TITAN: A Scheduler for Foundation Model Fine-tuning Workloads

Wei Gao[1,2], Peng Sun[3], Yonggang Wen[1], Tianwei Zhang[1]
[1]School of Computer Science and Engineering, Nanyang Technological University
[2]S-Lab, Nanyang Technological University [3]SenseTime
gaow0007@e.ntu.edu.sg, sunpeng1@sensetime.com, {ygwen, tianwei.zhang}@ntu.edu.sg

ABSTRACT

The recent breakthrough of foundation model (FM) research raises a new trend to acquire efficient DL models by fine-tuning FMs with low-resource datasets. Current GPU clusters are mainly established to develop DL models by training from scratch. How to tailor a GPU cluster scheduler for FM fine-tuning workloads is still not explored.

We propose TITAN, a scheduler to improve the efficiency of FM fine-tuning workloads based on their three distinct features. (1) It takes full advantage of the fixed model structure to estimate the job duration accurately and configure the fine-tuning workload efficiently. (2) The multi-task adaptivity of FMs enables multiple fine-tuning workloads to share the same model parameters, which can significantly reduce the GPU resource consumption. (3) It considers the pipeline parallelism of FM fine-tuning workloads and concurrently executes the parameter transmission and gradient computation to hide the overhead of context switch. Preliminary simulation result validates the efficiency of TITAN.

CCS CONCEPTS

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

KEYWORDS

GPU Datacenter, Deep Learning Training, Cluster Management System, Foundation Models

ACM Reference Format:
Wei Gao[1,2], Peng Sun[3], Yonggang Wen[1], Tianwei Zhang[1]. 2022. TITAN: A Scheduler for Foundation Model Fine-tuning Workloads. In *Symposium on Cloud Computing (SoCC '22), November 7–11, 2022, San Francisco, CA, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3542929.3563460

SoCC '22, November 7–11, 2022, San Francisco, CA, USA

ACM ISBN 978-1-4503-9414-7/22/11.
https://doi.org/10.1145/3542929.3563460

1 INTRODUCTION

The foundation model (FM) is an emerging technique to push the envelope of various artificial intelligence (AI) tasks [1, 10, 11]. Training FMs typically demands thousands of GPUs for a long time. The expensive cost drives the AI community to approach a quantity of downstream AI tasks through fine-tuning. Such *FM fine-tuning* workloads are becoming increasingly important in modern GPU clusters.

This paper aims to *design a scheduler to efficiently fine-tune FMs in a large-scale GPU cluster*. It is easy to leverage existing schedulers for general DL training workloads [17, 29, 42] to manage FM fine-tuning workloads. However, they do not consider the unique features of FM fine-tuning workloads, and leave large efficiency space to improve. First, fine-tuning FMs usually fixes the model structure and explores the optimal model parameters. Several works [4, 8, 15] have discussed the impacts of resource allocations on the job run-time speed, which raise challenges of estimating the job duration. The fixed model structure provides an opportunity to predict the run-time of each job without online profiling. The execution of FM fine-tuning commonly adopts a hybrid of data-parallelism, tensor-parallelism and pipe-line parallelism. An effective parallelism execution plan can improve the job throughput by up to ten times [40], but usually takes several minutes to search for it. Since the best plan depends on the model structure, batch size, and input shape, we can cache these optimal solutions for future use.

Second, a FM fine-tuning workload can be multi-task adaptive, i.e., FMs can perform well over several tasks when their adopted datasets share similar semantic information. This indicates that several fine-tuning workloads can share the same model parameters. Thus, we can merge several fine-tuning workloads into one to improve the GPU utilization while maintaining the performance.

Third, fine-tuning FMs demands extensive GPU resources for a relatively short time. This leads to frequent context switches, each taking up to minutes. The overhead is intolerable considering the job duration length. Pipeline parallelism is a common adoption in FM fine-tuning workloads. Hence we can pipeline the gradient computation of one job and parameter access of another job to amortize this overhead.

