on modules without modifying the Megatron core codebase. Our system primarily consists of two new core classes: Scheduler and Profiler, which together achieve robust dynamic parallelism. Below, we will detail several key aspects of the FCP implementation:

1. **Decoupling Scheduling and Training.** In FCP, the workflow of each training batch consists of two phases: (1) the scheduling phase, which generates a grouping and CP-degree plan; and (2) the execution phase, which uses the allocated strategy to perform the real computation. To hide scheduling latency, while the device is computing the current batch, the CPU simultaneously computes the next batch's plan and prepares the necessary communication groups.

2. **Profiler Integration and Cost Modeling.** Before the entire training process begins, the Profiler collects execution times for different sequence lengths and CP degrees, then establishes a functional relationship between runtime and factors (sequence length and CP degree), storing the corresponding parameters in the Profiler class. During the training, Profiler predicts the cost based on the stored parameters, providing the solver with a fast response.

3. **Integration with Megatron Parallel Utilities.** FCP leverages Megatron's Parallel State Manager (MPU). To execute the latest dynamic parallel plan, The Scheduler class dynamically updates the CP parallel communication group within the MPU during each batch execution. Only the CP communication group is modified; all other forms of parallelism continue to run according to the original grid. This allows FCP to integrate seamlessly with existing Megatron frameworks.

Scope and portability. TP/PP configurations remain fixed. Conceptually, scheduling can treat an entire TP group as one allocation unit. Compatibility with FSDP additionally requires consistent collective ordering, parameter/gradient ownership, and synchronization across changing CP groups; balancing iteration makespan alone does not guarantee zero FSDP overhead. The evaluated FCP implementation uses Ring CP on Ascend devices. NVIDIA execution and FSDP $\times$ CP $\times$ TP meshes require practical validation. Other backends can be represented by replacing the feasible-degree sets and recalibrating the cost model, but their end-to-end performance is not established by the Ring-CP experiments.

6 Experiments

6.1 Experimental Setup

Hardware Environment. End-to-end training experiments use up to four nodes. Each node is equipped with 16 Ascend 910C NPUs (64GB memory each), which are interconnected via HCCS within the node.

Baselines. Our static end-to-end baselines are DeepSpeed [23] and Megatron-LM [20]. Megatron-LM supports 4D parallelism, integrating TP, PP, DP, and CP. DeepSpeed supports Ulysses-style sequence parallelism.

Datasets. We evaluated various baselines under a workload with a maximum sequence length of 128k. For LLMs, we use Qwen3 and Llama3/Llama3.2 models on three text datasets with different sequence-length distributions: GitHub [9], CommonCrawl [21], and Wikipedia [5]. For MLLMs, we use InternVL3, InternVL2.5, and Qwen3VL models on three video-language datasets: MSRVTT [29], InternVid [27], and OpenVid [17]. Further details are provided in Appendix A.

Evaluation Protocol. For each static baseline, we use standard packing and select its best-performing parallelism configuration. Since FCP adaptively schedules parallelism parameters, no manual per-run CP specification is required. We use global batch size 512 for the main end-to-end comparisons; the scheduling-scale and dynamic-baseline experiments explicitly vary this value. The evaluation metrics are the end-to-end training iteration time and the token throughput per device. For training-performance measurements, we warm up for 5 steps and average the next 10 steps. The exact-reference study instead compares model-predicted makespans.

6.2 Evaluation

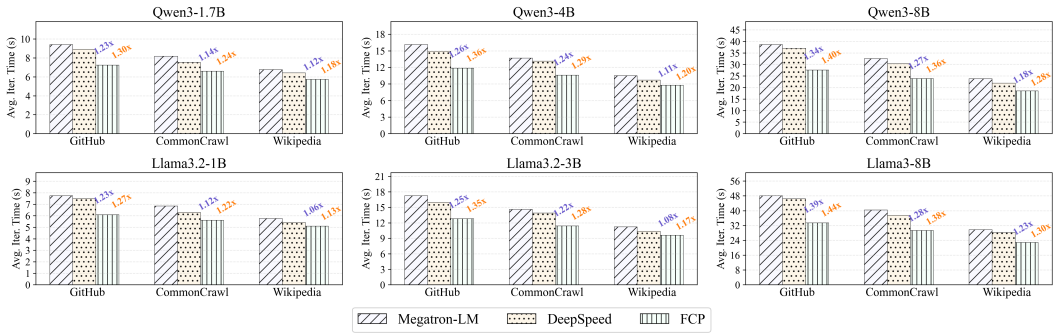


Figure 3: Average iteration time (in seconds) comparison on text datasets for Qwen3-1.7B/4B/8B, Llama3.2-1B/3B, and Llama3-8B. We compare Megatron-LM (diagonal stripes), DeepSpeed (dots), and FCP (vertical stripes) on GitHub, CommonCrawl, and Wikipedia.

