Efficient Multi-Task Large Model Training via Data Heterogeneity-aware Model Management

Yujie Wang[1], Shenhan Zhu[1], Fangcheng Fu[1], Xupeng Miao[2], Jie Zhang[3], Juan Zhu[3], Fan Hong[3], Yong Li[3], Bin Cui[1]

[1]Peking University [2]Purdue University [3]Alibaba Group

[1]{alfredwang, shenhan.zhu, ccchengff, bin.cui}@pku.edu.cn [2]xupeng@purdue.edu

[3]{wanglin.zj, zhujuan.zj, hongfan.hf, jiufeng.ly}@alibaba-inc.com

ABSTRACT

Recent foundation models are capable of handling multiple machine learning (ML) tasks and multiple data modalities with the unified base model structure and several specialized model components. However, the development of such multi-task (MT) multi-modal (MM) models poses significant model management challenges to existing training systems. Due to the sophisticated model architecture and the heterogeneous workloads of different ML tasks and data modalities, training these models usually requires massive GPU resources and suffers from sub-optimal system efficiency.

In this paper, we investigate how to achieve high-performance training of large-scale MT MM models through data heterogeneity-aware model management optimization. The key idea is to decompose the model execution into stages and address the joint optimization problem sequentially, including both heterogeneity-aware workload parallelization and dependency-driven execution scheduling. Based on this, we build a prototype system and evaluate it on various large MT MM models. Experiments demonstrate the superior performance and efficiency of our system, with speedup ratio up to 71% compared to state-of-the-art training systems.

PVLDB Reference Format:
Yujie Wang, Shenhan Zhu, Fangcheng Fu, Xupeng Miao, Jie Zhang, Juan Zhu, Fan Hong, Yong Li, Bin Cui. Efficient Multi-Task Large Model Training via Data Heterogeneity-aware Model Management. PVLDB, 18(1): XXX-XXX, 2025.
doi:XX.XX/XXX.XX

1 INTRODUCTION

Machine learning (ML) has become an essential tool for understanding and generating knowledge from data and tackling complex tasks for humans. In the past few years, our data management community has put great efforts in developing systems to support the whole ML lifecycle [15, 39, 73, 84], such as data preparation [16, 38, 116], model development [55, 57, 97], model selection [63, 64] and model deployment [46, 103]. Recently, as the rapid rise of large-scale foundation models [2, 14, 22, 77–79, 87, 88], developing these large models is becoming increasingly challenging due to the substantial GPU resource requirements. For example, how to design training systems for those models consisting of billion of parameters over more than dozens of GPUs demands extensive expertise, which has attracted lots of research interests from our community [44, 56, 58, 69, 115, 117]. As a result, ML models themselves have become another form of data, and their management techniques are becoming increasingly important [54, 71].

Considering the multi-modal nature of real-world data, ML researchers have shifted their focus to developing model beyond the the language domain (e.g., ChatGPT [70]) to many other data modalities (e.g., images [11, 23, 49, 75], speech [7, 76, 92], video [5, 86]). The recent extension further involves composite scenarios [3, 4, 9, 10, 52, 62, 82, 95], where models are capable of processing and interpreting data across several tasks simultaneously.

However, existing large model training systems are mainly designed for a single model with only one input data modality. Despite the extensive research and engineering efforts aimed at optimizing these systems from multiple perspectives, including distributed communication [68, 81, 93], memory management [17, 80, 83], and GPU computation [20, 21], their performance is still limited when it comes to handling the increasingly complex requirements of multi-task (MT) multi-modal (MM) models. We identify two unique obstacles when building training systems for MT MM models.

One is the workload heterogeneity due to the divergent data flows across modalities or tasks. On the one hand, MM models often handle data that vary significantly in structure and size, demanding specialized preprocessing and computational approaches. For example, language models (e.g., GPT-family [2, 14, 77, 78], LLaMA-family [87, 88]) are usually equipped with dozens of layers with the same configuration (e.g., hidden size), while vision models may involve uneven layers to compute in various resolutions [49]. On the other hand, as depicted in Fig. 1, multiple tasks usually leverage distinct data flows and activate individual model components, leading to inter-task workload heterogeneity. Due to such heterogeneous modality data flows and sub-models, different modalities and tasks exhibit distinct execution overhead (detailed in Fig. 4, §3.2). Existing training systems usually overlook such workload heterogeneity and apply sub-optimal training methodologies.

模型结构不同

子模型类型不同

Another is the data flow execution dependency among different model components. Recent MT MM model development usually adopts a sub-model sharing approach [9, 10, 26, 62, 95], where partial model layers containing common knowledge are shared across different modalities and tasks. As shown in Fig. 1, each data type also has its own learning component. Within every training iteration, the input data mixed with multiple modalities are simultaneously fed into the sophisticated model, where different model components are intricately activated and updated. To avoid redundant resource usage, the shared components are usually responsible for the data

Proceedings of the VLDB Endowment, Vol. 18, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX

flows from multiple sources, resulting in execution barriers and blocking the following model layers. In addition, the proportion of different data modalities in MT workloads may shift over time due to task addition and completion, introducing further training complexity. To the best of our knowledge, none of existing training systems can deal with these unforeseen dependency efficiently due to the lack of understanding MT MM model execution.

To tackle these obstacles, this paper introduces Spindle, a resource-efficient and high-performance training system for large-scale MT MM models via ***data heterogeneity-aware model management***. Considering the workload heterogeneity and execution dependency, a naïve solution is to *decouple* the model structure based on modality and task, replicate the shared components, and deploy them on separate devices. In this way, each sub-model can be optimized by existing systems, but it also brings significant *resource wastage and underutilization*, as well as additional overheads from replica synchronization. As an example, Fig. 1 showcases that such a naïve, decoupled execution suffers from fluctuating device utilization both intra-task and inter-task due to workload heterogeneity, leading to low or even idle GPU utilization for some time slots. Instead of decoupling, Spindle manages to directly train the whole complex model *without* disjoint sub-model to minimize the resource usage. A key insight behind Spindle's design is that *heterogeneous* and *dependent* sub-models can be decomposed into several *sequentially executed and independent* stages, each of which contains multiple parallel model modules with *similar* execution overheads.

To achieve resource-efficient and high-performance training of MT MM models, there are three key ***model management*** challenges for Spindle to address. In the following, we will introduce each challenge and how Spindle solves them.

C1: Model Parallelization. First, finding the optimal model parallel configuration for heterogeneous workloads with diverse computational characteristics is a complex combinatorial problem. Existing single-model automatic parallelization approaches (e.g., Alpa [118], Unity [89], Galvatron [58, 99]) assume a spatial pipeline stage partition, and each operator (Op) is executed by all devices of the corresponding pipeline stage. Unfortunately, such assumptions only work for homogeneous models, failing to adapt to heterogeneous MT MM models.

Instead of solving the parallel configuration directly, Spindle captures the workload heterogeneity at the operator granularity and estimates its execution overheads under different amount of allocated resources and parallel configurations (§3.2). The final configuration decision is left to the later step since it requires to be jointly optimized with considering the execution dependency. Spindle also introduces MetaOp to contract the graph (i.e., fusing continuous identical operators) to avoid redundant estimation overheads and shrink the problem scale (§3.1).

C2: Model Division. Second, breaking down the whole model into sequentially executed stages is straightforward, but it may easily result in inefficiencies. Determining the optimal division of stages is complicated since the operators differ significantly in their execution overheads and have intricate operator dependencies.

Spindle addresses this problem with two steps: 1) Spindle's *resource allocator* (§3.3) traverses the computation graph following the dependency topology and decides the optimal resource allocation for MetaOps in each candidate set (i.e., currently executable MetaOps). Here we reformulate this issue as a malleable project scheduling problem (MPSP) and subsequently derive the optimal solution. 2) After obtaining the parallel configuration of each MetaOp, the *stage scheduler* (§3.4) greedily slices and selects MetaOps to craft compact stages and minimizes the overall execution time.

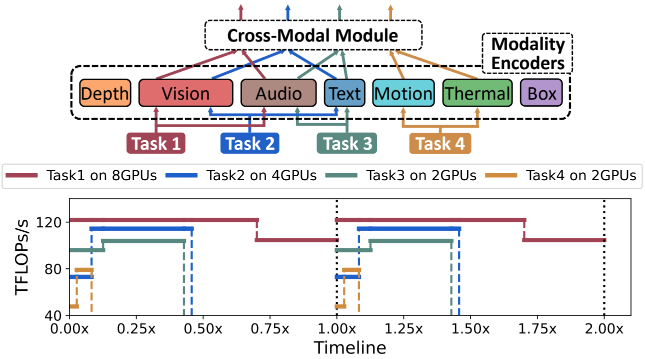


Figure 1: The upper portion illustrates the general model structure and training flow of MT MM training. The lower portion displays the current device utilization, measured in FLOPs per second, during the decoupled execution of four tasks across 2 iterations. Utilization fluctuation of different-colored and same-colored lines indicate the workload heterogeneity among tasks and within a single task among operators, respectively.

C3: Model Mapping. Third, given the stage-based resource allocation and execution schedule plan, how to map them into physical devices is still a problem, since different mapping may lead to distinct inter-stage communication overheads and per-device memory consumption. To further improve the overall system efficiency, Spindle carefully considers these trade-offs and the real environment constraints (e.g., inter-device bandwidth, memory capacity) when generating the device placement plan (§3.5).

Our contributions are summarized as follows:

- We present Spindle, a high-performance and resource-efficient training system for large-scale MT MM models via data heterogeneity-aware model management optimization.
- We propose a jointly optimization framework to achieve heterogeneity-aware workload parallelization and dependency-driven execution scheduling.
- We build a general runtime engine to perform the stage-based schedule, automatically resolving execution dependencies at the stage boundaries.
- We evaluate Spindle on various large MT MM models, and the results demonstrate the superior performance and efficiency of Spindle compared with the state-of-the-art baselines, with the speedup ratio up to 71%.

2 PRELIMINARY

2.1 Multi-Task Multi-Modal Models

Multi-Modal Application of Foundation Models. The advent of foundation models [2, 14, 22, 77–79, 87, 88] has revolutionized deep learning (DL). Beginning with the birth of BERT [22] and GPT [77] based on Transformer [90] structure, followers such as the GPT series [2, 14, 78], T5 [79], OPT [111], and the LLaMA series [87, 88] have set new benchmarks across a range of language tasks. Foundation models have also been successfully adapted for tasks of other data modalities, including image processing [11, 23, 49], audio processing [7, 76, 92], video analysis [5, 86]. Multi-modal models [9, 10, 19, 26, 75, 95] leverage these foundation models to integrate information from multiple data modalities. They can be primarily categorized into two types. The first category fuses modality information via contrastive learning objectives [26, 29, 35, 61, 75, 107, 109], with CLIP [75] being a notable example, and ImageBind [26] further extending CLIP to six modalities. These models typically have a multi-tower structure, where each modality has its own encoder. They take paired modality data (e.g., image-text pairs for CLIP), extract features via modality encoders, and perform cross-modal alignment using contrastive objectives. The second category merges modality features through a language model's generative loss [10, 12, 37, 41, 42, 51, 94, 95, 98, 100, 108]. These models usually consist of multi-tower modality encoders and a cross-modal module. Modality encoders extract features from each modality, which are then fed into the cross-modal module for feature fusion. Recently, with the success of open-sourced large language models (LLMs) [18, 85, 87, 88, 111], researchers have started to enhance multi-modal models with powerful pretrained LLMs [3, 9, 19, 24, 40, 47, 48, 62, 119]. These multi-modal LLMs, also falling into the second category.

Multi-Task Multi-Modal Models. Recently, researchers have begun to construct multi-task multi-modal (MT MM) models [3, 4, 9, 10, 62], enabling the support for diverse multi-modal tasks within a unified model. This is because each modality encompasses various tasks, and each task often involves multiple modalities as well. The general model structure and the training flow is illustrated in the upper side of Fig. 1. MT MM models reflects researchers' aspiration towards a general-purpose AI. Flamingo [3] is among the first to handle multiple vision-language tasks. OFASys [10] proposes a general MT MM learning paradigm, as shown in Fig. 1, designing distinct modality encoders and cross-modal modules for different tasks and modalities, allowing the activation of different components as required by the task and modality at hand. For example, speech recognition and image captioning tasks shall activate and share the text encoder but feed the visual- and audio-inputs into different encoders. Many empirical results have also shown that such a joint multi-task training paradigm achieves better multi-modal capabilities for MT MM models than performing single-task training separately [3, 4, 9, 10, 52, 82, 95].

2.2 Parallelisms in Distributed Training

As model sizes and training data volumes grow, modern DL systems commonly utilize clusters of multiple GPUs for distributed training, thereby enhancing efficiency. Various parallelisms are employed to manage model parameters or training data in a distributed manner. Data parallelism (DP) [45, 80, 117] splits the input data, with each device handling a portion of the data storage and computation, and synchronizing model gradients across devices. Model parallelism [33, 65, 66, 68] partitions model parameters, with each device responsible for storing and computing a segment of the model. Model parallelisms can be categorized into two popular types: tensor parallelism (TP) partitions the model vertically [68], while pipeline parallelism (PP) [33, 65, 66] splits the model horizontally, organizing model computations into a pipeline. Contemporary distributed training systems, such as Megatron-LM [68] and DeepSpeed [81], leverage multiple parallelisms and implement a hybrid parallelism approach for model training. For example, Megatron-LM introduces 3D parallelism, which concurrently utilizes DP, TP, and PP, enhancing training efficiency. Researchers have also developed advanced automatic parallelism techniques facilitate the tuning of optimal parallelism combinations. These automatic parallelism [36, 58, 99, 118] approaches integrate multiple parallelism dimensions, employ sophisticated optimization workflows, and automatically determine the most efficient hybrid parallelism strategy, significantly reducing the reliance on human effort. However, these existing training system are mainly deigned for single task and single model training, with limited performance on the complex scenario of training MT MM models.

