TESSERAE: SCALABLE MIGRATION POLICIES FOR DEEP LEARNING WORKLOADS

Anonymous authors
Paper under double-blind review

ABSTRACT

Training deep learning (DL) models has become a dominant workload in data-centers. Existing schedulers allocate computational resources to running jobs with the goal of optimizing objectives such as makespan, fairness and average job completion time. In addition, they often incorporate placement policies that determine where jobs are assigned within the cluster to improve overall resource utilization. However, as model sizes continue to grow, migration costs have become a major bottleneck for cluster schedulers. Our key insight is that migration problems during scheduling can be formulated as graph-matching problems. Building on this observation, we design novel migration policies that can minimize the number of job migrations during scheduling. We integrate these policies into existing schedulers and demonstrate how our design enables a scalable and efficient GPU cluster scheduler. Experimental results show that TESSERAE improves average JCT by up to $1.36\times$ and reduces the number of migrations by up to $2.68\times$ compared to existing approaches.

1 INTRODUCTION

Deep learning training, including pre-training, fine-tuning, and post-training, constitutes a substantial workload in modern data center environments Wang et al. (2018); Lin et al. (2019); Cobbe et al. (2021); Wang et al. (2022); Russakovsky et al. (2015); Sap et al. (2019). To enhance the performance of these training tasks, several domain-specific schedulers have been proposed Gu et al. (2019); Mahajan et al. (2020); Narayanan et al. (2020; 2021a); Hu et al. (2023); Zheng et al. (2023) which cater to the unique characteristics of these jobs. These schedulers are typically round-based, operating at fixed intervals (e.g., every 6 minutes), selecting jobs to execute from a pool of runnable jobs. To enforce scheduling decisions, they rely on job preemption and migration.

Many of these schedulers Gu et al. (2019); Mahajan et al. (2020); Qiao et al. (2021); Zhao et al. (2022); Zheng et al. (2023); Hu et al. (2023) were developed when model sizes were relatively small. Thus, these schedulers typically assume that preemption and migration costs are negligible and do not account for them when scheduling. However, with the rapid growth in model complexity and size Dubey et al. (2024); Liu et al. (2024); Team et al. (2025a), the cost of preemption has increased significantly. For instance, the cost to migrate models presented in Tiresias Gu et al. (2019), a DL scheduler proposed in 2019, ranges from 20-50 seconds, while migrating popular LLM workloads can take 40-60 minutes; as an example, it takes 42 minutes to checkpoint the model states of MT-NLG Smith et al. (2022). This shift in the training workload underscores that schedulers *should take the cost of preemption and restarts into careful consideration*. However, existing schedulers primarily optimize for specific objectives such as resource utilization, fairness, or makespan Xiao et al. (2018); Gu et al. (2019); Mahajan et al. (2020); Zheng et al. (2023), and modifying each of these schedulers to account for migration cost is challenging. This is because accounting for migration costs would require modifying the objective function and can potentially interfere with the scheduler's properties. Given these diverse objectives and strategies, we need a migration policy that can be seamlessly integrated into different scheduling frameworks without requiring any modifications.

To overcome this challenge, we introduce TESSERAE, a pluggable module that is compatible with a broad range of scheduling policies. As shown in Figure 3 (§3.2), TESSERAE works on the output of the scheduling policy and introduces a migration plan to minimize migrations. We show that TESSERAE seamlessly adapts to variations in scheduling policies, workloads, and training parameters.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

With TESSERAE our key insight is that the migration problem can be formulated as a *graph matching* problem, which can be efficiently solved using established algorithms such as the Hungarian Algorithm Kuhn (1955). Precisely, to minimize job migration, we construct a graph $G^M = (V_1^M, V_2^M, E^M)$ using placement information from consecutive scheduling rounds. Here, V_1^M represents current placement plan and V_2^M represents the placement plan after migration. Each edge $e^M = (x, y)$ is assigned a weight w_{e^M}, which quantifies the migration cost incurred when the jobs assigned to node x in the original placement are reassigned to node y in the next round. We show that migration problems arising during scheduling can be formulated within this unified framework, and the resulting graph-matching problem can be solved efficiently even for large-scale clusters.

To evaluate TESSERAE, we integrate it with existing scheduling policies such as Tiresias Gu et al. (2019), Themis Mahajan et al. (2020), and Gavel Narayanan et al. (2020). We evaluate TESSERAE using multiple workloads derived from prior schedulers Jeon et al. (2019); Narayanan et al. (2020); Zheng et al. (2023). Our experiments first demonstrate that TESSERAE achieves substantially higher throughput compared to an existing heuristic-based migration policy, improving average JCT by up to $1.36\times$ and reducing the number of migrations by $2.68\times$. Next, we show that TESSERAE integrates seamlessly with various scheduling policies, Finally, we evaluate the scalability of TESSERAE and find that it remains highly efficient as the cluster size grows, taking less than 3 seconds even on a cluster with 1200 GPUs.

In summary, we make the following contributions:

- We first show that as the size of large language models increases, migration costs during scheduling cause substantial slowdowns and reduce GPU utilization. (§2.4)
- We introduce a novel graph-based formulation for the migration policy, demonstrating that leveraging a weighted bipartite graph matching algorithm effectively reduces the number of migrations during scheduling. (§3.2)
- Based on extensive experimental results, we demonstrate that TESSERAE achieves up to a $1.36\times$ improvement in average JCT, reduces migration count by $2.68\times$, and improves the worst-case finish-time fairness ratio by $1.57\times$. (§5)

2 BACKGROUND & MOTIVATION

In this section, we begin by providing an overview of deep learning scheduling in GPU clusters in §2.1. Next, we highlight the migration overhead and its impact on performance in §2.2, followed by a discussion of how scheduling policies are augmented by placement constraints to improve cluster utilization in §2.3. Furthermore, we discuss the challenges arising from current deep learning scheduling in §2.4. Finally, we summarize the primary goal of this paper in §2.5.

2.1 DL SCHEDULING

A multitude of studies Xiao et al. (2018); Gu et al. (2019); Mahajan et al. (2020); Xiao et al. (2020); Zhao et al. (2022); Zheng et al. (2023); Hu et al. (2023) have developed schedulers for DL workloads, targeting diverse optimization goals. In DL scheduling, schedulers assign priorities to a set of jobs with the goal of optimizing a specific performance metric. For example, Gandiva Xiao et al. (2018) focuses on maximizing cluster utilization, whereas Tiresias Gu et al. (2019) is designed to reduce average job completion time. Similarly, Pollux Qiao et al. (2021) aims to optimize goodput, which is a metric combining system throughput and statistical efficiency.

