Deep Learning Workload Scheduling in GPU Datacenters: Taxonomy, Challenges and Vision

WEI GAO* and QINGHAO HU*, Nanyang Technological University, Singapore
ZHISHENG YE*, Peking University, China
PENG SUN, SenseTime, Singapore
XIAOLIN WANG and YINGWEI LUO, Peking University, China
TIANWEI ZHANG and YONGGANG WEN, Nanyang Technological University, Singapore

Deep learning (DL) shows its prosperity in a wide variety of fields. The development of a DL model is a time-consuming and resource-intensive procedure. Hence, dedicated GPU accelerators have been collectively constructed into a GPU datacenter. An efficient scheduler design for such GPU datacenter is crucially important to reduce the operational cost and improve resource utilization. However, traditional approaches designed for big data or high performance computing workloads can not support DL workloads to fully utilize the GPU resources. Recently, substantial schedulers are proposed to tailor for DL workloads in GPU datacenters. This paper surveys existing research efforts for both training and inference workloads. We primarily present how existing schedulers facilitate the respective workloads from the *scheduling objectives* and *resource consumption features*. Finally, we prospect several promising future research directions. More detailed summary with the surveyed paper and code links can be found at our project website: https://github.com/S-Lab-System-Group/Awesome-DL-Scheduling-Papers.

CCS Concepts: • **General and reference** → **Surveys and overviews**; • **Computing methodologies** → **Machine learning**; • **Computer systems organization** → **Cloud computing**.

Additional Key Words and Phrases: Deep Learning Systems, Datacenter Scheduling, Resource Allocation

1 INTRODUCTION

Recent decades have witnessed a dramatic increase in deep learning (DL) research, development, and application in many fields, including Go [130], medical analysis [125], robotics [48], etc. A standard DL development pipeline consists of *model training* and *model inference*. Each stage requires high-grade hardware resources (GPU and other compute systems) to produce and serve production-level DL models [62, 71, 106, 149]. Therefore it becomes prevalent for IT industries [62, 149] and research institutes [18, 19, 71] to set up *GPU datacenters* to meet their ever-growing DL development demands. A GPU datacenter possesses large amounts of heterogeneous compute resources to host large amounts of DL workloads. An effective scheduler system is urgently required to orchestrate these resources and workloads to guarantee the efficiency of DL workload execution, hardware utilization, and other scheduling objectives.

The scheduler is responsible for determining the resource utilization of the entire datacenter and the performance of each job, which further affects the operation cost and user experience [42]. Specifically, (1) for model training, the scheduler allocates resources requested by the users to support the long-running offline training workloads. The scheduler needs to achieve high performance for each individual workload, high resource utilization for the entire datacenter, and high fairness among different users. Due to the unique and complicated features of DL training jobs, conventional scheduling algorithms for high performance computing (HPC) and big data workloads

*Equal contribution. Alphabetical order of surname.

Authors' addresses: Wei Gao, gaow0007@ntu.edu.sg; Qinghao Hu, Nanyang Technological University, Singapore, qinghao.hu@ntu.edu.sg; Zhisheng Ye, Peking University, China, yezhisheng@pku.edu.cn; Peng Sun, SenseTime, Singapore, sunpeng1@sensetime.com; Xiaolin Wang, wxl@pku.edu.cn; Yingwei Luo, Peking University, China, lyw@pku.edu.cn; Tianwei Zhang, tianwei.zhang@ntu.edu.sg; Yonggang Wen, Nanyang Technological University, Singapore, ygwen@e.ntu.edu.sg.

could cause unbalanced resource utilization and exorbitant infrastructure expense [157], and new solutions tailored for GPU datacenters are required. (2) For model inference, DL applications often serve as online services to answer users' requests. They often have a higher expectation on the response latency and inference accuracy [25, 172]. Applications that fail to be completed within the specified time (Service Level Agreement) or have lower accuracy than expected may have little or no commercial values. Therefore, it is critical for the scheduler to balance the inference latency, accuracy and cost.

Over the years a variety of DL schedulers have been proposed for GPU datacenters [25, 46, 106, 117, 121, 152, 172]. However, most of these systems are designed in an ad-hoc way for some specific objectives. There is still a lack of comprehensive exploration towards efficient scheduling of DL workloads. We are interested in the following questions: (1) what are the main challenges for designing a satisfactory scheduler to manage DL workloads and resources? (2) Do existing solutions share common strategies to achieve their scheduling objectives? (3) How do we need to refine the schedulers to adapt to the rapid development of DL technology? Those questions are important for system researchers and practitioners to understand the fundamental principles of DL workload scheduling and management, and design innovative schedulers for more complex scenarios and objectives. Unfortunately, there are currently no such works to summarize and answer these questions from a systematic point of view.

To the best of our knowledge, this paper presents the *first* survey for scheduling both DL training and inference workloads in research and production GPU datacenters. We make the following contributions. First, we perform an in-depth analysis about the characteristics of DL workloads and identify the inherent challenges to manage various DL workloads in GPU datacenters. Second, we comprehensively review and summarize existing DL scheduling works. We categorize these solutions based on the scheduling objectives and resource consumption features. We also analyze their mechanisms to address the scheduling challenges. Such summary can disclose the common and important considerations for existing DL scheduler designs. Third, we conclude the limitations and implications from existing designs, which can shed new light on possible directions of scheduler designs in GPU datacenters. We expect this survey can help the community understand the development of DL schedulers and facilitate future designs.

Existing surveys. Past works also presented some surveys, which are relevant to but distinct from ours. (1) Some works summarized the optimization techniques for DL applications, such as distributed training acceleration [112, 139], efficient model inference [44, 90], etc. These surveys primarily focused on the acceleration of individual jobs, while we consider the global optimization of the entire datacenter with plenty of workloads for various objectives. (2) Some works surveyed the scheduler designs for conventional cloud big data [128, 171] and HPC [109, 120] workloads. As discussed in Sec. 2.1, DL workloads have significantly distinct characteristics from these traditional jobs, and their scheduling mechanisms are not quite adaptable for DL training or inference. (3) Very few surveys conducted investigations on DL workload scheduling. Mayer and Jacobsen [98] summarized early designs of DL training job schedulers before 2019. This summary is outdated due to the emerging scheduling algorithms in recent years. Yu *et al.* [164] proposed a taxonomy for DL inference system optimization based on the computing paradigm. However, it mainly investigated the single node scenario instead of the datacenter scale. A recent work [163] considered the inference scheduling by colocating multiple workloads on the same GPU from both the cluster level and workload level. Different from those works, we provide a very comprehensive and up-to-date survey for scheduling techniques of both DL training and inference in the entire GPU datacenters.

Paper organization. The paper is organized as follows: Sec. 2 describes the unique characteristics of DL workloads and challenges for scheduling in GPU datacenters. It also illustrates the scope of this survey. The main body of this survey is presented in Fig 1. Concretely, Sec. 3 and Sec.

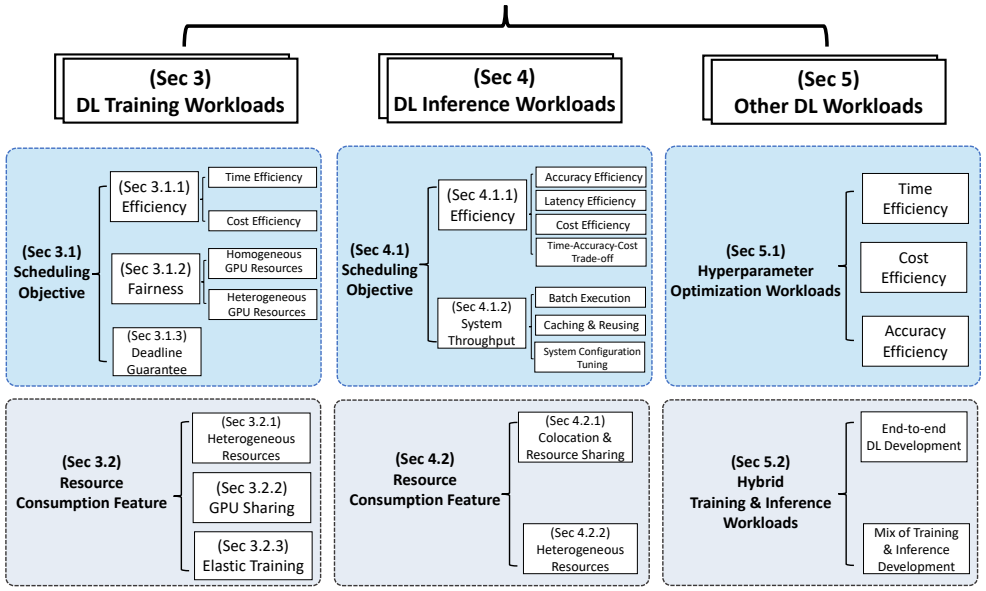

Fig. 1. **The overall structure of this survey.**

4 present detailed categorizations of training and inference workloads based on the scheduling objectives and resource consumption features, respectively. Sec. 5 discusses the other workloads, e.g., hyperparameter optimization, mixed training and inference workloads. Implications from these works are also given at the end of each section. Sec. 6 concludes this survey paper and identifies the future directions of scheduler designs.

2 BACKGROUND

2.1 DL Workloads and Their Characteristics

A DL development pipeline typically consists of three stages: data processing, model training and model inference. In this survey, we narrow down our focus to training and inference workloads which account for the most computation and consume the majority of resources in the datacenter.

2.1.1 DL Training. A DL training workload builds models by extracting features from existing data. A DL framework (e.g., PyTorch [114], TensorFlow [6]) is commonly adopted to fully utilize heterogeneous compute resources to accelerate the training process. To further reduce the training time, the workload is deployed across multiple GPUs with a data-parallel training scheme, which is implemented via distributed training libraries (e.g., Horovod [124], `DistributedDataParallel` in Pytorch, `MultiWorkerMirroredStrategy` in Tensorflow).

DL training workloads exhibit some unique features compared to traditional big data or HPC jobs, which need to be particularly considered for GPU datacenter scheduling. A series of studies have characterized training workloads from the production GPU datacenters, including Microsoft [71], SenseTime [62] and Alibaba [144, 149]. The characteristics are summarized as below.

T1: Inherent heterogeneity [71, 152]. GPU resources play a dominant role in DL training. However, CPUs and memory might interfere with the input processing and then delay the training execution. A GPU datacenter generally offers an ample pool of CPU and memory resources compared to GPUs. Arbitrary selection of heterogeneous resource combinations by users may lead to imperfect training progress. Figure 2 (f) shows the training performance speedups of common DL models with various generations of GPUs. Different models have diverse affinities to GPU types.

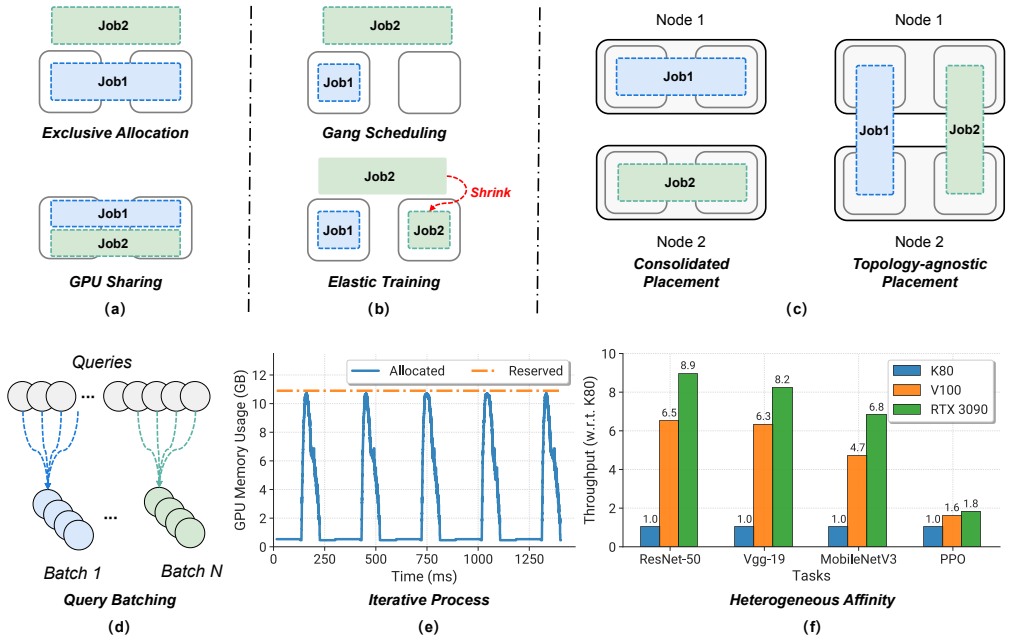

Fig. 2. **Characteristics of training and inference workloads.** (**a**) Exclusive Allocation versus GPU Sharing. (**b**) Gang Scheduling versus Elastic Training. (**c**) Consolidated Placement versus Topology-agnostic Placement. (**d**) Query Batching mechanism in inference. (**e**) Iterative Process: allocated and reserved GPU memory trace profiled through `torch.profiler` (ResNet-50 ImageNet classification task). (**f**) Heterogeneous Affinity: the magnitude of speedup across GPU generations varies significantly across different tasks.

T2: Placement sensitivity [97, 152]. Distributed DL jobs are sensitive to the locality of allocated GPU resources. Specifically, the runtime speed of some distributed DL jobs are bounded by device-to-device communication. Figure 2 (c) shows two types of placement, where a consolidated placement can efficiently reduce the communication overhead compared with topology-agnostic placement. The communication sensitivity of training jobs depends on the inherent property of the model structure. Advanced interconnect link (e.g., NVlink) can offer an order of magnitude higher bandwidth than PCIe. Therefore, distributed training jobs tend to request advanced interconnect to further obtain communication time reduction. Besides, jobs colocated in one server may suffer from PCIe bandwidth contention.

T3: Iterative process [42, 115]. DL training repeats a similar iterative pattern for up to thousands of times, as shown in Figure 2 (e). Each iteration consists of forward propagation, backward propagation and parameter update. It motivates that profiling a small number of iterations suffices to predict the pattern of future GPU memory usage and job completion time.

T4: Feedback-driven exploration [152, 183]. Training a DL model is a typical trial-and-error process. Users may explore a number of trial configurations and terminate unpromising trials by the early feedback. Such early feedback can further motivate to launch new trial configurations. Hence, a GPU datacenter hosts abundant repetitive training trials and short duration trials.

T5: Exclusive allocation [62] **versus GPU sharing** [149]. Figure 2 (a) depicts the difference between exclusive allocation and GPU. Exclusive allocation refers to that a DL job exclusively has the resource usage ownership. On the contray, GPU sharing allows multiple jobs to co-locate in the same GPU device and take advantage of resources in a time-/space- sharing manner. Unlike CPUs, GPUs basically do not have the intrinsic hardware-level support for fine-grained sharing across

users and thus they are allocated to DL training jobs exclusively. Due to the increasing hardware compute capability, plenty of DL training jobs can not fully utilize recent generations of GPU chips. To address this issue, datacenters enable GPU sharing through various technologies, e.g., NVIDIA Multi-Instance GPU (MIG) [3], Multi-Process Service (MPS) [4], GPU virtualization [61].

T6: Gang scheduling [62] **versus elastic training** [117]. Figure 2 (b) illustrates two scheduling mechanisms for data-parallel DL jobs. In particular, gang scheduling is that DL training requires all the GPUs to be allocated simultaneously in an all-or-nothing manner [35]. The requirement of gang scheduling results from the native support of DL frameworks and runtime speed performance guarantee. In contrast, elastic training removes the strict GPU request constraint, and allows a dynamic number of GPUs to run training jobs. Many scheduling systems support elastic training in order to improve GPU utilization and accelerate the training process. They take advantage of the elasticity of DL training workloads: a DL training job can adapt to a wide range of GPU counts and the training processes can be suspended and resumed via checkpoints [62].

2.1.2 DL inference. Model inference is the process of making predictions to users' inputs. It is commonly applied as online services (e.g., personalized recommendations, face recognition, language translation). DL frameworks also make efforts to support inference workloads, like TensorFlow Serving [111], MXNet Model Server [1], etc. The inference jobs must be performed in a real-time manner, facing dynamic queries with strict latency requirements [172]. They may process each inference request individually, or batch multiple requests concurrently to balance the resource usage and latency. Since many inference systems are deployed in the public cloud alternative to on-premise clusters, there exist many works emphasizing how to exploit cloud resources at scale to handle inference requests. According to the report from AWS [63], the cost of DL inference has already taken up the majority (more than 90%) of the total infrastructure cost for machine learning as a service. A DL inference workload also gives unique characteristics that can affect the scheduling system designs. They are summarized as follows.

I1: Deterministic online execution [28, 51]. Different from offline training which could be resource-intensive and last for days or weeks, the inference for each query is often completed with sub-second response time and consumes much less resources. Moreover, many inference jobs reveal deterministic execution flows and duration under fixed-size query input. This gives predictable resource usage and execution speed, offering the opportunities of fine-grained optimization.

I2: High demands on latency and accuracy [25, 172]. First, the inference service is expected to respond to the incoming queries promptly. Delays of inference responses can cause bad user experience. For example, an online recommend service is required to provide recommendations at interactive latencies (<100ms) to prevent user losses [25]. Other kinds of inference services also have strong latency requirements (e.g., <200ms [172]). Second, the prediction accuracy is also critical for building a reliable inference service. Inference workloads in some critical domains, e.g., healthcare and finance, may have stronger accuracy requirements [55]. The tight latency and accuracy demands pose great difficulty in managing inference jobs on GPUs, and there exist a trade-off between high accuracy and low latency. The datacenter managers need to carefully balance the latency overhead and prediction performance of the inference workloads.