LLM Training Performance. We compare FCP with the static baselines on GitHub, CommonCrawl, and Wikipedia using six Qwen3 and Llama-family models. Against the faster static baseline in each configuration, Figure 3 shows speedups of 1.06×–1.39×, with 11 of 18 configurations exceeding 1.2×. The largest gain is on GitHub with Llama3-8B. For Qwen3-8B, speedups over Megatron-LM are 1.40×, 1.36×, and 1.23× on the three datasets, respectively; this denominator is also used in Table 4.

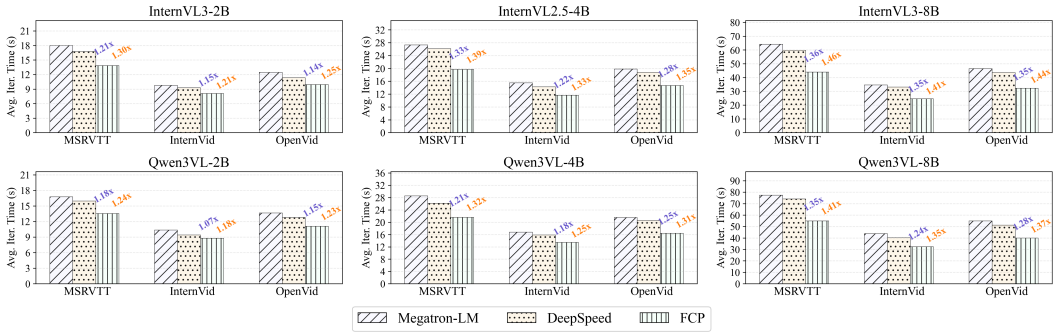


Figure 4: Average iteration time (in seconds) comparison for unfrozen MLLM training across MSRVTT, InternVid, and OpenVid datasets. The evaluated models include InternVL3-2B/8B, InternVL2.5-4B and Qwen3VL-2B/4B/8B.

Unfrozen MLLM Training Performance. We train the vision encoder, projector, and LLM jointly on MSRVTT, InternVid, and OpenVid using InternVL3, InternVL2.5, and Qwen3VL models at 2B, 4B, and 8B scales. Against the faster static baseline, Figure 4 shows speedups of 1.07× (Qwen3VL-2B on InternVid) to 1.36× (InternVL3-8B on MSRVTT), with 12 of 18 configurations exceeding 1.2×. These results show benefits across heterogeneous multimodal workloads and model sizes.

Scalability Analysis. In this section, we analyze the performance of FCP on training clusters of different sizes. We run experiments on 1/2, 1, 2, and 4 nodes (corresponding to 8, 16, 32, and 64 devices respectively) and record the per-device token throughput. As shown in Figure 5, The results show that as the cluster scale increases, while per-device throughput decreases due to growing communication overhead, FCP's flexible and dynamic scheduling strategy mitigates this reduction effectively. Specifically, when scaling from 1/2 node (8 devices) to 4 nodes (64 devices), FCP maintains and even improves its relative throughput against DeepSpeed (1x) increasing from 1.02x to 1.16x. In contrast, DeepSpeed's per-device throughput drops from approximately 2.0 to 1.7 k tokens/s, and Megatron-LM's throughput declines from 1.8 to 1.52 k tokens/s. This demonstrates that FCP achieves superior scalability by minimizing the negative impact of communication overhead as the cluster size expands.

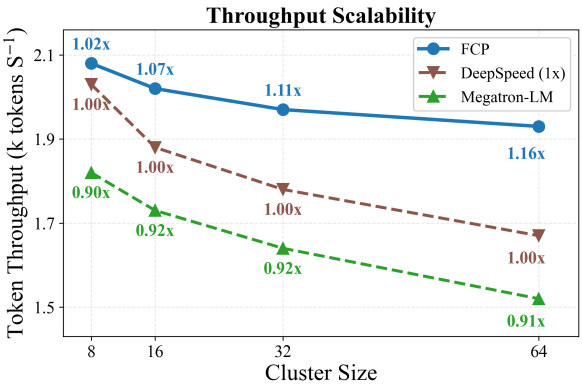


Figure 5: Per-device token throughput (in k tokens/s) across cluster sizes of 8, 16, 32, and 64 devices. Speedups are relative to DeepSpeed at each size.