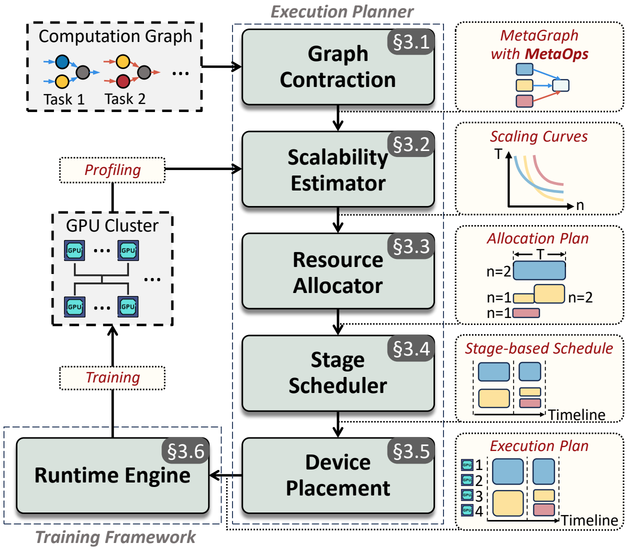
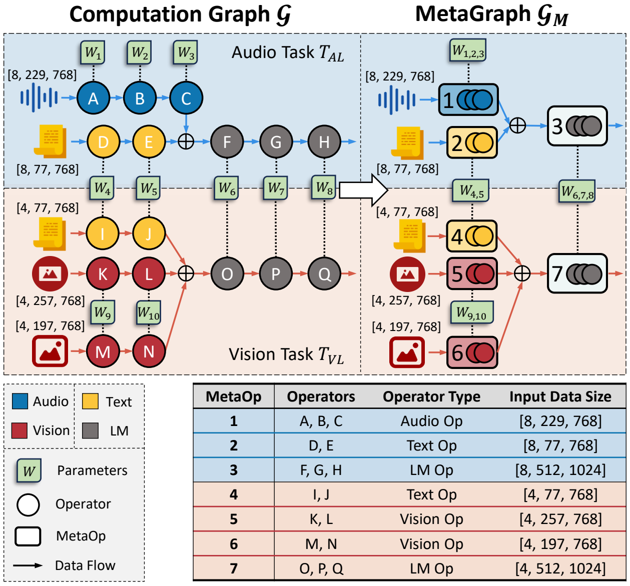


Figure 2: Architecture overview of Spindle.

3 SYSTEM DESIGN

Spindle is a highly efficient and scalable training framework designed for MT MM models. Fig. 2 depicts its system architecture, comprising the execution planner and the training framework. Given the diverse user-defined training tasks and the GPU cluster, the goal of Spindle is to devise the most efficient execution plan to facilitate effective MT MM training.

Problem Formulation. We formalize the optimization problem of Spindle as follows. Firstly, Spindle interprets the input tasks as a unified directed acyclic computation graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where each node $i \in \mathcal{V}$ represents a computational operator and each edge $\langle i, j \rangle \in \mathcal{E}$ denotes the data flow from operator i to j. Each task activates specific operators and parameters with unique data flows. For instance, a vision-related task activates a vision Transformer layer as an operator, with image features serving as the data flow. The left side of Fig. 3 displays an example of a computation graph. Then, given the computation graph $\mathcal{G}$ and the GPU cluster with N devices, Spindle aims to minimize the maximal operator completion time C. Specifically, we need to find an execution plan P, which assigns each operator $i \in \mathcal{V}$ with an **AS**-tuple $\langle n_i, s_i \rangle \in \mathcal{U}$, such



MetaOp	Operators	Operator Type	Input Data Size
1	A, B, C	Audio Op	[8, 229, 768]
2	D, E	Text Op	[8, 77, 768]
3	F, G, H	LM Op	[8, 512, 1024]
4	I, J	Text Op	[4, 77, 768]
5	K, L	Vision Op	[4, 257, 768]
6	M, N	Vision Op	[4, 197, 768]
7	O, P, Q	LM Op	[4, 512, 1024]

Figure 3: An illustration of the computation graph $\mathcal{G}$ and the contracted MetaGraph $\mathcal{G}_M$.

that the operator i is **A**llocated n_i devices and is **S**cheduled to execute from time s_i. Here the set $\mathcal{U} = \{\langle n, s\rangle | n \in \mathbb{N}, s \geq 0\}$ is formed by all valid AS-tuples. We further denote the execution time of operator i when allocated n_i devices as $t_i = T_i(n_i)$. Then, the optimization problem is formulated as follows:

$$\underset{\substack{P=\{i\to\langle n_i,s_i\rangle|\\ i\in\mathcal{V},\langle n_i,s_i\rangle\in\mathcal{U}\}}}{\arg\min} \quad C := \max_{i\in\mathcal{V}}\{s_i + t_i\} \tag{1}$$

$$\text{s.t.} \quad \sum_{t\in(s_i,s_i+t_i),i\in\mathcal{V}} n_i \leq N \quad \text{for } \forall t \in \mathbb{R}^+ \tag{2}$$

$$s_i + t_i \leq s_j \quad \text{for } \forall\langle i, j\rangle \in \mathcal{E} \tag{3}$$

Here (2) is the allocation capacity constraint for any time t, and (3) is the operator dependency constraint.

Sketch of Solution. Before stepping into the solution of Spindle, we would like to first present an overview for better readability. First, Spindle initiates a graph contraction process (§3.1), contracting the original graph $\mathcal{G}$ into a MetaGraph $\mathcal{G}_M$ composed of *MetaOps* (Fig. 3), where each MetaOp characterizes a unique workload. This process further decouples MetaOps into different *MetaLevels*, ensuring that there are no dependencies among MetaOps within the same MetaLevel. Second, the scalability estimator (§3.2) estimates the execution time and resource scalability for each MetaOp, producing scaling curves (Fig. 4). Following this, the resource allocator (§3.3) deduces the allocation plan for each MetaLevel individually (Fig. 5a). Given the allocation plan, the stage scheduler (§3.4) slices the MetaOps and organizes them into *Stages*, and produces the Stage-based schedule for execution. Subsequently, device placement (§3.5) strategies are then employed to assign MetaOps to appropriate devices, resulting in the Spindle execution plan (Fig. 5b). Finally, the runtime engine (§3.6) utilizes this plan to instantiate the model on each device and facilitate an efficient MT MM training process.

3.1 Graph Contraction

Depicting Workload Heterogeneity with MetaOps. Spindle is designed to minimize the execution time by optimizing resource allocation and scheduling for each operator within $\mathcal{G}$. This optimization process necessitates an understanding of the workload characteristics for each operator $i \in \mathcal{V}$, which can be reflected by its execution time function $t_i = T_i(n_i)$, which varies with the device allocation amount n_i. Given that $\mathcal{G}$ typically includes a large number of operators while many of them share similar workload characteristics (such as stacked Transformer layers), Spindle initiates a graph contraction process to streamline the complicated graph. It categorizes operators based on their computational workload characteristics, as illustrated in Fig. 3. In this process, operators are contracted into a MetaOp if they meet the following criteria:

(1) There is a data flow between operator i and j, i.e., $\langle i, j\rangle \in \mathcal{E}$, and both the out-degree of operator i and the in-degree of operator j are 1, ensuring that they are direct predecessors and successors to each other.
(2) Operator i and j share the same computational operator type, parameter size, and input data size, confirming identical computational workloads.

During the graph contraction procedure, we traverse the original graph $\mathcal{G}$ in topological order, contracting operators based on the specified criteria until no further pairs of operators meeting these conditions remain. This results in a contracted MetaGraph $\mathcal{G}_M = (\mathcal{V}_M, \mathcal{E}_M)$, with each node $m \in \mathcal{V}_M$ representing a MetaOp that consists of L_m consecutive operators in $\mathcal{G}$. Since operators in the same MetaOp share the same workload, we slightly abuse the notation and denote the execution time function for each operator in MetaOp m as $T_m(n)$.

Disentangling MetaOp Dependency with MetaLevels. To facilitate operator-level resource allocation and scheduling, we further introduce an abstraction called MetaLevel, which signifies the level of dependency. MetaOps at the same level are independent to each other. The level of each MetaOp can be derived by a Breadth-First-Search (BFS), with the level assigned based on the search depth, which inherently ensures no dependency among the MetaOps of same level. By doing so, the problem (1) can be dissected into several simplified sub-problems for different MetaLevels. Next, we introduce how Spindle derives the allocation and scheduling for each MetaLevel individually, and merges them into the final plan.

3.2 Scalability Estimator

As MetaOps differ in operator types and/or input data sizes, they characterize heterogeneous workloads and thus necessitate different amount of resources. Furthermore, there's no doubt that these MetaOps have distinct resource scalability (i.e., how its execution time varies w.r.t. the amount of allocated resources). For instance, the left side of Fig. 4 shows the execution time of different MetaOps, $T_m(n)$, in Multitask-CLIP (an multi-task extension of CLIP [26, 75], refer to §5.1 for details). Some MetaOps show almost linear decreases in execution time as resources increase (e.g., Task2-Vision), while others decrease much more slowly (e.g., Task1-Text). The right side of Fig. 4 further shows the value of $\varsigma_m(n) = T_m(1)/T_m(n)$, which measures how much the operator accelerates when using more GPUs, and a value of $\varsigma_m(n)$ closer to n signifies better resource

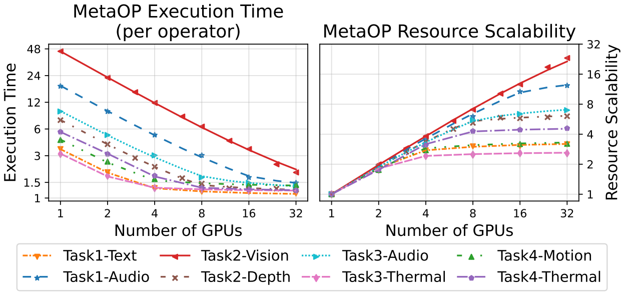


Figure 4: An example of the execution time and resource scalability of MetaOps in 4-task Multitask-CLIP, denoted as *scaling curves*.

scalability. As can be seen, different MetaOps not only have varying execution time, but also exhibit different resource scalability, posing a significant challenge for resource allocation.

In response to this issue, Spindle employs a scalability estimator to accurately capture the execution time and the resource scalability of each MetaOp. Previous works [58, 89, 118] have designed effective estimation methods for distributed training, commonly utilizing the α-β modelling [31]. However, although this may work well for homogeneous workloads (e.g., large language models with homogeneous layers), we find that it does not fit the workload heterogeneous nature of MT MM models. This is because different MetaOps have distinct workload and resource scalability, and the invoked kernels may vary across different per-device workloads, therefore causing distinct performance. In a nutshell, our scalability estimator adopts the *piecewise* α-β modelling for more accurate estimation of heterogeneous MT MM workloads. Given the target MT MM model, it profiles several discrete data points $(n_i, T_m(n_i))$ for each MetaOp under different parallel configurations, and then fits the curve of piecewise α-β function. To estimate the execution time $T_m(n)$, it locates the range that n falls into, and returns the estimated time according to the corresponding piecewise function. In practice, the profiling and estimating process for each MT MM model takes within 5 minutes, which is negligible compared to the massive training time. In Fig. 4, the scatter points represent empirical measurements, while the curves depict the function estimated by our scalability estimator, which we denote as *scaling curves*. As can be seen, our scalability estimator effectively and accurately estimates the execution time $T_m(n)$ for each MetaOp. More details are illustrated in Appendix § A.

3.3 Resource Allocator

In this subsection, we introduce the resource allocator of Spindle, which allocates appropriate computational resources to each MetaOp. We begin by transitioning the problem (1) into the subproblem on one MetaLevel. We then detail our allocation strategies, which first relax constraints and optimize the continuous problem, and then discretize the optimal solution to obtain practical allocation plans.

Problem Formulation on MetaLevel. We first re-formulate the problem (1) on one MetaLevel with a set of MetaOps denoted by $\widetilde{\mathcal{V}}_M$. In this formulation, we split each MetaOp into different execution part, by assigning it with several ASL-tuples $\langle n, s, l\rangle \in \mathcal{U}_M$, such that l consecutive operators of this MetaOp are scheduled to execute from time s with n devices. Here $\mathcal{U}_M = \{\langle n, s, l\rangle | n, l \in \mathbb{N}, s \geq 0\}$ is formed by all valid ASL-tuples. For each MetaOp $m \in \widetilde{\mathcal{V}}_M$, its execution plan is a set of ASL-tuples P_m. For a MetaLevel, the execution plan P consists of P_m for all MetaOps $m \in \widetilde{\mathcal{V}}_M$, i.e., $P = \{m \to P_m\}$. Given $m \in \widetilde{\mathcal{V}}_M$ and one ASL-tuple $p = \langle n_m^{(p)}, s_m^{(p)}, l_m^{(p)}\rangle \in P_m$, we denote the execution time span, end time, and time interval by $t_m^{(p)} = T_m(n_m^{(p)}) \cdot l_m^{(p)}$, $e_m^{(p)} = s_m^{(p)} + t_m^{(p)}$, and $I_m^{(p)} = (s_m^{(p)}, e_m^{(p)})$, respectively. The problem can be re-written as:

$$\mathop{\arg\min}_{P=\{m\to P_m | m\in\widetilde{\mathcal{V}}_M, P_m \subset 2^{\mathcal{U}_M}\}} \quad \widetilde{C} := \max_{m\in\widetilde{\mathcal{V}}_M, p\in P_m} \{e_m^{(p)}\} \tag{4}$$

$$\text{s.t.} \quad \sum_{t\in I_m^{(p)}, m\in\widetilde{\mathcal{V}}_M, p\in P_m} n_m^{(p)} \leq N \quad \text{for } \forall t \in \mathbb{R}^+ \tag{5}$$

$$I_m^{(p_1)} \cap I_m^{(p_2)} = \varnothing \quad \text{for } \forall m \in \widetilde{\mathcal{V}}_M, p_1, p_2 \in P_m \tag{6}$$

$$\sum_{p\in P_m} l_m^{(p)} = L_m \quad \text{for } \forall m \in \widetilde{\mathcal{V}}_M \tag{7}$$

Compared with the original problem (1), the sub-problem (4) on MetaLevel gets rid of the dependency constraint, while the constraint (6) enforces the execution intervals of ASL-tuples in P_m to be pairwise disjoint, because operators within the same MetaOp cannot execute simultaneously, and (7) ensures all operators are executed for each MetaOp.

Optimum of the Continuous Problem. If we relax the constraints, allowing GPU resources and operators to be continuously divisible (i.e., n and l in ASL-tuples are not limited to integers), the problem is transformed into a well-established problem, malleable project scheduling problem (MPSP), with malleable projects and continuously divisible resources [25]. We denote the optimal solution of this relaxed problem by P_{MPSP}. Prior works [101, 102] have given the following theorem.