2.2 Scheduler in GPU Datacenters and Design Challenges

Scheduling has continuously drawn public attention for several decades [34, 36, 37]. Similar to scheduling at the level of the operating system, networking system or applications, parallel job scheduling at the datacenter level makes decisions about the allocation of computing resources to competing jobs for specific scheduling objectives [36], which forms an NP-hard problem. In particular, it matches available resources with pending jobs and decides the optimal moment and

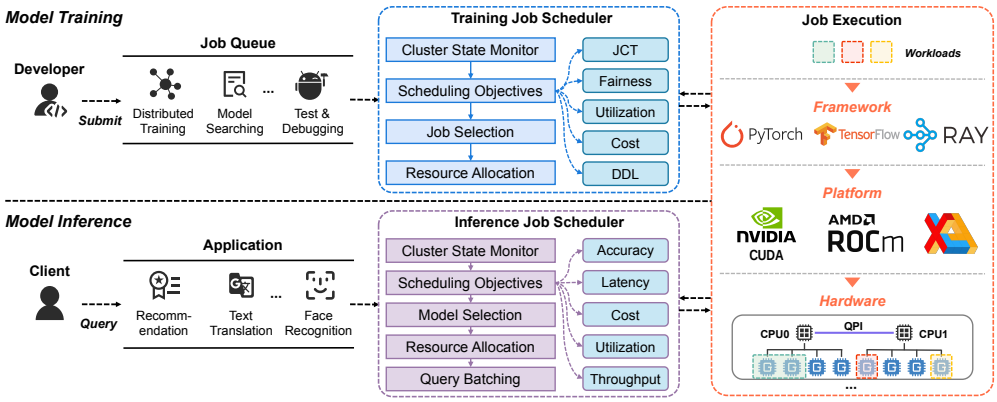

Fig. 3. **Scheduling workflow for model training and inference workloads.**

amount of resources to be allocated to each job. Modern datacenters have introduced a number of schedulers to manage conventional workloads. For instance, HPC schedulers (e.g., Slurm [162], OpenPBS [5]) are used to support HPC applications and scientific computing; cloud schedulers (e.g., Mesos [59], Kubernetes [14], Yarn [138]) help allocate heterogeneous compute resources for big data applications at scale.

As a special case, DL workload scheduling in GPU datacenters shares many similar features as conventional parallel job scheduling. Figure 3 shows the general workflow of DL schedulers in a GPU datacenter. The scheduler works on top of the DN frameworks, and assigns appropriate resources to satisfy a variety of DL workloads. It receives different types of workloads from the users. By monitoring the usages of existing compute resources in the datacenter, it delivers an efficient scheduling solution for these workloads to optimize the predetermined scheduling objective, e.g, JCT, fairness. Then it allocates the jobs to a set of hardware resources for execution. The schedulers for model training and model inference share similar logic flows but have totally different scheduling objectives, workload types, and target users. So our survey will investigate them separately (Sec. 3 and 4), and consider the mix of them in Sec. 5.

2.2.1 Scheduling Techniques. Some techniques and mechanisms of conventional parallel job scheduling may also apply to DL workloads scheduling in GPU datacenters. For example, to manage computing resources more efficiently and provide guaranteed service for users, it is common to divide computing resources into separate partitions and set up different queues for different users or jobs with different characteristics [38, 46, 157]. Queues may also have different priorities and be equipped with different queuing policies, e.g., First-Come-First-Served and Shortest-Remaining-Time-First. Schedulers also pursue a better comprehension of affinities between workloads and resources to make wiser decisions. Therefore, mechanisms like performance modeling of workloads (e.g., online profiling [42] and performance prediction [46]) and trace analysis for characterizing the cluster-level workload distribution [62, 149] are widely adopted. Other traditional scheduling techniques (e.g., backfilling [46, 103]) and mechanisms (e.g., time-slicing [152], checkpointing [157], and migration [16, 152]) are also adopted for more flexible job arrangements and better resource utilization in DL workloads scheduling.

However, due to the distinct characteristics of DL jobs (Sec. 2.1), simply adopting these techniques can cause a series of issues, e.g., serving job blocking, resource under-utilization, high operation cost. Below we summarize the challenges of scheduler designs caused by DL workload features.

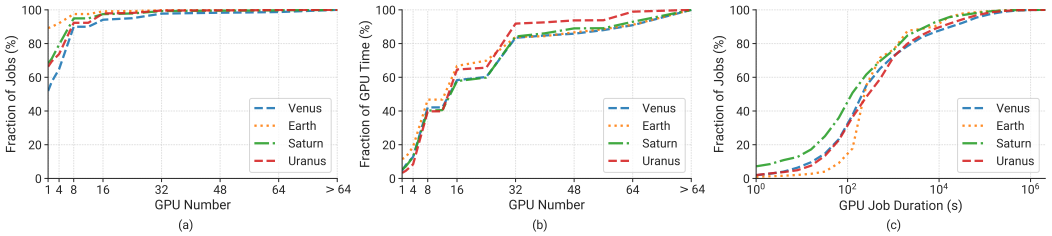

Fig. 4. **Characterization of four clusters in SenseTime GPU datacenter Helios**. (a) CDF of job size with job distributions. (b) CDF of job size with GPU time distribution. (c) CDF of job duration.

2.2.2 Challenges for Scheduling Training Jobs. As discussed in Sec. 2.1.1, DL training workloads have some unique requirements compared to HPC or cloud jobs, which raises some challenges for scheduling them. We discuss these challenges with a workload trace analysis from four private clusters (*Venus*, *Earth*, *Saturn* and *Uranus*) in SenseTime GPU datacenter `Helios` [62]. These clusters contain over 6000 GPUs and 1.5 million GPU jobs in total, spanning 6 months in 2020.

C1: Intensive resource consumption. The adoption of distributed training aims to reduce the training time yet prompts users to overclaim GPU resources for their jobs. Figures 4 (a) and (b) depict the distributions of requested GPUs pertaining to job and GPU resource occupation respectively. We observe that large-size jobs ($\leq$8 GPUs) account for 10% of the entire trace, and they consume over half of computing resources. Such intensive resource requests can aggravate the job pending issue due to the shortage of GPU resources. If the scheduler prioritizes those large-scale jobs, the situation becomes worse as subsequent jobs have to compete for much less resources. Existing solutions often favor small jobs or treat large and small jobs equally. How to balance the trade-off between intensive and light-weight resource consumption remains a challenging problem.

C2: Unbalanced runtime distribution. Recent trace analysis works [62, 71, 144, 149] presented the long-tail runtime distribution of DL training workloads in production GPU datacenters. Figure 4 (c) compares the GPU job duration distribution of each cluster. We observe it is common that job runtime varies from seconds to weeks even months among different production-level GPU clusters. The majority of workloads only finish within a short period of time, while the minority part consume many orders magnitudes of GPU time. Prioritizing short jobs is an effective way to reduce average job completion time but incurs low GPU utilization. More research efforts should be devoted to balance between short jobs and time-consuming jobs.

C3: Heterogeneous resource affinity. The runtime speed of a DL training job is affected by a variety of hardware factors, among which GPU heterogeneity and network link heterogeneity are the most important ones. For the impact of GPUs, DL training can benefit from newer generations of GPUs. However, the marginal benefit brought by new GPU versions varies significantly (Figure 2 (f)). Also, the speedup ratio is unpredictable, which complicates the heterogeneous GPU resource allocation. For the impact of network links, the recently released high-end GPU interconnect including NVlink and NVswitch can significantly reduce the communication overhead across GPUs in the same sever. Along with PCI Express, InfiniBand, Ethernet and QPI, distributed training has several alternatives for cross-GPU communications. As these links differ considerably in bandwidths, and different jobs have different data sizes for exchange, it is non-trivial to allocate these network resources to the jobs to maximize the benefits and minimize the bandwidth contention.

C4: Preemption overhead. DL frameworks usually provide functions to pause/resume the training jobs at any time for better fault-tolerance. The overhead of such processes primarily depends upon the job scale, which ranges from seconds to minutes. In this paper, the preemption overhead is considered as the addition of the costs of pausing and resuming the job. For time-consuming jobs, the preemption overhead is relatively small with the benefit of higher scheduling

flexibility. But for short jobs, the preemption overhead is non-negligible, and frequent preemption will delay their progress. Designing an appropriate preemptive mechanism requires meticulous considerations of both short and time-consuming jobs as well as their preemption overheads.

2.2.3 Challenges for Scheduling Inference Jobs. The online execution fashion and high latency requirement of inference workloads also give the following challenges for designing a scheduler.

C5: Low GPU utilization for each request. Compared to training jobs, the inference service mainly involves small convolutional operations (e.g., 1x1, 3x3), and consumes small amounts of GPU resources. Besides, the peak performance of new GPUs are increasing rapidly [32]. This often leads to low GPU utilization for inference workloads [83, 173]. A common practice to improve the GPU utilization is to batch multiple inference requests and execute them at the same time [25].

C6: Latency-accuracy-cost tradeoff. The inference jobs are relatively malleable in terms of latency, accuracy and cost. To improve the resource utilization and cluster-wide job throughput, we can colocate multiple inference jobs or increase the batch size. However, this can increase the inference latency. To increase the accuracy, effective ways include model ensemble or augmentation evaluation, which can also incur latency delay [52]. The adoption of high-class hardware resources can accelerate the inference execution, but charges more for online services. Different users or inference jobs may have different demands towards latency, accuracy and cost. The scheduler needs to figure out a sweet spot for each job over an assortment of algorithms and hardware units.

C7: Bursty and fluctuating requests. As an online service, it is common for the inference application to receive bursty and fluctuating requests, which are unpredictable. This must be considered when determining the resources for the workload. How to guarantee the latency with the minimal operational cost even in extremely overloading scenarios raises a new challenge. In practice, resources are often over-provisioned for inference workloads to guarantee their latency during the rush hours. Then an efficient scheduler needs to consider how to exploit the unused resources of these workloads when there are less queries.

2.3 Relevant Studies Not Included in This Survey

This survey mainly focuses on the scheduling of DL training and inference workloads in GPU datacenters. Other relevant works beyond the scope of this paper will not be summarized in the following sections. Here we briefly discuss these directions. Readers who are interested in these works can refer to relevant surveys [44, 90, 109, 112, 120, 128, 139, 171].

First, we do not consider the *optimization solutions* for *individual* training or inference jobs. Training job optimization mainly contains distributed training acceleration [17, 74, 175] and job placement optimization [84, 94, 161]. Inference job optimization techniques include workload characterization [15], pipeline execution [75], etc. Their objectives are to achieve high performance for a single job instead of an entire datacenter. It is worth noting that scheduling hyperparameter optimization jobs will not be considered as single job optimization, because it involves a collection of training tasks (e.g., RubberBand [101], HyperSched [91]). They will be summarized in Sec. 5.

Second, we consider the scheduling at the job level, and do not cover the scheduling approaches at the hardware resource level (e.g., network I/O, power). For instance, HIRE [13] proposed a novel in-network computing scheduling algorithm for datacenter switches. A number of works [49, 99, 145] utilized the DVFS mechanism on CPU and GPU chips to achieve cluster energy conservation. These works are not included in this survey.

Third, we focus on the GPU datacenters where GPUs are the primary resources for the DL workloads. Those datacenters can also exploit other resources (e.g., CPU, FPGA, ASIC) as subsidiary. This can reflect the current status of mainstream DL infrastructures. Some scheduling systems mainly utilize the CPU [11, 53, 66, 156], FPGA [65, 72], or hybrid resources [73] where GPUs are not

dominant. Some papers consider the DL services on mobile devices [110] or edge computing [123, 177] other than datacenters. Those works are also out of the scope of this survey.

Fourth, we target the scheduling of general DL training and inference workloads. Some works studied other types of DL jobs, e.g., data processing, model re-training, model validation. Some papers considered the optimization of specific DL applications based on their unique behaviors, including RNN-based service [41, 60], recommendation systems [54, 72, 73, 93] and video analytics [126, 174]. These works are not summarized in this paper. Besides, our aim is to enhance the system and workloads in terms of performance, efficiency and user experience. Other objectives like privacy protection [81, 96] is not considered either.

3 SCHEDULING DL TRAINING WORKLOADS

DL training jobs consume a majority of compute resources in GPU datacenters. Therefore an effective and efficient scheduler for training workloads is of critical importance. Existing scheduling systems can be generally categorized from two dimensions: scheduling objective and resource consumption feature. Table 1 summarizes the past works for DL training scheduling. We detail them in the rest of this section.

3.1 Scheduling Objectives

Different schedulers are designed to achieve different objectives, including efficiency, fairness and deadline guarantee. We first review past works from this perspective.

3.1.1 Efficiency. Efficiency is a main objective to pursue when designing the workload schedulers. The GPU datacenter manager can consider different types of efficiency. We classify the efficiency-aware schedulers into three categories, as discussed below.

1) Timing efficiency. This scheduling goal is to reduce the average queuing and execution time of training workloads in a datacenter. Some advanced strategies with special training configurations (e.g., sharing training, elastic training, heterogeneous training) can help improve the timing efficiency [64, 79, 85, 117, 151–154], which will be elaborated in Sec. 3.2. Here we mainly discuss the techniques over common training configurations that support gang scheduling, resource exclusive usage and preemptive operations.

One of the most common and effective ways for guaranteeing timing efficiency is to adopt some heuristic functions to determine the job scheduling priority C1[1]. For instance, Tiresias [46] designs the *Least Attained Service* (LAS) algorithm to prioritize jobs based on their *service*, a metric defined as the multiplication of requested GPU resources and execution time. It devises the priority discretization to mitigate the frequent preemption issue C4, which is inspired by the classic Multi-Level Feedback Queue (MLFQ) algorithm [8, 22, 23]. These techniques enable Tiresias to beat the classical YARN-CS [138] significantly. E-LAS [132] improves over Tiresias by prioritizing jobs with the *real-time epoch progress rate*, which is computed as the proportion of the current training epoch over the total number of training epochs. With such improvement, E-LAS outperforms Tiresias in terms of average job timing efficiency. FfDL [70] is an open-sourced scheduler platform developed by IBM. It uses the operating lessons from the industry practice to guide the management of DL training workloads in the cloud.

An alternative strategy is to use machine learning techniques for job scheduling. *Sched*^{2} [95] is a scheduler based on reinforcement learning (RL). It utilizes a Q-network which takes the job state and GPU datacenter state as input, and outputs the optimal job to be scheduled. MLFS [141] also leverages RL to determine the job priority and resource allocation. The RL model takes as input the

[1] CX indicates the challenge (Sec. 2.2.2 and 2.2.3) to be addressed. TX and IX indicate the training and inference job characteristic (Sec. 2.1.1 and 2.1.2) to be considered in the scheduler design.

Table 1. **Summary of schedulers for DL training workloads in GPU Datacenters**.

Year	Scheduler	Objectives	Approaches	Advantages	Heter.	Elastic	AutoML	Exp. Scale	Source Code
2017	Dorm [133]	♥	Linear Programming	Fairness Guarantee; JCT Reduction	-	-	-	S	-
	Topology-Aware [7]	♣	Best-effort topology-aware placement	Lower interference	-	-	-	S	✔
2018	Gandiva [152]	✿♣	Time-slicing; Migration; Grow-shrink	Better GPU Utilization	-	✔	✔	L	-
	OASiS [10]	✿♣	Primal-dual framework	Better GPU utilization	-	✔	-	S	-
	Optimus [115]	♣	Performance Modelling	JCT Reduction	-	✔	-	S	✔
2019	FC^2 [134]	♦	Automatic Resource Configuration	Cost-effectiveness	✔*	✔	-	S	-
	$Sched^2$ [95]	♣	Q-Network based Scheduler	Reduce Cluster Fragmentation	-	-	-	L	-
	Cynthia [182]	♦	Performance Modelling	Monetary Cost Reduction	-	✔	-	S	-
	Dragon [92]	✿♣	GPU time-sharing; Autoscaling	Better GPU utilization	-	✔	-	S	-
	FfDL [70]	♣	Lesson-motivated Design	production DL platform	-	-	-	S	✔
	Harmony [9]	♣	RL Scheduler; Bin Packing	JCT Reduction; Better GPU Utilization	-	-	-	S	-
	JPAS [183]	♦	Accuracy Curve Modelling	JCT Reduction	-	-	✔	S	-
	Philly [71]	♣	Locality-Relaxity	Workload Analysis; JCT Reduction	-	-	-	-	✔
	Tiresias [46]	♣	Gittins index; Least-Attained Service (LAS)	Information-agnostic	-	-	-	M	✔
2020	$Gandiva_{fair}$ [16]	♣	Gang-Aware Lottery; Automatic Trading	Inter-user fairness guarantee	✔	-	-	XL	-
	Ada-SRSF [146]	♥♣	AdaDUAL; Least workload first	Less communication contention	✔*	-	-	S	-
	Antman [153]	♣	Framework-Cluster Co-Design	Better GPU utilization	-	✔	-	XL	✔
	CODA [122]	✿♣	Adaptive CPU allocator; Contention eliminator; etc	Non-dominant resource aware	✔*	-	-	-	-
	Co-scheML [76]	✿♣	Interference-Aware Scheduler; Random Forest	Better GPU Utilization; JCT Reduction	-	-	-	S	-
	Elan [154]	♣	Hybrid scaling; IO-free replication;etc	Better GPU utilization; Less IO	-	✔	-	L	-
	E-LAS [132]	✿♣	Real-time Epoch Progress Rate; LAS	Information-agnostic	-	-	-	-	-
	Gavel [106]	♣	Linear Programming; Round-based Scheduling	Heterogeneity-Aware	✔	-	-	M	✔
	GENIE [20]	♣♥	Light Profiler	QoS Guarantee	-	✔	-	S	-
	HiveD [181]	♠	Buddy Cell Allocation	Better Resource Utilization	-	-	-	L	✔
	MARBLE [56]	♣	Offline profiling based scaling	Better GPU utilization	-	✔	-	S	-
	MLCloudPrice [105]	✿♣	Linear programming; Spot-Instance Training	Cloud Cost Reduction	-	-	-	-	✔
	MLFS [141]	♣♦	RL Scheduler	Optimize Multiple Objectives	-	-	-	-	✔
	Non-Intrusive [147]	♣♠	SideCar; Early initialization	Framework non-intrusive	-	✔	-	S	-
	Parrot [88]	✿♣	Linear Programming	Better Bandwidth Utilization	-	-	-	-	-
	Salus [168]	♣	Fast job switching; Memory sharing	Better GPU utilization	-	-	-	S	✔
	SPIN [57]	✿♣	Rounding-based Randomized Approximation	Robust Time Misestimation	-	-	-	-	-
	Themis [97]	♣	Finish-Time Fairness; Auction Bid	Better Fairness; GPU Utilization	-	-	-	L	-
	Vaibhav et al. [122]	♥	Dynamic programming optimization	Better GPU utilization	-	✔	-	M	-
	Yeung [159]	✿♣	GPU Utilization Prediction	Better GPU utilization	-	-	-	-	-
2021	DL^2 [116]	✿	RL Scheduler	JCT Reduction	-	✔	-	S	✔
	AFS [64]	♣	Apathetic Future Share; CoDDL	Better GPU utilization	-	✔	-	L	-
	ANDREAS [39]	✿♣	Randomized Greedy Algorithm	Energy Cost Reduction	-	-	-	S	-
	Astraea [157]	♦	Long-Term GPU-time Fairness	Fairness Guarantee	-	-	-	-	-
	Chronus [42]	♥	Linear Programming; Local Search Allocation	SLO Guarantee	-	-	-	S	✔
	DynamoML [21]	♠	Combine KubeShare and Dragon	Better GPU utilization	-	✔	-	S	-
	Helios [62]	✿♣	Data Driven Prediction; QSSF; CES	Workload Analysis; Energy Conservation	-	-	-	-	✔
	Horus [160]	♣	XGBoost-based interference prediction	No need to online profiling	-	-	-	S	-
	Jigsaw [79]	✿♣	Structured Partial Training	Algorithm-System Co-Design	-	-	-	M	-
	Liquid [47]	♣	Best-fit; Grouping genetic	Accelerate job execution	-	-	-	M	✔
	ONES [12]	♣	Online evolutionary search	Better GPU utilization	-	✔	-	S	✔
	Pollux [117]	✿♣	Goodput; Dynamic batch size and learning rate	Better GPU Utilization	-	✔	✔	L	✔
	POP [104]	✿♣♥	Partitioned Optimization	Reduce Scheduling Overhead	✔	-	-	L	✔
	SMD [167]	♦	Multi-dimensional-knapsack Decomposition	JCT Reduction	-	-	-	-	-
2022	Aonline [178, 184]	♣	Integer Linear Programming	JCT Reduction	-	✔	-	-	-
	EDL [151]	✿♣	Stop-free scaling; Graceful exit;etc	Better GPU utilization	-	✔	-	L	-
	GADGET [166]	✿♣	Greedy; G-VNE	JCT Reduction	-	✔	-	-	✔
	Ali-MLaaS [149]	✿♣	GPU Sharing; Predictable Duration	Fine-grained Workload Analysis	-	-	-	-	✔
	Singularity [129]	✿♣	Device Proxy; Replica splicing; etc	User code non-intrusive; Efficient	-	✔	-	M	-
	Synergy [102]	✿♣	Optimistic Profiling; Greedy Scheduling	Non-dominant resource aware	-	-	-	M	-

