Ymir: A Scheduler for Foundation Model Fine-tuning Workloads in Datacenters

Wei Gao
gaow0007@e.ntu.edu.sg
S-Lab, Nanyang Technological University
Singapore

Weiming Zhuang
weiming001@e.ntu.edu.sg
Nanyang Technological University
Singapore

Minghao Li
minghao002@e.ntu.edu.sg
Nanyang Technological University
Singapore

Peng Sun
sunpeng1@sensetime.com
Sensetime & Shanghai AI Lab
China

Yonggang Wen
ygwen@ntu.edu.sg
Nanyang Technological University
Singapore

Tianwei Zhang
tianwei.zhang@ntu.edu.sg
Nanyang Technological University
Singapore

ABSTRACT

The breakthrough of foundation models makes foundation model fine-tuning (FMF) workloads prevalent in modern GPU datacenters. However, existing schedulers tailored for model training do not consider the unique characteristics of FMs, making them inefficient in handling FMF workloads. To bridge the gap, we propose Ymir, a scheduler to improve the efficiency of FMF workloads in GPU datacenters. Ymir leverages the shared FM backbone architecture to expedite FMF workloads from two aspects: (1) Ymir investigates the task transferability among different FMF workloads and automatically merges FMF workloads with the same FM into one to improve the cluster-wide efficiency via transfer learning. (2) Ymir reuses the fine-tuning runtime of FMF workloads to reduce the significant context switch overhead. We conduct 32-GPU physical experiments and 240-GPU trace-driven simulations to validate the effectiveness of Ymir. Ymir can reduce the average job completion time by up to 4.3 × compared with existing state-of-the-art schedulers. It also promotes scheduling fairness by fully exploiting the task transferability. More supplementary materials can be found on our project website https://sites.google.com/view/ymir-project.

CCS CONCEPTS

• **Computing methodologies** → **Distributed computing methodologies**.

KEYWORDS

Foundation Model Fine-tuning, Cluster Management System

ACM Reference Format:
Wei Gao, Weiming Zhuang, Minghao Li, Peng Sun, Yonggang Wen, and Tianwei Zhang. 2024. Ymir: A Scheduler for Foundation Model Fine-tuning Workloads in Datacenters. In *Proceedings of the 38th ACM International Conference on Supercomputing (ICS '24), June 04–07, 2024, Kyoto, Japan.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3650200.3656599

ICS '24, June 04–07, 2024, Kyoto, Japan

ACM ISBN 979-8-4007-0610-3/24/06
https://doi.org/10.1145/3650200.3656599

1 INTRODUCTION

Foundation models (FMs) have pushed the state-of-the-art performance envelope across a wide range of artificial intelligence tasks [6, 20, 21, 56, 63]. An FM is a machine learning model (commonly large-scale in parameters) trained over massive data and adaptable to various downstream tasks [12]. The fine-tuned FMs have shown impressive performance in many downstream tasks [14, 64, 65], leading to an increasing of foundation model fine-tuning (FMF) workloads in public and private GPU datacenters [12, 23]. To meet the growing resource demand of FMF workloads, it is crucial to improve their efficiency from datacenter perspective.

Compared with conventional deep learning training (DLT) workloads, FMF workloads exhibit several distinct characteristics. First, FMs typically have substantial parameter sizes. Hence, *FMF workloads demand predominant GPU memory* [14, 54, 65]. Second, FMF workloads tend to require multiple GPUs for distributed execution to support large-scale models [23, 33, 54], which consequently increases the time needed to initiate the distributed execution runtime. Therefore, *FMF workloads have much higher context switch overhead than general DLT workloads* [5, 37, 75]. Third, FM users adopt a limited number of common FMs (e.g., RoBERTa [48], Vicuna [19]), as observed in [2]. Figure 1 shows the distribution of FM downloads in HuggingFace Model Hub [1]. The top 10 downloaded FMs account for 83% and 89% of the top 100 vision and language FMs, respectively. Also, existing commercial FM services (e.g., OpenAI [2]) only release a few FMs for public access. Due to the high expense of building an FM from scratch, it is cost-efficient to reuse existing FMs instead of providing diverse FMs for different tasks. Accordingly, it is common to see *many FMF workloads share the same backbone architecture in a GPU datacenter*.

Previous studies have proposed many efficient schedulers to optimize DLT workloads [16, 36, 50, 59, 62, 86]. They consider two prominent advanced practices. The first is to co-locate DLT workloads on the same GPUs to reduce the long queuing delay [16, 86]. However, the job colocation might cause out-of-memory issues for FMF workloads due to their vast GPU memory consumption. The second one is to dynamically scale up the allocated GPUs to improve the job throughput [36, 59, 62]. The frequent GPU allocation adjustment aggravates the context switch overhead and could yield significant job progress delays for FMF workloads. Some studies [7, 29] aim to reduce the context switch overhead but only

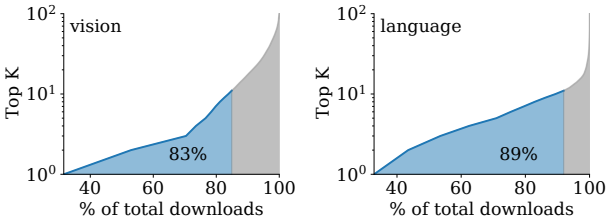


Figure 1: Proportion (x-axis) of accumulated Top-K (y-axis) FM downloads (top 10 in blue) to the top 100 downloads of vision (left) and language (right) FMs in HuggingFace [1].

for inference workloads. In summary, little systematic efforts are dedicated to accelerating the FMF workloads in GPU datacenters. Given the shared architecture of FMs, this gap could be bridged by (1) *reusing* weights across tasks to expedite fine-tuning through transfer learning; (2) *reusing* the fine-tuning runtime to reduce the context switch overhead in scenarios where FMF workloads primarily differ in model weights and task-specific datasets.

We propose Ymir, an elastic scheduling system to capitalize on these opportunities presented by the same backbone architecture to accelerate FMF workloads. Ymir consists of three key designs for FMF workload scheduling. First, we devise YmirEstimator to estimate the execution time of FMF workloads with and without task merging. Task merging indicates merging two workloads into one and subsequently fine-tuning it via transfer learning. It involves two decisions: determining which tasks to combine and selecting the appropriate transfer learning modes (illustrated in § 2.1). Specifically, YmirEstimator profiles each new workload's statistical information (e.g., loss, gradients). Based on the profiled information, YmirEstimator predicts the execution time to reach the model convergence for FMF workloads under various resource allocations and task merging scenarios.

Second, we develop YmirSched to automate the task merging and resource allocations for FMF workloads to improve cluster-wide efficiency. Task merging can expedite the model convergence, however, *randomly combining tasks might not necessarily yield speedup and could even result in a degradation of model accuracy*[1]. Ymir introduces *speedup gain* to quantify the reduction in execution time resulting from various task merging scenarios, thereby mitigating the risk of poor task merging choices. In each scheduling interval, YmirSched leverages the estimation results of YmirEstimator to compute the speedup gain. Then, YmirSched incorporates the speedup gain into the FMF workload scheduling objective, favoring task merging with higher speedup gains. Through optimizing this objective, YmirSched determines how to merge tasks and allocate GPUs for cluster-wide workloads.

Third, we implement YmirTuner to reduce the context switch overhead by reusing the fine-tuning runtime. YmirTuner comprises two modules, the *task constructor* and the *pipeline switch* to facilitate the context switch between FMF workloads. The task constructor provides a universal implementation to different FM fine-tuning algorithms [31] and allows only modification of task-specific datasets, model weights, and other hyper-parameters to perform the context switch. The *pipeline switch* pipelines the dataset preparation and parameter transfer with the model execution to hide the context switch overhead. Moreover, the pipeline switch tailors the pipeline concept to data- and pipeline-parallel FMF workloads respectively, ensuring the context switch that takes no more than one minute.

We implement Ymir atop transformers library [85], PyTorch [58] and Kubernetes [15]. It is deployed in a cluster of 8 servers and 32 Tesla V100-32GB (A100-80GB for Vicuna-7B) GPUs. We evaluate Ymir over ViT, RoBERTa and Vicuna using 9 vision, 9 language understanding, and 9 language generation datasets. Compared with existing DLT schedulers (e.g., Pollux [62], Optimus [59], Tiresias [26]), Ymir achieves 1.1 - 4.3× job completion time (JCT) speedup across various FMs. Large-scale simulation in a cluster with 240 GPUs demonstrates the scalability of Ymir. Also, comprehensive simulation experiments are conducted to disclose the impact of each component in Ymir. Our contributions are as follows:

- We present Ymir, a scheduler to exploit the shared backbone architecture to optimize FMF workloads.
- We automate the task merging and resource allocations for FMF workloads.
- We reuse the fine-tuning runtime of FMF workloads to enormously reduce the context switch overhead.
- We implement and evaluate Ymir with representative FMs and datasets to demonstrate its efficiency.

2 BACKGROUND AND MOTIVATION

We begin with an in-depth exploration of task transferability, followed by characterizing FMF workloads.

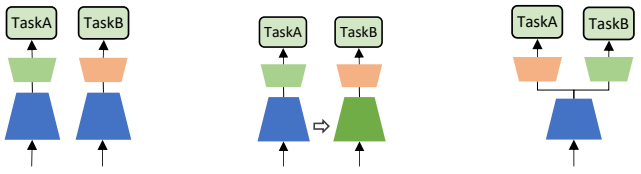
2.1 Task Transferability

As a core idea of Ymir, we provide a thorough exploration of task transferability. Task transferability refers to the ability of a model, initially trained on one task, to be used in another related but different task. In the context of FMs, downstream models sharing the same FM can expedite training convergence. Here, we discuss the transfer learning modes and benefits of task transferability.

Transfer Learning Modes. Recent theoretical [77] and empirical [4, 61, 69, 84, 87] analysis from transfer learning show that task transferability can improve the accuracy of FMs on downstream tasks. Unlike their focus on model accuracy, we consider how transfer learning expedites training convergence. By investigating existing transfer learning studies [3, 10, 22, 34, 55, 76, 76, 80], we identify three predominant transfer learning modes to accelerate FMF workloads, as illustrated in Figure 2. (1) *Normal transfer*: this is the conventional solution, where the downstream model for each task is fine-tuned on a given dataset from the pre-trained weights of the FM. (2) *Temporal transfer*: a new task B is fine-tuned from the FM fine-tuned previously on another task A. We denote this mode as $A \mapsto B$. (3) *Spatial transfer*: both task A and B are fine-tuned together using a multi-task learning scheme. We denote this as $A \| B$. §9 further discusses the extension of these modes.

