DFLOP: A Data-driven Framework for Multimodal LLM Training Pipeline Optimization

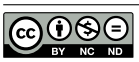
HYEONJUN AN, Yonsei University BDAI Lab, Republic of Korea
SIHYUN KIM, Yonsei University BDAI Lab, Republic of Korea
CHAERIM LIM, Yonsei University BDAI Lab, Republic of Korea
HYUNJOON KIM, Yonsei University BDAI Lab, Republic of Korea
RATHIJIT SEN, Microsoft Gray Systems Lab, United States
SANGMIN JUNG, SK Telecom, Republic of Korea
HYEONSOO LEE, SK Telecom, Republic of Korea
DONGWOOK KIM, SK Telecom, Republic of Korea
TAKKI YU, SK Telecom, Republic of Korea
JINKYU JEONG, Yonsei University, Republic of Korea
YOUNGSOK KIM, Yonsei University, Republic of Korea
KWANGHYUN PARK*, Yonsei University BDAI Lab, Republic of Korea

Multimodal Large Language Models (MLLMs) have achieved remarkable advances by integrating text, image, and audio understanding within a unified architecture. However, existing distributed training frameworks remain fundamentally data-blind: they parallelize computation without accounting for variations in input data characteristics. This data unawareness leads to severe computation skew across stages and microbatches, where heterogeneous multimodal inputs incur different processing costs. Consequently, GPU resources are unevenly utilized, synchronization delays accumulate, and overall training efficiency degrades. To address this limitation, we present DFLOP, a **d**ata-driven **f**ramework for multimodal **LLM** training pipeline **op**timization. DFLOP continuously profiles runtime behavior to capture data-induced computation variance and employs predictive scheduling to balance workloads across stages and microbatches. By coupling data characteristics with execution planning, DFLOP substantially improves GPU utilization and throughput. Extensive experiments on large-scale multimodal benchmarks show that DFLOP achieves up to 3.6× faster training compared to state-of-the-art distributed training frameworks.

CCS Concepts: • **Computing methodologies → Parallel computing methodologies**; • **Information systems → Language models**; • **Computer systems organization → Distributed architectures**; **Parallel architectures**.

*Corresponding author.

Authors' Contact Information: Hyeonjun An, hyeonjun.an@yonsei.ac.kr, Yonsei University BDAI Lab, Seoul, Republic of Korea; Sihyun Kim, sihyun.kim@yonsei.ac.kr, Yonsei University BDAI Lab, Seoul, Republic of Korea; Chaerim Lim, chaerim.lim@yonsei.ac.kr, Yonsei University BDAI Lab, Seoul, Republic of Korea; Hyunjoon Kim, wns41559@yonsei.ac.kr, Yonsei University BDAI Lab, Seoul, Republic of Korea; Rathijit Sen, rathijit.sen@microsoft.com, Microsoft Gray Systems Lab, Redmond, Washington, United States; Sangmin Jung, tojsm@sktelecom.com, SK Telecom, Seoul, Republic of Korea; Hyeonsoo Lee, hyeonsoo.lee@sktelecom.com, SK Telecom, Seoul, Republic of Korea; Dongwook Kim, kimdw@sktelecom.com, SK Telecom, Seoul, Republic of Korea; Takki Yu, takki.yu@sktelecom.com, SK Telecom, Seoul, Republic of Korea; Jinkyu Jeong, jinkyu@yonsei.ac.kr, Yonsei University, Seoul, Republic of Korea; Youngsok Kim, youngsok@yonsei.ac.kr, Yonsei University, Seoul, Republic of Korea; Kwanghyun Park, kwanghyun.park@yonsei.ac.kr, Yonsei University BDAI Lab, Seoul, Republic of Korea.



ACM 2836-6573/2026/6-ART160
https://doi.org/10.1145/3802037

Additional Key Words and Phrases: Multimodal LLM, Distributed training, GPU cluster optimization, Large multimodal models

ACM Reference Format:
Hyeonjun An, Sihyun Kim, Chaerim Lim, Hyunjoon Kim, Rathijit Sen, Sangmin Jung, Hyeonsoo Lee, Dongwook Kim, Takki Yu, Jinkyu Jeong, Youngsok Kim, and Kwanghyun Park. 2026. DFLOP: A Data-driven Framework for Multimodal LLM Training Pipeline Optimization. *Proc. ACM Manag. Data* 4, 3 (SIGMOD), Article 160 (June 2026), 29 pages. https://doi.org/10.1145/3802037

1 Introduction

Multimodal Large Language Models (MLLMs) have rapidly gained prominence as transformative advancements in artificial intelligence, expanding the capabilities of traditional LLMs by integrating diverse data modalities such as text, image, audio, and so on [2, 31, 55, 64, 65, 68]. Their widespread adoption spans various domains, including healthcare, autonomous systems, and human-computer interaction, where their ability to process and generate multimodal content enables new possibilities [12, 16, 60, 62, 66].

While initial research primarily focused on tasks involving paired single images and text, state-of-the-art models are now designed to handle a far richer and more complex set of inputs, frequently processing multiple images or even video frames within a single instance [1, 3, 8, 33]. Consequently, modern MLLM training datasets are characterized by a highly heterogeneous mixture of data types. This growing diversity of input data fundamentally complicates the training process [71]. Despite their potential, training MLLMs remains a challenging task. The process demands immense computational resources, intricate model architectures, and the handling of complex multimodal data, often resulting in inefficient GPU utilization and prolonged training times [19].

Distributed Training and Its Limitations for MLLMs. To meet the immense computational and memory demands of large-scale model training, modern frameworks employ a combination of data [35, 53, 72], tensor [27, 44], and pipeline [28, 43] parallelism. These three techniques are often combined as 3D parallelism [44], the de facto paradigm for training trillion-parameter models in frameworks such as Megatron-LM [44], which build upon PyTorch [47] primitives. However, these distributed frameworks remain fundamentally data-agnostic. They parallelize computation under an assumption of homogeneous workloads, the assumption violated by the diverse and dynamic nature of multimodal data [19, 27, 71]. Two critical challenges arise from this mismatch.

A critical challenge in MLLM training is *load imbalance across heterogeneous stages*. While 3D parallelism systems assume consistent microbatch execution times, MLLM pipelines are inherently heterogeneous, comprising architecturally distinct modality encoders and LLMs. Furthermore, processing diverse visual inputs creates data heterogeneity within microbatches. This violates the homogeneity assumption, leading to severe load imbalances. Concurrently, *performance variability stemming from input-dependent throughput* presents another significant hurdle. A model's achievable throughput is highly sensitive to input tensor shapes. This is particularly problematic for MLLMs, where the modality encoder experiences variable effective batch sizes and the LLM processes sequences with highly variable lengths. This variability is further compounded by tensor parallelism (TP), where different degrees yield distinct throughputs for identical input shapes. Existing data-agnostic systems fail to model or adapt to this input-dependent behavior, resulting in suboptimal parallelism strategies and reduced end-to-end performance.

Towards Data-driven MLLM Training. While 3D parallelism enables unprecedented scaling, its data-blind design renders it inefficient for MLLM training pipelines. The resulting computation skew and synchronization delays highlight the need for a data-driven framework that adapts

distributed training to the characteristics of multimodal data–an approach realized in our proposed system, DFLOP.

DFLOP is a data-driven framework for optimizing MLLM training pipelines. It explicitly models the heterogeneity of both multimodal architecture and input data to co-optimize parallelism configuration and runtime scheduling, addressing stage imbalance and input-dependent throughput variability. DFLOP comprises three major components: (i) Profiling Engine, (ii) Data-aware 3D Parallelism Optimizer, and (iii) Online Microbatch Scheduler. Together, these components bridge data characteristics and distributed execution to achieve balanced and efficient training (§3.1).

The Profiling Engine quantitatively characterizes both the model and the data workload. It measures the model's memory consumption and throughput across a range of input shapes to construct predictive performance models, and concurrently analyzes the input shape distribution of the actual training dataset (§3.2).

The Data-aware 3D Parallelism Optimizer leverages the information provided by the Profiling Engine to determine the 3D parallelism strategy for both the modality encoder and the LLM independently, with the objective of minimizing the expected makespan under the observed data workload (§3.3).

Finally, the Online Microbatch Scheduler addresses uneven computation across individual data items during training. In each step, it calculates estimated computation times and partitions the global batch into microbatches with the objective of balancing the processing load across all pipeline stages, thereby reducing idle GPU time and improving overall throughput (§3.4).

Novelty & Contributions. Our work introduces a data-driven approach that co-designs the static parallelism strategy and the dynamic runtime scheduling for MLLM training. Unlike prior data-agnostic approaches that optimize solely for a given model architecture, DFLOP is the first system, to the best of our knowledge, to mitigate the dual challenges of pipeline imbalance and input-dependent performance variability by explicitly leveraging data distribution statistics and component-level performance profiles. By treating the workload as a first-class component of the optimization problem, our approach mitigates critical structural inefficiencies in large-scale MLLM training pipelines. The main contributions of this paper are as follows:

- **Formalization of MLLM Parallelism Challenges** (§2.3): We are the first to formally identify and study the dual challenges of stage load imbalance and input-dependent throughput variability in the context of 3D parallelism for MLLMs, demonstrating that data heterogeneity is a critical performance bottleneck.
- **The DFLOP Framework** (§3): We design and implement DFLOP, a novel framework that resolves the dual challenges of MLLM training. DFLOP comprises three interconnected components: a Profiling Engine that gathers essential model and data characteristics, a Data-aware 3D Parallelism Optimizer that utilizes this information to determine independent 3D parallelism strategies for the modality encoder and the LLM, and an Online Microbatch Scheduler that operates asynchronously during training to dynamically schedule microbatches and balance workloads in real-time.
- **A Flexible and Open-source Parallelism Framework** (§4): We implement a new parallelism framework on top of PyTorch that enables separate and independent 3D parallelism for the modality encoder and the LLM. This includes a novel Inter-model Communicator abstraction to efficiently handle communication between disparate data-parallel groups, which we release as open source[1].
- **Comprehensive Experimental Evaluation** (§5): We conduct a comprehensive evaluation using state-of-the-art MLLMs and realistic, heterogeneous datasets. Our experiments demonstrate that

[1]https://github.com/BDAI-Research/DFLOP

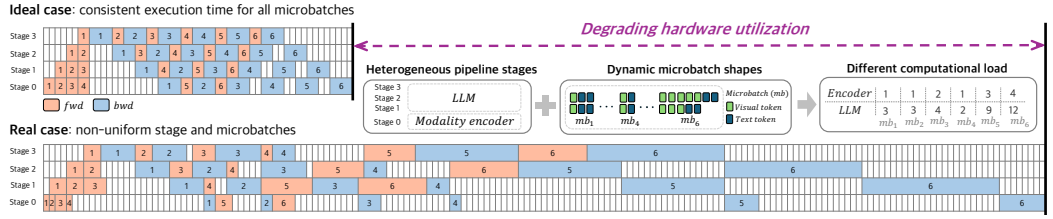


Fig. 1. 1F1B [17] pipeline schedules with backward passes (blue) taking twice as long as the forward pass (pink) with 6 microbatches (represented by numbers) from a mixed dataset consisting of single image [32, 37, 40], multiple images [30], and videos [70] on LLaVA-OV [33]. The top figure illustrates the ideal 1F1B case, assuming all microbatches have the same execution time. The bottom figure shows the real 1F1B case, where modality encoder is assigned to stage 0 and LLM is assigned to stages 1, 2, and 3. The size of each microbatch in the pipeline varies due to heterogeneous stages and dynamic microbatch shapes. Modality encoder processes only visual tokens, while LLM processes both visual and text tokens.

DFLOP substantially outperforms existing baseline systems, improving end-to-end throughput by up to 3.6×, without compromising model accuracy.

2 Background and Motivation

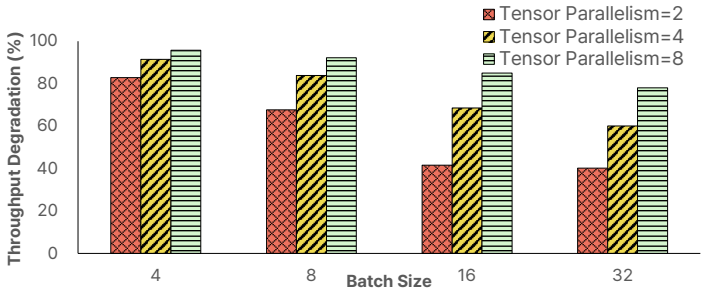
2.1 MLLM Architecture

MLLMs are designed to extend the capabilities of traditional LLMs by incorporating multiple data modalities beyond text. MLLMs typically comprise three key submodules: modality encoders, an LLM, and connectors that seamlessly integrate the two components.