Objectives: ✿ Utilization ♣ JCT ♦ Cost ♥ Fairness ♠ DDL; **Heterogeneous:** ✔ heterogeneous GPUs of different generations. * heterogeneous resources (e.g., CPU, networking); **Experiment GPU Scales:** the scale of physical testbed. S $(0, 30]$ M $(30, 60]$ L $(60, 120]$ XL $(120, \infty]$ -: no evaluation on a physical cluster or not clearly specified.

job time information, resource demand, and accuracy requirements. It can effectively improve the average latency of a mix of data-parallel and model-parallel training jobs. Helios [62] characterizes the production training jobs from a shared GPU datacenter in SenseTime, and then adopts a variety of machine learning algorithms to predict the job priority from the history job information. The prediction result suffices to minimize the cluster-wide job latency. JPAS [183] is a scheduler based on the accuracy curve fitting technique to expedite the feedback-driven exploration of general training workloads **T4**. The feedback-driven exploration readily expects the scheduler to allocate more resources for more accurate models. JPAS leverages the accuracy curve fitting to predict the potential maximal accuracy improvement of each job, and then prioritize the jobs in a time interval. With this technique, JPAS can facilitate the early-stage progress of the training workloads and satisfy the needs for the feedback-driven exploration.

The timing efficiency of DL training jobs is highly dependent on the job placement **C3**, where different placement policies can lead to different communication overheads. Users prefer the strict placement locality to maintain the DL training speed **T2**. Amaral *et al.* [7] found that packing

jobs on the same CPU socket could bring up to 1.3× speedup compared to spreading jobs across different sockets. Then they designed the Topology-Aware scheduler, which uses a profiler to measure the placement sensitivity of each job, and thus performs a best-effort approach to schedule locality-sensitive jobs in a packing manner. Similarly, Tiresias [46] and E-LAS [132] also adopt the profiling strategy to identify the optimal job placement solutions. SMD [167] is a scheduler for parameter-server (PS) training jobs, which allows multiple jobs to contend the communication bandwidth. It models the scheduling problem as a non-convex integer non-linear program with the bin-packing constraints, and then develops an ϵ-approximation algorithm called sum-of-ratio multi-dimensional-knapsack decomposition to solve it. The effectiveness of the SMD scheduler is validated both theoretically and empirically. Philly [71] investigates a production workload trace from Microsoft and conducts a thorough analysis about the impact of gang scheduling and locality constraints on the queuing delay and job runtime. Motivated by this, it proposes to relax locality constraints to improve the job timing efficiency.

Sometimes the scheduler can satisfy the GPU capacity request but fail to meet the placement locality. This will lead to the cluster fragmentation issue, which is often caused by the scattered GPU resource allocation. HiveD [181] emphasizes that sharing the GPU datacenter without the consideration of cluster fragmentation will cause significant job queuing delay. Therefore it develops a buddy cell allocation mechanism to ensure *sharing safety*. HiveD can be easily incorporated with Tiresias [46] to reduce the queuing delay and further improve the job latency. *Sched*^{2} [95] addresses the cluster fragmentation problem with an RL model, which is able to satisfy the locality constraint as much as possible. SPIN [57] observes that delay scheduling [170] can bring reward to the GPU datacenter in the long term for satisfying the placement locality in the near future. It requires the job runtime information to determine the delay scheduling policy. SPIN proposes a rounding-based randomized approximation method to achieve this goal, which has strong robustness even with inaccurate job runtime estimation.

2) Cost efficiency. This refers to the reduction of power consumption or financial cost for renting cloud services. This is another significant objective for training workload scheduling.

Existing GPU datacenters have considerable power waste as not all the GPUs are actively used all the time, while the datacenter managers prefer to keep all the devices on. To reduce the energy cost, ANDREAS [39] considers a scenario where the execution of each job can be postponed within a certain period. Then it judiciously schedules jobs at appropriate moments to keep all the GPUs busy in the datacenter. It formulates the power consumption as a Mixed Integer Non-Linear Programming problem, and proposes an effective greedy heuristic algorithm to achieve a significant cost reduction. Different from ANDREAS, the *Cluster Saving Service* (CES) in Helios [62] has no assumption about postponing the execution of DL training jobs. It leverages a prediction model to estimate the future resource utilization from the history logs. Then the scheduler can decide how many GPU nodes should be turned on/off. CES can save the electricity by up to 1.65 million kilowatt-hours per year in four production clusters from SenseTime. Additionally, recent energy optimization frameworks such as GPOEO [140] can significantly save the power consumption of training workloads. Although they are not tailored for GPU datacenters, they can be easily transplanted into the GPU datacenter with a customized scheduler to orchestrate between datacenters and jobs.

Cloud GPU resources are billed based on the amount and duration of usage. Training a model can be very time-consuming and resource-intensive C1. As such, the cost of a training workload could be considerably expensive. It is critical to reduce such financial cost to produce the model with the same quality. Particularly, PS training is a common method for distributed data-parallel model training in the cloud. Cynthia [182] is a scheduler to guarantee the cost-effectiveness of cloud resource provision for PS training. It introduces an analytical performance model to characterize the relationship between throughput and resource provision. Through this performance model,

this scheduler can identify an optimal resource type and PS configurations to maintain the training throughput while minimizing the monetary cost. Analogously, FC^2 [134] is a scheduler, which recommends cost-effective and high-performing cloud resource configurations for PS training jobs. It selects the instances with the largest network bandwidth within the budget for the parameter server in order to avoid the communication bottleneck. It also proposes a heuristic method named *Scala-Opt* to decide the work instances which can guarantee the job throughput while maximizing the cost savings. Jahani [67] treats the compute node with different numbers of GPUs as different virtual machines (VMs). The renting cost and job throughput vary with different VM types. Then it models the scheduling process as a Mixed Integer Linear Programming (MILP) problem, and reduces the renting cost in a global manner while maintaining the job latency. MLCloudPrice [105] makes a quantitative analysis on the price difference among different GPU specifications and dynamic prices of the public cloud. It moves the workloads between spot and on-demand instances, which opportunistically utilizes the low-pricing spot instance to push forward the training progress.

3.1.2 Fairness. Fairness indicates how fairly the compute resources are allocated among different entities, including user groups (i.e., tenants) and workloads. Fairness schedulers aim to guarantee that each entity can achieve better performance with the resource sharing mechanism than exclusively using the same portion of resources. For conventional workloads, the design of fairness schedulers follows some typical fairness principles, such as sharing incentive, strategy-proofness, envy-freeness and pareto efficiency [43]. It is more challenging to maintain fairness for DL training workloads for two reasons: (1) A GPU is an indivisible resource in common settings (gang scheduling) T6; (2) DL training exhibits resource heterogeneity preference T1 C3. Below we discuss the new works that can address these two challenges for fairness scheduling of training workloads.

1) Homogeneous GPU resources. A datacenter with only one generation of GPU devices can be considered as a homogeneous GPU environment. The scheduler in this system achieves fairness sharing of indivisible GPU resources from the timing dimension. For instance, Themis [97] maintains the job-level fairness by introducing a new metric called *finish-time fairness*. This metric inspires the scheduler to allocate more resources to the jobs whose attained service is less than the deserved amount. Moreover, in existing fairness schedulers (e.g., DRF [43]), the placement preferences of DL training workloads can result in severe fairness sharing loss. To address this problem, Themis builds a two-level scheduling architecture for biding resource allocation among jobs and uses the game theory to guarantee the performance. Astraea [157] concentrates on the fairness across workloads and tenants. It introduces the Long-Term GPU-time Fairness (LTGF) metric to measure the sharing benefit of each tenant and job, and proposes a two-level max-min scheduling discipline to enforce job-level and tenant-level LTGF in a shared GPU datacenter.

2) Heterogeneous compute resources. It is relatively easy to maintain fairness over one type of GPUs. However, the existence of multiple generations of GPUs and other compute resources (e.g., CPUs, network links) can also exacerbate the fairness of workloads or user groups T1 C3. A couple of works have introduced solutions to achieve fairness in the heterogeneous environment[2].

To achieve the fairness over GPUs and other compute resources, Allox [80] is a fairness scheduler, which assumes that both GPUs and CPUs are interchangeable resources, and takes into account the affinity of workloads towards different compute resources. It models the resource allocation as a min-cost bipartite matching problem with a theoretically optimal property. Then it proposes a greedy heuristic solution to solve this problem in an effective and scalable way. Dorm [133] is another fairness scheduler for the fair sharing of GPUs, CPUs and memory resources. It assumes that GPUs, CPUs and memory are complementary resources and the capacity of each one can

[2]Note here we focus on how to fairly allocate heterogeneous resources. The consumption optimization of specific heterogeneous resources will be discussed in Sec 3.2.1.

influence the training job throughput. It dynamically partitions different types of compute resources for each DL training job. It formulates the resource allocation as an MILP problem with the resource utilization fairness as the optimization objective. The scheduling decision in each round is made by calling the MILP solver to optimize the utilization fairness.

It is also challenging to achieve fairness over different generations of GPUs. Datacenter users prefer to request the most powerful GPU resources for their training jobs. However, many jobs can not saturate the peak performance of these high-end GPUs. Besides, different DL training jobs have different sensitivities of runtime speed to the compute capability of GPUs. $Gandiva_{fair}$ [16] is an early fairness scheduler dedicated for the heterogeneous GPU resource environment. It targets the inter-user fairness in the GPU heterogeneity. To maintain such fairness while maximizing the cluster-wide job efficiency, $Gandiva_{fair}$ allows users to transparently trade heterogeneous GPU-time by a couple of techniques including profiling and automatic trade pricing. Gavel [106] is another heterogeneity-aware fairness scheduler. It profiles the performance heterogeneity between different types of GPUs and DL model architectures. A round-based scheduling technique is adopted to improve the scheduling flexibility and ensure timely GPU-time re-allocation. This scheduler can satisfy different types of fairness defnitions, e.g., max-min fairness, makespan minimization, finish-time fairness minimization. However, it is prohibitive to scale up Gavel to a large datacenter due to the time-consuming mathematical solving process. To this end, POP [104] proposes to partition a large datacenter into several smaller ones. Then the original complex optimization formulation is decomposed into multiple smaller problems and can be solved in parallel. It provides a theoretical proof and several empirical evidences to demonstrate the effectiveness of this optimization technique.

3.1.3 Deadline Guarantee. Different from the efficiency goal which aims to complete the job as soon as possible, this objective is to ensure the job can be done before the specified deadline. It is relatively less studied due to the lack of comprehensive analysis about the deadline requirement in DL workloads. An early deadline-aware scheduler for DL training workloads is GENIE [20]. It develops a performance model to predict the job throughput on different resource placement policies. The performance model only requires a small number of training iterations to profile without any significant degradation of job execution [13]. With this performance model, GENIE can identify the best placement policy for each job to satisfy the corresponding deadline requirement. However, GENIE [20] does not investigate the deadline requirement from users and cannot support a mixed workload of deadline and best-effort jobs. In [42], a user survey is conducted to uncover users' latent needs about the deadline guarantee, and comprehensively discuss the deadline requirement from GPU datacenter users. Motivated by this survey, it introduces Chronus, a scheduler to improve the deadline guarantee for Service-Level-Objective (SLO) jobs and latency of best-effort jobs at the same time. It formulates the deadline-guarantee scheduling task as an MILP problem with the resource and time constraints. The MILP solver can make effective scheduling decisions for a collection of jobs. Moreover, in consideration of the placement sensitivity of different training jobs, it proposes round-up and local-search techniques to make placement decisions. These designs successfully enable Chronus to outperform existing deadline schedulers in reducing deadline miss rates and improving the latency of best effort jobs.

3.2 Resource Consumption Feature

In addition to the scheduling objective, another orthogonal view to categorize training workloads is their resource consumption features. We discuss prior works based on whether they adopt heterogeneous resources, GPU sharing and elastic training.

3.2.1 Heterogeneous Resources. Most schedulers focus on the allocation of GPU resources, as they dominate the DL training. However, the consumption of CPUs and memory can also affect the training performance C3. Synergy [102] observes that different DL training jobs exhibit different levels of sensitivity to the CPU and memory allocation. An optimal allocation can improve the overall cluster utilization and efficiency. Therefore, it introduces *optimistic profiling* to empirically profile the job throughput for various CPU allocations and analytically estimate all the combinations of CPUs and memory along with the respective storage bandwidth requirement. Based on the profiling results, it performs round-based scheduling and greedily packs runnable jobs along multiple resource dimensions with the objective of minimizing the fragmentation in each round T1. CODA [180] observes that CPU jobs colocating within the same compute node can interfere with the training jobs due to the CPU resource contention. It then designs three components to optimize system-wide performance: an *adaptive CPU allocator* identifies the optimal CPU cores for each DL training job; a *real-time contention eliminator* monitors and throttles the memory bandwidth of each CPU job to reduce its interference with the GPU training jobs; a *multi-array job scheduler* allows CPU jobs to preempt the CPU cores reserved by the GPU jobs accordingly, and vice versa. Experimental results demonstrate CODA can efficiently improve the GPU utilization without sacrificing the performance of CPU jobs.

Beyond the CPU and memory resources, network bandwidth is another bottleneck for efficient DL training. Ada-SRSF [146] is a two-stage framework for mitigating the communication contention among DLT jobs. In the job scheduling stage, it is combined with the classical SRSF algorithm to relax the contention of two jobs if it can reduce the job completion time. In the job placement stage, it strives to balance the resource utilization and communication overhead. Liquid [47] proposes a cluster network-efficient scheduling solution to achieve better placement for PS-based distributed workloads. Specifically, it adopts a random forest model to predict job resource requirements and then uses the best-fit algorithm and grouping genetic algorithm to optimize the execution performance of DL jobs. Parrot [88] is a framework to manage network bandwidth contention among training jobs using the PS architecture. The communication scheme in a PS workload exhibits a coflow chain dependency where the event of parameter-pull happens after the event of parameter-push. Parrot tries to assign the bandwidth of each physical link to coflows while satisfying the dependency constraints in order to minimize the JCT. It adopts a least per-coflow attained service policy to prioritize jobs. Then it uses a linear program (LP) solution to derive a weighted bandwidth scaling strategy to minimize the time cost in the communication stage.

3.2.2 GPU Sharing. With the increased compute capability and memory capacity of GPUs, the conventional placement approach which makes each DL job exclusively use the GPU can lead to severe resource underutilization. It is now more promising to perform GPU sharing to fully exploit GPU resources and improve the system throughput T5. In this context, *utilization* is more inclined to the usage of every single GPU instead of the occupied GPU quantity at the datacenter scale.