6.3 Time Analysis

Scheduling Overhead. Our proposed method uses an efficient solver with low time complexity. To validate this in real scenarios, we record empirical scheduling time, solver time, and the shortest GBS computation time. The tables below summarize the results under varying global batch sizes and cluster size when training Qwen3VL-2B.

Table 1. Varying global batch size.

	GBS		
	128	256	512
Compute (s)	6.34	7.58	11.65
Schedule (ms)	448	696	905
Solver (ms)	23	45	86

Table 2. Varying Device count.

	Device Number		
	16	32	64
Compute (s)	34.45	18.26	11.65
Schedule (ms)	282	537	905
Solver (ms)	20	44	86

As shown in Tables 1 and 2, the solver time is at most 86 ms at the largest tested configuration. The overall scheduling time is also consistently shorter than a single GBS computation time when either increasing the GBS or scaling the cluster size. For Qwen3VL-2B with GBS 512 on 64 devices, scheduling takes 905 ms (including 86 ms in the solver), while computation takes 11.65 s. FlexSP's reported 5–15 s [28] comes from its own setup, not a matched timing experiment. With full overlap, scheduler latency leaves the critical path; the throughput comparisons in Section 6.5 do not isolate that condition.

Communication Group Management Overhead. We also measure the overhead of creating and deleting CP communication groups, since FCP dynamically updates the CP group assignment across batches. The measured group-management overhead is around 1 ms in our implementation, which is negligible compared with the per-step computation time and therefore does not slow down the training process.

6.4 Profiler Error Analysis

To evaluate the accuracy of our profiler and cost estimator, we measure the prediction error across different modules and computational processes, then report the average error percentage.

Table 3 shows that the estimation error remains below 8% across Qwen3VL and InternVL2.5/3 models. This level of accuracy is sufficient for scheduling as FCP mainly relies on the relative ordering of candidate CP strategies rather than exact absolute latency prediction.

Table 3: Time cost estimation error (%).

Model	Model Parameters		
	2B	4B	8B
Qwen3VL	7.93	6.71	4.27
InternVL2.5/3	7.48	6.54	4.12

Noise-injection results (Appendix D) show unchanged CP allocations up to 20% coefficient noise in the tested cases; at 50% noise, step-time increases are 4.84% and 6.16% on 32 and 64 devices. Representative 32-rank allocations and the extreme-length example are in Appendix E.

6.5 Scheduling Ablations

Assignment and degree flexibility. Table 4 compares Qwen3-4B/8B across three datasets. The cost-aware fixed-CP control sorts sequences by $L_i^2/d + \mathrm{comm}(L_i)$ and assigns each to the least-loaded memory-feasible group. The power-of-two variant restricts FCP's degree candidates. FCP retains geometric-mean speedups of 1.10× and 1.08× over these controls, respectively; its gain over standard packing is therefore not attributable solely to non-power-of-two degrees.

Table 4: Assignment and degree ablations: throughput relative to Megatron-LM (1.00×), with 64 devices and GBS 512.

Model	Dataset	Fixed CP + cost	Power-of-two FCP	FCP
Qwen3-4B	GitHub	1.20	1.20	1.36
	CommonCrawl	1.12	1.17	1.29
	Wikipedia	1.08	1.14	1.20
Qwen3-8B	GitHub	1.30	1.35	1.40
	CommonCrawl	1.23	1.24	1.36
	Wikipedia	1.17	1.16	1.23

Dynamic scheduling strategies. On GitHub/Qwen3-4B (Table 5), the *ByteScale-style* baseline constructs length-sorted, FLOPs-balanced tasks, fixes their minimum feasible CP degrees, and greedily assigns tasks by decreasing predicted duration to the earliest available rank sets. The *FlexSP-style* baseline combines DP bucket construction with a programming solver over power-of-two degrees. FCP achieves geometric-mean speedups of 1.10× and 1.12×, respectively. These in-framework strategy comparisons include backend and candidate-set differences; they do not reproduce every optimization of the original systems or isolate a backend-independent scheduling advantage.

Table 5: Dynamic strategies on GitHub/Qwen3-4B: throughput relative to Megatron-LM for each device count and GBS.