To generalize a broad range of existing scheduling policies, Gavel Narayanan et al. (2020) introduces a linear programming framework that unifies optimization objectives, placement strategies, and packing policies within a single framework.

2.2 MIGRATION OVERHEAD

Job migrations can cause substantial slowdowns and degrade GPU utilization, particularly in large language model workloads. Figure 1 left illustrates the associated warmup and checkpointing overheads for GPT3-XL and GPT3-3B. Figure 1 right shows that round-based schedulers, including Tiresias Gu et al. (2019) and Gavel Narayanan et al. (2020), incur considerable migration overhead.

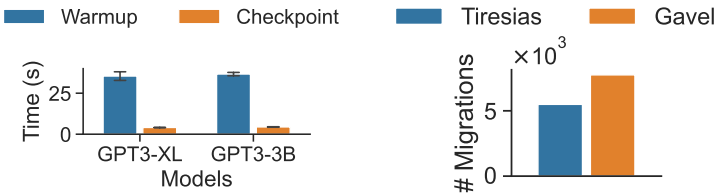

Figure 1: **Migration Overhead:** (left) The warmup and checkpoint time for GPT3 models; (right) The number of job migrations of Tiresias and Gavel. The warmup time is the duration from entering the command to the start of the first iteration. The checkpoint overhead refers to the total time spent loading and saving checkpoints. The number of migrations is measured during the scheduling of 900 jobs on a 128-GPU cluster, where Gavel applies job packing to enhance cluster utilization.

In particular, while packing policies are designed to improve GPU utilization, they also increase the number of migrations during scheduling, thereby amplifying the overall migration overhead. Therefore, minimizing job migrations is crucial for round-based schedulers.

2.3 Placement Policies

Several schedulers Xiao et al. (2018); Gu et al. (2019); Mahajan et al. (2020) have highlighted the importance of placement constraints on the throughput achieved by jobs. The placement constraints determine which GPUs in the cluster are used to run the chosen jobs. For example, Gandiva Xiao et al. (2018) depicted that distributed jobs prefer consolidated placements, *i.e.*, jobs requiring more than one GPU prefer that both GPUs exist on the same machine. Subsequently, Tiresias Gu et al. (2019) highlighted that certain jobs are more placement-sensitive than others, and cluster utilization can be improved by considering the properties of the individual job for performing placement.

2.4 Challenges

To reduce the migration cost, several heuristic methods have been proposed Xiao et al. (2018); Narayanan et al. (2020). However, existing migration approaches often fail to minimize the number of migrations between adjacent rounds in round-based schedulers. For instance, Gavel's migration policy states that job migration is unnecessary if a job uses the same GPU in two consecutive placement rounds; otherwise, migration is required. However, in Figure 2, we observe that Gavel's migration policy Narayanan et al. (2020) results in three job migrations, whereas the optimal solution requires none.

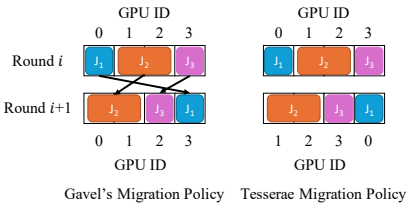

Figure 2: **Performance Limitations:** (left) Gavel's policy migrates three jobs between two nearby plans. J_2 employs data parallelism in both rounds; (right) this overhead can be avoided, and throughput can be improved, by remapping GPU IDs.

2.5 Goals

These challenges underscore the need for a migration policy that minimizes job migrations in round-based schedulers. In the following paragraphs, we describe the desirable characteristics of such a policy.

Improved Performance. A primary objective of the migration policy should be to reduce the number of migrations and enhance hardware utilization. In addition, it should outperform existing migration policy (*e.g.*, Gavel Narayanan et al. (2020)).

Compatibility. The framework should also be compatible with existing scheduling policies, including Tiresias Gu et al. (2019), Themis Mahajan et al. (2020), and Gavel Narayanan et al. (2020). Users should be able to employ their preferred scheduler while leveraging the proposed policy solely to reduce migration overhead, thereby facilitating broader adoption.

Scalability. As deep learning clusters continue to scale and accommodate an increasing number of jobs Weng et al. (2022); Hu et al. (2024); Xu et al. (2025); Subramanya et al. (2025), the proposed policy is expected to effectively manage scheduling at large scale.

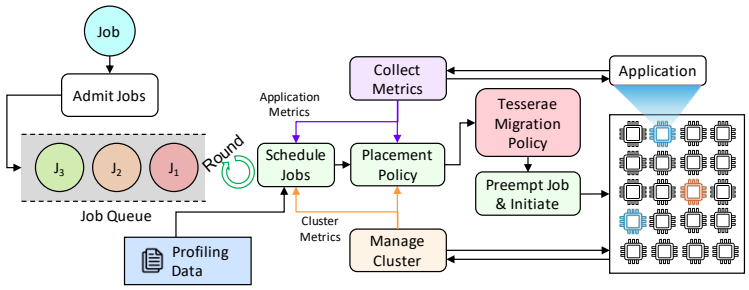

Figure 3: **Overview of TESSERAE system architecture:** all components of the system and their interactions. We design TESSERAE as the migration policy for the whole system.

Listing 1: Scheduling Framework

```
1  placed_jobs ← []
2  active_jobs ← all submitted and unfinished jobs
3  Sort active_jobs based on priority
4  num_gpus_remain ← number of total GPUs
5  while num_gpus_remain > 0 do
6      j ← active jobs with highest priority
7      Remove job j from active_jobs
8      if we fail to place job j then
9          continue
10     num_gpus_remain -= GPUs required by job j
11     placed_jobs.append(job j)
12 Develop job migration strategy
13 Preempt running jobs and initiate new ones within the cluster.
```

3 TESSERAE

3.1 OVERVIEW

To tackle the challenges outlined in §2, we present TESSERAE. We present a high-level design overview of TESSERAE in Figure 3. Next, we discuss a concrete example of how the TESSERAE scheduler can apply policies such as migration-awareness (Listing 1).