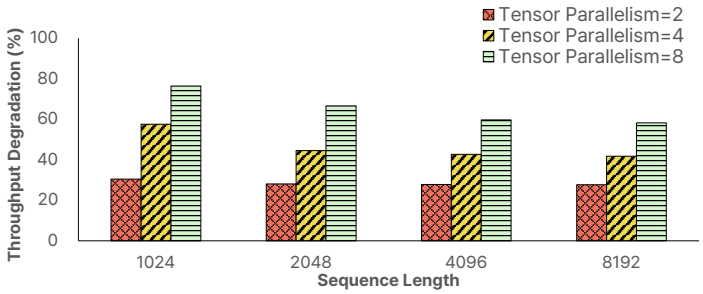
Modality encoders are specialized for processing non-textual data such as images, videos, audio, and 3D point clouds by encoding them into vector representations suitable for LLM. Images, sampled video frames, audio data, and 3D point cloud data are preprocessed and encoded to extract visual [5, 15, 18, 52, 59, 69], auditory [6, 7, 26], and spatial [63] features, respectively. LLM [2, 9, 11, 24, 57, 58] serves as the backbone of the MLLM, processing textual inputs and modality features provided by the encoders to generate textual outputs. Since LLMs are inherently designed to process only textual data, bridging the gap between natural language and other modalities is critical. To address this, a learnable connector is introduced to align the modality encoder outputs with a format compatible with the LLM. These connectors can be implemented in various forms, including linear projectors, multi-layer perceptrons (MLPs), and advanced architectures [29, 34, 39].

2.2 Scaling Visual Inputs in Modern MLLMs

To address the challenge of processing varied visual inputs, a critical innovation has been the development of dynamic resolution [3, 8, 33], a departure from earlier models that uniformly resized images, often at the cost of crucial information. This paradigm enables MLLMs to process images at much higher fidelity and in their native aspect ratios. Orthogonal to these improvements in single-image fidelity, the focus of MLLM development is also expanding beyond single-image contexts. A significant trend is the development of models capable of processing interleaved sequences of multiple images and videos within a single training instance. This evolution is driven by the need to perform more complex reasoning and comprehension tasks that require understanding relationships across multiple visual inputs or over time. Recent models explicitly incorporate multi-image and video data into their training pipelines [1, 8, 33], enabling task transfer between modalities and unlocking new capabilities in areas like comparative reasoning and temporal understanding. This move towards handling heterogeneous, multi-visual inputs represents the next step in building



(a) Throughput degradation of modality encoder (SigLIP) varying effective batch size.



(b) Throughput degradation of LLM (Qwen-2.5) varying sequence length.

Fig. 2. Throughput variation with respect to input shape, measured on an 8-GPU HGX A100 node interconnected via NVLink. Throughput degradation with increasing tensor parallelism (TP) arises from computation partitioning. This can lead to GPU underutilization if the resulting workload fragments are too small, compounded by the inherent communication overhead required for synchronization.

more versatile and capable MLLMs, consequently leading to a substantial increase in the number of visual tokens processed per instance.

2.3 Motivation

The training paradigm for MLLMs has evolved to incorporate increasingly diverse data modalities, including mixtures of a single image, multiple images, and video frames. This growing workload heterogeneity fundamentally challenges the design principles of existing 3D parallelism frameworks, which are predicated on a flawed assumption of workload homogeneity. This mismatch leads to critical, interconnected challenges in MLLM training, primarily load imbalance caused by non-uniform computational loads from architecturally distinct stages and heterogeneous data within microbatches, coupled with input-dependent throughput variability, where the achievable throughput of these stages is not a static value but a dynamic function of the shape of the input data.

Non-uniform Computation Time Across Stages and Microbatches. A core premise of pipeline parallelism is the uniform execution time of microbatches, which enables a tightly packed schedule with minimal idle time, as depicted in the ideal case in Figure 1. However, this premise is violated in MLLM training due to two sources of heterogeneity. First, the pipeline stages themselves are architecturally distinct, comprising different modules such as a modality encoder and an LLM. Second, the workload within each microbatch is non-uniform, as a single global batch can contain

Table 1. Notations.

Notations	Description
N_{mb}	Number of microbatches
E_{batch_size}	Effective batch size of modality encoder
E_{seq_len}	Number of visual tokens on modality encoder
L_{seq_len}	Number of text and visual tokens on LLM.
E_{dp}, E_{pp}, E_{tp}	Degrees of data, pipeline, and tensor parallelism on the modality encoder
L_{dp}, L_{pp}, L_{tp}	Degrees of data, pipeline, and tensor parallelism on LLM
E_l	Number of layers on modality encoder
L_l	Number of layers on LLM
N_{gpus}	Total Number of GPUs
N_{gpu_node}	Number of GPUs per node
M_{gpu}	Memory capacity of a single GPU
E_{mem}	GPU Memory used by modality encoder
L_{mem}	GPU Memory used by LLM
GBS	Global Batch Size

a mix of data types with varying computational demands. As shown in the real case of Figure 1, this dual heterogeneity results in severe load imbalances across pipeline stages. Systems that do not account for this non-uniformity incur substantial GPU idle time, which significantly inflates the end-to-end training time.

Input-dependent Throughput Variability. A model's throughput in real workloads is not a static value but a complex function of its input tensor's shape, particularly effective batch size and sequence length. As illustrated in Figure 2, the throughput degradation when increasing the TP degree from one varies significantly depending on these input dimensions. This variability is critical in MLLM training, where the modality encoder processes inputs with fluctuating effective batch sizes and the LLM handles sequences of highly variable lengths. Consequently, a statically tuned 3D parallelism strategy must navigate an intricate trade-off space. The potential throughput degradation from a higher TP degree must be balanced against the performance and memory implications of the corresponding data parallel (DP) and pipeline parallel (PP) degrees. Crucially, this optimization cannot be performed based on a single, pre-configured input shape. Instead, it must consider the entire distribution of input shapes present in the actual training dataset to minimize the expected makespan. Existing data-agnostic systems fail to model this complex interplay and thus result in parallelism configurations that are misaligned with the true workload, leading to suboptimal end-to-end throughput.

3 DFLOP

To address the challenges of load imbalance and performance variability in MLLM training, we designed DFLOP, a data-driven framework for optimizing MLLM distributed training. This section details the architecture and core components of our system. The key notations used throughout this section are summarized in Table 1. We begin with a high-level overview (§3.1), followed by a detailed description of our three key components. These are the Profiling Engine for gathering essential performance and data characteristics (§3.2), the Data-aware 3D Parallelism Optimizer for determining a statically tuned parallelism strategy in an offline phase (§3.3), and the Online Microbatch Scheduler for performing dynamic load balancing at runtime (§3.4).

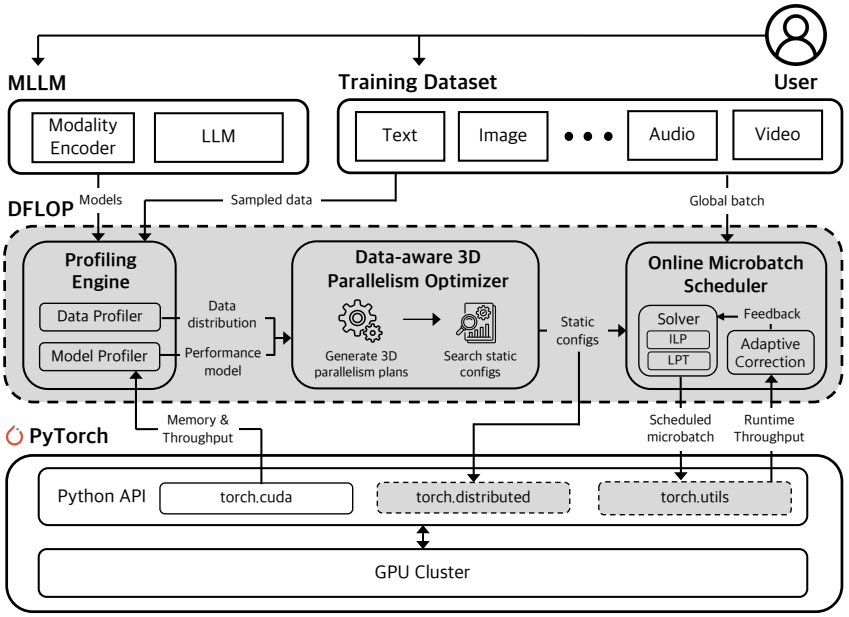


Fig. 3. DFLOP's three main modules and their integration with an MLLM training framework (PyTorch) shown in gray.

3.1 Overview

This section details the architecture and the operational flow of DFLOP, as illustrated in Figure 3. Our system integrates three distinct components to optimize MLLM training. The Profiling Engine and the Data-aware 3D Parallelism Optimizer operate prior to the main training process to gather essential performance characteristics and determine a static parallelization strategy. Subsequently, during the training execution, the Online Microbatch Scheduler operates asynchronously, scheduling the next data batch in parallel with the model's current computation to eliminate scheduling overhead.

The process begins with the Profiling Engine, which takes the user-provided MLLM and training dataset as input. It consists of two sub-components, a Model Profiler and a Data Profiler, that decouple system characterization from workload analysis. The Model Profiler constructs a general, reusable performance model by measuring the throughput and memory consumption of the MLLM on a wide range of synthetic data. Concurrently, the Data Profiler extracts an empirical distribution of input shapes by sampling the specific training dataset. The resulting dataset distribution and performance models are then passed to the Data-aware 3D Parallelism Optimizer.

The Data-aware 3D Parallelism Optimizer then uses this information to determine the static configuration. It first generates a comprehensive search space of all possible 3D parallelism strategies for both the modality encoder and the LLM, given the number of available GPUs. It then verifies each configuration against the memory constraints predicted by the performance models. From the configurations that satisfy these constraints, it selects the configuration that minimizes the objective function, which is the expected makespan. This configuration is passed to our custom 3D parallelism framework, built on torch.distributed, which partitions the model and deploys it across the GPU cluster. The configuration details are also forwarded to the Online Microbatch Scheduler.

Finally, during the training execution, the Online Microbatch Scheduler operates. At each iteration, it receives a global batch of data. Using the configuration provided by the optimizer, it predicts the execution duration for each individual data item within the batch. Based on these calculations, it partitions the global batch into the specified number of microbatches in a way that balances the load. This mechanism is integrated into the data loading pipeline, built upon torch.utils.

3.2 Profiling Engine

The Profiling Engine is an offline component that gathers the empirical data needed for our optimization. It performs two complementary functions. A Model Profiler uses synthetic data to build a general performance model for memory and throughput. In parallel, a Data Profiler analyzes the actual training dataset to create an empirical distribution of its input shapes.

3.2.1 Model Profiler. The Model Profiler leverages diverse synthetic data to construct a generalizable performance model spanning a continuous space of input dimensions. This approach characterizes the model's memory and throughput behavior across the entire input spectrum, rather than being coupled to the sparse distribution of a specific training workload.

We employ empirical profiling as static analytical models fail to capture the non-deterministic execution dynamics inherent to MLLMs. Theoretical formulations often obscure the non-linear performance shifts induced by dynamic input shapes, as illustrated in Figure 2, and the scaling disparity between memory-bound attention and compute-bound linear operations [13, 14]. Furthermore, they remain agnostic to low-level runtime intricacies such as specialized kernels [50, 51]. Capturing these ground-truth characteristics is thus indispensable for ensuring precise load balancing and preventing unexpected OOM errors.

Memory Profiling. To construct a predictive model for GPU memory consumption, we first profile the target MLLM across a grid of specific configurations. This grid is defined by varying the number of layers between two distinct small values and scaling the TP degree in powers of two up to N_{gpu_node}. From these measurements, we characterize the total memory footprint as a composition of two primary components: model states and activation states.

Model states, which encompass parameters, gradients, and optimizer states, are modeled through linear interpolation to derive $model\_state_E(l, E_{tp})$ for the modality encoder and $model\_state_L(l, L_{tp})$ for the LLM, where l denotes the total number of layers. For activation states, which store intermediate results for backpropagation, we account for the distinct input characteristics of each module. While E_{seq_len} remains fixed for the modality encoder, we employ sequence packing [45] for the LLM to concatenate instances, effectively fixing the batch size to 1 while making L_{seq_len} highly variable. Consequently, we model activation memory via linear interpolation based on the effective batch size for the modality encoder and sequence length for the LLM. This yields two distinct functions: $act\_state_E(l, E_{tp}, E_{batch\_size}, E_{seq\_len})$ and $act\_state_L(l, L_{tp}, 1, L_{seq\_len})$.