Provider	Name	Model Size	Task	Code
OpenAI	GPT-3 [5]	175 B	✿	✗
Ali Cloud	M6 [24]	10 B	✿	Δ
Baidu	ERNIE 3.0 [39]	10 B	✿ ♥ ♣	Δ
Hugging Face	AutoTrain [13]	N/A	Any	✗

Table 1: Comparisons of existing fine-tuning service. We denote ✿, ♥, ♣ as NLP, CV, and multi-modal tasks, respectively. Δ indicates that the user needs to call the API instead of programming from scratch.

These insights lead to Titan, a novel scheduler that accounts for the distinct characteristics of FM fine-tuning workloads to improve the cluster efficiency. Existing DL schedulers (e.g., Themis [27], Optimus [29], AFS [22], Pollux [30]) assume that DL workloads support the light-weight checkpoint and/or data-parallelism to make effective decisions about resource allocations. However, such assumption is not applicable to FM fine-tuning workloads. Instead, we choose two baseline systems for comparisons: a non-preemptive Shortest Remaining Time First (SRTF) scheduler and Tiresias [17], which reduce the number of resource re-allocations. We simulate Titan on a cluster of 16 nodes with 64 GPUs, running a ViT-Large/16 [11], a FM for computer vision tasks. The results present the advantages of Titan over both baselines.

2 BACKGROUND

Foundation Model. Recently, AI researchers expect to train large-scale pre-trained models on massive datasets, which can further transfer their knowledge over different tasks. These large-scale models are referred to as foundation models (FMs). Basically, FMs can serve as a knowledge base to boost the performance across various tasks, including NLP [5, 10, 14, 31, 32, 36, 43], CV [7, 18, 25, 26, 34], and biology [12, 19, 35].

Fine-tuning-Foundation-Models as a Service. The success of FMs motivates AI practitioners to explore their business practice. We summarize existing commercial services about fine-tuning FMs in Tab. 1. Commonly, to fine-tune a FM, the user needs to prepare a dataset. The scheduler receives the request, and makes effective resource allocation to each workload and returns a downstream model with satisfactory performance.

Pipeline DL Training. Pipeline parallelism is an approach to partition layers across machines so as to train a FM which can not be held by a single machine. It also adopts micro-batches to saturate the pipeline as much as possible to improve resource utilization. DeepSpeed [33], Megatron [37], and fairscale [3] are popular DL training libraries to support pipeline parallelism. Our system is empowered by fairscale to implement pipeline switch, as illustrated in Sec. 4.3.

3 PROBLEM FORMULATION

We design a non-preemptive scheduler to minimize the average job completion time (JCT) of given FM fine-tuning workloads. We formulate this job scheduling task as a Mixed Integer Linear Programming (MILP) problem, and call the MILP solver to determine the resource allocations in the events of job completion and job submission.

We consider N pending jobs: $\mathcal{J} = \{j_1, j_2, j_3 ... j_N\}$ and M available GPUs. The number of allocated GPUs to each job belongs to a given set $\mathbb{A} = \{1, 2, 4m | m \in N^+\}$. We denote $T_{i,a}$ as the progress per time unit of job j_i assigned with a GPUs. The progress per time unit is computed as the number of iterations per time unit divided by the total number of iterations. We will discuss how to predict $T_{i,a}$ in Sec. 4.1. We use a binary variable $x_{i,a}$ to denote whether j_i is allocated with a GPUs. The MILP solver can yield a solution to the following objective:

$$\max \sum_{i=1}^{N} \sum_{a \in \mathbb{A}} x_{i,a} * T_{i,a} \tag{1}$$

subject to:

$$x_{i,a} \in \{0, 1\}, \forall a \in \mathbb{A}, \forall i \in [1, N] \tag{2}$$

$$\sum_{a \in \mathbb{A}} x_{i,a} \leq 1, \forall i \tag{3}$$

$$\sum_{i=1}^{N} \sum_{a \in \mathbb{A}} x_{i,a} * a \leq M \tag{4}$$

Objective (1) is to maximize the cluster-wide progress per-time unit. Constraint (3) ensures that all pending jobs should be given at most one resource allocation policy. Constraint (4) ensures that the total number of allocated GPUs does not exceed the resource capacity.

