A Survey of *Multi-Tenant* Deep Learning Inference on GPU

Fuxun Yu[†], Di Wang[‡], Longfei Shangguan[‡], Minjia Zhang[‡], Chenchen Liu[§], Tolga Soyata[†], Xiang Chen[†]
[†]George Mason University, [‡]Microsoft, [§]University of Maryland, Baltimore County

***Abstract*—Deep Learning (DL) models have achieved superior performance. Meanwhile, the computing hardware like NVIDIA GPUs also demonstrated strong computing scaling trends with 2× throughput and memory bandwidth for each generation. With such strong computing scaling of GPUs, *multi-tenant deep learning inference* by co-locating multiple DL models onto the same GPU become widely deployed to improve resource utilization, enhance serving throughput, and reduce energy cost, etc. However, achieving efficient multi-tenant DL inference is challenging which requires thorough full-stack system optimization. Previous surveys either target at summarizing single tenant deep learning inference optimizations, or only focus on certain single optimization layer alone, such as graph-level, kernel-level, etc. This survey aims to summarize and categorize the emerging challenges and optimization opportunities for multi-tenant DL inference on GPU. By overviewing the entire optimization stack, summarizing the multi-tenant computing innovations, and elaborating the recent technique advances, we hope that this survey could shed light on new optimization perspectives and motivate novel works in future large-scale DL inference system optimization.**

I. Introduction

DL Application and Computing Trends Deep Learning (DL) models have achieved superior performance in cognitive tasks like vision, speech and language domain, and have been adopted in medical analysis, machine translation, product recommendation, *etc.* The momentum of DL-based intelligence has appealed millions of users and created a wide-spectrum of cloud & edge applications like VR/AR games, intelligent robots and vehicles, large-scale recommendation systems, and even metaverse applications [12], many of which are featured with *multi-modality, multi-tasking and substantial task complexity*, as shown in Figure 1 (a).

The emergence of such massive DL applications motivates the adoption of DL accelerators, especially GPUs, in both cloud and edge. According to the report [33], GPUs accounted for 85% of the $2.98B cloud data center accelerator market in 2018. The edge hardware market, with the emerging smart manufacturing, surveillance applications, is also projected to grow from $920M in 2021 to $2,080M by 2026 and the edge GPUs are also taking a steady growth to more than 50% market share with Nvidia Jetson, TX2, Xavier, Orin, etc.

Within such trends, the capacity of recent generations of GPUs demonstrates exponential growing speed.[1] From K80, P40, and P100 to recent T4, V100, A100 architectures, GPUs maintain a trend of doubling performance. The last generation of V100 [6] offers 120 Tera floating point operations per second (TFLOPS) and 900 GB/s memory bandwidth, and the numbers further increase to 312 TFLOPS and 1.6TB/s memory bandwidth for the newer A100 [5]. A100 reports the ResNet50 [14] inference speed of 36,436 images/second, showing the computing capacity that overwhelms the limited needs from conventional single DL model execution schemes. Therefore, with such scaling trends in both application complexity and GPU capacity, single model execution cannot fulfill the needs of application scenarios nor fully utilizing the GPUs.

Multi-Tenant DL Inference, as shown in Figure 1 (b), is *one promising solution to the aforementioned scenarios with multi-modality and multi-tasking needs by running mixed DL model workloads simultaneously on one powerful GPU to improve the utilization, throughput, and power efficiency, etc.*