Theorem 1. *If the execution time functions $T_m(n)$, $n \in \mathbb{R}^+$, are positive and non-increasing for every MetaOp $m \in \widetilde{\mathcal{V}}_M$, then $P_{MPSP} = \{m \to P_m\}$ satisfies that $P_m = \{\langle n_m^*, 0, L_m\rangle\}, \forall m \in \widetilde{\mathcal{V}}_M$, where the optimum objective $\widetilde{C}^*$ and allocations n_m^* can be found from*

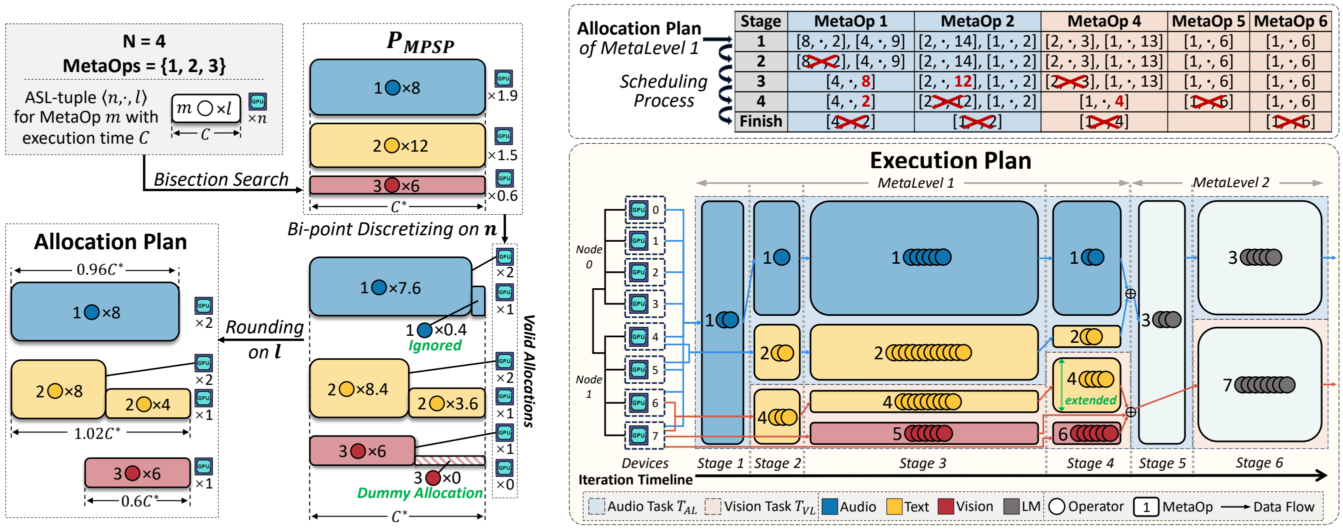
$$T_m(n_m^*) \cdot L_m = \widetilde{C}^* \textit{ for } \forall m \in \widetilde{\mathcal{V}}_M \textit{ and } \sum\nolimits_{m\in\widetilde{\mathcal{V}}_M} n_m^* = N. \tag{8}$$

From Theorem 1, it follows that in the optimal situation, all MetaOps start simultaneously, execute all their operators, and finish together. They share an identical end time $e_m = \widetilde{C}^*$, which is exactly the minimized operator completion time.

To achieve P_{MPSP}, our allocator utilizes the scaling curves from §3.2 to acquire an estimation of $T_m(n)$, and performs a bisection search procedure over $\widetilde{C}^*$ with the following equation. The details are illustrated in Appendix §B.

$$\sum\nolimits_{m\in\widetilde{\mathcal{V}}_M} T_m^{-1}\left(\widetilde{C}^*/L_m\right) = N. \tag{9}$$

Bi-point Discretized Allocation. From the continuous problem, we've determined the optimal time $\widetilde{C}^*$, as well as the optimal allocations for each MetaOp, n_m^*, which is a real number. To reinstate n's as integers, our allocator computes each MetaOp's proper discrete allocations individually. For every MetaOp m, it uses two discrete



Stage	MetaOp 1	MetaOp 2	MetaOp 4	MetaOp 5	MetaOp 6
1	[8, ·, 2], [4, ·, 9]	[2, ·, 14], [1, ·, 2]	[2, ·, 3], [1, ·, 13]	[1, ·, 6]	[1, ·, 6]
2	~~[8, ·, 2]~~, [4, ·, 9]	[2, ·, 14], [1, ·, 2]	[2, ·, 3], [1, ·, 13]	[1, ·, 6]	[1, ·, 6]
3	[4, ·, 8]	[2, ·, 12], [1, ·, 2]	~~[2, ·, 3]~~, [1, ·, 13]	[1, ·, 6]	[1, ·, 6]
4	[4, ·, 2]	~~[2, ·, 12]~~, [1, ·, 2]	[1, ·, 4]	~~[1, ·, 6]~~	[1, ·, 6]
Finish	~~[4, ·, 2]~~	~~[1, ·, 2]~~	~~[1, ·, 4]~~		~~[1, ·, 6]~~

(a) Illustration of workflow of Spindle allocator.

(b) Illustration of Spindle execution plan.

Figure 5: Fig. 5a shows an example that Spindle allocates resources to 3 MetaOps on 4 devices. Fig. 5b shows an example of Spindle execution plan consisting of 6 Stages.

ASL-tuples $\langle \overline{n_m}, \cdot, \overline{l_m} \rangle$, $\langle \underline{n_m}, \cdot, \underline{l_m} \rangle$ to linearly represent the continuous, optimal solution $\langle n_m^*, 0, L_m \rangle$ in P_{MPSP}. To preserve the optimum property of P_{MPSP}, we require the discretized allocation plan to satisfy the following two conditions:

$$L_m = \overline{l_m} + \underline{l_m} \quad (10a) \qquad \widetilde{C}^* = T_m(\overline{n_m}) \cdot \overline{l_m} + T_m(\underline{n_m}) \cdot \underline{l_m} \quad (10b)$$

Cond. (10a) ensures these two discrete ASL-tuples complete the workload of MetaOp m, and Cond. (10b) ensures their total execution time is exactly equal to the minimum operator completion time $\widetilde{C}^*$ in P_{MPSP}, thus perserving the optimum property. Here we first select $\overline{n_m}, \underline{n_m}$ as the closest *valid* integer numbers such that $n_m^* \in [\underline{n_m}, \overline{n_m}]$, and $\overline{l_m}, \underline{l_m} \in \mathbb{R}^+$ are derived naturally. For instance, as shown in Fig. 5a, MetaOp 2 with $n_2^* = 1.5, L_2 = 12$ in P_{MPSP} is discretized as $\overline{n_2} = 2, \underline{n_2} = 1$ and $\overline{l_2} = 8.4, \underline{l_2} = 3.6$ in this step. Here we impose the *valid* constraint on the allocation n for MetaOp m for practical reasons. For instance, if an MetaOp is applied data parallelism, its allocation n is supposed to divide its global batch size B_m to avoid resource under-utilization due to uneven partition of samples. For another example, if an MetaOp is applied tensor parallelism or sequence parallelism with degree 2, its allocation n is supposed to be divisible by this degree, thus $n = 3, 5, 7$ as invalid. Such *valid* constraint ensures the allocation plan for each MetaOp is practical. Specially, allocation with $\underline{n_m} = 0$ is treated as a dummy allocation (e.g., MetaOp 3 in Fig. 5a), which preserves the optimum property of Cond. (10b) but will then be ignored.

Then, we reinstates l's as integers by rounding $\overline{l_m}, \underline{l_m}$ to the nearest integers. If the rounded l equals 0, this ASL-tuple will be ignored. This rounding procedure preserves the integrity of Cond. (10a) and introduces only minor bias to Cond. (10b). Finally, the discretized ASL-tuples of all MetaOps form the allocation plan. Note that the allocation plan only ensures the longest execution time among all MetaOps is approximately $\widetilde{C}^*$, yet it does not specify the start time for each ASL-tuple, which is determined by stage scheduler in §3.4.

3.4 Stage Scheduler

In this subsection, we describe how Spindle schedules the execution of MetaOps guided by the allocation plan generated by the resource allocator. We first introduce the concept of *Stage*, which is a scheduling unit of Spindle. Then we introduce our Stage-greedy scheduling algorithm, which schedules the execution of MetaOps greedily for each Stage. Finally, the operator dependencies among MetaLevels are reinstated by merging the Stage-based schedules together.

Definition of Stage. It is worthy to note that, although Theorem 1 implies that all MetaOps share the same start and end time in the continuous form, this property does not hold after the discretization process. The reason is that the execution time of ASL-tuples may vary, or the resources are insufficient to execute all tuples concurrently. To cope with this problem, we devise a fine-grained scheduler that slices the MetaOps and selects a few of them to execute concurrently, where the slicing and selection aim to achieve that (1) the devices are occupied as many as possible, and (2) their execution time are as close as possible. To ease the description, we define Stage as the scheduling unit, which corresponds to one concurrent execution as aforementioned. Next, we introduce our greedy algorithm that crafts the Stages to form the scheduling plan.

Stage-greedy Scheduling. As outlined in Alg. 1, the scheduler iteratively crafts Stages in a greedy manner. Below we introduce how one Stage is crafted with Fig. 5b as an example.

(1) First, the scheduler greedily proposes ASL-tuples to form a candidate set, aiming to utilize as many devices as possible (line 3). For instance with Fig. 5b, the scheduler proposes the first ASL-tuple of MetaOp 1 to craft Stage 1 since it occupies all devices. Similarly, for Stage 2, it proposes the ASL-tuples of MetaOp 1, 2, and 4, which correspond to 4, 2, 2 devices, respectively, in order to make full use of all devices.

Algorithm 1: Stage-greedy Scheduling for one MetaLevel

```
   Input: # Devices $N$, start time $T_{start}$,
          $alloc\_plan = \{m \rightarrow \{\langle \overline{n_m}, \cdot, \overline{l_m} \rangle, \langle \underline{n_m}, \cdot, \underline{l_m} \rangle\}\}$
   Output: Stage-based schedule $P = \bigcup_k \mathcal{S}_k$, end time $T_{end}$
 1 $T_{current} \leftarrow T_{start}$; $P \leftarrow \emptyset$; $\mathcal{S}_{remain} \leftarrow alloc\_plan$;
 2 while $\mathcal{S}_{remain}$ is not empty do // schedule for Stage $k$
 3     $\mathcal{S}_{cand} \leftarrow$ Propose_Candidate_Set($N, \mathcal{S}_{remain}$);
 4     $\mathcal{S}_{cand} \leftarrow$ Extend_Resources_If_Needed($\mathcal{S}_{cand}$);
 5     $T_{stage}, \mathcal{S}_{sched} \leftarrow$ Align_Time_Span($\mathcal{S}_{cand}$);
 6     $\mathcal{S}_k \leftarrow$ Set_Start_Time($\mathcal{S}_{sched}, T_{current}$); $P \leftarrow P \cup \mathcal{S}_k$;
 7     $\mathcal{S}_{remain} \leftarrow \mathcal{S}_{remain} - \mathcal{S}_{sched}$; $T_{current} \leftarrow T_{current} + T_{stage}$;
 8 return $P, T_{current}$
```

(2) If the candidate set fails to occupy all devices, the cluster resources will be underutilized. To address this issue, we extend the allocated resources in specific tuples to ensure all devices are utilized (line 4). For instance, in Stage 4 of Fig. 5b, the allocation of MetaOp 4 is extended from 1 device to 2 devices. Resource extension is prioritized for MetaOps with larger remaining execution time, with the hope of balancing the remaining workload among the MetaOps.

(3) In most cases, the proposed ASL-tuples differ in execution time. If we directly craft a Stage with them, it would be inefficient since there must be idle devices. Fortunately, this can be avoided by dissecting the ASL-tuples to align their time span (i.e., only a few number of operators in the MetaOp are scheduled in this Stage). For instance, in Stage 2 of Fig. 5b, the proposed ASL-tuples for MetaOp 1, 2, and 4 correspond to 9, 14, and 3 operators, respectively. To align the execution time, the ASL-tuples for MetaOp 1 and 2 are dissected, with only 1 and 2 operators of them being scheduled, while the remaining 8 and 13 operators left to be scheduled in subsequent Stages. Our scheduler simply aligns the time span w.r.t. the ASL-tuple with shortest execution time (e.g., the one for MetaOp 4 in the previous example), and computes the aligned time span as the duration of current stage (line 5).

(4) After the time span alignment, the scheduler concludes the current Stage (lines 6-7), including specifying the start time for operators that are scheduled in this Stage, and removing them from the remaining set.

Merging MetaLevels. As stated in §3.1, MetaOps are decoupled into MetaLevels to disentangle operator dependencies. Spindle invokes the aforementioned allocation and scheduling for each MetaLevel individually, and merges their Stage-based schedules together as the final execution schedule.

3.5 Device Placement

Given the Stage-based schedule, which consists of the allocation amount and the execution time of each MetaOp, we now discuss how Spindle determines the specific devices to allocate to each MetaOp, known as device placement. Device placement affects the inter-Stage communication overhead, as well as the memory consumption of each device. Spindle employs several guidelines based on empirical insights or observations to optimize device placement for MetaOps, as detailed below.

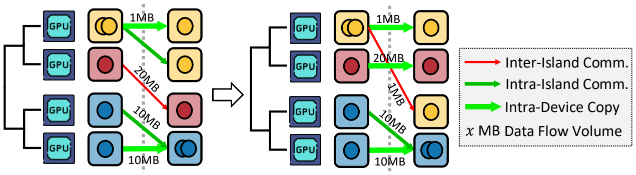


Figure 6: Illustration of Spindle device placement.

Intra-Device-Island Placement. Placement within a device island is always preferred for each MetaOp and each data flow between MetaOps. A device island consists of a group of devices connected by high-bandwidth interconnects (e.g., NVLink, PCIe), typically comprising adjacent devices, such as adjacent GPUs within one node. For MetaOps, prioritizing placement within the device island reduces the potential intra-MetaOp communication costs. For example, on a cluster with two GPUs per node, it's more efficient to place a MetaOp to a contiguous device group like GPU 0 and 1 within one node rather than a scattered group across two nodes. For data flow between MetaOps across Stages, intra-island placement reduces transmission costs leveraging the high intra-island bandwidth or even faster intra-device copying. For example, if data flow exists between MetaOp m and m', and they are assigned 1 and 2 GPUs, respectively, Spindle strives to place m on GPU 1 and m' on GPU 0 and 1. This arrangement allows data flow via intra-island communication (i.e., 1 to 0) or intra-device copying (i.e., 1 to 1), avoiding the inter-island communication costs that would occur if m were on device 1 and m' on device 2 and 3 on the other node.