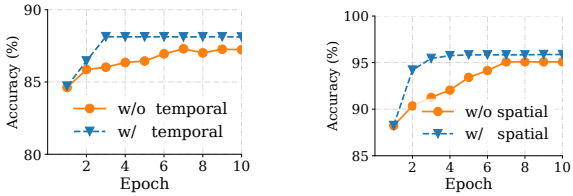
Benefits of Task Transferability. Compared to normal transfer, temporal and spatial transfer can better leverage the knowledge from other tasks [10, 80]. Figure 3 compares the validation accuracy during training in different transfer learning modes. Figure 3(a) shows that temporal transfer reduces the number of epochs to

[1]For the sake of simplicity, we use accuracy as a universal term to denote any performance evaluation metric, such as F1 score or BLEU score.



(a) Normal (b) Temporal (c) Spatial

Figure 2: Illustration of different transfer learning modes. (a) Normal transfer: the downstream model is fine-tuned from the pre-trained weight (blue trapezoid). (b) Temporal transfer: task B is fine-tuned from the FM fine-tuned previously on another task A. (c) Spatial transfer: both task A and B are fine-tuned together.



(a) STSB ↦ **QQP** (b) FOOD101 ‖ **ImageNet75**

Figure 3: Transfer learning performance: (a) QQP accuracy in temporal transfer learning on RoBERTa-Base; (b) ImageNet75 accuracy in spatial transfer learning on ViT-Base.

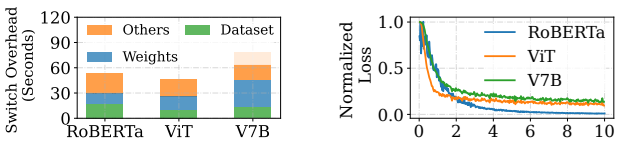
fine-tune the QQP dataset [81] by 2.3× when the FM is previously fine-tuned on the STSB dataset [81]. Similarly, Figure 3(b) shows that spatial transfer reduces the number of epochs to fine-tune the ImageNet75 dataset [68] by 2.0× when the FM is fine-tuned together on the FOOD101 dataset [13]. The speedup benefits stress the need for an automated approach to identify task combinations and transfer learning modes for cluster-wide workloads.

2.2 Characterization of FMF Workloads

FMF workloads possess some unique characteristics. We demonstrate them with three representative FMs (ViT-Base, RoBERTa-Base, Vicuna-7B) and corresponding datasets discussed in §7.1 on a server of 4 A100-80GB GPUs.

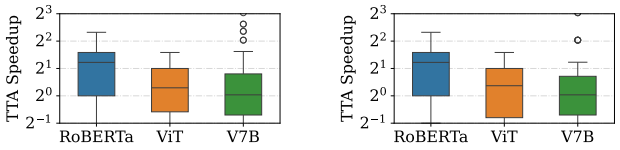
Exorbitant Context Switch Overhead. Figure 4(a) illustrates the measured context switch overhead for RoBERTa, ViT, and Vicuna-7B on STSB [81], CIFAR100 [41], and SAMSUM [25]. The overhead, mainly attributed to weight loading and dataset preparation, surpasses one minute. This high overhead hinders scaling up GPUs to improve the job throughput.

Smooth Loss Curve. Prior works [26, 50] emphasize that loss curves may not always exhibit smooth decreases, and curve fitting techniques may not extrapolate the relationship between loss and iteration. Fortunately, current ML studies [30, 39, 49] point out FMs possess well-behaved loss curves. In Figure 4(b), we use the same dataset in context switch overhead measurement and present the normalized training loss across training epochs. The training loss is normalized to the maximum loss observed throughout the training. The normalized loss exhibits relatively smooth, even in the early stages of training. Also, one study [79] provides theoretical evidence that a well-initialized model (e.g., FM) presents smooth loss curves



(a) Context Switch (b) Loss Curve

Figure 4: (a) Breakdown of context switch overhead across FMs. (b) Normalized training loss (y-axis) versus epoch (x-axis) across various FMs.



(a) Temporal Transfer (b) Spatial Transfer

Figure 5: The TTA speedup box plot of (a) temporal transfer and (b) spatial transfer across various FMs.

for downstream tasks. Followed by prior studies [59, 93], we can adopt curve fitting techniques to predict model convergence.

Pervasive Task Transferability. Task transferability provides new opportunities to optimize FMF workloads in a datacenter: workloads sharing the same FM can be combined to enhance the performance and cluster efficiency, even for different tasks with different datasets. Task transferability manifests pervasive across diverse FMs and tasks. For FMs, previous studies [12, 14, 65] emphasize their remarkable ability to adapt to various tasks. FM developers strategically optimize their models across a spectrum of tasks, enhancing the generalization and transferability of FMs. Consequently, robust task transferability is a common phenomenon within FMs. For tasks, recent ML studies [3, 55, 73, 80, 84] have analyzed transferability between numerous language and vision tasks. Their findings reveal that over 50% of task combinations can benefit from spatial or temporal transfer learning. To present this, we compute the Time-To-Accuracy (TTA) metric, which is defined as the time required to achieve the target accuracy on a task. We utilize the targeted accuracy of our evaulated FMF tasks, and measure the TTA of various task combinations for different FMs. In Figure 5(a), we illustrate the box plot of relative TTA speedup for temporal and spatial transfer, in comparison to normal transfer. Both temporal and spatial transfer can speedup FMF workloads up to 10 ×. Furthermore, more than half of the task combinations exhibit positive speedup (≥ 1). This underscores selecting optimal task combinations and transfer learning modes can expedite FMF workloads significantly.

Indeed, users have a desire to share task-specific model parameters with the ML community. Every day, hundreds of new task-specific models built upon representative FMs are released on HuggingFace [1]. ModelKeeper [42] and Sommelier [28] harness the potential of model sharing to expedite model training in GPU datacenters. Naturally, task transferability opens a new venue to expedite training progress for cluster-wide FMF workloads.

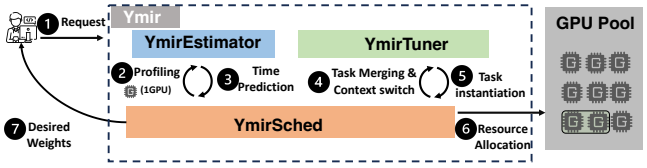


Figure 6: Workflow of Ymir. It comprises three key designs: (1) YmirEstimator estimates the execution time of FMF workloads; (2) YmirSched determines the task merging scenarios and resource allocations; (3) YmirTuner provides efficient context switch for FMF workloads.

3 YMIR OVERVIEW

We introduce Ymir, a scheduler for FMF workloads to unleash the potential of task transferability of FMs and improve the cluster-wide efficiency and scheduling fairness. We discuss the system assumptions and workflow below.

System Assumptions. We make several assumptions about our system. (1) We assume all FMF workloads share the same FM backbone in the GPU datacenter, as discussed in §1. We discuss extending Ymir to multiple FMs in §9. (2) A task is denoted as a (dataset, objective function) pair. The same dataset might be employed with different objective functions, which could be considered various tasks. (3) We focus on the widely adopted data-parallel and pipeline-parallel mechanisms in FMF workloads. Other parallelism schemes can be easily integrated into Ymir.

System Workflow. Ymir contains three key components: YmirEstimator is responsible for predicting the execution time of FMF workloads with different task merging scenarios, including task combinations and transfer learning modes; YmirSched automates the efficient task merging and resource allocations for cluster-wide workloads; YmirTuner improves the efficiency of FMF workloads with lightweight context switch mechanisms.

Figure 6 shows the workflow of Ymir. First, a user submits an FMF request to Ymir in a YAML format. The YAML file specifies a list of system parameters, as presented in Table 1. (❶). Then, YmirEstimator demands resources (e.g., 1 GPU) for each new workload from YmirSched for profiling, collecting relevant statistical information (e.g., loss, gradient) (❷). YmirEstimator utilizes profiling results to perform time prediction for each new workload and send prediction results to YmirSched (❸). Second, YmirSched decides how to merge tasks and makes the resource (re-)allocations for cluster-wide workloads (❹). Third, YmirTuner receives task merging decisions and instantiates the FMF workloads based on transfer learning modes and other hyperparameters (❺). It also pipelines the context switch to reduce corresponding overhead. YmirSched places FMF workloads on appropriate GPUs (❻). Lastly, Ymir returns the desired model weights to the user when the FMF workload is finished (❼).

4 YMIRESTIMATOR

YmirEstimator consists of three components to estimate the execution time of FMF workloads over various task merging scenarios with profiling results, as shown in Figure 7. First, *transferability estimator* computes the transferability score and predicts the transfer gain (defined in Eqn. 1) between the new workload and other FMF workloads. Then, *iteration estimator* uses the transfer gain to predict the number of iterations (defined in Eqn. 2) that reach the target accuracy in different learning modes. Last, *time estimator* estimates the execution time by multiplying the number of iterations with the time estimated for each iteration under any resource allocations (defined in Eqn. 5). The estimation process is performed only once for each new workload, significantly reducing the computational overhead and improving efficiency. We emphasize that the YmirEstimator's design is highly modularized, and its components can be replaced with other techniques that perform the same functions. Below, we present the technical details of each component.

Table 1: Description of System Parameters in Ymir.

Parameters	Description
`Model`	The model name.
`Dataset`	A path (e.g., AWS S3) where training and evaluation samples are stored.
`Hyperparam`	batch size, learning rate, optimizer, etc.
`Target`	The job completion criteria, including a maximum number of iterations and an accuracy target[2].
`Sharing`	Whether to share parameters with other tasks.
`Pipeline`	Whether to adopt pipeline parallelism.

