# On Efficiency, Fairness and Security in AI Accelerator Resource Sharing: A Survey

JIAHUA HUANG, South China University of Technology, China and Pengcheng Laboratory, Guangzhou, China
WEIWEI LIN, South China University of Technology, China and Pengcheng Laboratory, Guangzhou, China
WENTAI WU, Department of Computer Science, Jinan University, Guangzhou, China
YANG WANG, Shenzhen Institutes of Advanced Technology Chinese Academy of Sciences, Shenzhen, China
HAOCHENG ZHONG, South China University of Technology, Guangzhou, China
XINHUA WANG, South China University of Technology, Guangzhou, China
KEQIN LI, Department of Computer Science, State University of New York, New Paltz, United States

The effective and efficient utilization of AI accelerators represents a critical issue for the practitioners engaged in the field of deep learning. Practical evidence from companies such as Alibaba, SenseTime, and Microsoft reveals that the utilization of production GPU clusters in the industry is generally between 25% and 50%. This indicates a significant opportunity for improvement. To this end, AI accelerator resource sharing has emerged as a promising approach to the performance optimization of multi-tenant clusters. This survey covers this line of studies from 2016 to 2024, focusing primarily on system efficiency while also including discussion on fairness, interference, and security in AI accelerator sharing. We revisit the fundamentals and key concepts, followed by a comprehensive review of recent advances in the field. We find that over 70% of the studies focus on efficiency improvement. We also observe that approximately half of the reviewed studies have made their source code publicly available, while fewer than one-third of the studies did not utilize a physical machine for experimentation. Finally, based on the limitations of existing research, we outline several directions for future research concerning the integration of sharing with large language models (LLMs), coordination between schedulers and application-layer metrics, and collaboration among heterogeneous accelerators.

CCS Concepts: • **General and reference** → **Surveys and overviews**; • **Computing methodologies** → **Massively parallel algorithms**; • **Computer systems organization** → **Multicore architectures**; • **Security and privacy** → **Privacy protections;**

This work is supported by Guangdong Major Project of Basic and Applied Basic Research (2019B030302002), National Natural Science Foundation of China (62402198), Guangxi Key Research and Development Project(2024AB02018), Guangzhou Development Zone Science and Technology Project (2023GH02), Fundamental Research Funds for the Central Universities (21624348), and in part by the Major Key Project of PCL (PCL2023A09).

Authors' Contact Information: Jiahua Huang, South China University of Technology, China and Pengcheng Laboratory, China; e-mail: ftjiah.huang@mail.scut.edu.cn; Weiwei Lin (Corresponding author), South China University of Technology, China and Pengcheng Laboratory, China; email: linww@scut.edu.cn; Wentai Wu, Department of Computer Science, Jinan University, Guangzhou, China; e-mail: wentaiwu@jnu.edu.cn; Yang Wang, Shenzhen Institutes of Advanced Technology Chinese Academy of Sciences, Shenzhen, China; e-mail: yang.wang1@siat.ac.cn; Haocheng Zhong, South China University of Technology, Guangzhou, Guangdong, China; e-mail: cshczhong@mail.scut.edu.cn; Xinhua Wang, South China University of Technology, Guangzhou, Guangdong, China; e-mail: 18340826607@163.com; Keqin Li, Department of Computer Science, State University of New York, New Paltz, New York, United States; e-mail: lik@newpaltz.edu.

ACM 0360-0300/2025/04-ART221
https://doi.org/10.1145/3721427

Additional Key Words and Phrases: AI accelerators, resource sharing, Artificial Intelligence, fairness, security

**ACM Reference Format:**
Jiahua Huang, Weiwei Lin, Wentai Wu, Yang Wang, Haocheng Zhong, Xinhua Wang, and Keqin Li. 2025. On Efficiency, Fairness and Security in AI Accelerator Resource Sharing: A Survey. *ACM Comput. Surv.* 57, 9, Article 221 (April 2025), 35 pages. https://doi.org/10.1145/3721427

## 1 Introduction

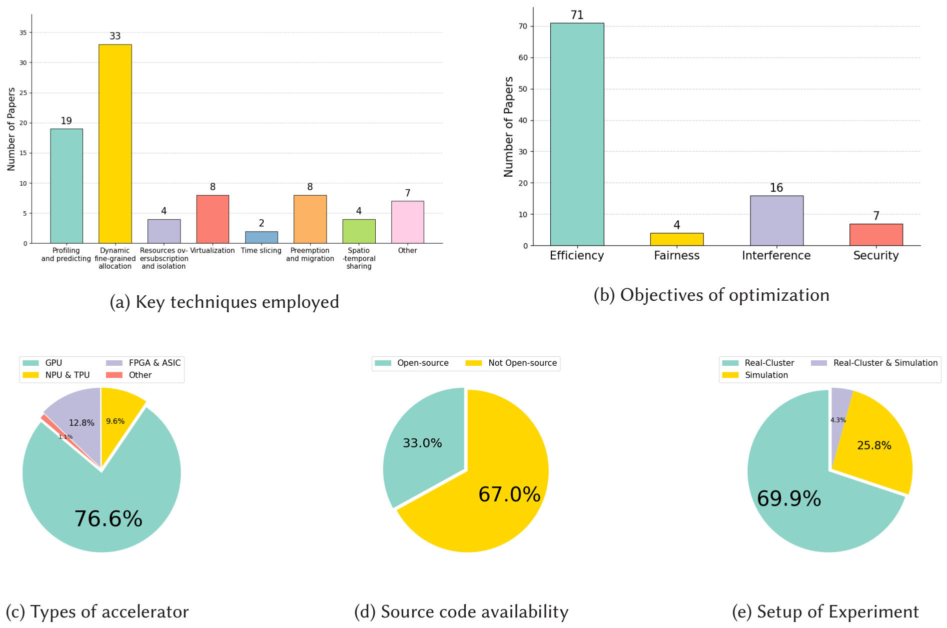
The rapid advancements in **artificial intelligence (AI)** and the growing complexity of deep learning models have led to an unprecedented demand for high-performance computing resources. AI accelerators, including **Graphics Processing Units (GPUs)** [41], **Tensor Processing Units (TPUs)** [52], and other custom AI chips, have emerged as critical components in datacenters designed to handle the intensive computational requirements of deep learning workloads.

AI accelerators are specifically engineered to provide significant performance improvements for deep learning tasks through massive parallelism and specialized hardware features. Despite their capabilities, managing and scheduling resources within datacenters equipped with AI accelerators remains a significant challenge. As demonstrated by data released by companies including Alibaba [115], SenseTime [42], and Microsoft [47], the utilization of production GPU clusters is typically below 50%, which suggests that there is still considerable room for improvement. The heterogeneity of these accelerators and the dynamic nature of deep learning workloads necessitate advanced resource management strategies to optimize performance, cost, and resource utilization.

Currently, resource sharing in AI accelerator-based datacenters is a new area of interest. Techniques such as static partitioning, dynamic resource allocation, virtualization, and multi-tenancy have been developed to address these challenges. Static partitioning involves dividing resources into fixed segments for different tasks. Dynamic resource allocation adjusts resources in real-time based on demand. Virtualization abstracts physical resources to create flexible and isolated environments for multiple workloads, while multi-tenancy allows multiple users or applications to share the same physical resources. Effective resource sharing strategies can significantly enhance the efficiency and flexibility of AI accelerator utilization, ensuring that resources are dynamically allocated to match the computational demands of deep learning applications. This not only maximizes resource utilization but also helps in maintaining **quality of service (QoS)** and reducing operational costs.

This survey aims to provide a comprehensive review of state-of-the-art technologies for resource sharing in datacenters equipped with AI accelerators. To the best of our knowledge, it is the first to specifically focus on accelerator sharing technologies in both research and production environments for datacenters that handle multiple types of workloads. Our main contributions are as follows:

— We provide an overview of recent works in the field, revisit the architectures of mainstream AI accelerators, and summarize the key concepts and fundamentals for accelerator resource sharing.
— We navigate the readers through the latest studies in the field by the different aims of system optimization including efficiency, fairness, interference, and security. This taxonomy reveals where the majority of interest is and where more effort should be made.
— We analyze the limitations of existing technologies, explore emerging trends, and propose future research directions to address the evolving needs of deep learning applications in AI-accelerated environments.



(a) Key techniques employed

(b) Objectives of optimization

(c) Types of accelerator

(d) Source code availability

(e) Setup of Experiment

Fig. 1. A statistic view of the studies surveyed in this article.

## 1.1 Overview of the Field

This survey covers the studies from 2016 to 2024 in relevance to accelerator sharing technology. We first present the statistics from different aspects in Figure 1. Our first observation is that most of recent studies consider efficiency improvement as their primary objective, indicating that the main role of sharing technology is to enhance the performance of AI accelerators. Therefore, our survey will concentrate on efficiency, categorizing different types of efficiency to explore the distinctions between various optimization methods. We also find that a dominating percentage of research in the field is conducted on GPUs, while other types of AI accelerator only account for <15% combined. Most of the studies experimented on real clusters and over 40% of them have source code publicly available (Figure 1(d)).

## 1.2 Existing Surveys

We compare our work with related surveys to provide a better understanding of our contributions to the community.

— Many previous surveys limit their attention to GPU sharing [40, 53]. In contrast, our work encompasses GPUs, TPUs, **Neural Processing Units (NPUs)**, and other custom AI chips, and we survey sharing technologies across several levels.

— Zhao et al. [141] surveyed commercial GPU architectures to support GPU multitasking. However, we include both software and hardware sharing approaches.

— Yu et al. [130] summarized the challenges and optimization opportunities for multi-tenant DL inference on a single GPU. But our survey considers both training and inference workloads.

— Liang et al. [66] surveyed GPU sharing technologies that apply various approaches and network bandwidth-sharing technologies operating at different granularity levels. However, this survey does not cover as much ground as our work does, as it primarily focuses on GPUs and largely overlooks other accelerators such as NPUs.

Table 1. A Comparison of Related Surveys

| Survey | Year | Including NPU or TPU | Including Training and Inference Workloads | Focus on Card-level Sharing | Comparison of Effects |
|---|---|---|---|---|---|
| [40] | 2017 | | ✓ | | |
| [53] | 2018 | | ✓ | | |
| [79] | 2020 | ✓ | ✓ | | |
| [141] | 2021 | | | ✓ | |
| [130] | 2022 | | | ✓ | |
| [128] | 2024 | ✓ | ✓ | | |
| [66] | 2024 | | ✓ | | |
| ours | - | ✓ | ✓ | ✓ | ✓ |

— There are surveys of GPU workload scheduling at the datacenter level [79, 128], while our primary focus is accelerator sharing at the node or device level.

The comparison of all related surveys is shown in Table 1. This survey focuses on recent advancements of sharing technologies in AI accelerators, with a particular emphasis on the optimization of resources and key performance indicators such as efficiency, fairness, interference, and security. Furthermore, we identify current trends, elucidate technological constraints, and propose avenues for future research in the domain of deep learning in AI-accelerated environments.

### 1.3 Article Organization

The structure of this article is organized as follows: Section 2 introduces the architecture of AI accelerators, performance measures of AI workloads and key concepts of sharing technology. The main body of this article is presented in Figure 2. Section 3 discusses various works that optimize the efficiency of the accelerators, which are categorized by training, inference, and mixed workloads. Section 4 examines research focused on fairness, interference, and security for AI accelerators. Section 5 highlights existing challenges, in addition to those addressed in the aforementioned sections. Section 6 concludes this survey article.

## 2 Background

As depicted in Figure 2, this section lays the ground for understanding AI accelerator sharing by focusing on three critical aspects. It begins with a detailed comparison of mainstream AI accelerators—GPUs, TPUs, NPUs, and edge-specific accelerators—highlighting their architectures and capabilities. Next, it defines key performance metrics relevant to AI workloads, such as throughput, latency, and utilization, providing a foundation for evaluating resource efficiency. Finally, it introduces essential concepts and principles of accelerator sharing, supported by clear visualizations of sharing mechanisms and resource allocation strategies, setting the stage for a deeper exploration of optimization approaches in subsequent sections.

### 2.1 Brief View of AI Accelerators

This section establishes the foundational knowledge required to understand AI accelerators and their significance in modern computing. It introduces the key concepts related to AI accelerator technologies. Table 2 provides a comparative analysis of mainstream accelerators, complementing the discussion in this section.

*2.1.1 Graphics Processing Unit.* GPUs, initially designed for rendering graphics, are now widely used for deep learning and other general-purpose computing tasks. They feature a large number of parallel processing units, making them well-suited for large-scale matrix operations and supporting various machine learning frameworks and algorithms. Notable products include NVIDIA's A100, V100, and Tesla series, as well as AMD's Radeon Instinct series.

As shown in Figure 3(a), the structure of a modern GPU features multiple **Graphics Processing Clusters (GPCs)**, each containing several **Streaming Multiprocessors (SMs)**. The SMs are

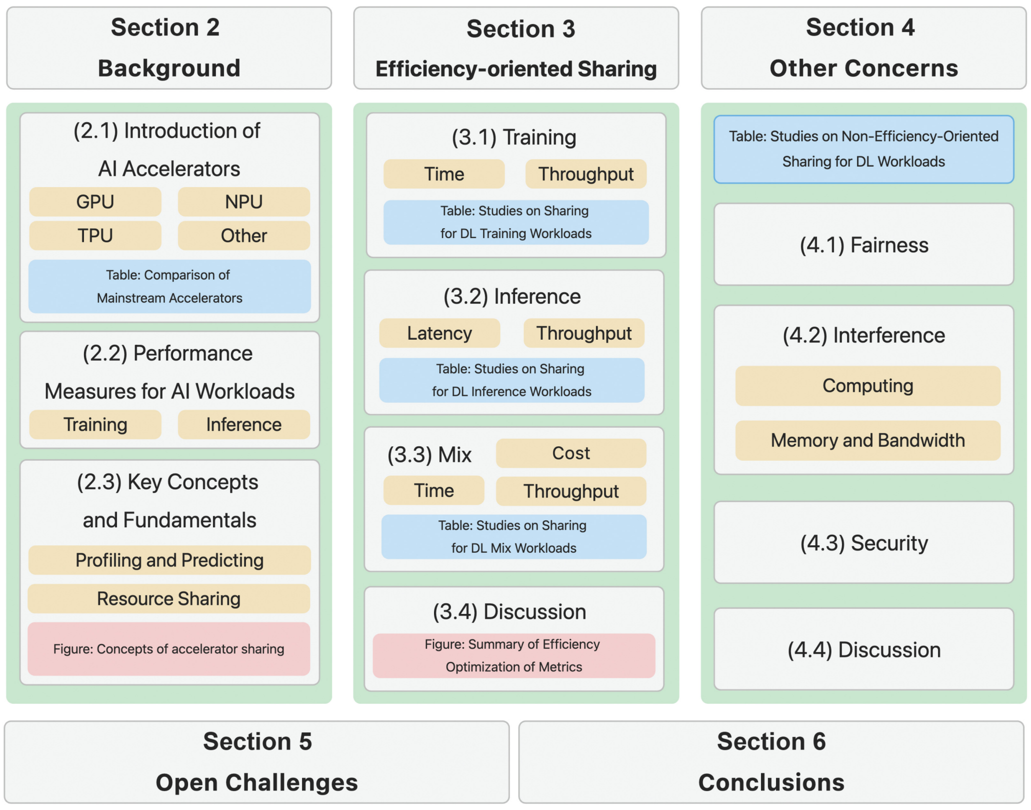

Section 2
Background
(2.1) Introduction of AI Accelerators
GPU
NPU
TPU
Other
Table: Comparison of Mainstream Accelerators
(2.2) Performance Measures for AI Workloads
Training
Inference
(2.3) Key Concepts and Fundamentals
Profiling and Predicting
Resource Sharing
Figure: Concepts of accelerator sharing
Section 3
Efficiency-oriented Sharing
(3.1) Training
Time
Throughput
Table: Studies on Sharing for DL Training Workloads
(3.2) Inference
Latency
Throughput
Table: Studies on Sharing for DL Inference Workloads
(3.3) Mix
Cost
Time
Throughput
Table: Studies on Sharing for DL Mix Workloads
(3.4) Discussion
Figure: Summary of Efficiency Optimization of Metrics
Section 4
Other Concerns
Table: Studies on Non-Efficiency-Oriented Sharing for DL Workloads
(4.1) Fairness
(4.2) Interference
Computing
Memory and Bandwidth
(4.3) Security
(4.4) Discussion
Section 5
Open Challenges
Section 6
Conclusions

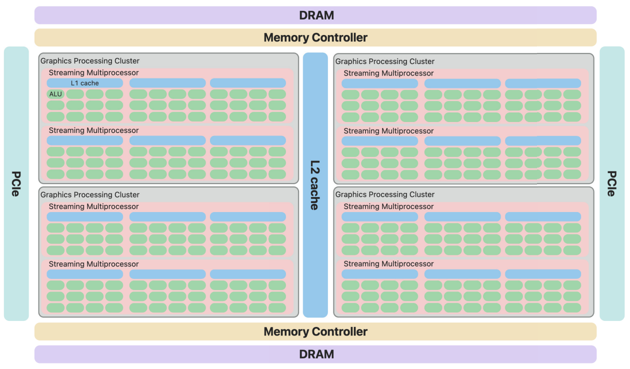
Fig. 2. Structure of this article.