Prioritizing High Communication Workloads. When the ideal scenarios outlined above are not achievable — that is, when it's infeasible to place all MetaOps within the device island nor to align all data flows on the same device group — MetaOps and data flows with higher communication volumes should be prioritized. Spindle estimates the communication volume of each MetaOp and each data flow to prioritize placing those with higher volumes within a device island and aligning high-volume data flows on the same device group. For instance, in Fig. 6, the data flow volumes between red MetaOps and blue MetaOps are significantly higher than that between yellow ones. Therefore, Spindle prefer to place the data flow between red and blue ones within the device island, while place the data flow between yellow ones across the island. This guideline ensures that the most communication-intensive components receive the most efficient hardware configuration to minimize communication overhead.

Device Memory Balance. As each device holds heterogeneous MetaOps, the memory overhead varies across devices. Placing too many memory-intensive MetaOps on a single device may cause out-of-memory errors. Therefore, Spindle actively strives to balance the memory load across all devices during device placement. Specifically, Spindle estimates the memory consumption of each MetaOp, record the available memory capacity of each device during placement, and prioritizes placing MetaOps to the device with the highest available memory capacity. Besides, for MetaOps sharing the same parameters, we prioritize placing them on the same device to minimize redundant storage.

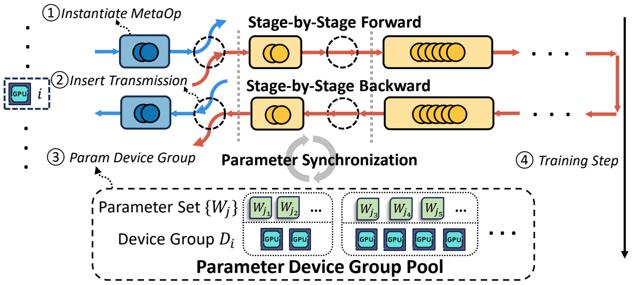


Figure 7: Illustration of Spindle runtime engine.

Based on these guidelines, Spindle performs device placement Stage by Stage greedily, prioritizing the minimization of communication overhead, such as inter-Stage transmission, while simultaneously maintaining device memory balance. When out-of-memory occurs due to imbalanced placement, Spindle will consider alternative placements with sub-optimal communication costs and better memory balance. If necessary, backtracking is employed to adjust the placements from earlier Stages to effectively address the out-of-memory issues.

3.6 Runtime Engine

The runtime engine is responsible for running the execution plan to facilitate efficient multi-task multi-modal training. This process is more complex than conventional single-task training, as each device handles heterogeneous MetaOps and local computation graphs. The Spindle runtime engine operates in four main steps:

(1) **Localization.** Initially, Spindle localizes the execution plan to each device. Specifically, each device instantiates the corresponding MetaOp of each Stage locally, and initializes the required model components and parameters.

(2) **Intra-task Data Dependency.** Secondly, Spindle inserts transmission operators to connect the MetaOps across Stages to handle the data flow dependencies, including activations from the forward pass and gradients from the backward pass. According to the device groups of MetaOps and data format requirements, operations such as *copy*, *shard*, *concat*, *send*, and *receive* are used to transmit data flows with minimal overhead. For example, a simple *copy* is sufficient for MetaOps that share the same device group. However, for complicated cases, more complex *send* and *receive* operations are necessary to transmit the data appropriately. This step not only correctly handles data flow dependencies between MetaOps but also links the MetaOps on each device into a complete local computation graph ready for execution.

(3) **Inter-task Model Dependency.** Then, Spindle manages parameter device groups for synchronization among various tasks by maintaining a global parameter device group pool. Specifically, during each iteration, for each parameter W_j, all tasks or modalities that activate it on different devices contribute to its gradient computation. These gradients need to be accumulated and synchronized to facilitate parameter sharing. Therefore, before the training process, Spindle scans all devices to determine the device group D_i for each parameter W_j, which represents W_j is shared and should be synchronized within group D_i. For efficiency, Spindle manages parameters with the same device group collectively and maintains a global parameter device group pool $\{D_i \rightarrow \{W_j\}\}$, where each device group D_i corresponds to a set of parameters $\{W_j\}$.

(4) **Training Step.** After the first three steps, the training process is ready to begin. In each iteration of Spindle, each device executes the forward and backward propagation of the local computation graph in a Stage-by-Stage manner, which is comprised of the interleaved execution of MetaOps and transmission of data flow. Following the forward and backward phases, Spindle performs group-wise parameter synchronization to maintain the parameter consistency. Specifically, each parameter set $\{W_j\}$ is synchronized within its corresponding device group D_i in the parameter device group pool.

4 IMPLEMENTATION

Spindle is an efficient and scalable MT MM training system built on PyTorch with 10K Loc in Python: 2.1K LoC for the Spindle execution planner and 7.9K LoC for the Spindle runtime framework. We implement the data flow transmission with NCCL batched P2P primitives and the parameter device groups with NCCL communication groups. Spindle provides the users with simple, user-friendly and flexible API for defining MT MM training workloads. Specifically, training tasks in Spindle are represented as *SpindleTask*, and users can define various multi-modal tasks by customizing PyTorch modules and connecting them flexibly through the *add_flow* API in Spindle. For example, a user can create a vision task by linking a vision encoder with a language model, or an audio task by linking an audio encoder with a language model. Alternatively, user can also define different computational logic for various tasks implicitly within a single unified model. Spindle can automatically split the modules and construct *SpindleTasks* via PyTorch FX Tracer, streamlining the process of task definition. After the definition of multi-modal tasks, Spindle conducts the optimization workflow automatically, as illustrated in Fig. 2, and the Spindle runtime engine provides efficient and scalable model training process.

5 EXPERIMENTS

5.1 Experimental Setups

Competitors. We evaluate the efficiency of Spindle by comparing it with state-of-the-art distributed training systems, Megatron-LM [68] and DeepSpeed [81]. As discussed in §1, these systems are primarily developed for single-task training and do not cater specifically to the complexities of multi-task multi-modal training scenarios. To further explore the advantages of Spindle's flexible resource allocation and scheduling capabilities, we introduce several baselines implemented on Spindle that represent typical strategies for multi-task training. The features of these competitors are summarized in Table 1.

(1)&(2) **Megatron-LM & DeepSpeed:** Megatron-LM [68] and DeepSpeed [81] are widely used state-of-the-art training systems tailored for single-task training. The naïve approach to train MT MM models on these systems is to decouple all sub-models on separate devices (§1), which requires plenty of resources and is impractical. Therefore, we decouple sub-models on temporal dimension within each iteration, where each sub-model takes up the whole cluster within a short time period, and is dependently and sequentially executed.

(3) **Spindle-Seq:** This baseline on Spindle allocates all available devices to each task and execute tasks sequentially within each

Table 1: Overview of system competitors.

Competitors	Heterogeneity Awareness	
	Inter-Task	Intra-Task
Megatron-LM / DeepSpeed / Spindle-Seq / Spindle-Uniform	✗	✗
Spindle-Optimus	✔	✗
Spindle-STMM	✗	✔
Spindle	✔	✔

iteration, similar to Megatron-LM and DeepSpeed. It reflects the performance of our Spindle system without specific optimizations for MT MM workloads.

(4) **Spindle-Uniform:** This baseline demonstrates a basic, workload-unaware task-level resource allocation strategy for multi-task multi-modal training. It allocates available devices uniformly to each task, and executes each task in parallel within each iteration.

(5) **Spindle-Optimus:** This baseline represents a workload-aware task-level resource allocation strategy, which adapts allocations according to the workload at the task level granularity. It's inspired by Optimus [72], an effective cluster job scheduling system which proposes a greedy resource allocation scheme and iteratively assigns devices to the job that has the largest marginal gain. Despite differences between job scheduling and multi-task training (§6), we apply a similar principle and devise the marginal gain as $(T_m^{(c)}(n) - T_m^{(c)}(n')/(n' - n)$, i.e., the task completion time reduction scaled by the allocation increment from n to n'. Here n' is the next valid allocation number larger than n. This baseline is aware of inter-task heterogeneity, whereas unaware that of intra-task.

(6) **Spindle-STMM:** This baseline represents a naïve multi-task (MT) extension of single-task (ST) multi-modal (MM) model training systems. It decouples multi-tasks, and for each single MM task it allocates resources to different modality encoders, akin to DistMM [32], a recent system designed for ST MM models. Then it executes tasks sequentially. Contrary to Spindle-Optimus, Spindle-STMM is aware of intra-task workload heterogeneity, whereas unaware that of inter-task.

Experimental Workloads. We conduct experiments on three different workloads of MT MM models, namely Multitask-CLIP [26, 75], OFASys [10], QWen-VAL [9, 19]. The configuration of these models are summarized in Table 2.

(1) **Multitask-CLIP:** Multitask-CLIP is a generalized version of CLIP [75], which extends CLIP to 6 modalities and multiple contrastive learning tasks of paired data modalities. We utilize the same model structure and configuration of ImageBind [26]. We select 10 different contrastive learning tasks for evaluation, each with distinct workloads.

(2) **OFASys:** OFASys[10] is a more general MT MM training workload, allowing modalities and tasks to activate the model components flexibly as needed. OFASys utilizes modality-specific adaptors for different modalities, e.g., ViT for vision data, and adopts a unified encoder-decoder LM with generative loss. We select 7 different multi-modal tasks for evaluation.

(3) **QWen-VAL:** QWen-VAL is a larger-scale MT MM model with up to 9.25 billion parameters, supporting three modalities, including text, vision, and audio. It adopts the same structure and configuration of the popular open-sourced multi-modal LLMs, QWen-VL [9] and QWen-Audio [19]. It has modality encoders for audio and vision, and the extracted modality-specific features are combined with text tokens and together fed into the unified LLM, QWen [8]. We select three tasks for evaluation, i.e., vision-language (VL) task, audio-language (AL) task, and vision-audio-language (VAL) task, representing different combinations of modalities.

Table 2: Configuration of MT MM models for evaluation.

MM MT Model	# Param.	Modalities
Multitask-CLIP	1.20 B	Text, Vision, Audio, Motion, Thermal, Depth
OFASys	0.66 B	Text, Vision, Audio, Motion, Box, Structure
QWen-VAL	9.25 B	Text, Vision, Audio

Protocols. We conduct all the experiments on an 8-node GPU cluster. Each node consists of 8 NVIDIA A800 80 GB GPUs equipped with NVLink, and the nodes are interconnected by 400 Gbps InfiniBand network. Since the baseline systems do not support automatic planning given a targeted MT MM model training workload, to achieve a fair comparison, we manually tune their parallel configurations and memory optimization techniques (e.g., data parallelism degree, tensor parallelism degree, ZeRO stage, activation checkpointing, and etc.) to achieve the best performance. For each system on each workload, we evaluate the system performance on different cluster sizes and report the iteration time averaged over 100 iterations.

5.2 End-to-End Performance

Fig. 8 displays end-to-end comparisons between Spindle and baseline systems across various model workloads, multi-modal task configurations, and cluster sizes.

Comparison with SOTA systems. In general, compared to state-of-the-art (SOTA) training systems, i.e., Megatron-LM and DeepSpeed, Spindle achieves speedup ratios of up to 67% and 71%, respectively. Below we delve into the performance advantages of Spindle.

To begin with, Spindle consistently outperforms the competitors across different task configurations and numbers of tasks. Notably, Spindle excels when handling a larger number of tasks. When comparing with Megatron-LM and DeepSpeed on the 10-task Multitask-CLIP workloads, Spindle achieves speedup ratios ranging from 31% to 63% and 33% to 71% compared to Megatron-LM and DeepSpeed, respectively. Similar results are shown for the 7-task OFASys workloads, with the speedup ratios ranging from 31% to 67% and 33% to 71%, respectively. This underscores Spindle's excellent scalability with increasing task counts.

In addition, Spindle consistently achieves optimal performance across various cluster sizes. For instance, on Multitask-CLIP, compared to SOTA systems, Spindle achieves the highest speedup ratios of 37%, 33%, and 71% on 8, 16, and 32 GPUs, respectively. Similarly, on OFASys, Spindle achieves acceleration ratios up to 71%, 46%, and 51% on 8, 16, and 32 GPUs, respectively. These results highlight Spindle's excellent scalability w.r.t. cluster size. Notably, Spindle maintains high efficiency even when the scalability of SOTA systems begins to diminish — that is, when the increase in resources does not correspond to significant speed improvements. For example, in the 4-task Multitask-CLIP scenario, expanding the cluster size from 16 to 32 GPUs results in only modest speedup of 1.21×

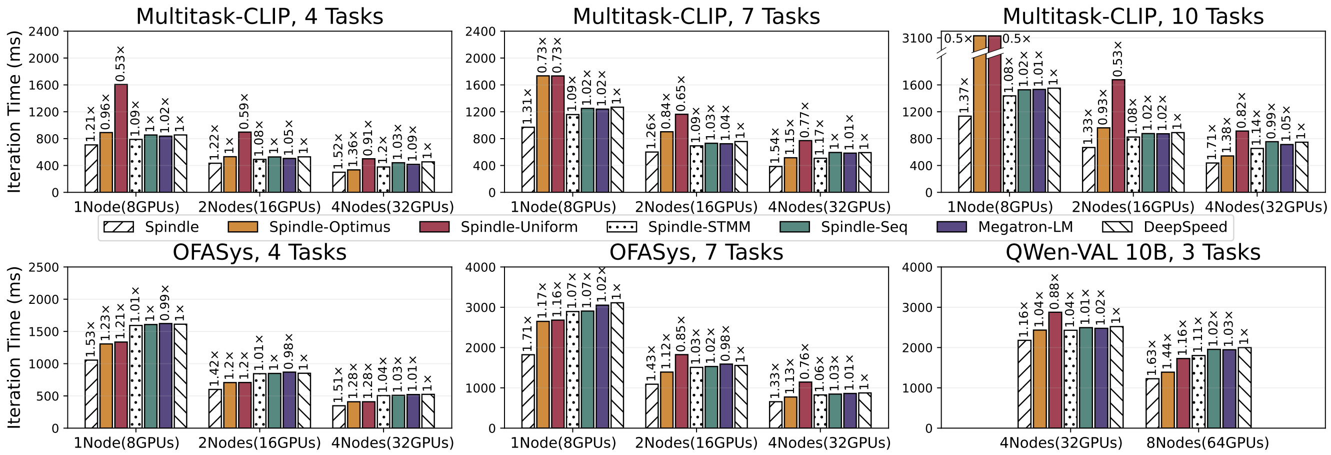


Figure 8: End-to-end performance comparison for Spindle and baseline systems. Shorter bars indicate superior system performance. The numbers above the bars denote each system's speedup compared to DeepSpeed (larger than 1 is faster).

and 1.17× for Megatron-LM and DeepSpeed, respectively, whereas Spindle still achieves a 1.45× speedup. This efficiency stems from Spindle 's carefully designed resource allocation and scheduling mechanisms. Unlike existing systems that naively allocate all resources across all operators and tasks, Spindle ensures that each operator is allocated suitable resources when the cluster size increases, to maintain high computational efficiency.