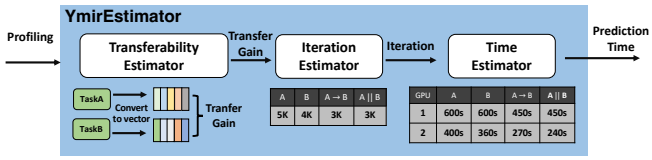


Figure 7: The workflow of YmirEstimator. It contains three components: (1) The *transferability estimator* estimates the transfer gain between new requests and other FMF requests; (2) The *iteration estimator* estimates the number of iterations needed to reach the target accuracy in different transfer learning modes; (3) The *time estimator* estimates the execution time of new FMF requests.

Table 2: Prediction accuracy of YmirEstimator

Model	Transferability			Iteration (APE)		Iteration-Transfer (APE)	
	Pearson's r ↑	MAPE (%) ↓	ACC (%) ↑	Max (%) ↓	Mean (%) ↓	Max (%) ↓	Mean (%) ↓
ViT-Base	0.439	15.53	97.2%	8.13	5.69	26.8	15.35
RoBERTa-Base	0.791	16.76	98.6%	24.75	8.27	31.61	13.67
Vicuna-7B	0.568	18.05	98.6%	22.3	11.3	32.9	11.07

4.1 Transferability Estimator

This component estimates the *transfer gain* for each joint transfer learning mode (§ 4). Given two tasks A and B, the transfer gain from A to B is calculated as follows:

$$G_{A,B} = \frac{P_{A,B} - P_B}{P_B}, \tag{1}$$

$P_{A,B}$ is the performance (e.g., accuracy) of B when jointly fine-tuned with A, while P_B is the performance of B when fine-tuned alone. If

joint fine-tuning improves the performance of B, $G_{A,B}$ is positive. Otherwise, it is negative or zero.

A straightforward way to obtain the transfer gain is to fine-tune the tasks in different learning modes, measure the performance, and compute $G_{A,B}$ with Eqn. 1. This is computationally expensive and impractical in workload scheduling. Instead, inspired by previous works [3, 10, 55, 80], we adopt statistical information and ML techniques to predict the transfer gain. As Ymir requires the least computation overhead and satisfactory prediction accuracy, we empirically find that Task2Vec [3] is the most suitable technique (discussed in §7.5). Its underlying principle is that tasks with high gradient similarity exhibit high transferability. We make two modifications over Task2Vec to adapt to our scenario. First, Task2Vec only considers the temporal transfer learning and provides the corresponding transferability score $S(A, B)$ from tasks A to B. We extend this metric to spatial transfer learning: we compute the bidirectional transferability scores $S(A, B)$ and $S(B, A)$ and take their average as the final transferability score for spatial transfer learning.

Second, we take the transferability score $S(A, B)$ as input to predict the transfer gain $G_{A,B}$. Table 2 (**Transferability**) shows the Pearson correlation between $S(A, B)$ and $G_{A,B}$ for different FMs. The high linear correlation between these two metrics suggests the feasibility of using linear regression to predict the transfer gain from the transferability score.

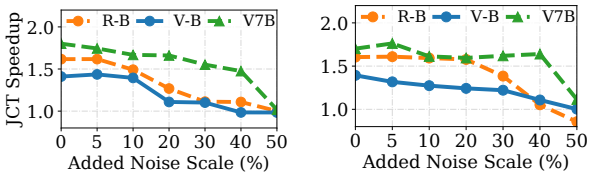
Error Analysis. In Table 2 (**Transferability**), we choose two metrics to evaluate *transferability estimator* by considering various task combinations across different transfer learning modes: (1) The mean absolute percentage error (MAPE) between the transfer gain and estimated gain using the transferability score; (2) We categorize the transfer gain estimation into two classes: positive ($G_{A,B} \geq 0$) and negative ($G_{A,B} < 0$) transfer, and then report the classification accuracy (ACC). The low MAPE and high accuracy across different FMs indicate that *transferability estimator* is a general and practical approach for estimating the transfer gain.

Sensitivity Analysis. We further analyze the impact of *transferability estimator*'s errors on the JCT speedup performance brought by *task merger* (as discussed in §5.1). Specifically, we add random noise with the scale following a uniform distribution over $[-1, 1]$ on the prediction results of *transferability estimator*. Figure 8 (a) presents the JCT speedup compared to the case without *task merger*. Even when the added noise scale is up to 40%, the JCT speedup brought by *task merger* is still larger than 1. Despite potential deviations in estimation accuracy, the overall performance improvement remains satisfactory.

4.2 Iteration Estimator

This component estimates the number of iterations required for joint fine-tuning to reach (or exceed) the same validation accuracy as the normal transfer. It estimates the training loss curve using the predicted transfer gain $G_{A,B}$ for different joint transfer learning modes. Then, following previous works [9, 93], it identifies the minimum number of iterations that makes the training converge. Formally, for task i, the number of iterations K_i is estimated as follows:

$$K_i = \arg\min_k \mathbf{1}(\mathcal{L}_i(k) - \mathcal{L}_i(k+1) \leq 0.001), \tag{2}$$



(a) Transferability Estimator (b) Iteration Estimator

Figure 8: Sensitivity analysis of Transferability Estimator (a) and Iteration Estimator (b) on JCT speedup between w/ and w/o task merger.

where $\mathbf{1}$ is the indicator function and $\mathcal{L}_i(k)$ is the training loss value at the k^{th} training step.

It is challenging to obtain the training loss $\mathcal{L}_i(k)$ efficiently. The smoothing loss curve of FMF workloads motivates us to adopt a curve function proposed by Optimus [59] to characterize the job progress and training loss for DLT workloads. FMF workloads commonly use the Adam optimizer [40], which has a faster convergence rate than SGD. We introduce an additional second-order term k^2 to characterize better the job progress and normalized training loss of FMF workloads:

$$\mathcal{L}_i(k) = \frac{1}{\beta_{i,3} \cdot k^2 + \beta_{i,2} \cdot k + \beta_{i,1}} + \beta_{i,0}, \tag{3}$$

where $\beta_{i,3}$, $\beta_{i,2}$, $\beta_{i,1}$, and $\beta_{i,0}$ are learnable non-negative coefficients. We empirically observe that our adopted curve-fitting technique performs better than Optimus. Also, the user can provide appropriate fitting functions based on their experience.

We can use loss traces during profiling to fit Eqn. 3 and obtain a general set of $\beta_{i,3}$, $\beta_{i,2}$, $\beta_{i,1}$, and $\beta_{i,0}$ for each task i. Specifically, we assume the joint transfer learning task follows a similar training loss convergence pattern as normal transfer, as investigated by previous studies [39, 49]. This is empirically validated in Table 2 (**Iteration-Transfer**) as well. Then, we use the estimated transfer gain $G_{A,B}$ to derive the normalized loss curve as $\mathcal{L}_{A,B}(k) = \frac{\mathcal{L}_B(k)}{(1+G_{A,B})}$ for either spatial or temporal transfer learning from task A to B. A higher $G_{A,B}$ can reduce the number of training iterations using spatial or temporal transfer learning. Lastly, we use this loss to estimate K_B. For temporal transfer learning from tasks A to B, we calculate $K_B^{A \mapsto B}$ with $\mathcal{L}_B(k)$ with Eqn. 2. For spatial transfer learning, the estimated number of iterations is

$$K_A^{A\|B} = K_B^{A\|B} = \max\left(\frac{K_A M_A}{D_A}, \frac{K_B M_B}{D_B}\right) \cdot \frac{D_A + D_B}{M_A + M_B}, \tag{4}$$

where for a task i, K_i is obtained from Eqn. 2 with $\mathcal{L}_i(k)$, M_i is the global batch size, and D_i is the training set size.

Error Analysis. We report the mean/max absolute percentage error (APE) for different FMs with normal transfer in the fifth and sixth columns of Table 2 (**Iteration**). We use transfer gain to predict corresponding training iterations for both temporal and spatial transfer learning. The prediction error of *iteration estimator* for both temporal and spatial transfer learning modes are presented in the seventh and eighth columns of Table 2 (**Iteration-Transfer**). The estimation error of **Iteration-Transfer** is typically larger than **Iteration**, resulting from the accumulated estimation error brought by *transferability estimator*. The maximal prediction APE is within

an acceptable range (40%). Our *iteration estimator* performs well in estimating the number of iterations needed.

Sensitivity Analysis. We use the similar technique as *transferability estimator* to analyze the sensitivity of *iteration estimator*'s error in Figure 8 (b). Our findings indicate that the JCT speedup gradually decreases with the increased noise scale. When the noise scale is up to 40%, *task merger* still decreases the JCT. Moreover, Vicuna can benefit from the added noises to a certain degree, which might result from the internal prediction error of our *iteration estimator*.

4.3 Time Estimator

After obtaining K_i from *iteration estimator*, the next step is to attain the job speed under a given resource allocation. Considering the fixed backbone architecture of FMF workloads, our *time estimator* provides accurate job speed via offline profiling. We utilize a simple yet effective method called lookup table (LUT). It accepts resource allocations and training configurations as input and returns the job speed of each training iteration. In particular, LUT constructs a map $\mathrm{S}(a, \text{cfgs})$, where a is the number of GPUs assigned to the job and cfgs are the training configurations. Given such information, we use LUT to obtain the execution time of the task i as follows:

$$T_{i,a} = \mathrm{S}(a, \text{cfgs}) \cdot K_i. \tag{5}$$

The execution time of temporal transfer learning from tasks B to A and spatial transfer learning is denoted as $T_{A \mapsto B,a}$ and $T_{A \| B,a}$, respectively. Their main difference is reflected in the calculation of K_i in §4.2. Specifically, cfgs include $\{s, m, amp, \ell, ckpt, pipeline\}$, where a is the number of GPUs assigned to the workload, s is the number of gradient accumulation steps, m is the local batch size per device, ℓ is the number of frozen layers during fine-tuning, amp is a boolean value for automatic mixed-precision training, $ckpt$ is a boolean value for the gradient checkpoint, and $pipeline$ is a boolean value for parameter-efficient transfer learning. pipeline also implies the selection of data-parallelism or pipeline-parallelism, which will be discussed in § 6.1. Building the Look-Up Table (LUT) offline poses a great challenge due to a large number of potential configurations. We reduce the number of configurations needed to profile and implement the offline profiling within 5 hours per FM. We continuously update the LUT online to minimize the gap between LUT and practical scenarios.