Table 2. Comparison of Mainstream Accelerators

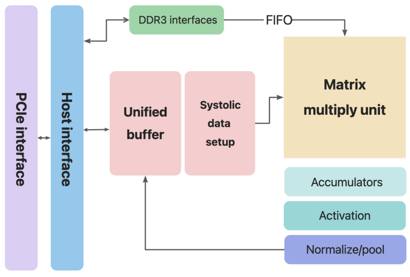
| **Dimension** | **GPU** | **TPU** | **NPU** | **Edge-specific Accelerators** |
|---|---|---|---|---|
| **Ecosystem Maturity** | Mature CUDA ecosystem, large community | TensorFlow-focused ecosystem | Growing ecosystem, limited tool support | Vendor-specific, fragmented |
| **Framework Compatibility** | PyTorch, TensorFlow, JAX | TensorFlow, Limited JAX | Limited PyTorch/TensorFlow | Vendor-specific only |
| **Precision Support** | FP32, FP16, INT8, FP8 | BF16, INT8 | FP16, INT8, INT4 | Primarily INT8, INT4 |
| **Memory Architecture** | HBM3, 4.8 TB/s | HBM2, 1,200 GB/s | HBM2e, 392 GB/s | Limited on-chip memory |
| **Key Metrics (Performance)** | 500–4,000 TOPS/s (H200), 350–700W | 275–420 TOPS/s (v4), 175–250W | 512 TOPS/s (910B), 160–400W | 200 TOPS/s (Intel Agilex 9), 10–120 W |

**BF16**: Brain Floating Point, 16-bit format optimized for deep learning.

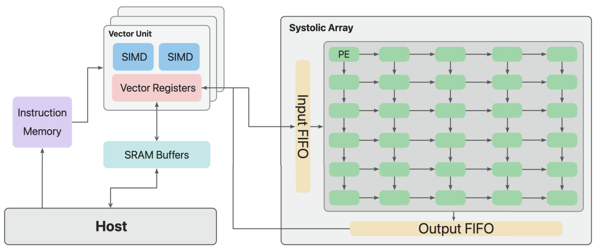
responsible for executing parallel computational tasks. Each SM has its own ALUs and L1 cache, enabling the performance of mathematical operations and rapid data access. The L2 cache is shared across the GPU, facilitating enhanced data access efficiency. The Memory Controller oversees communication with external DRAM, guaranteeing efficient data transfer, while the PCIe interface



(a) GPU Architecture



(b) TPU Architecture [52]



(c) NPU Architecture [124]

Fig. 3. Architecture of part AI accelerators.

connects the GPU to other components of the computer, allowing for data exchange between the CPU and GPU.

Defined by Nvidia [4], GPU utilization refers to the percentage of time that certain activities occur during the past sample period. This can be expressed as:

$$U_{GPU} = \frac{T_{active}}{T_{total}}, \tag{1}$$

where:

— $U_{GPU}$ is GPU utilization.
— $T_{active}$ is active time during which the GPU is performing computations within a given time slice.
— $T_{total}$ is total length of the slice.

*2.1.2 Neural Processing Unit.* NPUs have significantly advanced, offering increased computational power and energy efficiency for AI tasks. They are being integrated into **System-on-Chip (SoC)** designs for seamless AI processing and are widely deployed in edge devices such as smartphones and IoT gadgets for real-time, on-device AI computations. Leading implementations include Google's Edge TPU [2], Apple's Neural Engine [1], Huawei's Ascend [3], and Intel's Movidius Myriad X, all supporting a range of AI models with enhanced software and ecosystem support. The future of NPUs focuses on greater scalability, broader AI model compatibility, and improved developer tools.

The architecture of an NPU, similar to Figure 3(c), comprises a host for controlling operations, an instruction memory for storing execution commands, and a vector unit with SMID units and vector registers for parallel processing. The use of SRAM buffers facilitates rapid data storage, while the systolic array of **processing elements (PEs)** enables the efficient execution of matrix operations. Input and output FIFO buffers facilitate a seamless dataflow to and from the systolic array, thereby enabling high-throughput processing, which is well-suited to the demands of large-scale neural network tasks.

The utilization for NPUs can be defined as the ratio of used PE to the total available PE over a given time slice. The formula is:

$$U_{NPU} = \frac{PE_{used}}{PE_{total}}, \tag{2}$$

where:

— $U_{NPU}$ is NPU utilization.
— $PE_{used}$ is number of PEs actively used during a time slice.
— $PE_{total}$ is total number of PEs available in the NPU.

*2.1.3 Tensor Processing Unit.* TPU was developed by Google to accelerate machine learning workloads, particularly for deep learning applications. Introduced in 2016, TPUs [51, 52, 87] are designed to handle the demanding computational requirements of training and inference for large neural networks. They are optimized for Google's TensorFlow framework, enabling faster and more efficient execution of machine learning models.

Figure 3(b) depicts the architectural design of the TPU system, which has been optimized for high-performance deep learning operations. The system comprises a PCIe interface and a host interface for communication with the host system, as well as DDR3 interfaces for access to external memory. A **first-in, first-out (FIFO)** queue buffers data for the matrix multiply unit, which performs the core matrix computations. The data is organized by the systolic data setup, stored in a unified buffer, and processed through accumulators. The results then proceed through the activation and normalization/pooling stages, where non-linear functions and dimensionality reduction are applied, thus optimizing the TPU for fast and efficient deep learning tasks.

The utilization of TPU is often measured in terms of the matrix processing units; below are the formulas:

$$U_{TPU} = \frac{MXU_{used}}{MXU_{total}}, \tag{3}$$

where:

— $U_{TPU}$ is MXU utilization.
— $MXU_{used}$ is the number of active MXU cycles (or operations) during a time slice.
— $MXU_{total}$ is the total available MXU cycles (or operations) during the same time slice.

*2.1.4 Other AI Accelerators.* **Field Programmable Gate Arrays (FPGAs)** [76, 81, 101, 102, 135, 144] are reconfigurable hardware accelerators that offer high flexibility and parallel processing capabilities. While FPGAs are less common for large-scale AI training due to their programming complexity and the dominance of GPUs, they have shown increasing relevance in AI inference tasks, particularly in edge and embedded systems. Their ability to achieve low latency and power efficiency makes them well-suited for real-time inference scenarios. **Application Specific Integrated Circuits (ASICs)** [15, 17–19, 36, 39, 70, 89, 95, 140], designed for specialized tasks, deliver superior efficiency and performance through hardware-level optimization. While ASICs lack the programmability needed for diverse training workloads, their deterministic architecture and energy efficiency make them highly effective for inference. DSPs excel in digital signal processing with low latency and high throughput. Although DSPs are not widely utilized for AI training due to their limited support for large-scale matrix operations, they are employed in lightweight AI inference tasks, especially in applications requiring real-time processing and constrained computational resources. Hybrid accelerators combine multiple accelerator types to handle diverse workloads. However, they are not the primary choice for AI training, where specialized GPUs and TPUs prevail.

While GPUs and TPUs remain the predominant accelerators for AI training due to their programmability and computational throughput, accelerators such as FPGAs, ASICs, and DSPs demonstrate significant potential for inference, particularly in scenarios prioritizing energy efficiency, low latency, and real-time performance. Hybrid accelerators further broaden the design space for AI inference but remain underexplored for large-scale training.

## 2.2 Performance Measures for AI Workloads

In the realm of AI accelerators, workloads are primarily categorized into training and inference tasks. These two types of workloads exhibit distinct characteristics and demands, influencing how AI accelerators are designed and optimized. This section explores the specific objectives of training and inference workloads and highlights the key performance metrics for each. It is important to note that we will focus exclusively on metrics that can be optimized through scheduling technologies. To ensure comprehensive coverage of the literature, the metrics in this study are expressed in a more generalized form.

*2.2.1 Training.* Deep learning training tasks involve using large datasets to adjust the model parameters to minimize prediction errors. This process, known as training, requires substantial computational resources and time, because it involves both forward propagation and backpropagation. The model learns by iteratively updating its weights through multiple epochs until it converges to an optimal set of parameters.

These tasks are characterized by high computational intensity, extensive memory and storage requirements, and long duration. Key performance metrics for training workloads are shown below.

*Training Time.* Training time, the duration required to complete the entire deep learning model training process, is critically important for several reasons. The training time is defined in Equation (4). Additionally, optimizing training time reduces computational and labor costs, especially in cloud environments where resource usage is billed by time, thus saving significant expenses.

$$T = \frac{N \times E \times F}{P \times B \times \eta}, \tag{4}$$

where:

- $T$ is the total training time.
- $N$ is the number of samples in the dataset.
- $E$ is the total number of epochs (training iterations).
- $F$ is the computational cost per sample for forward and backward propagation, typically proportional to the model's complexity.
- $P$ is the hardware performance, measured in **floating-point operations per second (FLOPS)**. If sharing technologies involve sharing hardware resources (such as accelerators or processors), then this can impact hardware performance.
- $B$ is the batch size, which is the number of samples processed together in one iteration. Sharing data across nodes or devices could allow for larger batch sizes, which can improve computational efficiency.
- $\eta$ is the overall efficiency factor, accounting for I/O performance, memory bandwidth, parallel computation efficiency, and other overheads. By sharing data buffers and optimizing memory access patterns, sharing technologies can improve I/O performance and memory bandwidth utilization.

As data scales and model complexity increases, optimizing training time becomes essential for efficiently handling large datasets and developing complex models. Techniques such as efficient scheduling algorithms [43, 83, 112], distributed training [49, 64, 97], and resources sharing [69, 133, 143] can significantly reduce training times by improving resource utilization, preventing resource idleness, and ensuring proper task distribution and coordination among nodes.

*System Throughput.* System throughput, as defined in Equation (5), the rate at which a system processes training tasks or data samples, is crucial for deep learning training, as it accelerates model training, optimizes resource utilization, reduces costs, and effectively handles large-scale data and complex models. Efficient scheduling technologies, such as dynamic resource allocation [34, 105, 119] and parallelism [45, 80, 85, 86, 100] significantly enhance throughput. This leads to faster development cycles, lower operational costs, and better scalability, ultimately advancing deep learning capabilities and applications.

$$T_p = \frac{\sum_{i=1}^{n} R_i}{T}, \tag{5}$$

where:

— $T_p$ is the system throughput.
— $R_i$ is the number of tasks completed in the $i$th interval.
— $T$ is the total time taken to complete those tasks. Sharing resources for reducing synchronization overhead and communication delays can lead to faster task completion.

*2.2.2 Inference.* Deep learning inference tasks involve using a pre-trained model to make predictions on new, unseen data. The computational demands for inference are significantly lower than those for training, as inference only requires forward propagation through an already-optimized model. Inference tasks are designed for real-time or near-real-time prediction and are often deployed on edge devices or servers. The focus during inference is on reducing latency and enhancing efficiency, making it crucial to optimize for low power consumption and quick response times. Techniques such as model compression and hardware acceleration are commonly employed to ensure that the inference can run effectively in resource-constrained environments.

The primary characteristics of inference workloads include latency sensitivity, moderate computational resource requirements, and high concurrency. Key performance metrics for optimizing inference workloads are summarized below.

*Latency.* Latency, as defined in Equation (6), refers to the time taken from the moment an input is received by the system until the corresponding output is produced in the context of inference tasks. It is a critical performance metric for real-time and near-real-time applications. Low latency ensures quick, responsive interactions, timely and accurate decisions, higher customer satisfaction, and efficient resource utilization. Optimizing for low latency is essential for delivering high-performance, reliable AI solutions across various domains, enhancing both operational efficiency and market competitiveness. Low-latency systems can also handle a higher number of concurrent users or requests, making them more scalable. This is crucial for applications with high user traffic or those deployed in cloud environments where resources must be efficiently managed.

$$L = I + C + O, \tag{6}$$

where:

— $L$ is the total latency.
— $I$ is the input processing time, which includes data preprocessing and transfer to the accelerator. Techniques such as shared memory buffers and optimized data pipelines can significantly lower input processing time.

— $C$ is the computation time on the accelerator, which includes the forward pass through the neural network. Sharing accelerator resources can more effectively lead to better utilization of computational power, reducing the time required for the forward pass through the neural network.
— $O$ is the output processing time, which includes data post-processing and transfer back from the GPU.

Methods to optimize latency include using model compression technologies [8, 127], optimizing communication [71], and leveraging hardware accelerators [124, 136]. These strategies collectively help in achieving the low latency necessary for superior AI performance.

*System Throughput.* System throughput in the context of inference tasks refers to the number of inference requests or data samples the system can process in a given period. It measures the system's capacity to handle concurrent tasks and is crucial for evaluating the efficiency and scalability of AI applications. High throughput is essential for applications with a lot of users, such as online services, real-time analytics, and large-scale IoT deployments. While both training and inference benefit from high throughput, the optimization technologies and performance metrics differ.

The main approaches to improve system throughput of inference task include resources sharing [37, 46, 134], requests preemption [23] and Profiling [22].

*Power Consumption.* Power consumption, as defined in Equation (7), refers to the amount of electrical energy used by a system to perform AI workloads, including both training and inference tasks. Power consumption is a critical consideration for AI systems, especially in large-scale data centers, battery-powered edge devices, and energy-constrained environments. Efficient power usage leads to extended battery life, reduced operational costs, improved thermal management, and a smaller environmental footprint. By leveraging specialized hardware [25], edge computing [59, 122], dynamic power management [78], and software optimization [30, 113], it is possible to significantly reduce the power consumption of tasks, ensuring efficient and sustainable AI deployments across various environments.

$$P_c = \frac{E_t \times R_t}{\eta}, \tag{7}$$

where:

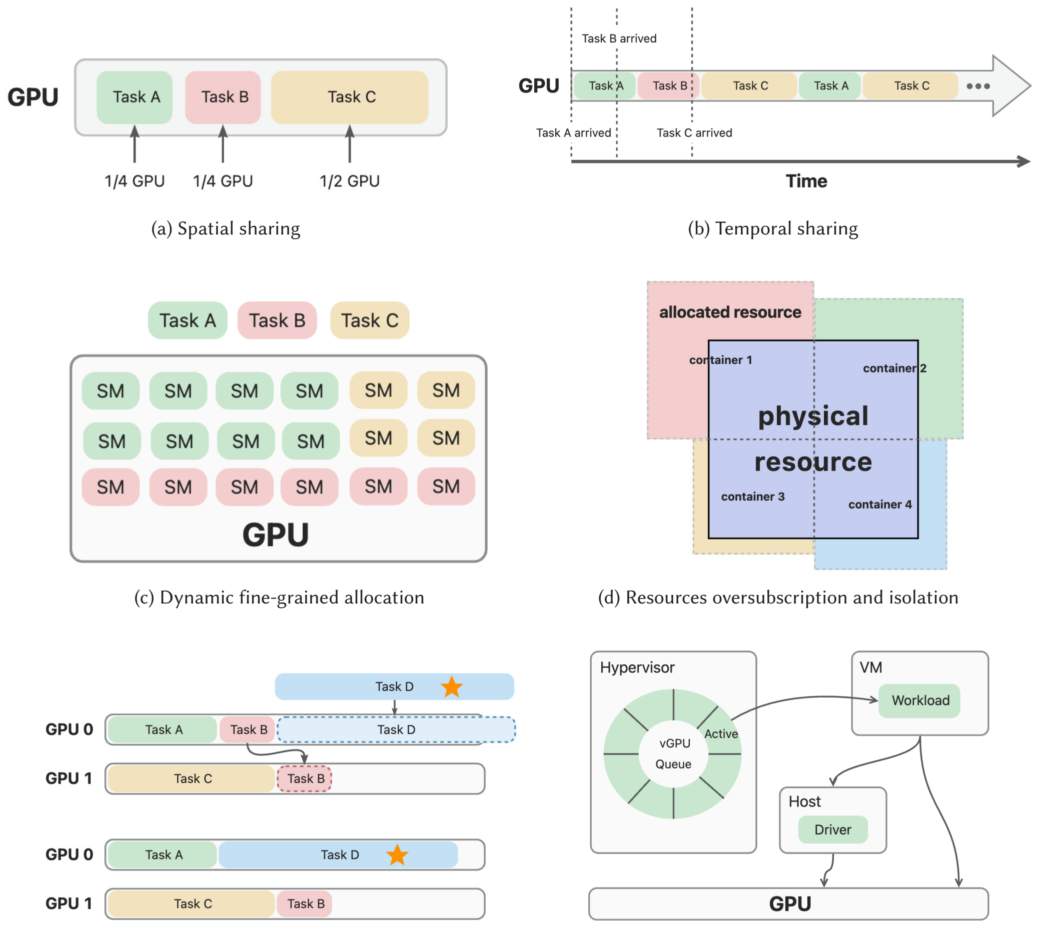
— $P_c$ is the power consumption.
— $E_t$ is the energy consumption per task. Balancing the load and reducing idle times can both lower the energy consumption per task.
— $R_t$ is the rate of tasks (number of tasks or batches per second). Sharing technologies that enable better parallel processing can increase the rate of tasks by allowing more tasks to be processed simultaneously.
— $\eta$ is the overall efficiency factor, accounting for hardware and software efficiencies. Ideally, the efficiency is 1, but in practice it is usually less than 1 due to various losses.