More importantly, Spindle also exhibits excellent scalability w.r.t. model size. On larger models QWen-VAL with 9.25 billion parameters, Spindle achieves a maximum speedup of 1.16× on 32 GPUs and 1.63× on 64 GPUs, compared to SOTA systems. Notably, when training the QWen-VAL over 64 GPUs, Spindle shows remarkable scalability: it achieves a 1.78× speedup when scaling from 32 to 64 GPUs, whereas Megatron-LM and DeepSpeed only achieve 1.27× and 1.26× speedups, respectively. This is unsurprising since Spindle allocates cluster resources across different operators more flexibly, thereby avoiding the unsatisfactory scalability of MetaOps with light workloads, as discussed in §3.2.

Comparison with other baselines. Next, we discuss the performance of the variants of Spindle. Since Spindle-Seq has a comparable performance against Megatron-LM and DeepSpeed in most cases — which is reasonable as all three counterparts execute tasks sequentially — we focus on the comparison with task-level resource allocation strategies, i.e., Spindle-Uniform and Spindle-Optimus, as well as the single-task strategy, i.e., Spindle-STMM.

We find that the workload-unaware uniform allocation of Spindle-Uniform performs well only in limited scenarios, achieving a maximum speedup ratio of 28% over DeepSpeed. This suggests that resource allocation can enhance computational efficiency to some extent. However, it generally underperforms compared to SOTA systems due to its tendency to distribute resources evenly, leading to unbalanced workloads across tasks and system performance being constrained by the most resource-intensive tasks.

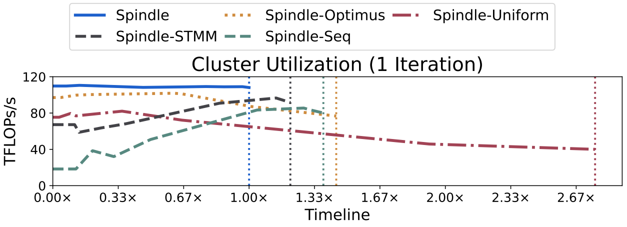
In contrast, Spindle-Optimus, which allocates resources based on task workloads, shows better performance, especially in larger-scale cluster scenarios, with the speedup ratio up to 44% compared to DeepSpeed. However, there are still many scenarios where Spindle-Optimus underperforms, sometimes even falling behind DeepSpeed. This is because Spindle-Optimus's task-level resource allocation overlooks the workload heterogeneity within tasks, thereby limiting training efficiency. Moreover, its coarse granularity of task-level allocation can sometimes fail to achieve ideal load balancing among tasks, often resulting in performance being constrained by the slowest task.

In comparison, the operator-level strategy employed by Spindle enables finer-grained resource allocation and load balancing, consistently achieving higher efficiency compared to task-level strategies. We find that even in scenarios where Spindle-Optimus has already surpassed the performance of SOTA systems, Spindle still manages to achieve a speedup ratio of up to 45% over Spindle-Optimus (4-task Multitask-CLIP on 8GPUs), verifying its superiority.

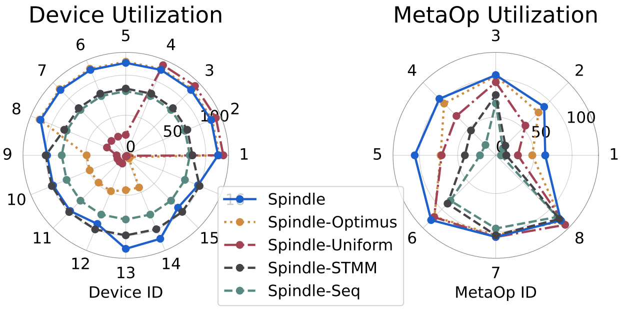
As for Spindle-STMM, we find it perform better than SOTA systems in most cases, with the speedup ratio up to 20%, benefiting from its intra-task workload awareness and resource allocation. However, it's designed for single-task (ST) multi-modal (MM) models, which decouples tasks and optimizes each task separately, and such single-task strategy is not the global optimum for multi-task cases. The lack of awareness of inter-task heterogeneity limits its performance, causing it to underperform compared to the task-level strategy Spindle-Optimus in many cases. For OFASys, Spindle-STMM shows almost similar performance to SOTA systems. This is because Spindle-STMM gains acceleration by parallelizing sub-models of the multi-tower structure. In contrast, OFASys utilizes a lightweight text adaptor, so most tasks that pair a modality with text are dominated by the other modality, making the intra-task parallelization of sub-models ineffective. Compared to Spindle-STMM, Spindle jointly optimizes the allocation and scheduling of all tasks and operators, taking into account both intra-task and inter-task workload heterogeneity. This enables Spindle to consistently outperform the single-task strategy of Spindle-STMM, achieving a speedup ratio of up to 59%.

5.3 Case Study

To better understand the advantages and performance gain of Spindle over the other competitors, we further conduct an in-depth case study of Multitask-CLIP (4 tasks, 16 GPUs). Fig. 9 presents system



(a) Average cluster utilization over time within one training iteration. Higher positions on the y-axis indicate higher utilization.



(b) Utilization of each device and each MetaOp. Points closer to the outer edge of the spider chart represent higher utilization.

Figure 9: Performance case study of Multitask-CLIP (4 tasks, 16 GPUs). Utilization is measured in computation FLOPs per second.

performance considering three key metrics: cluster average utilization over time, average utilization per device, and computational utilization of each MetaOp.

Firstly, Spindle-Seq, which executes the tasks sequentially with all resources, experiences fluctuating utilization due to the workload heterogeneity, leading to generally low overall utilization. Spindle-Uniform, which allocates resources uniformly at the task level, improves cluster utilization to some extent at the iteration beginning, but as tasks with light workloads finish, more devices become idle, declining overall utilization. Spindle-Optimus partially mitigates this imbalance with workload-aware allocation, though it still suffers from utilization drops due to its coarse granularity of task-level allocation. Spindle-STMM manages to enhance utilization via intra-task resource allocation for each task compared to Spindle-Seq, but the ignorance of inter-task heterogeneity limits its utilization. In contrast, Spindle maintains consistently high utilization and the shortest iteration times thanks to its joint optimization of the unified computation graph of all tasks and operators, which addresses the heterogeneity both within and among tasks.

Furthermore, Spindle significantly elevates the utilization of all devices and all MetaOps, showcasing its superior handling of workload balance through operator-level strategies. In contrast, Spindle-Seq shows lower utilization across all devices and MetaOps. Although task-level strategies can enhance the computational efficiency of certain devices, the coarse granularity of allocation inevitably leads to workload imbalances, leaving many devices underutilized, sometimes even worse than Spindle-Seq, resulting in poor average cluster utilization. Spindle-STMM also improves the utilization of certain devices and MetaOps, but the results are still unsatisfactory as it fails to capture the workload differences among tasks, and fails to reach the global optimal allocation and scheduling plan for multi-tasks.

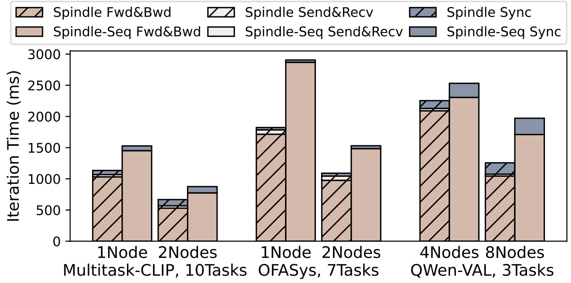


Figure 10: Time breakdown analysis.

Overall, Spindle's unified optimization of MT MM models captures both intra-task and inter-task heterogeneity, and effectively balances workloads. Thus, it consistently enhances utilization across all operators and devices, and maintains high computational efficiency across the cluster.

5.4 Time Breakdown

Fig. 10 shows the runtime breakdown for Spindle and Spindle-Seq across various workloads, primarily consisting of forward and backward propagation, parameter synchronization, and inter-Stage *send* and *receive*. We've isolated parameter synchronization from the backward phase for individual analysis. In MT MM training, we find that forward and backward propagation dominate the runtime, typically accounting for 80%-95% due to the large number of tasks and computational demands. Spindle focuses on reducing this significant time component through flexible resource allocation and scheduling. Parameter synchronization usually consumes a small fraction of the time, about 5%-15%, since it only occurs after accumulating gradients from multiple tasks. Notably, Spindle consistently achieves equal or lower synchronization cost compared to Spindle-Seq. For instance, on 32 GPUs with QWen-VAL, Spindle cuts synchronization time to just 55% of that of Spindle-Seq. Although not the primary optimization focus, Spindle's operator-level design inherently reduces synchronization overhead. This is achieved by synchronizing each parameter only within the device group that activates it and leveraging Spindle's device island placement to convert potentially inter-island synchronizations into more local communications within device islands. Furthermore, we find that while Spindle introduces extra overhead for inter-Stage *send* and *receive*, this overhead remains minimal, typically not exceeding 6%, thanks to the Spindle device placement mechanism that avoids unnecessary communications. Detailed ablation study of device placement is in §5.5.2.

5.5 Component Analysis

5.5.1 Optimality Analysis of Execution Planner. We analyze the optimality of Spindle execution planner in Fig. 11. Specifically, we compare the iteration time of Spindle to the theoretical optimal time $\widetilde{C}^*$ derived from Theorem 1 in §3. As discussed in §3.3, when relaxing the constraints of the optimization problem (4) and allowing the continuous divisibility of GPU resources n and operator number l, Theorem 1 offers the theoretical optimum P_{MPSP} and corresponding optimal time $\widetilde{C}^*$. Such a solution is unachievable due

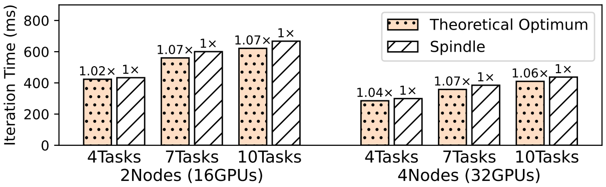


Figure 11: Optimality analysis. Evaluated on Multitask-CLIP 4/7/10-tasks on 16/32 GPUs. The theoretical optimum represents $\widetilde{C}^*$ in Theorem 1 in §3.

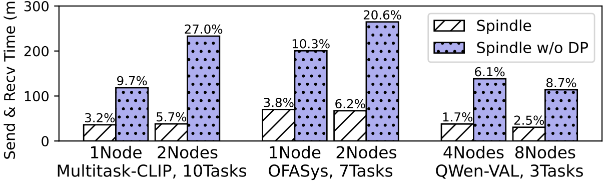


Figure 12: Inter-Stage communication duration comparison whether device placement is applied. The percentage in end-to-end training time is labeled on top of each bar.

to these relaxed constraints, but serves as a theoretical upper bound of performance. The Spindle execution planner preserves most of the optimum property when finding the practical solution (e.g., Cond. (10a) and Cond. (10b) in §3.3), but still introducing minor biases (e.g., reinstating $l's$ to integers in §3.3, resource extension in in §3.4). In Fig. 11, we calculate and estimate the theoretical optimum $\widetilde{C}^*$ according to Theorem 1, and compare it with Spindle's iteration time. We find that across various task configurations and cluster sizes, the deviation between Spindle and theoretical optimum is consistently low, below 7%. This observation underscores the effectiveness of Spindle in offering a practical and near-optimal execution plan for MT MM models. Besides, Spindle efficiently generates the execution plans within 3 seconds across all experiments, which is negligible compared to model training time.

5.5.2 Ablation on Device Placement. We conduct an ablation study on the device placement strategy in §3.5, focusing on its impact on inter-Stage communication overhead, which is the extra overhead introduced by our system. Specifically, we compare Spindle's device placement strategy with a sequential placement strategy, which naïvely assigns each MetaOp with consecutive devices. Our results indicate that the inter-Stage communication overhead of the sequential placement strategy is approximately 3-6 times greater than that of Spindle, taking up to 27% of the end-to-end training time, which is considerably high. However, with Spindle's placement strategies, this overhead only takes up to 6%. This demonstrates the effectiveness of our locality-aware placement, which significantly reduces the extra communication overhead.

5.6 Dynamicity Performance

We evaluate the performance of various systems during dynamic changes of the multi-task workloads, a common occurrence in MT MM training. For instance, tasks with fewer training data may exit early, and new tasks may join partway through training. We simulate these dynamic changes by altering the training task set. When the multi-task workloads change, the current model is first saved, and the new set of tasks and the saved model is loaded to continue training. Fig. 13 illustrates the performance of each system under such conditions. Spindle consistently achieves optimal training efficiency and the shortest overall training time. This advantage is due to Spindle's adaptability to dynamically changing workloads, enabling it to adopt an appropriate execution plan for the efficient training of MT MM models.

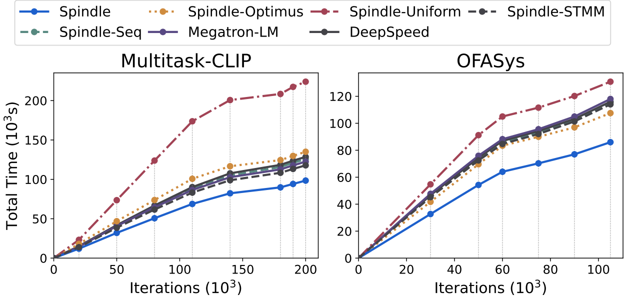


Figure 13: Comparison on dynamic multi-task workloads. Dots on the curve mark the points when the multi-task workload changes.

