# Deep Learning Workload Scheduling in GPU Datacenters: A Survey

ZHISHENG YE, Peking University, China
WEI GAO and QINGHAO HU, S-Lab, Nanyang Technological University, Singapore
PENG SUN, Shanghai AI Laboratory & SenseTime Research, China
XIAOLIN WANG and YINGWEI LUO, Peking University, China
TIANWEI ZHANG and YONGGANG WEN, Nanyang Technological University, Singapore

Deep learning (DL) has demonstrated its remarkable success 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 a GPU datacenter is crucially important to reduce 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, many schedulers are proposed to tailor for DL workloads in GPU datacenters. This article 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 utilization manner*. Finally, we discuss several promising future research directions including emerging DL workloads, advanced scheduling decision making, and underlying hardware resources. A more detailed summary of 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

**ACM Reference Format:**
Zhisheng Ye, Wei Gao, Qinghao Hu, Peng Sun, Xiaolin Wang, Yingwei Luo, Tianwei Zhang, and Yonggang Wen. 2024. Deep Learning Workload Scheduling in GPU Datacenters: A Survey. *ACM Comput. Surv.* 56, 6, Article 146 (January 2024), 38 pages. https://doi.org/10.1145/3638757

The research is supported under the National Key R&D Program of China under Grant No. 2022YFB4500701 and the RIE2020 Industry Alignment Fund - Industry Collaboration Projects (IAF-ICP) Funding Initiative, as well as cash and in-kind contributions from the industry partner(s). It is also supported by the National Science Foundation of China (Nos. 62032001, 62032008, 62372011).

Z. Ye, W. Gao, and Q. Hu equal contribution. Determined by rolling dice.

Authors' addresses: Z. Ye, X. Wang, and Y. Luo, Peking University, No. 5 Yiheyuan Road, Haidian District, Beijing, 100871, China; e-mails: {yezhisheng, wxl, lyw}@pku.edu.cn; W. Gao and Q. Hu, S-Lab, Nanyang Technological University, 50 Nanyang Avenue, Singapore, 639798, Singapore; e-mails: {gaow0007, qinghao.hu}@ntu.edu.sg; P. Sun, Shanghai AI Laboratory & SenseTime Research, No. 701 Yunjin Road, Xuhui District, Shanghai, 200232, China; e-mail: sunpeng1@sensetime.com; T. Zhang (Corresponding author) and Y. Wen, Nanyang Technological University, 50 Nanyang Avenue, Singapore, 639798, Singapore; e-mails: {tianwei.zhang, ygwen}@ntu.edu.sg.



ACM 0360-0300/2024/01-ART146
https://doi.org/10.1145/3638757

## 1 INTRODUCTION

Recent decades have witnessed a dramatic increase in **deep learning (DL)** research, development, and application in many fields, including Go [151], medical analysis [145], robotics [49], and so on. A standard DL development pipeline consists of *model training* and *model inference*. Each stage requires high-grade hardware resources (GPU and other computing systems) to produce and serve production-level DL models [65, 80, 123, 171]. Therefore, it has become common for IT industries [65, 171] and research institutes [18, 80] to set up *GPU datacenters* to meet their ever-growing DL development demands. A GPU datacenter possesses large amounts of heterogeneous computing resources to host large amounts of DL workloads. An effective scheduler is 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 [41]. 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 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 could cause unbalanced resource utilization and exorbitant infrastructure expense [184], 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 [24, 199]. 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 value. Therefore, it is critical to balance the inference latency, accuracy, and cost.

A variety of DL schedulers have been proposed for GPU datacenters [24, 47, 123, 134, 141, 175, 199]. However, most of these systems are designed in an ad hoc way for some specific objectives. There is still a lack of comprehensive exploration toward 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 article 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 of 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. The summary, referred to the "we summarize" in the above sentence can disclose the common and important considerations for existing DL scheduler designs. Third, we conclude the limitations and implications of 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.

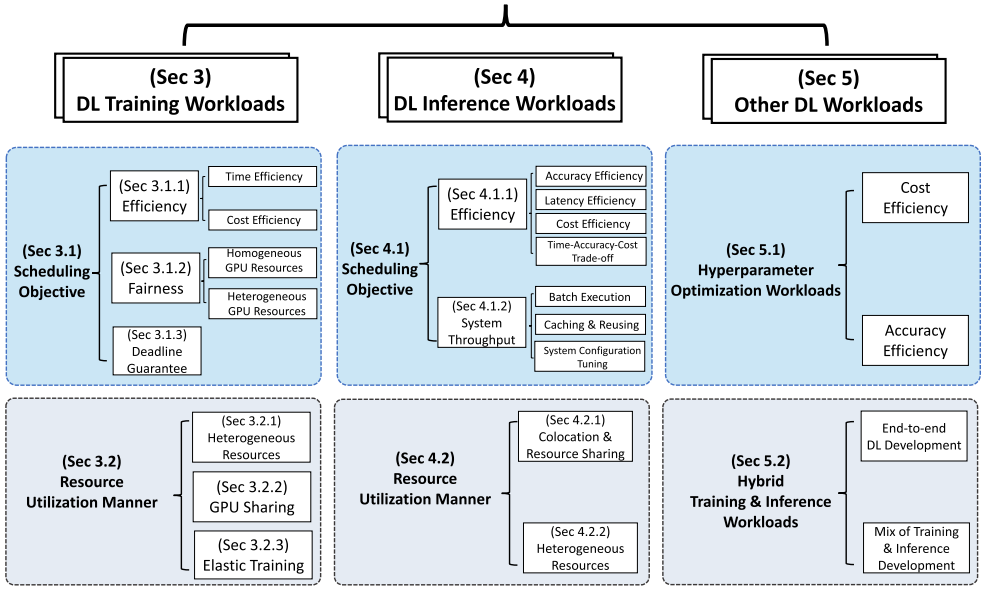


Fig. 1. The overall structure of this survey.

**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 [129, 161], efficient model inference [43, 106], and so on. 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 [149, 198] and HPC [126, 139] workloads. As discussed in Section 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 [114] 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. [191] 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 [190] considered 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.

**Article organization.** The article is organized as follows: Section 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 Figure 1. Concretely, Section 3 and Section 4 present detailed categorizations of training and inference workloads based on the scheduling objectives and resource consumption features, respectively. Section 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. Section 6 concludes this survey article and identifies the future directions of scheduler designs.

## 2 BACKGROUND

### 2.1 DL Workloads Characteristics and Challenges

A DL development pipeline typically consists of three stages: data processing, model training, and model inference. Data processing typically refers to preparing, cleaning, transforming, and organizing the raw data so it can be used as input of a DL model. This stage requires a significant amount of CPU and storage resources. In this article, we narrow down our focus to training and inference workloads that account for the most computation and consume the majority of GPU resources in a datacenter.

*2.1.1 DL Training.* A DL training workload builds models by extracting features from existing data. A DL framework (e.g., PyTorch, TensorFlow) 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 [144], `DistributedDataParallel` in Pytorch, `MultiWorkerMirroredStrategy` in Tensorflow). Note that we only discuss elastic training based on data parallelism as it is adopted by the majority of training workloads. Other paradigms, including model and pipeline parallelism [125], are mainly for large models and beyond the scope of our survey.

DL training jobs share some similar features as traditional big data or HPC jobs and also exhibit some special features. A series of studies have characterized training workloads from the production GPU datacenters, including Microsoft [80], SenseTime [65], and Alibaba [164, 171]. The characteristics and scheduling challenges are summarized below.

**T1: Inherent heterogeneity** [80, 175]. 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 [123].

**T2: Placement sensitivity** [113, 175]. Distributed DL jobs are sensitive to the locality of allocated GPU resources. Specifically, the runtime speed of some distributed DL jobs is 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 links (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.

*Challenge: DL training can benefit from newer generations of GPUs. However, the marginal benefit brought by new GPU versions varies significantly (Figure 2(f)). Besides, along with PCIe, InfiniBand, Ethernet, and QPI, distributed training has several alternatives for communication. It is non-trivial to properly allocate these resources to jobs.*

**T3: Iterative process** [41, 132]. DL training repeats a similar iterative pattern up to thousands of times, as shown in Figure 2(e). Each iteration consists of forward propagation, backward propagation, and parameter update. Profiling a small number of iterations suffices to predict the pattern of future GPU memory usage and job completion time.

**T4: Feedback-driven exploration** [175, 216]. Training a DL model is a typical trial-and-error process. Users may explore a number of trial configurations and terminate unpromising trials by

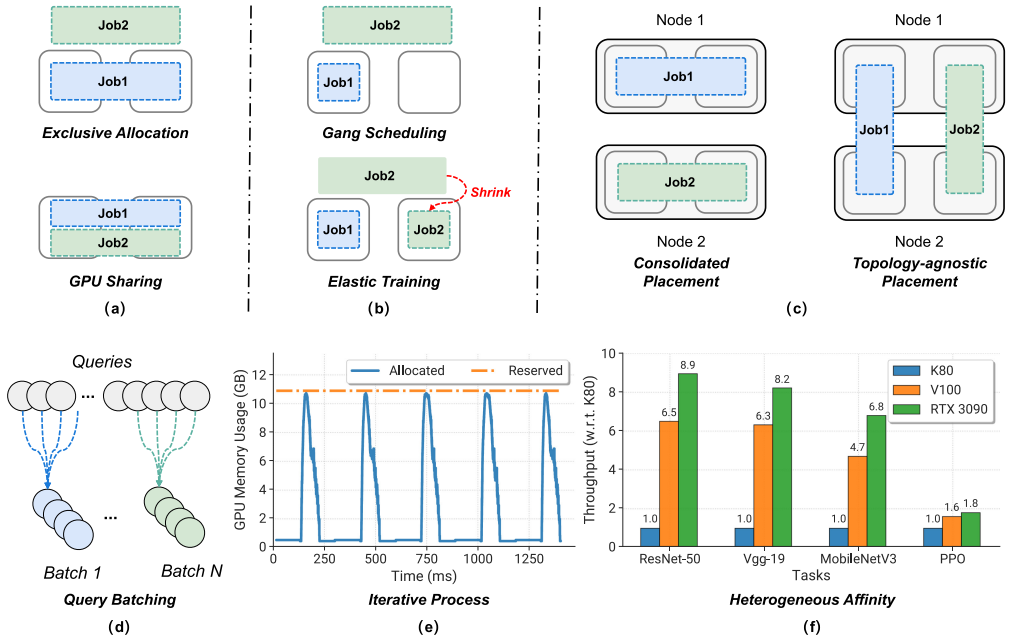


Fig. 2. Characteristics of training and inference workloads. (**a**) Exclusive Allocation versus GPU Sharing. (**b**) Gang Scheduling versus Elastic Training. (**c**) Consolidated Placement versus Topology-agnostic Placement. (**d**) Query Batching mechanism in inference. (**e**) Iterative Process: allocated and reserved GPU memory trace profiled through `torch.profiler` (ResNet-50 on ImageNet). Allocated memory is the amount of memory that is being actively used by the model, and reserved memory is the amount of memory that has been reserved for potential future use. (**f**) Heterogeneous Affinity: the magnitude of speedup across GPU generations differ across different tasks.

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** [65] **versus GPU sharing** [171]. Figure 2(a) depicts the difference between exclusive allocation and GPU. Exclusive allocation refers to a DL job exclusively having resource usage ownership. On the contrary, 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 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)**, **Multi-Process Service (MPS)**, and GPU virtualization.

**T6: Gang scheduling** [65] **versus elastic training** [134]. 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 [32]. The requirement of gang scheduling results from the native support of DL frameworks. 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 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 [65].

*Challenge: DL frameworks provide functions to pause/resume training jobs at any time for better fault tolerance. The overhead primarily depends upon the job scale, which ranges from seconds to minutes and seriously deteriorates for large models.*

*2.1.2 DL Inference.* Model inference is the process of making predictions about 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, such as TensorFlow Serving [128], MXNet Model Server [1], and so on. The inference jobs must be performed in a real-time manner, facing dynamic queries with strict latency requirements [199]. They may process each inference request individually or batch multiple requests concurrently to balance 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 [69], 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** [26, 51]. Different from offline training that 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 fewer 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 opportunities for fine-grained optimization.

*Challenge: Compared to training jobs, the inference service mainly involves the forward propagation stage and consumes small amounts of GPU resources. This often leads to low GPU utilization for inference workloads [96, 200]. Besides, as an online service, it is common for the inference application to receive bursty and fluctuating requests, which are unpredictable. Operators need to guarantee the latency with minimal operational cost even in extremely overloading scenarios.*

**I2: High demands on latency and accuracy** [24, 199]. First, the inference service is expected to respond to incoming queries promptly. Delays of inference responses can cause a bad user experience. For example, an online recommend service is required to provide recommendations at interactive latencies (<100 ms), such as traditional web-serving workloads [12] to prevent user losses [24]. Other kinds of inference services also have strong latency requirements (e.g., <200 ms [199]). Second, 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 [54]. The tight latency and accuracy demands pose great difficulty in managing inference jobs on GPUs, and there exists a tradeoff between high accuracy and low latency. The datacenter managers need to carefully balance the latency overhead and prediction performance of the inference workloads.