Scheduling in deep learning clusters typically occurs in rounds Narayanan et al. (2020); Hu et al. (2023); Zheng et al. (2023) (each round is a few minutes long). At the end of each round, the cluster scheduler is invoked to determine the set of jobs that should run for the next round. In every round, the job scheduling policy sorts active jobs by their priority. The priority for jobs is determined based on the underlying scheduling policy, such as arrival time for FIFO or LAS for Tiresias Gu et al. (2019), etc. Given the list of sorted jobs, a placement plan needs to constructed. However, we note that placement (in line 8) can fail if there are fewer available GPUs than the number needed by the jobs or some placement constraint that cannot be satisfied, *e.g.*, a multi-GPU job cannot find a consolidated placement. After these steps, given the significant migration overheads, we employ TESSERAE, our proposed migration algorithm, which minimizes the number of migrations between successive rounds based on the formulated placement plan (in line 12). Finally, we preempt the running jobs and schedule new ones across the cluster according to the migration strategy (in line 13).

3.2 MIGRATION ALGORITHM

In this section, we present the migration algorithm. We begin by formally defining the migration problem. As a first step, we first define job migration in Definition 1, which serves as the foundation for the subsequent algorithm design.

Definition 1 (Job Migration) *A job is migrated between consecutive rounds if it is present in both rounds and utilizes different sets of GPUs within the cluster.*

Algorithm 2: TESSERAE Migration Policy

Input: placement_plan for round i: P_i, placement_plan for round $i+1$: P_{i+1}.
Output: migration plan M

1 $C \leftarrow []$, $M \leftarrow []$
2 Remove all jobs j in P_i and P_{i+1}, where job j satisfies $j \in ((P_i \cup P_{i+1}) - (P_i \cap P_{i+1}))$
3 **foreach** *placement plan of node k,* $P_{i,k} \in P_i$ **do**
4 **foreach** *placement plan of node* ℓ, $P_{i+1,\ell} \in P_{i+1}$ **do**
5 $C_{k,\ell}, M_{k,\ell} \leftarrow$ Node-Level Matching($P_{i,k}$, $P_{i+1,\ell}$)
6 `match_solution` $\leftarrow$ Hungarian Algorithm(C)
7 **return** M`[match_solution]`

Algorithm 3: Node-Level Matching

Input: placement_plan for round i of node k: $P_{i,k}$, placement_plan for round $i+1$ of node ℓ: $P_{i+1,\ell}$.
Output: migration cost $C_{k,\ell}$, migration plan $M_{k,\ell}$

1 $C_{\text{sum}} \leftarrow 0$, $C \leftarrow [0]_{k_\ell \times k_\ell}$, where k_ℓ is the number of GPUs in the node.
2 **foreach** *GPU* $u \in P_{i,k}$ **do**
3 **foreach** *GPU* $v \in P_{i+1,\ell}$ **do**
4 $JS_u \leftarrow$ job sets on GPU u and $JS_v \leftarrow$ job sets on GPU v
5 **foreach** *job* $j \in JS_u \cup JS_v$ **do**
6 **if** *job* $j \in ((JS_u \cup JS_v) - (JS_u \cap JS_v))$ **then**
7 $C_{u,v} \leftarrow C_{u,v} + 1/(2 \cdot \texttt{num_GPUs}(j))$
8 $C_{\text{sum}}, M_{k,\ell} \leftarrow$ Hungarian Algorithm(C)
9 **return** $C_{\text{sum}}, M_{k,\ell}$

Note that if a job does not appear in both round i and round $i+1$, it is not considered to have been migrated. Next, we present the problem we study in this work below:

Definition 2 (Job Migration Minimization) *Given two placement plans, P_i from round i and P_{i+1} from round $i+1$, the job migration minimization problem aims to determine a migration strategy that reduces the number of job migrations between these rounds while still meeting the constraints of consolidated placement.*

Furthermore, we introduce our algorithm, which effectively reduces migration costs in round-based deep learning schedulers. Finally, we analyze the characteristics and properties of the proposed migration algorithm in §3.3.

Migration Minimization Algorithm. We design a migration algorithm based on the following observation: say we have two placement plans from round i and round $i+1$, as $\{(0,1),(1,2),(2,2),(3,4)\}$ and $\{(0,4),(1,1),(2,2),(3,2)\}$, respectively. In each tuple, the first number represents the GPU ID and the second number represents the job ID. We assume GPUs in the cluster are homogeneous. Although the two placement plans differ, job migrations are unnecessary as we can simply rename the GPU IDs through the following reassignments: $0 \rightarrow 1$, $1 \rightarrow 3$, and $3 \rightarrow 0$. Therefore, it is unnecessary to migrate any jobs.

In view of this, we draw inspiration from established assignment problems and show our migration algorithm in Algorithm 2. In particular, the algorithm first removes any jobs that are not present in both rounds concurrently (line 2 of Algorithm 2). Next, the algorithm computes the migration cost for every pair of nodes, where each pair consists of one node k from round i and one node ℓ from round $i+1$ (lines 3-5 of Algorithm 2). We use Algorithm 3 to compute the migration cost between node pairs. Specifically, we first calculate the migration cost for each GPU pair across two given nodes (lines 2–7 of Algorithm 3). The cost is determined by the number of GPUs required by each job, as the migration cost is amortized across all processes in a multi-GPU job. Moreover, each move-in or move-out operation incurs a cost of 0.5 per job, which is why we multiply the cost by $1/2$ in line 7

of Algorithm 3. Finally, we apply the Hungarian algorithm Kuhn (1955) to determine the optimal GPU-level migration between node pairs.

After determining the migration cost for every pair of nodes, we can then apply the Hungarian algorithm Kuhn (1955) again to derive a migration plan that minimizes the total number of migrations between the two placement plans (lines 6-7 of Algorithm 2). In conclusion, Algorithm 2 applies the Hungarian algorithm twice: the first application computes GPU-level migration costs between node pairs, and the second determines the optimal migration cost across the entire cluster. This two-step process preserves the placement constraint, as distributed jobs typically prefer consolidated placements. We illustrate the complete process using the example shown in §B.