6 RELATED WORKS

Cluster Scheduling for DL Jobs. GPU clusters often design cluster schedulers to coordinate resource allocation and the execution order among multiple DL jobs. Some cluster schedulers [27, 104] allocate resources to jobs based directly on user-specified requirements. Others [43, 53, 60, 67, 72, 74, 105, 110] automatically allocate resources to each job based on the job scalability to the computing resource. Many of these schedulers aim to minimize job completion time (JCT). For instance, Optimus [72] introduces the concept of marginal gain to guide resource allocation, aiming to minimize job completion time. Here, we highlight the difference of these works and MT MM model training. Unlike the independence among jobs in cluster scheduling, MT MM training involves execution dependencies among tasks. Furthermore, while traditional scheduling focuses on job-level allocation, MT MM training requires finer-grained strategies to address intra-task workload heterogeneity.

Data and Model Management Optimization. The data management community has developed effective systems for managing data and models in specific domains, such as graph-structured data and models [91, 96, 97, 112–114], recommendation system data [28, 57, 59], tabular data [1, 6, 34, 50], and video data [13, 30, 106], etc. However, no existing work focuses on optimizing multi-task (MT) multi-modal (MM) data and model management, which is the key problem that Spindle addresses.

7 CONCLUSION

Efficient training of MT MM models faces significant system challenges due to the workload heterogeneity and complex execution dependency. In this paper, we propose Spindle to facilitate efficient training of MT MM models via data heterogeneity-aware model management optimization, which jointly optimizes heterogeneity-aware workload parallelization and dependency-driven execution scheduling. Extensive experiments demonstrate the consistent superior performance of Spindle, outperforming existing state-of-the-art training systems with speedup ratio up to 71%.

REFERENCES

[1] Mahmoud Abo Khamis, Hung Q Ngo, XuanLong Nguyen, Dan Olteanu, and Maximilian Schleich. 2018. In-database learning with sparse tensors. In *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*. 325–340.

[2] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774* (2023).

[3] Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katherine Millican, Malcolm Reynolds, Roman Ring, Eliza Rutherford, Serkan Cabi, Tengda Han, Zhitao Gong, Sina Samangooei, Marianne Monteiro, Jacob L. Menick, Sebastian Borgeaud, Andy Brock, Aida Nematzadeh, Sahand Sharifzadeh, Mikolaj Binkowski, Ricardo Barreira, Oriol Vinyals, Andrew Zisserman, and Karén Simonyan. 2022. Flamingo: a Visual Language Model for Few-Shot Learning. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh (Eds.). http://papers.nips.cc/paper_files/paper/2022/hash/960a172bc7fbf0177ccccbb411a7d800-Abstract-Conference.html

[4] Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M. Dai, Anja Hauth, Katie Millican, David Silver, Slav Petrov, Melvin Johnson, Ioannis Antonoglou, Julian Schrittwieser, Amelia Glaese, Jilin Chen, Emily Pitler, Timothy P. Lillicrap, Angeliki Lazaridou, Orhan Firat, James Molloy, Michael Isard, Paul Ronald Barham, Tom Hennigan, Benjamin Lee, Fabio Viola, Malcolm Reynolds, Yuanzhong Xu, Ryan Doherty, Eli Collins, Clemens Meyer, Eliza Rutherford, Erica Moreira, Kareem Ayoub, Megha Goel, George Tucker, Enrique Piqueras, Maxim Krikun, Iain Barr, Nikolay Savinov, Ivo Danihelka, Becca Roelofs, Anaïs White, Anders Andreassen, Tamara von Glehn, Lakshman Yagati, Mehran Kazemi, Lucas Gonzalez, Misha Khalman, Jakub Sygnowski, and et al. 2023. Gemini: A Family of Highly Capable Multimodal Models. *CoRR* abs/2312.11805 (2023). https://doi.org/10.48550/ARXIV.2312.11805 arXiv:2312.11805

[5] Anurag Arnab, Mostafa Dehghani, Georg Heigold, Chen Sun, Mario Lucic, and Cordelia Schmid. 2021. ViViT: A Video Vision Transformer. In *2021 IEEE/CVF International Conference on Computer Vision, ICCV 2021, Montreal, QC, Canada, October 10-17, 2021*. IEEE, 6816–6826. https://doi.org/10.1109/ICCV48922.2021.00676

[6] Gilbert Badaro and Paolo Papotti. 2022. Transformers for tabular data representation: A tutorial on models and applications. *Proceedings of the VLDB Endowment* 15, 12 (2022), 3746–3749.

[7] Alexei Baevski, Yuhao Zhou, Abdelrahman Mohamed, and Michael Auli. 2020. wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (Eds.). https://proceedings.neurips.cc/paper/2020/hash/92d1e1eb1cd6f9fba3227870bb6d7f07-Abstract.html

[8] 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. *CoRR* abs/2309.16609 (2023). https://doi.org/10.48550/ARXIV.2309.16609 arXiv:2309.16609

[9] Jinze Bai, Shuai Bai, Shusheng Yang, Shijie Wang, Sinan Tan, Peng Wang, Junyang Lin, Chang Zhou, and Jingren Zhou. 2023. Qwen-VL: A Frontier Large Vision-Language Model with Versatile Abilities. *CoRR* abs/2308.12966 (2023). https://doi.org/10.48550/ARXIV.2308.12966 arXiv:2308.12966

[10] Jinze Bai, Rui Men, Hao Yang, Xuancheng Ren, Kai Dang, Yichang Zhang, Xiaohuan Zhou, Peng Wang, Sinan Tan, An Yang, Zeyu Cui, Yu Han, Shuai Bai, Wenbin Ge, Jianxin Ma, Junyang Lin, Jingren Zhou, and Chang Zhou. 2022. OFASys: A Multi-Modal Multi-Task Learning System for Building Generalist Models. *CoRR* abs/2212.04408 (2022). https://doi.org/10.48550/ARXIV.2212.04408 arXiv:2212.04408

[11] Hangbo Bao, Li Dong, Songhao Piao, and Furu Wei. 2022. BEiT: BERT Pre-Training of Image Transformers. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net. https://openreview.net/forum?id=p-BhZSz59o4

[12] Hangbo Bao, Wenhui Wang, Li Dong, Qiang Liu, Owais Khan Mohammed, Kriti Aggarwal, Subhojit Som, Songhao Piao, and Furu Wei. 2022. VLMo: Unified Vision-Language Pre-Training with Mixture-of-Modality-Experts. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh (Eds.). http://papers.nips.cc/paper_files/paper/2022/hash/d46662aa53e78a62afd980a29e0c37ed-Abstract-Conference.html

[13] Favyen Bastani, Oscar Moll, and Sam Madden. 2020. Vaas: video analytics at scale. *Proceedings of the VLDB Endowment* 13, 12 (2020), 2877–2880.

[14] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. In *NeurIPS*.

[15] Christoph Brücke, Philipp Härtling, Rodrigo D Escobar Palacios, Hamesh Patel, and Tilmann Rabl. 2023. TPCx-AI-An Industry Standard Benchmark for Artificial Intelligence and Machine Learning Systems. *Proceedings of the VLDB Endowment* 16, 12 (2023), 3649–3661.

[16] Chengliang Chai, Jiayi Wang, Yuyu Luo, Zeping Niu, and Guoliang Li. 2022. Data management for machine learning: A survey. *IEEE Transactions on Knowledge and Data Engineering* 35, 5 (2022), 4646–4667.

[17] Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. 2016. Training Deep Nets with Sublinear Memory Cost. *CoRR* abs/1604.06174 (2016). arXiv:1604.06174 http://arxiv.org/abs/1604.06174

[18] Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E Gonzalez, et al. 2023. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality. _See https://vicuna. lmsys. org (accessed 14 April 2023)_ 2, 3 (2023), 6.

[19] Yunfei Chu, Jin Xu, Xiaohuan Zhou, Qian Yang, Shiliang Zhang, Zhijie Yan, Chang Zhou, and Jingren Zhou. 2023. Qwen-Audio: Advancing Universal Audio Understanding via Unified Large-Scale Audio-Language Models. *CoRR* abs/2311.07919 (2023). https://doi.org/10.48550/ARXIV.2311.07919 arXiv:2311.07919

[20] Tri Dao. 2023. FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning. *CoRR* abs/2307.08691 (2023). https://doi.org/10.48550/ARXIV.2307.08691 arXiv:2307.08691

[21] 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 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh (Eds.). http://papers.nips.cc/paper_files/paper/2022/hash/67d57c32e20fd0a7a302cb81d36e40d5-Abstract-Conference.html

[22] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL-HLT*. 4171–4186.

[23] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, et al. 2021. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *ICLR*.

[24] Danny Driess, Fei Xia, Mehdi S. M. Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, Wenlong Huang, Yevgen Chebotar, Pierre Sermanet, Daniel Duckworth, Sergey Levine, Vincent Vanhoucke, Karol Hausman, Marc Toussaint, Klaus Greff, Andy Zeng, Igor Mordatch, and Pete Florence. 2023. PaLM-E: An Embodied Multimodal Language Model. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA (Proceedings of Machine Learning Research)*, Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (Eds.), Vol. 202. PMLR, 8469–8488. https://proceedings.mlr.press/v202/driess23a.html

[25] Maciej Drozdowski. 2009. *Scheduling for Parallel Processing* (1st ed.). Springer Publishing Company, Incorporated.

[26] Rohit Girdhar, Alaaeldin El-Nouby, Zhuang Liu, Mannat Singh, Kalyan Vasudev Alwala, Armand Joulin, and Ishan Misra. 2023. ImageBind One Embedding Space to Bind Them All. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2023, Vancouver, BC, Canada, June 17-24, 2023*. IEEE, 15180–15190. https://doi.org/10.1109/CVPR52729.2023.01457

[27] Juncheng Gu, Mosharaf Chowdhury, Kang G. Shin, Yibo Zhu, Myeongjae Jeon, Junjie Qian, Hongqiang Harry Liu, and Chuanxiong Guo. 2019. Tiresias: A GPU Cluster Manager for Distributed Deep Learning. In *16th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2019, Boston, MA, February 26-28, 2019*, Jay R. Lorch and Minlan Yu (Eds.). USENIX Association, 485–500. https://www.usenix.org/conference/nsdi19/presentation/gu

[28] Saket Gurukar, Nikil Pancha, Andrew Zhai, Eric Kim, Samson Hu, Srinivasan Parthasarathy, Charles Rosenberg, and Jure Leskovec. 2022. MultiBiSage: A Web-Scale Recommendation System Using Multiple Bipartite Graphs at Pinterest. *Proceedings of the VLDB Endowment* 16, 4 (2022), 781–789.

[29] Andrey Guzhov, Federico Raue, Jörn Hees, and Andreas Dengel. 2022. Audioclip: Extending Clip to Image, Text and Audio. In *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2022, Virtual and Singapore, 23-27 May 2022*. IEEE, 976–980. https://doi.org/10.1109/ICASSP43922.2022.9747631

[30] Brandon Haynes, Amrita Mazumdar, Magdalena Balazinska, Luis Ceze, and Alvin Cheung. 2019. Visual road: A video data management benchmark. In *Proceedings of the 2019 International Conference on Management of Data*. 972–987.

[31] Roger W. Hockney. 1994. The Communication Challenge for MPP: Intel Paragon and Meiko CS-2. *Parallel Comput.* 20, 3 (1994), 389–398. https://doi.org/10.1016/S0167-8191(06)80021-9

[32] 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)*. 1157–1171.

[33] Yanping Huang, Youlong Cheng, Ankur Bapna, et al. 2019. GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism. In *NeurIPS*.

[34] Amir Ilkhechi, Andrew Crotty, Alex Galakatos, Yicong Mao, Grace Fan, Xiran Shi, and Ugur Cetintemel. 2020. Deepsqueeze: Deep semantic compression for tabular data. In *Proceedings of the 2020 ACM SIGMOD international conference on management of data*. 1733–1746.

[35] Chao Jia, Yinfei Yang, Ye Xia, Yi-Ting Chen, Zarana Parekh, Hieu Pham, Quoc V. Le, Yun-Hsuan Sung, Zhen Li, and Tom Duerig. 2021. Scaling Up Visual and Vision-Language Representation Learning With Noisy Text Supervision. In *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event (Proceedings of Machine Learning Research)*, Marina Meila and Tong Zhang (Eds.), Vol. 139. PMLR, 4904–4916. http://proceedings.mlr.press/v139/jia21b.html

[36] Zhihao Jia, Matei Zaharia, and Alex Aiken. 2019. Beyond Data and Model Parallelism for Deep Neural Networks. In *MLSys*.

[37] Wonjae Kim, Bokyung Son, and Ildoo Kim. 2021. ViLT: Vision-and-Language Transformer Without Convolution or Region Supervision. In *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event (Proceedings of Machine Learning Research)*, Marina Meila and Tong Zhang (Eds.), Vol. 139. PMLR, 5583–5594. http://proceedings.mlr.press/v139/kim21k.html

[38] Arun Kumar. 2021. Automation of data prep, ML, and data science: New cure or snake oil?. In *Proceedings of the 2021 International Conference on Management of Data*. 2878–2880.

[39] Arun Kumar, Matthias Boehm, and Jun Yang. 2017. Data management in machine learning: Challenges, techniques, and systems. In *Proceedings of the 2017 ACM International Conference on Management of Data*. 1717–1722.

[40] Junnan Li, Dongxu Li, Silvio Savarese, and Steven C. H. Hoi. 2023. BLIP-2: Bootstrapping Language-Image Pre-training with Frozen Image Encoders and Large Language Models. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA (Proceedings of Machine Learning Research)*, Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (Eds.), Vol. 202. PMLR, 19730–19742. https://proceedings.mlr.press/v202/li23q.html

[41] Junnan Li, Dongxu Li, Caiming Xiong, and Steven C. H. Hoi. 2022. BLIP: Bootstrapping Language-Image Pre-training for Unified Vision-Language Understanding and Generation. In *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA (Proceedings of Machine Learning Research)*, Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvári, Gang Niu, and Sivan Sabato (Eds.), Vol. 162. PMLR, 12888–12900. https://proceedings.mlr.press/v162/li22n.html

[42] Junnan Li, Ramprasaath R. Selvaraju, Akhilesh Gotmare, Shafiq R. Joty, Caiming Xiong, and Steven Chu-Hong Hoi. 2021. Align before Fuse: Vision and Language Representation Learning with Momentum Distillation. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan (Eds.). 9694–9705. https://proceedings.neurips.cc/paper/2021/hash/505259756244493872b7709a8a01b536-Abstract.html

[43] Jiamin Li, Hong Xu, Yibo Zhu, Zherui Liu, Chuanxiong Guo, and Cong Wang. 2023. Lyra: Elastic scheduling for deep learning clusters. In *Proceedings of the Eighteenth European Conference on Computer Systems*. 835–850.