Throughput Profiling. To characterize computational efficiency, the Profiling Engine measures the model across a grid of key dimensions: the effective batch size and TP degree for the modality encoder, and the sequence length and TP degree for the LLM.

For the modality encoder, the function $E_{thr}(E_{batch\_size}, E_{tp})$ is derived by linearly interpolating over the effective batch size and TP degree. For the LLM, the core operations including attention and fully-connected layers exhibit distinct scaling behaviors relative to the packed sequence. Attention operations are dependent on individual sequence lengths and must process each original instance separately to maintain causal integrity. In contrast, linear operations depend on the hidden size and can be applied to the entire concatenated sequence at once. To accurately capture these behaviors, these two operations are measured independently and fit using linear interpolation

to derive separate throughput models: $L_{attn\_thr}(1, L_{seq\_len}, L_{tp})$ for the attention mechanism and $L_{lin\_thr}(1, L_{seq\_len}, L_{tp})$ for the linear layers.

3.2.2 Data Profiler. The specific dimensions that vary and their impact on the model's inputs are strictly governed by the MLLM's architecture and its data processing pipeline. For instance, some models partition vision data into a variable number of fixed-size patches based on resolution [33, 36], while others employ a variable image sequence length [3]. Similarly, certain architectures preserve the full sequence dimension of modality vectors, whereas others utilize connectors to reduce this dimension to alleviate the computational burden on the LLM [8]. These model-specific transformations imply that even with the same dataset, the input shape distributions for the core modules can differ drastically. To characterize this workload, the Data Profiler first identifies the varying input dimensions for both the modality encoder and the LLM. It then performs random sampling across the dataset, calculating the precise input shapes for each sampled item within the target architecture to construct empirical histograms that represent the statistical distribution of the workload.

3.2.3 Re-profiling Conditions. The necessity for re-profiling is determined by changes in the structural or statistical characteristics of the workload. The Model Profiler requires re-profiling only when the model architecture is modified, as its performance model is built upon the hardware-specific execution behavior of fixed operations. In contrast, the Data Profiler must be re-executed when either the model architecture or the underlying dataset changes, as the input shape distributions are jointly determined by the model's preprocessing logic and the raw data characteristics.

3.3 Data-aware 3D Parallelism Optimizer

The Data-aware 3D Parallelism Optimizer identifies a static parallelism strategy, comprising E_{tp}, E_{pp}, E_{dp} for the encoder, L_{tp}, L_{pp}, L_{dp} for the LLM, and the number of microbatches per pipeline stage N_{mb}. It minimizes the expected makespan subject to the constraints on per-GPU memory capacity and the total number of GPUs, leveraging the throughput and memory models from the Profiling Engine and the training dataset's input statistics provided by the Data Profiler.

3.3.1 Problem Formulation. The primary objective of the optimizer is to find a 3D parallelism configuration that minimizes the training iteration time, or makespan. The makespan (T) of a pipelined execution is determined by the N_{mb}, the pipeline depth, and the duration of the slowest stage. For an MLLM, the total pipeline depth is the sum of the respective pipeline degrees ($E_{pp} + L_{pp}$), and the makespan is defined as:

$$T = (N_{mb} + E_{pp} + L_{pp} - 1) \cdot \max(E_{dur}, L_{dur})$$

Here, E_{dur} and L_{dur} represent the execution durations for a microbatch on the modality encoder and the LLM. We define a parameter vector $\theta = (E_{tp}, E_{pp}, E_{dp}, L_{tp}, L_{pp}, L_{dp}, N_{mb})$ to represent a complete 3D parallelism strategy. For data d in the sampled dataset D, the duration for each module is then expressed as:

$$E_{dur}(d;\theta) = \frac{E_{FLOP}(d;\theta)}{E_{thr}(b(d), E_{tp})},\ L_{dur}(d;\theta) = \frac{L_{FLOP}(d;\theta)}{L_{thr}(s(d), L_{tp})}$$

The throughput functions, E_{thr} and L_{thr}, depend on the input shape, specifically the modality encoder's effective batch size $b(d)$ and the LLM's sequence length $s(d)$, as well as the chosen tensor parallelism degrees. Because both the computational load per GPU ($E_{FLOP}(d;\theta)$, $L_{FLOP}(d;\theta)$) and the throughput functions (E_{thr}, L_{thr}) are dependent on each data item d, the stage durations

(E_{dur}, L_{dur}) are not static values. Consequently, the makespan, which we denote as $T(d;\theta)$, is also a variable dependent on the specific data item being processed. A data-agnostic approach that relies on a single point-estimate for the workload is therefore insufficient. Instead, our objective is to find the parameter vector θ^* that minimizes the expected makespan over the entire distribution of the sampled training dataset D. This is formulated as:

$$\theta^* = \arg\min_{\theta \in \Theta} \frac{1}{|D|} \sum_{d \in D} T(d;\theta) \tag{1}$$

The optimization is performed over a set of valid configurations Θ, defined by hardware and memory constraints. The TP degrees are typically limited to GPUs within the same node, as tensor parallelism requires high-bandwidth, low-latency communication.

$$E_{tp} \in \{1, 2, \ldots, N_{gpu\_node}\},\ L_{tp} \in \{1, 2, \ldots, N_{gpu\_node}\} \tag{2}$$

The total number of GPUs utilized must equal the number available.

$$E_{pp} \cdot E_{tp} \cdot E_{dp} + L_{pp} \cdot L_{tp} \cdot L_{dp} = N_{gpus} \tag{3}$$

Finally, the memory footprint for each module must not exceed the per-GPU memory capacity, M_{gpu}. The memory model incorporates both model states and activation states. A critical consideration is that the activations for the modality encoder must be retained in memory for the duration of the entire pipeline, making their memory cost proportional to the total pipeline depth, $E_{pp} + L_{pp}$. The memory constraints for the modality encoder and the LLM are therefore:

$$Mem_E(p_E) = model\_state_E\left(\frac{E_l}{E_{pp}}, E_{tp}\right) + (E_{pp} + L_{pp}) \cdot act\_state_E\left(\frac{E_l}{E_{pp}}, E_{tp}, E_{batch\_size}, E_{seq\_len}\right) \leq M_{gpu}, \tag{4}$$

where $p_E = \{E_{tp}, E_{pp}, E_{batch\_size}, E_{seq\_len}, L_{pp}\}$.

$$Mem_L(p_L) = model\_state_L\left(\frac{L_l}{L_{pp}}, L_{tp}\right) + L_{pp} \cdot act\_state_L\left(\frac{L_l}{L_{pp}}, L_{tp}, 1, L_{seq\_len}\right) \leq M_{gpu} \tag{5}$$

where $p_L = \{L_{tp}, L_{pp}, L_{seq\_len}\}$.

Consequently, the Data-aware 3D Parallelism Optimizer aims to find θ^* that minimizes the expected makespan in Equation 1, subject to the hardware and memory constraints defined by Equations 2, 3, 4, and 5.

3.3.2 Parallelism Optimization. To solve the optimization problem, we employ the search algorithm detailed in Algorithm 1. The algorithm proceeds in two main phases. The first phase enumerates the entire search space. It begins by iterating through all possible ways to partition the available GPUs (N_{gpus}) between the modality encoder and the LLM. For each partition, the algorithm then identifies all valid 3D parallelism combinations including TP, PP, and DP for each module to create a comprehensive set of candidate strategies $P_{configs}$.

The second phase evaluates this space to select the configuration that minimizes the expected makespan. The algorithm iterates through each candidate in $P_{configs}$ and calculates the theoretical maximum number of microbatches N_{max_mbatch}. It then initiates an inner loop that iterates through each possible microbatch count i. Within this loop, the algorithm performs a memory feasibility check. Using the performance models from the Profiling Engine, it estimates the peak memory consumption for both modules denoted as E_{mem} and L_{mem}. Any configuration predicted to

Algorithm 1 Find Data-aware MLLM 3D Parallelism Configuration

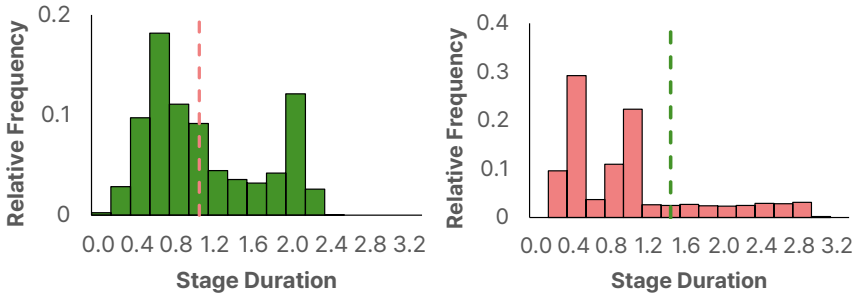
Input: $N_{gpus}, M_{gpu}, Profiled.Data, GBS$
Output: $\theta^* = (E^*_{tp}, E^*_{pp}, E^*_{dp}, L^*_{tp}, L^*_{pp}, L^*_{dp}, N^*_{mb})$

Phase 1: Generate all possible 3D parallelism configurations
$P_{configs} \leftarrow []$
for $E_{gpus} \leftarrow 1$ **to** $N_{gpus} - 1$ **do**
  $L_{gpus} \leftarrow N_{gpus} - E_{gpus}$
  $E_{configs} \leftarrow FindCombs(E_{gpus})$
  # Find all 3D parallel combinations (TP, PP, DP) whose product is E_gpus
  $L_{configs} \leftarrow FindCombs(L_{gpus})$
  # Find all 3D parallel combinations (TP, PP, DP) whose product is L_gpus
  $P_{configs}.add(E_{configs} + L_{configs})$
end for

Phase 2: Find the configuration θ^* which minimizes T
$min\_T \leftarrow +\infty,\ \theta^* \leftarrow$ NULL
$(mean\_bsz, mean\_seq\_len) \leftarrow Profiled.Data.mean()$
for $(E_{tp}, E_{pp}, E_{dp}, L_{tp}, L_{pp}, L_{dp})$ **in** $P_{configs}$ **do**
  $N_{max\_mbatch} \leftarrow GBS // L_{dp}$
  for $i = 1$ to N_{max_mbatch} **do**
    $t\_bsz = mean\_bsz \cdot GBS/(i \cdot E_{dp})$
    $t\_seq\_len = mean\_seq\_len \cdot GBS/(i \cdot L_{dp})$
    $E\_mem \leftarrow Mem_E\left(E_{tp}, E_{pp}, t\_bsz, E_{seq\_len}, L_{pp}\right)$
    $L\_mem \leftarrow Mem_L\left(L_{tp}, L_{pp}, t\_seq\_len\right)$
    if $E\_mem > M_{gpu}$ **or** $L\_mem > M_{gpu}$ **then**
      continue
    end if
    $E_{dur} \leftarrow E_{FLOP}/\left(E_{thr}(t\_bsz, E_{tp}) \cdot E_{tp} \cdot E_{pp}\right)$
    $L_{dur} \leftarrow L_{FLOP}/\left(L_{thr}(t\_seq\_len, L_{tp}) \cdot L_{tp} \cdot L_{pp}\right)$
    $T \leftarrow (i + E_{pp} + L_{pp} - 1) \cdot \max\left(E_{dur}, L_{dur}\right)$
    if $T < min\_T$ **then**
      $min\_T \leftarrow T$
      $\theta^* \leftarrow (E_{tp}, E_{pp}, E_{dp}, L_{tp}, L_{pp}, L_{dp}, i)$
    end if
  end for
end for
return θ^*

exceed the GPU memory capacity M_{gpu} is discarded. For memory-feasible instances, the algorithm calculates the makespan T using the average FLOP from the dataset analysis in conjunction with the profiled throughput models to approximate expected stage durations. The algorithm records the configuration θ^* that yields the minimum estimated makespan across all evaluated candidates and returns this as the selected strategy.

Parallelism Optimization Time Complexity Analysis. The time complexity of Algorithm 1 stems from search space generation and configuration evaluation. The generation phase depends on the number of valid configurations, bounded by the divisor function $d(n) \leq O(n^\epsilon)$ for any arbitrarily small $\epsilon > 0$ [56]. Summing the products of divisors across all GPU partitions yields a total search space size of $O((N_{gpus})^{1+\epsilon})$. Since the evaluation phase iterates through microbatch



(a) Stage duration of modality encoder. (b) Stage duration of LLM.