*Challenge: The inference jobs are relatively malleable in terms of latency, accuracy, and cost. For instance, 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].*

## 2.2 Prior Schedulers in Datacenters

Scheduling has continuously drawn public attention for several decades [31, 33, 34]. 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

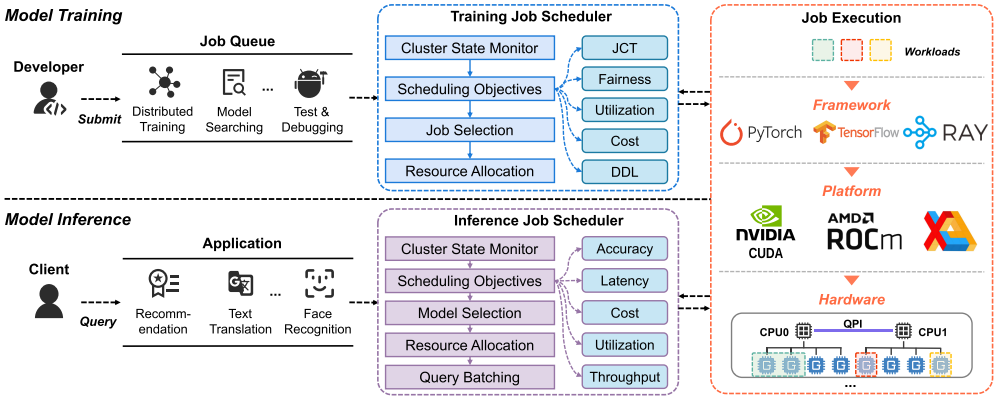


Fig. 3. Scheduling workflow for model training and inference workloads.

to competing jobs for specific scheduling objectives [33], which forms an NP-hard problem. In particular, it matches available resources with pending jobs and decides the optimal moment and 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 [189], OpenPBS [2]) are used to support HPC applications and scientific computing; cloud schedulers (e.g., Mesos [61], Kubernetes [11], Yarn [160]) 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 DL frameworks and assigns appropriate resources to satisfy a variety of DL workloads. It receives different types of workloads from the users. By monitoring the usage 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 (Sections 3 and 4) and consider the mix of them in Section 5.

Some techniques and mechanisms of conventional parallel job scheduling may also apply to DL workload 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 [35, 47, 184]. 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 such as performance modeling of workloads (e.g., online profiling [41] and performance prediction [47]) and trace analysis for characterizing the cluster-level workload distribution [65, 171] are widely adopted. Other traditional scheduling techniques (e.g., backfilling [47, 120]) and mechanisms (e.g., time-slicing [175], checkpointing [184], and migration [14, 175]) 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 (Section 2.1), simply adopting these techniques can cause a series of issues, e.g., query blocking, resource under-utilization, high operation cost. Below, we summarize the challenges of scheduler designs caused by DL workload features.

### 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 article 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 cited in References [43, 106, 126, 129, 139, 149, 161, 198].

First, we do not consider the *optimization solutions* for *individual* training or inference jobs. Training job optimization mainly contains distributed training acceleration [84] and job placement optimization [188] as well as memory optimization [74, 75, 90, 140, 213, 218]. Inference job optimization techniques include workload characterization [13], pipeline execution [85], and so on. These techniques can improve the scheduling performance, but 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 [118], HyperSched [107]). They will be summarized in Section 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 [10] proposed a novel in-network computing scheduling algorithm for datacenter switches. A number of works [50, 115, 165] 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, PIM architecture) as subsidiary. This can reflect the current status of mainstream DL infrastructures. Some scheduling systems mainly utilize the CPU [58, 59, 72, 180], FPGA [71, 82], or hybrid resources [83] where GPUs are not dominant. Some papers consider the DL services on mobile devices [127] or edge computing [143, 204] other than datacenters. Since DNN training and inference workloads are memory-intensive, new compute-capable memory devices such as **PIM (Processing-in-Memory)** [19, 38, 56, 94] are proposed to accelerate DL training and inference. 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. Our surveyed schedulers do not differentiate the DNN models (e.g., RNN, LSTMs, CNNs, MLPs). Some papers considered the optimization of specific DL applications based on their unique behaviors, including RNN-based service [39, 62], recommendation systems [53, 82, 83, 109], and video analytics [146, 201]. These works are not summarized in this article. Besides, our aim is to enhance system and workloads in terms of performance, efficiency, and user experience. Other objectives like privacy protection [93, 112] is not considered either.

## 3 SCHEDULING DL TRAINING WORKLOADS

DL training jobs consume a majority of computing 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 utilization manner. Table 1 summarizes the past works for DL training scheduling in GPU datacenters. 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.

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 [154] | ♥ | Linear Programming | Fairness Guarantee; JCT Reduction | - | - | - | S | - |
| | Topology-aware [3] | ♣ | Best-effort topology-aware Placement | Lower Interference | - | - | - | S | ✔ |
| 2018 | Gandiva [175] | ✿♣ | Time-slicing; Migration; Grow-shrink | Better GPU Utilization | - | ✔ | ✔ | L | - |
| | OASiS [7] | ✿♣ | Primal-dual framework | Better GPU Utilization | - | ✔ | - | S | - |
| | Optimus [132] | ♣ | Performance Modelling | JCT Reduction | - | ✔ | - | S | ✔ |
| 2019 | $FC^2$ [155] | ♦ | Automatic Resource Configuration | Cost Effectiveness | ✔* | ✔ | - | S | - |
| | $Sched^2$ [110] | ♣ | Q-network-based Scheduler | Reduce Cluster Fragmentation | - | - | - | L | - |
| | Cynthia [214] | ♦ | Performance Modelling | Monetary Cost Reduction | - | ✔ | - | S | - |
| | Dragon [108] | ✿♣ | GPU time-sharing; Autoscaling | Better GPU Utilization | - | ✔ | - | S | - |
| | FfDL [78] | ♠ | Lesson-motivated Design | production DL platform | - | - | - | S | ✔ |
| | Harmony [6] | ♣ | RL Scheduler; Bin Packing | JCT Reduction; Better GPU Utilization | - | - | - | S | - |
| | JPAS [216] | ♦ | Accuracy Curve Modelling | JCT Reduction | - | - | ✔ | S | - |
| | Philly [80] | ♣ | Locality-relaxity | Workload Analysis; JCT Reduction | - | - | - | - | ✔ |
| | Tiresias [47] | ♣ | Gittins index; Least-Attained Service (LAS) | Information-agnostic | - | - | - | M | ✔ |
| 2020 | $Gandiva_{fair}$ [14] | ♥♣ | Gang-aware Lottery; Automatic Trading | Inter-user fairness guarantee | ✔ | - | - | XL | - |
| | Ada-SRSF [166] | ♣ | AdaDUAL; Least workload first | Less communication contention | ✔* | - | - | S | - |
| | Antman [176] | ✿♣ | Framework-cluster Co-design | Better GPU Utilization | - | ✔ | - | XL | ✔ |
| | CODA [142] | ✿♣ | Adaptive CPU allocator; Contention eliminator | Non-dominant resource aware | ✔* | - | - | - | - |
| | Co-scheML [86] | ♣ | Interference-aware Scheduler; Random Forest | Better GPU Utilization; JCT Reduction | - | - | - | S | - |
| | Elan [177] | ✿♣ | Hybrid scaling; IO-free replication;etc | Better GPU Utilization; Less IO | - | ✔ | - | L | - |
| | E-LAS [153] | ♣ | Real-time Epoch Progress Rate; LAS | Information-agnostic | - | - | - | - | - |
| | Gavel [123] | ♣♥ | Linear Programming; Round-based Scheduling | Heterogeneity-aware | ✔ | - | - | M | ✔ |
| | GENIE [18] | ♠ | Light Profiler | QoS Guarantee | - | ✔ | - | S | - |
| | HiveD [211] | ♣ | Buddy Cell Allocation | Better Resource Utilization | - | - | - | L | ✔ |
| | MARBLE [55] | ✿♣ | Offline profiling-based scaling | Better GPU Utilization | - | ✔ | - | S | - |
| | MLCloudPrice [122] | ♣♦ | Linear Programming; Spot-instance Training | Cloud Cost Reduction | - | - | - | - | ✔ |
| | MLFS [162] | ♣♠ | RL Scheduler | Optimize Multiple Objectives | - | - | - | - | ✔ |
| | Non-Intrusive [167] | ✿♣ | SideCar; Early initialization | Framework non-intrusive | - | ✔ | - | S | - |
| | Parrot [103] | ♣ | Linear Programming | Better Bandwidth Utilization | - | - | - | - | - |
| | Salus [195] | ✿♣ | Fast job switching; Memory sharing | Better GPU Utilization | - | - | - | S | ✔ |
| | SPIN [57] | ♣ | Rounding-based Randomized Approximation | Robust Time Misestimation | - | - | - | - | - |
| | Themis [113] | ♥ | Finish-time Fairness; Auction Bid | Better Fairness; GPU Utilization | - | - | - | L | - |
| | Vaibhav et al. [142] | ✿♣ | Dynamic programming optimization | Better GPU Utilization | - | ✔ | - | M | - |
| | Yeung [186] | ✿ | GPU Utilization Prediction | Better GPU Utilization | - | - | - | - | - |
| 2021 | $DL^2$ [133] | ♣ | RL Scheduler | JCT Reduction | - | ✔ | - | S | ✔ |
| | AFS [70] | ✿♣ | Apathetic Future Share; CoDDL | Better GPU Utilization | - | ✔ | - | L | - |
| | ANDREAS [36] | ♦ | Randomized Greedy Algorithm | Energy Cost Reduction | - | - | - | S | - |
| | Astraea [184] | ♥ | Long-term GPU-time Fairness | Fairness Guarantee | - | - | - | - | - |
| | Chronus [41] | ♠ | Linear Programming; Local Search Allocation | SLO Guarantee | - | - | - | S | ✔ |
| | DynamoML [20] | ✿♣ | Combine KubeShare and Dragon | Better GPU Utilization | - | ✔ | - | S | - |
| | Helios [65] | ♣ | Data-driven Prediction; QSSF; CES | Workload Analysis; Energy Conservation | - | - | - | - | ✔ |
| | Horus [187] | ✿♣ | XGBoost-based interference prediction | No need to online profiling | - | - | - | S | - |
| | Jigsaw [89] | ♣ | Structured Partial Training | Algorithm-System Co-design | - | - | - | M | - |
| | Liquid [48] | ♣ | Best-fit; Grouping Genetic | Accelerate job execution | - | - | - | M | ✔ |
| | ONES [9] | ✿♣ | Online evolutionary search | Better GPU Utilization | - | ✔ | - | S | ✔ |
| | Pollux [134] | ✿♣♥ | Goodput; Dynamic batch size & lr | Better GPU Utilization | - | ✔ | ✔ | L | ✔ |
| | POP [121] | ♦ | Partitioned Optimization | Reduce Scheduling Overhead | ✔ | - | - | L | ✔ |
| | SMD [194] | ♣ | Multi-dimensional-knapsack Decomposition | JCT Reduction | - | - | - | - | - |
| 2022 | Ali-MLaaS [171] | ✿♣ | GPU Sharing; Predictable Duration | Fine-grained Workload Analysis | - | - | - | - | ✔ |
| | Aonline [206, 217] | ✿♣ | Integer Linear Programming | JCT Reduction | - | ✔ | - | - | - |
| | CloudBrain [182] | ✿ | GPU Sharing | Fine-grained Workload Analysis | - | - | - | - | ✔ |
| | EDL [174] | ✿♣ | Stop-free scaling; Graceful exit;etc | Better GPU Utilization | - | ✔ | - | L | - |
| | GADGET [193] | ✿♣ | Greedy; G-VNE | JCT Reduction | - | ✔ | - | - | ✔ |
| | Muri [212] | ✿♣ | Multi-resource Interleaving | Minimize Interference | ✔* | - | - | L | ✔ |
| | Singularity [150] | ✿♣ | Device Proxy; Replica Splicing; etc | User code non-intrusive; Efficient | - | ✔ | - | M | - |
| | Synergy [119] | ✿♣ | Optimistic Profiling; Greedy Scheduling | Non-dominant Resource-aware | - | - | - | M | - |
| | Titan [40] | ✿♣ | Task Merge; Pipeline Switch | Foundation Model Fine-tuning | - | - | - | - | - |
| 2023 | EasyScale [101] | ✿♣ | EasyScale Thread Switching | Consistent Model Accuracy | ✔ | ✔ | - | L | ✔ |
| | ElasticFlow [46] | ♠♣ | Minimum Satisfactory Share | SLO Guarantee | - | ✔ | - | XL | ✔ |
| | FGD [172] | ✿♣ | Fragmentation Gradient Descent | Minimize GPU Fragmentation | - | - | - | - | ✔ |
| | Hydro [67] | ♦✿ | Model Scaling; Trial Interleaving | Efficient Large Model Tuning | ✔ | ✔ | ✔ | XL | ✔ |
| | Lucid [68] | ✿♣ | Non-intrusive Profiling & Packing | Code Non-intrusive | - | - | - | M | ✔ |
| | ModelKeeper [91] | ♦♣ | Automated Weight Transformation | Faster Convergence | - | - | ✔ | M | ✔ |
| | PowerFlow [45] | ♦♣ | Network Packing; GPU Scaling | Energy-aware | - | ✔ | - | - | - |
| | Shockwave [215] | ✿♣♥ | Volatile Fisher Market | Better Fairness | - | ✔ | - | M | ✔ |
| | Sia [79] | ✿♣♥ | Goodput; ILP | Heterogeneity-aware | ✔ | ✔ | ✔ | M | ✔ |
| | SiloD [210] | ✿♣♥ | Cache Subsystem | Cache and IO-aware | ✔* | - | - | L | - |
| 2024 | Acme [66] | ♣ | Decoupled Scheduling | LLM Workload Analysis | - | - | - | - | ✔ |
| | Cassini [136] | ✿♣♥ | Network Interleaving | Communication-aware | - | ✔ | - | S | - |

**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, ∞] -: no evaluation on a physical cluster or not clearly specified.