From the solution of MILP solver, we can identify the resource allocation for each job. The number of GPUs assigned to each job is in set $\mathbb{A}$. Hence we can find a communication-optimal resource topology for each job. The powerful MILP solver can yeild the solution very rapidly: the average/maximal time of optimizing Eq. 1 for the scale of 320 jobs is 0.005/0.16 seconds on 1 vCPU. Next, we discuss how to utilize the characteristics of FM fine-tuning workloads to improve the scheduling efficiency.

4 SYSTEM DESIGN

Fig. 1 illustrates the workflow of our designed Titan. The user submits the dataset to the scheduler. Based on the dataset scale, the scheduler leverages a lookup table (LUT) to derive the job duration. Next, it adopts a task merger to combine several datasets into one to improve the resource utilization. When certain jobs complete, we leverage a fast pipelined context switch to hide the context switch overhead.

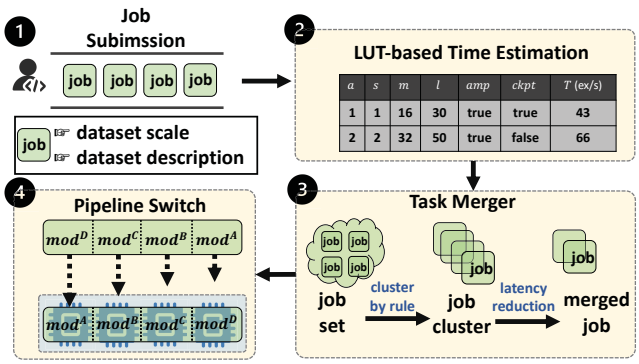

Figure 1: Workflow of TITAN.

	A	B	Task Merger	SRTF
Case 1	20	20	25	30
Case 2	10	20	21	20
Case 3	5	100	102	55

Table 2: The average JCT (min) of individual task, task merger, and Shortest Remaining Time First.

4.1 Time Estimator

Existing works adopt mathematical models [30] or machine learning algorithms [44] to estimate the job throughput. However, they primarily consider the data-parallelism setting and decompose the time cost per training iteration into gradient computation and synchronization. Besides, they do not fit into the complicated FM training acceleration techniques including gradient checkpoint, mixed precision training, frozen training. Fortunately, the fixed model structure allows to profile the job information in the offline manner. We propose a simple yet effective method named lookup table (LUT), which constructs a map $\mathsf{T}(a, s, m, \ell, amp, ckpt)$ between resource allocation solutions, training configurations and job throughput. a is the number of GPUs assigned to the job. s is the number of gradient accumulation steps. m is the global batch size. ℓ is the number of frozen layers during fine-tuning. amp and $ckpt$ denote whether to adopt the mixed-precision training and gradient checkpoint, respectively. To build such LUT, we use 64 GPUs to profile as many training configurations as possible, which can be completed within 37 hours.

For each fine-tuning request, the scheduler determines an appropriate batch size range and number of epochs based on the dataset scale. Given a resource allocation strategy, it searches the LUT to determine the best optimal training configurations to minimize the job latency. A linear interpolation can be adopted to estimate the run-time of unseen training configurations. Overall, LUT can not only estimate the job throughput but also determine the training configurations.

4.2 Task Merger

Apart from time estimation, the fixed model structure also enables to merge several individual workloads into one by combining their datasets together. However, the inappropriate dataset combination might sacrifice the model performance. The task merger shares a similar challenge with multi-task learning, which needs to balance the impacts of various tasks on the model performance. Since the performance improvement is not our focus, we pursue to improve the resource utilization while maintaining the model performance. Besides, the generality and transferability of FMs are relatively robust to task merging. We approach it by requiring users to provide a dataset card. Hugging Face abstracts away a dataset card to ease the management of a variety of datasets. The creation of a new dataset card requires the description of the domain and topic of the dataset, which contains certain key words, e.g., task category (classification), object category (cat, dog). These key words encode the semantic information of datasets, which can be extracted from the dataset description. Hence, the similarity score of these key words implicitly suggests whether the merged datasets negatively affect the model performance.