Fig. 4. Stage-wise duration distributions across data items for (a) modality encoder (SigLIP) and (b) LLM (Qwen-2.5) stages under a mixed dataset. Vertical lines indicate the mean duration.

counts up to the *GBS*, the overall complexity is $O(GBS \cdot (N_{gpus})^{1+\epsilon})$. This sub-polynomial growth ensures that the optimization overhead remains negligible even for large clusters.

3.4 Online Microbatch Scheduler

The Data-aware 3D Parallelism Optimizer establishes a static configuration based on dataset statistics. However, this approach cannot account for the workload variations that occur at runtime within individual batches. During training, the pipeline divides the global batch into m buckets where m is defined as the product of N_{mb} and L_{dp}. Existing scheduling strategies assign data items to these buckets in a random manner. As demonstrated in Figure 4, this random assignment results in high variability in stage durations due to the heterogeneity of input data shapes. Consequently, this variability induces pipeline bubbles.

To address this dynamic workload variance, we introduce the Online Microbatch Scheduler. It dynamically partitions the global batch items into m buckets to minimize the variance in stage durations. By leveraging per-instance duration calculations, the scheduler actively balances the computational load across all pipeline stages to minimize pipeline bubbles and maximize hardware utilization.

3.4.1 Problem Formulation. The core task of the Online Microbatch Scheduler is to solve a load-balancing problem for each incoming global batch. For a given global batch consisting of N data items, the goal is to partition these items into m distinct buckets to minimize the pipeline's makespan for that iteration. We formulate this partitioning task as an Integer Linear Programming (ILP) problem. The objective is to minimize the bottleneck, represented by C_{max}, which is the maximum execution time across any stage of any microbatch. The complete formulation is as follows:

$$\textbf{minimize} \quad C_{max} = \max\left(\max_{1\le j\le m} E_j, \max_{1\le j\le m} L_j\right) \tag{6}$$

$$\textbf{subject to:} \quad \sum_{j=1}^{m} x_{ij} = 1, \qquad \forall i \in \{1,\dots,N\}$$

$$\sum_{i=1}^{N} E_{dur}(d_i;\theta^*)\cdot x_{ij} = E_j, \qquad \forall j \in \{1,\dots,m\}$$

$$\sum_{i=1}^{N} L_{dur}(d_i;\theta^*)\cdot x_{ij} = L_j, \qquad \forall j \in \{1,\dots,m\}$$

$$x_{ij} \in \{0,1\}, \qquad \forall i,j$$

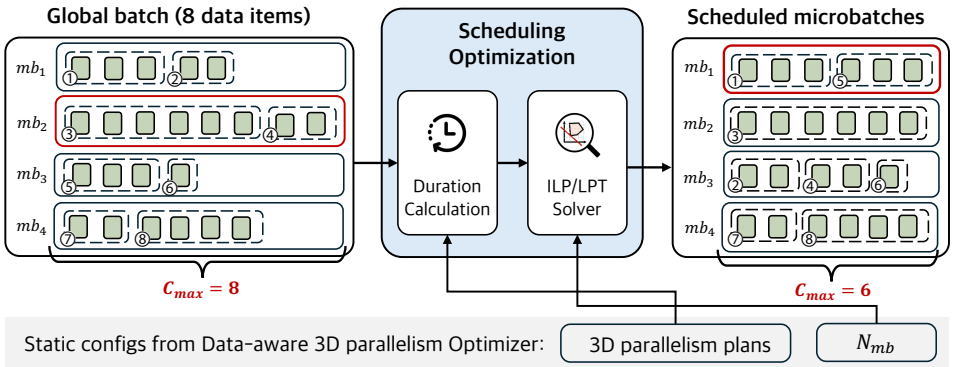


Fig. 5. The scheduling optimization process dynamically partitions each global batch with N data items into N_{mb} microbatches, when $N = 8$ and $N_{mb} = 4$.

In this formulation, the binary decision variable x_{ij} is 1 if data item d_i is assigned to microbatch j, and 0 otherwise. The first constraint ensures that each data item is assigned to exactly one microbatch. The subsequent two constraints define the total execution time for the modality encoder stage (E_j) and the LLM stage (L_j) for each microbatch j. These are calculated by summing the durations ($E_{dur}(d_i; \theta^*)$ and $L_{dur}(d_i; \theta^*)$) of the data items assigned to that microbatch. The objective function then seeks to minimize the maximum of these stage durations across all microbatches.

3.4.2 Scheduling Optimization. The process for solving the load balancing problem is detailed in Figure 5. The scheduler operates asynchronously to eliminate scheduling overhead. While the model executes the computation for the current iteration, the scheduler processes the subsequent global batch in parallel on the CPU. The scheduler begins by receiving the static 3D parallelism plan and N_{mb} from the Data-aware 3D Parallelism Optimizer. When a new global batch of N data items arrives, the scheduler first calculates the execution duration for each item. This calculation is derived by dividing the computational load of each item by its expected throughput under the active 3D parallel plan.

These per item duration values serve as inputs to a hybrid solving mechanism. To ensure low runtime latency, we first employ an ILP solver with a strict time limit to search for a solution that minimizes the objective function C_{max}. If the solver fails to converge within the allocated time, the scheduler reverts to the Longest Processing Time (LPT) heuristic. This greedy approach sorts the data items in descending order of duration and iteratively assigns each item to the microbatch with the lowest current accumulated load. Upon determining a valid assignment through either method, the mechanism returns a set of index groups that defines the target partition of the global batch. Based on these indices, the scheduler partitions the data into scheduled microbatches.

Scheduling Optimization Time Complexity Analysis. The complexity of the hybrid scheduling mechanism derives from its two components. The ILP formulation assigns each of the GBS items to one of m buckets, resulting in an exponential worst-case complexity of $O(m^{GBS})$. Despite this theoretical bound, the solver remains efficient by leveraging branch pruning to reduce the search space [22, 48], while the scheduler utilizes asynchronous execution to hide solving latency. The fallback LPT heuristic incurs polynomial complexity; by iteratively assigning items to the bucket with the lowest accumulated load, it achieves a complexity of $O(GBS \cdot \log m)$ [20].

3.4.3 Adaptive Correction. Under stable operating conditions, stage throughput is largely predictable as a function of input shape using profiling-based interpolation. However, modern GPU

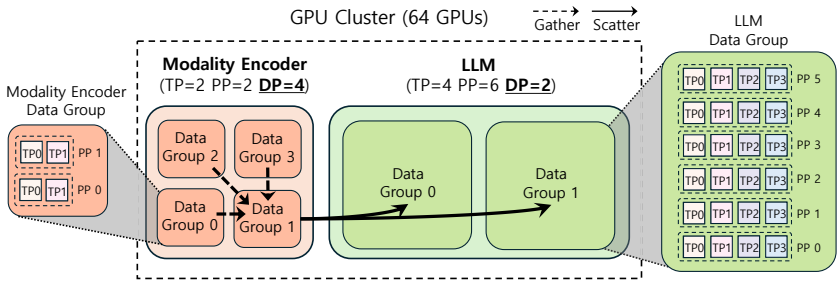


Fig. 6. The Inter-model Communicator resolves data group mismatches in heterogeneous configurations. In the forward pass, activations from the modality encoder are gathered and scattered to the LLM data groups. This operation reverses during the backward pass to propagate gradients from the LLM back to the modality encoder.

execution stacks frequently invoke different specialized kernels or optimized execution strategies depending on the specific dimensions of the input tensor [25]. This behavior leads to non-smooth and regime-dependent performance characteristics. Consequently, a small subset of rare input shapes may exhibit throughput behavior that consistently deviates from interpolation-based predictions.

The Adaptive Correction mechanism addresses these deviations by continuously tracking the execution metrics. The system quantifies the expected benefit B for a given input shape as the difference between the actual and predicted throughput:

$$B = Th_{actual} - Th_{pred} \tag{7}$$

where Th_{actual} and Th_{pred} denote the measured runtime throughput and the interpolated prediction, respectively. This deviation translates to potential makespan degradation in the worst-case scenario. To mitigate this, the mechanism identifies such edge cases and provides feedback to the scheduling solver by updating a penalty function with the observed data. The scheduling solver utilizes this adjusted information when partitioning future global batches to ensure accurate load balancing.

To manage the overhead associated with continuous monitoring, we employ a cost-benefit analysis. The cost C is quantified during the initial training phase by measuring the throughput differential when the tracking mechanism is first deactivated and subsequently activated. The system evaluates the average benefit B over a sequence of I iterations. If this average B fails to exceed the recurring cost C, the tracking mechanism is deactivated, allowing the training process to proceed without this monitoring overhead to preserve overall efficiency.

4 Implementation

The system is implemented on top of PyTorch [47] as it offers the necessary flexibility to leverage native distributed communication primitives and parallelism functionalities. The core contribution lies in the development of a mechanism that enables independent 3D parallelism for distinct model components, a feature not supported by standard frameworks.

DFLOP supports heterogeneous 3D parallel configurations for the modality encoder and the LLM. This flexibility contrasts with conventional distributed training frameworks that enforce identical TP and DP degrees across the entire model, which necessitates static alignment of data groups and restricts module-specific optimization. To eliminate these constraints, the available GPUs are partitioned into distinct process groups for each module. While allowing distinct data groups expands the search space for efficient configurations, it introduces a communication mismatch when the data groups of the two modules differ in size.

Table 2. Composition of the mixed dataset.

Dataset Name	Data Type	# of Samples
LLaVA-Wild [37]	Single Image	28k
AI2D [32]		18k
Infographic VQA [40]		19k
M4-Instruct [30]	Multiple Images	60k
LLaVA-Video [70]	Video	60k

Table 3. Model configurations of the evaluated MLLMs.

MLLM Architecture	Modality Encoder	LLM Backbone
LLaVA-OV [33]	SigLIP [67]	Qwen-2.5 [2] (7B, 32B, 72B) Llama-3 [21] (8B, 70B)
InternVL-2.5 [8]	InternViT [8]	Qwen-2.5 [2] (72B)

As illustrated in Figure 6, we introduce an inter-model communicator mechanism to resolve this disparity. Conventional frameworks cannot facilitate communication in the scenario shown in the figure, where the modality encoder is configured with $DP = 4$ and the LLM with $DP = 2$, due to the group size mismatch. DFLOP overcomes this by designating a single predetermined rank within the modality encoder's data group as the communicator. Specifically, the communicator gathers output activations from the four data groups of the modality encoder and subsequently scatters them to the two corresponding data groups of the LLM during the forward pass. Conversely, during the backward pass, the mechanism operates in reverse, gathering gradients from the LLM groups and scattering them back to the modality encoder groups to ensure consistency.

Our decoupled design generalizes to frameworks supporting native 3D parallelism and flexible process group management, such as PyTorch and Megatron-LM. However, DFLOP is currently incompatible with DeepSpeed, as it does not natively support 3D parallelism and lacks the flexible pipeline parallelism primitives required for independent module execution.

5 Evaluation

In this section, we present a comprehensive evaluation of DFLOP to demonstrate its effectiveness in optimizing MLLM training. We begin by detailing our experimental setup, outlining the hardware testbed, the baseline systems used for comparison, and the models and datasets selected for our experiments (§5.1). The evaluation then proceeds with a series of macro-experiments that measure the end-to-end performance of DFLOP against these baselines across a variety of MLLM configurations (§5.2). To provide a deeper understanding of the underlying sources of the observed performance gains, we subsequently conduct a series of micro-experiments (§5.3). Our key findings are summarized as follows:

- DFLOP achieves up to 3.6× higher training throughput compared to PyTorch and Megatron-LM across various MLLM architectures and model scales, demonstrating its effectiveness in optimizing distributed training efficiency.
- The performance gap between DFLOP and baseline systems widens as the number of nodes increases, indicating that DFLOP scales more effectively in larger cluster environments.

- Fine-grained micro-level analysis shows that DFLOP reduces pipeline idle time by up to 84% while simultaneously increasing stage-wise throughput and maintaining balanced performance across all pipeline stages, leading to significantly improved GPU utilization. This demonstrates that our approach effectively resolves the dual challenges of load imbalance and performance variability outlined in §2.3.

5.1 Experimental Setup

The experiments evaluate the end-to-end training efficiency of DFLOP against state-of-the-art baselines. To ensure a comprehensive assessment, the evaluation employs representative MLLM architectures and diverse datasets that mirror the workload heterogeneity found in real-world scenarios.

Baselines. We compare our system against two strong baselines: Megatron-LM [44] and a custom PyTorch-based implementation. Megatron-LM is widely regarded as a state-of-the-art framework for large-scale model training, providing a highly optimized implementation of 3D parallelism. As our system is built upon PyTorch [47], we also developed a baseline that leverages PyTorch's native distributed functionalities. For both baselines, the 3D parallelism configurations were manually tuned following conventional best practices to achieve their best possible performance [44].