*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 [70, 89, 99, 134, 174–177], which will be elaborated in Section 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 explicitly use some heuristic functions as the job scheduling priority. For instance, Tiresias [47] 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, which is inspired by the classic **Multi-Level Feedback Queue (MLFQ)** algorithm [22]. These techniques enable Tiresias to beat the classical YARN-CS [160] significantly. E-LAS [153] 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.

An alternative strategy is to use **machine learning (ML)** techniques for job scheduling. *Sched*$^2$ [110] is a scheduler based on **reinforcement learning (RL)**. It utilizes a Q-network that takes the job state and GPU datacenter state as input and outputs the optimal job to be scheduled. MLFS [162] also leverages RL to determine the job priority and resource allocation. The RL model takes as input the job time information, resource demand, and accuracy requirements and yields scheduling decisions to reduce the average latency. Helios [65] characterizes the production training jobs from a shared GPU datacenter in SenseTime and then adopts a variety of ML algorithms to predict the job priority from the history job information. The prediction result suffices to minimize the cluster-wide job latency. JPAS [216] is a scheduler based on the accuracy curve fitting technique to expedite the feedback-driven exploration of general training workloads T4.[1] 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.

> **Insight 1**: The heuristic functions depend on job-related information (e.g., duration, speed, resource utilization) to identify the job priority. Some Schedulers lacking this information could use ML algorithms to make predictions.

The timing efficiency of DL training jobs is highly dependent on the job placement, where different placement policies can lead to different communication overheads. Users prefer the strict placement locality to maintain the training speed T2. Aforementioned schedulers including Philly and Tiresias as well as E-LAS consider meeting the strict placement locality as much as possible. Amaral et al. [3] also focused on this direction, and they used a profiler to measure the placement sensitivity of each job and thus perform a best-effort approach to schedule locality-sensitive jobs in a packing manner. SMD [194] 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 $\epsilon$-approximation algorithm called sum-of-ratio multi-dimensional-knapsack decomposition to solve it. Similar to conventional cloud trace characterization from Google [138], Philly [80] investigates a production GPU cluster trace from Microsoft and conducts a thorough analysis about the

[1] TX and IX indicate the training and inference job characteristic (Section 2.1) need to be considered.

impact of gang scheduling and locality constraints on the queuing delay and job runtime. Based on this analysis, 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 [211] 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 [47] and Philly [80] to reduce the queuing delay. *Sched*$^2$ [110] 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 [197] can bring reward to the GPU datacenter in the long term for satisfying the placement locality in the near future. SPIN proposes a rounding-based randomized approximation method to achieve this goal, which has strong robustness even with inaccurate job runtime estimation.

Moreover, ModelKeeper [91] identifies model architectural similarity with previously trained models, selects a parent model with high similarity and good model accuracy, and performs structure-aware transformation of weights to preserve maximal information from the parent model during the warmup of new model weights. It significantly achieves faster training completion time and no reduction in model accuracy.

**Insight 2**: The effectiveness of placement policy is affected by the jobs, cluster status, and future job arrival. We recommend plugin-style schedulers (e.g., HiveD) to improve the quality of placement decisions without the need of changing deployed schedulers. For better performance, ML/RL algorithms can be used to incorporate future knowledge to make more informative placement decisions (e.g., *Sched*$^2$, SPIN).

**(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 [36] 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 [65] has no assumption about postponing the execution of DL training jobs. It predicts the future resource utilization from the history logs and decides how many GPU nodes should be turned on/off to reduce the electricity consumption. PowerFlow [45] aims to save energy via dynamically allocating GPUs and adjusting the GPU-level or job-level configurations of DL training 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. As such, the cost of a training workload could be considerably expensive. It is critical to reducing such financial costs 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 [214] is a scheduler to guarantee the cost-effectiveness of cloud resource provision for PS training. This scheduler uses an analytical performance model to identify an optimal resource type and PS configurations to maintain the training throughput while minimizing the monetary cost. Analogously, *FC*$^2$ [155] is a scheduler to recommend cost-effective and high-performing cloud resource configurations for PS training. It proposes a heuristic method named *Scala-Opt* to decide the worker instances that can guarantee the job throughput while

maximizing the cost savings. Jahani [73] 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. Additionally, FfDL [78] 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 to attain the cost-effective objective in the cloud. MLCloudPrice [122] proposes to move the workloads between spot and on-demand instances, which opportunistically utilizes the low-pricing spot instance to reduce the training cost.

**Insight 3**: Existing cost-effective schedulers mainly minimize the power consumption and financial costs in a single datacenter. Few efforts have been done to leverage the cost difference among different datacenters to reduce the cost. Such research endeavors have been made in prior works [111, 208]. We expect more studies in this direction.

*3.1.2 Fairness.* Fairness indicates how fairly the compute resources are allocated among different entities, including user groups (i.e., tenants) and workloads. The design of fairness schedulers for conventional workloads follows some typical fairness principles, such as sharing incentive, strategy-proofness, envy-freeness, and Pareto efficiency [42]. Maintaining fairness for DL training jobs also follows these principles, but requires new scheduler designs for two reasons: (1) A GPU is an indivisible resource in common settings (gang scheduling) for DL training **I6**; (2) DL training exhibits resource-heterogeneity preference **I1**. Below, we discuss the new works that can address these two challenges.

**(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 [113] maintains 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. Themis also considers the placement preferences of training workloads and builds a two-level scheduling architecture for biding resource allocation to resolve severe fairness sharing loss brought by poor resource allocations. Astraea [184] concentrates on 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. Besides, Shockwave [215] extends classic market theory from static settings to dynamic settings to co-optimize efficiency and fairness.

**(2) Heterogeneous compute resources.** It is relatively easy to maintain fairness over one type of GPU. 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 **I1**. A couple of works have introduced solutions to achieve fairness in the heterogeneous environment.[2]

To achieve fairness over GPUs and other compute resources, Allox [92] is a fairness scheduler that 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 and proposes a greedy heuristic solution to solve it in an effective and scalable way. Dorm [154] 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 influence the training job throughput. It dynamically

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

partitions different types of compute resources for each DL training job. It formulates a MILP problem with resource-utilization fairness as the optimization objective. The scheduling decision in each round is made by calling the MILP solver to optimize 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. $Gandiva_{fair}$ [14] 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 [123] 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 computationally prohibitive to scale up Gavel to a large datacenter. To this end, POP [121] proposes to partition a large datacenter into several smaller ones. The original complex optimization formulation is decomposed into multiple smaller problems and can be solved in parallel.

**Insight 4**: The presence of heterogeneous GPU (e.g., GPUs from different generations in the same environment) increases the scheduling decision space. The mathematical optimization becomes a preferred way to maintain heterogeneity-aware fairness. RL-based schedulers are expected to further promote fairness.

*3.1.3 Deadline Guarantee.* Different from the efficiency goal, 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 of the deadline requirement in DL workloads. An early deadline-aware scheduler for DL training workloads is GENIE [18]. 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 [18] does not investigate the deadline requirement from users and cannot support a mixed workload of deadline and best-effort jobs, a common scenario discussed in the cloud [159]. In Reference [41], a user survey is conducted to uncover users' latent needs regarding 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 simultaneously. It adopts the MILP solver to address the deadline-guarantee scheduling task with the resource and time constraints. Chronus outperforms existing deadline schedulers [18, 131] in reducing deadline miss rates and improving the latency. ElasticFlow [46] further leverages elastic scaling to dynamically allocate resources for every single job to meet their deadlines.

**Insight 5**: We anticipate optimizing the deadline guarantee and cost efficiency receive more attention. A more interesting research direction is to jointly optimize both objectives.

### 3.2 Resource Utilization Manner

In addition to the scheduling objective, another orthogonal view to categorize schedulers is based on the resource-utilization manner of DL training workloads. 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 training performance. Synergy [119] observes that different DL training jobs exhibit different levels of sensitivity to the CPU and memory allocation. 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 greedily packs runnable jobs along multiple resource dimensions to minimize the cluster fragmentation as much as possible (i.e., maximize the number of idle GPU nodes) T1. CODA [209] observes that CPU jobs colocating within the same compute node can interfere with the training jobs due to CPU resource contention. It proposes an adaptive CPU allocator to identify the optimal CPU cores for each DL training job. Moreover, Muri [212] proposes a technique to interleave critical resources (e.g., GPU, CPU, network, storage) using a Blossom-based scheduler to mitigate interference. Cassini [136] focuses on accommodating multiple ML jobs' network needs. SiloD [210] advocates for the co-design of data caching and job scheduling.

Beyond the CPU and memory resources, network bandwidth is another bottleneck for efficient DL training. Ada-SRSF [166] aims to reduce the communication contention among jobs. It is combined with the classical SRSF algorithm to relax the communication contention of two jobs if it can reduce the job completion time. Liquid [48] proposes a network-efficient scheduling solution to achieve better placement for PS-based distributed workloads. 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 [103] is a framework to manage network bandwidth contention among training jobs using the PS architecture. It uses a **linear program (LP)** solution to derive a weighted bandwidth scaling strategy to minimize the time cost in the communication stage.

**Insight 6**: We observe that current schedulers consider a subset of heterogeneous resources (e.g., CPU and memory, networking). In the future, the datacenter might incorporate other new hardware, and existing schedulers for heterogeneous resources might become obsolete. A universal scheduler design is anticipated to manage various heterogeneous resources by identifying the root cause of the performance bottleneck of DL training.

*3.2.2 GPU Sharing.* With the increased compute capability and memory capacity of GPUs, the conventional placement approach that makes each DL job exclusively use the GPU can lead to severe resource underutilization. Performing GPU sharing becomes a promising technique to improve the system throughput T5. In this context, *utilization* refers 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 [195] 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 [175] aims 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 [89] 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 computing resources, however, it might lead to accuracy loss to some extent. Recently, Antman [176] 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 [171] provides a comprehensive analysis of large-scale workload traces in Alibaba and discloses the benefit of GPU sharing in production GPU datacenters. FGD [172] proposes a Fragmentation Gradient Descent algorithm to minimize GPU fragmentation.

Alternatively, some works use data-driven approaches to make the GPU sharing decision. Horus [186, 187] 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 severe interference and communication delays. Co-scheML [86] 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 [48] also supports fine-grained GPU sharing for further resource-utilization improvement using a random forest model. Harmony [6] applies an RL model to make placement decisions for minimizing interference and maximizing the throughput for bin-packing jobs in a GPU datacenter. Lucid [68] achieves non-intrusive workload profiling and packing based on interpretable models.

With the rapid popularity of **large language models (LLMs)** in recent years, there are more datacenter-level optimization opportunities tailored for LLM workloads. Hydro [67] extends resources of hyperparameter tuning workloads by interleaving them with pipeline-enabled LLM pretraining tasks, effectively utilizing idle time intervals on each node known as bubbles, which are caused by the gaps between the forward and backward processing of microbatches. Titan [40] merges multiple LLM fine-tuning workloads to share the same GPU resource, which can significantly reduce JCT. Moreover, Acme [66] presents an in-depth characterization study of a six-month LLM development workload trace collected from Shanghai AI Laboratory and provides some insights to optimize systems tailored for LLMs.