Devices	GBS	ByteScale-style	FlexSP-style	FCP
32	128	1.22	1.25	1.34
32	256	1.19	1.19	1.29
32	512	1.26	1.15	1.33
64	128	1.13	1.14	1.29
64	256	1.21	1.24	1.35
64	512	1.20	1.17	1.36

Gap to an exact reference. For $N = 16$, we compare BFD-plus-DP with joint grouping/degree optimization under the same profiled cost and memory models. The exact reference permits idle ranks ($\sum_p d_p \leq N$), whereas Algorithm 1 uses all N ranks. It therefore supplies a relaxed lower

bound: the comparison includes both grouping and rank-use policy. Over 100 paired micro-batches per configuration (25 each from GitHub, CommonCrawl, Wikipedia, and OpenVid), the mean per-batch gaps $100(\mathcal{T}_{\mathrm{FCP}}/\mathcal{T}^*_{\leq N} - 1)$ are 9.41%, 12.39%, and 10.60% for $K = 16, 18, 20$, respectively. These are model-predicted gaps, not measured throughput differences or a worst-case guarantee.

7 Conclusion

In this paper, we presented Flexible Context Parallelism (FCP), an adaptive framework designed to optimize LLM training on heterogeneous datasets. By enabling non-power-of-two parallelism degrees and employing a two-stage approximation algorithm, FCP reduces load imbalance and redundant communication with sub-second scheduling time in the evaluated configurations. Our experiments show consistent throughput improvements over static baselines and additional gains over the evaluated in-framework dynamic strategies. Gains diminish for balanced workloads. The small-instance gaps use a model-based exact reference that permits idle ranks, and broader hardware and hybrid-mesh support remain to be validated.

Acknowledgments and Disclosure of Funding

This research was supported by NSFC 62572418, the Guangdong Provincial Talent Program under Grant (No.2024TQ08X366).

Competing Interests

The authors declare no additional competing interests beyond the institutional affiliations and support disclosed in this paper.

References

[1] Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, et al. Qwen3-vl technical report, 2025.

[2] William Brandon, Aniruddha Nrusimha, Kevin Qian, Zachary Ankner, Tian Jin, Zhiye Song, and Jonathan Ragan-Kelley. Striped attention: Faster ring attention for causal transformers, 2023.

[3] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc V. Le, Mark Z. Mao, Marc'Aurelio Ranzato, Andrew W. Senior, Paul A. Tucker, Ke Yang, and Andrew Y. Ng. Large scale distributed deep networks. In *In 26th Annual Conference on Neural Information Processing Systems*, pages 1232–1240, 2012.

[4] Weiqi Feng, Yangrui Chen, Shaoyu Wang, Yanghua Peng, Haibin Lin, and Minlan Yu. Optimus: Accelerating large-scale multi-modal llm training by bubble exploitation, 2025.

[5] Wikimedia Foundation. Wikimedia downloads.

[6] Hao Ge, Junda Feng, Qi Huang, Fangcheng Fu, Xiaonan Nie, Lei Zuo, Haibin Lin, Bin Cui, and Xin Liu. Bytescale: Communication-efficient scaling of llm training with a 2048k context length on 16384 gpus. In *Proceedings of the ACM SIGCOMM 2025 Conference*, pages 963–978, New York, NY, USA, 2025. Association for Computing Machinery.

[7] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Mia Xu Chen, Dehao Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V. Le, Yonghui Wu, and Zhifeng Chen. Gpipe: efficient training of giant neural networks using pipeline parallelism. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, Red Hook, NY, USA, 2019. Curran Associates Inc.

[8] Sam Ade Jacobs, Masahiro Tanaka, Chengming Zhang, Minjia Zhang, Reza Yazdani Aminadabi, Shuaiwen Leon Song, Samyam Rajbhandari, and Yuxiong He. System optimizations for enabling training of extreme long sequence transformer models. In *Proceedings of the 43rd ACM Symposium on Principles of Distributed Computing*, pages 121–130, New York, NY, USA, 2024. Association for Computing Machinery.

[9] Denis Kocetkov, Raymond Li, Loubna Ben Allal, Jia Li, Chenghao Mou, Carlos Muñoz Ferrandis, Yacine Jernite, Margaret Mitchell, Sean Hughes, Thomas Wolf, et al. The stack: 3 tb of permissively licensed source code, 2022.

[10] Vijay Korthikanti, Jared Casper, Sangkug Lym, Lawrence McAfee, Michael Andersch, Mohammad Shoeybi, and Bryan Catanzaro. Reducing activation recomputation in large transformer models, 2022.