## 2.3 Key Concepts and Fundamentals

This section examines the fundamental rationale behind AI accelerator sharing, focusing on two key paradigms: profiling and prediction techniques that analyze workload patterns to optimize resource allocation, and resource sharing techniques that implement strategies such as fine-grained partitioning and virtualization for dynamic workload management.

### 2.3.1 *Profiling and Predicting Techniques.*

Profiling and predicting [83, 114, 121, 133] involves collecting detailed performance data through experiments or simulations before the actual



(a) Spatial sharing

(b) Temporal sharing

(c) Dynamic fine-grained allocation

(d) Resources oversubscription and isolation

(e) Preemption and Migration

(f) Virtualization

Fig. 4. Related Concepts of accelerator sharing. Note that these also apply to TPU, NPU and other AI accelerator architectures. (d) shows that four containers are shared among one node and an isolation environment. Due to oversubscription of resources, these containers can be allocated more than a quarter of the total resources available on the node.

execution of tasks. This data includes metrics such as execution time, resource utilization, and behavior patterns of the tasks. Based on this profiling data, predictive models are developed to estimate the performance of tasks under various conditions. These models can use historical data, statistical methods, or machine learning technologies to provide accurate predictions. The primary goal is to assist the scheduler in making informed decisions about resource allocation and task scheduling, ensuring efficient and optimized execution.

### 2.3.2 *Resource Sharing Techniques.*

*Spatial sharing.* Spatial sharing, as shown in Figure 4(a), refers to the simultaneous utilization of different hardware resources by multiple tasks within the same system. For instance, different processor cores or distinct sections of an accelerator can be allocated to different tasks at the same time. This method leverages the parallel execution capabilities of modern hardware to increase throughput and resource utilization. By distributing tasks across available resources, spatial sharing aims to minimize idle times and maximize the effective use of computational power. However, it requires sophisticated resource allocation strategies to prevent conflicts and ensure fair distribution among tasks.

Nvidia offers two distinct spatial sharing methods: **Multi-Process Service (MPS)** [6] and **Multi-Instance GPU (MIG)** [5]. MPS enables the concurrent execution of multiple CUDA applications on a single GPU by establishing a shared environment that optimizes resource utilization and minimizes latency through parallel execution. However, it necessitates sophisticated scheduling to effectively manage resource contention. MIG partitions a single GPU into multiple isolated instances, each with dedicated resources such as memory and compute cores. This enables efficient and secure multi-tenant usage and flexible resource allocation, which are ideal for environments requiring strict resource isolation. The following section will present a discussion of works employing spatial sharing technologies, with the exception of MIG and MPS.

**Dynamic fine-grained allocation:** Dynamic fine-grained allocation [26, 44, 103, 132] is a method that dynamically allocates hardware resources at a very granular level to different tasks based on their immediate needs and workload characteristics, as shown in Figure 4(c). This approach involves continuously monitoring the resource demands of tasks and adjusting allocations in real-time to ensure optimal utilization. For instance, specific parts of a processing unit, such as individual cores or even cache lines, or segments of memory down to the level of cache blocks or individual memory pages, can be allocated to different tasks as their requirements change. This allows for efficient resource use without significant overhead. The primary advantage of this method is its flexibility and responsiveness to changing workloads, which can lead to improved performance and reduced resource wastage. However, it requires sophisticated monitoring and allocation mechanisms to function effectively.

**Resources oversubscription and isolation:** Resources oversubscription and isolation [92, 107, 126], as shown in Figure 4(d), is a strategy where more virtual resources are allocated to tasks than the actual physical resources available, based on the observation that not all tasks will use their peak resources simultaneously. This approach can significantly increase resource utilization and overall system throughput. However, to ensure that performance does not degrade during peak demand, isolation mechanisms are implemented. These mechanisms guarantee that critical tasks have access to necessary resources when needed, preventing interference from other tasks.

*Temporal sharing.* Temporal sharing [68, 69], as shown in Figure 4(b), involves the sequential sharing of the same hardware resource by multiple tasks over different time periods. In this approach, tasks are assigned specific time slices during which they can use the resource exclusively. Once a task's time slice is over, the resource is allocated to another task. This time-multiplexing strategy allows for dynamic adaptation to changing task demands and can enhance overall resource utilization. However, it can also introduce overhead due to context switching, where the state of a task is saved and restored repeatedly, potentially affecting execution efficiency.

**Preemption and migration:** Figure 4(e) illustrates the concepts of preemption and migration in GPU task scheduling, two pivotal strategies used to address resource contention and improve overall system performance. These techniques are particularly important in handling dynamic and mixed workloads, such as combining real-time inference tasks with long-running training jobs. By reallocating or redistributing tasks, they ensure that high-priority or latency-sensitive tasks are given timely access to computational resources while maintaining overall system efficiency and balance.

Preemption [38, 124] allows AI accelerators to interrupt lower-priority tasks to allocate resources to higher-priority tasks, such as real-time inference, improving responsiveness and resource efficiency. However, it introduces context switching overhead and requires sophisticated scheduling.

Migration [104] involves moving tasks between computational units within AI accelerators to balance load, optimize energy efficiency, and manage thermal conditions. While it enhances resource utilization and performance, it also introduces latency and complexity in data management.
**Virtualization:** Virtualization [54, 56, 58, 74], as shown in Figure 4(f), involves creating virtual instances of hardware resources that can be allocated to tasks as needed. This method abstracts the physical hardware, allowing multiple tasks to run on the same physical resource as if they each had their own dedicated hardware. Virtualization enables efficient resource sharing, isolation, and flexibility in resource allocation. It also simplifies the management of resources by providing a consistent interface regardless of the underlying hardware. Virtualization can improve security and fault isolation, as tasks running in separate virtual environments are less likely to interfere with each other. However, virtualization introduces some overhead due to the need for a hypervisor or virtual machine manager to coordinate and manage the virtual instances.

*Spatio-temporal sharing.* Spatio-temporal sharing [22, 37, 46] combines the principles of both spatial and temporal sharing to optimize resource utilization in two dimensions. Tasks are allocated to different resources concurrently (spatial sharing) while also being scheduled to share the same resources at different times (temporal sharing). This hybrid approach aims to fully exploit the capabilities of modern multi-core and multi-accelerator systems, achieving high levels of efficiency and performance. While it offers significant benefits in terms of flexibility and resource optimization, it also demands more complex and sophisticated scheduling algorithms to manage the dual dimensions of sharing effectively.

## 3 Efficiency-oriented AI Accelerator Sharing

As highlighted in Figure 2, this section focuses on efficiency as a central goal in resource sharing for AI accelerators, emphasizing key metrics, such as time, latency, throughput, and cost. Efficiency is a crucial objective in scheduling optimization, particularly for AI accelerators, where different aspects, including time, cost, and system throughput, must be considered. Due to the distinct characteristics of training and inference workloads, the section categorizes these workloads to discuss their efficiency strategies. Training optimization focuses on reducing time while maximizing throughput, whereas inference prioritizes low latency and high throughput. Additionally, mixed-use scenarios where training and inference coexist are examined, with an analysis of the associated tradeoffs and synergies.

### 3.1 Efficiency of Training

The efficiency of AI accelerator sharing technology has significant implications for training workloads. As outlined in Section 2.2.1, we categorize these works focusing on training workloads by time and throughput. Training time is a critical metric in machine learning, as it directly impacts the speed at which models can be developed and deployed. Faster training times mean quicker iterations and faster turnaround from model conception to implementation. Throughput measures the number of tasks or operations completed in a given period. Higher throughput indicates a system's capability to process more data or train more models simultaneously. At the conclusion of this section, we will present insights gleaned from these works. A summary of these works is provided in Table 3.

*3.1.1 Execution Time.* The concept of time efficiency is defined in Section 2.2.1. One of the most effective methods for enhancing time efficiency is to share computing and memory resources. With regard to the GPU computing resources, Synergy [83] emphasizes scheduling and sharing various types of computational resources (such as GPUs, FPGAs, and TPUs) in a multi-tenant

Table 3. Summary of Studies on Efficiency-oriented Sharing under DL Training Workloads

| Year | Name | Obj. | Approaches | Claimed Effect | Dev. | Exp.T | Exp.S | Code |
|---|---|---|---|---|---|---|---|---|
| 2024 | Parcae[29] | ★ | Profiling and Predicting | Better System robustness; Throughput 10× ↑ | G | R | 32 V100 | ✓ |
| 2023 | TGS[117] | ♠ | Dynamic Fine-grained Allocation | Throughput 15× ↑ | G | R | 2 A100 | ✓ |
| | Layer-Puzzle[34] | ♠ | Profiling and Predicting | PE Utilization 1.95× ↑ | N | S | SCALE-Sim | - |
| | V10[124] | ♠★♣ | Fine-grained Preemption | NPU Utilization 1.64× ↑; Throughput 1.57× ↑; Latency 1.56× ↓ | N | S | - | - |
| | Flexer[82] | ♦ | Out-of-Order Scheduling | JCT 2.2× ↓ | N | R | 8 NPU | - |
| | DM-NPU[21] | ♠★ | Dynamic Fine-grained Allocation | Throughput 10.8%–27.8%↑;PE Utilization 2.68×↑ | N | S | - | - |
| 2022 | MAGMA[55] | ★♠ | Dynamic Fine-grained Allocation | Throughput 1.4%–1.6× ↑ | A | R | 6 Accelerator | ✓ |
| | Muri[143] | ♦♠ | Dynamic Fine-grained Allocation | JCT 3.6× ↓; Makespan 1.6× ↑ | G | R | 64 V100 | ✓ |
| | Synergy[83] | ♦♠ | Profiling and Predicting | JCT 3.4× ↓ | G | R | 32 V100 | ✓ |
| | DISC[69] | ♦ | Time Slicing | JCT 1.15× ↓; Accuracy 1.58× ↑ | G | R | 15 GPU | - |
| | NeiDty[28] | ♦ | Dynamic Fine-grained Allocation | JCT 10% ↓ | G | S | Gem5-GPU | - |
| | Miso[62] | ♦ | Prediction and Dynamic Partitioning | JCT 16%–49% ↓ | G | R | 8 A100 | - |
| | Arax[91] | ♦ | Dynamic Migration | JCT 20% ↓ | G & F | R | Arria 10 RX550X RTX 2080Ti | - |
| 2021 | Zico[67] | ♠★ | Dynamic Fine-grained Allocation | Throughput 1.6-8.3× ↑ | G | R | V100;2080Ti | - |
| | OM[33] | ♠ | Profiling and Predicting | 125%–150% Memory Oversubscription | G | S | GPGPU-Sim | - |
| | Layerweaver[88] | ♠★ | Dynamic Fine-grained Allocation | NPU Utilization 44% ↑; Throughput 60.1% ↑ | N | S | MAESTRO[90] | - |
| 2020 | TVT[54] | ♠ | Tensor Virtualization | Reduce DRAM writes 2× | N | S | - | - |
| | Salus[132] | ♦♠ | Dynamic Fine-grained Allocation | JCT 3.19× ↓; GPU Utilization 2.38× ↑ | G | R | 2 P100 | ✓ |
| | CPPE[133] | ♦ | Predict and Oversubscription | JCT 1.56%–1.64× ↓ | G | S | Gem5-GPU | - |
| | SIGMA[95] | ♠ | Dynamic Fine-grained Allocation | Utilization 3%–5.7× ↑ | A | R | SIGMA Engine | - |
| 2018 | MASK[9] | ★♥ | Low-overhead Virtual Memory | Throughput 58.7% ↑; Unfairness 22.4% ↓ | G | S | Mosaic | - |
| | Gandiva[118] | ♠♣ | Dynamic Migration | GPU Utilization 26% ↑ | G | R | 180 P100&P40 | - |

**Obj.(Objectives):** ★(Throughput) ♠(Utilization) ♣(Latency) ♦(Job Completion Time) ♥(Fairness) **Dev.(Device Type):** G(GPU) N(NPU) F(FPGA) A(ASIC) **Exp.T(Experiment Type):** R(Real Cluster) S(Simulation) **Exp.S(Experiment Scales):** the scale of physical cluster. -: not clearly specified.

cluster environment. It utilizes a new near-optimal online algorithm to perform multi-resource, workload-aware assignments. DISC [69] only focuses on GPU time sharing to optimize hyperparameter tuning processes. It leverages dynamic priority scheduling and real-time load monitoring to improve resource utilization. Unlike the time-share mechanism, Salus [132] introduces fine-grained GPU sharing mechanisms that support concurrent execution of multiple deep learning tasks on the same GPU. It emphasizes fine-grained resource allocation, concurrent execution, and resource isolation. Muri [143] considers optimizing overall training efficiency by interleaving multiple types of computational resources (CPU, GPU, memory, storage). It emphasizes multi-resource interleaving, resource partitioning, and dynamic resource scheduling. Gandiva's [118] Key methodologies encompass time-slicing GPUs across multiple jobs for the purpose of low-latency feedback, dynamic migration to improve locality and efficiency, and adaptive resource allocation through packing and grow-shrink mechanisms. This approach enhances early feedback, increases cluster utilization by 26%, and accelerates hyper-parameter searches, achieving up to a 10× speedup in certain tasks. Miso [62] leverages performance prediction to dynamically allocate NVIDIA **Multi-Instance GPU (MIG)** resources, optimizing workload placement and ensuring fairness in multi-tenant GPU clusters. Arax [91] is a runtime framework that decouples applications from heterogeneous accelerators, enabling dynamic task mapping, efficient accelerator sharing, and elastic resource allocation, while providing a simple API for transparent and adaptable accelerator utilization. All these methods use sharing of computing power to improve time efficiency.

From the perspective of GPU memory resources, several perspectives can be optimized, and a multitude of potential avenues exist for optimization, encompassing **Translation Lookaside Buffer (TLB)** sharing, inter-GPU and inter-host memory sharing, and memory oversubscription. NeiDty [28] reduces translation latency and improves GPU performance by sharing address translation results between different TLBs. CPPE [133] presents a coordinated page prefetch and eviction mechanism for managing memory oversubscription in GPUs.

Flexer [82] introduces an out-of-order scheduling mechanism in NPU that allows tasks to execute out of their submission order. This strategy dynamically adjusts the execution order of tasks based on their dependencies and resource requirements. MAGMA [55] proposes an optimization framework that uses intelligent algorithms to map multiple **deep neural network (DNN)** tasks onto multiple accelerator cores. DM-NPU [21] presents Dataflow Mirroring technical, which involves replicating dataflows to allow shared data paths among multiple tasks and enables fine-grained spatial multitasking on systolic-array NPUs by optimizing dataflow control. TVT [54] focuses on CNN Accelerators and it proposes Tensor Virtualization abstracts tensor data into virtual tensors, optimizing data storage.

*3.1.2 System throughput.* Higher system throughput ensures that the AI accelerator remains fully occupied, thereby increasing its utilization, reflecting the overall efficiency and processing capacity of the system.

Prediction and profiling are critical approaches to increase training throughput. Parcae [29] employs an availability predictor to forecast future instance preemptions. By predicting which instances are likely to be preempted in advance, the system can proactively adjust and minimize the impact of preemption. Similarly, OM [33] leverages the output of a transformer model to accurately perform prefetching and pre-eviction by monitoring and predicting memory usage. TGS [117] overcomes the limitations of native Kubernetes by intercepting each Docker's kernel commits. TGS maintains high GPU utilization through continuous monitoring and adaptive kernel rate control.

An effective methodology for memory sharing will serve to decrease the latency associated with memory access. MASK [9] redesigns the GPU memory hierarchy, including cache, TLBs, and page table, and monitors the demand of each application to allocate appropriate memory capacity and enable fast recovery of idle memory. Meanwhile, it improves the cache coherency protocol to ensure fast synchronization when multiple applications access the same data, reducing conflicts caused by inconsistent data. Unlike MASK [9] involves modifications to the GPU memory hardware architecture, SMM [125] focuses primarily on the software level. It allows multiple thread blocks to share the same shared memory region simultaneously. Meanwhile, it interleaves memory data into different memory banks so thread blocks accessing memory concurrently can operate on different memory banks. Zico [67] proposes a shared memory pool and uses an on-demand memory allocation mechanism to optimize memory management.

In the context of NPUs or other accelerator, researchers often share resources at the layer level to increase utilization. Layerweaver [88] dynamically adjusts the execution order of layers by analyzing the different computational requirements and dependencies of each neural network layer. Layer-Puzzle [34] leverages layer heterogeneity and fine-grained task division to allocate different computing tasks to the most suitable NPU cores. The most advanced approach, V10 [124], employs preemptive multi-tasking, enabling time-sharing of an NPU core by preempting workloads at the task level. V10 uses preemption to balance the utilization between the Systolic Array and the Vector Unit, addressing operator imbalances and enhancing overall efficiency. SIGMA [95], an ASIC chip, employs flexible interconnects and distributed dataflow to enable efficient sharing of compute and memory resources, optimizing sparse and irregular GEMM operations for diverse deep learning workloads.