Hardware. Our experiments were conducted on a cluster of up to 8 nodes, each equipped with an NVIDIA HGX A100 8-GPU system. The GPUs within each node are interconnected via high-bandwidth NVLink, and the nodes are connected with an 800 Gbps InfiniBand network.

Datasets. To create a realistic MLLM training scenario characterized by workload variability, we constructed a composite dataset from several public sources. This dataset mirrors the heterogeneity commonly found in MLLM pre-training and instruction tuning by including a mix of data types: single images, multiple images, and videos. The specific composition of the dataset is detailed in Table 2.

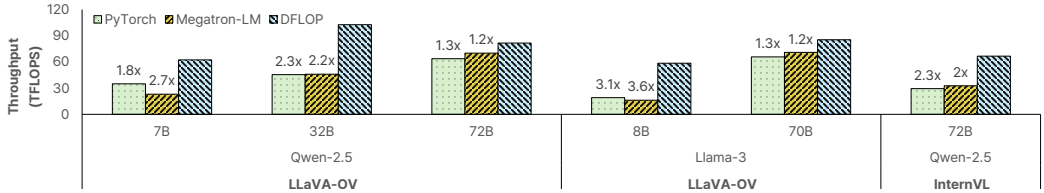
Models. We selected two state-of-the-art, publicly available MLLMs for our evaluation: LLaVA-OneVision (LLaVA-OV) [33] and InternVL-2.5 [8]. To demonstrate the general applicability and robustness of our system across various architectures and scales, we configured these models with different encoders and LLM backbones. Specifically, our experiments utilized powerful modality encoders including SigLIP [67] and InternViT [8]. These were paired with a range of LLM backbones. The specific model configurations used in our experiments are detailed in Table 3.

5.2 Macro Experiments

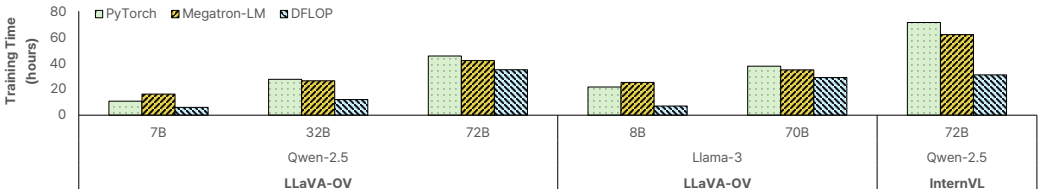
This section presents the end-to-end performance of DFLOP compared against baseline systems. The experiments are designed to demonstrate the overall training efficiency gains achieved by the integrated optimization approach across a range of representative MLLM architectures and scales.

5.2.1 End-to-end Performance. In this section, we present the end-to-end performance of DFLOP, comparing its training throughput and total training time against the baseline systems. DFLOP consistently demonstrates substantial performance improvements. As shown in Figure 7a, GPU throughput gains range from 1.2× to 3.6×, and Figure 7b illustrates that this translates to a reduction in total training time by 5 to 40 hours across various MLLM configurations.

The throughput results highlight the robustness of our approach. For the LLaVA-OV architecture, DFLOP maintains its high performance advantage over the baselines even when the internal LLM backbone is varied in both architecture and scale. Furthermore, the significant gains observed with the InternVL architecture confirm that our system's effectiveness is not limited to a single MLLM design but generalizes across different architectural paradigms. In terms of practical impact, we applied the official training recipes for LLaVA-OV and InternVL to measure the total training time.



(a) End-to-end training throughput.



(b) End-to-end training time.

Fig. 7. End-to-end training performance of DFLOP over baseline systems.

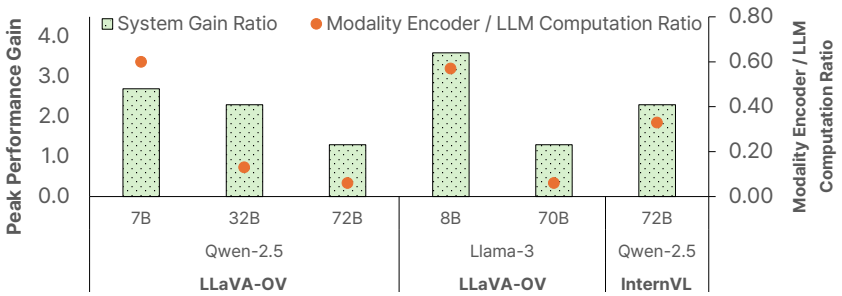


Fig. 8. Correlation between the computational load ratio (modality encoder FLOP / LLM FLOP) and the maximum performance gain achieved by DFLOP when compared to baseline systems, shown for various MLLM configurations.

The results confirm that the observed throughput gains translate directly into significant real-world time savings.

This performance advantage stems from the fundamental difference in optimization methodology. Unlike baseline systems that enforce a uniform, data-agnostic strategy, DFLOP adopts a holistic approach that aligns system resources with the specific computational demands of each module. By leveraging comprehensive profiling, the system derives heterogeneous parallel configurations that minimize the expected makespan based on dataset statistics. This static optimization is bridged with runtime execution through dynamic scheduling, which actively mitigates pipeline bubbles caused by input variability.

5.2.2 Impact of Computational Asymmetry. This section evaluates the impact of the computational ratio between the modality encoder and the LLM on the performance gain of DFLOP. Figure 8 demonstrates that the performance advantage of DFLOP amplifies as the computational loads between the two modules become more balanced. This trend stems from the structural limitations of baseline systems, which enforce homogeneous parallel configurations across the entire model. As the computational distribution becomes symmetric, this monolithic constraint precludes independent optimization of heterogeneous modules, leading to significant resource underutilization. In contrast, DFLOP maximizes end-to-end throughput by decoupling parallel strategies for each module and

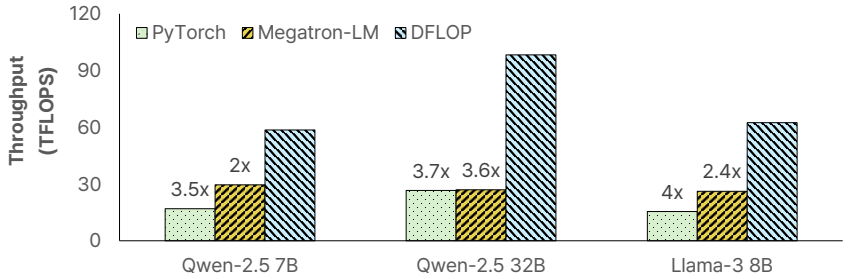


Fig. 9. Performance gain demonstrating the cross-modal generalization of DFLOP, measured on a 4-node cluster with 8×A100 GPUs per node. Using the Qwen2-Audio architecture as a representative case.

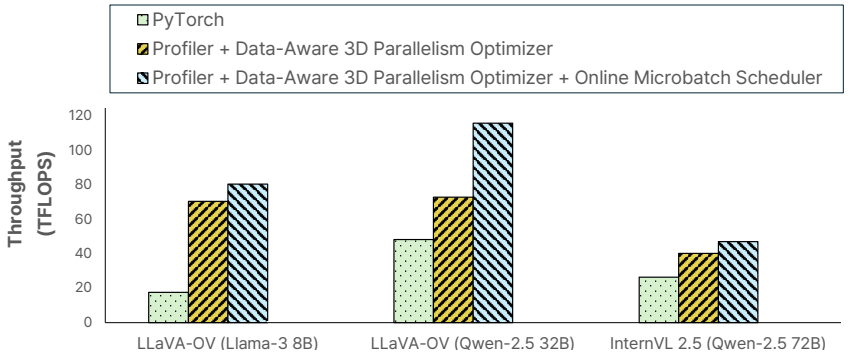


Fig. 10. Performance gain on the mixed dataset using a 4-node cluster with 8×A100 GPUs per node, obtained by incrementally adding DFLOP components to the PyTorch baseline across various MLLM configurations.

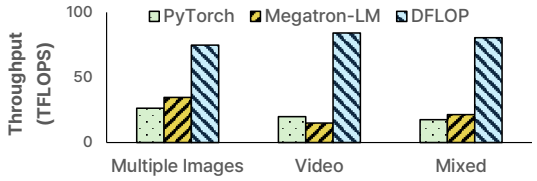
dynamically mitigating workload skew, thereby widening the performance gap against baselines in balanced workloads.

This capability is particularly significant given the current trajectory of MLLM development toward balanced cross-modal computation. This shift is driven by the emergence of larger modality encoders [3, 8] and more efficient LLM backbones [41, 61], alongside strategies reducing the number of tokens processed by the LLM [3, 8]. Consequently, the utility of DFLOP is expected to grow as future MLLM architectures converge toward equiproportional computation.

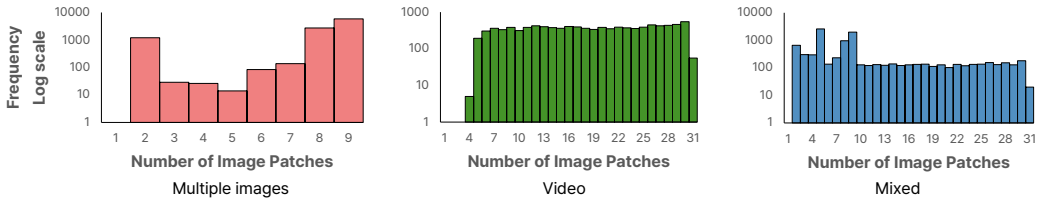
5.3 Micro Experiments

To provide a deeper analysis of the end-to-end performance gains and their underlying sources, we conduct a series of targeted experiments, each designed to examine a specific aspect of DFLOP's design and behavior.

5.3.1 Generalization Across Modalities. To evaluate the extensibility of DFLOP beyond vision-language models, we conduct experiments on the Qwen2-Audio [10] architecture, a representative audio-language MLLM. As illustrated in Figure 9, DFLOP achieves significant throughput gains ranging from 2× to 4× for the audio modality as well. These substantial improvements are attributed to the balanced computational intensity between the audio modality encoder and the LLM. Specifically, while the audio modality encoder involves high computational demands, it utilizes an average pooling layer at the end of its pipeline to reduce the number of tokens fed into the LLM.



(a) Performance on diverse datasets.



(b) Input frequency distribution by the number of image patches.

Fig. 11. System performance of LLaVA-OV (Llama-3 8B) on a 4-node cluster with 8xA100 GPUs per node and input characteristics across diverse datasets. The mixed dataset integrates single-image, multiple-image, and video inputs, revealing the degree of data heterogeneity in each dataset.

This architectural feature decreases the LLM workload, resulting in a comparable distribution of compute requirements between the encoder and the LLM.

5.3.2 Ablation Study. The individual contributions of the Data-aware 3D Parallelism Optimizer and the Online Microbatch Scheduler are evaluated across diverse MLLM configurations, as shown in Figure 10.

The primary source of performance gain shifts depending on the model configuration. For LLaVA-OV (Llama-3 8B), the Data-aware 3D Parallelism Optimizer delivers the majority of the improvement, whereas for LLaVA-OV (Qwen-2.5 32B), the Online Microbatch Scheduler becomes the dominant contributor. InternVL 2.5 (Qwen-2.5 72B) occupies a middle ground, with both components contributing comparably to the total gain. This variation correlates with the degree of computational asymmetry. In configurations with low computational asymmetry, such as LLaVA-OV (Llama-3 8B), the pipeline is highly sensitive to the static partitioning strategy, making the optimizer critical. Conversely, in highly asymmetric cases like LLaVA-OV (Qwen-2.5 32B), the dominant module dictates the pipeline bottleneck, diminishing the impact of static partitioning inefficiencies. Consequently, the optimization burden shifts to the scheduler to mitigate runtime bubbles caused by input variability.

5.3.3 Performance on Diverse Datasets. This experiment evaluates the robustness of DFLOP across three distinct workload scenarios: multiple-image, video, and mixed datasets. As illustrated in Figure 11a, the throughput of PyTorch and Megatron-LM degrades as workload heterogeneity increases. While the limitations of these frameworks are less pronounced on the multiple-image dataset, their throughput declines on the heterogeneous video and mixed workloads. In contrast, DFLOP maintains consistent high performance regardless of the dataset composition.