Estimation Error Handling of YmirEstimator. From the sensitivity analysis of *transferability estimator* and *iteration estimator*, Ymir achieves a satisfactory speedup, even when the prediction of our estimators is not accurate enough. This highlights the robustness of our system. However, it is imperative to proactively address potential estimation errors of YmirEstimator, as they could undermine model accuracy and impede training progress. We monitor the accuracy changes of merged tasks to prevent these issues. For temporal transfer $A \mapsto B$, we assess the validation accuracy of task B when fine-tuning task A during the accuracy evaluation stage. If it fails to enhance the accuracy of task B in the first two epochs, we disable the temporal transfer and schedule both tasks independently. For spatial transfer, if the accuracy of either task A or B does not improve in the first two epochs, we decouple the spatial transfer and schedule both tasks separately.

Overhead Analysis of YmirEstimator. The overhead of YmirEstimator consists of the workload profiling and the ML model estimation in the middle scheduling interval. The workload profiling overhead has been discussed in §7.5. The maximal ML model estimation overhead for ViT-B, RoberTa-B, and Vicuna-7B is 7.8, 8.6, and 11.2 seconds respectively. Overall, the estimation overhead is acceptable compared to the FMF workload execution time (tens of minutes).

5 YMIRSCHED

In YmirSched, we first introduce the *task merger* determines task combinations and transfer learning modes. Next, we discuss how YmirSched addresses special cases and scalability issues.

5.1 Task Merger

Fairness objective. Achieving resource allocation fairness in workload scheduling is critical to incentivizing users to share GPU resources [50, 62]. Fairness aims to assign GPU resources evenly across all FMF workloads. Formally, given a set of N tasks $\mathcal{J} = \{j_1, j_2, j_3 ... j_N\}$ and R available GPUs, the number of allocated GPUs to each job a belongs to a given set $\mathbb{A} = \{0, 1, 2, 3, 4m \mid m \in \mathbb{Z}^+\}$. A fair share of GPU resources is $\bar{a} = \lceil R/N \rceil$. From § 4.3, we obtain the execution time $T_{i,a}$ of task j_i assigned with a GPUs. YmirSched optimizes the following objective:

$$\min_{\mathbf{X}} \left(\sum_{i=1}^{N} \sum_{a \in \mathbb{A}} x_{i,a} \cdot \left(\frac{T_{i,\bar{a}}}{T_{i,a}} \right) \right), \tag{6}$$

where $x_{i,a}$ is an element of a binary matrix $\mathbf{X} \in \mathbb{B}^{N \times R}$, indicating whether j_i is allocated with a GPUs; $T_{i,\bar{a}}/T_{i,a}$ measures the reciprocal of the job speedup brought by elastic training. The definition of this objective is inspired from previous fairness schedulers [50, 62]. It minimizes the sum of the slowdown for each job (i.e., maximizes the speedup of each job) and enforces each workload to share a similar job speedup/slowdown.

Transfer gain and resource allocation. YmirSched considers maximizing the speedup benefits of task merging to determine the transfer learning modes and resource allocations. Combining two FMF workloads with different transfer gains or allocated resources favors different optimal modes. For a more in-depth exploration of preferences regarding transfer learning modes, please refer to the detailed discussion in §7.2.

Optimization problem. Considering the fairness and impact of the transfer learning modes, YmirSched introduces the *task merger*

to optimize the following objective:

$$\min_{\mathbf{X},\mathbf{Y},\mathbf{Z}} \underbrace{\sum_{i=1}^{N}\sum_{a\in\mathbb{A}} x_{i,a}\cdot\frac{T_{i,\bar{a}}}{T_{i,a}}}_{\text{normal transfer}} + \underbrace{\sum_{i=1}^{N}\sum_{k=1,k\neq i}^{N}\sum_{a\in\mathbb{A}} y_{i,k,a}\cdot\frac{1}{\texttt{TranWt}(i,k,\mapsto,a)}\cdot\frac{2T_{i\mapsto k,2\bar{a}}}{T_{i\mapsto k,a}}}_{\text{temporal transfer}} + \underbrace{\sum_{i=1}^{N}\sum_{k=1,k\neq i}^{N}\sum_{a\in\mathbb{A}} z_{i,k,a}\cdot\frac{1}{\texttt{TranWt}(i,k,\|,a)}\cdot\frac{2T_{i\|k,2\bar{a}}}{T_{i\|k,a}}}_{\text{spatial transfer}}, \quad (7)$$

subject to:

$$x_{i,a}, y_{i,k,a}, z_{i,k,a} \in \{0,1\}, \forall a \in \mathbb{A}, \forall i,k \in \mathbb{Z}(N), \quad (8)$$

$$\sum_{a\in\mathbb{A}} x_{i,a} = 1, \sum_{a\in\mathbb{A}} y_{i,k,a} = 1, \sum_{a\in\mathbb{A}} z_{i,k,a} = 1, \forall i,k \in \mathbb{Z}(N), \quad (9)$$

$$\sum_{a\in\mathbb{A}\setminus\{0\}} x_{i,a} + \sum_{k=1,k\neq i}^{N} (y_{i,k} + y_{k,i} + z_{i,k} + z_{k,i}) \leq 1, \forall i \in \mathbb{Z}(N), \quad (10)$$

$$\sum_{i=1}^{N}\sum_{a\in\mathbb{A}} a\cdot x_{i,a} + \sum_{k=1,k\neq i}^{N} a\cdot(y_{i,k} + y_{k,i} + z_{i,k}) \leq R \quad (11)$$

where $\mathbb{Z}(N) = \{1,\ldots,N\}$, $x_{i,a}$ is a binary variable to denote whether to allocate a GPUs to j_i, $y_{i,k,a}$ is a binary variable to denote whether to allocate a GPUs and use temporal transfer learning from j_i to j_k, and $z_{i,k,a}$ is a binary variable to denote whether to allocate a GPUs and use spatial transfer learning between j_i and j_k.[3] Note that we use $2T_{i\mapsto k,2\bar{a}}$ ($2T_{i\|k,2\bar{a}}$) to compute the slowdown of the merged task. Constraint (9) ensures at most one allocation policy for each job. Constraint (10) guarantees no overlap between individual workload and merged workload in resource allocations. Constraint (11) ensures the total number of allocated GPUs does not exceed the resource capacity.

In Objective (7), we introduce `TranWt` to favor the task combinations and transfer learning modes that lead to more significant JCT speedup. In particular, we quantify the speedup of temporal and spatial transfer learning modes compared to normal training as $\texttt{TranWt}(A,B,\mapsto,a)$ and $\texttt{TranWt}(A,B,\|,a)$, respectively. For a given resource allocation a, these two metrics can be formulated as follows:

$$\texttt{TranWt}(A,B,\mapsto,a) = \frac{2\min(T_{A,a},T_{B,a}) + \max(T_{A,a},T_{B,a})}{2T_{A,a} + T_{A\mapsto B,a}}, \quad (12)$$

$$\texttt{TranWt}(A,B,\|,a) = \frac{2\min(T_{A,a},T_{B,a}) + \max(T_{A,a},T_{B,a})}{2T_{A\|B,a}}. \quad (13)$$

The numerator of each equation is the JCT of executing A and B with the Shortest Remaining Time First (SRTF) scheduling algorithm. The denominator of Eqn. 12 is the JCT of executing A and then B with temporal transfer learning; the denominator of Eqn. 13 is the JCT of executing A and B with spatial transfer learning. We compute $\texttt{TranWt}(B,A,\mapsto,a)$ similarly as Eqn. 12. For spatial transfer learning, $\texttt{TranWt}(A,B,\|,a)$ and $\texttt{TranWt}(B,A,\|,a)$ are numerically equal.

Using the Integer Linear Programming (ILP) solver, we obtain a solution to Eqn. 7, i.e., the resources allocated to each job and the transfer learning mode. Then, we pack each workload with as few nodes as possible to minimize the communication overhead.

[3] In practice, spatial transfer learning can only be applied to jobs with zero progress in that they share the same backbone weights.

5.2 Discussion

Worklod Profiling. YMIRSched needs to provide profiling resources for new workloads to gather statistical information. YMIRSched does not take into account joint fine-tuning for profiling workloads. Additionally, the allowable resource allocations for profiling workloads are one GPU for data-parallel workloads and four GPUs for pipeline-parallel workloads.

Pipeline Workloads Scheduling. Following typical resource request practice of pipeline-parallel workloads [35, 54], we restrict the resource allocation set as $\mathbb{A} = \{4m | m \in \mathbb{N}\}$, reserving entire GPU servers for each pipeline-parallel workloads. The job throughput of the pipeline-parallel workloads depends upon some configurations (e.g., model partition, the number of pipelines). Given the fixed backbone architecture, We profile these configurations offline and use them during model execution.

Scalability. In solving the above optimization problem, the scalability of YMIRSched is related to the square of the number of jobs. In practice, YMIRSched can quickly filter out unnecessary task combinations (e.g., $\texttt{TranWt} < 1$) to reduce the number of optimization variables. We provide further investigations in §7.2 to validate its scalability.

Machine Failure Handling. In the event of machine failures, the default epoch-based checkpoint allows us to resume from the latest checkpoint. Moreover, we maintain the transfer learning modes and restore the execution of FMF workloads until the next scheduling interval (at most 120 seconds). The efficiency might be undermined slightly in this scenario. We leave it as our future work.

6 YMIRTUNER

We introduce the *task constructor* and *pipeline switch* to reuse the fine-tuning runtime of FMF workloads to improve efficiency and mitigate the context switch overhead.

6.1 Task Constructor

Task constructor has two main functions. First, it supports three transfer learning modes as illustrated in Figure 2. The only difference between normal and temporal transfer learning is the path storing the initialized weights. For spatial transfer learning, *task constructor* adopts the same hyperparameters (e.g., learning rate, batch size.) to fine-tune task-specific inputs. The dataloader adopts the annealed sampling [47] to yield the inputs.