## 3.2 Efficiency of Inference

Training involves iterative, computationally intensive tasks where reducing training time and maximizing throughput are crucial, achieved through optimized resource utilization and parallel

Table 4. Summary of Studies on Efficiency-oriented Sharing under DL Inference Workloads

| Year | Name | Obj. | Approaches | Claimed Effect | Dev. | Exp.T | Exp.S | Code |
|---|---|---|---|---|---|---|---|---|
| 2024 | Llumnix[104] | ♣ | Preemption and Migration | latency 1.5× ↓; Cost 36% ↓ | G | R | 16 Nvdia A10 | ✓ |
| | vFPGA_layer[81] | ★ | Virtualization | Throughput 2.31–3.96× ↑ | F | R | Alveo U250 | - |
| | ParvaGPU[60] | ♠❡ | Dynamic Fine-grained Allocation | GPU Usage 12.5%↓with No SLO Violations | G | R | 8 A100 | - |
| 2023 | Gost[134] | ♣★ | Profiling and Predicting | Minimize the End-to-end Latency | G | R | 1 2080 | - |
| | FaST-GShare[37] | ★♠ | Spatio-temporal Sharing | Throughput 3.15× ↑; GPU Utilization 1.34× ↑ | G | R | 4 V100 | - |
| | SPLIT[75] | ♣ | Profiling and Predicting | Latency Violation Rate 43% ↓; Jitter 69.3% ↓ | G | R | Jetson Nano | ✓ |
| | KRISP[24] | ★ | Profiling and Predicting | Throughput 2× ↑; Energy 33% ↓ | G | R | AMD MI50 GPU | - |
| | H3M[135] | ♣ | Dynamic Fine-grained Allocation | Energy-delay-product 3.6–7.5× ↓ | F | R | Xilinx U200; U280 | - |
| 2022 | Gpulet[22] | ★ | Spatio-temporal Sharing | Throughput 61.7% ↑ | G | R | 4 2080Ti | ✓ |
| | REEF[38] | ★ | Preemption | Preemption Latency 12.3×↓; Throughput 7.7×↑ | G | R | AMD MI50 GPU | ✓ |
| | DGSF[32] | ♠♣ | Virtualization | Latency 53% ↓; GPU Utilization 16% ↑ | G | R | 8 V100 | - |
| 2020 | PREMA[23] | ❡♣★ | Profiling and Predicting | SLA Satisfaction 4.8× ↑; Latency 1.4× ↓ | N | S | - | ✓ |
| | Optimus[76] | ★ | Spatio-temporal Sharing | Throughput 1.98–7× ↑ | F | R | Intel HARP | - |
| | GSLICE[26] | ★♠ | Dynamic Fine-grained Allocation | GPU Utilization 1.6–9× ↑; Throughput 2–13× ↑ | G | R | 1 V100 | - |
| 2019 | ETC[63] | ♠★ | Preemption | GPU Memory Utilization 60%–270% ↑ | G | S | Mosaic | - |
| 2018 | TSM[46] | ★ | Temporal and Spatial Multiplexing | GPU Utilization 5× ↑ | G | R | 1 V100 | - |
| | gScale[123] | ★ | Dynamic Fine-grained Allocation | Virtual GPU 5× in Linux; 4× in Windows | G | R | - | - |
| 2016 | Baymax[14] | ♠♣ | Profiling and Predicting | 99%-ile Latancy 195× ↓; Utilization 91.3% ↑ | G | R | Nvidia K40 | - |
| | EIE[39] | ★ | Weight Share | Throughput 2.9×↑; Energy 19×↑ to [17] | A | S | - | - |
| | Cambricon-X[140] | ♣ | Dynamic Fine-grained Allocation | Latency 1.9–4.3× ↓ | A | S | - | - |

**Obj.(Objectives):** ★(Throughput) ♠(Utilization) ♣(Latency) ❡(SLA: Service Level Agreements) **Dev.(Device Type):** G(GPU) N(NPU) F(FPGA) A(ASIC) **Exp.T(Experiment Type):** R(Real Cluster) S(Simulation) **Exp.S(Experiment Scales):** the scale of physical cluster. -: not clearly specified. **Service Level Agreement:** It is a commitment between a service provider and a client that defines specific performance metrics, such as response time, throughput, and availability, which the service must meet.

processing. In contrast, inference focuses on real-time or near-real-time predictions, where low latency and high throughput are paramount. Latency is the time taken for an inference request to be processed from input to output. Low latency is crucial for real-time applications such as autonomous driving, online recommendations, and interactive AI systems, where delays can impact user experience and system effectiveness. High throughput is important for applications that need to handle a large volume of requests simultaneously, such as cloud-based AI services and large-scale deployment scenarios. These works are classified according to their impact on the efficiency of inference workloads, with a focus on latency and throughput, as outlined in Section 2.2.2. The exclusion of energy is justified by the fact that it is not the most prevalent factor in the sharing of technologies. Table 4 provides a detailed overview of these works.

*3.2.1 Latency.* Latency is the time delay between the input being provided to the system and the output (or result) being received.

Monitoring techniques are among the most common methods for improving efficiency in inference workloads. Gost [134] employs a monitoring system to track both spatial and temporal utilization of GPU resources, thereby enabling adaptive resource allocation for network function virtualization. In addition, the monitoring system in SPLIT [75] focuses on tracking the performance and resource usage of individual chunks, ensuring that QoS requirements are met. It uses a different strategy by splitting DNN models into equally sized chunks and scheduling these chunks for inference. H3M [135] introduces a coordinated FPGA framework that integrates heterogeneous sub-accelerators, layer-wise scheduling, and dynamic mapping strategies, leveraging real-time workload monitoring to optimize multi-DNN execution, achieving up to 7.5× **Energy-Delay Product (EDP)** reduction compared to state-of-the-art accelerators.

Different from monitoring, PREMA [23] introduces a predictive scheduling algorithm for NPUs that supports preemption, which makes it suitable for environments where tasks have varying execution times and need to be managed dynamically to optimize performance. In contrast, Baymax [14] optimizes non-preemptive accelerators by focusing on QoS. This involves predicting the duration and resource requirements of tasks to avoid conflicts due to the non-preemptive nature. In addition, it addresses queueing delays for computational resources by implementing a runtime system that orchestrates the execution of computing tasks from different applications. DGSF [32] introduces a disaggregated GPU resource-sharing framework for serverless functions, enabling efficient and low-latency inference by dynamically allocating and consolidating GPU resources across multiple functions using a virtualized GPU pool. Cambricon-X [140] leverages a PE-based architecture with efficient indexing and asynchronous processing to optimize computation and memory handling for sparse neural networks.

*3.2.2 System throughput.* KRISP [24] and GSLICE [26] both employ spatial partitioning to divide GPU resources among multiple tasks. However, there are key differences between them. KRISP [24] uses predictive models to forecast the resource needs of each kernel and dynamically adjusts the resources allocated to each kernel based on real-time requirements. This approach allows KRISP to operate at the kernel level, focusing on the specific needs of individual kernels within DNN models. GSLICE [26], however, employs both static and dynamic partitioning of GPU resources. This ensures that tasks do not interfere with each other by managing the allocation of GPU partitions at the task level. However, KRISP's fine-grained, kernel-level resource allocation contrasts with GSLICE's task-level management, highlighting their different approaches to achieving similar goals. vFPGA_layer [81] proposes a full-stack solution for enabling multi-tenancy on FPGAs, featuring an intra-FPGA virtualization layer, memory segmentation, and a network-on-chip architecture, achieving up to 3.96× throughput improvement in isolated settings while ensuring secure resource sharing and high-quality service.

Gpulet [22] and FaST-GShare [37] both utilize spatio-temporal sharing technologies to optimize GPU usage, but they cater to different environments. While the former is geared towards multi-GPU servers with a focus on heterogeneous models, the latter is designed for the flexibility and scalability requirements of serverless computing environments. Optimus [76] introduces a hypervisor for shared-memory FPGA platforms, enabling secure and efficient resource sharing through spatial and temporal multiplexing, with key techniques such as page table slicing for DMA isolation, a multiplexer tree for interconnect management, and a preemption interface for workload flexibility. Furthermore, TSM [46] integrates both spatial and temporal aspects into scheduling. It introduces dynamic query batching, which groups multiple execution kernels from disjoint DNN graphs into larger super-kernels. This approach allows for the scalable execution of hundreds of models on a single GPU by leveraging CUDA streams and inter-model batching, thereby optimizing both latency and throughput.

Besides spatial and temporal sharing, gScale [123] introduces two innovative mechanisms: the Private Shadow **Graphics Translation Table (GTT)** and Ladder Mapping with Fence Memory Space Pool. These mechanisms allow the GPU to access physical memory directly, effectively bypassing global graphics memory constraints. ETC [63] presents three key technologies: eviction, throttling, and compression. Eviction involves proactively removing less-critical data from GPU memory to free up space for more urgent data. Throttling controls the rate at which data is processed to prevent memory overload. Compression reduces the size of data stored in GPU memory,

Table 5. Summary of Studies on Efficiency-oriented Sharing under DL Mix Workloads

| Year | Name | Obj. | Approaches | Claimed Effect | Dev. | Exp.T | Exp.S | Code |
|---|---|---|---|---|---|---|---|---|
| 2023 | FGD[116] | ♠✷ | Dynamic Fine-grained Allocation | Unallocated Resources by Fragmentation 49% ↓ | G | S | 6.2k GPU | ✓ |
| | HRP[99] | ★ | Dynamic Fine-grained Allocation | Throughput 1.87× ↑ | G | R | 1 A100 | - |
| | AuRORA[58] | ★♥ | Virtualization | SLA Satisfaction 2.02× ↑; Throughput 1.33× ↑ | G | R | - | ✓ |
| | IGS-TLB[44] | ★ | Dynamic Fine-grained Allocation | L1 TLBs Hit Rate 18% ↑ | G | S | Gem5-GPU | - |
| | Sparse-DySta[31] | ♣ | Dynamic Fine-grained Allocation | Latency Violation Rate 10% ↓ | G | R | RTX 2080 | ✓ |
| | AVEC[56] | ♠★ | Accelerator Virtualization | Latency 7× ↓ | G | R | 3 GPU | - |
| | RealArch[114] | ♠ | Profiling and Predicting | Latency 2.16–8.54× ↓ | N | R | - | ✓ |
| | FPGAPooling[144] | ♦ | Dynamic FPGA allocation | Avg JCT 7× ↓; Tail JCT 4× ↓ | F | R & S | 3 Xilinx xc7vx690t | - |
| 2022 | gOver[126] | ✷ | Dynamic Oversubscription | Cost 20% ↓ | G | R | Intel NUC Kit | - |
| | DeepBoot[20] | ♦ | Dynamic Fine-grained Allocation | JCT 32%–38% ↓ | G | R & S | 8 Nvidia P40 | ✓ |
| 2021 | MIG-serving[106] | ✷ | Dynamic Fine-grained Allocation | Save 40% GPU | G | R & S | 24 A100 | - |
| | Gemini[12] | ★ | Profiling and Predicting | Performance Overhead Less than 5% | G | R | 1 V100 | ✓ |
| | CPSpatial[48] | ★♣ | Preemption | Preemption Latency 87.3% ↓; Throughput 1.43× ↑ | G | R | AMD Radeon VII | ✓ |
| 2020 | KubeShare[129] | ♠★ | Dynamic Fine-grained Allocation | Throughput 2× ↑ | G | R | 32 v100 | ✓ |
| | AntMan[119] | ♠ | Dynamic Fine-grained Allocation | GPU Memory Utilization 42% ↑; Computation Utilization 34%↑ | G | R | 64 V100 GPU | ✓ |
| | AvA[131] | ♠✜ | Virtualization | Virtualize 9 Accelerators and 11 Framework APIs | G | R | 4 GPU | ✓ |
| | PERSEUS[61] | ✷ | Dynamic Fine-grained Allocation | Cost 12% ↓ | G | S | Nvidia TensorRT | ✓ |
| | OAS[7] | ♠ | Dynamic Fine-grained Allocation | Improve Memory Share | G | R | P100, P4, TitanV | ✓ |
| 2018 | FELIPE[142] | ★ | Virtualization | Throughput 19.7%–21.5% ↑ | G | R | 2 GPU | - |
| | G-NET[139] | ★♣ | Dynamic Fine-grained Allocation | Throughput 70.8% ↑; Latency 44.3% ↓ | G | R | TITAN X | - |
| 2017 | Maestro[90] | ★ | Dynamic Fine-grained Allocation | Throughput 12.9%–20.2% ↑ | G | S | GPGPU-Sim | - |
| 2016 | vDNN[98] | ♠ | Memory virtualization | Reduce GPU Memory Usage 89%–95% ↓ | G | R | Titan X | ✓ |
| | Eyeriss[18] | ✷ | Row-Stationary (RS) Dataflow | Energy Consumption per Operation 1.4—2.5× ↓ | A | S | - | - |

**Obj.(Objectives):** ★(Throughput) ♠(Utilization) ♣(Latency) ♦(Job Completion Time) ✷(Cost) ♥(Fairness)✜(Interference) **Dev.(Device Type):** G(GPU) N(NPU) F(FPGA) A(ASIC) **Exp.T(Experiment Type):** R(Real Cluster) S(Simulation) **Exp.S(Experiment Scales):** the scale of physical cluster. -: not clearly specified.

maximizing the available space and improving overall efficiency. REEF [38] takes two steps to improve throughput, which are reset-based preemption and dynamic kernel padding. The preemption mechanism allows tasks to be interrupted and resumed with minimal delay, facilitating concurrent execution. The dynamic kernel padding enhances the ability to handle multiple concurrent DNN tasks on a single GPU. EIE [39] presents a specialized inference engine that has been optimized for the operation of compressed deep neural networks. This engine leverages the concept of weight sparsity and on-chip processing.

## 3.3 Efficiency of Mixed Workloads

Sharing AI accelerators for mixed workloads presents several challenges due to the differing characteristics and requirements of training and inference tasks. The key difficulties lie in balancing the conflicting demands for resources between training and inference tasks, dynamically managing workload variations, maintaining low latency for inference, optimizing overall efficiency, and handling the added infrastructure complexities and overheads. This section categorizes approaches for both training and inference workloads based on three factors: time, cost, and throughput. Table 5 provides a comprehensive overview of the detailed information presented.

*3.3.1 Execution Time.* Since training and inference tasks have different resource requirements and execution times, the scheduling system must adapt in real-time to these changes to maintain high time efficiency. Sparse-DySta [31] effectively recognizes and exploits sparsity in DNN workloads to minimize unnecessary computations, thereby accelerating task execution. Conversely, IGS-TLB [44] concentrates on hardware-level optimizations, specifically, TLBs sharing. Although it does not directly address scheduling algorithms, it enhances time efficiency by reducing memory operation latency. DeepBoot [20] designs **adaptive task scaling (ATS)** algorithm to utilize idle GPUs in the inference cluster for the training DLTs and implements **auto-fast elastic (AFE)** to reduce the restart overhead by inference GPU reclaiming.

RealArch [114] includes estimation models that predict the execution time and resource requirements for different DNN tasks; and then its real-time scheduling algorithm, which prioritizes tasks based on their deadlines and resource needs, dynamically maps tasks to the available cores, balancing the load and reducing contention. OaSM [7] presents an overlap-and-save method that reduces redundant calculations by dividing the input data into overlapping segments, processing each segment separately, and then combining the results. The approach leverages the fast shared memory available on GPUs to store intermediate data. AVEC [56] framework virtualizes GPU resources by intercepting API calls from applications and redirecting them to remote GPU accelerators. This allows lightweight devices to offload computationally intensive tasks to more powerful GPUs located either in the cloud or at the edge. The use of containers ensures that applications can be easily migrated and managed across different nodes in the network.

In addition to GPU-related work, the sharing of certain accelerators (e.g., FPGA, ASIC) has also been demonstrated to accelerate AI workloads. FPGAPooling [144] introduces a centralized FPGA resource pooling framework that dynamically allocates and shares FPGA accelerators among multiple tenants, addressing the inefficiency of static allocation in cloud environments. FPGAPooling improves the average and tail job completion time by up to 7 and 4 times, respectively.

#### 3.3.2 *Cost.*

The objective of cost-effective scheduling of mixed deep learning workloads is to achieve a balance between performance and cost. A number of recent studies use dynamic scaling and resource allocation technologies that adjust to real-time demand, with the aim of enhancing cost efficiency. GOver [126] introduces an economy-oriented approach to GPU virtualization, which leverages dynamic and adaptive oversubscription. AntMan [119] automatically scales GPU resources up or down based on real-time workload demands. Concurrently, it incorporates cost-awareness in scaling decisions, thereby reducing unnecessary expenditures on GPU resources during low-demand periods. FGD [116] presents Fragmentation Gradient Descent as a method for the management and reduction of memory fragmentation in GPU-sharing workloads. Eyeriss [18] leverages a row-stationary dataflow to optimize energy efficiency in convolutional neural networks by minimizing data movement and maximizing local data reuse on a spatial architecture.

The majority of these alternative approaches concentrate on enhancing the efficacy of GPU virtualization, minimizing overheads, and optimizing resource allocation to reduce costs. AvA [131] implements technologies to reduce the overhead associated with GPU virtualization and Uses hardware-assisted virtualization and optimized software stacks to achieve lower latency and higher throughput. Unlike that, vDNN [98] virtualizes deep neural networks to achieve scalable and memory-efficient neural network design.

Beside these works, numerous research studies implement adaptive scheduling strategies that optimize resource utilization and cost based on workload characteristics. MIG-serving [106] employs a dynamic reconfiguration of GPU instances based on current workload, avoiding over-provisioning and reducing costs. PERSEUS [61] analyzes the tradeoffs between performance and cost in multi-tenant environments and implements scheduling strategies that consider the specific needs and costs of different tenants.