Merge as many tasks as possible is not an optimal way to minimize the average JCT. In the example of Table. 2, when the duration of two workloads is close, the task merger can outperform the SRTF solution in the overall JCT, as illustrated in case 1. When the duration of task B is twice that of task A. The average JCT of the task merger is comparable to that of SRTF. However, when the duration of task B is much longer than task A (case 3), the SRTF-ordered sequential execution is a better option. To address this issue, we denote a set of K potential task merging policies: $\mathcal{S} = \{mj_1, mj_2, mj_3, ...mj_K\}$. Each merged job mj_k is a subset of $\mathcal{S}$. We introduce a binary variable $y_{k,a}$ to denote whether we merge tasks in the set of mj_k and assign a GPUs to it. Otherwise we use $T_{N+k,a}$ to represent corresponding progress per time unit estimated through LUT. Hence, we can make a re-formulation of our scheduling problem as follows.

$$\max \left(\sum_{i=1}^{N} \sum_{a \in \mathbb{A}} x_{i,a} * T_{i,a} \right) + \left(\sum_{k=1}^{S} \sum_{a \in \mathbb{A}} y_{k,a} * T_{N+k,a} * \frac{1 + |mj_k|}{2} \right) \quad (5)$$

subject to:

$$x_{i,a} \in \{0, 1\}, \forall a \in \mathbb{A}, \forall i \in [1, N] \quad (6)$$

$$y_{k,a} \in \{0, 1\}, \forall a \in \mathbb{A}, \forall k \in [1, K] \quad (7)$$

$$\sum_{a \in \mathbb{A}} x_{i,a} \leq 1, \forall i \quad (8)$$

$$\sum_{a \in \mathbb{A}} \left(y_{k,a} + \sum_{j_i \in mj_k} x_{i,a} \right) \leq 1, \forall k \quad (9)$$

$$\sum_{a\in\mathbb{A}}\left(y_{k,a}+\sum_{k_1\neq k,k_1\cap k\neq\emptyset}y_{k_1,a}\right)\leq 1,\forall k \tag{10}$$

$$\sum_{a\in\mathbb{A}}a*\left(\sum_{i=1}^{N}x_{i,a}+\sum_{k=1}^{K}y_{k,a}\right)\leq M \tag{11}$$

We introduce $\frac{1+|mj_k|}{2}$ to re-weight the progress of the merged task in objective (5), and favor the merged one with a shorter job latency. Constraints (9-10) ensure that each job is assigned with at most one allocation policy, and there is no overlap between individual workload and merged workload. Constraint (11) ensures the total number of allocated GPUs does not exceed the resource capacity under the circumstance of the merged task.

The task merger might lead to the accuracy loss and data privacy issues. For accuracy, several papers [9, 16, 38] discussed how FMs succeed in multi-tasking learning. In practice, the user can specify the target accuracy. If the task merger cannot satisfy the target, the model will be trained alone for a while to recover the accuracy. For privacy, users can disable the task merger if they care about data privacy. Our scheduler can lower the priority of such jobs to incentive users to opt for the task merger.

4.3 Pipeline Switch

The expensive overhead of context switch between fine-tuning workloads exacerbates the scheduling flexibility and delays the job progress, especially for short-term ones. Fig. 3 presents the context switch overhead of ViT can be up to 46 seconds. Inspired by PipeSwitch [2], we propose pipeline switch to amortize such overhead.