Some works profile and revoke unsuitable jobs to achieve efficient GPU sharing. Salus [168] focuses on fine-grained GPU sharing with two primitives: *fast job switching* enables efficient time sharing and rapid preemption for active DL jobs on a GPU; *memory sharing* addresses the memory management issues to ensure high utilization by packing more small DL jobs on the same device. Gandiva [152] designs a *packing* mechanism to pack multiple jobs on one GPU under the constraints of GPU memory and job performance. It utilizes a profiling mechanism to monitor and unpack jobs that could affect jobs' performance. Jigsaw [79] is designed upon a novel distributed training scheme named *Structured Partial Backpropagation* (SPB). SPB allows each worker not to perform the entire backward pass in the distributed training. This can save lots of compute resources, and enable efficient time- and space-multiplexing across jobs in a single GPU. Although SPB can reduce

the cluster-wide JCT, it might lead to accuracy loss to some extent. Recently, Antman [153] is introduced, which co-designs the infrastructure between the cluster scheduler and DL framework engine to efficiently manage GPU resources in a fine-grained manner. It supports the co-execution of multiple jobs on a GPU device, and thus largely improves the overall compute resource utilization. Ali-MLaaS [149] provides a comprehensive analysis of large-scale workload traces in Alibaba, and discloses the benefit of GPU sharing in production GPU datacenters.

Alternatively, some works use data-driven approaches to make the GPU sharing decision. Horus [159, 160] designs a prediction-based interference-aware mechanism that can be integrated with existing DL training scheduling frameworks. The *prediction engine* in Horus is in charge of estimating the GPU usage of each DL job by accessing its graph and dry running the model upon the job submission. Based on the prediction results, Horus allocates GPU resources to DL jobs via de-prioritizing co-location placement decisions that would result in JCT slowdown from the severe interference and communication delays. Co-scheML [76] also measures some metrics for each DL job and uses a random forest model to predict the interference. Then the scheduler makes the decision with the aim of fully utilizing the cluster resources. Analogously, Liquid [47] also supports fine-grained GPU sharing for further resource utilization improvement using a random forest model. Harmony [9] applies an RL model to make placement decisions for minimizing the interference and maximizing the throughput for bin packing DL workloads in a GPU datacenter. It contains a reward function for the prediction module and RL placement decision-making module. This reward function aims to maximize the normalized training speed across all the concurrent jobs in a fixed scheduling interval. The training speed estimation of bin-packing jobs can not be directly obtained, and it depends upon a neural network model via supervised learning from historical logs. To stabilize and accelerate RL model training, Harmony adopts several techniques including actor-critic, job-aware action space, and experience replay. Putting them together, Harmony outperforms significantly over heuristic baselines.

3.2.3 Elastic Training. In order to maximize the GPU utilization and improve the training efficiency, many novel schedulers support elastic training, which dynamically adjusts the parallelism and resource allocation of workloads to achieve the objectives **C1** **T6**.

Gandiva [152] designs a *Grow-Shrink* mechanism which uses the profiling information to estimate each job's progress rate and then allocates GPUs accordingly. Optimus [115] estimates the loss reduction rate on any placement policies based on a performance model. then it designs a greedy resource allocation scheme to prioritize the maximum marginal loss reduction. This greedy policy successfully maximizes the cluster-wide training throughput. Elan [154] designs several mechanisms to achieve efficient elastic training: *hybrid scaling* can better trade-off the training efficiency and model performance; *concurrent IO-free replication* leverages RDMA to reduce the numbers of IO and CPU-GPU memory copy operations; *asynchronous coordination* avoids the high overhead of start and initialization during re-adjustments. With the integration of the FIFO and backfill scheduling algorithms, Elan successfully improves the cluster resource utilization and reduces the job pending time. AFS [64] is proposed based on the insight that the scheduler should proactively prepare for the resource contention in the future by utilizing the current resources. It considers both resource efficiency and job length for resource allocation while amortizing the cost of future jobs into the calculation of the current share. Besides, a DL training system framework, CoDDL, is also implemented to deliver automatic job parallelization and efficient re-adjustments. EDL [151] also supports elasticity in DL job scheduling. It implements *stop-free scaling* and *graceful exit* to minimize the scale-out and scale-in overheads respectively. Furthermore, EDL optimizes the data allocation pipeline by on-demand and pre-fetching data. MARBLE [56] enables elastic DL training in HPC systems. It determines the optimal number of GPUs through offline profiling and employs

a FIFO-based policy for scheduling. Vaibhav *et al.* [122] designs a job scalability analyzer and a dynamic programming based optimizer to determine the batch sizes and GPU counts for DL jobs. OASiS [10] introduces a primal-dual framework for optimizing the distributed PS-based jobs, which is coupled with efficient dual subroutines to achieve good long-term performance guarantees with polynomial time complexity. During the training, OASiS dynamically scales in or scales out the number of workers and parameter servers for the best resource utilization and training expedition.

Some online scheduling algorithms adopt the elastic training mechanism for datacenter optimization. For instance, GADGET [166] formulates a resource scheduling analytical model for ring-all-reduce DL and uses a greedy approach to maximize the utility of unfinished jobs. It obtains provable guarantee for high performance. AOnline [178, 184] uses the integer linear program to formulate the maximum weighted schedule problem. It schedules a job if its weight is higher than its estimated serving cost to maximize the total weight of scheduled jobs.

A number of works apply RL to optimize the elastic training policy. Specifically, RIFLING [18] adopts K-means to divide concurrent jobs into several groups based on the computation-communication ratio similarity. The group operation reduces the state space and accelerates the convergence speed of the RL model. The RL model only determines the number of GPUs and nodes for each job. This can effectively reduce the action space. A reward function is designed to minimize the resource fragmentation and improve the job throughput. RIFLING chooses the Q-Learning algorithm and allows the RL model to perform online update from historical logs to adapt to the workload variation. DL^2 [116] is another RL scheduler focusing on the PS architecture and dynamically adjusts the resources allocated to the parameter server and workers. It mitigates the optimization instability by a combination of offline supervised learning and online actor-critic reinforcement learning. The RL model also takes the job state and resource state as input and then makes the resource allocation decision for each job. The reward function targets the cluster-wide normalized epoch progress. These techniques enable DL^2 to present satisfactory job latency reduction even for unseen job types.

Some works focus on the optimization of elasticity implementation in practical schedulers, e.g., Kubernetes. Wang *et al.* [147] developed an elastic scheduling framework as plugins in Kubernetes. It uses the training job progress information to allocate and reallocate GPUs to minimize JCT. It efficiently reallocates GPUs based on a *SideCar* process, which introduces an early initialization mechanism for fast reshaping down and achieves non-intrusion to DL training frameworks. DynamoML [21] is a Kubernetes platform which combines KubeShare [158] and Dragon [92] for DL workload scheduling. Dragon [92] fills the gap that existing Kubernetes schedulers fail to manage the distributed training workloads. It resolves this issue by introducing three enhancements including gang-scheduling, locality-aware placement and autoscaling of training workloads. DynamoML also supports scheduling optimization for inference jobs, which will be discussed in Sec. 4.

In addition to the elasticity of GPU resources, DL job configurations can also be dynamically adjusted C3. Pollux [117] aims to achieve higher utilization by automatically configuring the DL training jobs and co-adaptively allocating resources to them. Specifically, it defines *goodput*, a metric for comprehensively measuring training performance including system throughput and statistical efficiency. It designs a joint scheduling architecture to maximize the goodput. At the job-level granularity, Pollux dynamically tunes the batch size and learning rate for the best utilization of the allocated resources. At the cluster-level granularity, Pollux dynamically (re-)allocates resources based on the goodput of all the jobs sharing the cluster as well as other cluster-level objectives (e.g., fairness, JCT). Aryl [85] further extends Pollux by dynamically loaning idle inference GPU nodes to training jobs. It brings higher cluster utilization and lower queuing time. Similar to Pollux, ONES [12] automatically manages the elasticity of each job based on the training batch size. It designs an online evolutionary search algorithm to continuously optimize the scheduling decisions,

which achieves superior performance compared with greedy strategies. More recently, Microsoft presents Singularity [129], an epochal distributed scheduling system for Azure DL training and inference workloads. It achieves transparent preemption, migration and elasticity across a global fleet of AI accelerators (e.g., GPUs, FPGAs). It implements *device proxy* for the decoupled execution and elastic scheduling across the workers. Although it is developed for public cloud services, the promising techniques are also effective in managing private GPU datacenters.

3.3 Implications

The scheduling objective plays an important role in designing schedulers for GPU datacenters. A majority of schedulers consider timing-efficiency and fairness. In contrast, other objectives including deadline guarantee, cost efficiency and accuracy efficiency are not fully explored yet, although they have been thoroughly considered in the conventional cloud and HPC systems. Modern cloud providers are accelerating the pace of building GPU platforms to support a sizable number of training workloads. We anticipate these objectives are also important for training workload management. This inspires researchers and developers to jointly optimize their objectives with the constraints of deadline guarantee and cost.

According to the unique resource consumption features of DL training jobs, datacenter managers can enhance the overall resource utilization and improve users' experience through designing more efficient resource allocation mechanisms, e.g., fine-grained job placement on GPUs, dynamic job parallelism adjustment, adaptive CPU allocation, etc. However, these approaches have their limitations that can hinder their deployment in practice. For instance, adaptive training could change jobs' batch size, learning rate and GPU amount, which can cause model convergence issues. Its generalization for more application scenarios also requires more validations. Job colocation can cause potential performance degradation and fault tolerance issue, which can make users unwilling to adopt this feature. How to address these practical issues is a promising and challenging future direction. We look forward to seeing more progress in this topic.

Although different scheduling algorithms for conventional workloads and systems have been extensively studied for decades, it still requires more efforts to design effective scheduling algorithms for large-scale GPU datacenters to reduce the operational cost and improve the job throughput. The rapid development of AI technology motivates researchers to investigate the possibility of using machine learning to optimize scheduler designs. From our summary, ML-based schedulers have shown their effectiveness in some scenarios. However, the datacenter managers are still concerned about the reliability and scalability of these ML-based schedulers. We expect more research works will be performed to address these concerns and make these schedulers more practical.

4 SCHEDULING DL INFERENCE WORKLOADS

As more DL-based applications are released as online services in our daily life, it becomes more critical to manage and schedule large-scale inference workloads in the GPU datacenter. Different from the resource-intensive and long-term training workloads, inference jobs have unique characteristics and requirements (Sec. 2.1.2), which demand new scheduling solutions. Similar as Sec. 3, we categorize these inference scheduling techniques based on their objectives, and resource consumption features. Then we give some implications from these works at the end of this section. Table 2 summaries the relevant papers and their features.

4.1 Scheduling Objectives

We first review prior works based on the scheduling objectives.

Table 2. **Summary of schedulers for DL inference workloads in GPU Datacenters.**

Year	Scheduler	Objectives	Approaches	Advantages	Batching	Colocate	Cloud	Exp. Scale	Source Code
2017	Clipper [25]	♣♥♠	Query-level caching; Layered architecture	General abstraction for model selection	✔	-	-	M	✔
2018	Space-Time [68]	♠✿	GPU sharing across space and time	Performance isolation under sharing	✔	✔	-	S	-
	Ease.ml [87]	♣	Multi-tenant model selection	Homogeneous declarative inference platform	-	-	-	-	✔
	HiveMind [107]	♠	Sharings of pipelines, weights, and layers	Multi-model training and inference	✔	✔	-	S	-
2019	MArk [172]	♥♦	Predictive autoscaling on serverless instances	Flexible to burst requests	✔	-	✔	S	✔
	Tolerance Tiers [55]	♣♥♦	Service Version Ensembling	Explicit accuracy-latency trade-off in requests	-	-	✔	S	-
	ParM [77]	♥	Coded-computation via a learning-based approach	Erasure-coded resilience for inference	✔	-	-	M	✔
	Gilman et al. [45]	♥♠	Preloading model into GPU memory	DNN model execution caching	-	✔	-	-	-
	Nanily [136]	♥♠	Adaptive batching; autoscaling	Batch size adjustment by remaining time	✔	-	-	S	-
	RRL [118]	♥	Region-based Reinforcement Learning	Parallelism configuration tuning	✔	✔	-	M	✔
	TrIMS [30]	♥♠✿	Multi-layered caching across FaaS	Memory efficiency by sharing	✔	✔	✔	S	✔
	Ebird [27]	♥♠✿	CUDA stream paralleism; GPU-side memory pool	Transfer-computation overlapping	✔	✔	-	S	✔
2020	GSLICE [31]	♠✿	Dynamic GPU resource apportioning	Efficient fine-grained sharing	✔	✔	-	S	-
	Clockwork [51]	♥♠	Consolidating choice	Predictable E2E performance	✔	-	-	M	✔
	Irina [150]	♥♠✿	Batching, colocation and preemption	Graceful general preemption	✔	✔	-	S	-
	PERSEUS [83]	♥♦♠	Performance and cost characterization on Cloud	Cost savings under GPU instances	✔	-	✔	S	✔
	AutoDeep [89]	♥♦♠	BO and DRL	Cloud configuration and device placement	-	✔	✔	S	-
	DyBatch [179]	♥♠	Task slicing and reordering	Fine-grained batching; Fairness-driven scheduling	✔	✔	-	S	-
	Inferline [24]	♥♦	Low-frequency planner; high-frequency tuner	Near-optimal scaling cost-efficiency	✔	-	✔	M	✔
2021	INFaaS [121]	♥♦♠	VM- and model-level autoscaling	Automatical model variants selection	-	✔	✔	-	✔
	Mendoza et al. [100]	♥	Latency degradation prediction during colocation	Safe colocation	-	✔	-	M	-
	Morphling [142]	♦♠	Model-agnostic meta-learning	Performance prediction under different configuration	✔	✔	✔	-	✔
	Abacus [29]	♥♠	Runtime operator scheduling	Deterministic latency under colocation	-	✔	-	M	✔
	MIG-SERVING [135]	♥♦	Greedy algorithm, GA, and MCTS	MIG enabled inference scheduling	✔	✔	-	S	-
2022	Cocktail [52]	♣♥♦	Weighted majority voting policy	Ensembling-based model selection	-	-	✔	-	-

Objectives: ✿ Utilization ♣ Accuracy ♦ Cost ♥ Latency ♠ Throughput; **Experiment Scale:** S (Single Node) M (Multi Nodes) -: no evaluation on a physical cluster or not clearly specified.

4.1.1 Efficiency. As discussed in Sec. 2.2.3, the main objective for scheduling an inference workload is to improve its efficiency. This includes the reduction of inference latency and cost, and improvement of the prediction accuracy I2. The challenge here is that there exist trade-offs among different efficiency goals. Here we discuss the techniques to improve each goal as well as to jointly balance and improve them.

1) Accuracy efficiency. Improving the prediction accuracy is a perpetual objective in an inference system. To achieve this, one approach is to collect a set of models, and select the best one to predict the result for each input query. The scheduling decision made includes model selection and resource allocation among different candidates. Ease.ml [87] leverages the input and output shape information of the query sample to automatically select the model. It estimates the potential accuracy improvement of each candidate model and then picks the highest one for actual inference. It also formulates the cost-aware model selection process under both single-tenant and multi-tenant settings with multi-armed bandit and Bayesian Optimization. Another effective approach is model ensemble, which combines the prediction results from multiple models to improve the prediction accuracy and generalization. Clipper [25] examines the benefits brought from the model ensemble in computer vision tasks and applies a linear ensemble method to compute a weighted average of the base model predictions. The linear weights are decided by bandit- and learning-based approaches. Rafiki [148] leverages an RL model to determine the model set for the ensemble. This model is also used to identify the final optimal model combinations, and tune critical parameters, e.g., batch size.

2) Latency efficiency. An inference system should have a satisfactory response time, even for burst and fluctuating query requests C7. The latency requirement poses challenges for the scheduler to decide which jobs to be prioritized in the job assignment and rearrangement process. This objective can be achieved via carefully optimizing the resource allocation.

It is common to launch multiple inference execution instances concurrently to meet the corresponding latency requirement as much as possible due to the low GPU utilization for each request C5. Therefore, the inference scheduler can make scheduling decisions aiming at scaling up resources according to the request density to maintain a satisfactory latency. Clipper [25] conducts linear scaling of inference instances and uses separate docker containers to isolate different models. It replicates the model containers according to the number of queries and applies adaptive batching independently for each model due to the varied execution time. MArk [172, 173] scales the inference

instances with the cloud services. It selects and combines different cloud services like AWS EC2 and Lambda in order based on their prices and scaling abilities. Also, it monitors the system loads and request queuing situations proactively and leverages Lambda to scale up instances when there exist requests violating the latency demands. InferLine [24] targets the pipelined inference workloads with multiple stages. It monitors the frequency of queries to each model and makes the scaling decisions of each component separately, to maintain the latency SLOs even during sharp bursts.

A number of works aim to provide bounded latency for inference execution at the system level considering its deterministic execution [1]. Clockwork [51] discovers that many DL inference models have deterministic performance because of the underlying deterministic computations. Thus, it guarantees deterministic latency by alleviating the uncertainty introduced by other components of the system. To overcome the uncertainty from memory and cache usages, hardware interactions, and other uncontrollable external performance variations, Clockwork consolidates the configurations among all the system layers during the inference execution, by proactively controlling the memory allocation and deallocation, and disabling concurrent execution of multiple inference workloads to eliminate the interaction. Reducing the parallelism of execution eliminates the interference from other tasks, but inevitably brings lower throughput and resource utilization. To address this issue, Abacus [29] tries to guarantee SLO for query requests under the GPU co-location scenarios. It controls the execution sequence and the co-location situation proactively, rather than the default random-ordered execution overlap. Given the explicit order and specific co-location operators on GPUs, Abacus could predict the estimated running time under co-location from the early offline profiling stage. Based on the estimation, the query controller schedules all the simultaneous inference workloads to guarantee the QoS by searching the optimal execution sequence of DNN operators. ParM [78] migrates the concept of erasure codes from distributed computing to model inference systems, and uses learning-based coded computation to introduce redundancy and thus supports the recovery of inference executions with tail latency or failures.