Running Time. We assume that the cluster has k_c nodes and each node contains k_ℓ GPUs. The Hungarian algorithm Kuhn (1955) is widely used to address the assignment problem, and the time complexity of the Hungarian algorithm is $\mathcal{O}(n^3)$, where n is the number of tasks in the assignment problems. Therefore, the time complexity of Algorithm 3 is $\mathcal{O}(k_\ell^3)$ and the time complexity of Algorithm 2 is $\mathcal{O}(k_c^2 k_\ell^3 + k_c^3)$. Since the time complexity of Algorithm 2 is independent of the number of queued jobs, it remains efficient even for large-scale cluster scheduling. In §5, we evaluate the overhead of TESSERAE by increasing the number of GPUs in the cluster and compare it with the scheduling overhead. The empirical results closely align with the analytical findings presented here.

3.3 DISCUSSION

Extensibility. TESSERAE is compatible with a wide range of existing round-based scheduling policies, as it does not alter the scheduling or placement strategy but simply updates the assigned GPU IDs based on the previous round's scheduling results. In addition, Algorithm 3 constructs the migration cost matrix C based on GPU mappings, enabling our migration policy to be readily applied to job packing strategies (*e.g.*, Gavel Narayanan et al. (2020), Lucid Hu et al. (2023)) as well.

Performance. Gavel's migration policy Narayanan et al. (2020) specifies that job migration is unnecessary when a job continues to use the same GPU across nearby rounds. However, Gavel's migration policy represents only one feasible solution to the matching problem formulated in Algorithm 3 with migration cost C. Since the Hungarian algorithm guarantees the optimal solution to this matching problem, our approach consistently achieves better results than Gavel's migration policy.

Job Placement. If the jobs from placement plan P_i and P_{i+1} in rounds i and $i+1$ respectively are consolidated, Algorithm 2 will ensure that all jobs remain in a consolidated setting. Because Algorithm 3 performs GPU matching at the node level, it ensures that the processes of a distributed job either remain on the same node or are collectively relocated to other nodes.

4 IMPLEMENTATION

TESSERAE is built on Blox Agarwal et al. (2023) using Python in approximately 2000 lines of code for both real cluster mode and simulation mode. We have incorporated TESSERAE into the Blox framework by developing it as a migration policy. We use Scipy Virtanen et al. (2020) to generate the migration plan for each round outlined in §3.2. Moreover, we employ gRPC grp (2024) for message communication between schedulers and applications. Lastly, we use CUDA-MPS MPS (2023) to run multiple jobs on the same GPU.

Profiling. In this paper, we adopt a straightforward approach to measure the migration cost of each job. Specifically, we run all possible jobs offline on the NERSC Perlmutter ner (2024) to record their warm-up times as well as the time spent saving and loading checkpoints.

Schedulers. Following prior round-based schedulers Hu et al. (2023); Narayanan et al. (2020); Xiao et al. (2020); Zheng et al. (2023), all evaluated schedulers in this paper make scheduling decisions every six minutes, taking the following steps: for all active jobs, it develops placement plans based on the offline profiling data and the given scheduling method. After generating the placement plans, the scheduler signals all nodes to stop existing jobs and start new ones. Note that we only preempts the job after the job finishes the current iteration. Then, the scheduler pauses for six minutes to gather updated metrics, such as service attained and training progress. Following that, it makes decisions for the next round.

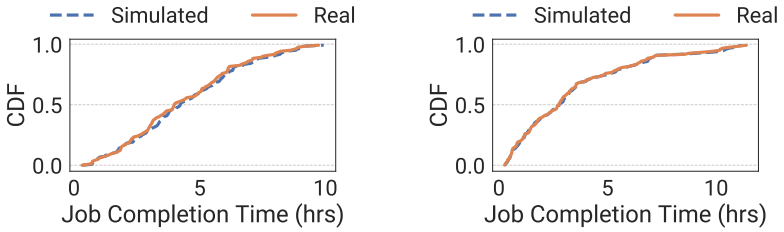

Figure 4: **Comparison of CDFs between cluster and simulator:** (left) The CDF for Tiresias$^+$; (right) The CDF for FTF$^+$. We depict the CDF for Tiresias$^+$ and FTF$^+$ obtained from physical experiments compared with simulated results. The results demonstrate our simulator's low fidelity.

Applications. All applications are implemented using PyTorch Paszke et al. (2019). For all models listed in Table 2 of §A, we classify the models into two groups depending on the presence of Transformer layers Vaswani et al. (2017), leading to models being divided into two groups: (1) ResNet-50, VGG-19, DCGAN, and PointNet; (2) GPT3-Medium, GPT3-XL, and GPT3-3B. For the first group, distributed training is conducted using PyTorch DDP. In contrast, for the second group, 3D parallelism training is implemented using Megatron-LM Shoeybi et al. (2019). The parallelism strategy can be adjusted before launching jobs on the cluster. Furthermore, for all evaluated schedulers, we impose a limit of two concurrent models per GPU, as packing more than two jobs generally yields negligible additional benefits Narayanan et al. (2020); Hu et al. (2023).

5 Evaluation

In this section, we evaluate Tesserae and focus on the following aspects: (i) The effectiveness of Tesserae; (ii) The compatibility of Tesserae; (iii) The scalability of Tesserae. Due to space constraints, we present our ablation study in §D.

5.1 Experimental Setup

Testbed. We conduct cluster experiments using 32 GPUs across 8 nodes on NERSC Perlmutter ner (2024). Each node has four 40 GB NVIDIA A100 (Ampere) GPUs, 256 GB DDR4 DRAM, and a single AMD EPYC 7763 (Milan) CPU. We also run simulation experiments on a Cloudlab server Duplyakin et al. (2019). The server contains 126 GB memory and two Intel E5-2630 v3 8-core CPUs at 2.40 GHz. For the simulation experiments, we use 32 GPUs to run a 120-job workload and 128 GPUs to run a 900-job workload.

Traces. To evaluate the schedulers, we use traces introduced by two prior works Shockwave Zheng et al. (2023) and Gavel Narayanan et al. (2020). Table 2 in §A lists the workload details including models, dataset, and batch size. Our default trace is similar to Shockwave Zheng et al. (2023), we follow the same setting as Shockwave by setting the probability of generating Small, Medium, Large, and Extra Large jobs to be 0.72, 0.2, 0.05, and 0.03. We also set the probability of generating 1-GPU, 2-GPU, 4-GPU, and 8-GPU jobs to be 0.6, 0.3, 0.09, and 0.01 to align with those used in Shockwave. The job arrival rate is configured at 80 jobs per hour, consistent with the settings used in prior work. We show how Tesserae's benefits change across workloads by using an additional trace similar to the one used in Gavel Narayanan et al. (2020) in §D. By default, we use a 120 jobs trace for physical experiments and a 900 jobs trace for simulated experiments, both with a job arrival rate of 80 jobs per hour.