[11] Dacheng Li, Rulin Shao, Anze Xie, Eric Xing, Joseph Gonzalez, Ion Stoica, Xuezhe Ma, and Hao Zhang. Lightseq: : Sequence level parallelism for distributed training of long context transformers. In *Workshop on Advancing Neural Network Training: Computational Efficiency, Scalability, and Resource Optimization (WANT@NeurIPS 2023)*, 2023.

[12] Dacheng Li, Rulin Shao, Anze Xie, Eric P. Xing, Xuezhe Ma, Ion Stoica, Joseph E. Gonzalez, and Hao Zhang. Distflashattn: Distributed memory-efficient attention for long-context llms training, 2024.

[13] Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, and Soumith Chintala. Pytorch distributed: experiences on accelerating data parallel training. *Proc. VLDB Endow.*, 13(12):3005–3018, 2020.

[14] Shenggui Li, Fuzhao Xue, Chaitanya Baranwal, Yongbin Li, and Yang You. Sequence parallelism: Long sequence training from system perspective. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2391–2404, Toronto, Canada, 2023.

[15] Hao Liu, Matei Zaharia, and Pieter Abbeel. Ring attention with blockwise transformers for near-infinite context, 2023.

[16] Tsendsuren Munkhdalai, Manaal Faruqui, and Siddharth Gopal. Leave no context behind: Efficient infinite context transformers with infini-attention, 2024.

[17] Kepan Nan, Rui Xie, Penghao Zhou, Tiehan Fan, Zhenheng Yang, Zhijie Chen, Xiang Li, Jian Yang, and Ying Tai. Openvid-1m: A large-scale high-quality dataset for text-to-video generation. In *The Thirteenth International Conference on Learning Representations*, 2025.

[18] Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R. Devanur, Gregory R. Ganger, Phillip B. Gibbons, and Matei Zaharia. Pipedream: generalized pipeline parallelism for dnn training. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, pages 1–15, New York, NY, USA, 2019. Association for Computing Machinery.

[19] Deepak Narayanan, Amar Phanishayee, Kaiyu Shi, Xie Chen, and Matei A. Zaharia. Memory-efficient pipeline-parallel dnn training. In *International Conference on Machine Learning*, 2020.

[20] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, 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*, pages 1–15, 2021.

[21] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020.

[22] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: memory optimizations toward training trillion parameter models. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2020.

[23] Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *SIGKDD*, pages 3505–3506, 2020.

[24] Kele Shao, Keda TAO, Can Qin, Haoxuan You, Yang Sui, and Huan Wang. Holitom: Holistic token merging for fast video large language models. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

[25] Kele Shao, Keda Tao, Kejia Zhang, Sicheng Feng, Mu Cai, Yuzhang Shang, Haoxuan You, Can Qin, Yang Sui, and Huan Wang. When tokens talk too much: A survey of multimodal long-context token compression across images, videos, and audios, 2025.

[26] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism, 2020.

[27] Yi Wang, Yinan He, Yizhuo Li, Kunchang Li, Jiashuo Yu, Xin Ma, Xinhao Li, Guo Chen, Xinyuan Chen, Yaohui Wang, Conghui He, Ping Luo, Ziwei Liu, Yali Wang, Limin Wang, and Yu Qiao. Internvid: A large-scale video-text dataset for multimodal understanding and generation, 2024.

[28] Yujie Wang, Shiju Wang, Shenhan Zhu, Fangcheng Fu, Xinyi Liu, Xuefeng Xiao, Huixia Li, Jiashi Li, Faming Wu, and Bin Cui. Flexsp: Accelerating large language model training via flexible sequence parallelism. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pages 421–436, New York, NY, USA, 2025. Association for Computing Machinery.

[29] Jun Xu, Tao Mei, Ting Yao, and Yong Rui. Msr-vtt: A large video description dataset for bridging video and language. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[30] 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*, SIGCOMM '25, pages 24–38. ACM, August 2025.

[31] 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. Pytorch fsdp: Experiences on scaling fully sharded data parallel. *Proc. VLDB Endow.*, 16:3848–3860, 2023.

A Details of Experimental Setups

A.1 Dataset Details

Text datasets. We use three text corpora with different sequence-length distributions to evaluate FCP under text-only long-context training.