#### 3.3.3 *System throughput.*

A number of studies concentrate on the administration of GPU resources in multi-tenant settings. AuRORA [58] employs virtualization technologies to abstract physical accelerator resources into **Virtual Accelerator Instances (VAIs)**, which can be allocated to different tenants as needed. Concurrently, it continuously monitors the usage of each virtual accelerator instance and adjusts resources according to load changes. Gemini [12] detects and characterizes the burstiness of GPU workloads by analyzing their execution patterns. This helps in understanding how workloads can be interleaved without causing significant performance degradation. The system ensures that high-priority or bursty workloads receive sufficient resources while allowing low-priority or less bursty workloads to utilize the remaining capacity. KubeShare [129] includes a GPU device plugin for Kubernetes, which abstracts the physical GPUs into logical GPU slices. The mechanism allows GPUs to be divided into smaller, shareable units (slices) that can be allocated to different containers based on their requisite specifications.

A significant body of literature emphasizes the importance of dynamically adjusting resource allocation to meet the needs of different tasks. HRP [99] divides GPU resources into multiple hierarchical levels, each representing different granularity of resource partitions. Furthermore, a reinforcement learning model is employed to facilitate dynamic adjustments in resource allocation at each hierarchical level. The reinforcement learning model continuously optimizes resource allocation strategies by observing task performance and feedback. Similarly, G-NET [139] ensures that the GPU is kept busy by dynamically scheduling GPU kernels from different network functions. The dynamic partitioning of GPU resources in Maestro [90] entails the continuous monitoring of task performance and resource usage, adapting resource allocations in real-time based on current demands, and using a feedback loop to refine and optimize allocations.

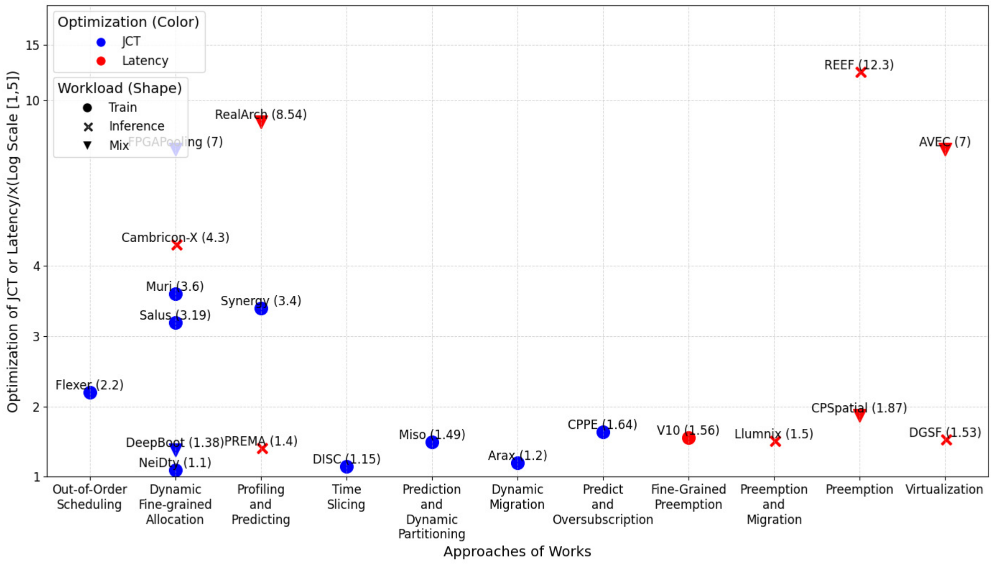
Similarly, fine-grained resource sharing [48, 142] is a common strategy for improving system throughput. In contrast to CPSpatial [48], which focuses on dividing the GPU into partitions and using preemption to manage task priorities, FELIPE [142] focuses on creating vGPUs and fine-grained scheduling of these virtual resources.

## 3.4 Discussion

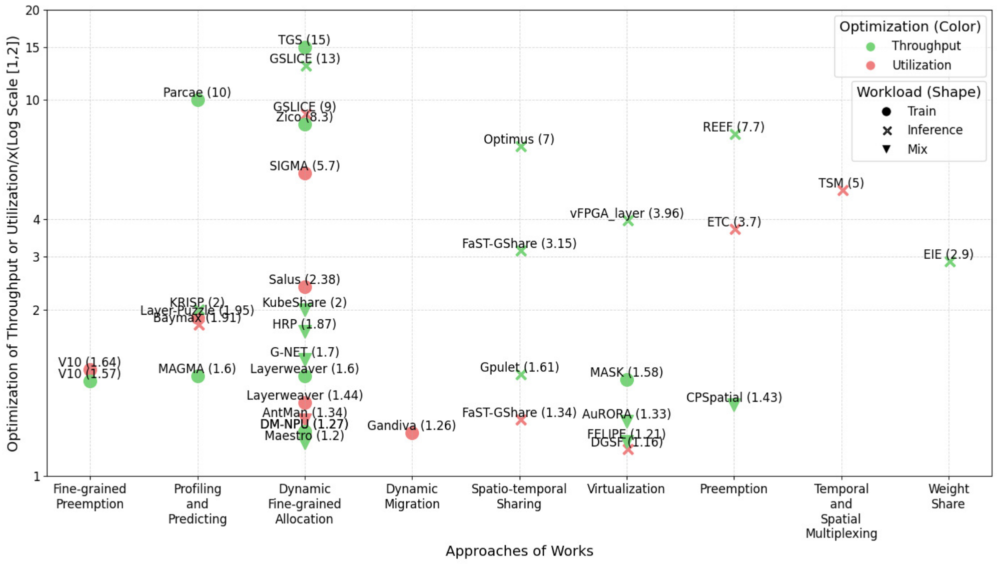
In summary, optimizing resource-sharing efficiency in AI accelerators involves a variety of methodological approaches, each achieving varying levels of success. Figure 5 highlights optimization improvements across key metrics, ranging from 1.2× to 15×. For **job completion time (JCT)** and latency optimization, solutions such as REEF and AIFM deliver significant gains of 12.3× and 7×, respectively, particularly in inference workloads. In terms of utilization and throughput, methods such as TGLS and GSLICE demonstrate remarkable improvements of up to 15× and 13×. The subsequent discourse will meticulously examine the most salient points delineated in this section.

*Dynamic fine-grained allocation and profiling & predicting are most common approaches for efficient sharing.* As illustrated in Figure 5, the majority of existing research adopts these two approaches to optimize the accelerator. These methodologies result in enhanced operational efficacy by guaranteeing that tasks are not postponed due to inadequate resources. The reduction of latency is achieved by ensuring that high-priority tasks are allocated the requisite computational resources without delay. Furthermore, they contribute to the enhanced efficiency of AI systems by ensuring that resources are neither underutilized nor overcommitted, thereby maintaining an optimal balance that enhances system throughput and responsiveness. This results in a more seamless and dependable operation of AI services, particularly in environments with dynamic and varied workloads, which aligns more closely with production environments characterized by large and complex workloads.

*Inference workloads achieve superior optimization compared to training.* The optimization of inference workloads generally surpasses that of training workloads, as shown in Figure 5, where



(a) JCT and Latency



(b) Utilization and Throughput

Fig. 5. Summary of efficiency optimization of metrics. The optimization results for some works are a range interval, and we have chosen the maximum of the range to show here.

inference consistently achieves higher gains across key metrics. This can be attributed to inference's more predictable and lightweight computational patterns, which allow for more effective resource scheduling and management. In contrast, training workloads, often characterized by higher complexity and variability, show relatively modest improvements, particularly in systems employing fine-grained resource allocation and dynamic optimization strategies.

*NPUs and TPUs have become increasingly popular as AI accelerators for efficient shared environments.* Since 2023, the evolution of chips dedicated to deep learning has accelerated significantly, leading to a surge in research focused on optimizing the performance of NPUs [34, 82, 124] and TPUs. These processors have a simpler architecture compared to GPUs, and their openness to a wider range of internal hardware interfaces enables more nuanced resource sharing. This, in turn, enhances the efficiency of data sharing. As NPU computational capabilities continue to expand, it is expected that even more researchers will delve into this field, further advancing the technology.

*It is very common to consider system throughput as the main objective.* This focus on system throughput aims to maximize the number of tasks processed within a given timeframe, which is crucial for improving the overall efficiency and performance of AI systems. By prioritizing throughput, researchers can ensure that AI accelerators such as GPUs and TPUs are used to their fullest potential, handling multiple tasks simultaneously and reducing idle times. This approach not only enhances the productivity of AI systems but also makes them more scalable and responsive to varying workload demands. As a result, achieving high system throughput is a key goal in optimizing resource allocation and utilization in shared AI accelerator environments.

*Optimizing efficiency requires simultaneous consideration of user experience.* Optimizing efficiency in GPU accelerator scheduling requires a careful balance between maximizing resource utilization and maintaining a positive user experience. While achieving high throughput and low latency is critical for efficiency, it is equally important to ensure that user-centric metrics, such as responsiveness and fairness, are not compromised. Modern scheduling algorithms must account for diverse workloads with varying priorities, from real-time inference tasks to large-scale training jobs. For example, PREMA[23] achieves a 1.4× increase in throughput alongside a 4.8× improvement in SLA satisfaction, illustrating that it is possible to enhance system performance without compromising service quality. This underscores the need to consider both efficiency and user experience in scheduling strategies, particularly when addressing challenges such as worst-case latency or percentile guarantees (e.g., P95 or P99 latency), which are critical for maintaining user satisfaction.

*Shared accelerator clusters, not individual devices, are now an efficient way to train LLMs.* Shared accelerator clusters have become a preferred method for training LLMs [50]. By leveraging multiple devices in a distributed setup, they reduce training time and optimize resource utilization. Techniques such as tensor and pipeline parallelism divide computations across devices, overcoming memory bottlenecks and enabling the training of larger models. Compared to single devices, shared clusters provide greater scalability and efficiency, making them indispensable for modern AI workflows.

## 4 Rising Concerns in AI Accelerator Sharing

As shown in Figure 2, this section delves beyond the realm of efficiency, encompassing critical concerns such as fairness, interference, and security in the context of AI accelerator resource sharing. It explores the impact of computing and memory bandwidth interference on performance, methodically examines approaches to ensure equitable resource allocation, and underscores the significance of robust security measures. A synopsis of advancements in the pertinent literature is enumerated in Table 6.

### 4.1 Fairness

The concept of fairness in computational resource allocation refers to the equitable distribution of resources among tasks and users, preventing monopolization. It involves balancing task

Table 6. Summary of Studies on Non-efficiency-oriented Sharing under DL Workloads

| Year | Name | Obj. | Approaches | Claimed Effect | Dev. | Exp.T | Exp.S | Code |
|---|---|---|---|---|---|---|---|---|
| 2024 | Orion[103] | ✤ | Dynamic Fine-grained Allocation | Throughput 7.3× ↑; Cost 1.49× ↓ | G | R | 1 V100 | ✓ |
|  | Guardian[92] | ▲ | Resources Isolation | Overhead Only 4%–12% | G | R | Quadro A4000;3080 Ti | - |
|  | DOTPBA[84] | ▲♦ | Dynamic OTP Buffer Allocation | Excution time 13.2%–17.5% ↓ | G | S | MGPUSim | - |
| 2023 | iGniter[120] | ✤✷ | Profiling and Predicting | Guarantee the SLOs by Cost 25% ↓ | G | R | 10 V100 | ✓ |
|  | Libra[72] | ✤✷ | Dynamic Fine-grained Allocation | Backward Time 6%–20% ↓ | G | R & S | 8 V100 | ✓ |
|  | IADeep[16] | ♠✤ | Co-optimizing Task Assignments | Utilization 29%–31%↑; JCT49%↓; Makespan 67%↓ | G | R | 20 RTX 3090 | ✓ |
|  | ctmGPU[68] | ♠✤ | Interleave PCIe Channel Accesses | JCT 31.8%–38.3% ↓ when GPU Memory 1.33–2× | G | R | 8 P40 | - |
| 2022 | Astraea[111] | ♦♠♥ | Primal-Dual Algorithm; Sharing Rewards | Fairness 20% ↑ | G | S | - | - |
|  | VELTAIR[73] | ♣✤ | Dynamic Fine-grained Allocation | Latency 50% ↓; Throughput 45%–71% ↑ | G | R | - | ✓ |
|  | GDC[10] | ✤ | Track the Contention | Help Designer Know How APPs Are Affected | G | S | GPGPU-Sim | - |
|  | MoCA[57] | ★♥✤ | Dynamic Fine-grained Allocation | SLA 1.8× ↑; Throughput 1.7× ↑; Fairness 1.2× ↑ | G | S | FireSim | - |
| 2021 | MAPA[96] | ♦✤ | Dynamic Fine-grained Allocation | 75% JCT 12.4% ↓; Worst Execution Time 35% ↓ | G | R & S | 1 V100 | ✓ |
|  | ParSecureML[137] | ▲ | Parallel Processing | Makespan 33.8× ↓ | G | R | 3 V100 | ✓ |
| 2020 | Themis[77] | ♥ | Finish-time Fairness | Fairness 2.25× ↑ | G | R | 64 K80 GPU | - |
|  | Fingerprint[108] | ▲ | Extracting FPGA Fingerprint | Identify Cloud FPGA Instances | F | R | f1.2xlarge;f1.4xlarge | ✓ |
|  | Gandiva_fair[11] | ▲ | Job Migration and Trading | Fair-share in a Heterogeneous Setting | G | R | V100;P100;K80 | - |
| 2019 | IAVS[121] | ✤ | Profiling and Predicting | Accuracy 15%–40% ↑ | G | R | 1 Nvidia P100 | - |
|  | GAugur[65] | ♠✤ | Profiling and Predicting | GPU Utilization 20%–60% ↑ | G | R | 1 RTX 1060 | - |
|  | gQoS[74] | ✤ | Adaptive Virtualized under the QoS Target | GPU Utilization 25.85% ↓ under QoS | G | R | Intel HD Graphics 5500 | - |
| 2018 | VMCG[107] | ♠♥ | Separate V-channel | GPU Allocation Fairness 60%–80% ↑ | G | R | 1 GTX 750Ti | - |
|  | FREFI[101] | ▲ | Wide Parametrizable Secret Sharing Core | Throughput 6.4 Gbit/s | F | R | - | - |
|  | Leaky Wires[35] | ▲ | Leaky Wires Covert Communication | Bandwidth 6 kbps | F | R | Virtex | - |
|  | Graviton[110] | ▲ | Static Analysis Validation | Latency 17%–33% ↓ | G | R | GTX 780; GTX Titan | - |
| 2017 | Prophet[13] | ♠✤ | Profiling and Predicting | Utilization 49.9%↑by Prediction Error 5.47% | G | R | Nvidia K40 | - |
| 2016 | Mystic[109] | ✤ | Profiling and Predicting | Throughput 27.5% ↑; GPU Utilization 16.3% ↑ | G | R | 34 Nvidia K40m | - |

**Obj.(Objectives):** ★(Throughput) ♠(Utilization) ♣(Latency) ♦(Job Completion Time) ✷(Cost) ♥(Fairness) ✤(Interference) ▲(Security) **Dev.(Device Type):** G(GPU) N(NPU) F(FPGA) **Exp.T(Experiment Type):** R(Real Cluster) S(Simulation) **Exp.S(Experiment Scales):** the scale of physical cluster. -: not clearly specified.

priorities in multi-tenancy environments, ensuring a tradeoff between latency and throughput, and dynamically adapting to changing workloads and resource availability.

Themis [77] design allocates GPUs to winning bids by trading off fairness for efficiency in the short term, but ensuring finish-time fairness in the long term, rather than prioritizing one over the other. Astraea [111] employs incentives to encourage fair resource sharing among tenants. By offering rewards or benefits for efficient resource sharing, it encourages tenants to cooperate and share GPU resources. VMCG's [107] approach is to ensure fairness by allocating dedicated GPU channels to virtual machines. Each VM is allocated its own channel, preventing interference and ensuring that resources are distributed fairly. Gandiva_fair [11] is a scheduling framework for heterogeneous GPU clusters that balances efficiency and fairness. It employs dynamic profiling, job migration, and GPU trading mechanisms to optimize resource allocation across multiple GPU models, improving cluster utilization and performance while ensuring fair resource distribution among users.

## 4.2 Interference

In the context of accelerator sharing, interference refers to the negative impact on task performance due to competition for shared resources such as memory bandwidth, compute units, and cache. This can lead to increased latency, reduced throughput, and unpredictable task execution. Addressing interference involves strategies such as resource isolation, fair scheduling, dynamic resource management, and performance isolation to ensure efficient and predictable task performance.

*4.2.1 Computing.* In a GPU, compute units include **Streaming Multiprocessors (SMs)** with CUDA cores, Tensor Cores, **Ray Tracing Cores (RT Cores)**, Texture Units, Shader Units, **Raster Operations Pipelines (ROPs)**, **Special Function Units (SFUs)**, and **Load/Store Units (LD/ST Units)**, all working together for efficient parallel computation and graphics rendering. Most interference in GPUs typically occurs in the **Streaming Multiprocessors (SMs)**, which house the CUDA cores. This is where the bulk of parallel computation happens, leading to contention for compute resources.