Baselines. We compare Tesserae with the heuristic migration policy proposed in Gavel Narayanan et al. (2020) by integrating it with the following scheduling policies: (i) Tiresias Gu et al. (2019); (ii) FTF Mahajan et al. (2020); (iii) Gavel Narayanan et al. (2020); (iv) Gavel-FTF Narayanan et al. (2020). Tiresias Gu et al. (2019) uses 2D-LAS to perform fair sharing of the cluster. FTF employs a greedy algorithm to minimize the finish-time fairness (FTF) metric. Gavel Narayanan et al. (2020) formulates scheduling as an optimization-based framework that supports the LAS policy while incorporating job packing. In addition, we integrate Tesserae with Gavel-FTF to study how Tesserae affects fairness metrics. Gavel-FTF Narayanan et al. (2020) performs job packing and solves an optimization problem associated with the FTF metric.

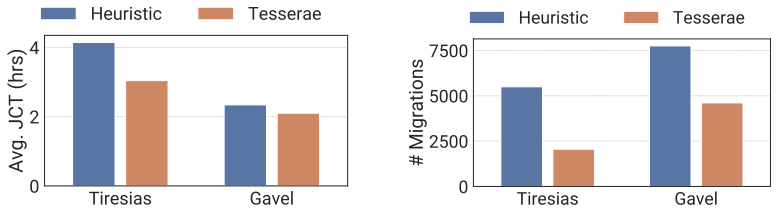

Figure 5: **Evaluating TESSERAE against heuristic solution:** (left) Average JCT of Tiresias and Gavel under TESSERAE and the heuristic migration policy; (right) Number of migrations during scheduling when using TESSERAE and the heuristic migration policy with Tiresias and Gavel as the scheduling policies.

Configuration. Following prior works Zheng et al. (2023); Narayanan et al. (2020), we set the round duration to six minutes. TESSERAE is designed as a modular migration policy that can work with any round-based scheduling policy. We use Tiresias to denote the configuration that employs the Tiresias scheduling policy together with the migration policy proposed in Gavel Narayanan et al. (2020). In contrast, Tiresias$^+$ represents the combination of the Tiresias scheduling policy with our proposed TESSERAE migration policy. The same notation is applied to other scheduling policies as well.

Performance Metrics. Following prior DL schedulers, we report standard metrics: the average job completion time (Avg. JCT) and the total time required to complete all jobs (Makespan). In addition, we evaluate the finish-time fairness (FTF) ratio to assess fairness, as well as the number of job migrations incurred when scheduling the entire workload.

5.2 END-TO-END REAL CLUSTER EXPERIMENTS

Due to lack of access to large scale production clusters, prior works Qiao et al. (2021); Narayanan et al. (2020); Jayaram Subramanya et al. (2023); Zheng et al. (2023); Hu et al. (2023); Zhao et al. (2022); Gu et al. (2019); Mahajan et al. (2020); Mohan et al. (2022); Weng et al. (2022) have used simulations to perform detailed study of large scale traces and compare different metrics. We follow a similar approach.

In this section, we verify that our simulation closely approximates runs on a real cluster. Since profiling can often have significant noise, we run profiling five different times, and during simulation, we choose one of them at random. To account for noise, we also run the simulation five different times. In Table 1, we show that our simulation shows the maximum average deviation for average JCT between the physical cluster and the simulation is 3.36%, and the maximum deviation for makespan is 2.05%.

In Figure 4, we also randomly sample one simulation run and present the CDF of JCTs to highlight the accuracy of our simulation. The average JCT deviation between the physical cluster and simulation results for FTF$^+$ is 1.42%, as shown in Figure 4. Both results above indicate that our simulation closely matches the experimental results from the real cluster.

Table 1: **The Fidelity of Simulator:** We run the simulation five times and report the mean deviation and standard deviation. The maximum deviation is 3.36%, highlighting that our simulator closely follows the real cluster.

Method	Avg. JCT (s)	Makespan (s)
Tiresias	$3.36\% \pm 0.46\%$	$2.05\% \pm 0.03\%$
FTF	$1.82\% \pm 0.24\%$	$0.99\% \pm 0.42\%$

5.3 END-TO-END RESULTS IN SIMULATION

To further evaluate TESSERAE, we conduct simulations using a large 900-job workload on a 128-GPU cluster. We first evaluate the effect of TESSERAE compared with the migration policy mentioned in Gavel Narayanan et al. (2020). Next, we investigate the compatibility of TESSERAE with various schedulers. Finally, we evaluate the scalability of TESSERAE by varying the number of active jobs and GPUs in the cluster, and by analyzing the overhead introduced by each policy.

Performance Comparison against Gavel. Figure 5 compares TESSERAE with the heuristic migration policy proposed in Gavel Narayanan et al. (2020), which skips job migration if a job is assigned to the same GPU in two consecutive placements. As shown in Figure 5 left, TESSERAE improves the Avg. JCT by up to $1.36\times$ compared to the heuristic migration algorithm. This improvement arises because TESSERAE reduces the number of migrations during scheduling by $2.68\times$, as illustrated in Figure 5 right. Reducing migration cost lowers queueing time and improves cluster-wide GPU utilization.

However, since Gavel already enhances GPU utilization through packing, the improvement from TESSERAE on average JCT is more significant for Tiresias than for Gavel. These results demonstrate the effectiveness of our migration policy in enhancing scheduling efficiency.