Fig. 2 presents how to pipeline the gradient computation of job X and parameter transmission of job Y. Each machine maintains the entire model structure and partial model parameters. Both jobs X and Y adopt the pipeline parallelism on 4-GPU machines, and the FM is partitioned into four parts. For naming conventions, we use the subscript of job X and Y to denote the partition. Also, we use the superscript F, B and T to represent the forward propagation, backward propagation and parameter transmission respectively. When the context switch happens between jobs X and Y, we can overlap the parameter store of job X and the parameter load of job Y across machines. We also can pipeline the gradient computation and parameter transmission as much as possible in each machine. To this end, we require the task Y to compute from machine 4 to machine 1. On machine 4, after completing X_4^F, we can save the partial parameters of job X subsequently. Next, the partial parameters of task Y can be loaded into machine 4, and Y_1^F starts execution. Note that our pipeline adoption differs from PipeSwitch: (1) we consider the pipeline parallelism while PipeSwitch only focuses on

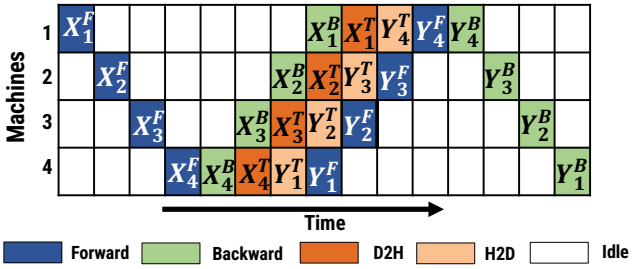

Figure 2: Pipeline model propagation and parameter transmission. D2H indicates saving parameters from device (GPU) to host (CPU). H2D indicates loading parameters from host (CPU) to device (GPU).

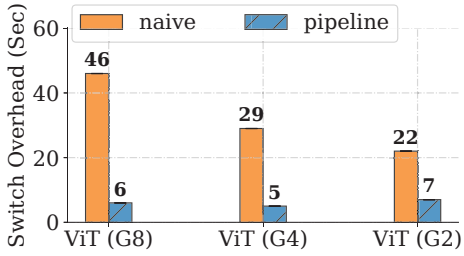

Figure 3: Context switch overhead of ViT-Large/16 with an allocation of 8 GPUs, 4 GPUs and 2 GPUs between naive methods and our adopted pipeline switch.

single GPU tasks; (2) our proposed reverse direction parameter load enables to hide the overhead of parameter switch while PipeSwitch fails to achieve it.

5 PRELIMINARY RESULTS

Trace Analysis. We study a trace of FM training/fine-tuning workloads from a production GPU cluster in our institute. This trace contains 18471 jobs over a period of 3 months on a cluster of 88 nodes, a total of 704 NVIDIA V100 GPUs. We select a subset of jobs the duration of which ranges from 5 minutes to 10 hours to emphasize the difference among different traces. Fig. 4 (left) compares the job duration distribution over Philly [23], Helios [21], MLaaS [41] and our adopted trace (FMTrace). We observe that short job duration (within 30 minutes) accounts for exceeding 50% of FMTrace. This implies that frequent context switch will lead to the performance loss of FM fine-tuning workloads. Fig. 4 (right) presents the GPU request distribution, and suggests that the distributed execution prefers FM fine-tuning workloads.

Simulator Constructor. We profile the job throughput of ViT-Large/16 [11] over various training configurations including accumulation steps, batch size, the number of frozen layers, mixed-precision training and gradient checkpoint. Note that the batch size scale is relatively large compared to other configurations. Therefore we profile across a batch size scale of $(64 \sim 512)$ with a base-$\sqrt{2}$ exponential increasing

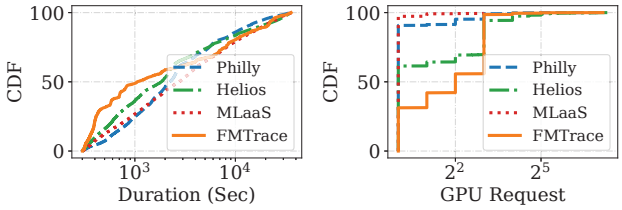

Figure 4: Trace characterization.