Some solutions proactively schedules the inference workloads and rearranges the execution sequence at the job level. Irina [150] is the first online inference scheduling system, modeling the satisfaction of latency demands as a scheduling problem. By leveraging preemption for DL inference workloads, Irina dynamically decides whether to preempt the ongoing query and launch the later arrived one, which brings significant reduction of average completion time for inference workloads. The main challenge is that existing ML frameworks are not designed and suitable for preemption during execution. Irina carefully manages the preemption process by adding an exit node to the existing dataflow graph of the inference workload, thus enabling safe preemption at arbitrary moments. It is necessary to have more runtime information about the inference workloads for effective scheduling. Kube-Knots [137] makes predictions about the resource utilization of each inference workload from two aspects. From the spatial aspect, Kube-Knots discovers the correlations across different resource utilization metrics, and then forecasts the future resource utilization. From the temporal aspect, Kube-Knots predicts the peak inference usage and tries to avoid co-locating jobs which could attain peak consumption of the same resources simultaneously.

3) Cost-efficiency. The monetary cost becomes one of the main concerns when using public cloud resources to deploy DL inference workloads. Considering the varied compute capabilities and prices for different types of resources and services, a couple of schedulers implement many mechanisms to achieve cost-efficient inference. MArk [172, 173] analyzes the cost of utilizing different levels of resource abstractions in Amazon Web Services (AWS) and Google Cloud Platform (GCP) for inference. It finds that the Infrastructure-as-a-Service (IaaS) provides better cost efficiency than Content-as-a-Service (CaaS), while Function-as-a-Service (FaaS) could compensate for the relatively long cold start latency of IaaS at the cost of increased costs. Small instances with advanced CPUs and burstable instances are also recommended. For GPU instances, the cost can be greatly

reduced by batch processing as well. Given different levels of capability, scalability, and pricing, MArk greedily selects the most cost-effective type of instances and leverages the spot instances for cost-saving. AutoDeep [89] considers not only the resource type in the cloud but also the device placement for DL inference. It leverages Bayesian Optimization for the nearly optimal cloud configuration and Deep Reinforcement Learning for the nearly optimal device placement. Constrained by the SLO requirements, AutoDeep performs joint optimization to minimize the total cost of the inference execution. Cocktail [52] develops a distributed weighted auto-scaling policy and leverages the spot instances to minimize the cost.

4) Trade-offs between accuracy, latency and cost. The objectives of accuracy, latency and cost are not independent C6. Improving one goal may possibly compromise another goal if the solution is not designed properly. Besides, users may also have their specific expectations about different objectives. This motivates researchers to explore the trade-offs between these objectives, and devise more flexible and comprehensive scheduling systems.

The adoption of multiple models can improve the model inference accuracy, but might also increase the response latency and cost. Several works track the latency and prediction accuracy of different models and implement mechanisms to select the most appropriate ones determined by the schedulers. Clipper [25] introduces a model selection abstraction, which supports both single model selection and model ensemble selection. It executes the inference for all the models and combines their results. It observes the corresponding accuracy and latency feedback continuously to make the selection with a best-effort search method. Model-Switching [176] pursues the trade-offs between computational cost and service accuracy by switching different model variants proactively, to improve the accuracy of responses under the latency constraint. By maximizing the ratio of correct predictions returned within the deadline, it makes selections among model variations with different computation demands and accuracy. Cocktail [52] balances the cost with accuracy and latency on the public cloud via the optimization of the model ensemble. With a dynamic model selection policy that searches models tightly with the accuracy and latency bounds, Cocktail reduces the candidates in the ensemble and accomplishes fast and efficient model selection.

Some schedulers allow users to specify their demands about accuracy, latency and cost, and make scheduling decisions directly according to the demands. Tolerance Tiers [55] discloses the efforts the system can offer to achieve each objective, and makes users programmatically select their demands. Observing that improving the accuracy of some extreme requests can increase the latency greatly, Tolerance Tiers relaxes and sacrifices the accuracy demand to improve the latency and service cost. Each tier defines an error tolerance to indicate the tolerable accuracy loss, and an optimization objective. Then, Tolerance Tiers optimizes the objective under the constraint of the maximum error tolerance. INFaaS [121, 155] also asks for the performance and accuracy demands from the users. It generates some variants from the existing models with different specific parameters (e.g., batch size, hardware configurations, hardware-specific parameters). After one-time profiling for each variant, INFaaS selects the model variant based on the resource consumption and performance information from the profiling to serve users' requests. Since each model variant may have different performance and monetary costs during execution, INFaaS makes the selection via a heuristic-based approach, which selects the variant with the minimum cost while meeting the SLO constraint, or upgrades existing variants with higher performance to fulfill the burst queries.

4.1.2 System Throughput. Another important objective for scheduling inference workloads is to improve its throughput capability. The techniques to achieve this goal is summarized as follows.

1) Batching execution. One common approach is to batch multiple inference queries, and execute them concurrently. Handling individual inference queries usually leads to GPU underutilization C5. Hence, batching inference can efficiently improve the utilization and reduce the system

overhead. Like job queuing in parallel job scheduling, batching multiple queries can delay the execution of the requests that come earlier, and possibly jeopardize the SLO requirement. Setting a proper batch size is critical to balance such delays and system throughput. Most schedulers dynamically adjust this hyperparameter based on the actual SLO requirement and queuing situation.

First, some schedulers adopt heuristic methods to tune the batch size. Clipper [25] and Rafiki [148] apply the practical Additive-Increase-Multiplicative-Decrease (AIMD) algorithm to adjust the inference batch size. Specifically, the batch size is additively increased by a fixed amount until the latency of processing a batch exceeds the latency requirement and then multiplicatively decreased by a fixed percent. Clipper evaluates that AIMD is simple yet effective and adaptive to the changing throughput of a model in special cases. It also aggressively delays the execution of queries under moderate loads to the subsequent batch, which can bring a significant throughput increase for some models. GSLICE [31] also applies a similar adaptive and self-learning approach to determine the optimal batch size. It carefully tracks the execution time and increases the batch size until the execution of the last batch exceeds the SLO requirement. Ebird [26, 27] proposes a novel elastic batching mechanism, which runs different CUDA streams concurrently for different batches. Motivated by the observation that processing multiple queries in a large batch has similar performance as multiple CUDA streams with smaller batch sizes, Ebird dynamically adjusts the batch size per stream to utilize the spare GPU resources and fulfill the whole GPU. In each scheduling round, it selects and launches a job based on its batch size and remaining GPU resources in a best-effort manner, squeezing the full GPU utilization and minimizing inference queuing delay.

Second, some schedulers propose optimization-based methods to balance the inference delay and throughput. MArk [172, 173] considers the maximum time of delaying a query, profiles the processing rate without batching, and searches for the optimal batch size under the SLO constraint. Nanily [136] presents the upper bound of the batch size by retrieving the maximum remaining time for the requests, calculated as the remaining time towards the deadline subtracted by the least queuing time for the available resources. It then derives the corresponding batch size, which makes the inference execution time equal or close to the maximum remaining time. DyBatch [179] considers the fairness of the delay for each independent workload when batching. It implements fine-grained batching schemes along with fairness-driven scheduling, which can compensate for the deviation of slowdown for small inference workloads. DyBatch organizes the workload batches in a time-sharing manner and selects the batch with the lowest resource utilization for running, thus maintaining fairness and minimizing the slowdown of each workload.

2) Caching and reusing. Another widely-used strategy for throughput improvement is caching and reusing the prediction results across different requests **C7**. The scheduler selects the request that benefits most from caching and allocates proper resources. This can be done at two levels.

The first direction is to perform optimization at the query level. To provide fast responses to different queries, the inference system can cache the inference execution and prediction results for burst queries. Clipper [25] maintains a prediction cache for requests with the same target model and the query input. Then it can produce the results for some queries without evaluating the model, thus increasing the inference throughput. Clipper also applies an LRU cache eviction policy to optimize the caching efficiency. However, this approach may be less efficient when the queries do not have high similarities in practical scenarios, which leads to high cache miss rates and evictions.

The second direction is to perform optimization at the device level. Gillman *et al.* [45] proposed to cache the DL models instead of the inference results. It schedules models to be loaded into the limited GPU memory to maximize the probability of servicing an incoming request without swapping the models in and out of the memory, thus accelerating the inference by eliminating the cold start latency with cache hits. The caching and eviction policy considers many runtime aspects of DL inference workloads, including model size, frequency, model accuracy, and speed.

This work also discusses some future directions for more dynamic caching mechanisms and policies, like framework-level GPU memory-friendly optimization, proactively loading and evicting, and cluster-level GPU memory allocation. To address the limitation of GPU memory, GSLICE [31] and HiveMind [108] explore the common GPU memory component in different inference models, and propose to save GPU resources via memory sharing. Particularly, GSLICE enables efficient GPU memory sharing by allowing the reuse of model parameters via modifications to the DL framework, exposing the CUDA address to different instances. Therefore, it supports loading the inference model-related parameters only once to the GPUs, resulting in faster module loading. HiveMind extends the shared content and brings more possibilities of shared model weights and layers across different inference workloads, saving the overhead from both model loading and inference evaluation. TrIMS [30] organizes the memory sharing of different models in a more systematic design. The model resource manager in TrIMS offers a multi-tiered cache for DL models to be shared across users' FaaS functions and enables DL model sharing across all levels of the memory hierarchy in the GPU, CPU, local storage, and remote storage in the cloud. TrIMS reconciles the lifecycle of model memory consumption and carefully handles the cache misses and evictions. It also considers multi-node, isolation, and fairness problems during sharing. Extensive evaluations on different models show its general abilities to improve the inference performance by mitigating the model loading overhead.

3) System configuration tuning. Besides the optimization techniques detailed above, there exist some schedulers leveraging end-to-end configuration tuning to improve the system throughput. Morphling [143] formulates the optimal configuration search as a few-shot learning problem. Then it adopts model-agnostic meta-learning (MAML) [40] to combine offline meta-model training for inference serving performance modeling under varied hardware and runtime configurations, and performs online few-shot learning to predict the service performance. Based on the prediction, Morphling auto-tunes the resource provisioning configurations and makes better scheduling decisions. RRL [118] concentrates on optimizing the parallelism configurations from different levels, including request level parallelism and intra-request level (inter-op and intra-op) parallelism, which have strong impacts on the latency of the entire system. RRL utilizes a region-based RL method to tune the parallelism configurations and reduce the inference processing latency, based on the system performance similarity between different configurations within a similar parallelism setting.

4.2 Resource Consumption Feature

Similar to training workloads, the inference schedulers can also be categorized based on the resource consumption feature. Below we detail the scheduling solutions designed to target these features.

4.2.1 Colocation and resource sharing. From Challenge C5 in Sec. 2.2.3, executing one inference request can lead to GPU resource underutilization. The recent development of GPU architecture designs motivates the GPU sharing from both the hardware perspective [2, 4] and software perspective [127, 168]. GPU sharing across different inference requests can significantly improve the resource utilization by better leveraging GPUs' parallel compute capability. However, it can also incur the risks of violating the latency requirements due to the uncertain running time.

One line of research works adopt the static colocation strategy to guarantee the inference latency. Space-Time [68] calls for both space-sharing and time-sharing for DL inference with GPUs. It preserves the predictability and isolation during virtualization by monitoring the inference latency per kernel. Then it improves the utilization by merging concurrent small kernels into larger super-kernels that can fill up the GPU utilization under the time-sharing mechanism. Irina [150] only considers a safe GPU colocation situation based on the peak GPU requirement of the workloads, when their total GPU requirement does not exceed the capacity. It assumes there is no interference

and slowdown on the job completion time and heuristically places the newly arrived job on the GPU with the smallest JCT. Nexus [126] also applies a heuristic approach to select the requests to be co-located on the same GPU. First, it identifies the most appropriate batch size for throughput and SLO needs of the existing inference workload. Afterward, it establishes all possible combinations within a GPU's duty cycle on a single GPU in a best-fit manner, and maximizes the utilization without violating the latency requirement.

Some other works introduce dynamic colocation mechanisms for managing the inference workloads. GSLICE [31] is an inference system to systematically achieve safe and efficient GPU sharing. It leverages spatial GPU multiplexing for different inference requests on top of the state-of-the-art GPU spatial multiplexing framework MPS. It intensively evaluates the colocation performance with and without MPS, and with the resource provisioning limits. It discovers that the colocation interference could be amortized under the careful configuration of the resource limits. The performance improvement reaches a point of diminishing returns (i.e., kneepoint) after certain configurations, which has a non-linear relationship with the throughput and latency of the inference. Based on these observations, GSLICE tracks the kneepoint of different inference workloads and partition the whole GPU according to their kneepoints. It also designs a hot-standby mechanism to dynamically adjust the resource limit configuration of the specific inference job, along with other implementation optimizations including minimizing the data transfer overhead. MIG-SERVING [135] leverages the hardware support for GPU virtualization (i.e., MIG [3] on NVIDIA A100[2]) for efficient GPU colocation. With MIG, an A100 card could be dynamically partitioned into several instances with smaller hardware capacities under some hard constraints. MIG-SERVING discovers that the throughput of most models does not grow linearly with the increase of resources on different instances. It establishes a reconfigurable scheduling problem and applies a generic algorithm to find a sub-optimal and feasible solution, and improve it via a search-based method.

Since the colocation of multiple inference workloads multiplexes the GPU, it is hard to measure and predict their running time due to the colocation interference. Therefore, several inference scheduling systems focus on estimating and handling the colocation interference. INFaaS [121, 155] targets the inference services in the cloud. It identifies the colocation interference caused by the shared hardware resources. Then it allocates the available resources to the interfered instances by migration or VM-level scaling. The evaluation shows that INFaaS can save the monetary cost by GPU sharing and satisfy the latency requirements by VM-level scaling. Mendoza *et al.* [100] proposed an interference-aware scheduler for inference workloads, which proactively considers the impact of interference on the latency from co-location. By predicting the performance degradation, it minimizes the latency degradation and reduces the SLO violations. It improves the prediction accuracy by exploiting the similarity of co-location configurations, including inference model attributes and machine types. PERSEUS [83] compares the cost and throughput under exclusive execution and colocation for inference workloads. It concludes that the mixed ratio of different models affects the cost efficiency of colocation. The interference on the model and data loading time during cold start also differs across different models because of the cache and data transfer requirements under colocation.

4.2.2 Heterogeneous Resources. Several works exploit both CPU and GPU machines of different sizes for DL inference, especially for the cloud environment. MArk [172, 173] characterizes and compares the cost and performance of inference workloads on different types of instances available in GCP and AWS. It concludes that smaller instances with advanced CPU models achieve a higher performance-cost ratio. It also suggests that GPU instances can achieve lower per-request cost and smaller inference latency than CPU instances with appropriate batch sizes. PERSEUS [83] performs a similar cost-efficiency characterization, considering instances with multiple onboard GPUs. It

examines that some DL models with high GPU utilization could introduce intensive interference with other inference workloads in multi-GPU instances. AutoDeep [89] and Cocktail [52] focus more on the price of different cloud configurations and search for the best configuration according to the pricing information.

4.3 Implications

In Sec. 4.1.1 we mainly discuss the monetary cost of deploying DL inference workloads. In a GPU datacenter, how to improve the efficiency of energy cost is also critical. While this objective has been extensively explored for training workloads (Sec. 3.1.1), it is relatively less studied for the inference workloads. Some works [58, 113] provide some energy characterizations of production DL inference clusters. Kube-Knots [137] presents simple energy efficiency comparisons of inference workloads between GPUs and CPUs. It is necessary to comprehensively explore the energy optimization of different DL inference models with different types of compute resources, and design more sophisticated energy-saving mechanisms with the consideration of latency and resource utilization. This will be a promising research direction in the future.

Most of the above works treat the inference jobs as a black box for management and optimization. In reality, an inference pipeline may consist of several separate stages to fulfill the query. It is interesting to consider the characteristics of internal stages to optimize the execution in a white-box manner. PRETZEL [82] leverages this idea, which stores and re-uses the model parameters and computation among similar white-box representations of pipeline stages to reduce the resource utilization. We expect more future works will focus on the optimization of inference workloads at this sub-request level.

As the scale of DL models grows faster than the GPU compute capacity, it is more difficult to accommodate single models on a single GPU or machine. This problem becomes more serious when cloud users adopt the resource-constraint serverless platform for inference services. A natural solution to this problem is model partitioning, Gillis [165] splits the network layers and minimizes the inference latency via dynamic programming. It also adopts some searching methods like Bayesian Optimization and RL to minimize the cost. AMPS-Inf [69] jointly considers the model partitioning and resource allocation by calculating the optimal execution and resource provisioning plans under the constraint of the response time in the serverless platforms. It is worth more effort to explore the methodologies of serving large models with limited resources for different objectives.

Some inference systems for CPU clusters predict the future trends of query requests to satisfy the latency requirement. For instance, BARISTA [11] predicts the future workload patterns based on the historical data and estimates the required resources for the application to maintain the SLOs. Then it makes resource provisions based on the difference between the desired throughput and latency requirement with the current ones. Swayam [50] ensures an appropriate replica of service instances by predicting the load and auto-scales the service in a fully distributed way along with the self-reclamation of resources. It adopts linear regression to predict the request arrival rate over short periods to tolerate the rapid changes in time. Since these solutions are hardware-agnostic, they can be applied to the GPU clusters as well. We also expect this strategy can inspire more solutions dedicated to the GPU clusters.

5 SCHEDULING OTHER TYPES OF DL WORKLOADS

The previous two sections categorize the scheduling of general training and inference workloads, respectively. In this section, we discuss the works for optimizing other types of DL workloads. Table 3 presents the past works for hyperparameter optimization workloads as well as hybrid training and inference workloads, with the detailed description as below.