Second, *task constructor* decides the configurations of data and pipeline parallelism for high throughput. It adopts Parameter-Efficient Transfer Learning (PETL), a common practice in fine-tuning FMs to enable data parallelism for FMF workloads. With PETL, we can fine-tune a small portion of task-specific parameters instead of the entire model to reduce GPU memory consumption. As

such, we can also execute most fine-tuning tasks in a data-parallel manner and take advantage of its benefits, e.g., elastic training and performance modeling. There are different types of PETL architectures [32, 33], and we choose a unified architecture proposed in [31]. Particularly, *task constructor* decides the steps of gradient accumulation s to alleviate the GPU memory consumption in the case of a large batch size. Additionally, it supports pipeline parallelism when requested by users. It profiles the optimal pipeline stage and model partition offline and adopts them on demand. In evaluation, only fine-tuning Vicuna-7B on ROC [53] dataset assumes the pipeline parallelism for better throughput in consideration of large batch size (96) and model parameter size (7 billion). § 5.2 have discussed scheduling pipeline-parallel workloads.

6.2 Pipeline Switch

The context switch between FMF workloads exacerbates the scheduling flexibility and delays the job progress, especially for short-term ones. Based on the analysis in §2.2, we consider hiding the overhead of parameter load and data loader preparation for pipeline- and data-parallel workloads.

First, we hide the latency between loading weights and launching the CUDA stream to execute gradient computation for pipeline-parallel workloads. We propose to pipeline the gradient computation of task A and parameter transmission of task B, as illustrated in Figure 9. Each machine maintains the entire model structure and partial model parameters. Both A and B adopt the pipeline parallelism on a 4-GPU machine, and the FM is partitioned into four parts. For naming conventions, we use the subscript of 1-4 to denote the partition, and the superscript f, b, and t to represent the forward propagation, backward propagation, and parameter transmission. When the context switch happens between A and B, we overlap the parameter store of A and the parameter load of B across machines. We also pipeline the gradient computation and parameter transmission as much as possible in each machine. To this end, we require B to compute from machines 4 to 1. On machine 4, after completing A_4^b, we save the partial parameters of A subsequently. Next, the partial parameters of B is loaded into machine 4, and B_1^f starts execution. Note that our pipeline schemes differ from PipeSwitch [7] in two aspects: (1) we consider the pipeline parallelism while PipeSwitch only focuses on single-GPU tasks; (2) the reverse direction of the model execution between task A (machine 1 to 4) and task B (machine 4 to 1) facilitates hiding the latency between parameter store of task A and parameter load of $taskB$, which PipeSwitch cannot achieve.

Second, we hide the latency between dataloader preparation and model execution for data-parallel workloads. Dataloader preparation mainly involves spawning multiple processes for efficient data loading and preprocessing. It does not request GPU resources and brings less system overhead for the main process. Hence, we implement a simple handler for user signals (e.g., SIGUSR1 in UNIX) to accomplish on-demand dataloader preparation ahead of time. For the scheduling interval, YmirSched will notify the YmirTuner to prepare the dataloader for preempted tasks 30 seconds ahead.

We emphasize that the benefits of our proposed pipeline switch depend upon the PCIe bandwidth. With the increased bandwidth, the overhead of context switching diminishes, resulting in shorter execution time. Consequently, the ratio of context switch overhead over computation time decreases, making computation time the new bottleneck. Moreover, the pipeline switch alleviates the context switch overhead, thereby providing a way to enhance hardware utilization rates.

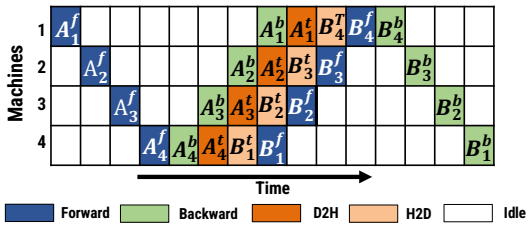


Figure 9: Pipeline model propagation and parameter transmission. D2H indicates saving parameters from device (GPU) to host (CPU). H2D indicates loading parameters from host (CPU) to device (GPU).

7 EVALUATION

We first present the setup of our evaluation experiments in§7.1. Then, we perform physical and simulation experiments for three FMs to validate the effectiveness and scalability of Ymir in §7.2. Next, we analyze the impact of several key system components in §7.3-7.5

7.1 Experimental Setup

Implementation. We implement YmirEstimator and YmirTunner on transformers 2.4.1 [85] and PyTorch 1.7 [58], and YmirSched on Kubernetes 1.18.2 [15]. The implementation only consists of 5,967 lines of Python code.

Cluster testbed. We conduct physical experiments in a cluster of 8 GPU nodes. Each node has 4 × Tesla V100 SXM2 32 GB, 1 × 200 Gbs HDR InfiniBand, 64 CPU cores, and 256 GB memory, connected via PCIe-III. Particularly, we evaluate Vicuna-7B on GPU servers containing A100 SXM4 80GB GPUs due to its high GPU memory consumption. Our physical implementation is built upon Pollux [62]. We use CephFS 14.2.8 to store checkpoints. Additionally, we set the cluster capacity as 60 4-GPU nodes in our simulation to demonstrate the scalability of Ymir.

FMF tasks. We evaluate Ymir on 9 vision datasets, 9 language understanding, and 9 language generation for ViT-Base, RoBERTa-Base, and Vicuna-7B, respectively. We have conducted a hyper-parameter sweep to search each task's optimal learning rate and batch size. As we evaluate Ymir on 27 FMF different tasks, we present a full suite of FMF tasks, including hyperparameters and target validation metrics, in Part A of our project website.

Workloads. Our evaluation workloads are sampled from a trace from Shanghai AI Lab where users submit extensive jobs related to FMs. For physical evaluation, we sample 120 - 240 jobs for different FMs and construct one workload accounting for the expensive cost. For large-scale simulation experiments, we sample 1500 - 3000 jobs for different FMs and construct three workloads for evaluation. The number of sampled workloads is based on the model scale to match the GPU time usage of our adopted trace. We follow Pollux's

Table 3: JCT diff. between simulator and physical implementations.

Scheduler	Ymir	Optimus	Pollux	Tiresias
Average JCT Diff (%)	8.77	4.37	3.24	5.40
Tail JCT Diff (%)	10.82	0.74	6.03	3.96

Table 4: Accuracy improvement over normal transfer.

Foundation Models	Max		Min		Avg	
	Temporal	Spatial	Temporal	Spatial	Temporal	Spatial
ViT-B	2.12%	2.4%	0.24%	0.37%	1.11%	0.95%
RoBERTa-B	3.72%	1.51%	0.0%	0.9%	0.85%	1.21%
Vicuna-7B	7.59%	68.82%	2.75%	0.86%	3.72%	14.74%

workload generator to synthesize our evaluation workloads. Specifically, we categorize FMF tasks based on their GPU time and set the probability of generating these jobs on their scales. The detailed workload synthesis can be found in Part A of our project website.

Baselines. In the physical experiments, we compare Ymir with three schedulers, Tiresias [26], Optimus [59] and Pollux [62]. They are all implemented atop Pollux's official implementation. Tiresias fixes the number of workers for each workload. Similar to Ymir, Optimus and Pollux dynamically change the number of workers to maximize the cluster-wide performance. However, due to the sensitivity of FMF workloads toward batch size [44, 72], we disable GNS [52] to tune the batch size for Pollux throughout the training[4]. Besides, we also compare with Themis [50] to show how Ymir balances fairness and efficiency. We also add preemptive SRTF to reinforce the effectiveness of Ymir. Following Pollux's practice, we construct our simulator, detailed in Part A of our project website.

We set the lease term interval of Themis as 600 seconds. The scheduling interval of Pollux and Optimus is set as 300 seconds for the exorbitant context switch overhead. The scheduling interval of Tiresias, Themis, and SRTF is set as 120 seconds because of their infrequent resource re-allocations. Thanks to the pipeline switch, Ymir adopts a short scheduling interval of 120 seconds. To show the generality and applicability of Ymir, we choose three representative FMs (ViT-Base, RoBERTa-Base, and Vicuna-7B). We evaluate them on 9 vision datasets, 9 language understanding datasets, and 9 language generation datasets. More detailed descriptions of datasets are available in Part A of our project website.

Simulator fidelity. To validate the fidelity of our simulator, we measure the difference of average JCT and tail JCT between the simulation and physical experiments in Table 3. The average JCT gap is within 10%, and tail JCT difference is around 10%. This shows our simulator can provide reliable and accurate evaluation results. Without special explanation, we use our simulator in §7.3-7.5.

7.2 End-to-end Performance

Physical evaluation results. We adopt average JCT and 99% tail JCT to measure the efficiency of Ymir. Figure 10 presents the performance of Ymir and baselines over different FMs normalized to Ymir. Additionally, Figure 10 shows the average and tail JCT (seconds) of Ymir. Ymir can reduce 1.11 - 4.34× average JCT, and 0.89 - 3.56× tail JCT compared to baselines. Unlike discussed in [62], Pollux and Optimus do not outperform Tiresias considerably for language FMs. The frequent resource re-allocations might delay the job progress and degrade the performance benefit of elastic training. Besides, Vicuna attains better performance improvements than smaller FMs, as they facilitate task transferability and perform well in model generalization and transferability. §7.5 provides empirical evidence that Vicuna enjoys the most JCT speedup brought by *task merger*.

[4]GNS leads to NAN issues when fine-tuning Vicuna on COQAQG [67].

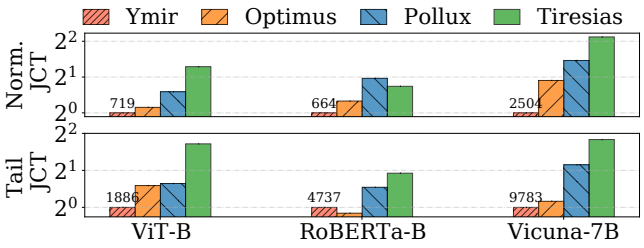


Figure 10: Physical evaluation results over different FMs.

Table 5: The fractions of tasks participating in different transfer modes in the physical experiment.