Scheduler Policy	Average JCT	Makespan
SRTF	1.68h(ours)	33.09h
Tiresias	1.67h	33.09h
TITAN (w/o task merger)	1.23h	33.11h
TITAN (w/o pipeline switch)	1.16h	**29.01h**
TITAN	**1.04h**	**29.01h**

Table 3: Summary of evaluation results.

workload density	0.5	1	1.5	2
% of jobs with JCT reduction	18%	19%	20%	19%
% of jobs with JCT increase	6%	8%	7%	11%

Table 4: Detailed comparison between TITAN and SRTF.

order. We also collect the time cost of context switch with and without our pipeline switch mechanism over resource allocations, as depicted in Fig. 3. Regarding the simulation trace, we transform the requested GPU-time to the dataset scale. We follow the submission pattern and distribution of FMTrace to synthesize our workload including 320 jobs submitted within first 8 hours.

Evaluation Results. Tab. 3 presents the average job completion time and makespan of TITAN and its variants as well as other baselines. Tiresias can achieve a competitive performance to SRTF policy because SRTF suffers from significant context switch overhead. TITAN can reduce 38% average JCT and 12% makespan compared to baseline schedulers. Moreover, task merger and pipeline switch contribute to 15% and 10% average JCT reduction respectively. We also vary the workload density relative to a total number of 320 jobs while keep the cluster capacity fixed, as shown in Fig. 5. When we increase the job density, the average JCT of all scheduling policies become longer. Moreover, the performance gap between TITAN and baselines also increases in the case of the denser job workload. Tab. 4 reports the percentage of jobs that receive the significant JCT reduction and increase compared to SRTF over different workload densities. Exceeding 50% jobs experience at most 10% JCT increase or decrease compared to SRTF. We ignore these jobs with relatively small JCT changes, and only consider at least 10% JCT reduction and increase as significant JCT changes. We find that around 20% jobs get benefits from TITAN, and only a small proportion of jobs are delayed by TITAN. Overall, preliminary results show promising performance of our proposed TITAN to manage FM fine-tuning workloads.

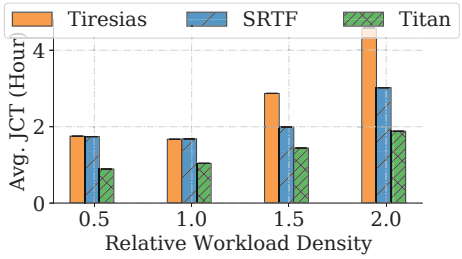

Figure 5: Performance across various workload density.

6 CONCLUSION AND FUTURE WORKS

This paper presents TITAN, a scheduling system tailored for FM fine-tuning workloads in GPU clusters. We leverage a lookup table for training time estimation and configuration identification. Whereby the multi-task adaptivity of FM fine-tuning, we propose the task merger to improve the cluster-wide job throughput. Besides, we design pipeline switch to address the expensive overhead of initializing a new job.

As future works, we aim to extend TITAN from the following perspectives. First, data processing can impact FM fine-tuning in terms of job throughput. A poor policy makes data processing a bottleneck of FM fine-tuning. Furthermore, the task merger requires to combine several datasets into one and complicates the data processing. Existing GPU clusters can offer abundant CPU resources, and data processing can be isolated from gradient computation. Hence we can allocate certain CPU resources to process data samples ahead of FM fine-tuning. This is one optimization direction to accelerate FM fine-tuning workloads.

Second, the billion-scale parameter size of FMs and intensive fine-tuning requests can result in formidable storage consumption. It is critical but challenging to reduce the storage cost. Parameter-efficient fine-tuning approaches [20, 28] aim to tune much fewer trainable parameters while maintaining a satisfactory performance. We can incorporate these approaches into TITAN to reduce the storage overhead.

Third, our current pipeline switch only supports to overlap parameter load and gradient computation across machines. Indeed, PipeSwitch [2] has demonstrated the feasibility of context switch in a single GPU. We can also asynchronously execute the parameter load and gradient computation in a single GPU to further amortize the overhead and improve the resource utilization.