Table 3. **Summary of schedulers for other workloads in GPU Datacenters.**

Type	Year	Scheduler	Objectives	Approaches	Advantages	Exp. Scale	Source Code
HPO	2017	HyperDrive [119]	♣	Dynamic Probabilistic Accuracy Prediction	JCT Reduction	S	-
	2019	HyperSched [91]	♠	ASHA; Dynamic Resource Allocation	Efficient HPO under DDL	S	✔
	2021	Hermes [131]	♣♦	Time-sharing Execution with Low Overhead	JCT Reduction	S	-
	2021	RubberBand [101]	♣♦	Model JCT and Cost Prior to Runtime	Cost-Effectiveness	S	-
	2021	SEER [33]	♣♦	Dynamic Resource Allocation	Cost Constraint HPO	S	-
	2021	Fluid [169]	✿ ♣♦	Job Packing; Elastic Training	Better Resource Utilization	S	✔
Hybrid	2018	Rafiki [148]	♥♦	RL to optimize accuracy and latency	Unified platform for training and inference	S	✔
	2019	Kube-Knots [137]	♦✿	Dynamic container resizing	GPU utilization-aware colocation	S	-
	2020	CMS [86]	♥✿	Common architecture for trainers and modelets	Continuous learning	S	-
	2022	Aryl [85]	✿♥	Capacity Loaning	Cluster Size Extension	L	-

Objectives: ✿ Utilization ♣ Makespan ♦ Cost ♥ Accuracy ♠ DDL; **Experiment GPU Scales:** the scale of physical testbed. S $(0, 30]$ M $(30, 60]$ L $(60, 120]$ XL $(120, \infty]$ -: no evaluation on a physical cluster or not clearly specified.

5.1 Hyperparameter Optimization Workloads

A Hyperparameter Optimization (HPO) job aims to identify the best hyperparameters for a DL task. Technically speaking, HPO belongs to the category of DL training workloads. Here we discuss it separately because it has unique features compared to the general DL training jobs. Specifically, a HPO job typically needs to search a set of hyperparameter configurations. Each configuration is associated with a *trial* [101], which is a common DL training workload containing training and evaluation procedures. However, in the HPO context, these trails are extremely similar and orchestrated by a HPO scheduling policy. To accelerate the HPO process, the policy can kill poor-performing trails through early stopping and allocate more resources to promising trails. These optimizations on HPO workloads can deliver remarkable acceleration.

Accuracy efficiency. Hermes [131] is a scheduler to expedite HPO workloads in GPU datacenters. It provides a container preemption mechanism to enable migration between DL jobs with minimal overhead. Besides, it also considers the algorithmic property of HPO workloads and devises a convergence-aware scheduling algorithm to favor promising hyperparameter configurations. HyperSched [91] aims to boost the accuracy performance of HPO workloads within the given time and resource budgets. In an HPO workload, the promising hyperparameter trial generally has higher accuracy at the early training stage. Then, HyperSched allocates more resources to the promising trials and terminates unpromising ones. With this technique, HyperSched can parallel orders of magnitude more hyperparameter trials and therefore maximize the final accuracy substantially. As an extension of HyperSched [91], SEER [33] replaces the resource budget with the monetary cost budget in the cloud. Without the resource constraint, SEER can launch numerous training trials at the early training stage to explore promising hyperparameter combinations. Then it will terminate poor training trials and allocate more cost budget to promising trials during the training progress. The accommodation of more trials at the hyperparameter exploration stage can improve the final model accuracy. HyperDrive [119] is another scheduler framework for hyperparameter exploration as well. It develops an accuracy curve fitting model to extrapolate the accuracy in the subsequent training iterations. The accuracy prediction engine allows HyperDrive to terminate low-quality jobs early and adjust the resource allocation dynamically. Moreover, Fluid [169] further leverages the job packing mechanism to improve resource utilization and accelerate the HPO process.

Cost efficiency. RubberBand [101] aims to reduce the monetary cost of an HPO job with the constraint of timing budget in the cloud. Since the optimal amount of resources for an HPO workload differs in early and later stages of the hyperparameter search and model optimization processes, RubberBand needs to jointly accommodate the job throughput, resource amount and cloud pricing. By building a profiling model, RubberBand can predict the job throughput and corresponding cost for a given resource allocation policy. Then it uses this model to generate an optimal cost-efficient solution for satisfactory cost reduction.

5.2 Hybrid of Training and Inference Workloads

In Sec. 3 and Sec. 4 we consider the scheduling techniques for training and inference solutions separately. Actually, there are some DL systems that host these two workloads in a unified manner. Due to the distinct features of the two types of workloads, new solutions are needed for efficient scheduling in GPU datacenters. Below we review and summarize these works.

End-to-end DL development. Some works consider the entire pipeline of DL development and deployment, including model training, inference, as well as periodically retraining and updating. CMS [86] designs a continuous machine learning and serving platform, which orchestrates the model training executions, model deployments, and model update services. It unifies different training jobs into a simple trainer contract and provides common essential pipeline abstracts of training. Then the scheduler monitors the resource consumption of these resource-intensive training tasks to avoid contention. Other techniques including model validation are also applied to guarantee the quality of newly-trained models. Rafiki [148] proposes to reuse the datasets and parameters across different training and inference jobs. It implements a unified distributed dataset storage and a parameter server. The parameter server is kept in memory and shares model parameters across different trials in the training and dumped for later reuse in the inference. Other common underlying components are also shared across training and inference workloads to reduce the operational cost, e.g., storage, communication protocols, and compute resources.

Mix of training and inference development. Some works optimize the datacenter with mixed workloads of DL training and inference. Kube-Knots [137] minimizes the resource waste by allocating offline batch jobs to better utilize the spare resources. It discovers that the GPU energy efficiency increases with its utilization. Therefore, achieving higher GPU utilization indicates higher energy efficiency and less resource wastes. To this end, Kube-Knots colocates the predictable GPU batch jobs with the online inference workloads, as the inference jobs generally underutilize the GPUs. During the scheduling, it also predicts the peak resource utilization of the jobs from their online resource usages. Aryl [85] also observes the resource can be wasted due to the resource over-provisioning for burst inference requests. It proposes the concept of capacity loaning, which allows the inference cluster to loan the idle GPU servers during low-traffic periods to run training jobs. Since the preemption cost is not negligible, it minimizes the total number of preemptions during the scheduling. Combined with the elasticity on resource demands for part of training jobs, Aryl successfully guarantees the timely resource allocation to inference jobs via a heuristic method.

5.3 Implications

In addition to optimizing general DL workloads, there are some opportunities for further resource efficiency improvement. One direction is to apply specific optimizations for hyperparameter search workloads. Compared with the workload-agnostic manner, it can deliver over one magnitude resource and time conservation. However, how to integrate this mechanism well into the scheduler and coordinate with general DL workloads is a challenging topic, requiring more future research investigation. Besides, considering the whole DL model development pipeline instead of focusing on a certain stage can bring extra system performance enhancement. For instance, breaking the shackles between the training and inference cluster resource not only considerably diminishes the queuing delay of training workloads but also improves the model serving quality. These directions deserve more attention in future research on GPU datacenter scheduling systems.

6 CONCLUSIONS AND OUTLOOKS

The scheduler design is an ever-lasting topic in the system research community. The prosperity of DL workloads considerably pushes forward the progress of this research area in all stages of

scheduling. The unique features exhibited by DL workloads advocate novel DL scheduler design to manage the GPU resources and jobs in a more intelligent and efficient manner. Our comprehensive summary draws three conclusions. First, new works prefer to adopt advanced algorithms (e.g., RL, MAML), which can significantly improve the scheduling performance. Second, it is necessary to take advantages of emerging hardware resources (e.g., heterogeneous GPU, GPU colocation and sharing, elastic training) when designing efficient schedulers. Third, new scheduling systems are motivated by the emerging DL workloads and applications, as well as users' new requirements.

DL workload scheduling in GPU datacenters remains premature. There are multiple interesting future research directions, as summarized below.

DL workloads. The diverse DL workloads pose different challenges to the scheduler. Domain-specific schedulers are required for extreme efficiency for specific applications. Therefore, a set of novel DL workloads with special resource requirements (e.g., HPO, hybrid workloads) call for more research efforts. Searching-based DL workloads like HPO eagerly rely on being served as early as possible to get search results earlier and thus optimize the search direction. Training extremely large models like Transformers needs extensive and high-performance resources. Surging needs from DL jobs for debugging purposes should also be balanced with those production jobs under hybrid situations. Otherwise, the under-efficiency problem will arise. Another important direction is the co-design of both the scheduling system and DL framework. Better scheduling decisions could be made by negotiating the fine-grained resource demand of workloads and delegating framework-level control to the scheduling system.

Scheduling decision making. Many existing schedulers may encounter problems with GPU datacenters at scale. First, a lot of scheduling systems require additional information about the workload from users or online profiling, posing great challenges when facing numerous workloads and resources. Second, some schedulers form the decision-making process as an optimation problem, which cannot be solved within an acceptable time online. Other systems such as online monitoring of DL workloads, resource management, and coordination also add difficulty to the operations of large-scale GPU datacenters.

Underlying hardware resources. GPU datacenters are also growing at an alarming speed. It is common for modern GPU datacenters to contain heterogeneous and complex generations of GPUs and other accelerators. Schedulers need to make scheduling decisions based on different affinities between workloads and GPUs. GPUs may also reveal different capabilities for serving the workloads, e.g., hardware-level support for multiplexing, advanced support for low-precision ALU. Other resources like emerging networking topology also draw the attention of schedulers for performance efficiency.

REFERENCES

[1] 2022. Multi Model Server: a tool for serving neural net models for inference. https://github.com/awslabs/multi-model-server.

[2] 2022. NVIDIA A100. https://www.nvidia.com/en-sg/data-center/a100/.

[3] 2022. NVIDIA Multi-Instance GPU. https://www.nvidia.com/en-us/technologies/multi-instance-gpu/.

[4] 2022. NVIDIA Multi-Process Service. https://docs.nvidia.com/deploy/mps/index.html.

[5] 2022. OpenPBS. https://www.openpbs.org/.

[6] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. TensorFlow: A System for Large-Scale Machine Learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI '16)*.

[7] Marcelo Amaral, Jordà Polo, David Carrera, Seetharami Seelam, and Malgorzata Steinder. 2017. Topology-Aware GPU Scheduling for Learning Workloads in Cloud Environments. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '17)*.

[8] Remzi H Arpaci-Dusseau and Andrea C Arpaci-Dusseau. 2018. *Operating systems: Three easy pieces*. Arpaci-Dusseau Books LLC.

[9] Yixin Bao, Yanghua Peng, and Chuan Wu. 2019. Deep Learning-based Job Placement in Distributed Machine Learning Clusters. In *IEEE Conference on Computer Communications (INFOCOM '19)*.

[10] Yixin Bao, Yanghua Peng, Chuan Wu, and Zongpeng Li. 2018. Online job scheduling in distributed machine learning clusters. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications (INFOCOM '18)*.

[11] Anirban Bhattacharjee, Ajay Dev Chhokra, Zhuangwei Kang, Hongyang Sun, Aniruddha Gokhale, and Gabor Karsai. 2019. BARISTA: Efficient and Scalable Serverless Serving System for Deep Learning Prediction Services. In *2019 IEEE International Conference on Cloud Engineering (IC2E)*.

[12] Zhengda Bian, Shenggui Li, Wei Wang, and Yang You. 2021. Online evolutionary batch size orchestration for scheduling deep learning workloads in GPU clusters. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '21)*.

[13] Marcel Blöcher, Lin Wang, Patrick Eugster, and Max Schmidt. 2021. Switches for HIRE: Resource Scheduling for Data Center in-Network Computing. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '21)*.

[14] 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).

[15] Dheeraj Chahal, Mayank Mishra, Surya Palepu, and Rekha Singhal. 2021. Performance and Cost Comparison of Cloud Services for Deep Learning Workload. In *Companion of the ACM/SPEC International Conference on Performance Engineering*.

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

[17] Chen Chen, Qizhen Weng, Wei Wang, Baochun Li, and Bo Li. 2020. Semi-dynamic load balancing: efficient distributed learning in non-dedicated environments. In *Proceedings of the 11th ACM Symposium on Cloud Computing (SoCC '20)*.

[18] Zhaoyun Chen. [n.d.]. RIFLING: A reinforcement learning-based GPU scheduler for deep learning research and development platforms. *Software: Practice and Experience* ([n. d.]).

[19] Zhaoyun Chen, Wei Quan, Mei Wen, Jianbin Fang, Jie Yu, Chunyuan Zhang, and Lei Luo. 2020. Deep Learning Research and Development Platform: Characterizing and Scheduling with QoS Guarantees on GPU Clusters. *IEEE Transactions on Parallel and Distributed Systems* (2020).

[20] Zhaoyun Chen, Wei Quan, Mei Wen, Jianbin Fang, Jie Yu, Chunyuan Zhang, and Lei Luo. 2020. Deep Learning Research and Development Platform: Characterizing and Scheduling with QoS Guarantees on GPU Clusters. *IEEE Transactions on Parallel and Distributed Systems* (2020).

[21] Min-Chi Chiang and Jerry Chou. 2021. DynamoML: Dynamic Resource Management Operators for Machine Learning Workloads.. In *CLOSER (CLOSER '21)*.

[22] Mosharaf Chowdhury and Ion Stoica. 2015. Efficient coflow scheduling without prior knowledge. *ACM SIGCOMM Computer Communication Review* (2015).

[23] Fernando J Corbató, Marjorie Merwin-Daggett, and Robert C Daley. 1962. An experimental time-sharing system. In *spring joint computer conference*.

[24] Daniel Crankshaw, Gur-Eyal Sela, Xiangxi Mo, Corey Zumar, Ion Stoica, Joseph Gonzalez, and Alexey Tumanov. 2020. InferLine: Latency-Aware Provisioning and Scaling for Prediction Serving Pipelines. In *Proceedings of the 11th ACM Symposium on Cloud Computing (SoCC '20)*.

[25] Daniel Crankshaw, Xin Wang, Guilio Zhou, Michael J. Franklin, Joseph E. Gonzalez, and Ion Stoica. 2017. Clipper: A Low-Latency Online Prediction Serving System. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*.

[26] Weihao Cui, Quan Chen, Han Zhao, Mengze Wei, Xiaoxin Tang, and Minyi Guo. 2021. E2bird: Enhanced Elastic Batch for Improving Responsiveness and Throughput of Deep Learning Services. *IEEE Transactions on Parallel and Distributed Systems* (2021).

[27] Weihao Cui, Mengze Wei, Quan Chen, Xiaoxin Tang, Jingwen Leng, Li Li, and Mingyi Guo. 2019. Ebird: Elastic Batch for Improving Responsiveness and Throughput of Deep Learning Services. In *2019 IEEE 37th International Conference on Computer Design (ICCD)*.

[28] Weihao Cui, Han Zhao, Quan Chen, Ningxin Zheng, Jingwen Leng, Jieru Zhao, Zhuo Song, Tao Ma, Yong Yang, Chao Li, and Minyi Guo. 2021. Enable simultaneous DNN services based on deterministic operator overlap and precise latency prediction. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '21)*.

[29] Weihao Cui, Han Zhao, Quan Chen, Ningxin Zheng, Jingwen Leng, Jieru Zhao, Zhuo Song, Tao Ma, Yong Yang, Chao Li, and Minyi Guo. 2021. Enable Simultaneous DNN Services Based on Deterministic Operator Overlap and

Precise Latency Prediction. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '21)*.

[30] Abdul Dakkak, Cheng Li, Simon Garcia de Gonzalo, Jinjun Xiong, and Wen-mei Hwu. 2019. TrIMS: Transparent and Isolated Model Sharing for Low Latency Deep Learning Inference in Function-as-a-Service. In *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*.

[31] Aditya Dhakal, Sameer G Kulkarni, and K. K. Ramakrishnan. 2020. GSLICE: controlled spatial sharing of GPUs for a scalable inference platform. In *Proceedings of the 11th ACM Symposium on Cloud Computing (SoCC '20)*.

[32] Yaoyao Ding, Ligeng Zhu, Zhihao Jia, Gennady Pekhimenko, and Song Han. 2021. IOS: Inter-Operator Scheduler for CNN Acceleration. In *Proceedings of Machine Learning and Systems (MLSys '21)*.

[33] Lisa Dunlap, Kirthevasan Kandasamy, Ujval Misra, Richard Liaw, Michael Jordan, Ion Stoica, and Joseph E. Gonzalez. 2021. Elastic Hyperparameter Tuning on the Cloud. In *Proceedings of the ACM Symposium on Cloud Computing (SoCC '21)*.

[34] Dick HJ Epema. 1995. An analysis of decay-usage scheduling in multiprocessors. *ACM SIGMETRICS Performance Evaluation Review* (1995).

[35] Dror G. Feitelson. 1996. Packing schemes for gang scheduling. In *Job Scheduling Strategies for Parallel Processing*.

[36] Dror G Feitelson. 1997. Job scheduling in multiprogrammed parallel systems. *IBM Research Report* (1997).

[37] Dror G Feitelson and Larry Rudolph. 1995. Parallel job scheduling: Issues and approaches. In *Workshop on Job Scheduling Strategies for Parallel Processing*.

[38] Dror G Feitelson, Larry Rudolph, Uwe Schwiegelshohn, Kenneth C Sevcik, and Parkson Wong. 1997. Theory and practice in parallel job scheduling. In *Workshop on Job Scheduling Strategies for Parallel Processing*.

[39] Federica Filippini, Danilo Ardagna, Marco Lattuada, Edoardo Amaldi, Maciek Riedl, Katarzyna Materka, Paweł Skrzypek, Michele Ciavotta, Fabrizio Magugliani, and Marco Cicala. 2021. ANDREAS: Artificial intelligence traiNing scheDuler foR accElerAted resource clusterS. In *2021 8th International Conference on Future Internet of Things and Cloud (FiCloud '21)*.

[40] Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*.

[41] Pin Gao, Lingfan Yu, Yongwei Wu, and Jinyang Li. 2018. Low Latency RNN Inference with Cellular Batching. In *Proceedings of the Thirteenth EuroSys Conference (EuroSys '18)*.

[42] Wei Gao, Zhisheng Ye, Peng Sun, Yonggang Wen, and Tianwei Zhang. 2021. Chronus: A Novel Deadline-aware Scheduler for Deep Learning Training Jobs. In *Proceedings of the ACM Symposium on Cloud Computing (SoCC '21)*.

[43] Ali Ghodsi, Matei Zaharia, Benjamin Hindman, Andy Konwinski, Scott Shenker, and Ion Stoica. 2011. Dominant Resource Fairness: Fair Allocation of Multiple Resource Types. In *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation (NSDI '11)*.