Compatibility with Other Schedulers. TESSERAE is implemented as a modular migration plugin on top of existing schedulers. We can use TESSERAE over existing schedulers without modifying the underlying scheduler. To show the modularity of TESSERAE and its impact on the underlying metrics, we implement TESSERAE over Gavel-FTF. To evaluate fairness, similar to prior work Mahajan et al. (2020) we use FTF ratio as a metric. FTF is defined as $\rho = \frac{T_s}{T_f}$, where T_s is the job completion time in a shared cluster and T_f is the job completion time in an isolated and fairly shared cluster. Figure 6 shows that TESSERAE improves the performance of Gavel-FTF, demonstrating its ability to achieve higher fairness compared to existing heuristic migration policies. To be specific, TESSERAE improves the worst-case finish-time fairness ratio by $1.57\times$. Since the fairness ratio is closely tied to job completion time, its improvement results from reducing the number of migrations, which lowers migration overhead and improves overall GPU utilization, thereby reducing the average JCT and improving the finish-time fairness ratio.

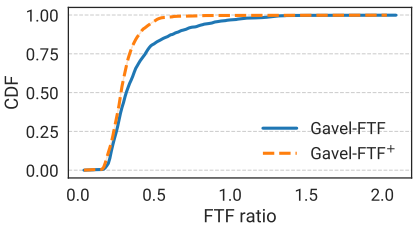

Figure 6: **Evaluation TESSERAE's fairness:** The CDF of Finish-time fairness (FTF) ratio Mahajan et al. (2020). The results indicate that TESSERAE enhances the fairness of Gavel-FTF compared with heuristic migration policy.

Scalability Analysis. To validate the correctness of our time complexity analysis in §3.2 and compare the overhead of migration policies against scheduling policies, we fix the number of active jobs and vary the number of GPUs in the cluster to evaluate the scalability of TESSERAE. In Figure 7, we observe that the running time of the scheduling policy remains largely independent of the number of GPUs in the cluster, as its time complexity primarily depends on the number of active jobs. However, as discussed in §3.2, the time complexity of TESSERAE scales with the number of GPUs in the cluster. The execution time of the migration policies increases from 0.8s to 2.66s as the number of GPUs grows from 600 to 1200. The results shown in Figure 7 match our discussion in §3.2. Nonetheless, the overhead introduced by TESSERAE remains small compared to the overall scheduling overhead, demonstrating its scalability.

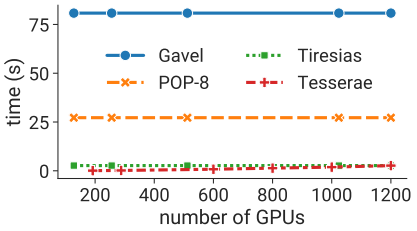

Figure 7: **Scalability of Schedulers:** This figure shows the overhead of TESSERAE compared with scheduling policies such as Tiresias Gu et al. (2019), Gavel Narayanan et al. (2020), and POP Narayanan et al. (2021a), as the number of GPUs used in each scheduling round increases.

6 RELATED WORK

GPU sharing is one possible way to improve GPU utilization, which has been deployed in GPU cluster schedulers Xiao et al. (2018); Weng et al. (2022); Xiao et al. (2020); Narayanan et al. (2020). Here, we introduce several GPU-sharing tools. Multi-Process Service MPS (2023) and Multi-Instance GPU MIG (2023) enable multiplex jobs on NVIDIA GPUs. PipeSwitch Bai et al. (2020) and Salus Yu & Chowdhury (2020) are designed for fast job switching and memory sharing. Zico Lim et al. (2021) efficiently shares memory among packing jobs without exceeding a given memory budget. MuxFlow Zhao et al. (2023) supports safe and fast space-sharing in the production cluster. Due to space constraints, we defer the discussion of additional related work to §E.

7 CONCLUSION

In this paper, we propose TESSERAE, a novel migration policy designed to complement existing scheduling policies by reducing job migrations during scheduling. Extensive experimental results demonstrate that TESSERAE consistently outperforms existing heuristic migration policies across a range of schedulers.

AI USE STATEMENT

Generative AI tools were used solely to improve the language, grammar, and readability of this paper. They were not used to develop the research ideas, design the methodology, conduct the experiments, or analyze the results. The authors reviewed all AI-assisted edits and take full responsibility for the content of the paper.

ETHICS STATEMENT

The authors have read and adhered to the ICLR Code of Ethics. This work uses only publicly available benchmark datasets and does not involve human subjects or the collection of private or personally identifiable information. We do not anticipate direct ethical risks arising from the methods or experiments presented in this paper.

REPRODUCIBILITY STATEMENT

We provide detailed descriptions of our system design, implementation, experimental setup, workloads, and configurations in the main paper and appendix. Upon acceptance, we will publicly release our source code and experiment scripts to facilitate reproduction of the results and further research.

REFERENCES

grpc. `https://grpc.io/`, 2024.

Nersc. `https://www.nersc.gov/`, 2024.

Saurabh Agarwal, Amar Phanishayee, and Shivaram Venkataraman. Blox: A modular toolkit for deep learning schedulers. *arXiv preprint arXiv:2312.12621*, 2023.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. Qwen technical report. *arXiv preprint arXiv:2309.16609*, 2023.

Zhihao Bai, Zhen Zhang, Yibo Zhu, and Xin Jin. {PipeSwitch}: Fast pipelined context switching for deep learning applications. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pp. 499–514, 2020.

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 7432–7439, 2020.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015.

Shubham Chaudhary, Ramachandran Ramjee, Muthian Sivathanu, Nipun Kwatra, and Srinidhi Viswanatha. Balancing efficiency and fairness in heterogeneous gpu clusters for deep learning. In *Proceedings of the Fifteenth European Conference on Computer Systems*, pp. 1–16, 2020.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Arnab Choudhury, Yang Wang, Tuomas Pelkonen, Kutta Srinivasan, Abha Jain, Shenghao Lin, Delia David, Siavash Soleimanifard, Michael Chen, Abhishek Yadav, et al. Mast: Global scheduling of ml training across geo-distributed datacenters at hyperscale. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 563–580, 2024.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Michael B Cohen, Yin Tat Lee, and Zhao Song. Solving linear programs in the current matrix multiplication time. *Journal of the ACM (JACM)*, 68(1):1–39, 2021.

Damai Dai, Chengqi Deng, Chenggang Zhao, RX Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Yu Wu, et al. Deepseekmoe: Towards ultimate expert specialization in mixture-of-experts language models. *arXiv preprint arXiv:2401.06066*, 2024.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Dmitry Duplyakin, Robert Ricci, Aleksander Maricq, Gary Wong, Jonathon Duerig, Eric Eide, Leigh Stoller, Mike Hibler, David Johnson, Kirk Webb, et al. The design and operation of {CloudLab}. In *2019 USENIX annual technical conference (USENIX ATC 19)*, pp. 1–14, 2019.