Fourth, fine-tuning FMs often requires pipeline parallelism with certain idle machines (Fig. 3). We can fill certain FM inference workloads into these idle blocks to improve the resource utilization. The management of both FM inference and fine-tuning workloads will bring a new challenge.

We will continue to improve our system design, and make the above potential extensions. Moreover, we will develop a

prototype atop Kubernetes [6], and deploy it in a production environment to validate its practicability.

7 ACKNOWLEDGEMENT

We thank our Shepherd Dr. Lin Wang and anonymous reviewers for their valuable comments. This study is supported under the RIE2020 Industry Alignment Fund – Industry Collaboration Projects (IAF-ICP) Funding Initiative, as well as cash and in-kind contributions from the industry partner(s).

REFERENCES

[1] Alexei Baevski, Yuhao Zhou, Abdelrahman Mohamed, and Michael Auli. 2020. wav2vec 2.0: A framework for self-supervised learning of speech representations. *Advances in Neural Information Processing Systems* 33 (2020), 12449–12460.

[2] 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)*.

[3] 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. https://github.com/facebookresearch/fairscale.

[4] Edmon Begoli, Seung-Hwan Lim, and Sudarshan Srinivasan. 2021. Performance Profile of Transformer Fine-Tuning in Multi-GPU Cloud Environments. In *2021 IEEE International Conference on Big Data (Big Data)*. IEEE, 3095–3100.

[5] 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 *Advances in Neural Information Processing Systems (NeurIPS '20)*.

[6] 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).

[7] Xinlei Chen, Haoqi Fan, Ross Girshick, and Kaiming He. 2020. Improved baselines with momentum contrastive learning. *arXiv preprint arXiv:2003.04297* (2020).

[8] Zhilu Chen, Jing Wang, Haibo He, and Xinming Huang. 2014. A fast deep learning system using GPU. In *2014 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 1552–1555.

[9] Alexandra Chronopoulou, Matthew E Peters, and Jesse Dodge. 2021. Efficient hierarchical domain adaptation for pretrained language models. *arXiv preprint arXiv:2112.08786* (2021).

[10] 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)*.

[11] 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).

[12] Ahmed Elnaggar, Michael Heinzinger, Christian Dallago, Ghalia Rihawi, Yu Wang, Llion Jones, Tom Gibbs, Tamas Feher, Christoph Angerer, Martin Steinegger, et al. 2020. ProtTrans: towards cracking the language of Life's code through self-supervised deep learning and high performance computing. *arXiv preprint arXiv:2007.06225* (2020).

[13] Hugging Face. 2022. https://huggingface.co/autotrain.

[14] William Fedus, Barret Zoph, and Noam Shazeer. 2021. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity.

[15] Xin Geng, Haitao Zhang, Zhengyang Zhao, and Huadong Ma. 2020. Interference-aware parallelization for deep learning workload in GPU cluster. *Cluster Computing* 23, 4 (2020), 2689–2702.

[16] Andrea Gesmundo and Jeff Dean. 2022. muNet: Evolving Pretrained Deep Neural Networks into Scalable Auto-tuning Multitask Systems. *arXiv preprint arXiv:2205.10937* (2022).

[17] 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 *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI '19)*.

[18] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. 2020. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 9729–9738.

[19] Liang He, Shizhuo Zhang, Lijun Wu, Huanhuan Xia, Fusong Ju, He Zhang, Siyuan Liu, Yingce Xia, Jianwei Zhu, Pan Deng, et al. 2021. Pre-training co-evolutionary protein representation via a pairwise masked language model. *arXiv preprint arXiv:2110.15527* (2021).

[20] 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 *International Conference on Machine Learning*. PMLR, 2790–2799.

[21] Qinghao Hu, Peng Sun, Shengen Yan, Yonggang Wen, and Tianwei Zhang. 2021. Characterization and Prediction of Deep Learning Workloads in Large-Scale GPU Datacenters. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '21)*.