[44] 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. *PVLDB* 13, 12 (2020), 3005–3018.

[45] 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 (2020), 3005–3018. https://doi.org/10.14778/3415478.3415530

[46] Qiuru Lin, Sai Wu, Junbo Zhao, Jian Dai, Meng Shi, Gang Chen, and Feifei Li. 2023. SmartLite: A DBMS-Based Serving System for DNN Inference in Resource-Constrained Environments. *Proceedings of the VLDB Endowment* 17, 3 (2023), 278–291.

[47] Haotian Liu, Chunyuan Li, Yuheng Li, and Yong Jae Lee. 2023. Improved Baselines with Visual Instruction Tuning. *CoRR* abs/2310.03744 (2023). https://doi.org/10.48550/ARXIV.2310.03744 arXiv:2310.03744

[48] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. 2023. Visual Instruction Tuning. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (Eds.). http://papers.nips.cc/paper_files/paper/2023/hash/6dcf277ea32ce3288914faf369fe6de0-Abstract-Conference.html

[49] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. 2021. Swin Transformer: Hierarchical Vision Transformer using Shifted Windows. In *ICCV*. IEEE, 9992–10002.

[50] Zifan Liu, Zhechun Zhou, and Theodoros Rekatsinas. 2022. Picket: guarding against corrupted data in tabular data during learning and inference. *The VLDB Journal* 31, 5 (2022), 927–955.

[51] Jiasen Lu, Dhruv Batra, Devi Parikh, and Stefan Lee. 2019. ViLBERT: Pretraining Task-Agnostic Visiolinguistic Representations for Vision-and-Language Tasks. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett (Eds.). 13–23. https://proceedings.neurips.cc/paper/2019/hash/c74d97b01eae257e44aa9d5bade97baf-Abstract.html

[52] Jiasen Lu, Christopher Clark, Rowan Zellers, Roozbeh Mottaghi, and Aniruddha Kembhavi. 2023. UNIFIED-IO: A Unified Model for Vision, Language, and Multi-modal Tasks. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net. https://openreview.net/pdf?id=E01k9048soZ

[53] Kshiteej Mahajan, Arjun Balasubramanian, Arjun Singhvi, Shivaram Venkataraman, Aditya Akella, Amar Phanishayee, and Shuchi Chawla. 2020. Themis: Fair and Efficient GPU Cluster Scheduling. In *17th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2020, Santa Clara, CA, USA, February 25-27, 2020*, Ranjita Bhagwan and George Porter (Eds.). USENIX Association, 289–304. https://www.usenix.org/conference/nsdi20/presentation/mahajan

[54] Xupeng Miao, Zhihao Jia, and Bin Cui. 2024. Demystifying Data Management for Large Language Models. In *Companion of the 2024 International Conference on Management of Data*. 547–555.

[55] Xupeng Miao, Xiaonan Nie, Yingxia Shao, Zhi Yang, Jiawei Jiang, Lingxiao Ma, and Bin Cui. 2021. Heterogeneity-Aware Distributed Machine Learning Training via Partial Reduce. In *SIGMOD*. ACM, 2262–2270.

[56] Xupeng Miao, Yining Shi, Zhi Yang, Bin Cui, and Zhihao Jia. 2023. SDPipe: A Semi-Decentralized Framework for Heterogeneity-aware Pipeline-parallel Training. *Proc. VLDB Endow.* 16, 9 (2023), 2354–2363. https://doi.org/10.14778/3598581.3598604

[57] Xupeng Miao, Yining Shi, Hailin Zhang, Xin Zhang, Xiaonan Nie, Zhi Yang, and Bin Cui. 2022. HET-GMP: A Graph-based System Approach to Scaling Large Embedding Model Training. In *SIGMOD*. 470–480.

[58] 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. *Proc. VLDB Endow.* 16, 3 (2022), 470–479. https://doi.org/10.14778/3570690.3570697

[59] Xupeng Miao, Hailin Zhang, Yining Shi, Xiaonan Nie, Zhi Yang, Yangyu Tao, and Bin Cui. 2022. HET: Scaling out Huge Embedding Model Training via Cache-enabled Distributed Framework. *PVLDB* 15, 2 (2022), 312–320.

[60] Zizhao Mo, Huanle Xu, and Chengzhong Xu. 2024. Heet: Accelerating Elastic Training in Heterogeneous Deep Learning Clusters. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. 499–513.

[61] Seungwhan Moon, Andrea Madotto, Zhaojiang Lin, Alireza Dirafzoon, Aparajita Saraf, Amy Bearman, and Babak Damavandi. 2022. IMU2CLIP: Multimodal Contrastive Learning for IMU Motion Sensors from Egocentric Videos and Text. *CoRR* abs/2210.14395 (2022). https://doi.org/10.48550/ARXIV.2210.14395 arXiv:2210.14395

[62] Seungwhan Moon, Andrea Madotto, Zhaojiang Lin, Tushar Nagarajan, Matt Smith, Shashank Jain, Chun-Fu Yeh, Prakash Murugesan, Peyman Heidari, Yue Liu, Kavya Srinet, Babak Damavandi, and Anuj Kumar. 2023. Any-MAL: An Efficient and Scalable Any-Modality Augmented Language Model. *CoRR* abs/2309.16058 (2023). https://doi.org/10.48550/ARXIV.2309.16058 arXiv:2309.16058

[63] Kabir Nagrecha and Arun Kumar. 2024. Saturn: An Optimized Data System for Multi-Large-Model Deep Learning Workloads. *Proc. VLDB Endow.* 17, 4 (mar 2024), 712–725. https://doi.org/10.14778/3636218.3636227

[64] Supun Nakandala, Yuhao Zhang, and Arun Kumar. 2020. Cerebro: A Data System for Optimized Deep Learning Model Selection. *PVLDB* 13, 11 (2020), 2159–2173.

[65] 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 *SOSP*. 1–15.

[66] Deepak Narayanan, Amar Phanishayee, Kaiyu Shi, Xie Chen, and Matei Zaharia. 2021. Memory-efficient pipeline-parallel dnn training. In *International Conference on Machine Learning*. PMLR, 7937–7947.

[67] Deepak Narayanan, Keshav Santhanam, Fiodar Kazhamiaka, Amar Phanishayee, and Matei Zaharia. 2020. Heterogeneity-Aware Cluster Scheduling Policies for Deep Learning Workloads. In *14th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2020, Virtual Event, November 4-6, 2020*. USENIX Association, 481–498. https://www.usenix.org/conference/osdi20/presentation/narayanan-deepak

[68] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, et al. 2021. Efficient large-scale language model training on GPU clusters using megatron-LM. In *SC*. ACM, 58:1–58:15.

[69] Xiaonan Nie, Xupeng Miao, Zilong Wang, Zichao Yang, Jilong Xue, Lingxiao Ma, Gang Cao, and Bin Cui. 2023. Flexmoe: Scaling large-scale sparse pre-trained model training via dynamic device placement. *Proceedings of the ACM on Management of Data* 1, 1 (2023), 1–19.

[70] OpenAI. 2023. ChatGPT. https://chat.openai.com/chat.

[71] Jian Pei, Raul Castro Fernandez, and Xiaohui Yu. 2023. Data and ai model markets: Opportunities for data and model sharing, discovery, and integration. *Proceedings of the VLDB Endowment* 16, 12 (2023), 3872–3873.

[72] Yanghua Peng, Yixin Bao, Yangrui Chen, Chuan Wu, and Chuanxiong Guo. 2018. Optimus: an efficient dynamic resource scheduler for deep learning clusters. In *Proceedings of the Thirteenth EuroSys Conference, EuroSys 2018, Porto, Portugal, April 23-26, 2018*, Rui Oliveira, Pascal Felber, and Y. Charlie Hu (Eds.). ACM, 3:1–3:14. https://doi.org/10.1145/3190508.3190517

[73] Neoklis Polyzotis, Sudip Roy, Steven Euijong Whang, and Martin Zinkevich. 2018. Data lifecycle challenges in production machine learning: a survey. *ACM SIGMOD Record* 47, 2 (2018), 17–28.

[74] Aurick Qiao, Sang Keun Choe, Suhas Jayaram Subramanya, Willie Neiswanger, Qirong Ho, Hao Zhang, Gregory R. Ganger, and Eric P. Xing. 2021. Pollux: Co-adaptive Cluster Scheduling for Goodput-Optimized Deep Learning. In *15th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2021, July 14-16, 2021*, Angela Demke Brown and Jay R. Lorch (Eds.). USENIX Association. https://www.usenix.org/conference/osdi21/presentation/qiao

[75] Alec Radford, Jong Wook Kim, Chris Hallacy, et al. 2021. Learning Transferable Visual Models From Natural Language Supervision. In *ICML*, Vol. 139. PMLR, 8748–8763.

[76] Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. 2023. Robust Speech Recognition via Large-Scale Weak Supervision. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA (Proceedings of Machine Learning Research)*, Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (Eds.), Vol. 202. PMLR, 28492–28518. https://proceedings.mlr.press/v202/radford23a.html

[77] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. 2018. Improving language understanding by generative pre-training. (2018).

[78] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog* 1, 8 (2019), 9.

[79] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *JMLR* (2020).

[80] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. 2020. ZeRO: memory optimizations toward training trillion parameter models. In *SC*. IEEE/ACM.

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

[82] Scott E. Reed, Konrad Zolna, Emilio Parisotto, Sergio Gómez Colmenarejo, Alexander Novikov, Gabriel Barth-Maron, Mai Gimenez, Yury Sulsky, Jackie Kay, Jost Tobias Springenberg, Tom Eccles, Jake Bruce, Ali Razavi, Ashley Edwards, Nicolas Heess, Yutian Chen, Raia Hadsell, Oriol Vinyals, Mahyar Bordbar, and Nando de Freitas. 2022. A Generalist Agent. *Trans. Mach. Learn. Res.* 2022 (2022). https://openreview.net/forum?id=1ikK0kHjvj

[83] Jie Ren, Samyam Rajbhandari, Reza Yazdani Aminabadi, Olatunji Ruwase, Shuangyan Yang, Minjia Zhang, Dong Li, and Yuxiong He. 2021. ZeRO-Offload: Democratizing Billion-Scale Model Training. In *2021 USENIX Annual Technical Conference, USENIX ATC 2021, July 14-16, 2021*, Irina Calciu and Geoff Kuenning (Eds.). USENIX Association, 551–564. https://www.usenix.org/conference/atc21/presentation/ren-jie

[84] Marius Schlegel and Kai-Uwe Sattler. 2023. Management of machine learning lifecycle artifacts: A survey. *ACM SIGMOD Record* 51, 4 (2023), 18–35.

[85] Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Stanford Alpaca: An Instruction-following LLaMA model. https://github.com/tatsu-lab/stanford_alpaca.

[86] Zhan Tong, Yibing Song, Jue Wang, and Limin Wang. 2022. VideoMAE: Masked Autoencoders are Data-Efficient Learners for Self-Supervised Video Pre-Training. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh (Eds.). http://papers.nips.cc/paper_files/paper/2022/hash/416f9cb3276121c42eebb86352a4354a-Abstract-Conference.html

[87] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurélien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. LLaMA: Open and Efficient Foundation Language Models. *CoRR* abs/2302.13971 (2023). https://doi.org/10.48550/ARXIV.2302.13971 arXiv:2302.13971

[88] 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, Aurélien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open Foundation and Fine-Tuned Chat Models. *CoRR* abs/2307.09288 (2023). https://doi.org/10.48550/ARXIV.2307.09288 arXiv:2307.09288

[89] Colin Unger, Zhihao Jia, Wei Wu, et al. 2022. Unity: Accelerating DNN Training Through Joint Optimization of Algebraic Transformations and Parallelization. In *OSDI*. 267–284.

[90] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *NeurIPS*. 5998–6008.

[91] Xinchen Wan, Kaiqiang Xu, Xudong Liao, Yilun Jin, Kai Chen, and Xin Jin. 2023. Scalable and Efficient Full-Graph GNN Training for Large Graphs. *Proc. ACM Manag. Data* 1, 2, Article 143 (jun 2023), 23 pages. https://doi.org/10.1145/3589288

[92] Changhan Wang, Anne Wu, Juan Pino, Alexei Baevski, Michael Auli, and Alexis Conneau. 2021. Large-Scale Self- and Semi-Supervised Learning for Speech Translation. In *Interspeech 2021, 22nd Annual Conference of the International Speech Communication Association, Brno, Czechia, 30 August - 3 September 2021*, Hynek Hermansky, Honza Cernocký, Lukás Burget, Lori Lamel, Odette Scharenborg, and Petr Motlícek (Eds.). ISCA, 2242–2246. https://doi.org/10.21437/INTERSPEECH.2021-1912

[93] Guanhua Wang, Heyang Qin, Sam Ade Jacobs, Connor Holmes, Samyam Rajbhandari, Olatunji Ruwase, Feng Yan, Lei Yang, and Yuxiong He. 2023. ZeRO++: Extremely Efficient Collective Communication for Giant Model Training. *CoRR* abs/2306.10209 (2023). https://doi.org/10.48550/ARXIV.2306.10209 arXiv:2306.10209

[94] Jianfeng Wang, Zhengyuan Yang, Xiaowei Hu, Linjie Li, Kevin Lin, Zhe Gan, Zicheng Liu, Ce Liu, and Lijuan Wang. 2022. GIT: A Generative Image-to-text Transformer for Vision and Language. *Trans. Mach. Learn. Res.* 2022 (2022). https://openreview.net/forum?id=b4tMhpN0JC

[95] Peng Wang, An Yang, Rui Men, Junyang Lin, Shuai Bai, Zhikang Li, Jianxin Ma, Chang Zhou, Jingren Zhou, and Hongxia Yang. 2022. OFA: Unifying Architectures, Tasks, and Modalities Through a Simple Sequence-to-Sequence Learning Framework. In *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA (Proceedings of Machine Learning Research)*, Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvári, Gang Niu, and Sivan Sabato (Eds.), Vol. 162. PMLR, 23318–23340. https://proceedings.mlr.press/v162/wang22al.html

[96] Qiange Wang, Yao Chen, Weng-Fai Wong, and Bingsheng He. 2023. HongTu: Scalable Full-Graph GNN Training on Multiple GPUs. *Proc. ACM Manag. Data* 1, 4, Article 246 (dec 2023), 27 pages. https://doi.org/10.1145/3626733

[97] Qiange Wang, Yanfeng Zhang, Hao Wang, Chaoyi Chen, Xiaodong Zhang, and Ge Yu. 2022. Neutronstar: distributed GNN training with hybrid dependency management. In *Proceedings of the 2022 International Conference on Management of Data*. 1301–1315.