Mode	ViT-B	RoBERTa-B	Vicuna-7B
Temporal	15%	20%	11.6%
Spatial	16.6%	7.2%	15%

We terminate FMF workloads when the accuracy reaches the validation target or epochs. However, an important question is whether the transfer learning would harm model performance. Table 4 presents the maximal, minimum, and average relative accuracy (performance) improvement of tasks fine-tuned with temporal and spatial transfer compared to normal transfer. Vicuna can attain maximal 68.82% accuracy improvement for the BLEU metric of SAMSUM [25] with spatial transfer with DA [45]. The minimum accuracy improvement is no less than zero. To summarize, both temporal and spatial transfer improve model accuracy. This is in line with previous works [4, 61, 69] that transfer learning can improve the model performance. Moreover, the fractions of tasks participating in different transfer learning modes are shown in Table 5. About 20-30% of workloads are assigned temporal or spatial transfer learning modes. Different FMs present various preferences toward transfer learning modes, and no single dominant transfer learning mode exists.

Large-scale simulation. We use our simulator to conduct large-scale simulation experiments. We set the cluster capacity as 60 4-GPU nodes, and vary the job load from 1 × to 2×. Specifically, based on the model scale, we set 1 × job load as 1500 - 3000 jobs. Figure 11 shows Ymir achieves 1.66 - 22.3× JCT speedup across different job loads and FMs. Also, Figure 11 presents the average JCT (seconds) of Ymir. The speedup gain of Vicuna is more significant than that of small FMs, especially compared to Optimus. Pollux cannot perform satisfactorily in large-scale simulation experiments due to the high search cost of its adopted evolutionary algorithm. With the increase of the job load, Ymir presents a better JCT speedup, as a higher job load potentially brings more beneficial task combinations and thus provides more chances to reduce the JCT. Besides, the maximal/average of the ILP solver latency for 2 ×

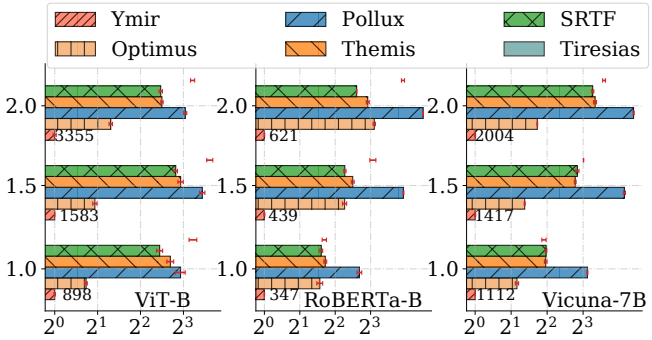


Figure 11: Scheduling efficiency results over FMs and job loads in simulation experiments. x-axis is the JCT normalized to Ymir while y-axis is the job load.

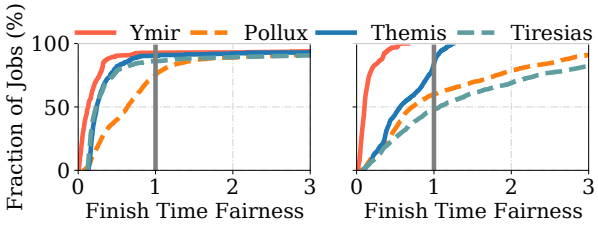


Figure 12: CDF of FTF for RoBERTa (left) and ViT (right).

jobs is 0.23/5.43 seconds using one CPU core, which does not have significant impact on the scheduling performance.

Figure 12 compares the cumulative distribution function (CDF) of the finish time fairness (FTF) metric between Ymir and other fairness baselines (Pollux, Themis, and Tiresias) for RoBERTa-Base and ViT-Base. We follow Shockwave's implementation [95] to compute FTF and draw the CDF curve under 1× job load. Our observation is that Ymir outperforms existing fairness baselines considerably. Note that Ymir even achieves zero FTF loss in the case of ViT-Base. We conclude that the benefit brought by transferability can enhance efficiency and fairness very well.

7.3 Evaluation of YmirEstimator

Time estimator. The key component of *time estimator* is LUT. To evaluate its robustness, we manually add uniform random noise to the result of LUT before reporting it.

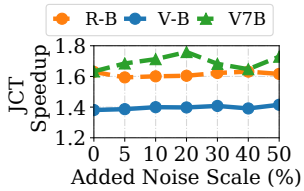
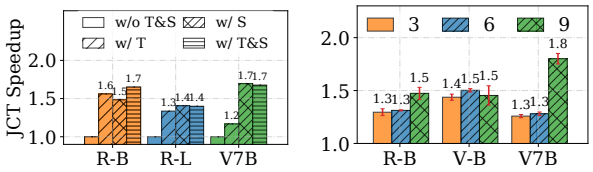


Figure 13: Time Estimator

Figure 13 varies the degree of the added noise (x-axis) and presents the speedup (y-axis) compared to Ymir without *task merger*. Increasing estimation error has no significant impact on scheduling efficiency. This primarily results from the fact that the scheduling objective (Eqn. 6) is not sensitive to the throughput estimation error.

Transfer learning modes. In Figure 14(a) investigates the contributions of different transfer learning modes to scheduling performance improvement over different FMs. No single transfer learning mode dominates across all FMs. Nevertheless, when both transfer



(a) Transfer Learning Modes (b) Number of datasets

Figure 14: Impact of key components.

learning modes are jointly considered, the scheduling performance experiences a further enhancement. Except that temporal transfer learning degrades the JCT speedup brought by spatial transfer learning in Vicuna. This could arise from the prediction error of our adopted estimator.

7.4 Impact of LUT and Pipeline Switch

Performance contribution of LUT. We compare LUT with the throughput estimator adopted in Pollux. Table 6 (row w/ LUT) reports the JCT of the throughput estimator normalized to that of our LUT. We observe that LUT is more beneficial to language FMs than vision FM. Little performance gain is shown for ViT-Base. The efficiency of the throughput estimator depends upon the fact that the job throughput scales linearly with the increase of the batch size and allocated GPUs. Its effectiveness is extensively validated in vision tasks [62], but is not satisfactory for language tasks.

Pipeline dataloader and model preparation. In this paper, we use PETL to reduce the size of parameters to compute and communicate gradients for most FMF workloads. Hence, most FMF tasks adopt data parallelism, and the pipeline switch between parameter transfer and gradient computation is insignificant for such a scenario. The dataloader preparation becomes a performance bottleneck that restricts scheduling flexibility. Ymir proactively invokes this step to hide the data preparation to the greatest extent before fine-tuning the next FMs. Table 6 (row w/ data pipe) shows the JCT without the dataloader pipeline normalized to that with the dataloader pipeline. The pipeline dataloader brings 1.1 - 1.7× JCT speedup.

Pipeline parameter transfer and model execution. We propose to execute the context switch between two pipeline parallelism workloads in a pipelined way. This pipeline context switch can considerably reduce the exorbitant time cost of the context switch. This technique is not applicable to all FMF tasks. We mainly examine how this pipeline practice benefits to fine-tuning Vicuna on ROC [53]. It does not bring apparent cluster-wide JCT speedup but reduces around 4% JCT for tasks fine-tuning Vicuna on ROC.

7.5 Impact of Transferability Estimation

Impact of transferability metrics. We categorize existing metrics for task transferability estimation into probability-based, feature-based, and gradient-based methods. (1) LEEP [55] is a representative probability-based method incorporating the entire dataset to estimate the data distribution accurately. The computation overhead of LEEP scales with the dataset size. The estimation accuracy of probability-based methods closely correlates with the number of classes [11]. Hence, LEEP fails to perform regression and generation

Table 6: Speedup brought by LUT and Pipeline Switch.

	ViT-B	RoBERTa-B	Vicuna-7B
w/ LUT	1.03	1.56	1.29
w/ data pipe	1.12	1.22	1.75

Table 7: Performance of various transferability metrics.

FM	LEEP [55]		Task2Feat [80]		Task2Vec [3]	
	Speedup	Max (s)	Speedup	Max (s)	Speedup	Max (s)
ViT-Base	0.85	30.50	1.26	**5.87**	**1.30**	24.63
RoBERTa-Base	0.74	202.46	1.12	923.46	**1.57**	**75.64**
Vicuna-7B	-	-	0.99	838.03	**1.72**	**102.85**

tasks (e.g., Vicuna). (2) Task2Feat [80] is a feature-based method that extracts the penultimate layer's features over the entire dataset and designs various metrics to measure the similarity between tasks. Hence, the computation overhead is exorbitant when the number of examples is enormous. (3) Our adopted Task2Vec [3] is a gradient-based method, which adopts a subset of the dataset to quantify the transferability between tasks. We compare the speedup brought by *task merger* using three task transferability estimation metrics in Table 7, and find Task2Vec achieves the best JCT speedup over different FMs. Task2Feat falls behind on JCT speedup. LEEP has adverse effects on cluster-wide efficiency for ViT-Based. The maximal profiling overhead of various metrics is shown in Table 7, and Task2Vec considerably reduces the overhead compared to other baselines on language FMs. Overall, Task2Vec is a suitable metric for transferability estimation.

Sensitivity to the number of datasets. We vary the number of datasets from 3 to 9 and present the JCT speedup between Ymir with and without using *task merger* in Figure 14(b). Our observation is that *task merger* can attain at least 1.3 × JCT speedup over different numbers of datasets and FMs. We acknowledge the JCT speedup brought by *task merger* correlates with the intrinsic task transferability. Our sensitivity analysis demonstrates that the performance improvement brought by *task merger* does not arise from our cherry-picking datasets.

8 RELATED WORKS

Transfer learning. Initially, this technique aims to transfer the weights of a pre-trained model to downstream tasks to reduce the training time and data [88]. Many works adopt heuristic methods [3, 10, 55, 80, 89] to determine the optimal pre-trained model for initialization based on the task similarity. Additionally, some works estimate the performance of different transfer learning modes [3, 10, 22, 34, 55, 76, 80, 89], as discussed in § 2.1. Other works [18, 82, 83] morph a well-trained model to a new one to warm start the training. The advancements in transfer learning can be leveraged to further improve Ymir.