[44] Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W. Mahoney, and Kurt Keutzer. 2021. A Survey of Quantization Methods for Efficient Neural Network Inference. *CoRR* (2021).

[45] Guin R. Gilman, Samuel S. Ogden, Robert J. Walls, and Tian Guo. 2019. Challenges and Opportunities of DNN Model Execution Caching. In *Proceedings of the Workshop on Distributed Infrastructures for Deep Learning (DIDL '19)*.

[46] 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)*.

[47] Rong Gu, Yuquan Chen, Shuai Liu, Haipeng Dai, Guihai Chen, Kai Zhang, Yang Che, and Yihua Huang. 2021. Liquid: Intelligent Resource Estimation and Network-Efficient Scheduling for Deep Learning Jobs on Distributed GPU Clusters. *IEEE Transactions on Parallel and Distributed Systems* (2021).

[48] Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. 2017. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE, 3389–3396.

[49] Joao Guerreiro, Aleksandar Ilic, Nuno Roma, and Pedro Tomas. 2018. GPGPU Power Modeling for Multi-domain Voltage-Frequency Scaling. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA '18)*.

[50] Arpan Gujarati, Sameh Elnikety, Yuxiong He, Kathryn S. McKinley, and Björn B. Brandenburg. 2017. Swayam: distributed autoscaling to meet SLAs of machine learning inference services with resource efficiency. In *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference (Middleware '17)*.

[51] Arpan Gujarati, Reza Karimi, Safya Alzayat, Wei Hao, Antoine Kaufmann, Ymir Vigfusson, and Jonathan Mace. 2020. Serving DNNs like Clockwork: Performance Predictability from the Bottom Up. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*.

[52] Jashwant Raj Gunasekaran, Cyan Subhra Mishra, Prashanth Thinakaran, Mahmut Taylan Kandemir, and Chita R. Das. 2021. Cocktail: Leveraging Ensemble Learning for Optimized Model Serving in Public Cloud. *CoRR* (2021).

[53] Jashwant Raj Gunasekaran, Prashanth Thinakaran, Nachiappan C. Nachiappan, Mahmut Taylan Kandemir, and Chita R. Das. 2020. Fifer: Tackling Resource Underutilization in the Serverless Era. In *Proceedings of the 21st International Middleware Conference (Middleware '20)*.

[54] Udit Gupta, Samuel Hsia, Vikram Saraph, Xiaodong Wang, Brandon Reagen, Gu-Yeon Wei, Hsien-Hsin S. Lee, David Brooks, and Carole-Jean Wu. 2020. DeepRecSys: A System for Optimizing End-To-End At-Scale Neural Recommendation Inference. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*.

[55] Matthew Halpern, Behzad Boroujerdian, Todd Mummert, Evelyn Duesterwald, and Vijay Janapa Reddi. 2019. One Size Does Not Fit All: Quantifying and Exposing the Accuracy-Latency Trade-Off in Machine Learning Cloud Service APIs via Tolerance Tiers. In *2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*.

[56] Jingoo Han, M Mustafa Rafique, Luna Xu, Ali R Butt, Seung-Hwan Lim, and Sudharshan S Vazhkudai. 2020. Marble: A multi-gpu aware job scheduler for deep learning on hpc systems. In *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID '20)*.

[57] Zhenhua Han, Haisheng Tan, Shaofeng H-C Jiang, Xiaoming Fu, Wanli Cao, and Francis CM Lau. 2020. Scheduling Placement-Sensitive BSP Jobs with Inaccurate Execution Time Estimation. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications (INFOCOM '20)*.

[58] Kim Hazelwood, Sarah Bird, David Brooks, Soumith Chintala, Utku Diril, Dmytro Dzhulgakov, Mohamed Fawzy, Bill Jia, Yangqing Jia, Aditya Kalro, et al. 2018. Applied machine learning at facebook: A datacenter infrastructure perspective. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA '18)*.

[59] Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D. Joseph, Randy Katz, Scott Shenker, and Ion Stoica. 2011. Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center. In *8th USENIX Symposium on Networked Systems Design and Implementation (NSDI '11)*.

[60] Connor Holmes, Daniel Mawhirter, Yuxiong He, Feng Yan, and Bo Wu. 2019. GRNN: Low-Latency and Scalable RNN Inference on GPUs. In *Proceedings of the Fourteenth EuroSys Conference 2019 (EuroSys '19)*.

[61] Cheol-Ho Hong, Ivor Spence, and Dimitrios S. Nikolopoulos. 2017. GPU Virtualization and Scheduling Methods: A Comprehensive Survey. *Comput. Surveys* (2017).

[62] 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)*.

[63] Gadi Hutt, Vibhav Viswanathan, and Adam Nadolski. 2019. Deliver high performance ML inference with AWS Inferentia.

[64] 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)*.

[65] Ranggi Hwang, Taehun Kim, Youngeun Kwon, and Minsoo Rhu. 2020. Centaur: a chiplet-based, hybrid sparse-dense accelerator for personalized recommendations. In *Proceedings of the ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA '20)*.

[66] Vatche Ishakian, Vinod Muthusamy, and Aleksander Slominski. 2018. Serving Deep Learning Models in a Serverless Platform. In *2018 IEEE International Conference on Cloud Engineering (IC2E)*.

[67] Arezoo Jahani, Marco Lattuada, Michele Ciavotta, Danilo Ardagna, Edoardo Amaldi, and Li Zhang. 2019. Optimizing on-demand gpus in the cloud for deep learning applications training. In *2019 4th International Conference on Computing, Communications and Security (ICCCS '18)*.

[68] Paras Jain, Xiangxi Mo, Ajay Jain, Harikaran Subbaraj, Rehan Sohail Durrani, Alexey Tumanov, Joseph Gonzalez, and Ion Stoica. 2018. Dynamic space-time scheduling for gpu inference. *arXiv preprint arXiv:1901.00041* (2018).

[69] Jananie Jarachanthan, Li Chen, Fei Xu, and Bo Li. 2021. AMPS-Inf: Automatic Model Partitioning for Serverless Inference with Cost Efficiency. In *50th International Conference on Parallel Processing (ICPP 2021)*.

[70] K. R. Jayaram, Vinod Muthusamy, Parijat Dube, Vatche Ishakian, Chen Wang, Benjamin Herta, Scott Boag, Diana Arroyo, Asser Tantawi, Archit Verma, Falk Pollok, and Rania Khalaf. 2019. FfDL: A Flexible Multi-tenant Deep Learning Platform. In *Proceedings of the 20th International Middleware Conference (Middleware '19)*.

[71] 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)*.

[72] Wenqi Jiang, Zhenhao He, Shuai Zhang, Thomas B. Preuß er, Kai Zeng, Liang Feng, Jiansong Zhang, Tongxuan Liu, Yong Li, Jingren Zhou, Ce Zhang, and Gustavo Alonso. 2021. MicroRec: Efficient Recommendation Inference by Hardware and Data Structure Solutions. In *Proceedings of Machine Learning and Systems*.

[73] Wenqi Jiang, Zhenhao He, Shuai Zhang, Kai Zeng, Liang Feng, Jiansong Zhang, Tongxuan Liu, Yong Li, Jingren Zhou, Ce Zhang, and Gustavo Alonso. 2021. FleetRec: Large-Scale Recommendation Inference on Hybrid GPU-FPGA

Clusters. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining (KDD '21)*.

[74] Yimin Jiang, Yibo Zhu, Chang Lan, Bairen Yi, Yong Cui, and Chuanxiong Guo. 2020. A Unified Architecture for Accelerating Distributed DNN Training in Heterogeneous GPU/CPU Clusters. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI '20)*.

[75] Daniel Kang, Ankit Mathur, Teja Veeramacheneni, Peter Bailis, and Matei Zaharia. 2020. Jointly Optimizing Preprocessing and Inference for DNN-Based Visual Analytics. *Proc. VLDB Endow.* (2020).

[76] Sejin Kim and Yoonhee Kim. 2020. Co-scheML: Interference-aware Container Co-scheduling Scheme Using Machine Learning Application Profiles for GPU Clusters. In *2020 IEEE International Conference on Cluster Computing (CLUSTER '20)*.

[77] Jack Kosaian, K. V. Rashmi, and Shivaram Venkataraman. 2019. Parity Models: Erasure-Coded Resilience for Prediction Serving Systems. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles (SOSP '19)*.

[78] Jack Kosaian, K. V. Rashmi, and Shivaram Venkataraman. 2019. Parity Models: Erasure-Coded Resilience for Prediction Serving Systems *(SOSP '19)*.

[79] Adarsh Kumar, Kausik Subramanian, Shivaram Venkataraman, and Aditya Akella. 2021. Doing more by doing less: how structured partial backpropagation improves deep learning clusters. In *Proceedings of the 2nd ACM International Workshop on Distributed Machine Learning*.

[80] Tan N. Le, Xiao Sun, Mosharaf Chowdhury, and Zhenhua Liu. 2020. AlloX: Compute Allocation in Hybrid Clusters. In *Proceedings of the Fifteenth European Conference on Computer Systems (EuroSys '20)*.

[81] Mathias Lécuyer, Riley Spahn, Kiran Vodrahalli, Roxana Geambasu, and Daniel Hsu. 2019. Privacy Accounting and Quality Control in the Sage Differentially Private ML Platform. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles (SOSP '19)*.

[82] Yunseong Lee, Alberto Scolari, Byung-Gon Chun, Marco Domenico Santambrogio, Markus Weimer, and Matteo Interlandi. 2018. PRETZEL: Opening the Black Box of Machine Learning Prediction Serving Systems. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI '18)*.

[83] Matthew LeMay, Shijian Li, and Tian Guo. 2020. PERSEUS: Characterizing Performance and Cost of Multi-Tenant Serving for CNN Models. In *2020 IEEE International Conference on Cloud Engineering (IC2E)*.

[84] Hongliang Li, Ting Sun, Xiang Li, and Haixiao Xu. 2020. Job Placement Strategy with Opportunistic Resource Sharing for Distributed Deep Learning Clusters. In *2020 IEEE 22nd International Conference on High Performance Computing and Communications (HPCC '20)*.

[85] Jiamin Li, Hong Xu, Yibo Zhu, Zherui Liu, Chuanxiong Guo, and Cong Wang. 2022. Aryl: An Elastic Cluster Scheduler for Deep Learning. *CoRR* (2022).

[86] KeDi Li and Ning Gui. 2020. CMS: A continuous machine-learning and serving platform for industrial big data. *Future Internet* (2020).

[87] Tian Li, Jie Zhong, Ji Liu, Wentao Wu, and Ce Zhang. 2018. Ease.Ml: Towards Multi-Tenant Resource Sharing for Machine Learning Workloads. *Proc. VLDB Endow.* (2018).

[88] Wenxin Li, Sheng Chen, Keqiu Li, Heng Qi, Renhai Xu, and Song Zhang. 2020. Efficient Online Scheduling for Coflow-aware Machine Learning Clusters. *IEEE Transactions on Cloud Computing* (2020).

[89] Yang Li, Zhenhua Han, Quanlu Zhang, Zhenhua Li, and Haisheng Tan. 2020. Automating Cloud Deployment for Deep Learning Inference of Real-time Online Services. In *IEEE Conference on Computer Communications (INFOCOM '20)*.

[90] Tailin Liang, John Glossner, Lei Wang, Shaobo Shi, and Xiaotong Zhang. 2021. Pruning and quantization for deep neural network acceleration: A survey. *Neurocomputing* (2021).

[91] Richard Liaw, Romil Bhardwaj, Lisa Dunlap, Yitian Zou, Joseph E. Gonzalez, Ion Stoica, and Alexey Tumanov. 2019. HyperSched: Dynamic Resource Reallocation for Model Development on a Deadline. In *Proceedings of the ACM Symposium on Cloud Computing (SoCC '19)*.

[92] Chan-Yi Lin, Ting-An Yeh, and Jerry Chou. 2019. DRAGON: A Dynamic Scheduling and Scaling Controller for Managing Distributed Deep Learning Jobs in Kubernetes Cluster. In *CLOSER (CLOSER '19)*.

[93] Hao Liu, Qian Gao, Jiang Li, Xiaochao Liao, Hao Xiong, Guangxing Chen, Wenlin Wang, Guobao Yang, Zhiwei Zha, Daxiang Dong, Dejing Dou, and Haoyi Xiong. 2021. JIZHI: A Fast and Cost-Effective Model-As-A-Service System for Web-Scale Online Inference at Baidu. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining (KDD '21)*.

[94] Rui Liu, Sanjay Krishnan, Aaron J. Elmore, and Michael J. Franklin. 2021. Understanding and optimizing packed neural network training for hyperparameter tuning. In *Proceedings of the Fifth Workshop on Data Management for End-To-End Machine Learning (DEEM '21)*.

[95] Yunteng Luan, Xukun Chen, Hanyu Zhao, Zhi Yang, and Yafei Dai. 2019. SCHED^2: Scheduling Deep Learning Training via Deep Reinforcement Learning. In *2019 IEEE Global Communications Conference (GLOBECOM '19)*.

[96] Tao Luo, Mingen Pan, Pierre Tholoniat, Asaf Cidon, Roxana Geambasu, and Mathias Lécuyer. 2021. Privacy Budget Scheduling. In *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI '21)*.

[97] 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)*.

[98] Ruben Mayer and Hans-Arno Jacobsen. 2021. Scalable Deep Learning on Distributed Infrastructures: Challenges, Techniques, and Tools. *Comput. Surveys* (2021).

[99] Xinxin Mei, Qiang Wang, Xiaowen Chu, Hai Liu, Yiu-Wing Leung, and Zongpeng Li. 2021. Energy-aware Task Scheduling with Deadline Constraint in DVFS-enabled Heterogeneous Clusters. *CoRR* (2021).

[100] Daniel Mendoza, Francisco Romero, Qian Li, Neeraja J. Yadwadkar, and Christos Kozyrakis. 2021. Interference-Aware Scheduling for Inference Serving. In *Proceedings of the 1st Workshop on Machine Learning and Systems (EuroMLSys '21)*.

[101] Ujval Misra, Richard Liaw, Lisa Dunlap, Romil Bhardwaj, Kirthevasan Kandasamy, Joseph E. Gonzalez, Ion Stoica, and Alexey Tumanov. 2021. RubberBand: Cloud-Based Hyperparameter Tuning. In *Proceedings of the Sixteenth European Conference on Computer Systems (EuroSys '21)*.

[102] Jayashree Mohan, Amar Phanishayee, Janardhan Kulkarni, and Vijay Chidambaram. 2022. Synergy: Looking Beyond GPUs for DNN Scheduling on Multi-Tenant Clusters. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI '22)*.

[103] A.W. Mu'alem and D.G. Feitelson. 2001. Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling. *IEEE Transactions on Parallel and Distributed Systems* (2001). https://doi.org/10.1109/71.932708

[104] Deepak Narayanan, Fiodar Kazhamiaka, Firas Abuzaid, Peter Kraft, Akshay Agrawal, Srikanth Kandula, Stephen Boyd, and Matei Zaharia. 2021. Solving Large-Scale Granular Resource Allocation Problems Efficiently with POP. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles (SOSP '21)*.

[105] Deepak Narayanan, Keshav Santhanam, Fiodar Kazhamiaka, Amar Phanishayee, and Matei Zaharia. 2020. Analysis and exploitation of dynamic pricing in the public cloud for ml training. In *VLDB DISPA Workshop 2020*.

[106] Deepak Narayanan, Keshav Santhanam, Fiodar Kazhamiaka, Amar Phanishayee, and Matei Zaharia. 2020. Heterogeneity-Aware Cluster Scheduling Policies for Deep Learning Workloads. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI '20)*.

[107] Deepak Narayanan, Keshav Santhanam, Amar Phanishayee, and Matei Zaharia. 2018. Accelerating deep learning workloads through efficient multi-model execution. In *NeurIPS Workshop on Systems for Machine Learning*.

[108] Deepak Narayanan, Keshav Santhanam, Amar Phanishayee, and Matei Zaharia. 2018. Accelerating deep learning workloads through efficient multi-model execution. In *NeurIPS Workshop on Systems for Machine Learning*.

[109] Marco A. S. Netto, Rodrigo N. Calheiros, Eduardo R. Rodrigues, Renato L. F. Cunha, and Rajkumar Buyya. 2018. HPC Cloud for Scientific and Business Applications: Taxonomy, Vision, and Research Challenges. *Comput. Surveys* (2018).

[110] Samuel S. Ogden, Xiangnan Kong, and Tian Guo. 2021. PieSlicer: Dynamically Improving Response Time for Cloud-based CNN Inference. In *Proceedings of the ACM/SPEC International Conference on Performance Engineering (ICPE '21)*.

[111] Christopher Olston, Noah Fiedel, Kiril Gorovoy, Jeremiah Harmsen, Li Lao, Fangwei Li, Vinu Rajashekhar, Sukriti Ramesh, and Jordan Soyke. 2017. TensorFlow-Serving: Flexible, High-Performance ML Serving. *CoRR* abs/1712.06139 (2017).

[112] Shuo Ouyang, Dezun Dong, Yemao Xu, and Liquan Xiao. 2021. Communication optimization strategies for distributed deep neural network training: A survey. *J. Parallel and Distrib. Comput.* (2021).

[113] Jongsoo Park, Maxim Naumov, Protonu Basu, Summer Deng, Aravind Kalaiah, Daya Khudia, James Law, Parth Malani, Andrey Malevich, Satish Nadathur, Juan Pino, Martin Schatz, Alexander Sidorov, Viswanath Sivakumar, Andrew Tulloch, Xiaodong Wang, Yiming Wu, Hector Yuen, Utku Diril, Dmytro Dzhulgakov, Kim Hazelwood, Bill Jia, Yangqing Jia, Lin Qiao, Vijay Rao, Nadav Rotem, Sungjoo Yoo, and Mikhail Smelyanskiy. 2018. Deep Learning Inference in Facebook Data Centers: Characterization, Performance Optimizations and Hardware Implications. *CoRR* (2018).