This performance differential is directly correlated with the input shape distributions presented in Figure 11b. The multiple-image workload exhibits a narrow distribution concentrated within a limited range, resembling a static workload. In such homogeneous scenarios, the structural rigidity of static partitioning results in limited performance degradation compared to dynamic cases. However, video and mixed datasets introduce broad, uniform distributions with high per-iteration

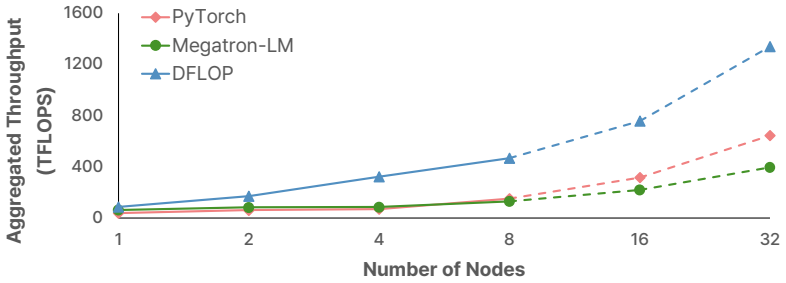


Fig. 12. Total cluster throughput variation of the LLaVA-OV (Llama-3 8B) on the mixed dataset, with increasing GPU node scale and 8×A100 GPUs per node. Measured results (1–8 GPU nodes) are shown with solid lines, and projected performance (16–32 GPU nodes) with dashed lines.

variance. These dynamic conditions expose the inflexibility of data-agnostic frameworks, which lack the mechanisms to adapt to fluctuating computational demands, leading to resource underutilization. Conversely, DFLOP mitigates these variations through data-aware optimization and online scheduling, demonstrating its inherent resilience to the workload heterogeneity characteristic of modern MLLM training.

5.3.4 GPU Cluster Scalability. This experiment evaluates the scalability of our system and projects its performance as the number of GPU nodes increases. The results, depicted in Figure 12, illustrate the aggregate GPU throughput across different system scales. For this test, we used the LLaVA-OV model configured with a Llama-3 8B LLM. We measured the actual performance on clusters ranging from 1 to 8 nodes and then used these measurements to project the expected throughput for 16 and 32-node configurations.

The results clearly indicate that the performance gap between DFLOP and the baseline systems not only persists but also widens as the cluster size grows. This superior scalability can be attributed to two primary factors, corresponding to the core components of our system. First, as more GPUs become available, the search space for the Data-aware 3D Parallelism Optimizer expands exponentially. This allows the optimizer to discover more fine-grained and effective parallelism strategies that are inaccessible at smaller scales, leading to greater performance improvements. Second, scaling up the number of GPUs often involves increasing the data parallelism degree, which in turn increases the number of data-parallel groups that must synchronize. In the baseline systems, the lack of a data-aware scheduling mechanism leads to significant overhead from straggler effects during these synchronization steps. In contrast, the Online Microbatch Scheduler in DFLOP effectively mitigates this issue by balancing the workload across all data-parallel ranks, preventing stragglers and minimizing idle time. This effect becomes more pronounced at larger scales, further amplifying the performance advantage of our system.

5.3.5 Analysis of Pipeline Bubble. As illustrated in Figure 13, DFLOP reduces the empirically measured idle time by 82% and 84% compared to PyTorch and Megatron-LM, respectively. This result is particularly insightful because DFLOP's theoretical minimum idle time, derived from the $(p-1)/m$ formula [44], is slightly higher than that of the baselines. This apparent contradiction highlights a fundamental limitation in relying on idealized metrics for optimizing dynamic, real-world systems.

The theoretical minimum idle time is predicated on a perfectly homogeneous workload, a condition fundamentally violated in MLLM training, as illustrated in Figure 1. The baseline systems,

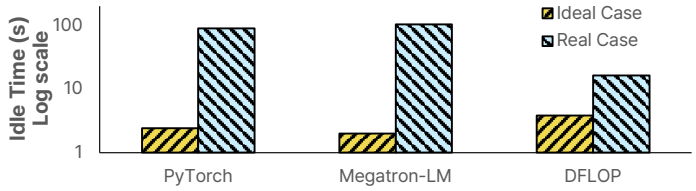


Fig. 13. GPU idle time due to pipeline bubbles on the mixed dataset for LLaVA-OV (Llama-3 8B) on a 4-node cluster with 8×A100 GPUs per node. The Ideal case represents the theoretical idle time from pipeline bubbles inherent to the 1F1B schedule for the given 3D parallel configuration, while the Real case represents the empirically measured idle time.

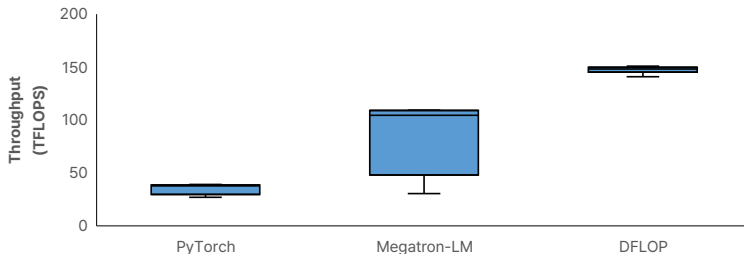


Fig. 14. Boxplots illustrating the distribution of throughput across pipeline stages on the mixed dataset for each evaluated system using LLaVA-OV (Llama-3 8B) on a 4-node cluster with 8×A100 GPUs per node.

implicitly optimized for this flawed ideal, are brittle to workload variations. This structural mismatch causes their empirically measured idle time to deviate substantially from their theoretical minimum.

In contrast, DFLOP is explicitly designed for such heterogeneity. Its optimizer deliberately selects a smaller number of microbatches, a choice that is suboptimal from a purely theoretical standpoint but is strategically advantageous in a dynamic environment. This approach prioritizes real-world throughput and provides the Online Microbatch Scheduler with larger, more effective batches to balance. Consequently, DFLOP's measured performance remains in close alignment with its theoretical potential. This demonstrates that the higher efficiency of DFLOP is achieved not by optimizing for an unrealistic theoretical formula, but by robustly managing the real-world dynamics of the workload.

5.3.6 Stage-wise Throughput. To evaluate how effectively each system utilizes the computational capacity of the GPUs assigned to different modules, this experiment analyzes the distribution of throughput for each pipeline stage. We calculate the stage-wise throughput by dividing the total computational load of a stage by its pure computation time, which is defined as the total iteration time minus any idle time attributable to pipeline bubbles. The resulting distributions, visualized in Figure 14, show that the average stage throughput in DFLOP is substantially higher than in the baseline systems, while the variance of throughput across its stages is significantly lower. This indicates that DFLOP not only achieves a higher level of performance for each stage but also maintains a consistent and balanced throughput across the entire pipeline. This outcome is a direct result of our optimizer, which explicitly selects a 3D parallelism strategy to both maximize the expected throughput for each stage and balance performance across the architecturally distinct modality encoder and LLM modules. In contrast, the baseline systems employ statically configured parallelism that does not account for input-dependent throughput variations, resulting in both

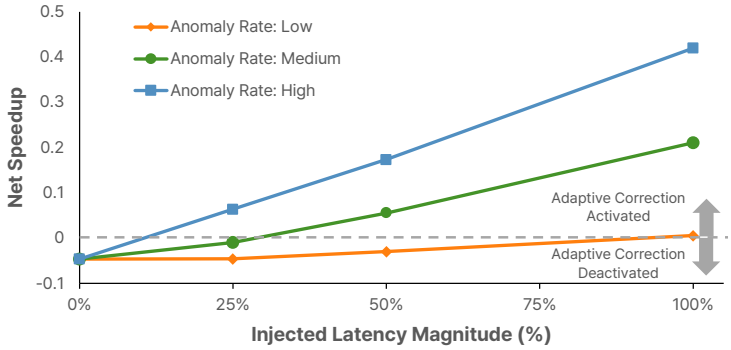


Fig. 15. Cost-benefit analysis of Adaptive Correction on LLaVA-OV (Llama-3 8B) using a 4-node cluster with 8xA100 GPUs per node. Net speedup denotes the correction gain minus profiling overhead, evaluated across varying anomaly rates (low, medium, high) representing the outlier frequency and injected latencies relative to the maximum stage duration. The mechanism deactivates when the net speedup is negative to prevent performance degradation.

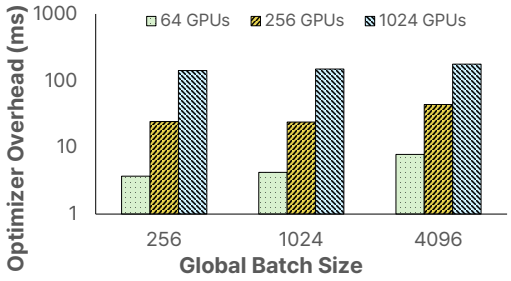
Table 4. Total training time and DFLOP overhead across different MLLMs on the mixed dataset using an 8-node cluster with 8×A100 GPUs per node. The relative overhead is the ratio of system overhead to the total training time.

Models	Total Training Time (hours)	DFLOP Overhead (min)	Relative Overhead (%)
LLaVA-OV (Qwen-2.5 7B)	6.12	7.58	2.1
LLaVA-OV (Llama-3 8B)	6.82	7.55	1.8
LLaVA-OV (Qwen-2.5 32B)	11.62	9.09	1.3
LLaVA-OV (Llama-3 70B)	28.94	10.04	0.5
LLaVA-OV (Qwen-2.5 72B)	30.62	9.87	0.5
InternVL (Qwen-2.5 72B)	34.55	9.88	0.5

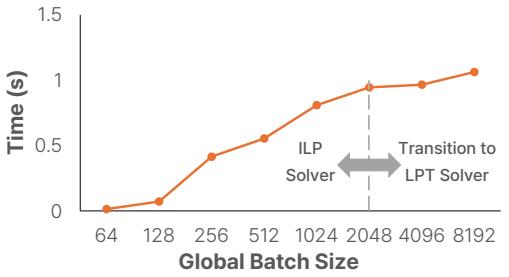
a lower average throughput for each stage and a significant performance imbalance across the pipeline.

5.3.7 Cost-Benefit of Adaptive Correction. This experiment characterizes the activation and deactivation behavior of the Adaptive Correction mechanism across various workload scenarios. The objective is to identify the specific conditions—defined by the frequency of edge cases and the magnitude of their deviation—under which the system toggles the monitoring mechanism. The evaluation employs a cost-benefit analysis with synthetic delays injected into a subset of input shapes, where workload anomalies are categorized by rate—low (1%), medium (3%), and high (5%)—and injected latencies are quantified as a percentage of the observed maximum stage duration.

The evaluation metric, net speedup, is defined as the correction gain minus the profiling overhead (~4%). As illustrated in Figure 15, the net speedup scales positively with both the anomaly rate and latency magnitude. The results clearly show the operational boundaries of the mechanism; for instance, in scenarios where the benefit is insufficient—such as a low anomaly rate with latency magnitudes $\leq 50\%$ or a medium rate with $\leq 25\%$—the mechanism remains deactivated. This confirms that the system successfully manages the monitoring toggle, ensuring that Adaptive Correction is engaged only when the predicted performance gain justifies the overhead.



(a) Overhead of the Data-aware 3D Parallelism Optimizer across varying global batch sizes and numbers of GPUs.



(b) Overhead of the Online Microbatch Scheduler across varying global batch sizes.

Fig. 16. Overhead analysis of DFLOP components. The Data-aware 3D Parallelism Optimizer maintains negligible overhead (< 200 ms) even at large scales (1024 GPUs). The Online Microbatch Scheduler overhead is fully overlapped with computation via asynchronous prefetching, given that training iterations span tens to hundreds of seconds.

5.3.8 DFLOP Overhead. The computational overhead of DFLOP is evaluated through two distinct analyses. The first subsection quantifies the one-time initialization cost, encompassing both the offline profiling phase and the strategy generation by the Data-aware 3D Parallelism Optimizer. The subsequent subsection investigates the execution overhead of the Data-aware 3D Parallelism Optimizer and the Online Microbatch Scheduler with respect to resource scaling, assessing their efficiency under large-scale configurations.

One-time Profiling Overhead. The aggregate initialization overhead of DFLOP ranges from 7 to 10 minutes across configurations, primarily driven by the Profiling Engine, as detailed in Table 4. Experimental results indicate individual execution times of 6 to 10 minutes for throughput profiling, 3 to 9 minutes for memory profiling, and 1.45 to 1.62 minutes for the Data Profiler. Since these components execute concurrently, the effective profiling overhead is determined by the maximum duration among them, corresponding to the throughput profiling phase. Finally, the Data-aware 3D Parallelism Optimizer generates the strategy in approximately 3.7 ms. Consequently, the total DFLOP overhead constitutes at most 2.1% of the end-to-end training duration, demonstrating that significant performance gains are achieved with minimal amortized cost.