DLT schedulers. Recent efforts of DLT schedulers primarily focus on effective resource allocations towards data-parallel training [9, 16, 26, 36, 38, 59, 62, 86, 95]. Nevertheless, existing DLT schedulers cannot adapt to FMF workloads because they overlook the optimization opportunities presented by the unique characteristics of FMF workloads. While Titan [24] focuses on scheduling pipeline-parallel FMF workloads in GPU data centers, it lacks a systematic solution to exploit task transferability to enhance overall cluster-wide efficiency. Ymir automates task merging scenarios and optimizes resource allocations. Furthermore, Ymir contributes to reducing context switch overhead for both data- and pipeline-parallel workloads.

Fine-tuning FMs. Recent advances in model fine-tuning are primarily limited to individual jobs from the algorithm and system perspectives. Many PETL architectures have been proposed to improve the model accuracy on language tasks [27, 32, 33, 43, 60, 91, 94] and vision tasks [17, 57, 74, 90]. Apart from [31], Ymir can utilize another unified PETL architecture called Unipelt [51], which learns to activate the PETL architectures for downstream tasks. These works can attain competitive model accuracy compared to fine-tuning all the parameters. Apart from the algorithmic innovations, several system works [8, 23, 66, 70] provide efficient pipeline parallelism for FMF workloads. Different from these single-workload optimization, Ymir optimizes cluster-wide FMF workloads.

9 DISCUSSION

Extensions to other transfer learning modes. Ymir considers combining at most two tasks. Intuitively, jointly fine-tuning more tasks can increase the potential benefit of transfer learning, but the lack of ML studies to estimate transfer gains when combining multiple tasks (≥ 3) impedes combining more tasks. Moreover, our empirical results have shown that merging two tasks can yield sufficiently good results.

Managing multiple FMs. This paper mainly evaluates the scenario with one FM. There can be numerous FMs in the datacenter for fine-tuning. Then, we can adopt a load-balancing policy to determine the GPU quotas for each FM, and more sophisticated designs can be our future work. Nevertheless, our empirical results have demonstrated the potential of Ymir in improving the efficiency.

Catastrophic forgetting. Temporal transfer learning is susceptible to the catastrophic forgetting issue. Fortunately, many works [46, 78, 92] point out that PETL can effectively avoid catastrophic forgetting. Empirically, our adopted Task2Vec metric can identify positive temporal transfer to mitigate catastrophic forgetting.

Privacy concerns. Ymir merges FMF workloads from different users to achieve high efficiency. Although Ymir does not directly share datasets but just parameters, there is still a potential privacy threat from malicious users, e.g., membership inference [71]. To handle this, Ymir allows users to disable sharing parameters.

10 CONCLUSION

This paper presents Ymir, a novel scheduler tailored for FMF workloads in GPU clusters. We propose YmirEstimator and YmirSched to determine the optimal transfer learning modes and resource allocations. We design YmirTuner to improve the efficiency of individual FMF workloads with PETL architectures and pipeline schemes. Our extensive experiments demonstrate that Ymir outperforms existing DLT schedulers in job efficiency and resource allocation fairness.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their valuable comments. The research is supported under the RIE2020 Industry Alignment

Fund - Industry Collaboration Projects (IAF-ICP) Funding Initiative, as well as cash and in-kind contribution from the industry partner(s).

REFERENCES

[1] 2022. HuggingFace Model Hub. https://huggingface.co/models?sort=downloads..

[2] 2022. OpenAI Fine-tuning Service. https://beta.openai.com/docs/guides/fine-tuning..

[3] Alessandro Achille, Michael Lam, Rahul Tewari, Avinash Ravichandran, Subhransu Maji, Charless C Fowlkes, Stefano Soatto, and Pietro Perona. 2019. Task2vec: Task embedding for meta-learning. In *CVPR*. 6430–6439.

[4] Vamsi Aribandi, Yi Tay, Tal Schuster, Jinfeng Rao, Huaixiu Steven Zheng, Sanket Vaibhav Mehta, Honglei Zhuang, Vinh Q Tran, Dara Bahri, Jianmo Ni, et al. 2021. Ext5: Towards extreme multi-task scaling for transfer learning. *arXiv preprint arXiv:2111.10952* (2021).

[5] Sanjith Athlur, Nitika Saran, Muthian Sivathanu, Ramachandran Ramjee, and Nipun Kwatra. 2022. Varuna: scalable, low-cost training of massive deep learning models. In *Eurosys*. 472–487.

[6] Alexei Baevski, Yuhao Zhou, Abdelrahman Mohamed, and Michael Auli. 2020. wav2vec 2.0: A framework for self-supervised learning of speech representations. *Neurips* 33 (2020), 12449–12460.

[7] Zhihao Bai, Zhen Zhang, Yibo Zhu, and Xin Jin. 2020. PipeSwitch: Fast Pipelined Context Switching for Deep Learning Applications. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI '20)*.

[8] Mandeep Baines, Shruti Bhosale, Vittorio Caggiano, Naman Goyal, Siddharth Goyal, Myle Ott, Benjamin Lefaudeux, Vitaliy Liptchinsky, Mike Rabbat, Sam Sheiffer, Anjali Sridhar, and Min Xu. 2021. FairScale: A general purpose modular PyTorch library for high performance and large scale training.

[9] Zhengda Bian, Shenggui Li, Wei Wang, and Yang You. 2021. Online evolutionary batch size orchestration for scheduling deep learning workloads in GPU clusters. In *SC*.

[10] Daniel Bolya, Rohit Mittapalli, and Judy Hoffman. 2021. Scalable Diverse Model Selection for Accessible Transfer Learning. *Neurips* 34 (2021), 19301–19312.

[11] Daniel Bolya, Rohit Mittapalli, and Judy Hoffman. 2021. Scalable Diverse Model Selection for Accessible Transfer Learning. *Neurips* 34 (2021), 19301–19312.

[12] Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. 2021. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258* (2021).

[13] Lukas Bossard, Matthieu Guillaumin, and Luc Van Gool. 2014. Food-101 – Mining Discriminative Components with Random Forests. In *ECCV*.

[14] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D 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 Ziegler, Jeffrey Wu, Clemens Winter, Chris 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] Brendan Burns, Brian Grant, David Oppenheimer, Eric Brewer, and John Wilkes. 2016. Borg, Omega, and Kubernetes: Lessons Learned from Three Container-Management Systems over a Decade. *Queue* (2016).

[16] Shubham Chaudhary, Ramachandran Ramjee, Muthian Sivathanu, Nipun Kwatra, and Srinidhi Viswanatha. 2020. Balancing Efficiency and Fairness in Heterogeneous GPU Clusters for Deep Learning. In *Proceedings of the Fifteenth European Conference on Computer Systems (EuroSys '20)*.

[17] Hao Chen, Ran Tao, Han Zhang, Yidong Wang, Wei Ye, Jindong Wang, Guosheng Hu, and Marios Savvides. 2022. Conv-Adapter: Exploring Parameter Efficient Transfer Learning for ConvNets. *arXiv preprint arXiv:2208.07463* (2022).

[18] Tianqi Chen, Ian Goodfellow, and Jonathon Shlens. 2015. Net2net: Accelerating learning via knowledge transfer. *arXiv preprint arXiv:1511.05641* (2015).

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

[20] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL '19)*.

[21] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. 2020. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929* (2020).

[22] Kshitij Dwivedi and Gemma Roig. 2019. Representation similarity analysis for efficient task taxonomy & transfer learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 12387–12396.

[23] Saar Eliad, Ido Hakimi, Alon De Jagger, Mark Silberstein, and Assaf Schuster. 2021. Fine-tuning giant neural networks on commodity hardware with automatic pipeline model parallelism. In *USENIX ATC*.

[24] Wei Gao, Peng Sun, Yonggang Wen, and Tianwei Zhang. 2022. Titan: a scheduler for foundation model fine-tuning workloads. In *ACM SoCC*. 348–354.

[25] Bogdan Gliwa, Iwona Mochol, Maciej Biesek, and Aleksander Wawer. 2019. SAMSum Corpus: A Human-annotated Dialogue Dataset for Abstractive Summarization. In *Proceedings of the 2nd Workshop on New Frontiers in Summarization*. Association for Computational Linguistics, Hong Kong, China, 70–79. https://doi.org/10.18653/v1/D19-5409

[26] Juncheng Gu, Mosharaf Chowdhury, Kang G. Shin, Yibo Zhu, Myeongjae Jeon, Junjie Qian, Hongqiang Liu, and Chuanxiong Guo. 2019. Tiresias: A GPU Cluster Manager for Distributed Deep Learning. In *NSDI*.

[27] Demi Guo, Alexander M Rush, and Yoon Kim. 2020. Parameter-efficient transfer learning with diff pruning. *arXiv preprint arXiv:2012.07463* (2020).

[28] Peizhen Guo, Bo Hu, and Wenjun Hu. 2022. Sommelier: Curating DNN Models for the Masses. In *ICDM*. 1876–1890.

[29] Mingcong Han, Hanze Zhang, Rong Chen, and Haibo Chen. 2022. Microsecond-scale preemption for concurrent {GPU-accelerated}{DNN} inferences. In *OSDI*. 539–558.

[30] Yaru Hao, Li Dong, Furu Wei, and Ke Xu. 2019. Visualizing and understanding the effectiveness of BERT. *arXiv preprint arXiv:1908.05620* (2019).

[31] Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. 2022. Towards a Unified View of Parameter-Efficient Transfer Learning. In *ICLR*.

[32] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for NLP. In *ICML*. PMLR, 2790–2799.

[33] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685* (2021).

[34] Long-Kai Huang, Junzhou Huang, Yu Rong, Qiang Yang, and Ying Wei. 2022. Frustratingly easy transferability estimation. In *ICML*. 9201–9225.

[35] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, et al. 2019. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *Neurips* (2019).

[36] Changho Hwang, Taehyun Kim, Sunghyun Kim, Jinwoo Shin, and KyoungSoo Park. 2021. Elastic Resource Sharing for Distributed Deep Learning. In *NSDI*.

[37] Insu Jang, Zhenning Yang, Zhen Zhang, Xin Jin, and Mosharaf Chowdhury. 2023. Oobleck: Resilient Distributed Training of Large Models Using Pipeline Templates. In *SOSP*. 382–395.

[38] Myeongjae Jeon, Shivaram Venkataraman, Amar Phanishayee, Junjie Qian, Wencong Xiao, and Fan Yang. 2019. Analysis of Large-Scale Multi-Tenant GPU Clusters for DNN Training Workloads. In *USENIX ATC*.