**Insight 7**: The low single-GPU utilization leaves a large optimization space for GPU sharing. ML algorithms present advantages in packing training jobs in a single GPU to improve GPU utilization and cluster-wide efficiency.

*3.2.3 Elastic Training.* Elastic training indicates dynamically changing the resource allocations of training jobs to improve the resource utilization and job efficiency [16]. We consider two primary elastic training approaches adopted by existing elastic schedulers: (1) resource elasticity, which dynamically allocates GPUs for jobs, and (2) batch elasticity, which dynamically determines the batch sizes to adapt the allocated resources to maximize the job throughput. Resource elasticity in big data workloads has been investigated for many years [4]. Recent resource-elastic schedulers for DL training mainly focus on placement sensitivity and heavy preemption overhead. We first discuss how DL schedulers handle the uncertain job runtime led by placement sensitivity and address heavy preemption overhead brought by resource autoscaling. We also discuss the scheduling approaches to handle the resource elasticity of DL training. Followed by resource elasticity, we investigate how batch elasticity further improves cluster-wide efficiency.

Elastic training exacerbates the effect of placement sensitivity on DL training speed as a result of dynamic resource allocations. To make informed scheduling decisions, many schedulers focus on the design of accurate performance modeling for elastic training. Gandiva [175] designs a *Grow-Shrink* mechanism that uses the profiling information to estimate each job's progress rate and then allocates GPUs accordingly. Optimus [132] estimates the loss reduction rate on any placement policies based on a performance model. Leveraging the performance model, Optimus successfully maximizes the cluster-wide training throughput. MARBLE [55] 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. AFS [70] is proposed based on the insight that the scheduler should proactively prepare for resource contention in the future by utilizing the current resources. It considers both resource efficiency and job duration for resource allocation while amortizing the cost of future jobs into the calculation of the current share. The effectiveness of AFS depends upon the accurate job duration prediction. GADGET [193] 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 a provable guarantee for high performance. One challenge of elastic training is that it changes the hyperparameters and the training procedure according to available resources and thus introduces the non-determinism during training and inevitably affects model accuracy. To this end, EasyScale [101] preserves the data-parallel training behaviors strictly, traces the consistency-relevant factors carefully, and utilizes the deep learning characteristics for EasyScaleThread abstraction and fast context-switching.

Frequent resource re-allocation in elastic training leads to the high overhead of preemption, making the reduction of resource re-allocation overhead a priority for many elastic schedulers. For instance, ELAN [177] employs asynchronous coordination to minimize start and initialization overhead during adjustments, while Wang et al. [167] introduce an early initialization mechanism through a SideCar process in a Kubernetes-based framework, resulting in efficient GPU re-allocation and minimal disruption to deep learning training frameworks. EDL [174] 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. DynamoML [20] is a Kubernetes platform that combines KubeShare [185] and Dragon [108] for DL workload scheduling. It supports easy and fast resource autoscaling for training workloads. More recently, Microsoft presents Singularity [150], a scheduler 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* to decouple DL training and resource allocation. In other words, jobs are unaware of resource elasticity.

Some works propose mathematical optimization to elastically determine resource allocations for DL training jobs. OASiS [7] 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. AOnline [206, 217] 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 [17] adopts K-means to divide concurrent jobs into several groups based on the computation-communication ratio similarity to reduce the state space and accelerates the convergence speed of the RL model. RIFLING chooses the Q-Learning algorithm and allows the RL model to perform online update from historical logs to adapt to the workload variation. $DL^2$ [133] is another RL scheduler focusing on the PS architecture and dynamically adjusting 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 takes

the job state and resource state as input and makes the resource allocation decision. The reward function targets the cluster-wide normalized epoch progress.

In addition to the resource elasticity, the batch size of DL training jobs can also be dynamically adjusted. Vaibhav et al. [142] design a job scalability analyzer and a dynamic programming-based optimizer to determine the batch sizes and GPU counts for DL jobs. However, Vaibhav et al. [142] assume that changing batch sizes within an appropriate range does not sacrifice the model accuracy. Pollux [134] 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 measuring training performance including system throughput and statistical efficiency. It designs a joint scheduling architecture to maximize the goodput. Lyra [99] 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 [9] automatically manages the elasticity of each job based on the training batch size using an online evolutionary search algorithm. Sia [79] further schedules adaptive DL jobs on heterogeneous resources to achieve better cluster performance.

**Insight 8**: Elastic training can expedite the training speed but incurs accuracy loss and reproducibility crisis, particularly for batch elasticity (e.g., Pollux). It is advisable to implement elastic schedulers when the model accuracy has demonstrated resilience to elastic training, such as fine-tuning BERT as demonstrated in Reference [28], or in scenarios where prioritizing speed over accuracy is necessary.

### 3.3 Discussion

The scheduling objective plays an important role in designing schedulers for GPU datacenters. We have discussed the scheduling objective when DL training jobs use a fixed exclusive resource allocation in Section 3.1. Section 3.2 mainly discusses latency minimization and fairness using different resource-utilization manners. Attaining the scheduling objectives, including cost-efficiency and deadline guarantee for heterogeneous resources, GPU sharing, and elastic training, deserves greater attention and consideration.

According to the unique resource utilization manner of DL training, datacenter managers can enhance the overall resource utilization and minimize the job latency as well as promote fairness. 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 validation. Job colocation can cause potential performance degradation and fault tolerance issues, 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, further efforts are necessary to reduce operational cost and enhance job throughput in the scheduling of deep learning training jobs. Scheduling algorithms encompass heuristic, mathematical optimization, and ML-based approaches. Heuristic solutions serve as useful benchmarks for evaluating the performance of new algorithms. Mathematical optimization, while potentially providing optimal solutions, can incur significant computational overhead. ML-based schedulers are particularly advantageous in complex scheduling scenarios, such as GPU sharing and elastic training. However, the design of these ML-based schedulers, including the model architecture, learning algorithms, and loss functions, requires further refinement and

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 [24] | ♣♥♠ | Query-level caching; Layered architecture | General abstraction for model selection | ✔ | - | - | M | ✔ |
| 2018 | Ease.ml [102] | ♣ | Multi-tenant model selection | Homogeneous declarative inference platform | - | - | - | - | ✔ |
| | HiveMind [124] | ♠ | Sharings of pipelines, weights, and layers | Multi-model training and inference | ✔ | ✔ | - | S | - |
| | Space-time [76] | ♠✿ | GPU sharing across space and time | Performance isolation under sharing | ✔ | ✔ | - | S | - |
| 2019 | Ebird [25] | ♥♠✿ | CUDA stream paralleism; GPU-side memory pool | Transfer-computation overlapping | ✔ | ✔ | - | S | ✔ |
| | Gilman et al. [44] | ♥♠ | Preloading model into GPU memory | DNN model execution caching | - | ✔ | - | - | - |
| | MArk [199] | ♥♦ | Predictive autoscaling on serverless instances | Flexible to burst requests | ✔ | - | ✔ | S | ✔ |
| | Nanily [157] | ♥♠ | Adaptive batching; autoscaling | Batch size adjustment by remaining time | ✔ | - | - | S | - |
| | ParM [87] | ♥ | Coded-computation via a learning-based approach | Erasure-coded resilience for inference | ✔ | - | - | M | ✔ |
| | RRL [135] | ♥ | Region-based Reinforcement Learning | Parallelism configuration tuning | ✔ | ✔ | - | M | ✔ |
| | Tolerance Tiers [54] | ♣♥♦ | Service Version Ensembling | Explicit accuracy-latency tradeoff in requests | - | - | ✔ | S | - |
| | TrIMS [27] | ♥♠✿ | Multi-layered caching across FaaS | Memory efficiency by sharing | ✔ | ✔ | ✔ | S | ✔ |
| 2020 | AutoDeep [104] | ♥♦♠ | BO and DRL | Cloud configuration and device placement | - | ✔ | ✔ | S | - |
| | Clockwork [51] | ♥♠ | Consolidating choice | Predictable E2E performance | ✔ | - | - | M | ✔ |
| | DyBatch [207] | ♥♠ | Task slicing and reordering | Fine-grained batching; Fairness-driven scheduling | ✔ | ✔ | - | S | - |
| | GSLICE [29] | ♠✿ | Dynamic GPU resource apportioning | Efficient fine-grained sharing | ✔ | ✔ | - | S | - |
| | Inferline [23] | ♥♦ | Low-frequency planner; high-frequency tuner | Near-optimal scaling cost-efficiency | ✔ | - | ✔ | M | ✔ |
| | Irina [173] | ♥♠✿ | Batching, colocation, and preemption | Graceful general preemption | ✔ | ✔ | - | S | - |
| | PERSEUS [96] | ♥♦♠ | Performance and cost characterization on Cloud | Cost savings under GPU instances | ✔ | - | ✔ | S | ✔ |
| 2021 | Abacus [26] | ♥♠ | Runtime operator scheduling | Deterministic latency under colocation | - | ✔ | - | M | ✔ |
| | INFaaS [141] | ♥♦♠ | VM- and model-level autoscaling | Automatical model variants selection | - | ✔ | ✔ | - | ✔ |
| | Mendoza et al. [116] | ♥ | Latency degradation prediction during colocation | Safe colocation | - | ✔ | - | M | - |
| | MIG-SERVING [156] | ♥♦ | Greedy algorithm, GA, and MCTS | MIG-enabled inference scheduling | ✔ | ✔ | - | S | - |
| | Morphling [163] | ♦♠ | Model-agnostic meta-learning | Performance prediction under different configuration | ✔ | ✔ | ✔ | - | ✔ |
| 2022 | Cocktail [52] | ♣♥♦ | Weighted majority voting policy | Ensembling-based model selection | - | - | ✔ | - | - |
| | Gpulet [21] | ♦♠✿ | Spatio-temporal Sharing | Higher Throughput | ✔ | ✔ | - | M | ✔ |
| 2023 | AlpaServe [105] | ♥♠ | Integrating Model Parallelism | Higher Throughput | ✔ | ✔ | - | M | ✔ |
| | Clover [97] | ♦ | Mixed-quality Models; MIG | Carbon-aware Inference | ✔ | ✔ | - | M | ✔ |
| | DeepPlan [81] | ♥♠ | Direct-host-access | Lower Tail Latency | ✔ | - | - | S | ✔ |
| | DeltaZip [183] | ♦♠ | Compressing Model Deltas | Efficient LLM Serving | ✔ | ✔* | - | S | ✔ |
| | iGniter [178] | ♦✿ | Cost-efficient GPU Provision | Lower Monetary Cost | ✔ | ✔ | ✔ | M | ✔ |
| | Kairos [98] | ♥♦♠ | Query Distribution; Optimize Configuration | Cost Budget Aware | ✔ | - | ✔ | M | ✔ |
| | MOSEL [63] | ♥♠ | Modality Selection | Tailored for Multi-modal Model | ✔ | - | - | S | ✔ |
| | Punica [16] | ♦♠ | LoRA; SGMV | Efficient LLM Serving | - | ✔* | - | M | ✔ |
| | Shepherd [202] | ♦♠♥ | Preemption; Periodic Planning | Higher Throughput | ✔ | - | - | M | - |
| | S-LoRA [147] | ♦♠ | LoRA; Unified Paging | Efficient LLM Serving | - | ✔* | - | M | ✔ |
| | Symphony [15] | ♦♠♥ | Non-work-conserving; Autoscaling | Higher Throughput | ✔ | - | - | M | - |
| | Tabi [170] | ♥♣ | Multi-level Inference; Attention-based Pruning | Tailored for Discriminative LLM | - | - | - | S | - |
| 2024 | SpotServe [117] | ♥♦ | Kuhn-Munkres Algorithm; Stateful Recovery | Lower Monetary Cost | ✔ | - | ✔ | M | ✔ |

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

evaluation through benchmark studies. Further collective efforts are needed to advance the practicality of ML-based schedulers.

## 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 (Section 2.1.2), which demand new scheduling solutions. Similar to Section 3, we categorize these inference scheduling techniques based on their objectives and resource-utilization manner. Then, we give some implications from these works at the end of this section. Table 2 summarizes the relevant papers and their features. The table only includes schedulers for general DL inference workloads, while we mention some related scheduling techniques from other works in the following discussion.

### 4.1 Scheduling Objectives

We first review prior works based on the scheduling objectives.