We can categorize existing approaches in two groups. First, several works focus on predicting and mitigating interference through real-time monitoring and forecasting, ensuring efficient and predictable resource allocation. Prophet [13] focuses on predicting QoS metrics to improve resource utilization and ensure performance compliance in non-preemptive accelerator environments. IAVS [121] aims to predict and manage performance interference for interference-aware scheduling in virtualized GPU environments. GAugur [65] quantifies performance interference among colocated gaming workloads to optimize resource utilization in cloud gaming scenarios. Each approach targets different environments and types of interference, using tailored prediction technologies to enhance performance and resource management. IGniter [120] ensures predictable DNN inference performance in the cloud by employing interference-aware GPU resource provisioning, dynamically allocating resources based on predicted interference levels. IADeep [16] employs a middleware approach that intelligently multiplexes deep learning workloads, using interference models to predict and mitigate contention. Mystic [109] employs a collaborative filtering framework to predict the interference caused by incoming applications based on their similarity to currently running applications. This prediction enables the scheduler to minimize interference and optimize system throughput.

Second, many works manage interference by focusing specifically on fine-grained GPU sharing and thread allocation technologies. Orion [103] provides fine-grained GPU sharing with interference awareness, dynamically adjusting GPU usage for ML applications by monitoring interference levels and reallocating resources. Libra [72] introduces contention-aware GPU thread allocation for data parallel training, optimizing thread distribution in high-speed network environments by assigning threads based on contention metrics. GQoS [74] provides a QoS-oriented GPU virtualization framework with adaptive capacity sharing, which allows for the dynamic adjustment of resource allocation according to real-time workload demands to maintain QoS for multiple tenants. MAPA [96] introduces a multi-accelerator pattern allocation policy that optimizes GPU resource sharing and reduces contention in multi-tenant GPU servers by identifying and leveraging workload patterns. VELTAIR [73] enhances multi-tenant deep learning services through adaptive compilation and scheduling, optimizing performance by dynamically adjusting compilation strategies and scheduling decisions based on current system states.

*4.2.2 Memory and Bandwidth.* Memory interference occurs when multiple tasks compete for memory resources, resulting in increased latency, reduced throughput, and unpredictable performance. Mitigation strategies may include resource partitioning, priority scheduling, dynamic resource management, performance isolation, and effective caching, with the objective of ensuring efficient and consistent task execution.

GDC [10] is concerned with the real-time tracking and management of cache contention, allowing for more efficient use of last-level cache resources by identifying and mitigating cache-related performance issues. MoCA [57] introduces memory-centric, adaptive execution strategies that dynamically adjust memory allocation based on the specific demands of multi-tenant DNN workloads. G10 [138] integrates GPU memory and storage with smart tensor migrations, creating a unified architecture that enables efficient data movement and reduces memory access contention.

CtmGPU [68] develops advanced scheduling technologies for tensor movements across multiple GPUs, optimizing the timing and coordination of data transfers to prevent bottlenecks.

## 4.3 Security

Security in resource sharing involves protecting sensitive data through encryption and access control, ensuring resource isolation to prevent interference between tasks, implementing fair scheduling to avoid resource contention, maintaining network security with protocols and intrusion detection, and preventing side-channel attacks. It ensures that multiple tasks or tenants can share GPU resources efficiently without compromising data privacy, task performance, or system integrity.

Guardian [92] is designed to maintain isolation and enforce security policies in multi-tenant GPU environments. To address the bandwidth issue of additional security metadata, DOTPBA [84] uses a dynamic batching scheme to transfer only a single set of metadata for each batched group of data responses. The proposed design constantly tracks the communication pattern of each GPU, periodically adjusts the allocated buffer size, and dynamically forms batches of data transfers. ParSecureML [137] employs methods such as data parallelism, where large datasets are divided and processed concurrently across multiple GPU cores, and model parallelism, which splits the machine learning model itself for parallel execution. Additionally, it incorporates cryptographic protocols such as homomorphic encryption and secure multiparty computation to ensure data privacy and security during processing. Graviton [110] addresses the need for secure execution on heterogeneous systems, introducing a co-designed hardware-software framework that ensures kernel isolation and encryption efficiency with minimal performance impact. These combined methods optimize computational performance while maintaining robust data protection.

In addition to the GPU-related work mentioned above, some research has focused on edge accelerators, such as FPGAs, to address security challenges. Leaky Wires [35] explores vulnerabilities in FPGA routing resources, revealing how crosstalk effects in long wires can be exploited for covert communication and proposing mitigation strategies to secure the routing infrastructure. FREFI [101] designs an optimized FPGA architecture for secure data storage, achieving significant improvements in throughput and resource efficiency compared to traditional methods. Furthermore, Fingerprint [108] investigates security risks in cloud FPGA deployments by using **Physical Unclonable Functions (PUFs)** to identify unique FPGA instances, highlighting potential threats and suggesting countermeasures to mitigate them. These studies highlight the unique challenges and solutions for ensuring the security of edge accelerators in distributed computing environments.

## 4.4 Discussion

In conclusion, this section analyzes three critical aspects of AI accelerator sharing optimization: fairness, interference, and security, with various approaches achieving different levels of improvement. Fairness optimization approaches like Themis achieve up to 2.25× improvement, while interference mitigation solutions such as VELTAIR and IADEEP demonstrate significant gains of 1.71× and 1.67×, respectively. Security-focused solutions such as ParSecureML and Guardian show moderate but stable improvements of 1.33× and 1.12×.

The optimization techniques can be categorized by their primary objectives: fairness-oriented solutions (Themis, VMCG, Astraea) focus on equitable resource allocation with improvements ranging from 1.2× to 2.25×; interference-focused approaches (IAVS, Prophet, Mystic) address resource contention with enhancements from 1.25× to 1.6×; and security-centered solutions (ParSecureML, Guardian, DOTPBA) prioritize protected execution with improvements from 1.12× to 1.33×.

*Balancing competition and security is a challenge in AI accelerator resource sharing.* Balancing competition and security in AI accelerator resource sharing is a complex challenge that requires careful consideration of tradeoffs. Competition-oriented methods prioritize efficient resource utilization among tasks or users, often leading to higher improvement ratios and enhanced system performance. However, these approaches may not adequately address potential security vulnerabilities. Security-focused solutions prioritize protecting the system from threats, such as unauthorized access or data breaches, but the additional overhead of implementing these protective mechanisms often results in more modest performance gains. This inherent tradeoff highlights the difficulty of designing systems that can effectively meet diverse optimization goals, requiring innovative strategies to strike a balance between competitive resource allocation and robust system security.

*Fairness should be achieved through isolation.* The concept of scheduling fairness can be interpreted from two distinct perspectives [27]. From the scheduler's perspective, fairness implies an even allocation of resources to each task, thereby maximizing the overall utilization of resources. Conversely, from the users' perspective, fairness entails that the resources requested by the users will be honored, even if they are unable to fully utilize the majority of the requested resources within a given time slot. This results in a situation where the scheduler must observe some tasks experiencing difficulties in executing their instructions while simultaneously expending resources on idle tasks to ensure the desired level of fairness to the users.

These two perspectives, which are contradictory, must be reconciled through competitive isolation. When some tasks have more free resources, the scheduler allocates these resources to other tasks that require them more urgently or have been waiting for an extended period, thus achieving the scheduler's fairness. Consequently, when the load of these free tasks suddenly increases, the scheduler must employ robust resource isolation to reclaim the resources and maintain user-level fairness. The crux of this coordination lies in the efficacy of the resource isolation policy. It is our contention that future research on these two factors—competitive isolation and resource allocation—will prove mutually beneficial. By enhancing the mechanisms for dynamic resource allocation and isolation, it is possible to achieve a balance that satisfies both the scheduler's and users' perspectives of fairness, thereby optimizing overall system performance and user satisfaction.

*Memory security will be the important issue for LLMs.* Efficient memory usage is critical for LLMs due to their size and computational demands. Memory sharing techniques significantly enhance throughput and scalability by reducing redundancy and improving hardware utilization. Examples include weight sharing, activation reuse, and memory pooling, which collectively lower costs and increase efficiency.

However, shared memory poses privacy risks, such as data leakage, timing side-channel attacks, and residual data exposure. Ensuring user isolation in memory sharing involves techniques such as memory partitioning, encryption, data sanitization, and access controls.

Balancing memory efficiency and privacy is key to optimizing LLMs. Future efforts should focus on dynamic memory management, hardware support for secure memory, and minimizing cross-user interference to achieve scalable and secure LLM deployments.

*Normal LLM inference request will be disrupted by a long-text request.* With the rapid advancement of LLMs, long-text inference requests have become increasingly common. These requests demand higher computational and memory resources and take longer to complete. The growing volume of long-text requests inevitably leads to resource contention with regular requests that have strict latency requirements. This issue has drawn significant attention from researchers [94], who are now employing efficient scheduling techniques to manage long-text requests and enhance overall cluster efficiency.

## 5 Open Challenges

This section will present a discussion of existing challenges that have not been considered in the papers included in this survey, as well as potential future directions for research in this area.

### 5.1 SLA-aware Resource Sharing and Job Packing

The optimization of efficiency represents a fundamental objective in the context of sharing technologies. Most sharing algorithms are designed to optimize various aspects of efficiency. The initial step is to define efficiency. Traditionally, these algorithms use system metrics, such as utilization, to assess efficiency. However, we believe this approach is insufficient for measuring the efficiency of an application. A more appropriate method would be to employ user-level metrics, such as latency, to assess the effectiveness of the application from the user's perspective. Due to privacy and security concerns, the system scheduler is unable to obtain user-level metrics. This makes mapping an application's **SLA (Service Level Agreement)** or user metrics using system metrics a significant challenge. UFO [93] addresses this by employing a scheduling frequency-based approach to map application latency. This involves adjusting CPU allocation based on predicted scheduling frequency. In the future, selecting an appropriate model with a limited set of system metrics to gauge an application's SLA on a GPU will likely gain considerable attention.

To address these challenges, future research should focus on leveraging machine learning models to bridge the gap between system-level metrics and user-level performance indicators. By training lightweight models on system data such as GPU utilization, memory bandwidth, and scheduling frequency, it is possible to predict user-level metrics such as latency or SLA compliance.

### 5.2 Resource Sharing over Heterogeneous Accelerators

Modern data centers and edge AI systems deploy a diverse range of AI accelerators with varying computational capabilities. These accelerators can range from highly advanced to relatively limited, with some featuring opaque internal mechanisms and others operating transparently. Resource allocation also differs widely, from fine-grained control to coarse-grained levels constrained by virtualization techniques.

This heterogeneity necessitates a unified sharing framework to optimize resource use across both data center and edge accelerators. Such a framework should integrate sharing technologies tailored to the unique attributes of each device, enabling efficient utilization and boosting overall system performance.

### 5.3 Coordination of Global Scheduler and Local Schedulers

AI accelerator sharing frequently relies on profiling and prediction methodologies. However, the global scheduler's tasks, which include calculating remaining resources, synchronizing information, and selecting suitable nodes, can introduce significant overhead, particularly under substantial workloads. It is important to note that certain processes involved in node selection are characterized by high-complexity optimization. The present study posits that the offloading of certain computations to the local scheduler—including the determination of resource sharing feasibility and the calculation of the remaining resource capacity after sharing—can alleviate the global scheduler's burden. This delegation enables the global scheduler to prioritize its core functions, thereby enhancing the efficiency of the overall scheduling process.

To address the challenges outlined, a practical approach would be to implement a hierarchical scheduling system where the local scheduler handles computationally intensive tasks such as resource sharing feasibility and residual capacity calculations. This design reduces the global scheduler's workload, allowing it to focus on high-level decision-making and system-wide synchronization. Additionally, leveraging lightweight machine learning models at the local level can further optimize resource allocation and enhance scheduling efficiency.

### 5.4 Resource Contention and User Experience in LLM Workloads

The proliferation of LLMs has led to a surge in the availability of bot chat services for the general public. The response latency is a key factor in the service experience. While sharing technology can improve the overall system's resource use efficiency, it can also, to some extent, affect the user's experience.

To address this issue, techniques such as fast competition, localization, and resource isolation become exceptionally important. If the monitoring program identifies an impact on the user experience, then it is essential for the program to respond promptly to prevent the resource quota of the reasoning service from being exceeded or to make predictions about these scenarios in advance. These factors underscore the significance of detecting resource competition.

### 5.5 Accelerator Sharing Expands from Device-level to Cluster-level

As AI models, particularly **large language models (LLMs)**, grow in size and complexity, individual devices increasingly fall short of meeting the computational and memory demands of these tasks. This limitation has driven a shift toward cluster-level sharing, which introduces new challenges. Managing data synchronization across distributed accelerators becomes significantly more complex, especially for tasks requiring precise coordination in parallel processing. Network bottlenecks emerge as data transfers between nodes scale, diminishing the efficiency of both training and inference. Furthermore, maintaining workload fairness and minimizing interference across heterogeneous cluster hardware necessitate sophisticated scheduling algorithms. Resource fragmentation within clusters adds another layer of complexity, as underutilized accelerators often coexist with overloaded ones, further complicating resource optimization.

These challenges necessitate the development of robust cluster-level resource management systems that can dynamically allocate, optimize, and monitor resources to maximize performance and scalability.

## 6 Conclusions

This survey systematically investigates the latest resource-sharing technologies for AI accelerators. We first provide a statistical analysis of current research from multiple perspectives. Then, we introduce the key concepts and analyze the performance measures that are greatly impacted by AI accelerator sharing. This also includes an exploration of the common principles and fundamental techniques adopted in the literature. In particular, we categorize existing studies by their aim of system optimization with primary focus on efficiency, including time efficiency, cost efficiency, and system throughput efficiency. Additionally, we examine issues related to fairness, task interference, and the security implications associated with resource sharing.

Furthermore, we highlight critical open challenges that have not been addressed in existing efforts. These challenges encompass the need for better resource allocation strategies in shared environments. This survey provides a comprehensive overview of the state-of-the-art, guiding future research directions and emphasizing the necessity for further advancements in resource-sharing technologies.

## References

[1] 2024. Apple's Neural Engine. Retrieved from https://www.apple.com/newsroom/2024/05/apple-introduces-m4-chip/
[2] 2024. Google Edge TPU. Retrieved from https://cloud.google.com/edge-tpu?hl=zh-cn
[3] 2024. Huawei's Ascend. Retrieved from https://e.huawei.com/en/products/computing/ascend
[4] 2024. NVIDIA Management Library. Retrieved from https://developer.nvidia.com/management-library-nvml

[5] 2024. NVIDIA Multi-instance GPU. Retrieved from https://www.nvidia.com/en-us/technologies/multi-instance-gpu/

[6] 2024. NVIDIA Multi-process Service. Retrieved from https://docs.nvidia.com/deploy/mps/

[7] Karel Adámek, Sofia Dimoudi, Mike Giles, and Wesley Armour. 2020. GPU fast convolution via the overlap-and-save method in shared memory. *ACM Trans. Archit. Code Optim.* 17, 3, Article 18 (Aug. 2020), 20 pages.

[8] Hyunho Ahn, Munkyu Lee, Sihoon Seong, Gap-Joo Na, In-Geol Chun, Blesson Varghese, and Cheol-Ho Hong. 2024. ScissionLite: Accelerating distributed deep learning with lightweight data compression for IIoT. In *IEEE Transactions on Industrial Informatics* 20, 10 (2024), 11950–11960.

[9] Rachata Ausavarungnirun, Vance Miller, Joshua Landgraf, Saugata Ghose, Jayneel Gandhi, Adwait Jog, Christopher J. Rossbach, and Onur Mutlu. 2018. MASK: Redesigning the GPU memory hierarchy to support multi-application concurrency. In *23rd International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'18)*. Association for Computing Machinery, New York, NY, USA, 503–518.

[10] Javier Barrera, Leonidas Kosmidis, Hamid Tabani, Jaume Abella, and Francisco J. Cazorla. 2022. Contention tracking in GPU last-level cache. In *IEEE 40th International Conference on Computer Design (ICCD'22)*. 76–79.

[11] Shubham Chaudhary, Ramachandran Ramjee, Muthian Sivathanu, Nipun Kwatra, and Srinidhi Viswanatha. 2020. Balancing efficiency and fairness in heterogeneous GPU clusters for deep learning. In *15th European Conference on Computer Systems (EuroSys'20)*. Association for Computing Machinery, New York, NY, USA, Article 1, 16 pages. DOI: https://doi.org/10.1145/3342195.3387555

[12] Hung-Hsin Chen, En-Te Lin, Yu-Min Chou, and Jerry Chou. 2023. Gemini: Enabling multi-tenant GPU sharing based on kernel burst estimation. *IEEE Trans. Cloud Comput.* 11, 1 (2023), 854–867.

[13] Quan Chen, Hailong Yang, Minyi Guo, Ram Srivatsa Kannan, Jason Mars, and Lingjia Tang. 2017. Prophet: Precise QoS prediction on non-preemptive accelerators to improve utilization in warehouse-scale computers. *SIGARCH Comput. Archit. News* 45, 1 (Apr. 2017), 17–32.

[14] Quan Chen, Hailong Yang, Jason Mars, and Lingjia Tang. 2016. Baymax: QoS awareness and increased utilization for non-preemptive accelerators in warehouse scale computers. In *21st International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'16)*. Association for Computing Machinery, New York, NY, USA, 681–696.

[15] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. 2014. DianNao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. *ACM SIGARCH Comput. Archit. News* 42, 1 (2014), 269–284.