- **GitHub** contains code-oriented documents with a pronounced long-tail distribution in sequence length. It stresses the ability of FCP to avoid over-provisioning CP groups for short samples while still supporting very long samples. A commonly used public source is The Stack on HuggingFace: `https://huggingface.co/datasets/bigcode/the-stack`.
- **CommonCrawl** contains large-scale web text and has a broad but less code-specific length distribution. We use it to represent general web-scale pre-training data. One public entry point is C4: `https://huggingface.co/datasets/allenai/c4`.
- **Wikipedia** contains encyclopedic articles with a shorter and denser length distribution than GitHub and CommonCrawl. We use it as a comparatively regular text corpus. A public version is available at `https://huggingface.co/datasets/wikimedia/wikipedia`.

Multimodal datasets. We use three video-language datasets with different duration and visual-token distributions to evaluate FCP under multimodal long-context training.

- **MSRVTT** consists of 10,000 video clips paired with 200,000 natural language descriptions from 20 diverse categories, measuring the video-to-text generation and multimodal understanding capabilities of machine learning models. The dataset is available at `https://huggingface.co/datasets/friedrichor/MSR-VTT`.
- **InternVid** comprises 10 million video clips paired with generated high-quality captions sourced from publicly available web videos, facilitating large-scale video-language pre-training and representation learning. The dataset is available at `https://huggingface.co/datasets/OpenGVLab/InternVid`.
- **OpenVid** comprises a curated collection of high-aesthetic video clips with a minimum resolution of 512×512, tailored for research institutions to enhance visual clarity, facilitating high-fidelity text-to-video training and quality-tuning as a complement to existing datasets. The dataset is available at `https://huggingface.co/datasets/nkp37/OpenVid-1M`.

A.2 Model Details

LLM models. For LLM experiments, we evaluate Qwen3-1.7B, Qwen3-4B, Qwen3-8B, Llama3.2-1B, Llama3.2-3B, and Llama3-8B. The selected models cover small and medium LLM settings and allow us to test whether FCP scales with increasing language-model computation. Table 6 summarizes the LLM configurations used in the experiments.

Table 6: **LLM models for evaluation.**

Model	#Layers	#Heads	#Groups	Hidden Dim	FFN Hidden Dim
Qwen3-1.7B	28	16	8	2048	6144
Qwen3-4B	36	32	8	2560	9728
Qwen3-8B	36	32	8	4096	12288
Llama3.2-1B	16	32	8	2048	8192
Llama3.2-3B	28	24	8	3072	8192
Llama3-8B	32	32	8	4096	14336

MLLM models. For MLLM experiments, we evaluate InternVL3, InternVL2.5, and Qwen3VL models at different scales. Table 7 summarizes the model configurations used in the experiments.

A.3 Baseline Details

We use the same two static parallel training baselines for both LLM and MLLM experiments. For each baseline, we tune the parallelism hyperparameters and report the best-performing configuration under the same global batch size.

Table 7: **MLLM models for evaluation.**

Model	#Layers	#Heads	#Groups	Hidden Dim	Vision Hidden Dim
InternVL3-2B	28	12	2	1536	1024
InternVL2.5-4B	36	16	2	2048	1024
InternVL3-8B	28	28	4	3584	1024
Qwen3VL-2B	28	16	8	2048	1024
Qwen3VL-4B	36	32	8	2560	1024
Qwen3VL-8B	36	32	8	4096	1152

- **Deepspeed-Ulysses** [8] addresses the memory constraints imposed by extreme sequence lengths in LLM training through a sequence parallelism mechanism involving hybrid partitioning. Unlike Megatron sequence parallelism [10], Ulysses redistributes attention work across heads using all-to-all communication. To enable global attention without storing the full sequence on a single device, it employs an all-to-all collective communication primitive to transform the data layout.
- **Megatron-LM** [26] composes tensor, pipeline, and data parallelism (PTD-P) to scale language model training to trillions of parameters. To address the idle time inherent in standard pipeline parallelism, it proposes an interleaved pipelining schedule that assigns virtual model chunks to each pipeline stage.