[114] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems*.

[115] 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)*.

[116] Yanghua Peng, Yixin Bao, Yangrui Chen, Chuan Wu, Chen Meng, and Wei Lin. 2021. DL2: A Deep Learning-Driven Scheduler for Deep Learning Clusters. *IEEE Transactions on Parallel and Distributed Systems* (2021).

[117] 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).

[118] Heyang Qin, Syed Zawad, Yanqi Zhou, Lei Yang, Dongfang Zhao, and Feng Yan. 2019. Swift machine learning model serving scheduling: a region based reinforcement learning approach. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '19).*

[119] Jeff Rasley, Yuxiong He, Feng Yan, Olatunji Ruwase, and Rodrigo Fonseca. 2017. HyperDrive: Exploring Hyperparameters with POP Scheduling. In *Proceedings of the 18th International Middleware Conference (Middleware '17).*

[120] Albert Reuther, Chansup Byun, William Arcand, David Bestor, Bill Bergeron, Matthew Hubbell, Michael Jones, Peter Michaleas, Andrew Prout, Antonio Rosa, and Jeremy Kepner. 2018. Scalable system scheduling for HPC and big data. *J. Parallel and Distrib. Comput.* (2018).

[121] Francisco Romero, Qian Li, Neeraja J. Yadwadkar, and Christos Kozyrakis. 2021. INFaaS: Automated Model-less Inference Serving. In *2021 USENIX Annual Technical Conference (USENIX ATC '21).*

[122] Vaibhav Saxena, K. R. Jayaram, Saurav Basu, Yogish Sabharwal, and Ashish Verma. 2020. Effective Elastic Scaling of Deep Learning Workloads. In *28th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS '20).*

[123] Wonik Seo, Sanghoon Cha, Yeonjae Kim, Jaehyuk Huh, and Jongse Park. 2021. SLO-Aware Inference Scheduler for Heterogeneous Processors in Edge Platforms. *ACM Trans. Archit. Code Optim.* (2021).

[124] Alexander Sergeev and Mike Del Balso. 2018. Horovod: fast and easy distributed deep learning in TensorFlow. *CoRR* (2018).

[125] Dinggang Shen, Guorong Wu, and Heung-Il Suk. 2017. Deep learning in medical image analysis. *Annual review of biomedical engineering* 19 (2017), 221–248.

[126] Haichen Shen, Lequn Chen, Yuchen Jin, Liangyu Zhao, Bingyu Kong, Matthai Philipose, Arvind Krishnamurthy, and Ravi Sundaram. 2019. Nexus: A GPU Cluster Engine for Accelerating DNN-Based Video Analysis. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles (SOSP '19).*

[127] Lin Shi, Hao Chen, Jianhua Sun, and Kenli Li. 2012. vCUDA: GPU-Accelerated High-Performance Computing in Virtual Machines. *IEEE Trans. Comput.* (2012).

[128] S R Shishira, A. Kandasamy, and K. Chandrasekaran. 2017. Workload scheduling in cloud: A comprehensive survey and future research directions. In *International Conference on Cloud Computing, Data Science Engineering - Confluence.*

[129] Dharma Shukla, Muthian Sivathanu, Srinidhi Viswanatha, Bhargav Gulavani, Rimma Nehme, Amey Agrawal, Chen Chen, Nipun Kwatra, Ramachandran Ramjee, Pankaj Sharma, et al. 2022. Singularity: Planet-Scale, Preemptible, Elastic Scheduling of AI Workloads. *CoRR* (2022).

[130] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. 2017. Mastering the game of go without human knowledge. *nature* 550, 7676 (2017), 354–359.

[131] Jaewon Son, Yonghyuk Yoo, Khu-rai Kim, Youngjae Kim, Kwonyong Lee, and Sungyong Park. 2021. A GPU Scheduling Framework to Accelerate Hyper-Parameter Optimization in Deep Learning Clusters. *Electronics* (2021).

[132] Abeda Sultana, Li Chen, Fei Xu, and Xu Yuan. 2020. E-LAS: Design and Analysis of Completion-Time Agnostic Scheduling for Distributed Deep Learning Cluster. In *49th International Conference on Parallel Processing (ICPP '20).*

[133] Peng Sun, Yonggang Wen, Nguyen Binh Duong Ta, and Shengen Yan. 2017. Towards distributed machine learning in shared clusters: A dynamically-partitioned approach. In *2017 IEEE International Conference on Smart Computing (SMARTCOMP '17).*

[134] Nguyen Binh Duong Ta. 2019. FC2: cloud-based cluster provisioning for distributed machine learning. *Cluster Computing* (2019).

[135] Cheng Tan, Zhichao Li, Jian Zhang, Yu Cao, Sikai Qi, Zherui Liu, Yibo Zhu, and Chuanxiong Guo. 2021. Serving DNN Models with Multi-Instance GPUs: A Case of the Reconfigurable Machine Scheduling Problem. *CoRR* (2021).

[136] Xuehai Tang, Peng Wang, Qiuyang Liu, Wang Wang, and Jizhong Han. 2019. Nanily: A QoS-Aware Scheduling for DNN Inference Workload in Clouds. In *2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS).*

[137] Prashanth Thinakaran, Jashwant Raj Gunasekaran, Bikash Sharma, Mahmut Taylan Kandemir, and Chita R. Das. 2019. Kube-Knots: Resource Harvesting through Dynamic Container Orchestration in GPU-based Datacenters. In *2019 IEEE International Conference on Cluster Computing (CLUSTER).*

[138] Vinod Kumar Vavilapalli, Arun C. Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, Robert Evans, Thomas Graves, Jason Lowe, Hitesh Shah, Siddharth Seth, Bikas Saha, Carlo Curino, Owen O'Malley, Sanjay Radia, Benjamin Reed, and Eric Baldeschwieler. 2013. Apache Hadoop YARN: Yet Another Resource Negotiator. In *Proceedings of the 4th Annual Symposium on Cloud Computing (SoCC '13).*

[139] Joost Verbraeken, Matthijs Wolting, Jonathan Katzy, Jeroen Kloppenburg, Tim Verbelen, and Jan S. Rellermeyer. 2020. A Survey on Distributed Machine Learning. *Comput. Surveys* (2020).

[140] Farui Wang, Weizhe Zhang, Shichao Lai, Meng Hao, and Zheng Wang. 2021. Dynamic GPU Energy Optimization for Machine Learning Training Workloads. *IEEE Transactions on Parallel and Distributed Systems* (2021).

[141] Haoyu Wang, Zetian Liu, and Haiying Shen. 2020. Job scheduling for large-scale machine learning clusters. In *Proceedings of the 16th International Conference on emerging Networking EXperiments and Technologies (CoNEXT '20)*.

[142] Luping Wang, Lingyun Yang, Yinghao Yu, Wei Wang, Bo Li, Xianchao Sun, Jian He, and Liping Zhang. 2021. Morphling: Fast, Near-Optimal Auto-Configuration for Cloud-Native Model Serving. In *Proceedings of the ACM Symposium on Cloud Computing (SoCC '21)*.

[143] Luping Wang, Lingyun Yang, Yinghao Yu, Wei Wang, Bo Li, Xianchao Sun, Jian He, and Liping Zhang. 2021. Morphling: Fast, Near-Optimal Auto-Configuration for Cloud-Native Model Serving. In *Proceedings of the ACM Symposium on Cloud Computing (SoCC '21)*.

[144] Mengdi Wang, Chen Meng, Guoping Long, Chuan Wu, Jun Yang, Wei Lin, and Yangqing Jia. 2019. Characterizing Deep Learning Training Workloads on Alibaba-PAI. In *Proceedings of the 2019 IEEE International Symposium on Workload Characterization (IISWC '19)*.

[145] Qiang Wang and Xiaowen Chu. 2020. GPGPU Performance Estimation With Core and Memory Frequency Scaling. *IEEE Transactions on Parallel and Distributed Systems* (2020).

[146] Qiang Wang, Shaohuai Shi, Canhui Wang, and Xiaowen Chu. 2020. Communication Contention Aware Scheduling of Multiple Deep Learning Training Jobs. *CoRR* (2020).

[147] Shaoqi Wang, Oscar J Gonzalez, Xiaobo Zhou, Thomas Williams, Brian D Friedman, Martin Havemann, and Thomas Woo. 2020. An Efficient and Non-Intrusive GPU Scheduling Framework for Deep Learning Training Systems. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC '20)*.

[148] Wei Wang, Jinyang Gao, Meihui Zhang, Sheng Wang, Gang Chen, Teck Khim Ng, Beng Chin Ooi, Jie Shao, and Moaz Reyad. 2018. Rafiki: Machine Learning as an Analytics Service System. *Proc. VLDB Endow.* (2018).

[149] 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)*.

[150] Xiaorui Wu, Hong Xu, and Yi Wang. 2020. Irina: Accelerating DNN Inference with Efficient Online Scheduling. In *4th Asia-Pacific Workshop on Networking (APNet '20)*.

[151] Yidi Wu, Kaihao Ma, Xiao Yan, Zhi Liu, Zhenkun Cai, Yuzhen Huang, James Cheng, Han Yuan, and Fan Yu. 2022. Elastic Deep Learning in Multi-Tenant GPU Clusters. *IEEE Transactions on Parallel and Distributed Systems* (2022).

[152] 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*.

[153] Wencong Xiao, Shiru Ren, Yong Li, Yang Zhang, Pengyang Hou, Zhi Li, Yihui Feng, Wei Lin, and Yangqing Jia. 2020. AntMan: Dynamic Scaling on GPU Clusters for Deep Learning. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI '20)*.

[154] Lei Xie, Jidong Zhai, Baodong Wu, Yuanbo Wang, Xingcheng Zhang, Peng Sun, and Shengen Yan. 2020. Elan: Towards Generic and Efficient Elastic Training for Deep Learning. In *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS '20)*.

[155] Neeraja J. Yadwadkar, Francisco Romero, Qian Li, and Christos Kozyrakis. 2019. A Case for Managed and Model-less Inference Serving. In *Proceedings of the Workshop on Hot Topics in Operating Systems (HotOS '19)*.

[156] Feng Yan, Olatunji Ruwase, Yuxiong He, and Evgenia Smirni. 2016. SERF: Efficient Scheduling for Fast Deep Neural Network Serving via Judicious Parallelism. In *SC '16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*.

[157] Zhisheng Ye, Peng Sun, Wei Gao, Tianwei Zhang, Xiaolin Wang, Shengen Yan, and Yingwei Luo. 2021. ASTRAEA: A Fair Deep Learning Scheduler for Multi-tenant GPU Clusters. *IEEE Transactions on Parallel and Distributed Systems* (2021).

[158] Ting-An Yeh, Hung-Hsin Chen, and Jerry Chou. 2020. KubeShare: A Framework to Manage GPUs as First-Class and Shared Resources in Container Cloud. In *Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing (HPDC '20)*.

[159] Gingfung Yeung, Damian Borowiec, Adrian Friday, Richard Harper, and Peter Garraghan. 2020. Towards GPU Utilization Prediction for Cloud Deep Learning. In *USENIX Workshop on Hot Topics in Cloud Computing*.

[160] Gingfung Yeung, Damian Borowiec, Renyu Yang, Adrian Friday, Richard Harper, and Peter Garraghan. 2022. Horus: Interference-Aware and Prediction-Based Scheduling in Deep Learning Systems. *IEEE Transactions on Parallel and Distributed Systems* (2022).

[161] Xiaodong Yi, Shiwei Zhang, Ziyue Luo, Guoping Long, Lansong Diao, Chuan Wu, Zhen Zheng, Jun Yang, and Wei Lin. 2020. Optimizing distributed training deployment in heterogeneous GPU clusters. In *Proceedings of the 16th International Conference on emerging Networking EXperiments and Technologies (CoNext '20)*.

[162] Andy B. Yoo, Morris A. Jette, and Mark Grondona. 2003. SLURM: Simple Linux Utility for Resource Management. In *Job Scheduling Strategies for Parallel Processing*.
[163] Fuxun Yu, Di Wang, Longfei Shangguan, Minjia Zhang, Chenchen Liu, and Xiang Chen. 2022. A Survey of Multi-Tenant Deep Learning Inference on GPU. *CoRR* abs/2203.09040 (2022).
[164] Fuxun Yu, Di Wang, Longfei Shangguan, Minjia Zhang, Xulong Tang, Chenchen Liu, and Xiang Chen. 2021. A Survey of Large-Scale Deep Learning Serving System Optimization: Challenges and Opportunities. *CoRR* (2021).
[165] Minchen Yu, Zhifeng Jiang, Hok Chun Ng, Wei Wang, Ruichuan Chen, and Bo Li. 2021. Gillis: Serving Large Neural Networks in Serverless Functions with Automatic Model Partitioning. In *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*.
[166] Menglu Yu, Ye Tian, Bo Ji, Chuan Wu, Hridesh Rajan, and Jia Liu. 2022. GADGET: Online Resource Optimization for Scheduling Ring-All-Reduce Learning Jobs. *arXiv preprint arXiv:2202.01158* (2022).
[167] Menglu Yu, Chuan Wu, Bo Ji, and Jia Liu. 2021. A Sum-of-Ratios Multi-Dimensional-Knapsack Decomposition for DNN Resource Scheduling. In *IEEE Conference on Computer Communications (INFOCOM '21)*.
[168] Peifeng Yu and Mosharaf Chowdhury. 2020. Fine-Grained GPU Sharing Primitives for Deep Learning Applications. In *Proceedings of Machine Learning and Systems (MLSys '20)*.
[169] Peifeng Yu, Jiachen Liu, and Mosharaf Chowdhury. 2021. Fluid: Resource-aware Hyperparameter Tuning Engine. In *Proceedings of Machine Learning and Systems (MLSys '21)*.
[170] Matei Zaharia, Dhruba Borthakur, Joydeep Sen Sarma, Khaled Elmeleegy, Scott Shenker, and Ion Stoica. 2010. Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling. In *Proceedings of the 5th European conference on Computer systems*. 265–278.
[171] Zhi-Hui Zhan, Xiao-Fang Liu, Yue-Jiao Gong, Jun Zhang, Henry Shu-Hung Chung, and Yun Li. 2015. Cloud Computing Resource Scheduling and a Survey of Its Evolutionary Approaches. *Comput. Surveys* (2015).
[172] Chengliang Zhang, Minchen Yu, Wei Wang, and Feng Yan. 2019. MArk: Exploiting Cloud Services for Cost-Effective, SLO-Aware Machine Learning Inference Serving. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*.
[173] Chengliang Zhang, Minchen Yu, Feng Yan, et al. 2020. Enabling Cost-Effective, SLO-Aware Machine Learning Inference Serving on Public Cloud. *IEEE Transactions on Cloud Computing* (2020).
[174] Huaizheng Zhang, Yuanming Li, Qiming Ai, Yong Luo, Yonggang Wen, Yichao Jin, and Nguyen Binh Duong Ta. 2020. Hysia: Serving DNN-Based Video-to-Retail Applications in Cloud. In *Proceedings of the 28th ACM International Conference on Multimedia (MM '20)*.
[175] Hao Zhang, Zeyu Zheng, Shizhen Xu, Wei Dai, Qirong Ho, Xiaodan Liang, Zhiting Hu, Jinliang Wei, Pengtao Xie, and Eric P Xing. 2017. Poseidon: An efficient communication architecture for distributed deep learning on GPU clusters. In *2018 USENIX Annual Technical Conference (ATC '18)*.
[176] Jeff Zhang, Sameh Elnikety, Shuayb Zarar, Atul Gupta, and Siddharth Garg. 2020. Model-Switching: Dealing with Fluctuating Workloads in Machine-Learning-as-a-Service Systems. In *12th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 20)*.
[177] Jianfeng Zhang, Wensheng Zhang, Lingjun Pu, and Jingdong Xu. 2020. QoS Optimization of DNN Serving Systems Based on Per-Request Latency Characteristics. In *International Conference on Mobility, Sensing and Networking (MSN)*.
[178] Qin Zhang, Ruiting Zhou, Chuan Wu, Lei Jiao, and Zongpeng Li. 2020. Online scheduling of heterogeneous distributed machine learning jobs. In *Proceedings of the Twenty-First International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing (MOBIHOC '20)*.
[179] Shaojun Zhang, Wei Li, Chen Wang, Zahir Tari, and Albert Y. Zomaya. 2020. DyBatch: Efficient Batching and Fair Scheduling for Deep Learning Inference on Time-sharing Devices. In *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*.
[180] Han Zhao, Weihao Cui, Quan Chen, Jingwen Leng, Kai Yu, Deze Zeng, Chao Li, and Minyi Guo. 2020. CODA: Improving Resource Utilization by Slimming and Co-locating DNN and CPU Jobs. In *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS '20)*.
[181] Hanyu Zhao, Zhenhua Han, Zhi Yang, Quanlu Zhang, Fan Yang, Lidong Zhou, Mao Yang, Francis C.M. Lau, Yuqi Wang, Yifan Xiong, and Bin Wang. 2020. HiveD: Sharing a GPU Cluster for Deep Learning with Guarantees. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI '20)*.
[182] Haoyue Zheng, Fei Xu, Li Chen, Zhi Zhou, and Fangming Liu. 2019. Cynthia: Cost-Efficient Cloud Resource Provisioning for Predictable Distributed Deep Neural Network Training. In *Proceedings of the 48th International Conference on Parallel Processing (ICPP' 19)*.
[183] Pan Zhou, Xinshu He, Shouxi Luo, Hongfang Yu, and Gang Sun. 2020. JPAS: Job-progress-aware flow scheduling for deep learning clusters. *Journal of Network and Computer Applications* (2020).
[184] Ruiting Zhou, Jinlong Pang, Qin Zhang, Chuan Wu, Lei Jiao, Yi Zhong, and Zongpeng Li. 2022. Online Scheduling Algorithm for Heterogeneous Distributed Machine Learning Jobs. *IEEE Transactions on Cloud Computing* (2022).