Juncheng Gu, Mosharaf Chowdhury, Kang G Shin, Yibo Zhu, Myeongjae Jeon, Junjie Qian, Hongqiang Liu, and Chuanxiong Guo. Tiresias: A {GPU} cluster manager for distributed deep learning. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, pp. 485–500, 2019.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Qinghao Hu, Meng Zhang, Peng Sun, Yonggang Wen, and Tianwei Zhang. Lucid: A non-intrusive, scalable and interpretable scheduler for deep learning training jobs. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pp. 457–472, 2023.

Qinghao Hu, Zhisheng Ye, Zerui Wang, Guoteng Wang, Meng Zhang, Qiaoling Chen, Peng Sun, Dahua Lin, Xiaolin Wang, Yingwei Luo, et al. Characterization of large language model development in the datacenter. *arXiv preprint arXiv:2403.07648*, 2024.

Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *Advances in neural information processing systems*, 32, 2019.

Changho Hwang, Taehyun Kim, Sunghyun Kim, Jinwoo Shin, and KyoungSoo Park. Elastic resource sharing for distributed deep learning. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, pp. 721–739, 2021.

Sam Ade Jacobs, Masahiro Tanaka, Chengming Zhang, Minjia Zhang, Shuaiwen Leon Song, Samyam Rajbhandari, and Yuxiong He. Deepspeed ulysses: System optimizations for enabling training of extreme long sequence transformer models. *arXiv preprint arXiv:2309.14509*, 2023.

Suhas Jayaram Subramanya, Daiyaan Arfeen, Shouxu Lin, Aurick Qiao, Zhihao Jia, and Gregory R Ganger. Sia: Heterogeneity-aware, goodput-optimized ml-cluster scheduling. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pp. 642–657, 2023.

Myeongjae Jeon, Shivaram Venkataraman, Amar Phanishayee, Junjie Qian, Wencong Xiao, and Fan Yang. Analysis of {Large-Scale}{Multi-Tenant}{GPU} clusters for {DNN} training workloads. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, pp. 947–960, 2019.

Zhihao Jia, Matei Zaharia, and Alex Aiken. Beyond data and model parallelism for deep neural networks. *Proceedings of Machine Learning and Systems*, 1:1–13, 2019.

Mandar Joshi, Eunsol Choi, Daniel S Weld, and Luke Zettlemoyer. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. *arXiv preprint arXiv:1705.03551*, 2017.

Vijay Anand Korthikanti, Jared Casper, Sangkug Lym, Lawrence McAfee, Michael Andersch, Mohammad Shoeybi, and Bryan Catanzaro. Reducing activation recomputation in large transformer models. *Proceedings of Machine Learning and Systems*, 5:341–353, 2023.

Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

Mark Lee, Tom Gunter, Chang Lan, John Peebles, Hanzhi Zhou, Kelvin Zou, Sneha Bangalore, Chung-Cheng Chiu, Nan Du, Xianzhi Du, et al. Axlearn: Modular large model training on heterogeneous infrastructure. *arXiv preprint arXiv:2507.05411*, 2025.

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

Mingzhen Li, Wencong Xiao, Hailong Yang, Biao Sun, Hanyu Zhao, Shiru Ren, Zhongzhi Luan, Xianyan Jia, Yi Liu, Yong Li, et al. Easyscale: Elastic training with consistent accuracy and improved utilization on gpus. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–14, 2023b.

Gangmuk Lim, Jeongseob Ahn, Wencong Xiao, Youngjin Kwon, and Myeongjae Jeon. Zico: Efficient {GPU} memory sharing for concurrent {DNN} training. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, pp. 161–175, 2021.

Bill Yuchen Lin, Wangchunshu Zhou, Ming Shen, Pei Zhou, Chandra Bhagavatula, Yejin Choi, and Xiang Ren. Commongen: A constrained text generation challenge for generative commonsense reasoning. *arXiv preprint arXiv:1911.03705*, 2019.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

Hao Liu, Matei Zaharia, and Pieter Abbeel. Ring attention with blockwise transformers for near-infinite context. *arXiv preprint arXiv:2310.01889*, 2023.

Kshiteej Mahajan, Arjun Balasubramanian, Arjun Singhvi, Shivaram Venkataraman, Aditya Akella, Amar Phanishayee, and Shuchi Chawla. Themis: Fair and efficient {GPU} cluster scheduling. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pp. 289–304, 2020.

Aleksander Maricq, Dmitry Duplyakin, Ivo Jimenez, Carlos Maltzahn, Ryan Stutsman, and Robert Ricci. Taming performance variability. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pp. 409–425, 2018.

MIG. https://www.nvidia.com/en-us/technologies/multi-instance-gpu/, 2023.

Jayashree Mohan, Amar Phanishayee, Janardhan Kulkarni, and Vijay Chidambaram. Looking beyond {GPUs} for {DNN} scheduling on {Multi-Tenant} clusters. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pp. 579–596, 2022.

MPS. https://docs.nvidia.com/deploy/mps/index.html, 2023.

Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R Devanur, Gregory R Ganger, Phillip B Gibbons, and Matei Zaharia. Pipedream: Generalized pipeline parallelism for dnn training. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, pp. 1–15, 2019.

Deepak Narayanan, Keshav Santhanam, Fiodar Kazhamiaka, Amar Phanishayee, and Matei Zaharia. {Heterogeneity-Aware} cluster scheduling policies for deep learning workloads. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pp. 481–498, 2020.

Deepak Narayanan, Fiodar Kazhamiaka, Firas Abuzaid, Peter Kraft, Akshay Agrawal, Srikanth Kandula, Stephen Boyd, and Matei Zaharia. Solving large-scale granular resource allocation problems efficiently with pop. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, pp. 521–537, 2021a.

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

Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, et al. Efficient large-scale language model training on gpu clusters using megatron-lm. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–15, 2021c.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Yanghua Peng, Yixin Bao, Yangrui Chen, Chuan Wu, and Chuanxiong Guo. Optimus: an efficient dynamic resource scheduler for deep learning clusters. In *Proceedings of the Thirteenth EuroSys Conference*, pp. 1–14, 2018.

Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 652–660, 2017.