B FCP Optimization Algorithm

Algorithm 1: Optimal degree allocation for fixed atomic groups

Input: Groups $\mathcal{G}_1, \ldots, \mathcal{G}_{K'}$; feasible degree sets $\mathcal{D}_i \subseteq \{1, \ldots, N\}$; costs $\mathcal{T}(\mathcal{G}_i, d)$; rank budget N
Output: Degrees $(d_1, \ldots, d_{K'})$ using exactly N ranks, or infeasible

Set $DP[0{:}K', 0{:}N] \leftarrow +\infty$ and $Path[0{:}K', 0{:}N] \leftarrow \perp$;
$DP[0, 0] \leftarrow 0$;
for $i = 1$ **to** K' **do**
  for $j = 0$ **to** N **do**
    foreach $d \in \mathcal{D}_i$ *with* $d \leq j$ **do**
      if $DP[i-1, j-d] < +\infty$ **then**
        $c \leftarrow \max\{DP[i-1, j-d], \mathcal{T}(\mathcal{G}_i, d)\}$;
        if $c < DP[i, j]$ **then**
          $DP[i, j] \leftarrow c$; $Path[i, j] \leftarrow d$;
if $DP[K', N] = +\infty$ **then**
  return *infeasible*;
$j \leftarrow N$;
for $i = K', K' - 1, \ldots, 1$ **do**
  $d_i \leftarrow Path[i, j]$;
  $j \leftarrow j - d_i$;
return $(d_1, \ldots, d_{K'})$;

The recurrence considers only feasible predecessors. Every backtracking step decreases the group index once, preserving sequence coverage and recovering exactly the allocated rank budget. Costs need not be monotone in the CP degree.

Conditional optimality. Induct on the number of fixed groups. The empty prefix has cost zero only at rank usage zero. Any feasible allocation of the first i groups using j ranks assigns some $d \in \mathcal{D}_i$ to its last group and leaves $j - d$ ranks for the preceding prefix. By the induction hypothesis, that prefix has optimal value $DP[i-1, j-d]$; the combined makespan is the maximum of its value and the last group's cost. Enumerating all such d therefore considers every feasible allocation and selects its minimum makespan. Stored predecessors recover one attaining allocation. This argument assumes the supplied costs depend only on group contents and degree, and establishes no optimality of the preceding BFD grouping.

C Frozen MLLM Results

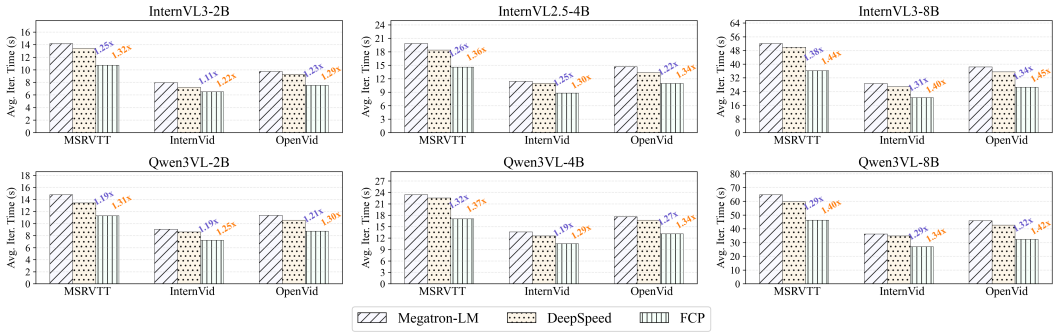


Figure 6: Average iteration time (in seconds) comparison for frozen-vision MLLM training across MSRVTT, InternVid, and OpenVid datasets. FCP keeps the same scheduling mechanism while the vision encoder is frozen, yielding shorter wall time than the unfrozen setting.

Frozen MLLM Training Performance. We evaluate frozen-vision training on the same three multimodal datasets and six model configurations. Against the faster static baseline, Figure 6 shows speedups of 1.11× (InternVL3-2B on InternVid) to 1.38× (InternVL3-8B on MSRVTT), with 14 of 18 configurations exceeding 1.2×. Freezing the vision encoder shortens the training step, while FCP retains benefits from adaptive scheduling.

D Robustness to Profiler Noise

To further study whether profiler errors can change the final scheduling decision, we inject Gaussian noise into each coefficient of the cost model, with standard deviations ranging from 5% to 50% of the original coefficient. Then, we regenerate scheduling strategies with the perturbed models and evaluate them on real clusters.

Table 8. Profiler noise on 32 Devs.

Noise (%)	Time/Step	CP Match (%)
0 (origin)	–	100.0
5	+0.00%	100.0
10	+0.00%	100.0
20	+0.00%	100.0
30	+3.27%	92.7
50	+4.84%	91.3

Table 9. Profiler noise on 64 Devs.