*4.1.1 Efficiency.* As discussed in Section 2.1.2, 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 [12]. The challenge here is that there exist tradeoffs 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 prediction accuracy is a perpetual objective in an inference system. Although the prediction accuracy of fixed input and model (with deterministic inference execution, which is the usual case) is determined, the scheduling system could achieve better accuracy by selecting **different** models and making resource allocation between models more intelligent. 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 the model selection and resource allocation among different candidates. Ease.ml [102] 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 the model ensemble, which combines the prediction results from multiple models to improve the prediction accuracy and generalization. Clipper [24] 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 [169] 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. In this section, we discuss the scheduling techniques to improve the latency efficiency from two perspectives: (1) how to reduce the latency of a fixed number of inference requests and (2) how to efficiently allocate resources to meet the latency requirement. 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 resource allocation.

It is common to launch multiple inference execution instances concurrently to reduce the latency of inference requests as much as possible due to the low GPU utilization for each request. Therefore, the inference scheduler can make scheduling decisions aiming at scaling up resources according to the request density to maintain a satisfactory latency. Clipper [24] 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 [199, 200] scales the inference instances with the cloud services. It selects and combines different cloud services such as AWS EC2 and Lambda 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 [23] 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 [26] 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 [87] 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 schedule the inference workloads and rearrange the execution sequence at the job level. Irina [173] models 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 a 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 [158] 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 future resource utilization. From the temporal aspect, Kube-Knots predicts the peak inference usage and tries to avoid co-locating jobs that 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 [199, 200] 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 [104] 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. Kairos [98] aims to maximize throughput under cost budget and SLO requirement via efficient query distribution among cloud instances and find a high-throughput heterogeneous configuration. iGniter [178] builds a lightweight analytical performance model to explicitly capture the performance interference among workloads and propose a cost-efficient GPU resource provisioning strategy to guarantee SLOs. Besides, spot instances also can be leveraged to minimize cost. Cocktail [52] develops a distributed weighted auto-scaling policy and leverages the spot instances to minimize cost. Similarly, SpotServe [117] leverages the autoregressive nature of LLMs and introduces stateful inference recovery, which

allows inference engines in cheap preemptible instances to commit their progress at the token level, rather than the request level as seen in prior work.

Besides monetary cost, 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 (Section 3.1.1), it is relatively less studied for the inference workloads. Some works [60, 130] provide some energy characterizations of production DL inference clusters. Kube-Knots [158] 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. Clover [97] achieves lower carbon emission using mixed-quality model variants.

**(4) Tradeoffs between accuracy, latency, and cost.** The objectives of accuracy, latency, and cost are not independent. 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 tradeoffs 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 [24] 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 [203] pursues the tradeoffs 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. Tabi [170] is optimized for discriminative language models via a multi-level inference engine, which uses the calibrated confidence score to decide whether to return the accurate results of small models or re-route them to LLMs. MOSEL [63] is designed for multi-modal models that carefully picks input modalities per request based on user-defined performance and accuracy requirements.

Some schedulers allow users to specify their demands about accuracy, latency, and cost and make scheduling decisions directly according to the demands. Tolerance Tiers [54] 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 [141, 179] also asks for the throughput 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 profiling information from the profiling to serve users' requests. Each model variant may have different inference latencies and monetary costs during execution. Therefore, INFaaS applies two levels of autoscaling to support guaranteed latency requirements and improve cost efficiency

under varying loads. In addition to autoscaling at the hardware (VM) level, INFaaS also monitors load by maintaining a state machine for each model variant and supports scaling between model variants with different types and replicas. 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 throughput to fulfill the burst queries.

**Insight 9**: Scheduling inference workloads for accuracy, latency, and cost efficiency is a trilemma. Latency is often more important in online inference systems. By intelligently provisioning and allocating resources among different models, a scheduling system could achieve a tradeoff between these objectives. Some schedulers allow users to specify their demands about accuracy, latency, and cost and make scheduling decisions directly according to the demands.

*4.1.2 System Throughput.* Another important objective for scheduling inference workloads is to improve throughput capability. The techniques to achieve this goal are 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. 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 [24] and Rafiki [169] 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.

Second, some schedulers propose optimization-based methods to balance the inference delay and throughput. MArk [199, 200] 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 [157] 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 [207] 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 is caching and reusing the prediction results across different requests. 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 [24] 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. Inference scheduling system resides models in the GPU device, thus increasing system throughput. Gillman et al. [44] 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, such as framework-level GPU memory-friendly optimization, proactively loading and evicting, and cluster-level GPU memory allocation. To address the limitation of GPU memory, TrIMS [27] organizes the memory sharing of different models in a more systematic design. TrIMS reconciles the lifecycle of model memory consumption and carefully handles 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 throughput 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 [163] formulates the optimal configuration search as a few-shot learning problem. Then, it adopts **model-agnostic meta-learning (MAML)** [37] 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 [135] 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. Shepherd [202] exploits the insight that aggregating request streams into moderately sized groups greatly improves predictability, permitting high resource utilization as well as scalability. Symphony [15] utilizes a non-work-conserving scheduling algorithm capable of achieving high batch efficiency while also enabling robust autoscaling.

**Insight 10**: It is a common strategy for an inference scheduling system to sacrifice the latency of inference requests in exchange for higher resource utilization and lower average inference request latency, thus achieving higher throughput. Other strategies, such as caching, can also be used in exchange for higher throughput at the cost of higher resource usage. The resource usage characteristics of inference workload leave potential throughput gains for schedulers.

### 4.2 Resource Utilization Manner

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

*4.2.1 Colocation and Resource Sharing.* From the challenges discussed in Section 2.1.2, executing one inference request can lead to resource underutilization. The development of GPU architecture designs motivates GPU sharing from the hardware perspective and software perspective [148, 195]. GPU sharing across different inference requests can improve resource utilization by better leveraging GPUs' parallel computing 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 [76] 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 fusing concurrent small kernels into larger one to fill up the GPU utilization under the time-sharing mechanism. Nexus [146] focuses on optimizing video analysis on GPU datacenters. It also applies a heuristic approach to select the requests to be co-located on the same GPU. First, it identifies the optimal batch size for throughput and SLO needs of the 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 inference workloads. GSLICE [29] 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. The performance improvement reaches a point of diminishing returns after certain configurations, which has a non-linear relationship with the throughput and latency of the inference. MIG-SERVING [156] leverages the hardware support for GPU virtualization (i.e., MIG on NVIDIA A100) for efficient GPU colocation. With MIG, an A100 could be partitioned into several instances with smaller hardware capacities under hard constraints. MIG-SERVING discovers 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 colocation interference. INFaaS [141, 179] 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. PERSEUS [96] 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. AlpaServe [105] proposes a novel model placement algorithm to incorporate model parallelism in serving systems to improve inference throughput. Gpulet [21] enables spatio-temporal sharing to maximize the resource efficiency. To achieve more stable tail latency, DeepPlan [81] enables a direct-host-access technique that allows access to particular layers of models in the host memory directly from GPU without loading.

LLMs become ubiquitous in modern applications, however, serving these models is expensive and challenging due to their substantial computing requirements and memory footprint. To this end, DeltaZip [183] efficiently serves multiple full-parameter fine-tuned models concurrently by aggressively compressing model deltas. Besides, **Low-Rank Adaptation (LoRA)** [64] is another

commonly adopted approach for model fine-tuning. Punica [16] contains a new CUDA kernel design that allows batching of GPU operations for different LoRA models. S-LoRA [147] proposes a unified paging technique that uses a unified memory pool to manage dynamic adapter weights with different ranks and KV cache tensors with varying sequence lengths.

*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 [199, 200] 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 [96] performs a similar cost-efficiency characterization, considering instances with multiple onboard GPUs. It examines that DL models with high GPU utilization could introduce intensive interference with other inference workloads in multi-GPU instances. AutoDeep [104] and Cocktail [52] focus on the price of different cloud configurations and search for the best configuration according to the pricing information.

**Insight 11**: Similar to training, GPU sharing benefits resource utilization and system throughput. However, the inference scheduling system needs to pay more attention to interference due to SLO violations. Heterogeneous resource scheduling is also important, especially for the cloud, which needs further investigation.

### 4.3 Discussion

Most of the above works treat 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 [95] leverages this idea, which stores and re-uses the model parameters and computation among similar white-box representations of pipeline stages to reduce resource utilization. Willump [88] develops intelligent input selection techniques based on input features and combines compiler optimizations. It could save huge computation cost while maintaining relatively acceptable accuracy. We expect more future works to 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 [192] splits the network layers and minimizes the inference latency via dynamic programming. It also adopts some searching methods such as Bayesian Optimization and RL to minimize cost. AMPS-Inf [77] jointly considers model partitioning and resource allocation by calculating the optimal execution and resource provisioning plans under the constraint of response time in 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 [8] predicts future workload patterns based on 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.

## 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 a detailed description as below.

### 5.1 Hyperparameter Optimization Workloads

A **Hyperparameter Optimization (HPO)** job aims to identify the best hyperparameters for a DL task. Technically, 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, an HPO job needs to search a set of hyperparameter configurations. Each configuration is associated with a *trial* [118], which is a common DL training workload containing training and evaluation procedures. However, in the HPO context, these trials are extremely similar and orchestrated by an HPO scheduling policy. To accelerate HPO, the policy can kill poor-performing trials through early stopping and allocate more resources to promising trials. These optimizations on HPO workloads can deliver remarkable acceleration.

**Accuracy efficiency.** Hermes [152] 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 [107] 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 [107], SEER [30] replaces the resource budget with the monetary cost budget in the cloud. Without resource constraints, 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 [137] 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 [196] further leverages the job-packing mechanism to improve resource utilization and accelerate the HPO process. Furthermore, HFTA [168] horizontally fuses the models from different trials deeply down to operators and then trains them simultaneously on a shared GPU. Retiarii [205] proposes a similar operator batching technique for non-trainable operators for **Neural Architecture Search (NAS)** jobs. Furthermore, Hydro [67] automatically applies the novel hyperparameter transfer theory MU parametrization [181] to search hyperparameters with a smaller model. Hydro further optimizes tuning efficiency via *inter-trial* and *intra-trial* fusion, which involve combining multiple models into a single entity and subsequently performing compiler-based optimization. Besides, it efficiently orchestrates the tuning process with *adaptive fusion* and *eager transfer* mechanisms. Besides, Hydro identifies the opportunities for cluster-wide optimization in the datacenter, including squeezing bubble resources with interleaving and heterogeneity-aware trial allocation.

**Cost efficiency.** RubberBand [118] 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

Table 3. Summary of Schedulers for Other Workloads in GPU Datacenters

| Type | Year | Scheduler | Objectives | Approaches | Advantages | Exp. Scale | Source Code |
|---|---|---|---|---|---|---|---|
| HPO | 2017 | HyperDrive [137] | ♣ | Dynamic Probabilistic Accuracy Prediction | JCT Reduction | S | - |
| | 2019 | HyperSched [107] | ♠ | ASHA; Dynamic Resource Allocation | Efficient HPO under DDL | S | ✔ |
| | 2020 | Retiarii [205] | ✿♣ | Cross-model Optimization | Accelerate Model Exploration | S | ✔ |
| | 2021 | Hermes [152] | ♣♦ | Time-sharing Execution with Low Overhead | JCT Reduction | S | - |
| | 2021 | HFTA [168] | ✿♣ | Horizontal Model Fusion | Resource Utilization Improvement | S | ✔ |
| | 2021 | RubberBand [118] | ♣♦ | Model JCT and Cost Prior to Runtime | Cost-effectiveness | S | - |
| | 2021 | SEER [30] | ♣♦ | Dynamic Resource Allocation | Cost Constraint HPO | S | - |
| | 2021 | Fluid [196] | ✿♣♦ | Job Packing; Elastic Training | Better Resource Utilization | S | ✔ |
| | 2023 | Hydro [67] | ✿♣♦ | Model Scaling; Trial Interleaving | Efficient Large Model Tuning | XL | ✔ |
| Hybrid | 2018 | Rafiki [169] | ♥♦ | RL to Optimize Accuracy and Latency | Unified Platform for Training and Inference | S | ✔ |
| | 2019 | Kube-Knots [158] | ♦✿ | Dynamic Container Resizing | GPU Utilization-aware Colocation | S | - |
| | 2020 | CMS [100] | ♥✿ | Common Architecture for Trainers and Modelets | Continuous Learning | S | - |
| | 2023 | Lyra [99] | ✿♥ | 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, ∞] -: no evaluation on a physical cluster or not clearly specified.

workload differs in the 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.

**Insight 12**: Compared with the general DL training workloads, HPO workloads have unique features and different objectives. For instance, we can leverage the diminishing resource requirement of HPO workloads to conserve cluster resources. Besides, the identity model of each HPO trial allows us to perform more training optimization.