[39] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361* (2020).

[40] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[41] Alex Krizhevsky. 2009. *Learning multiple layers of features from tiny images*. Technical Report.

[42] Fan Lai, Yinwei Dai, Harsha V. Madhyastha, and Mosharaf Chowdhury. 2023. ModelKeeper: Accelerating DNN Training via Automated Training Warmup. In *NSDI*.

[43] Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691* (2021).

[44] Conglong Li, Minjia Zhang, and Yuxiong He. 2022. The Stability-Efficiency Dilemma: Investigating Sequence Length Warmup for Training GPT Models. In *Neurips*.

[45] Junyi Li, Tianyi Tang, Gaole He, Jinhao Jiang, Xiaoxuan Hu, Puzhao Xie, Zhipeng Chen, Zhuohao Yu, Wayne Xin Zhao, and Ji-Rong Wen. 2021. TextBox: A Unified, Modularized, and Extensible Framework for Text Generation. In *ACL*. 30–39.

[46] Yingting Li, Ambuj Mehrish, Rishabh Bhardwaj, Navonil Majumder, Bo Cheng, Shuai Zhao, Amir Zadeh, Rada Mihalcea, and Soujanya Poria. 2023. Evaluating parameter-efficient transfer learning approaches on sure benchmark for speech understanding. In *ICASSP*. IEEE, 1–5.

[47] Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. 2019. Multi-task deep neural networks for natural language understanding. *arXiv preprint arXiv:1901.11504* (2019).

[48] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692* (2019).

[49] Kai Lv, Yuqing Yang, Tengxiao Liu, Qinghui Gao, Qipeng Guo, and Xipeng Qiu. 2023. Full Parameter Fine-tuning for Large Language Models with Limited Resources. *arXiv preprint arXiv:2306.09782* (2023).

[50] Kshiteej Mahajan, Arjun Balasubramanian, Arjun Singhvi, Shivaram Venkataraman, Aditya Akella, Amar Phanishayee, and Shuchi Chawla. 2020. Themis: Fair and Efficient GPU Cluster Scheduling. In *NSDI*.

[51] Yuning Mao, Lambert Mathias, Rui Hou, Amjad Almahairi, Hao Ma, Jiawei Han, Wen-tau Yih, and Madian Khabsa. 2021. Unipelt: A unified framework for parameter-efficient language model tuning. *arXiv preprint arXiv:2110.07577* (2021).

[52] Sam McCandlish, Jared Kaplan, Dario Amodei, and OpenAI Dota Team. 2018. An empirical model of large-batch training. *arXiv preprint arXiv:1812.06162* (2018).

[53] Nasrin Mostafazadeh, Nathanael Chambers, Xiaodong He, Devi Parikh, Dhruv Batra, Lucy Vanderwende, Pushmeet Kohli, and James Allen. 2016. A Corpus and Cloze Evaluation for Deeper Understanding of Commonsense Stories. In *ACL*. 839–849.

[54] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, Amar Phanishayee, and Matei Zaharia. 2021. Efficient large-scale language model training on GPU clusters using megatron-LM. In *SC*.

[55] Cuong Nguyen, Tal Hassner, Matthias Seeger, and Cedric Archambeau. 2020. Leep: A new measure to evaluate transferability of learned representations. In *International Conference on Machine Learning*. PMLR, 7294–7305.

[56] OpenAI. 2023. GPT-4 Technical Report. arXiv:2303.08774 [cs.CL]

[57] Junting Pan, Ziyi Lin, Xiatian Zhu, Jing Shao, and Hongsheng Li. 2022. Parameter-efficient image-to-video transfer learning. *arXiv e-prints* (2022), arXiv–2206.

[58] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Neurips*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett (Eds.). 8024–8035.

[59] Yanghua Peng, Yixin Bao, Yangrui Chen, Chuan Wu, and Chuanxiong Guo. 2018. Optimus: An Efficient Dynamic Resource Scheduler for Deep Learning Clusters. In *Eurosys*.

[60] Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. 2020. AdapterFusion: Non-destructive task composition for transfer learning. *arXiv preprint arXiv:2005.00247* (2020).

[61] Clifton Poth, Jonas Pfeiffer, Andreas Rücklé, and Iryna Gurevych. 2021. What to pre-train on? efficient intermediate task selection. *arXiv preprint arXiv:2104.08247* (2021).

[62] 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 *OSDI*.

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

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

[65] 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.

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

[67] Siva Reddy, Danqi Chen, and Christopher D. Manning. 2019. CoQA: A Conversational Question Answering Challenge. *Transactions of the Association for Computational Linguistics* 7 (2019), 249–266.

[68] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. 2015. ImageNet Large Scale Visual Recognition Challenge. *IJCV* (2015), 211–252.

[69] Victor Sanh, Albert Webson, Colin Raffel, Stephen H Bach, Lintang Sutawika, Zaid Alyafeai, Antoine Chaffin, Arnaud Stiegler, Teven Le Scao, Arun Raja, et al. 2021. Multitask prompted training enables zero-shot task generalization. *arXiv preprint arXiv:2110.08207* (2021).

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

[71] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. 2017. Membership inference attacks against machine learning models. In *SP*. 3–18.

[72] Andreas Steiner, Alexander Kolesnikov, Xiaohua Zhai, Ross Wightman, Jakob Uszkoreit, and Lucas Beyer. 2021. How to train your vit? data, augmentation, and regularization in vision transformers. *arXiv preprint arXiv:2106.10270* (2021).

[73] Yusheng Su, Xiaozhi Wang, Yujia Qin, Chi-Min Chan, Yankai Lin, Huadong Wang, Kaiyue Wen, Zhiyuan Liu, Peng Li, Juanzi Li, Lei Hou, Maosong Sun, and Jie Zhou. 2022. On Transferability of Prompt Tuning for Natural Language Processing. In *NAACL*. 3949–3969.

[74] Yi-Lin Sung, Jaemin Cho, and Mohit Bansal. 2022. Vl-adapter: Parameter-efficient transfer learning for vision-and-language tasks. In *CVPR*. 5227–5237.

[75] John Thorpe, Pengzhan Zhao, Jonathan Eyolfson, Yifan Qiao, Zhihao Jia, Minjia Zhang, Ravi Netravali, and Guoqing Harry Xu. 2023. Bamboo: Making Preemptible Instances Resilient for Affordable Training of Large {DNNs}. In *NSDI*. 497–513.

[76] Anh T Tran, Cuong V Nguyen, and Tal Hassner. 2019. Transferability and hardness of supervised classification tasks. In *ICCV*. 1395–1405.

[77] Nilesh Tripuraneni, Michael Jordan, and Chi Jin. 2020. On the theory of transfer learning: The importance of task diversity. *Neurips* 33 (2020), 7852–7862.

[78] Steven Vander Eeckt and Hugo Van Hamme. 2023. Using adapters to overcome catastrophic forgetting in end-to-end automatic speech recognition. In *ICASSP*. IEEE, 1–5.

[79] Tom Viering and Marco Loog. 2022. The shape of learning curves: a review. *TPAMI* (2022).

[80] Tu Vu, Tong Wang, Tsendsuren Munkhdalai, Alessandro Sordoni, Adam Trischler, Andrew Mattarella-Micke, Subhransu Maji, and Mohit Iyyer. 2020. Exploring and predicting transferability across NLP tasks. *arXiv preprint arXiv:2005.00770* (2020).

[81] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. 2018. GLUE: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461* (2018).

[82] Abdul Wasay, Brian Hentschel, Yuze Liao, Sanyuan Chen, and Stratos Idreos. 2020. Mothernets: Rapid deep ensemble learning. *Proceedings of Machine Learning and Systems* (2020), 199–215.

[83] Tao Wei, Changhu Wang, Yong Rui, and Chang Wen Chen. 2016. Network morphism. In *International conference on machine learning*. PMLR, 564–572.

[84] Orion Weller, Kevin Seppi, and Matt Gardner. 2022. When to Use Multi-Task Learning vs Intermediate Fine-Tuning for Pre-Trained Encoder Transfer Learning. *arXiv preprint arXiv:2205.08124* (2022).

[85] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. Transformers: State-of-the-Art Natural Language Processing. In *EMNLP*. 38–45.

[86] 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 *OSDI*.

[87] Qinyuan Ye, Bill Yuchen Lin, and Xiang Ren. 2021. Crossfit: A few-shot learning challenge for cross-task generalization in nlp. *arXiv preprint arXiv:2104.08835* (2021).

[88] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. 2014. How transferable are features in deep neural networks? *Neurips* 27 (2014).

[89] Kaichao You, Yong Liu, Jianmin Wang, and Mingsheng Long. 2021. Logme: Practical assessment of pre-trained models for transfer learning. In *International Conference on Machine Learning*. PMLR, 12133–12143.

[90] Bruce XB Yu, Jianlong Chang, Lingbo Liu, Qi Tian, and Chang Wen Chen. 2022. Towards a Unified View on Visual Parameter-Efficient Transfer Learning. *arXiv preprint arXiv:2210.00788* (2022).

[91] Elad Ben Zaken, Shauli Ravfogel, and Yoav Goldberg. 2021. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. *arXiv preprint arXiv:2106.10199* (2021).

[92] Guangtao Zeng, Peiyuan Zhang, and Wei Lu. 2023. One Network, Many Masks: Towards More Parameter-Efficient Transfer Learning. *arXiv preprint arXiv:2305.17682* (2023).

[93] Haoyu Zhang, Logan Stafman, Andrew Or, and Michael J. Freedman. 2017. SLAQ: Quality-Driven Scheduling for Distributed Machine Learning. In *ACM SoCC*.

[94] Mengjie Zhao, Tao Lin, Fei Mi, Martin Jaggi, and Hinrich Schütze. 2020. Masking as an efficient alternative to finetuning for pretrained language models. *arXiv preprint arXiv:2004.12406* (2020).

[95] Pengfei Zheng, Rui Pan, Tarannum Khan, Shivaram Venkataraman, and Aditya Akella. 2023. Shockwave: Fair and Efficient Cluster Scheduling for Dynamic Adaptation in Machine Learning. In *NSDI*.