[22] Changho Hwang, Taehyun Kim, Sunghyun Kim, Jinwoo Shin, and KyoungSoo Park. 2021. Elastic Resource Sharing for Distributed Deep Learning. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI '21)*.

[23] 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 *2019 USENIX Annual Technical Conference (USENIX ATC '19)*.

[24] Junyang Lin, Rui Men, An Yang, Chang Zhou, Ming Ding, Yichang Zhang, Peng Wang, Ang Wang, Le Jiang, Xianyan Jia, et al. 2021. M6: A chinese multimodal pretrainer. *arXiv preprint arXiv:2103.00823* (2021).

[25] Ze Liu, Han Hu, Yutong Lin, Zhuliang Yao, Zhenda Xie, Yixuan Wei, Jia Ning, Yue Cao, Zheng Zhang, Li Dong, et al. 2022. Swin transformer v2: Scaling up capacity and resolution. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 12009–12019.

[26] 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 *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 10012–10022.

[27] 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 '20)*.

[28] 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).

[29] 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 '18)*.

[30] 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 '21)*.

[31] 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.

[32] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, Peter J Liu, et al. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.* 21, 140 (2020), 1–67.

[33] Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. 2020. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 3505–3506.

[34] Carlos Riquelme, Joan Puigcerver, Basil Mustafa, Maxim Neumann, Rodolphe Jenatton, André Susano Pinto, Daniel Keysers, and Neil Houlsby. 2021. Scaling vision with sparse mixture of experts. *Advances in Neural Information Processing Systems* 34 (2021), 8583–8595.

[35] Alexander Rives, Joshua Meier, Tom Sercu, Siddharth Goyal, Zeming Lin, Jason Liu, Demi Guo, Myle Ott, C Lawrence Zitnick, Jerry Ma, et al. 2021. Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences. *Proceedings of the National Academy of Sciences* 118, 15 (2021), e2016239118.

[36] Justyna Sarzynska-Wawer, Aleksander Wawer, Aleksandra Pawlak, Julia Szymanowska, Izabela Stefaniak, Michal Jarkiewicz, and Lukasz Okruszek. 2021. Detecting formal thought disorder by deep contextualized word representations. *Psychiatry Research* 304 (2021), 114135.

[37] 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).

[38] Asa Cooper Stickland and Iain Murray. 2019. Bert and pals: Projected attention layers for efficient adaptation in multi-task learning. In *International Conference on Machine Learning*. PMLR, 5986–5995.

[39] Yu Sun, Shuohuan Wang, Shikun Feng, Siyu Ding, Chao Pang, Junyuan Shang, Jiaxiang Liu, Xuyi Chen, Yanbin Zhao, Yuxiang Lu, et al. 2021. Ernie 3.0: Large-scale knowledge enhanced pre-training for language understanding and generation. *arXiv preprint arXiv:2107.02137* (2021).

[40] Colin Unger, Zhihao Jia, Wei Wu, Sina Lin, Mandeep Baines, Carlos Efrain Quintero Narvaez, Vinay Ramakrishnaiah, Nirmal Prajapati, Pat McCormick, Jamaludin Mohd-Yusof, et al. 2022. Unity: Accelerating {DNN} Training Through Joint Optimization of Algebraic Transformations and Parallelization. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. 267–284.

[41] Qizhen Weng, Wencong Xiao, Yinghao Yu, Wei Wang, Cheng Wang, Jian He, Yong Li, Liping Zhang, Wei Lin, and Yu Ding. 2022. MLaaS in the Wild: Workload Analysis and Scheduling in Large-Scale Heterogeneous GPU Clusters. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI '22)*.

[42] 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 *USENIX Symposium on Operating Systems Design and Implementation*.

[43] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. *Advances in neural information processing systems* 32 (2019).

[44] Geoffrey X. Yu, Yubo Gao, Pavel Golikov, and Gennady Pekhimenko. 2021. Habitat: A Runtime-Based Computational Performance Predictor for Deep Neural Network Training. In *2021 USENIX Annual Technical Conference (USENIX ATC '21)*.