Penghui Qi, Xinyi Wan, Guangxing Huang, and Min Lin. Zero bubble pipeline parallelism. *arXiv preprint arXiv:2401.10241*, 2023.

Aurick Qiao, Sang Keun Choe, Suhas Jayaram Subramanya, Willie Neiswanger, Qirong Ho, Hao Zhang, Gregory R Ganger, and Eric P Xing. Pollux: Co-adaptive cluster scheduling for goodput-optimized deep learning. In *15th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 21)*, 2021.

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

Samyam Rajbhandari, Conglong Li, Zhewei Yao, Minjia Zhang, Reza Yazdani Aminabadi, Ammar Ahmad Awan, Jeff Rasley, and Yuxiong He. Deepspeed-moe: Advancing mixture-of-experts inference and training to power next-generation ai scale. In *International conference on machine learning*, pp. 18332–18346. PMLR, 2022.

Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. 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*, pp. 3505–3506, 2020.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115:211–252, 2015.

Maarten Sap, Hannah Rashkin, Derek Chen, Ronan LeBras, and Yejin Choi. Socialiqa: Commonsense reasoning about social interactions. *arXiv preprint arXiv:1904.09728*, 2019.

Jörg Schad, Jens Dittrich, and Jorge-Arnulfo Quiané-Ruiz. Runtime measurements in the cloud: observing, analyzing, and reducing variance. *Proceedings of the VLDB Endowment*, 3(1-2): 460–471, 2010.

Noam Shazeer, Youlong Cheng, Niki Parmar, Dustin Tran, Ashish Vaswani, Penporn Koanantakool, Peter Hawkins, HyoukJoong Lee, Mingsheng Hong, Cliff Young, et al. Mesh-tensorflow: Deep learning for supercomputers. *Advances in neural information processing systems*, 31, 2018.

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

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Shaden Smith, Mostofa Patwary, Brandon Norick, Patrick LeGresley, Samyam Rajbhandari, Jared Casper, Zhun Liu, Shrimai Prabhumoye, George Zerveas, Vijay Korthikanti, et al. Using deepspeed and megatron to train megatron-turing nlg 530b, a large-scale generative language model. *arXiv preprint arXiv:2201.11990*, 2022.

Suhas Jayaram Subramanya, Don Kurian Dennis, Virginia Smith, and Gregory R Ganger. Copter: Efficient large-scale resource-allocation via continual optimization. In *Symposium on Operating Systems Principles (SOSP)*, 2025.

Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, et al. Gemma: Open models based on gemini research and technology. *arXiv preprint arXiv:2403.08295*, 2024a.

Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, et al. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*, 2024b.

Kimi Team, Yifan Bai, Yiping Bao, Guanduo Chen, Jiahao Chen, Ningxin Chen, Ruijue Chen, Yanru Chen, Yuankun Chen, Yutian Chen, et al. Kimi k2: Open agentic intelligence. *arXiv preprint arXiv:2507.20534*, 2025a.

Kimi Team, Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun Xiao, Chenzhuang Du, Chonghua Liao, et al. Kimi k1. 5: Scaling reinforcement learning with llms. *arXiv preprint arXiv:2501.12599*, 2025b.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023a.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023b.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Pauli Virtanen, Ralf Gommers, Travis E Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, et al. Scipy 1.0: fundamental algorithms for scientific computing in python. *Nature methods*, 17(3):261–272, 2020.

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

Yizhong Wang, Swaroop Mishra, Pegah Alipoormolabashi, Yeganeh Kordi, Amirreza Mirzaei, Anjana Arunkumar, Arjun Ashok, Arut Selvan Dhanasekaran, Atharva Naik, David Stap, et al. Super-naturalinstructions: Generalization via declarative instructions on 1600+ nlp tasks. *arXiv preprint arXiv:2204.07705*, 2022.

Qizhen Weng, Wencong Xiao, Yinghao Yu, Wei Wang, Cheng Wang, Jian He, Yong Li, Liping Zhang, Wei Lin, and Yu Ding. {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)*, pp. 945–960, 2022.

Wencong Xiao, Romil Bhardwaj, Ramachandran Ramjee, Muthian Sivathanu, Nipun Kwatra, Zhenhua Han, Pratyush Patel, Xuan Peng, Hanyu Zhao, Quanlu Zhang, et al. Gandiva: Introspective cluster scheduling for deep learning. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pp. 595–610, 2018.

Wencong Xiao, Shiru Ren, Yong Li, Yang Zhang, Pengyang Hou, Zhi Li, Yihui Feng, Wei Lin, and Yangqing Jia. {AntMan}: Dynamic scaling on {GPU} clusters for deep learning. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pp. 533–548, 2020.

Zhiying Xu, Minlan Yu, and Francis Y Yan. Decouple and decompose: Scaling resource allocation with {DeDe}. In *19th USENIX Symposium on Operating Systems Design and Implementation (OSDI 25)*, pp. 393–409, 2025.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, et al. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*, 2024.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W Cohen, Ruslan Salakhutdinov, and Christopher D Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. *arXiv preprint arXiv:1809.09600*, 2018.

Fisher Yu, Ari Seff, Yinda Zhang, Shuran Song, Thomas Funkhouser, and Jianxiong Xiao. Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop. *arXiv preprint arXiv:1506.03365*, 2015.

Peifeng Yu and Mosharaf Chowdhury. Fine-grained gpu sharing primitives for deep learning applications. *Proceedings of Machine Learning and Systems*, 2:98–111, 2020.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*, 2019.

Yihao Zhao, Yuanqiang Liu, Yanghua Peng, Yibo Zhu, Xuanzhe Liu, and Xin Jin. Multi-resource interleaving for deep learning training. In *Proceedings of the ACM SIGCOMM 2022 Conference*, pp. 428–440, 2022.

Yihao Zhao, Xin Liu, Shufan Liu, Xiang Li, Yibo Zhu, Gang Huang, Xuanzhe Liu, and Xin Jin. Muxflow: Efficient and safe gpu sharing in large-scale production deep learning clusters. *arXiv preprint arXiv:2303.13803*, 2023.

Pengfei Zheng, Rui Pan, Tarannum Khan, Shivaram Venkataraman, and Aditya Akella. Shockwave: Fair and efficient cluster scheduling for dynamic adaptation in machine learning. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pp. 703–723, 2023.