Overhead Sensitivity to Scale. The impact of resource scaling on the execution latency of the Data-aware 3D Parallelism Optimizer and Online Microbatch Scheduler is investigated under large-scale configurations. As illustrated in Figure 16a, the execution time for the Data-aware 3D Parallelism Optimizer remains largely invariant to the number of GPUs and global batch size, staying within hundreds of milliseconds. Given that subsequent training runs typically span hours to days, the amortized cost of this preceding offline process is negligible.

As depicted in Figure 16b, the execution time for the Online Microbatch Scheduler exhibits a dependency on global batch size. At a global batch size of 2048, the ILP solver reaches its preset time limit and triggers a fallback to the LPT solver. Despite this transition, the resulting load imbalance deviates by less than 1% from the theoretical lower bound. Crucially, the asynchronous design ensures that this scheduling latency does not impact the critical path. Since the scheduler operates on the CPU to prefetch data for the subsequent step while a single training iteration consumes tens to hundreds of seconds, the scheduling latency of approximately one second is fully overlapped with model execution.

6 Related Work

Prior research has explored automated and hybrid parallelization strategies, yet they often fall short in the context of MLLMs. Several approaches focus on general matrix-based algorithms or data preprocessing [4, 23, 38, 46, 49], rather than the specific pipeline constraints of deep learning. Similarly, automating parallelism based on static computational graphs [42, 73] assuming uniform batches lacks the runtime adaptivity required for MLLMs, where input-dependent throughput causes severe imbalance.

Widely adopted distributed learning systems such as DeepSpeed [54], PyTorch [47], and Megatron-LM [44] provide robust 3D parallelism implementations (e.g., ZeRO, kernel fusion) but remain fundamentally data-agnostic. These frameworks apply a monolithic strategy that fails to account for the diverse data characteristics and heterogeneous module architectures inherent to MLLM training, often leading to suboptimal performance in multimodal settings. Recent research has explored systems specifically for large-scale multimodal models. DistMM [27] determines 3D parallelism based on the computational ratio between modality and language models, but its approach is limited to parallel architectures like CLIP [52], unlike the sequential modality-encoder-to-LLM structure common in modern MLLMs. Other works like Optimus [19] dynamically schedule encoders into idle stages, and DistTrain [71] uses disaggregated execution with input reordering. Although these studies address MLLM parallelism, they generally do not comprehensively tackle the intertwined challenges of non-uniform computation time and input-dependent throughput variability arising from MLLM heterogeneity. Furthermore, these referenced systems are closed-source, limiting their extensibility and impact on broader research development. In contrast, DFLOP addresses these dual challenges through its integrated, data-driven optimization approach, and its open-source nature facilitates future research and development in the community.

7 Conclusion

In this paper, we present DFLOP, a data-driven framework that optimizes the MLLM training pipeline. DFLOP addresses the fundamental inefficiency of data-blind distributed training by explicitly modeling the interaction between input data characteristics and parallel execution behavior. Through its three integrated components–the Profiling Engine, Data-aware 3D Parallelism Optimizer, and Online Microbatch Scheduler–DFLOP jointly optimizes parallelism configuration and runtime scheduling to mitigate stage imbalance and input-dependent variability. Our extensive experiments show that DFLOP achieves up to 3.6× faster training throughput compared to state-of-the-art frameworks such as PyTorch and Megatron-LM, while significantly improving GPU utilization. In summary, DFLOP bridges the gap between data diversity and distributed system efficiency, setting a new direction for data-driven optimization in large-scale multimodal model training.

Acknowledgments

This research was supported by NRF grants (No. RS-2025-16068623, No. RS-2024-NR121334), the Korea Basic Science Institute (National Research Facilities and Equipment Center) grant funded by the Ministry of Science and ICT (No. RS-2024-00403860), Advanced Database System Infrastructure (NFEC-2024-11-300458), and the Yonsei University Research Fund (2025-22-0057). We also thank SK Telecom for their dedicated support in providing cloud infrastructure with A100 GPU clusters, which enabled the large-scale experiments in this work.

References

[1] Xiang An, Yin Xie, Kaicheng Yang, Wenkang Zhang, Xiuwei Zhao, Zheng Cheng, Yirui Wang, Songcen Xu, Changrui Chen, Chunsheng Wu, Huajie Tan, Chunyuan Li, Jing Yang, Jie Yu, Xiyao Wang, Bin Qin, Yumeng Wang, Zizhen Yan, Ziyong Feng, Ziwei Liu, Bo Li, and Jiankang Deng. 2025. LLaVA-OneVision-1.5: Fully Open Framework for Democratized Multimodal Training. arXiv:2509.23661 [cs.CV] https://arxiv.org/abs/2509.23661

[2] Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, Binyuan Hui, Luo Ji, Mei Li, Junyang Lin, Runji Lin, Dayiheng Liu, Gao Liu, Chengqiang Lu, Keming Lu, Jianxin Ma, Rui Men, Xingzhang Ren, Xuancheng Ren, Chuanqi Tan, Sinan Tan, Jianhong Tu, Peng Wang, Shijie Wang, Wei Wang, Shengguang Wu, Benfeng Xu, Jin Xu, An Yang, Hao Yang, Jian Yang, Shusheng Yang, Yang Yao, Bowen Yu, Hongyi Yuan, Zheng Yuan, Jianwei Zhang, Xingxuan Zhang, Yichang Zhang, Zhenru Zhang, Chang Zhou, Jingren Zhou, Xiaohuan Zhou, and Tianhang Zhu. 2023. Qwen Technical Report. arXiv:2309.16609 [cs.CL] https://arxiv.org/abs/2309.16609

[3] Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, et al. 2025. Qwen2.5-vl technical report. *arXiv preprint arXiv:2502.13923* (2025).

[4] Matthias Boehm, Shirish Tatikonda, Berthold Reinwald, Prithviraj Sen, Yuanyuan Tian, Douglas R. Burdick, and Shivakumar Vaithyanathan. 2014. Hybrid parallelization strategies for large-scale machine learning in SystemML. *Proc. VLDB Endow.* 7, 7 (March 2014), 553–564. doi:10.14778/2732286.2732292

[5] Andy Brock, Soham De, Samuel L Smith, and Karen Simonyan. 2021. High-Performance Large-Scale Image Recognition Without Normalization. In *Proceedings of the 38th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 139)*, Marina Meila and Tong Zhang (Eds.). PMLR, 1059–1071. https://proceedings.mlr.press/v139/brock21a.html

[6] Feilong Chen, Minglun Han, Haozhi Zhao, Qingyang Zhang, Jing Shi, Shuang Xu, and Bo Xu. 2023. X-LLM: Bootstrapping Advanced Large Language Models by Treating Multi-Modalities as Foreign Languages. arXiv:2305.04160 [cs.CL] https://arxiv.org/abs/2305.04160

[7] Sanyuan Chen, Yu Wu, Chengyi Wang, Shujie Liu, Daniel Tompkins, Zhuo Chen, Wanxiang Che, Xiangzhan Yu, and Furu Wei. 2023. BEATs: audio pre-training with acoustic tokenizers. In *Proceedings of the 40th International Conference on Machine Learning* (Honolulu, Hawaii, USA) *(ICML'23)*. JMLR.org, Article 203, 16 pages.

[8] Zhe Chen, Weiyun Wang, Yue Cao, Yangzhou Liu, Zhangwei Gao, Erfei Cui, Jinguo Zhu, Shenglong Ye, Hao Tian, Zhaoyang Liu, et al. 2024. Expanding performance boundaries of open-source multimodal models with model, data, and test-time scaling. *arXiv preprint arXiv:2412.05271* (2024).

[9] Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing. 2023. Vicuna: An Open-Source Chatbot Impressing GPT-4 with 90%* ChatGPT Quality. https://lmsys.org/blog/2023-03-30-vicuna/

[10] Yunfei Chu, Jin Xu, Qian Yang, Haojie Wei, Xipin Wei, Zhifang Guo, Yichong Leng, Yuanjun Lv, Jinzheng He, Junyang Lin, et al. 2024. Qwen2-audio technical report. *arXiv preprint arXiv:2407.10759* (2024).

[11] Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Alex Castro-Ros, Marie Pellat, Kevin Robinson, Dasha Valter, Sharan Narang, Gaurav Mishra, Adams Yu, Vincent Zhao, Yanping Huang, Andrew Dai, Hongkun Yu, Slav Petrov, Ed H. Chi, Jeff Dean, Jacob Devlin, Adam Roberts, Denny Zhou, Quoc V. Le, and Jason Wei. 2024. Scaling Instruction-Finetuned Language Models. *Journal of Machine Learning Research* 25, 70 (2024), 1–53. http://jmlr.org/papers/v25/23-0870.html

[12] Can Cui, Yunsheng Ma, Xu Cao, Wenqian Ye, Yang Zhou, Kaizhao Liang, Jintai Chen, Juanwu Lu, Zichong Yang, Kuei-Da Liao, et al. 2024. A survey on multimodal large language models for autonomous driving. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*. 958–979.

[13] Tri Dao. 2024. FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning. In *International Conference on Learning Representations (ICLR)*.

[14] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[15] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2021. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. arXiv:2010.11929 [cs.CV] https://arxiv.org/abs/2010.11929

[16] Zane Durante, Qiuyuan Huang, Naoki Wake, Ran Gong, Jae Sung Park, Bidipta Sarkar, Rohan Taori, Yusuke Noda, Demetri Terzopoulos, Yejin Choi, et al. 2024. Agent ai: Surveying the horizons of multimodal interaction. *arXiv preprint arXiv:2401.03568* (2024).

[17] Shiqing Fan, Yi Rong, Chen Meng, Zongyan Cao, Siyu Wang, Zhen Zheng, Chuan Wu, Guoping Long, Jun Yang, Lixue Xia, Lansong Diao, Xiaoyong Liu, and Wei Lin. 2021. DAPPLE: a pipelined data parallel approach for training large models. In *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming* (Virtual Event, Republic of Korea) *(PPoPP '21)*. Association for Computing Machinery, New York, NY, USA, 431–445.

doi:10.1145/3437801.3441593

[18] Yuxin Fang, Wen Wang, Binhui Xie, Quan Sun, Ledell Wu, Xinggang Wang, Tiejun Huang, Xinlong Wang, and Yue Cao. 2023. EVA: Exploring the Limits of Masked Visual Representation Learning at Scale. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 19358–19369.

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

[20] R. L. Graham. 1969. Bounds on Multiprocessing Timing Anomalies. *SIAM J. Appl. Math.* 17, 2 (1969), 416–429. arXiv:https://doi.org/10.1137/0117039 doi:10.1137/0117039

[21] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783* (2024).

[22] Gurobi Optimization, LLC. 2026. Gurobi Optimizer Reference Manual. https://www.gurobi.com

[23] Dong He, Supun C Nakandala, Dalitso Banda, Rathijit Sen, Karla Saur, Kwanghyun Park, Carlo Curino, Jesús Camacho-Rodríguez, Konstantinos Karanasos, and Matteo Interlandi. 2022. Query processing on tensor computation runtimes. *Proc. VLDB Endow.* 15, 11 (July 2022), 2811–2825. doi:10.14778/3551793.3551833

[24] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. 2022. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556* (2022).

[25] Ke Hong, Guohao Dai, Jiaming Xu, Qiuli Mao, Xiuhong Li, Jun Liu, Kangdi Chen, Yuhan Dong, and Yu Wang. 2024. FlashDecoding++: Faster Large Language Model Inference with Asynchronization, Flat GEMM Optimization, and Heuristics. In *Proceedings of Machine Learning and Systems*, P. Gibbons, G. Pekhimenko, and C. De Sa (Eds.), Vol. 6. 148–161. https://proceedings.mlsys.org/paper_files/paper/2024/file/5321b1dabcd2be188d796c21b733e8c7-Paper-Conference.pdf

[26] Wei-Ning Hsu, Benjamin Bolte, Yao-Hung Hubert Tsai, Kushal Lakhotia, Ruslan Salakhutdinov, and Abdelrahman Mohamed. 2021. HuBERT: Self-Supervised Speech Representation Learning by Masked Prediction of Hidden Units. *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 29 (2021), 3451–3460. doi:10.1109/TASLP.2021.3122291

[27] Jun Huang, Zhen Zhang, Shuai Zheng, Feng Qin, and Yida Wang. 2024. DISTMM: Accelerating Distributed Multimodal Model Training. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. USENIX Association, Santa Clara, CA, 1157–1171. https://www.usenix.org/conference/nsdi24/presentation/huang

[28] 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. 2019. *GPipe: efficient training of giant neural networks using pipeline parallelism.* Curran Associates Inc., Red Hook, NY, USA.