Noise (%)	Time/Step	CP Match (%)
0 (origin)	–	100.0
5	+0.00%	100.0
10	+0.00%	100.0
20	+0.00%	100.0
30	+5.73%	90.4
50	+6.16%	88.8

Tables 8 and 9 show that FCP is robust to realistic profiler noise: up to 20% noise, the generated CP strategies are identical to the original strategy and the measured time per step does not change. Even under 50% noise, the CP match rate remains around 90%, and the overhead is only 4.84% on 32 devices and 6.16% on 64 devices.

The robustness comes from the structure of the cost model. The quadratic term captures the attention cost, while linear terms capture linear-layer computation and communication. These components are positive, and as the sequence length increases the cost is increasingly dominated by the quadratic term. Therefore, moderate coefficient perturbations rarely change the relative ordering of candidate sequence partitions, which is the key information required by the dynamic-programming solver.

E Case Studies

To concretely illustrate FCP's dynamic partitioning of communication groups under varying data distributions, we present two actual parallel strategies on 32 ranks employed within a single global batch. Table 10 reports the heterogeneous CP groups used in the two cases, where each $\langle d \rangle \times m$ indicates m CP=d groups.

Table 10: CP groups assigned by FCP.

Case Study	Case 1	Case 2
DeepSpeed	$\langle 8 \rangle \times 4$	$\langle 4 \rangle \times 8$
Megatron-LM	$\langle 8 \rangle \times 4$	$\langle 4 \rangle \times 8$
FCP	$\langle 8 \rangle \times 1$	
	$\langle 6 \rangle \times 2$	$\langle 4 \rangle \times 2$
	$\langle 4 \rangle \times 1$	$\langle 3 \rangle \times 4$
	$\langle 2 \rangle \times 2$	$\langle 2 \rangle \times 6$
	$\langle 1 \rangle \times 4$	

Case 1 is derived from OpenVid while Case 2 comes from MSRVTT. By mitigating communication redundancy and load imbalance introduced by static CP, FCP achieves speedups of $1.27\times$ in Case 1 and $1.19\times$ in Case 2.

Furthermore, we conducted experiments under some rather extreme data distributions. When computing a batch with an extremely long (128k) sequence and many medium-to-short sequences (2-8k), our speedup even reached 2.24 $\times$.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes].

   Justification: The abstract and introduction state the FCP algorithm, implementation, and LLM/MLLM performance claims, which are evaluated in the experiments.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes].

   Justification: Sections 5 and 7 discuss the evaluated hardware/backend scope, hybrid-mesh limitations, and the model-based interpretation of the exact-reference comparison.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes].

   Justification: Appendix B states the fixed-group and cost-independence assumptions and proves conditional optimality of the degree allocator. No global optimality or approximation-ratio guarantee is claimed for the preceding grouping heuristic.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [No].

   Justification: The paper specifies the scheduling algorithm, workloads, hardware, measurement protocol, and aggregate results. Complete settings and sample-generation details for the added exact-reference study are not provided, so full reproduction of all reported experiments is not supported by the current materials.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [No].

   Justification: The code URL in the abstract provides the core FCP implementation and launch scripts, and Appendix A links public datasets. Complete reproduction scripts for all added ablations, dynamic strategies, and the exact-reference study are not provided.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes].

   Justification: Section 6.1 specifies hardware, workloads, static baselines, global batch sizes, warmup, and averaging. Appendix A provides dataset/model details, and Section 6.5 describes the added scheduling comparisons.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No].

   Justification: Training measurements average 10 steps after 5 warmup steps, and the exact-reference study averages 100 per-batch gaps per configuration. Error bars, confidence intervals, and statistical significance tests are not reported.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes].

   Justification: We report a 4-node Ascend 910C cluster with 64GB memory per NPU and provide iteration-time/throughput measurements.

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes].

   Justification: The work is systems research using public datasets and models, with no human-subject experiments or known deviation from the Code of Ethics.

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A].

    Justification: This work studies general-purpose distributed-training efficiency and does not target a particular societal application or deployment.

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A].

    Justification: The paper does not release new pretrained models or new scraped datasets; it uses existing public datasets as experimental workloads.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [No].

    Justification: The paper credits and links the datasets, models, and baseline systems, but it does not explicitly enumerate the license names and terms for every existing asset.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes].

    Justification: We release the code of FCP via the URL in the abstract.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A].

    Justification: The paper does not involve crowdsourcing or human-subject experiments.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A].

    Justification: The paper does not involve human-subject research, so IRB approval is not applicable.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A].

    Justification: LLMs and MLLMs are the training workloads studied in this paper. The core method is a grouping and resource-allocation algorithm; it does not use an LLM as an important, original, or non-standard methodological component.