## 5.2 Hybrid of Training and Inference Workloads

In Section 3 and Section 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 periodic retraining and updating. CMS [100] designs a continuous machine learning and serving platform, which orchestrates the model-training executions, model inferences, and model update services. It unifies different training jobs into a simple trainer contract and the scheduler monitors the resource consumption of these resource-intensive training tasks to avoid contention. Other techniques including model validation and model switching mechanism are also applied to guarantee model serving quality. Rafiki [169] 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 is dumped for later reuse in the inference. Other common underlying components are also shared across training and inference workloads to reduce 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 [158] minimizes resource waste by allocating offline batch jobs to better utilize the spare resources. It discovers that GPU energy efficiency increases with its utilization. Therefore, achieving higher GPU utilization indicates higher energy efficiency and fewer 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 usage. Lyra [99] 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 of resource demands for part of training jobs, Lyra successfully guarantees timely resource allocation to inference jobs via a heuristic method.

**Insight 13**: Supporting the co-optimization of training and inference workloads can further improve GPU datacenter efficiency. The two types of workloads have different characteristics and demands on resource provisioning, offering greater flexibility and opportunities for scheduling techniques such as training and inference workload switching [5], fostering the development of sophisticated scheduling strategies.

### 5.3 Discussion

In addition to optimizing general DL workloads, there are some opportunities for further resource efficiency improvement. For hyperparameter search workloads, one direction is to apply resource-utilization optimizations (e.g., GPU sharing [196]) or graph-level optimizations (e.g., trial fusion [168]). Compared with the workload-agnostic manner, it can deliver over one magnitude of 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 efficiency 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 unique features exhibited by DL workloads advocate novel DL scheduler designs to manage GPU resources and jobs in a more intelligent and efficient manner. Our comprehensive summary draws three conclusions. First, the design of new scheduling systems is motivated by the advancement of DL, as well as the evolving needs of users. Second, new works opt for the implementation of advanced algorithms (e.g., RL, MAML) to significantly enhance scheduling performance. Third, emerging hardware resources (e.g., heterogeneous GPU) can be leveraged to design efficient new schedulers.

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. Section 3 indicates that the training scheduling system can leverage elasticity to speed up DL training but they are only applicable to DL workloads that are robust to elastic training. Section 4 also points out inference scheduling system needs to take care of the latency SLO constraint. Section 5 analyzes that 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. Beyond our discussed schedulers, emerging DL workloads might require domain-specific

schedulers for extreme efficiency. For instance, training extremely large models like Transformers needs extensive and high-performance resources. Some DL workloads cannot thoroughly consume the peak computing capabilities, and the co-design of both the scheduling system and DL framework is a promising solution. 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. Therefore, some schedulers are combined with ML algorithms to approximate such information to improve the scheduling decisions. Second, some schedulers form the decision-making process as an optimization problem, which cannot be solved within an acceptable time online. As a result, existing works apply heuristic and searching-based solutions, which have been discussed in the above sections, meeting the solving latency requirement but losing optimality.

**Underlying hardware resources.** GPU datacenters are growing at an alarming speed. It is common that GPU datacenters contain complex generations of GPUs and other accelerators. Heterogeneous resources provide opportunities for schedulers to make cost-effective decisions and expedite the cluster-wide throughput, but also bring new challenges to maintain fairness. Additionally, emerging hardware resources are coupled with algorithmic innovations. For example, low-precision ALU requires to balance between accuracy and speed.

## ACKNOWLEDGMENTS

We thank the anonymous reviewers for their valuable comments.

## REFERENCES

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

[2] OpenPBS Contributor. 2022. OpenPBS. https://www.openpbs.org/

[3] Marcelo Amaral, Jordà Polo, David Carrera, Seetharami Seelam, and Malgorzata Steinder. 2017. Topology-aware GPU scheduling for learning workloads in cloud environments. In *SC'17*.

[4] Marcos D. Assunção, Rodrigo N. Calheiros, Silvia Bianchi, Marco A. S. Netto, and Rajkumar Buyya. 2015. Big data computing and clouds: Trends and future directions. *J. Parallel Distrib. Comput.* (2015), 79–80.

[5] Zhihao Bai, Zhen Zhang, Yibo Zhu, and Xin Jin. 2020. PipeSwitch: Fast pipelined context switching for deep learning applications. In *OSDI'20*.

[6] Yixin Bao, Yanghua Peng, and Chuan Wu. 2019. Deep learning-based job placement in distributed machine learning clusters. In *INFOCOM'19*.

[7] Yixin Bao, Yanghua Peng, Chuan Wu, and Zongpeng Li. 2018. Online job scheduling in distributed machine learning clusters. In *INFOCOM'18*.

[8] 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. *(IC2E'19)*.

[9] Zhengda Bian, Shenggui Li, Wei Wang, and Yang You. 2021. Online evolutionary batch size orchestration for scheduling deep learning workloads in GPU clusters. In *SC'21*.

[10] Marcel Blöcher, Lin Wang, Patrick Eugster, and Max Schmidt. 2021. Switches for HIRE: Resource scheduling for data center in-network computing. In *ASPLOS'21*.

[11] 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* 14, 1 (2016).

[12] Valeria Cardellini, Emiliano Casalicchio, Michele Colajanni, and Philip S. Yu. 2002. The state of the art in locally distributed web-server systems. *Comput. Surv.* 34, 2 (2002).

[13] 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*.

[14] Shubham Chaudhary, Ramachandran Ramjee, Muthian Sivathanu, Nipun Kwatra, and Srinidhi Viswanatha. 2020. Balancing efficiency and fairness in heterogeneous GPU clusters for deep learning. In *EuroSys'20*.
[15] Lequn Chen, Weixin Deng, Anirudh Canumalla, Yu Xin, Matthai Philipose, and Arvind Krishnamurthy. 2023. Symphony: Optimized model serving using centralized orchestration. *arXiv preprint* (2023).
[16] Lequn Chen, Zihao Ye, Yongji Wu, Danyang Zhuo, Luis Ceze, and Arvind Krishnamurthy. 2023. Punica: Multi-tenant LoRA serving. *arXiv preprint* (2023).
[17] Zhaoyun Chen. 2021. RIFLING: A reinforcement learning-based GPU scheduler for deep learning research and development platforms. *Softw.: Pract. Exper.* 52, 6 (2021).
[18] 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. *Trans. Parallel Distrib. Syst.* 31, 1 (2020).
[19] Ping Chi, Shuangchen Li, Cong Xu, Tao Zhang, Jishen Zhao, Yongpan Liu, Yu Wang, and Yuan Xie. 2016. PRIME: A novel processing-in-memory architecture for neural network computation in ReRAM-based main memory. In *ISCA'16*.
[20] Min-Chi Chiang and Jerry Chou. 2021. DynamoML: Dynamic resource management operators for machine learning workloads. In *CLOSER'21*.
[21] Seungbeom Choi, Sunho Lee, Yeonjae Kim, Jongse Park, Youngjin Kwon, and Jaehyuk Huh. 2022. Serving heterogeneous machine learning models on Multi-GPU servers with spatio-temporal sharing. In *USENIX ATC'22*.
[22] Mosharaf Chowdhury and Ion Stoica. 2015. Efficient coflow scheduling without prior knowledge. *ACM SIGCOMM Comput. Commun. Rev.* (2015).
[23] 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 *SoCC'20*.
[24] Daniel Crankshaw, Xin Wang, Guilio Zhou, Michael J. Franklin, Joseph E. Gonzalez, and Ion Stoica. 2017. Clipper: A low-latency online prediction serving system. *(NSDI'17)*.
[25] 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. *(ICCD'19)*.
[26] 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 *SC'21*.
[27] 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. *(CLOUD'19)*.
[28] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *NAACL'19*.
[29] Aditya Dhakal, Sameer G. Kulkarni, and K. K. Ramakrishnan. 2020. GSLICE: Controlled spatial sharing of GPUs for a scalable inference platform. In *SoCC'20*.
[30] Lisa Dunlap, Kirthevasan Kandasamy, Ujval Misra, Richard Liaw, Michael Jordan, Ion Stoica, and Joseph E. Gonzalez. 2021. Elastic hyperparameter tuning on the cloud. In *SoCC'21*.
[31] Dick H. J. Epema. 1995. An analysis of decay-usage scheduling in multiprocessors. *ACM SIGMETRICS Perform. Eval. Rev.* (1995).
[32] Dror G. Feitelson. 1996. Packing schemes for gang scheduling. In *Job Scheduling Strategies for Parallel Processing*.
[33] Dror G. Feitelson. 1997. Job scheduling in multiprogrammed parallel systems. *IBM Res. Rep.* (1997).
[34] Dror G. Feitelson and Larry Rudolph. 1995. Parallel job scheduling: Issues and approaches. In *Workshop on Job Scheduling Strategies for Parallel Processing*.
[35] 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*.
[36] 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 *FiCloud'21*.
[37] Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. *(ICML'17)*.
[38] Mingyu Gao, Jing Pu, Xuan Yang, Mark Horowitz, and Christos Kozyrakis. 2017. TETRIS: Scalable and efficient neural network acceleration with 3D memory. In *ASPLOS'17*.
[39] Pin Gao, Lingfan Yu, Yongwei Wu, and Jinyang Li. 2018. Low latency RNN inference with cellular batching. In *EuroSys'18*.
[40] Wei Gao, Peng Sun, Yonggang Wen, and Tianwei Zhang. 2022. Titan: A scheduler for foundation model fine-tuning workloads. In *SoCC'22*.

[41] Wei Gao, Zhisheng Ye, Peng Sun, Yonggang Wen, and Tianwei Zhang. 2021. Chronus: A novel deadline-aware scheduler for deep learning training jobs. In *SoCC'21*.
[42] Ali Ghodsi, Matei Zaharia, Benjamin Hindman, Andy Konwinski, Scott Shenker, and Ion Stoica. 2011. Dominant resource fairness: Fair allocation of multiple resource types. In *NSDI'11*.
[43] 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. *arXiv preprint* (2021).
[44] Guin R. Gilman, Samuel S. Ogden, Robert J. Walls, and Tian Guo. 2019. Challenges and opportunities of DNN model execution caching. In *DIDL'19*.
[45] Diandian Gu, Xintong Xie, Gang Huang, Xin Jin, and Xuanzhe Liu. 2023. Energy-efficient GPU clusters scheduling for deep learning. *arXiv preprint* (2023).
[46] Diandian Gu, Yihao Zhao, Yinmin Zhong, Yifan Xiong, Zhenhua Han, Peng Cheng, Fan Yang, Gang Huang, Xin Jin, and Xuanzhe Liu. 2023. ElasticFlow: An elastic serverless training platform for distributed deep learning. In *ASPLOS'23*.
[47] Juncheng Gu, Mosharaf Chowdhury, Kang G. Shin, Yibo Zhu, Myeongjae Jeon, Junjie Qian, Hongqiang Liu, and Chuanxiong Guo. 2019. Tiresias: A GPU cluster manager for distributed deep learning. In *NSDI'19*.
[48] 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. *Trans. Parallel Distrib. Syst.* 33, 11 (2021).
[49] Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. 2017. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *IEEE International Conference on Robotics and Automation (ICRA)*.
[50] Joao Guerreiro, Aleksandar Ilic, Nuno Roma, and Pedro Tomas. 2018. GPGPU power modeling for multi-domain voltage-frequency scaling. In *HPCA'18*.
[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. *(OSDI'20)*.
[52] Jashwant Raj Gunasekaran, Cyan Subhra Mishra, Prashanth Thinakaran, Bikash Sharma, Mahmut Taylan Kandemir, and Chita R. Das. 2022. Cocktail: A multidimensional optimization for model serving in cloud. *(NSDI'22)*.
[53] 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. *(ISCA'20)*.
[54] 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)*.
[55] 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 *CCGRID'20*.
[56] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A. Horowitz, and William J. Dally. 2016. EIE: Efficient inference engine on compressed deep neural network. In *ISCA'16*.
[57] Zhenhua Han, Haisheng Tan, Shaofeng H.-C. Jiang, Xiaoming Fu, Wanli Cao, and Francis C. M. Lau. 2020. Scheduling placement-sensitive BSP jobs with inaccurate execution time estimation. In *INFOCOM'20*.
[58] Aaron Harlap, Andrew Chung, Alexey Tumanov, Gregory R. Ganger, and Phillip B. Gibbons. 2018. Tributary: Spot-dancing for elastic services with latency SLOs. In *USENIX ATC'18*.
[59] Aaron Harlap, Alexey Tumanov, Andrew Chung, Gregory R. Ganger, and Phillip B. Gibbons. 2017. Proteus: Agile ML elasticity through tiered reliability in dynamic resource markets. *(EuroSys'17)*.
[60] Kim Hazelwood, Sarah Bird, David Brooks, Soumith Chintala, Utku Diril, Dmytro Dzhulgakov, Mohamed Fawzy, Bill Jia, Yangqing Jia, Aditya Kalro, James Law, Kevin Lee, Jason Lu, Pieter Noordhuis, Misha Smelyanskiy, Liang Xiong, and Xiaodong Wang. 2018. Applied machine learning at facebook: A datacenter infrastructure perspective. *(HPCA'18)*.
[61] 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 *NSDI'11*.
[62] Connor Holmes, Daniel Mawhirter, Yuxiong He, Feng Yan, and Bo Wu. 2019. GRNN: Low-latency and scalable RNN inference on GPUs. In *EuroSys'19*.
[63] Bodun Hu, Le Xu, Jeongyoon Moon, Neeraja J. Yadwadkar, and Aditya Akella. 2023. MOSEL: Inference serving using dynamic modality selection. *arXiv preprint* (2023).
[64] Edward J. Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *ICLR'22*.
[65] 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 *SC'21*.