[29] Yiren Jian, Chongyang Gao, and Soroush Vosoughi. 2023. Bootstrapping Vision-Language Learning with Decoupled Language Pre-training. In *Advances in Neural Information Processing Systems*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (Eds.), Vol. 36. Curran Associates, Inc., 57–72. https://proceedings.neurips.cc/paper_files/paper/2023/file/002262941c9edfd472a79298b2ac5e17-Paper-Conference.pdf

[30] Dongfu Jiang, Xuan He, Huaye Zeng, Cong Wei, Max Ku, Qian Liu, and Wenhu Chen. 2024. Mantis: Interleaved multi-image instruction tuning. *arXiv preprint arXiv:2405.01483* (2024).

[31] Yizhang Jin, Jian Li, Yexin Liu, Tianjun Gu, Kai Wu, Zhengkai Jiang, Muyang He, Bo Zhao, Xin Tan, Zhenye Gan, et al. 2024. Efficient multimodal large language models: A survey. *arXiv preprint arXiv:2405.10739* (2024).

[32] Aniruddha Kembhavi, Mike Salvato, Eric Kolve, Minjoon Seo, Hannaneh Hajishirzi, and Ali Farhadi. 2016. A diagram is worth a dozen images. In *European conference on computer vision*. Springer, 235–251.

[33] Bo Li, Yuanhan Zhang, Dong Guo, Renrui Zhang, Feng Li, Hao Zhang, Kaichen Zhang, Peiyuan Zhang, Yanwei Li, Ziwei Liu, et al. 2024. Llava-onevision: Easy visual task transfer. *arXiv preprint arXiv:2408.03326* (2024).

[34] Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. 2023. BLIP-2: bootstrapping language-image pre-training with frozen image encoders and large language models. In *Proceedings of the 40th International Conference on Machine Learning* (Honolulu, Hawaii, USA) *(ICML'23)*. JMLR.org, Article 814, 13 pages.

[35] Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, and Soumith Chintala. 2020. PyTorch distributed: experiences on accelerating data parallel training. *Proc. VLDB Endow.* 13, 12 (Aug. 2020), 3005–3018. doi:10.14778/3415478.3415530

[36] Haotian Liu, Chunyuan Li, Yuheng Li, and Yong Jae Lee. 2024. Improved Baselines with Visual Instruction Tuning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 26296–26306.

[37] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. 2023. Visual Instruction Tuning. In *Advances in Neural Information Processing Systems*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (Eds.), Vol. 36. Curran Associates, Inc., 34892–34916. https://proceedings.neurips.cc/paper_files/paper/2023/file/

6dcf277ea32ce3288914faf369fe6de0-Paper-Conference.pdf

[38] Rui Liu, Kwanghyun Park, Fotis Psallidas, Xiaoyong Zhu, Jinghui Mo, Rathijit Sen, Matteo Interlandi, Konstantinos Karanasos, Yuanyuan Tian, and Jesús Camacho-Rodríguez. 2023. Optimizing Data Pipelines for Machine Learning in Feature Stores. *Proceedings of the VLDB Endowment* 16, 13 (Sept. 2023), 4230–4239. doi:10.14778/3625054.3625060

[39] Junyu Lu, Dixiang Zhang, Songxin Zhang, Zejian Xie, Zhuoyang Song, Cong Lin, Jiaxing Zhang, Bingyi Jing, and Pingjian Zhang. 2024. Lyrics: Boosting Fine-grained Language-Vision Alignment and Comprehension via Semantic-aware Visual Objects. arXiv:2312.05278 [cs.CL] https://arxiv.org/abs/2312.05278

[40] Minesh Mathew, Viraj Bagal, Rubèn Tito, Dimosthenis Karatzas, Ernest Valveny, and CV Jawahar. 2022. Infographicvqa. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*. 1697–1706.

[41] Meta AI. 2025. The Llama 4 Herd: The Beginning of a New Era of Natively Multimodal AI Innovation. https://ai.meta.com/blog/llama-4-multimodal-intelligence/. Accessed: 2025-12-04.

[42] Xupeng Miao, Yujie Wang, Youhe Jiang, Chunan Shi, Xiaonan Nie, Hailin Zhang, and Bin Cui. 2022. Galvatron: Efficient Transformer Training over Multiple GPUs Using Automatic Parallelism. *Proceedings of the VLDB Endowment* 16, 3 (Nov. 2022), 470–479. doi:10.14778/3570690.3570697 arXiv:2211.13878 [cs].

[43] Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R. Devanur, Gregory R. Ganger, Phillip B. Gibbons, and Matei Zaharia. 2019. PipeDream: generalized pipeline parallelism for DNN training. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*. ACM, Huntsville Ontario Canada, 1–15. doi:10.1145/3341301.3359646

[44] D. Narayanan, M. Shoeybi, J. Casper, P. LeGresley, M. Patwary, V. Korthikanti, D. Vainbrand, P. Kashinkunti, J. Bernauer, B. Catanzaro, A. Phanishayee, and M. Zaharia. 2021. Efficient Large-Scale Language Model Training on GPU Clusters Using Megatron-LM. In *SC21: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE Computer Society, Los Alamitos, CA, USA, 1–14. https://doi.ieeecomputersociety.org/

[45] NVIDIA. 2024. *Sequence Packing*. https://docs.nvidia.com/nemo-framework/user-guide/24.09/nemotoolkit/features/optimizations/sequence_packing.html

[46] Kwanghyun Park, Karla Saur, Dalitso Banda, Rathijit Sen, Matteo Interlandi, and Konstantinos Karanasos. 2022. End-to-end Optimization of Machine Learning Prediction Queries. In *Proceedings of the 2022 International Conference on Management of Data* (Philadelphia, PA, USA) *(SIGMOD '22)*. Association for Computing Machinery, New York, NY, USA, 587–601. doi:10.1145/3514221.3526141

[47] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. *PyTorch: an imperative style, high-performance deep learning library*. Curran Associates Inc., Red Hook, NY, USA.

[48] Laurent Perron and Vincent Furnon. 2025. *OR-Tools*. Google. https://developers.google.com/optimization/

[49] Arnab Phani, Lukas Erlbacher, and Matthias Boehm. 2022. UPLIFT: parallelization strategies for feature transformations in machine learning workloads. *Proceedings of the VLDB Endowment* 15, 11 (2022), 2929–2938.

[50] PyTorch. 2024. PyTorch Performance Tuning Guide. https://docs.pytorch.org/tutorials/recipes/recipes/tuning_guide.html. Accessed: 2026-01-29.

[51] PyTorch. 2024. *PyTorch Scaled Dot Product Attention*. https://pytorch.org/docs/stable/generated/torch.nn.functional.scaled_dot_product_attention

[52] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. 2021. Learning transferable visual models from natural language supervision. In *International conference on machine learning*. PMLR, 8748–8763.

[53] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. 2020. ZeRO: memory optimizations toward training trillion parameter models. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (Atlanta, Georgia) *(SC '20)*. IEEE Press, Article 20, 16 pages.

[54] Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. 2020. DeepSpeed: System Optimizations Enable Training Deep Learning Models with Over 100 Billion Parameters. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery Data Mining (KDD '20)*. Association for Computing Machinery, New York, NY, USA, 3505–3506. doi:10.1145/3394486.3406703

[55] Zineng Tang, Ziyi Yang, Mahmoud Khademi, Yang Liu, Chenguang Zhu, and Mohit Bansal. 2024. CoDi-2: In-Context Interleaved and Interactive Any-to-Any Generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 27425–27434.

[56] Terence Tao. 2009. *Poincare's Legacies, Part I: Pages from Year Two of a Mathematical Blog*. American Mathematical Soc.

[57] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. LLaMA: Open and Efficient Foundation Language Models. arXiv:2302.13971 [cs.CL] https://arxiv.org/abs/2302.13971

[58] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open Foundation and Fine-Tuned Chat Models. arXiv:2307.09288 [cs.CL] https://arxiv.org/abs/2307.09288

[59] Wenhui Wang, Hangbo Bao, Li Dong, Johan Bjorck, Zhiliang Peng, Qiang Liu, Kriti Aggarwal, Owais Khan Mohammed, Saksham Singhal, Subhojit Som, and Furu Wei. 2023. Image as a Foreign Language: BEIT Pretraining for Vision and Vision-Language Tasks . In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, Los Alamitos, CA, USA, 19175–19186. doi:10.1109/CVPR52729.2023.01838

[60] Jiayang Wu, Wensheng Gan, Zefeng Chen, Shicheng Wan, and S Yu Philip. 2023. Multimodal large language models: A survey. In *2023 IEEE International Conference on Big Data (BigData)*. IEEE, 2247–2256.

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

[62] Peng Xu, Xiatian Zhu, and David A Clifton. 2023. Multimodal learning with transformers: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 45, 10 (2023), 12113–12132.

[63] Le Xue, Mingfei Gao, Chen Xing, Roberto Martin-Martin, Jiajun Wu, Caiming Xiong, Ran Xu, Juan Carlos Niebles, and Silvio Savarese. 2023. ULIP: Learning a Unified Representation of Language, Images, and Point Clouds for 3D Understanding . In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, Los Alamitos, CA, USA, 1179–1189. doi:10.1109/CVPR52729.2023.00120

[64] Ling Yang, Zhaochen Yu, Chenlin Meng, Minkai Xu, Stefano Ermon, and Bin CUI. 2024. Mastering Text-to-Image Diffusion: Recaptioning, Planning, and Generating with Multimodal LLMs. In *Forty-first International Conference on Machine Learning*. https://openreview.net/forum?id=DgLFkAPwuZ

[65] Qinghao Ye, Haiyang Xu, Guohai Xu, Jiabo Ye, Ming Yan, Yiyang Zhou, Junyang Wang, Anwen Hu, Pengcheng Shi, Yaya Shi, Chenliang Li, Yuanhong Xu, Hehong Chen, Junfeng Tian, Qi Qian, Ji Zhang, Fei Huang, and Jingren Zhou. 2024. mPLUG-Owl: Modularization Empowers Large Language Models with Multimodality. arXiv:2304.14178 [cs.CL] https://arxiv.org/abs/2304.14178

[66] Nur Yildirim, Hannah Richardson, Maria Teodora Wetscherek, Junaid Bajwa, Joseph Jacob, Mark Ames Pinnock, Stephen Harris, Daniel Coelho De Castro, Shruthi Bannur, Stephanie Hyland, et al. 2024. Multimodal healthcare AI: identifying and designing clinically relevant vision-language applications for radiology. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*. 1–22.

[67] Xiaohua Zhai, Basil Mustafa, Alexander Kolesnikov, and Lucas Beyer. 2023. Sigmoid loss for language image pre-training. In *Proceedings of the IEEE/CVF international conference on computer vision*. 11975–11986.

[68] Hang Zhang, Xin Li, and Lidong Bing. 2023. Video-LLaMA: An Instruction-tuned Audio-Visual Language Model for Video Understanding. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, Yansong Feng and Els Lefever (Eds.). Association for Computational Linguistics, Singapore, 543–553. doi:10.18653/v1/2023.emnlp-demo.49

[69] Renrui Zhang, Ziyu Guo, Wei Zhang, Kunchang Li, Xupeng Miao, Bin Cui, Yu Qiao, Peng Gao, and Hongsheng Li. 2022. PointCLIP: Point Cloud Understanding by CLIP. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 8542–8552. doi:10.1109/CVPR52688.2022.00836

[70] Yuanhan Zhang, Bo Li, Haotian Liu, Yong Jae Lee, Liangke Gui, Di Fu, Jiashi Feng, Ziwei Liu, and Chunyuan Li. 2024. LLaVA-NeXT: A Strong Zero-shot Video Understanding Model. https://llava-vl.github.io/blog/2024-04-30-llava-next-video/

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

[72] Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Less Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, Alban Desmaison, Can Balioglu, Pritam Damania, Bernard Nguyen, Geeta Chauhan, Yuchen Hao, Ajit Mathews, and Shen Li. 2023. PyTorch FSDP: Experiences on Scaling Fully Sharded Data Parallel. 16, 12 (2023). doi:10.14778/3611540.3611569

[73] Lianmin Zheng, Zhuohan Li, Hao Zhang, Yonghao Zhuang, Zhifeng Chen, Yanping Huang, Yida Wang, Yuanzhong Xu, Danyang Zhuo, Eric P. Xing, Joseph E. Gonzalez, and Ion Stoica. 2022. Alpa: Automating Inter- and Intra-Operator Parallelism for Distributed Deep Learning. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. USENIX Association, Carlsbad, CA, 559–578. https://www.usenix.org/conference/osdi22/presentation/zheng-lianmin

Received October 2025; revised January 2026; accepted February 2026