[98] 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 *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2023, Vancouver, BC, Canada, June 17-24, 2023*. IEEE,

19175–19186. https://doi.org/10.1109/CVPR52729.2023.01838

[99] Yujie Wang, Youhe Jiang, Xupeng Miao, Fangcheng Fu, Shenhan Zhu, Xiaonan Nie, Yaofeng Tu, and Bin Cui. 2024. Improving Automatic Parallel Training via Balanced Memory Workload Optimization. *IEEE Transactions on Knowledge and Data Engineering* (2024).

[100] Zirui Wang, Jiahui Yu, Adams Wei Yu, Zihang Dai, Yulia Tsvetkov, and Yuan Cao. 2022. SimVLM: Simple Visual Language Model Pretraining with Weak Supervision. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022.* OpenReview.net. https://openreview.net/forum?id=GUrhfTuf_3

[101] Jan Weglarz. 1981. Project Scheduling with Continuously-Divisible, Doubly Constrained Resources. *Manage. Sci.* 27, 9 (sep 1981), 1040–1053. https://doi.org/10.1287/mnsc.27.9.1040

[102] Jan Weglarz. 1982. Modelling and control of dynamic resource allocation project scheduling systems. *Optimization and Control of Dynamic Operational Research Models* (1982), 105–140.

[103] Haojun Xia, Zhen Zheng, Yuchao Li, Donglin Zhuang, Zhongzhu Zhou, Xiafei Qiu, Yong Li, Wei Lin, and Shuaiwen Leon Song. [n.d.]. Flash-LLM: Enabling Cost-Effective and Highly-Efficient Large Generative Model Inference with Unstructured Sparsity. ([n. d.]).

[104] Wencong Xiao, Romil Bhardwaj, Ramachandran Ramjee, Muthian Sivathanu, Nipun Kwatra, Zhenhua Han, Pratyush Patel, Xuan Peng, Hanyu Zhao, Quanlu Zhang, Fan Yang, and Lidong Zhou. 2018. Gandiva: Introspective Cluster Scheduling for Deep Learning. In *13th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2018, Carlsbad, CA, USA, October 8-10, 2018*, Andrea C. Arpaci-Dusseau and Geoff Voelker (Eds.). USENIX Association, 595–610. https://www.usenix.org/conference/osdi18/presentation/xiao

[105] Wencong Xiao, Shiru Ren, Yong Li, Yang Zhang, Pengyang Hou, Zhi Li, Yihui Feng, Wei Lin, and Yangqing Jia. 2020. AntMan: Dynamic Scaling on GPU Clusters for Deep Learning. In *14th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2020, Virtual Event, November 4-6, 2020.* USENIX Association, 533–548. https://www.usenix.org/conference/osdi20/presentation/xiao

[106] Ziyang Xiao, Dongxiang Zhang, Zepeng Li, Sai Wu, Kian-Lee Tan, and Gang Chen. 2023. DoveDB: A Declarative and Low-Latency Video Database. *Proceedings of the VLDB Endowment* 16, 12 (2023), 3906–3909.

[107] Hu Xu, Gargi Ghosh, Po-Yao Huang, Dmytro Okhonko, Armen Aghajanyan, Florian Metze, Luke Zettlemoyer, and Christoph Feichtenhofer. 2021. VideoCLIP: Contrastive Pre-training for Zero-shot Video-Text Understanding. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (Eds.). Association for Computational Linguistics, 6787–6800. https://doi.org/10.18653/V1/2021.EMNLP-MAIN.544

[108] Jiahui Yu, Zirui Wang, Vijay Vasudevan, Legg Yeung, Mojtaba Seyedhosseini, and Yonghui Wu. 2022. CoCa: Contrastive Captioners are Image-Text Foundation Models. *Trans. Mach. Learn. Res.* 2022 (2022). https://openreview.net/forum?id=Ee277P3AYC

[109] Lu Yuan, Dongdong Chen, Yi-Ling Chen, Noel Codella, Xiyang Dai, Jianfeng Gao, Houdong Hu, Xuedong Huang, Boxin Li, Chunyuan Li, Ce Liu, Mengchen Liu, Zicheng Liu, Yumao Lu, Yu Shi, Lijuan Wang, Jianfeng Wang, Bin Xiao, Zhen Xiao, Jianwei Yang, Michael Zeng, Luowei Zhou, and Pengchuan Zhang. 2021. Florence: A New Foundation Model for Computer Vision. *CoRR* abs/2111.11432 (2021). arXiv:2111.11432 https://arxiv.org/abs/2111.11432

[110] Haoyu Zhang, Logan Stafman, Andrew Or, and Michael J. Freedman. 2017. SLAQ: quality-driven scheduling for distributed machine learning. In *Proceedings of the 2017 Symposium on Cloud Computing, SoCC 2017, Santa Clara, CA, USA, September 24-27, 2017.* ACM, 390–404. https://doi.org/10.1145/3127479.3127490

[111] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona T. Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. 2022. OPT: Open Pre-trained Transformer Language Models. *CoRR* abs/2205.01068 (2022). https://doi.org/10.48550/ARXIV.2205.01068 arXiv:2205.01068

[112] Wentao Zhang, Xupeng Miao, Yingxia Shao, Jiawei Jiang, Lei Chen, Olivier Ruas, and Bin Cui. 2020. Reliable data distillation on graph convolutional network. In *Proceedings of the 2020 ACM SIGMOD international conference on management of data.* 1399–1414.

[113] Wentao Zhang, Guochen Yan, Yu Shen, Yang Ling, Yangyu Tao, Bin Cui, and Jian Tang. 2024. NPA: Improving Large-scale Graph Neural Networks with Non-parametric Attention. In *Companion of the 2024 International Conference on Management of Data.* 414–427.

[114] Wentao Zhang, Zhi Yang, Yexin Wang, Yu Shen, Yang Li, Liang Wang, and Bin Cui. 2021. GRAIN: improving data efficiency of gra ph neural networks via diversified in fluence maximization. *Proceedings of the VLDB Endowment* 14, 11 (2021), 2473–2482.

[115] Zhen Zhang, Shuai Zheng, Yida Wang, Justin Chiu, George Karypis, Trishul Chilimbi, Mu Li, and Xin Jin. 2022. MiCS: Near-linear Scaling for Training Gigantic Model on Public. *Proceedings of the VLDB Endowment* 16, 1 (2022), 37–50.

[116] Hanyu Zhao, Zhi Yang, Yu Cheng, Chao Tian, Shiru Ren, Wencong Xiao, Man Yuan, Langshi Chen, Kaibo Liu, Yang Zhang, et al. 2023. Goldminer: Elastic scaling of training data pre-processing pipelines for deep learning. *Proceedings of the ACM on Management of Data* 1, 2 (2023), 1–25.

[117] Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Less Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, Alban Desmaison, Can Balioglu, Pritam Damania, Bernard Nguyen, Geeta Chauhan, Yuchen Hao, Ajit Mathews, and Shen Li. 2023. PyTorch FSDP: Experiences on Scaling Fully Sharded Data Parallel. *Proc. VLDB Endow.* 16, 12 (2023), 3848–3860. https://doi.org/10.14778/3611540.3611569

[118] 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 2022, Carlsbad, CA, USA, July 11-13, 2022*, Marcos K. Aguilera and Hakim Weatherspoon (Eds.). USENIX Association, 559–578. https://www.usenix.org/conference/osdi22/presentation/zheng-lianmin

[119] Deyao Zhu, Jun Chen, Xiaoqian Shen, Xiang Li, and Mohamed Elhoseiny. 2023. MiniGPT-4: Enhancing Vision-Language Understanding with Advanced Large Language Models. *CoRR* abs/2304.10592 (2023). https://doi.org/10.48550/ARXIV.2304.10592 arXiv:2304.10592

A DETAILS OF SCALABILITY ESTIMATOR

Spindle characterizes the execution time of MetaOp m over n devices, $T_m(n)$, by a generalized piecewise α-β function:

$$T_m(n) = \alpha_{m,i} + \beta_{m,i} \times c_m + \beta'_{m,i} \times w_m/n, \forall n \in [n_{i-1}, n_i], i = 1, \ldots, k$$

where k is the number of pieces, $\alpha_{m,i}$ represents the coefficient of fixed overheads (e.g., kernel launch costs), $\beta_{m,i}$ and $\beta'_{m,i}$ represent the reciprocal of execution efficiency (e.g., GPU computation speed and network bandwidth), w_m/n denotes the distributed workload of MetaOp m across n devices (e.g., computational workload), and c_m denotes the workload that doesn't scale with n (e.g., communication volume of data parallelism). Such piecewise function indicates that under varying resource scales, due to changes in the per-device workload, coefficients such as α, β and β' might differ, as the invoked kernels may vary across different workloads.

B DETAILS OF BISECTION SEARCH FOR OPTIMUM OF CONTINUOUS PROBLEM

Alg. 2 illustrates our bisection search algorithm to solve the optimum of malleable project scheduling problem, MPSP. The function $\texttt{Find\_Inverse\_Value}(T_m, \widetilde{C} = \frac{\widetilde{C}_{mid}}{L_m})$ finds the value of $T_m^{-1}(\widetilde{C})$. It first finds the closest valid allocations of MetaOp m, denoted as $\underline{n_m}$ and $\overline{n_m}$, such that $\widetilde{C} \in [T_m(\underline{n_m}), T_m(\overline{n_m})]$. It then returns

$$n_m = \frac{(\widetilde{C} - T_m(\underline{n_m})) \cdot \overline{n_m} + (T_m(\overline{n_m}) - \widetilde{C}) \cdot \underline{n_m}}{T_m(\overline{n_m}) - T_m(\underline{n_m})}, \tag{11}$$

which is the linear combination of $\underline{n_m}, \overline{n_m}$ such that $T_m(n_m) = \widetilde{C}$.

Algorithm 2: Bisection Search for MPSP

```
Input: # Devices $N$,
       linear-piecewise execution time functions $\{T_m\}_{m=1}^M$
Output: Optimum $P_{MPSP} = \{m \rightarrow \langle n_m^*, 0, L_m \rangle\}_{m=1}^M$
$\mathcal{T}_{min}, \mathcal{T}_{max} \leftarrow \{T_m(N) \cdot L_m\}_{m=1}^M, \{T_m(1) \cdot L_m\}$;
$\widetilde{C}_{low}, \widetilde{C}_{high} \leftarrow$ max $\mathcal{T}_{min}$, sum $\mathcal{T}_{max}$;
while $\widetilde{C}_{high} - \widetilde{C}_{low} > \varepsilon$ do
    $\widetilde{C}_{mid} \leftarrow (\widetilde{C}_{low} + \widetilde{C}_{high})/2$;
    $P_{MPSP} \leftarrow \{m \rightarrow$ Find_Inverse_Value$(T_m, \frac{\widetilde{C}_{mid}}{L_m}))\}_{m=1}^M$;
    if sum of allocations in $P_{MPSP} < N$ then
        $\widetilde{C}_{high} \leftarrow \widetilde{C}_{mid}$;
    else
        $\widetilde{C}_{low} \leftarrow \widetilde{C}_{mid}$;
return $P_{MPSP}$
```

C COMPARISON ON SINGLE-TASK MULTI-MODAL WORKLOAD

We also compare Spindle with baseline systems on single-task (ST) multi-modal (MT) scenario, which is a special case of MT MM training, as shown in Fig. 14. We are pleased to observe that even in the single-task scenario, Spindle outperformed SOTA systems by up to 48%. This is attributed to Spindle's fine-grained, operator-level resource allocation and scheduling, which recognize not only the inter-task workload heterogeneity but also intra-task operator workload variations — a capability beyond the reach of task-level strategies as well as SOTA systems. It's worth noting that Spindle-STMM has similar performance to Spindle on ST MM scenario, which is reasonable.

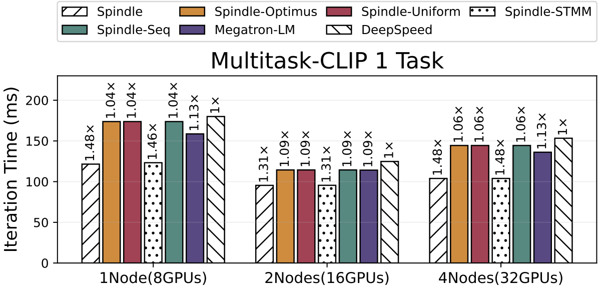


Figure 14: End-to-end performance comparison for Spindle and baseline systems on 1-task Multitask-CLIP workload.

D MEMORY CONSUMPTION

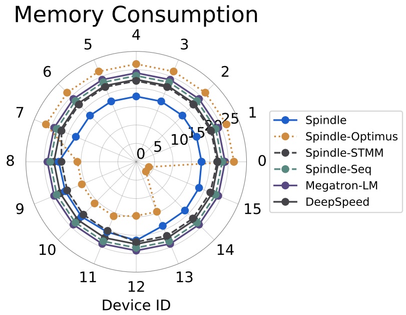


Figure 15: Memory consumption (GB) of each device in Multitask-CLIP (4 tasks, 16 GPUs). Points closer to the inner edge of the spider chart represent lower GPU peak memory usage.

We also conduct a comparative analysis of memory consumption between Spindle and the other competitors. Fig. 15 depicts the memory usage for each device in the scenario of Multitask-CLIP (4 tasks, 16 GPUs). Our findings indicate that Spindle generally exhibits lower memory consumption than SOTA systems such as Megatron-LM and DeepSpeed. This efficiency stems from Spindle's operator-level strategy and selective parameter storage feature, where only devices that activate a specific operator need to maintain its corresponding parameters, thereby minimizing redundant storage. Additionally, we've observed that task-level strategies, e.g., Spindle-Optimus, experience significant memory imbalances. This issue is even more pronounced with Spindle-Uniform, which we have omitted in Figure 15 for clarity. In contrast, Spindle maintains an excellent balance of memory consumption across devices, a success that is attributed to our device placement strategies.