[66] Qinghao Hu, Zhisheng Ye, Zerui Wang, Guoteng Wang, Meng Zhang, Qiaoling Chen, Peng Sun, Dahua Lin, Xiaolin Wang, Yingwei Luo, Yonggang Wen, and Tianwei Zhang. 2024. Characterization of large language model development in the datacenter. In *NSDI'24*.
[67] Qinghao Hu, Zhisheng Ye, Meng Zhang, Qiaoling Chen, Peng Sun, Yonggang Wen, and Tianwei Zhang. 2023. Hydro: Surrogate-based hyperparameter tuning service in datacenters. In *OSDI'23*.
[68] Qinghao Hu, Meng Zhang, Peng Sun, Yonggang Wen, and Tianwei Zhang. 2023. Lucid: A non-intrusive, scalable and interpretable scheduler for deep learning training jobs. In *ASPLOS'23*.
[69] Gadi Hutt, Vibhav Viswanathan, and Adam Nadolski. 2019. Deliver high performance ML inference with AWS Inferentia. (2019).
[70] Changho Hwang, Taehyun Kim, Sunghyun Kim, Jinwoo Shin, and KyoungSoo Park. 2021. Elastic resource sharing for distributed deep learning. In *NSDI'21*.
[71] Ranggi Hwang, Taehun Kim, Youngeun Kwon, and Minsoo Rhu. 2020. Centaur: A chiplet-based, hybrid sparse-dense accelerator for personalized recommendations. In *ISCA'20*.
[72] Vatche Ishakian, Vinod Muthusamy, and Aleksander Slominski. 2018. Serving deep learning models in a serverless platform. *(IC2E'18)*.
[73] 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 *ICCCS'18*.
[74] Animesh Jain, Amar Phanishayee, Jason Mars, Lingjia Tang, and Gennady Pekhimenko. 2018. Gist: Efficient data encoding for deep neural network training. In *ISCA'18*.
[75] Paras Jain, Ajay Jain, Aniruddha Nrusimha, Amir Gholami, Pieter Abbeel, Joseph Gonzalez, Kurt Keutzer, and Ion Stoica. 2020. Checkmate: Breaking the memory wall with optimal tensor rematerialization. In *MLSys'20*.
[76] 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).
[77] Jananie Jarachanthan, Li Chen, Fei Xu, and Bo Li. 2021. AMPS-Inf: Automatic model partitioning for serverless inference with cost efficiency. In *ICPP 2021*.
[78] 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 *Middleware'19*.
[79] Suhas Jayaram Subramanya, Daiyaan Arfeen, Shouxu Lin, Aurick Qiao, Zhihao Jia, and Gregory R. Ganger. 2023. Sia: Heterogeneity-aware, goodput-optimized ML-cluster scheduling. In *SOSP'23*.
[80] Myeongjae Jeon, Shivaram Venkataraman, Amar Phanishayee, Junjie Qian, Wencong Xiao, and Fan Yang. 2019. Analysis of large-scale multi-tenant GPU clusters for DNN training workloads. In *USENIX ATC'19*.
[81] Jinwoo Jeong, Seungsu Baek, and Jeongseob Ahn. 2023. Fast and efficient model serving using multi-GPUs with direct-host-access. In *EuroSys'23*.
[82] Wenqi Jiang, Zhenhao He, Shuai Zhang, Thomas B. Preußer, 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. *(MLSys'21)*.
[83] 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 *KDD'21*.
[84] 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 *OSDI'20*.
[85] 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.* 14, 2 (2020).
[86] Sejin Kim and Yoonhee Kim. 2020. Co-scheML: Interference-aware container co-scheduling scheme using machine learning application profiles for GPU clusters. In *CLUSTER'20*.
[87] Jack Kosaian, K. V. Rashmi, and Shivaram Venkataraman. 2019. Parity models: Erasure-coded resilience for prediction serving systems. In *SOSP'19*.
[88] Peter Kraft, Daniel Kang, Deepak Narayanan, Shoumik Palkar, Peter Bailis, and Matei Zaharia. 2020. Willump: A statistically-aware end-to-end optimizer for machine learning inference. In *Proceedings of Machine Learning and Systems*.
[89] 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*.
[90] Ravi Kumar, Manish Purohit, Zoya Svitkina, Erik Vee, and Joshua Wang. 2019. Efficient rematerialization for deep networks. In *NeurIPS'19*.

[91] Fan Lai, Yinwei Dai, Harsha V. Madhyastha, and Mosharaf Chowdhury. 2023. ModelKeeper: Accelerating DNN training via automated training warmup. In *NSDI'23*.
[92] Tan N. Le, Xiao Sun, Mosharaf Chowdhury, and Zhenhua Liu. 2020. AlloX: Compute allocation in hybrid clusters. In *EuroSys'20*.
[93] 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 *SOSP'19*.
[94] Sukhan Lee, Shin-haeng Kang, Jaehoon Lee, Hyeonsu Kim, Eojin Lee, Seungwoo Seo, Hosang Yoon, Seungwon Lee, Kyounghwan Lim, Hyunsung Shin, Jinhyun Kim, O. Seongil, Anand Iyer, David Wang, Kyomin Sohn, and Nam Sung Kim. 2021. Hardware architecture and software stack for PIM based on commercial DRAM technology: Industrial product. In *ISCA'21*.
[95] 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 *OSDI'18*.
[96] Matthew LeMay, Shijian Li, and Tian Guo. 2020. PERSEUS: Characterizing performance and cost of multi-tenant serving for CNN models. *(IC2E'20)*.
[97] Baolin Li, Siddharth Samsi, Vijay Gadepally, and Devesh Tiwari. 2023. Clover: Toward sustainable AI with carbon-aware machine learning inference service. In *SC'23*.
[98] Baolin Li, Siddharth Samsi, Vijay Gadepally, and Devesh Tiwari. 2023. Kairos: Building cost-efficient machine learning inference systems with heterogeneous cloud resources. In *HPDC'23*.
[99] Jiamin Li, Hong Xu, Yibo Zhu, Zherui Liu, Chuanxiong Guo, and Cong Wang. 2023. Lyra: Elastic scheduling for deep learning clusters. In *EuroSys'23*.
[100] KeDi Li and Ning Gui. 2020. CMS: A continuous machine-learning and serving platform for industrial big data. *Fut. Internet* 12, 6 (2020).
[101] Mingzhen Li, Wencong Xiao, Hailong Yang, Biao Sun, Hanyu Zhao, Shiru Ren, Zhongzhi Luan, Xianyan Jia, Yi Liu, Yong Li, Wei Lin, and Depei Qian. 2023. EasyScale: Elastic training with consistent accuracy and improved utilization on GPUs. In *SC'23*.
[102] 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.* 11, 5 (2018).
[103] Wenxin Li, Sheng Chen, Keqiu Li, Heng Qi, Renhai Xu, and Song Zhang. 2020. Efficient online scheduling for coflow-aware machine learning clusters. In *TCC'20*.
[104] 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 *INFOCOM'20*.
[105] Zhuohan Li, Lianmin Zheng, Yinmin Zhong, Vincent Liu, Ying Sheng, Xin Jin, Yanping Huang, Zhifeng Chen, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. AlpaServe: Statistical multiplexing with model parallelism for deep learning serving. In *OSDI'23*.
[106] Tailin Liang, John Glossner, Lei Wang, Shaobo Shi, and Xiaotong Zhang. 2021. Pruning and quantization for deep neural network acceleration: A survey. *Neurocomputing* 461 (2021).
[107] 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 *SoCC'19*.
[108] 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'19*.
[109] 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 *KDD'21*.
[110] Yunteng Luan, Xukun Chen, Hanyu Zhao, Zhi Yang, and Yafei Dai. 2019. SCHED$^2$: Scheduling deep learning training via deep reinforcement learning. In *GLOBECOM'19*.
[111] Jose Luis Lucas-Simarro, Rafael Moreno-Vozmediano, Ruben S. Montero, and Ignacio M. Llorente. 2013. Scheduling strategies for optimal service deployment across multiple clouds. *Fut. Gen. Comput. Syst.* 29, 6 (2013).
[112] Tao Luo, Mingen Pan, Pierre Tholoniat, Asaf Cidon, Roxana Geambasu, and Mathias Lécuyer. 2021. Privacy budget scheduling. In *OSDI'21*.
[113] Kshiteej Mahajan, Arjun Balasubramanian, Arjun Singhvi, Shivaram Venkataraman, Aditya Akella, Amar Phanishayee, and Shuchi Chawla. 2020. Themis: Fair and efficient GPU cluster scheduling. In *NSDI'20*.
[114] Ruben Mayer and Hans-Arno Jacobsen. 2021. Scalable deep learning on distributed infrastructures: Challenges, techniques, and tools. *Comput. Surv.* 53, 1 (2021).
[115] Xinxin Mei, Qiang Wang, Xiaowen Chu, Hai Liu, Yiu-Wing Leung, and Zongpeng Li. 2021. Energy-aware task scheduling with deadline constraint in DVSF-enabled heterogeneous clusters. *arXiv preprint* (2021).
[116] Daniel Mendoza, Francisco Romero, Qian Li, Neeraja J. Yadwadkar, and Christos Kozyrakis. 2021. Interference-aware scheduling for inference serving. In *EuroMLSys'21*.

[117] Xupeng Miao, Chunan Shi, Jiangfei Duan, Xiaoli Xi, Dahua Lin, Bin Cui, and Zhihao Jia. 2024. SpotServe: Serving generative large language models on preemptible instances. In *ASPLOS'24*.
[118] 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 *EuroSys'21*.
[119] Jayashree Mohan, Amar Phanishayee, Janardhan Kulkarni, and Vijay Chidambaram. 2022. Synergy: Looking beyond GPUs for DNN scheduling on multi-tenant clusters. In *OSDI'22*.
[120] A. W. Mu'alem and D. G. Feitelson. 2001. Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling. *Trans. Parallel Distrib. Syst.* 12, 6 (2001).
[121] 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 *SOSP'21*.
[122] 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*.
[123] Deepak Narayanan, Keshav Santhanam, Fiodar Kazhamiaka, Amar Phanishayee, and Matei Zaharia. 2020. Heterogeneity-aware cluster scheduling policies for deep learning workloads. In *OSDI'20*.
[124] 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*.
[125] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, Amar Phanishayee, and Matei Zaharia. 2021. Efficient large-scale language model training on GPU clusters using megatron-LM. In *SC'21*.
[126] 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. Surv.* 51, 1 (2018).
[127] Samuel S. Ogden, Xiangnan Kong, and Tian Guo. 2021. PieSlicer: Dynamically improving response time for cloud-based CNN inference. In *ICPE'21*.
[128] 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. *arXiv preprint* (2017).
[129] Shuo Ouyang, Dezun Dong, Yemao Xu, and Liquan Xiao. 2021. Communication optimization strategies for distributed deep neural network training: A survey. *J. Parallel Distrib. Comput.* 34, 12 (2021).
[130] 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. *arXiv preprint* (2018).
[131] Jun Woo Park, Alexey Tumanov, Angela Jiang, Michael A. Kozuch, and Gregory R. Ganger. 2018. 3Sigma: Distribution-based cluster scheduling for runtime uncertainty. In *EuroSys'18*.
[132] Yanghua Peng, Yixin Bao, Yangrui Chen, Chuan Wu, and Chuanxiong Guo. 2018. Optimus: An efficient dynamic resource scheduler for deep learning clusters. In *EuroSys'18*.
[133] Yanghua Peng, Yixin Bao, Yangrui Chen, Chuan Wu, Chen Meng, and Wei Lin. 2021. DL2: A deep learning-driven scheduler for deep learning clusters. *Trans. Parallel Distrib. Syst.* 32, 8 (2021).
[134] Aurick Qiao, Sang Keun Choe, Suhas Jayaram Subramanya, Willie Neiswanger, Qirong Ho, Hao Zhang, Gregory R. Ganger, and Eric P. Xing. 2021. Pollux: Co-adaptive cluster scheduling for goodput-optimized deep learning. In *OSDI'21*.
[135] 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 *SC'19*.
[136] Sudarsanan Rajasekaran, Manya Ghobadi, and Aditya Akella. 2024. Cassini: Network-aware job scheduling in machine learning clusters. In *NSDI'24*.
[137] Jeff Rasley, Yuxiong He, Feng Yan, Olatunji Ruwase, and Rodrigo Fonseca. 2017. HyperDrive: Exploring hyperparameters with POP scheduling. In *Middleware'17*.
[138] Charles Reiss, Alexey Tumanov, Gregory R. Ganger, Randy H. Katz, and Michael A. Kozuch. 2012. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. *(SoCC'12)*.
[139] 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 Distrib. Comput.* 111 (2018).
[140] Minsoo Rhu, Natalia Gimelshein, Jason Clemons, Arslan Zulfiqar, and Stephen W. Keckler. 2016. vDNN: Virtualized deep neural networks for scalable, memory-efficient neural network design. In *MICRO'16*.