[16] Wenyan Chen, Zizhao Mo, Huanle Xu, Kejiang Ye, and Chengzhong Xu. 2023. Interference-aware multiplexing for deep learning in GPU clusters: A middleware approach. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC'23)*. Association for Computing Machinery, New York, NY, USA, Article 30, 15 pages.

[17] Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Jia Wang, Ling Li, Tianshi Chen, Zhiwei Xu, Ninghui Sun et al. 2014. DaDianNao: A machine-learning supercomputer. In *47th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE, 609–622.

[18] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. 2016. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. In *43rd International Symposium on Computer Architecture (ISCA'16)*. IEEE Press, 367–379. DOI: https://doi.org/10.1109/ISCA.2016.40

[19] Yu-Hsin Chen, Tien-Ju Yang, Joel Emer, and Vivienne Sze. 2019. Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices. *IEEE J. Emerg. Select. Topics Circ. Syst.* 9, 2 (2019), 292–308.

[20] Zhenqian Chen, Xinkui Zhao, Chen Zhi, and Jianwei Yin. 2023. DeepBoot: Dynamic scheduling system for training and inference deep learning tasks in GPU cluster. *IEEE Trans. Parallel Distrib. Syst.* 34, 9 (2023), 2553–2567.

[21] Jinwoo Choi, Yeonan Ha, Jounghoo Lee, Sangsu Lee, Jinho Lee, Hanhwi Jang, and Youngsok Kim. 2023. Enabling fine-grained spatial multitasking on systolic-array NPUs using dataflow mirroring. *IEEE Trans. Comput.* 72, 12 (2023), 3383–3398.

[22] 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 Annual Technical Conference (USENIX ATC'22)*. USENIX Association, 199–216.

[23] Yujeong Choi and Minsoo Rhu. 2020. PREMA: A predictive multi-task scheduling algorithm for preemptible neural processing units. In *IEEE International Symposium on High Performance Computer Architecture (HPCA'20)*. 220–233.

[24] Marcus Chow, Ali Jahanshahi, and Daniel Wong. 2023. KRISP: Enabling kernel-wise right-sizing for spatial partitioned GPU inference servers. In *IEEE International Symposium on High-Performance Computer Architecture (HPCA'23)*. 624–637.

[25] Chunhua Deng, Siyu Liao, and Bo Yuan. 2021. PermCNN: Energy-efficient convolutional neural network hardware architecture with permuted diagonal structure. *IEEE Trans. Comput.* 70, 2 (2021), 163–173.

[26] Aditya Dhakal, Sameer G. Kulkarni, and K. K. Ramakrishnan. 2020. GSLICE: Controlled spatial sharing of GPUs for a scalable inference platform. In *11th ACM Symposium on Cloud Computing (SoCC'20)*. Association for Computing Machinery, New York, NY, USA, 492–506.

[27] Kate Donahue and Jon Kleinberg. 2020. Fairness and utilization in allocating resources with uncertain demand. In *Conference on Fairness, Accountability, and Transparency (FAT*'20)*. Association for Computing Machinery, New York, NY, USA, 658–668.

[28] Yajuan Du, Mingyang Liu, Yuqi Yang, Mingzhe Zhang, and Xulong Tang. 2022. Enhancing GPU performance via neighboring directory table based inter-TLB sharing. In *IEEE 40th International Conference on Computer Design (ICCD'22)*. 146–153.

[29] Jiangfei Duan, Ziang Song, Xupeng Miao, Xiaoli Xi, Dahua Lin, Harry Xu, Minjia Zhang, and Zhihao Jia. 2024. Parcae: Proactive, liveput-optimized DNN training on preemptible instances. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI'24)*. USENIX Association, 1121–1139.

[30] Amir Erfan Eshratifar and Massoud Pedram. 2020. Runtime deep model multiplexing for reduced latency and energy consumption inference. In *IEEE 38th International Conference on Computer Design (ICCD'20)*. 263–270.

[31] Hongxiang Fan, Stylianos I. Venieris, Alexandros Kouris, and Nicholas Lane. 2023. Sparse-DySta: Sparsity-aware dynamic and static scheduling for sparse multi-DNN workloads. In *56th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'23)*. Association for Computing Machinery, New York, NY, USA, 353–366.

[32] Henrique Fingler, Zhiting Zhu, Esther Yoon, Zhipeng Jia, Emmett Witchel, and Christopher J. Rossbach. 2022. DGSF: Disaggregated GPUs for serverless functions. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS'22)*. 739–750. DOI: https://doi.org/10.1109/IPDPS53621.2022.00077

[33] Debashis Ganguly, Rami Melhem, and Jun Yang. 2021. An adaptive framework for oversubscription management in CPU-GPU unified memory. In *Design, Automation & Test in Europe Conference & Exhibition (DATE'21)*. 1212–1217.

[34] Chengsi Gao, Ying Wang, Cheng Liu, Mengdi Wang, Weiwei Chen, Yinhe Han, and Lei Zhang. 2023. Layer-Puzzle: Allocating and scheduling multi-task on multi-core NPUs by using layer heterogeneity. In *Design, Automation & Test in Europe Conference & Exhibition (DATE'23)*. 1–6.

[35] Ilias Giechaskiel, Kasper B. Rasmussen, and Ken Eguro. 2018. Leaky wires: Information leakage and covert communication between FPGA long wires. In *Asia Conference on Computer and Communications Security (ASIACCS'18)*. Association for Computing Machinery, New York, NY, USA, 15–27. DOI: https://doi.org/10.1145/3196494.3196518

[36] Ashish Gondimalla, Noah Chesnut, Mithuna Thottethodi, and T. N. Vijaykumar. 2019. SparTen: A sparse tensor accelerator for convolutional neural networks. In *52nd Annual IEEE/ACM International Symposium on Microarchitecture*. 151–165.

[37] Jianfeng Gu, Yichao Zhu, Puxuan Wang, Mohak Chadha, and Michael Gerndt. 2023. FaST-GShare: Enabling efficient spatio-temporal GPU sharing in serverless computing for deep learning inference. In *52nd International Conference on Parallel Processing (ICPP'23)*. Association for Computing Machinery, New York, NY, USA, 635–644.

[38] Mingcong Han, Hanze Zhang, Rong Chen, and Haibo Chen. 2022. Microsecond-scale preemption for concurrent GPU-accelerated DNN inferences. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI'22)*. USENIX Association, 539–558.

[39] 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. *ACM SIGARCH Comput. Archit. News* 44, 3 (2016), 243–254.

[40] Cheol-Ho Hong, Ivor Spence, and Dimitrios S. Nikolopoulos. 2017. GPU virtualization and scheduling methods: A comprehensive survey. *ACM Comput. Surv.* 50, 3, Article 35 (June 2017), 37 pages.

[41] Liang Hu, Xilong Che, and Si-Qing Zheng. 2016. A closer look at GPGPU. *ACM Comput. Surv.* 48, 4, Article 60 (Mar. 2016), 20 pages.

[42] 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 *International Conference for High Performance Computing, Networking, Storage and Analysis (SC'21)*. 1–15.

[43] 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 *28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS'23)*. Association for Computing Machinery, New York, NY, USA, 457–472.

[44] Weiming Huang, Yajuan Du, and Mingyang Liu. 2023. GPU performance acceleration via intra-group sharing TLB. In *52nd International Conference on Parallel Processing (ICPP'23)*. Association for Computing Machinery, New York, NY, USA, 705–714.

[45] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V. Le, Yonghui Wu et al. 2019. GPipe: Efficient training of giant neural networks using pipeline parallelism. In *Advan. Neural Inf. Process. Syst.*, Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alche-Buc, Emily B. Fox, and Roman Garnett (Eds.). Vol. 32. NeurIPS, 103–112.

[46] 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).

[47] Myeongjae Jeon, Shivaram Venkataraman, Amar Phanishayee, Junjie Qian, Wencong Xiao, and Fan Yang. 2019. Analysis of large-scalemulti-tenant GPU clusters for DNN training workloads. In *USENIX Annual Technical Conference (USENIX ATC'19)*. 947–960.

[48] Zhuoran Ji and Cho-Li Wang. 2021. Collaborative GPU preemption via spatial multitasking for efficient GPU sharing. In *European Conference on Parallel Processing*. Springer, 89–104.

[49] Yimin Jiang, Yibo Zhu, Chang Lan, Bairen Yi, Yong Cui, and Chuanxiong Guo. 2020. A unified architecture for accelerating distributed DNN training in heterogeneous GPU/CPU clusters. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI'20)*. USENIX Association, 463–479.

[50] Ziheng Jiang, Haibin Lin, Yinmin Zhong, Qi Huang, Yangrui Chen, Zhi Zhang, Yanghua Peng, Xiang Li, Cong Xie, Shibiao Nong et al. 2024. MegaScale: Scaling large language model training to more than 10,000 GPUs. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI'24)*. USENIX Association, 745–760. Retrieved from https://www.usenix.org/conference/nsdi24/presentation/jiang-ziheng

[51] Norm Jouppi, George Kurian, Sheng Li, Peter Ma, Rahul Nagarajan, Lifeng Nai, Nishant Patil, Suvinay Subramanian, Andy Swing, Brian Towles et al. 2023. TPU v4: An optically reconfigurable supercomputer for machine learning with hardware support for embeddings. In *50th Annual International Symposium on Computer Architecture (ISCA'23)*. Association for Computing Machinery, New York, NY, USA, Article 82, 14 pages.

[52] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers et al. 2017. In-datacenter performance analysis of a tensor processing unit. *SIGARCH Comput. Archit. News* 45, 2 (June 2017), 1–12.

[53] Raju K. and Niranjan N. Chiplunkar. 2018. A survey on techniques for cooperative CPU-GPU computing. *Sustain. Comput.: Inform. Syst.* 19 (2018), 72–85.

[54] Donghyun Kang and Soonhoi Ha. 2020. Tensor virtualization technique to support efficient data reorganization for CNN accelerators. In *57th ACM/IEEE Design Automation Conference (DAC'20)*. 1–6.

[55] Sheng-Chun Kao and Tushar Krishna. 2022. MAGMA: An optimization framework for mapping multiple DNNs on multiple accelerator cores. In *IEEE International Symposium on High-Performance Computer Architecture (HPCA'22)*. 814–830.

[56] Jason Kennedy, Vishal Sharma, Blesson Varghese, and Carlos Reaño. 2023. Multi-tier GPU virtualization for deep learning in cloud-edge systems. *IEEE Trans. Parallel Distrib. Syst.* 34, 7 (2023), 2107–2123.

[57] Seah Kim, Hasan Genc, Vadim Vadimovich Nikiforov, Krste Asanović, Borivoje Nikolić, and Yakun Sophia Shao. 2023. MoCA: Memory-centric, adaptive execution for multi-tenant deep neural networks. In *IEEE International Symposium on High-Performance Computer Architecture (HPCA'23)*. 828–841.

[58] Seah Kim, Jerry Zhao, Krste Asanovic, Borivoje Nikolic, and Yakun Sophia Shao. 2023. AuRORA: Virtualized accelerator orchestration for multi-tenant workloads. In *56th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'23)*. Association for Computing Machinery, New York, NY, USA, 62–76.

[59] Young Geun Kim and Carole-Jean Wu. 2020. AutoScale: Energy efficiency optimization for stochastic edge inference using reinforcement learning. In *53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'20)*. 1082–1096.

[60] Munkyu Lee, Sihoon Seong, Minki Kang, Jihyuk Lee, Gap-Joo Na, In-Geol Chun, Dimitrios Nikolopoulos, and Cheol-Ho Hong. 2024. ParvaGPU: Efficient spatial GPU sharing for large-scale DNN inference in cloud environments. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC'24)*. 1–14. DOI: https://doi.org/10.1109/SC41406.2024.00048

[61] Matthew LeMay, Shijian Li, and Tian Guo. 2020. PERSEUS: Characterizing performance and cost of multi-tenant serving for CNN models. In *IEEE International Conference on Cloud Engineering (IC2E'20)*. 66–72.

[62] Baolin Li, Tirthak Patel, Siddharth Samsi, Vijay Gadepally, and Devesh Tiwari. 2022. MISO: Exploiting multi-instance GPU capability on multi-tenant GPU clusters. In *13th Symposium on Cloud Computing*. 173–189.

[63] Chen Li, Rachata Ausavarungnirun, Christopher J. Rossbach, Youtao Zhang, Onur Mutlu, Yang Guo, and Jun Yang. 2019. A framework for memory oversubscription management in graphics processing units. In *24th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'19)*. Association for Computing Machinery, New York, NY, USA, 49–63.

[64] Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania et al. 2020. PyTorch distributed: Experiences on accelerating data parallel training. *arXiv preprint arXiv:2006.15704* (2020).

[65] Yusen Li, Chuxu Shan, Ruobing Chen, Xueyan Tang, Wentong Cai, Shanjiang Tang, Xiaoguang Liu, Gang Wang, Xiaoli Gong, and Ying Zhang. 2019. GAugur: Quantifying performance interference of colocated games for improving resource utilization in cloud gaming. In *28th International Symposium on High-Performance Parallel and Distributed Computing (HPDC'19)*. Association for Computing Machinery, New York, NY, USA, 231–242.

[66] Feng Liang, Zhen Zhang, Haifeng Lu, Chengming Li, Victor Leung, Yanyi Guo, and Xiping Hu. 2024. Resource allocation and workload scheduling for large-scale distributed deep learning: A survey. *arXiv preprint arXiv:2406.08115* (2024).

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

[68] Shao-Fu Lin, Yi-Jung Chen, Hsiang-Yun Cheng, and Chia-Lin Yang. 2023. Tensor movement orchestration in multi-GPU training systems. In *IEEE International Symposium on High-Performance Computer Architecture (HPCA'23)*. 1140–1152.

[69] Liu Liu, Jian Yu, and Zhijun Ding. 2023. Adaptive and efficient GPU time sharing for hyperparameter tuning in cloud. In *51st International Conference on Parallel Processing (ICPP'22)*. Association for Computing Machinery, New York, NY, USA, Article 5, 11 pages.

[70] Shaoli Liu, Zidong Du, Jinhua Tao, Dong Han, Tao Luo, Yuan Xie, Yunji Chen, and Tianshi Chen. 2016. Cambricon: An instruction set architecture for neural networks. *ACM SIGARCH Comput. Archit. News* 44, 3 (2016), 393–405.

[71] Yan Liu, Yansha Deng, Arumugam Nallanathan, and Jinhong Yuan. 2023. Machine learning for 6G enhanced ultra-reliable and low-latency services. *IEEE Wirel. Commun.* 30, 2 (2023), 48–54.

[72] Yunzhuo Liu, Bo Jiang, Shizhen Zhao, Tao Lin, Xinbing Wang, and Chenghu Zhou. 2023. Libra: Contention-aware GPU thread allocation for data parallel training in high speed networks. In *IEEE Conference on Computer Communications (INFOCOM'23)*. 1–10.

[73] Zihan Liu, Jingwen Leng, Zhihui Zhang, Quan Chen, Chao Li, and Minyi Guo. 2022. VELTAIR: Towards high-performance multi-tenant deep learning services via adaptive compilation and scheduling. In *27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'22)*. Association for Computing Machinery, New York, NY, USA, 388–401.

[74] Qiumin Lu, Jianguo Yao, Haibing Guan, and Ping Gao. 2020. gQoS: A QoS-oriented GPU virtualization with adaptive capacity sharing. *IEEE Trans. Parallel Distrib. Syst.* 31, 4 (2020), 843–855.

[75] Diaohan Luo, Tian Yu, Yuewen Wu, Heng Wu, Tao Wang, and Wenbo Zhang. 2023. SPLIT: QoS-aware DNN inference on shared GPU via evenly-sized model splitting. In *52nd International Conference on Parallel Processing (ICPP'23)*. Association for Computing Machinery, New York, NY, USA, 605–614.

[76] Jiacheng Ma, Gefei Zuo, Kevin Loughlin, Xiaohe Cheng, Yanqiang Liu, Abel Mulugeta Eneyew, Zhengwei Qi, and Baris Kasikci. 2020. A hypervisor for shared-memory FPGA platforms. In *25th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'20)*. Association for Computing Machinery, New York, NY, USA, 827–844. DOI: https://doi.org/10.1145/3373376.3378482

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

[78] Mohammad-Ali Maleki, Alireza Nabipour-Meybodi, Mehdi Kamal, Ali Afzali-Kusha, and Massoud Pedram. 2021. An energy-efficient inference method in convolutional neural networks based on dynamic adjustment of the pruning level. *ACM Trans. Des. Autom. Electron. Syst.* 26, 6 (2021), 1–20.

[79] Ruben Mayer and Hans-Arno Jacobsen. 2020. Scalable deep learning on distributed infrastructures: Challenges, techniques, and tools. *ACM Comput. Surv.* 53, 1, Article 3 (Feb. 2020), 37 pages.

[80] Xupeng Miao, Yujie Wang, Youhe Jiang, Chunan Shi, Xiaonan Nie, Hailin Zhang, and Bin Cui. 2022. Galvatron: Efficient transformer training over multiple GPUs using automatic parallelism. *arXiv preprint arXiv:2211.13878* (2022).