[141] Francisco Romero, Qian Li, Neeraja J. Yadwadkar, and Christos Kozyrakis. 2021. INFaaS: Automated model-less inference serving. In *USENIX ATC'21*.

[142] Vaibhav Saxena, K. R. Jayaram, Saurav Basu, Yogish Sabharwal, and Ashish Verma. 2020. Effective elastic scaling of deep learning workloads. In *MASCOTS'20*.

[143] 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.* 18, 4 (2021).

[144] Alexander Sergeev and Mike Del Balso. 2018. Horovod: Fast and easy distributed deep learning in TensorFlow. *arXiv preprint* (2018).

[145] Dinggang Shen, Guorong Wu, and Heung-Il Suk. 2017. Deep learning in medical image analysis. *Ann. Rev. Biom. Eng.* 19 (2017).

[146] 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 *SOSP'19*.

[147] Ying Sheng, Shiyi Cao, Dacheng Li, Coleman Hooper, Nicholas Lee, Shuo Yang, Christopher Chou, Banghua Zhu, Lianmin Zheng, Kurt Keutzer, Joseph E. Gonzalez, and Ion Stoica. 2023. S-LoRA: Serving thousands of concurrent LoRA adapters. *arXiv preprint* (2023).

[148] Lin Shi, Hao Chen, Jianhua Sun, and Kenli Li. 2012. vCUDA: GPU-accelerated high-performance computing in virtual machines. In *TC'12*.

[149] 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*.

[150] Dharma Shukla, Muthian Sivathanu, Srinidhi Viswanatha, Bhargav Gulavani, Rimma Nehme, Amey Agrawal, Chen Chen, Nipun Kwatra, Ramachandran Ramjee, Pankaj Sharma, Atul Katiyar, Vipul Modi, Vaibhav Sharma, Abhishek Singh, Shreshth Singhal, Kaustubh Welankar, Lu Xun, Ravi Anupindi, Karthik Elangovan, Hasibur Rahman, Zhou Lin, Rahul Seetharaman, Cheng Xu, Eddie Ailijiang, Suresh Krishnappa, and Mark Russinovich. 2022. Singularity: Planet-scale, preemptible, elastic scheduling of AI workloads. *arXiv preprint* (2022).

[151] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Hui Fan, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. 2017. Mastering the game of go without human knowledge. *Nature* 550, 7676 (2017).

[152] 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* 10, 3 (2021).

[153] 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 *ICPP'20*.

[154] Peng Sun, Yonggang Wen, Nguyen Binh Duong Ta, and Shengen Yan. 2017. Towards distributed machine learning in shared clusters: A dynamically-partitioned approach. In *SMARTCOMP'17*.

[155] Nguyen Binh Duong Ta. 2019. FC2: Cloud-based cluster provisioning for distributed machine learning. *Cluster Comput.* 22 (2019).

[156] 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. *arXiv preprint* (2021).

[157] 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)*.

[158] 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 *CLUSTER'19*.

[159] Alexey Tumanov, Timothy Zhu, Jun Woo Park, Michael A. Kozuch, Mor Harchol-Balter, and Gregory R. Ganger. 2016. TetriSched: Global rescheduling with adaptive plan-ahead in dynamic heterogeneous clusters. In *EuroSys'16*.

[160] 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 *SoCC'13*.

[161] Joost Verbraeken, Matthijs Wolting, Jonathan Katzy, Jeroen Kloppenburg, Tim Verbelen, and Jan S. Rellermeyer. 2020. A survey on distributed machine learning. *Comput. Surv.* 53, 2 (2020).

[162] Haoyu Wang, Zetian Liu, and Haiying Shen. 2020. Job scheduling for large-scale machine learning clusters. In *CoNEXT'20*.

[163] 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 *SoCC'21*.

[164] 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 *IISWC'19*.
[165] Qiang Wang and Xiaowen Chu. 2020. GPGPU performance estimation with core and memory frequency scaling. *Trans. Parallel Distrib. Syst.* 31, 12 (2020).
[166] Qiang Wang, Shaohuai Shi, Canhui Wang, and Xiaowen Chu. 2020. Communication contention aware scheduling of multiple deep learning training jobs. *arXiv preprint* (2020).
[167] 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 *SC'20*.
[168] Shang Wang, Peiming Yang, Yuxuan Zheng, Xin Li, and Gennady Pekhimenko. 2021. Horizontally fused training array: An effective hardware utilization squeezer for training novel deep learning models. In *MLSys'21*.
[169] 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. 12, 2 (2018).
[170] Yiding Wang, Kai Chen, Haisheng Tan, and Kun Guo. 2023. Tabi: An efficient multi-level inference system for large language models. In *EuroSys'23*.
[171] 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 *NSDI'22*.
[172] Qizhen Weng, Lingyun Yang, Yinghao Yu, Wei Wang, Xiaochuan Tang, Guodong Yang, and Liping Zhang. 2023. Beware of fragmentation: Scheduling GPU-sharing workloads with fragmentation gradient descent. In *USENIX ATC'23*.
[173] Xiaorui Wu, Hong Xu, and Yi Wang. 2020. Irina: Accelerating DNN inference with efficient online scheduling. In *APNet'20*.
[174] 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. *Trans. Parallel Distrib. Syst.* 33, 1 (2022).
[175] 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. *(OSDI'18)*.
[176] 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 *OSDI'20*.
[177] 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 *ICDCS'20*.
[178] Fei Xu, Jianian Xu, Jiabin Chen, Li Chen, Ruitao Shang, Zhi Zhou, and Fangming Liu. 2023. iGniter: Interference-aware GPU resource provisioning for predictable DNN inference in the cloud. *IEEE Trans. Parallel Distrib. Syst.* 34, 3 (2023).
[179] Neeraja J. Yadwadkar, Francisco Romero, Qian Li, and Christos Kozyrakis. 2019. A case for managed and model-less inference serving. In *HotOS'19*.
[180] Feng Yan, Olatunji Ruwase, Yuxiong He, and Evgenia Smirni. 2016. SERF: Efficient scheduling for fast deep neural network serving via judicious parallelism. *(SC'16)*.
[181] Ge Yang, Edward Hu, Igor Babuschkin, Szymon Sidor, Xiaodong Liu, David Farhi, Nick Ryder, Jakub Pachocki, Weizhu Chen, and Jianfeng Gao. 2021. Tuning large neural networks via zero-shot hyperparameter transfer. In *NeurIPS'21*.
[182] Zehua Yang, Zhisheng Ye, Tianhao Fu, Jing Luo, Xiong Wei, Yingwei Luo, Xiaolin Wang, Zhenlin Wang, and Tianwei Zhang. 2022. Tear up the bubble boom: Lessons learned from a deep learning research and development cluster. In *ICCD'22*.
[183] Xiaozhe Yao and Ana Klimovic. 2023. DeltaZip: Multi-tenant language model serving via delta compression. *arXiv preprint* (2023).
[184] Zhisheng Ye, Peng Sun, Wei Gao, Tianwei Zhang, Xiaolin Wang, Shengen Yan, and Yingwei Luo. 2022. ASTRAEA: A fair deep learning scheduler for multi-tenant GPU clusters. *Trans. Parallel Distrib. Syst.* 33, 11 (2022).
[185] 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 *HPDC'20*.
[186] 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*.
[187] 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. *Trans. Parallel Distrib. Syst.* 33, 1 (2022).
[188] 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 *CoNext'20*.

[189] Andy B. Yoo, Morris A. Jette, and Mark Grondona. 2003. SLURM: Simple Linux utility for resource management. In *Job Scheduling Strategies for Parallel Processing*.
[190] Fuxun Yu, Di Wang, Longfei Shangguan, Minjia Zhang, Chenchen Liu, and Xiang Chen. 2022. A survey of multi-tenant deep learning inference on GPU. *arXiv preprint* (2022).
[191] 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. *arXiv preprint* (2021).
[192] 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. *(ICDCS'21)*.
[193] 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).
[194] Menglu Yu, Chuan Wu, Bo Ji, and Jia Liu. 2021. A sum-of-ratios multi-dimensional-knapsack decomposition for DNN resource scheduling. In *INFOCOM'21*.
[195] Peifeng Yu and Mosharaf Chowdhury. 2020. Fine-grained GPU sharing primitives for deep learning applications. In *MLSys'20*.
[196] Peifeng Yu, Jiachen Liu, and Mosharaf Chowdhury. 2021. Fluid: Resource-aware hyperparameter tuning engine. In *MLSys'21*.
[197] 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. *(EuroSys'10)*.
[198] 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. Surv.* 47, 4 (2015).
[199] Chengliang Zhang, Minchen Yu, Wei Wang, and Feng Yan. 2019. MArk: Exploiting cloud services for cost-effective, SLO-aware machine learning inference serving. *(ATC'19)*.
[200] Chengliang Zhang, Minchen Yu, Wei Wang, and Feng Yan. 2020. Enabling cost-effective, SLO-aware machine learning inference serving on public cloud. *TCC'20* (2020).
[201] 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 *MM'20*.
[202] Hong Zhang, Yupeng Tang, Anurag Khandelwal, and Ion Stoica. 2023. SHEPHERD: Serving DNNs in the wild. In *NSDI'23*.
[203] 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. *(HotCloud'20)*.
[204] 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)*.
[205] Quanlu Zhang, Zhenhua Han, Fan Yang, Yuge Zhang, Zhe Liu, Mao Yang, and Lidong Zhou. 2020. Retiarii: A deep learning exploratory-training framework. In *OSDI'20*.
[206] Qin Zhang, Ruiting Zhou, Chuan Wu, Lei Jiao, and Zongpeng Li. 2020. Online scheduling of heterogeneous distributed machine learning jobs. In *MOBIHOC'20*.
[207] 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. *(CCGrid'20)*.
[208] Yanwei Zhang, Yefu Wang, and Xiaorui Wang. 2011. GreenWare: Greening cloud-scale data centers to maximize the use of renewable energy. *(Middleware'11)*.
[209] 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 *ICDCS'20*.
[210] Hanyu Zhao, Zhenhua Han, Zhi Yang, Quanlu Zhang, Mingxia Li, Fan Yang, Qianxi Zhang, Binyang Li, Yuqing Yang, Lili Qiu, Lintao Zhang, and Lidong Zhou. 2023. SiloD: A co-design of caching and scheduling for deep learning clusters. In *EuroSys'23*.
[211] 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 *OSDI'20*.
[212] Yihao Zhao, Yuanqiang Liu, Yanghua Peng, Yibo Zhu, Xuanzhe Liu, and Xin Jin. 2022. Multi-resource interleaving for deep learning training. In *SIGCOMM'22*.
[213] Bojian Zheng, Nandita Vijaykumar, and Gennady Pekhimenko. 2020. Echo: Compiler-based GPU memory footprint reduction for LSTM RNN training. In *ISCA'20*.
[214] 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 *ICPP' 19*.
[215] Pengfei Zheng, Rui Pan, Tarannum Khan, Shivaram Venkataraman, and Aditya Akella. 2023. Shockwave: Fair and efficient cluster scheduling for dynamic adaptation in machine learning. In *NSDI'23*.
[216] Pan Zhou, Xinshu He, Shouxi Luo, Hongfang Yu, and Gang Sun. 2020. JPAS: Job-progress-aware flow scheduling for deep learning clusters. *J. Netw. Comput. Applic.* 158 (2020).

[217] 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. In *TCC'22*.

[218] Hongyu Zhu, Mohamed Akrout, Bojian Zheng, Andrew Pelegris, Anand Jayarajan, Amar Phanishayee, Bianca Schroeder, and Gennady Pekhimenko. 2018. Benchmarking and analyzing deep neural network training. In *IISWC'18*.

Received 24 May 2022; revised 11 February 2023; accepted 15 December 2023