[81] Panagiotis Miliadis, Dimitris Theodoropoulos, Dionisios Pnevmatikatos, and Nectarios Koziris. 2024. Architectural support for sharing, isolating and virtualizing FPGA resources. *ACM Trans. Archit. Code Optim.* 21, 2, Article 33 (May 2024), 26 pages. DOI: https://doi.org/10.1145/3648475

[82] Hyemi Min, Jungyoon Kwon, and Bernhard Egger. 2023. Flexer: Out-of-order scheduling for multi-NPUs. In *21st ACM/IEEE International Symposium on Code Generation and Optimization (CGO'23)*. Association for Computing Machinery, New York, NY, USA, 212–223.

[83] Jayashree Mohan, Amar Phanishayee, Janardhan Kulkarni, and Vijay Chidambaram. 2022. Looking beyond GPUs for DNN scheduling on multi-tenant clusters. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI'22)*. USENIX Association, 579–596.

[84] Seonjin Na, Jungwoo Kim, Sunho Lee, and Jaehyuk Huh. 2024. Supporting secure multi-GPU computing with dynamic and batched metadata management. In *IEEE International Symposium on High-Performance Computer Architecture (HPCA'24)*. 204–217.

[85] Supun Nakandala, Yuhao Zhang, and Arun Kumar. 2020. Cerebro: A data system for optimized deep learning model selection. *VLDB Endow.* 13, 12 (2020), 2159–2173.

[86] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro et al. 2021. Efficient large-scale language model training on GPU clusters using Megatron-LM. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC'21)*. 1–15.

[87] Thomas Norrie, Nishant Patil, Doe Hyun Yoon, George Kurian, Sheng Li, James Laudon, Cliff Young, Norman Jouppi, and David Patterson. 2021. The design process for Google's training chips: TPUv2 and TPUv3. *IEEE Micro* 41, 2 (2021), 56–63.

[88] Young H. Oh, Seonghak Kim, Yunho Jin, Sam Son, Jonghyun Bae, Jongsung Lee, Yeonhong Park, Dong Uk Kim, Tae Jun Ham, and Jae W. Lee. 2021. Layerweaver: Maximizing resource utilization of neural processing units via layer-wise scheduling. In *IEEE International Symposium on High-Performance Computer Architecture (HPCA'21)*. 584–597.

[89] Angshuman Parashar, Minsoo Rhu, Anurag Mukkara, Antonio Puglielli, Rangharajan Venkatesan, Brucek Khailany, Joel Emer, Stephen W. Keckler, and William J. Dally. 2017. SCNN: An accelerator for compressed-sparse convolutional neural networks. In *44th Annual International Symposium on Computer Architecture (ISCA'17)*. Association for Computing Machinery, New York, NY, USA, 27–40. DOI: https://doi.org/10.1145/3079856.3080254

[90] Jason Jong Kyu Park, Yongjun Park, and Scott Mahlke. 2017. Dynamic resource management for efficient utilization of multitasking GPUs. *SIGPLAN Not.* 52, 4 (Apr. 2017), 527–540.

[91] Manos Pavlidakis, Stelios Mavridis, Antony Chazapis, Giorgos Vasiliadis, and Angelos Bilas. 2022. Arax: A runtime framework for decoupling applications from heterogeneous accelerators. In *13th Symposium on Cloud Computing (SoCC'22)*. Association for Computing Machinery, New York, NY, USA, 1–15. DOI: https://doi.org/10.1145/3542929.3563467

[92] Manos Pavlidakis, Giorgos Vasiliadis, Stelios Mavridis, Anargyros Argyros, Antony Chazapis, and Angelos Bilas. 2024. Guardian: Safe GPU sharing in multi-tenant environments. In *25th International Middleware Conference (MIDDLEWARE'24)*. Association for Computing Machinery, New York, NY, USA, 313–326. DOI: https://doi.org/10.1145/3652892.3700768

[93] Yajuan Peng, Shuang Chen, Yi Zhao, and Zhibin Yu. 2024. UFO: The ultimate QoS-Aware core management for virtualized and oversubscribed public clouds. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI'24)*. USENIX Association, 1511–1530.

[94] Yifan Qiao, Shu Anzai, Shan Yu, Haoran Ma, Yang Wang, Miryung Kim, and Harry Xu. 2024. ConServe: Harvesting GPUs for low-latency and high-throughput large language model serving. *arXiv preprint arXiv:2410.01228* (2024).

[95] Eric Qin, Ananda Samajdar, Hyoukjun Kwon, Vineet Nadella, Sudarshan Srinivasan, Dipankar Das, Bharat Kaul, and Tushar Krishna. 2020. SIGMA: A sparse and irregular GEMM accelerator with flexible interconnects for DNN training. In *IEEE International Symposium on High Performance Computer Architecture (HPCA'20)*. IEEE, 58–70.

[96] Kiran Ranganath, Joshua D. Suetterlein, Joseph B. Manzano, Shuaiwen Leon Song, and Daniel Wong. 2021. MAPA: Multi-accelerator pattern allocation policy for multi-tenant GPU servers. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC'21)*. Association for Computing Machinery, New York, NY, USA, Article 99, 14 pages.

[97] Jinke Ren, Guanding Yu, and Guangyao Ding. 2021. Accelerating DNN training in wireless federated edge learning systems. *IEEE J. Select Areas Commun.* 39, 1 (2021), 219–232.

[98] 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 *49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'16)*. 1–13.

[99] Urvij Saroliya, Eishi Arima, Dai Liu, and Martin Schulz. 2023. Hierarchical resource partitioning on modern GPUs: A reinforcement learning approach. In *IEEE International Conference on Cluster Computing (CLUSTER'23)*. 185–196.

[100] Shaden Smith, Mostofa Patwary, Brandon Norick, Patrick LeGresley, Samyam Rajbhandari, Jared Casper, Zhun Liu, Shrimai Prabhumoye, George Zerveas, Vijay Korthikanti et al. 2022. Using DeepSpeed and Megatron to train Megatron-Turing NLG 530B, a large-scale generative language model. *arXiv preprint arXiv:2201.11990* (2022).

[101] Jakob Stangl, Thomas Lorünser, and Sai Manoj Pudukotai Dinakarrao. 2018. A fast and resource efficient FPGA implementation of secret sharing for storage applications. In *Design, Automation & Test in Europe Conference & Exhibition (DATE'18)*. IEEE, 654–659.

[102] Jakob Stangl, Thomas Lorünser, and Sai Manoj Pudukotai Dinakarrao. 2018. A fast and resource efficient FPGA implementation of secret sharing for storage applications. In *Design, Automation & Test in Europe Conference & Exhibition (DATE'18)*. 654–659. DOI: https://doi.org/10.23919/DATE.2018.8342091

[103] Foteini Strati, Xianzhe Ma, and Ana Klimovic. 2024. Orion: Interference-aware, fine-grained GPU sharing for ML applications. In *19th European Conference on Computer Systems (EuroSys'24)*. Association for Computing Machinery, New York, NY, USA, 1075–1092.

[104] Biao Sun, Ziming Huang, Hanyu Zhao, Wencong Xiao, Xinyi Zhang, Yong Li, and Wei Lin. 2024. Llumnix: Dynamic scheduling for large language model serving. *arXiv preprint arXiv:2406.03243* (2024).

[105] Qingxiao Sun, Liu Yi, Hailong Yang, Mingzhen Li, Zhongzhi Luan, and Depei Qian. 2022. QoS-aware dynamic resource allocation with improved utilization and energy efficiency on GPU. *Parallel Comput.* 113 (2022), 102958.

[106] 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 arXiv:2109.11067* (2021).

[107] Huailiang Tan, Yanjie Tan, Xiaofei He, Kenli Li, and Keqin Li. 2019. A virtual multi-channel GPU fair scheduling method for virtual machines. *IEEE Trans. Parallel Distrib. Syst.* 30, 2 (2019), 257–270.

[108] Shanquan Tian, Wenjie Xiong, Ilias Giechaskiel, Kasper Rasmussen, and Jakub Szefer. 2020. Fingerprinting cloud FPGA infrastructures. In *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA'20)*. Association for Computing Machinery, New York, NY, USA, 58–64. DOI: https://doi.org/10.1145/3373087.3375322

[109] Yash Ukidave, Xiangyu Li, and David Kaeli. 2016. Mystic: Predictive scheduling for GPU based cloud servers using machine learning. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS'16)*. 353–362. DOI: https://doi.org/10.1109/IPDPS.2016.73

[110] Stavros Volos, Kapil Vaswani, and Rodrigo Bruno. 2018. Graviton: Trusted execution environments on GPUs. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI'18)*. USENIX Association, 681–696. Retrieved from https://www.usenix.org/conference/osdi18/presentation/volos

[111] Ne Wang, Ruiting Zhou, Ling Han, Hao Chen, and Zongpeng Li. 2023. Online scheduling of distributed machine learning jobs for incentivizing sharing in multi-tenant systems. *IEEE Trans. Comput.* 72, 3 (2023), 653–667.

[112] Shaoqi Wang, Oscar J. Gonzalez, Xiaobo Zhou, Thomas Williams, Brian D. Friedman, Martin Havemann, and Thomas Woo. 2020. An efficient and non-intrusive GPU scheduling framework for deep learning training systems. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC'20)*. IEEE, 1–13.

[113] Xiaying Wang, Michele Magno, Lukas Cavigelli, and Luca Benini. 2020. FANN-on-MCU: An open-source toolkit for energy-efficient neural network inference at the edge of the Internet of Things. *IEEE Internet Things J.* 7, 5 (2020), 4403–4417.

[114] Xuhang Wang, Zhuoran Song, and Xiaoyao Liang. 2023. RealArch: A real-time scheduler for mapping multi-tenant DNNs on multi-core accelerators. In *IEEE 41st International Conference on Computer Design (ICCD'23)*. 158–165.

[115] Qizhen Weng, Wencong Xiao, Yinghao Yu, Wei Wang, Cheng Wang, Jian He, Yong Li, Liping Zhang, Wei Lin, and Yu Ding. 2022. MLaaS in the wild: Workload analysis and scheduling in large-scale heterogeneous GPU clusters. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI'22)*. USENIX Association, 945–960.

[116] 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 Annual Technical Conference (USENIX ATC'23)*. USENIX Association, 995–1008.

[117] Bingyang Wu, Zili Zhang, Zhihao Bai, Xuanzhe Liu, and Xin Jin. 2023. Transparent GPU sharing in container clouds for deep learning workloads. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI'23)*. USENIX Association, 69–85.

[118] Wencong Xiao, Romil Bhardwaj, Ramachandran Ramjee, Muthian Sivathanu, Nipun Kwatra, Zhenhua Han, Pratyush Patel, Xuan Peng, Hanyu Zhao, Quanlu Zhang et al. 2018. Gandiva: Introspective cluster scheduling for deep learning. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI'18)*. USENIX Association, 595–610. Retrieved from https://www.usenix.org/conference/osdi18/presentation/xiao

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

[120] 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), 812–827.

[121] Xin Xu, Na Zhang, Michael Cui, Michael He, and Ridhi Surana. 2019. Characterization and prediction of performance interference on mediated passthrough GPUs for interference-aware scheduler. In *11th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud'19)*. USENIX Association.

[122] Zichuan Xu, Liqian Zhao, Weifa Liang, Omer F. Rana, Pan Zhou, Qiufen Xia, Wenzheng Xu, and Guowei Wu. 2020. Energy-aware inference offloading for DNN-driven applications in mobile edge clouds. *IEEE Trans. Parallel Distrib. Syst.* 32, 4 (2020), 799–814.

[123] Mochi Xue, Jiacheng Ma, Wentai Li, Kun Tian, Yaozu Dong, Jinyu Wu, Zhengwei Qi, Bingsheng He, and Haibing Guan. 2018. Scalable GPU virtualization with dynamic sharing of graphics memory space. *IEEE Trans. Parallel Distrib. Syst.* 29, 8 (2018), 1823–1836.

[124] Yuqi Xue, Yiqi Liu, Lifeng Nai, and Jian Huang. 2023. V10: Hardware-assisted NPU multi-tenancy for improved resource utilization and fairness. In *50th Annual International Symposium on Computer Architecture (ISCA'23)*. Association for Computing Machinery, New York, NY, USA, Article 24, 15 pages.

[125] Yi Yang, Ping Xiang, Mike Mantor, Norm Rubin, and Huiyang Zhou. 2012. Shared memory multiplexing: A novel way to improve GPGPU throughput. In *21st International Conference on Parallel Architectures and Compilation Techniques (PACT'12)*. Association for Computing Machinery, New York, NY, USA, 283–292.

[126] Jianguo Yao, Qiumin Lu, Run Tian, Keqin Li, and Haibing Guan. 2023. An economy-oriented GPU virtualization with dynamic and adaptive oversubscription. *IEEE Trans. Comput.* 72, 5 (2023), 1371–1383.

[127] Shuochao Yao, Jinyang Li, Dongxin Liu, Tianshi Wang, Shengzhong Liu, Huajie Shao, and Tarek Abdelzaher. 2020. Deep compressive offloading: Speeding up neural network inference by trading edge computation for network latency. In *18th Conference on Embedded Networked Sensor Systems*. 476–488.

[128] 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 (Jan. 2024), 38 pages.

[129] 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 *29th International Symposium on High-Performance Parallel and Distributed Computing (HPDC'20)*. Association for Computing Machinery, New York, NY, USA, 173–184.

[130] 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 arXiv:2203.09040* (2022).

[131] Hangchen Yu, Arthur Michener Peters, Amogh Akshintala, and Christopher J. Rossbach. 2020. AvA: Accelerated virtualization of accelerators. In *25th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'20)*. Association for Computing Machinery, New York, NY, USA, 807–825.

[132] Peifeng Yu and Mosharaf Chowdhury. 2020. Fine-grained GPU sharing primitives for deep learning applications. In *Proceedings of Machine Learning and Systems*, I. Dhillon, D. Papailiopoulos, and V. Sze (Eds.), Vol. 2. 98–111.

[133] Qi Yu, Bruce Childers, Libo Huang, Cheng Qian, Hui Guo, and Zhiying Wang. 2020. Coordinated page prefetch and eviction for memory oversubscription management in GPUs. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS'20)*. 472–482.

[134] Deze Zeng, Andong Zhu, Lin Gu, Peng Li, Quan Chen, and Minyi Guo. 2023. Enabling efficient spatio-temporal GPU sharing for network function virtualization. *IEEE Trans. Comput.* 72, 10 (2023), 2963–2977.

[135] Shulin Zeng, Guohao Dai, Niansong Zhang, Xinhao Yang, Haoyu Zhang, Zhenhua Zhu, Huazhong Yang, and Yu Wang. 2023. Serving multi-DNN workloads on FPGAs: A coordinated architecture, scheduling, and mapping perspective. *IEEE Trans. Comput.* 72, 5 (2023), 1314–1328. DOI: https://doi.org/10.1109/TC.2022.3214113

[136] Bingyi Zhang, Hanqing Zeng, and Viktor K. Prasanna. 2023. GraphAGILE: An FPGA-based overlay accelerator for low-latency GNN inference. *IEEE Trans. Parallel Distrib. Syst.* 34, 9 (2023), 2580–2597.

[137] Feng Zhang, Zheng Chen, Chenyang Zhang, Amelie Chi Zhou, Jidong Zhai, and Xiaoyong Du. 2021. An efficient parallel secure machine learning framework on GPUs. *IEEE Trans. Parallel Distrib. Syst.* 32, 9 (2021), 2262–2276.

[138] Haoyang Zhang, Yirui Zhou, Yuqi Xue, Yiqi Liu, and Jian Huang. 2023. G10: Enabling an efficient unified GPU memory and storage architecture with smart tensor migrations. In *56th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'23)*. Association for Computing Machinery, New York, NY, USA, 395–410.

[139] Kai Zhang, Bingsheng He, Jiayu Hu, Zeke Wang, Bei Hua, Jiayi Meng, and Lishan Yang. 2018. G-NET: Effective GPU sharing in NFV systems. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI'18)*. USENIX Association, 187–200.

[140] Shijin Zhang, Zidong Du, Lei Zhang, Huiying Lan, Shaoli Liu, Ling Li, Qi Guo, Tianshi Chen, and Yunji Chen. 2016. Cambricon-X: An accelerator for sparse neural networks. In *49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'16)*. IEEE, 1–12.

[141] Chen Zhao, Wu Gao, Feiping Nie, and Huiyang Zhou. 2022. A survey of GPU multitasking methods supported by hardware architecture. *IEEE Trans. Parallel Distrib. Syst.* 33, 6 (2022), 1451–1463.

[142] Xiaohui Zhao, Jianguo Yao, Ping Gao, and Haibing Guan. 2018. Efficient sharing and fine-grained scheduling of virtualized GPU resources. In *IEEE 38th International Conference on Distributed Computing Systems (ICDCS'18)*. 742–752.

[143] Yihao Zhao, Yuanqiang Liu, Yanghua Peng, Yibo Zhu, Xuanzhe Liu, and Xin Jin. 2022. Multi-resource interleaving for deep learning training. In *ACM SIGCOMM Conference (SIGCOMM'22)*. Association for Computing Machinery, New York, NY, USA, 428–440.

[144] Zhuangdi Zhu, Alex X. Liu, Fan Zhang, and Fei Chen. 2018. FPGA resource pooling in cloud computing. *IEEE Trans. Cloud Comput.* 9, 2 (2018), 610–626.

Received 28 August 2024; revised 14 January 2025; accepted 24 February 2025