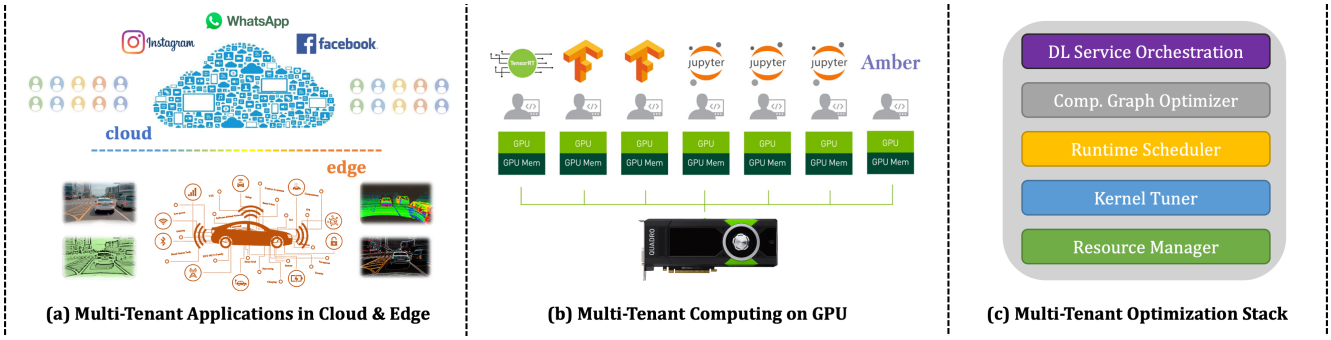
There are recently many emerging works that tackle the multi-tenant DL inference optimization on GPUs. These works usually take single optimization point of view drawing from traditional single-model optimization experiences, e.g., either from the DL model scheduling perspective [1, 9, 50], or the GPU resource management perspective [8, 19, 43]. However, achieving the ult-most efficiency for multi-tenant DL computing is more challenging as it needs to thoroughly consider the differences between single vs. multi-tenant DL inference, and requires multi-layer DL optimization or full-stack co-optimization. As so, there is a great need for a systematical review of opportunities and challenges on multi-tenant DL inference optimization.

Our survey is the first work that thoroughly analyzes the multi-tenant GPU scheduling problem, summarizes the major differences in single- vs. multi-tenant computing optimization, and reveals the emerging opportunities and potential benefits of multi-tenant DL inference on GPUs. To ease the understanding, our work also draws some experience from pevious DL computing stacks, and compares single vs multi-tenant DL inference on GPU from a hierarchical perspective.

Single vs Multi-Tenant: A Hierarchical Comparison. Traditional single-tenant DL compilers (Figure 1 (c)) already include multi-layer optimization: algorithm-level compression [22, 31, 51, 52], graph-level rewriting [15, 17, 54], runtime scheduling [9, 37] and kernel tuning [2, 10, 16, 46], etc. However, optimizations targeting at single-tenant are usually ill-fitted for multi-tenant inference with following examples.

From the top graph level, multi-tenant DL inference workloads represented in directed acyclic graphs (DAGs) come with significantly higher volume of multi-model operators

[1]Detailed scaling statistics of GPU capacities could be found in [42].



(a) Multi-Tenant Applications in Cloud & Edge (b) Multi-Tenant Computing on GPU (c) Multi-Tenant Optimization Stack

Fig. 1. The Emerging Trend of Multi-Tenant DL Computing on GPU.

than single model. This incurs many non-mergeable operators and exposes much larger scheduling space. Another example in the lower kernel level is TVM [2]. As one of the most competitive DNN optimization frameworks, TVM generates highly efficient code for heterogeneous hardware but with a built-in assumption of single-tenant execution setting, where the tuning strategies for the generated code aim to saturate SMs and memory bandwidth of the GPU. The assumption and single-tenant tuning however becomes unsuitable for multi-kernel concurrent execution with partial resource available for each kernel. According to [7], the maximum throughput gap comparing single-tenant vs. multi-tenant tuned configurations for the same computing kernel could reach $5\times$ difference.

Further down to the GPU hardware, multi-tenant DL inference requires many scheduling support to address problems like inter-tenant interference, such as dedicated GPU hardware primitives for resource partitioning, isolation and allocation, etc. With the current trends towards multi-tenant applications, GPU vendors like Nvidia have recently released many new features including Multi-Stream [24], Multi-Process Service (MPS) [27], Multi-Instance GPU (MIG) [26] and virtual GPUs (vCS) [28] to support both runtime scheduling and resource management, which exposes new research opportunities for multi-tenant DL scheduling and optimization.

With such differences considered, this work will adopt such a view of optimization stack to thoroughly analyze the multi-tenant challenges and introduce emerging works.

By reviewing the emerging challenges, opportunities, and research works on multi-tenant DL computing on the GPU, we hope this survey could motivate more design and innovations in this promising new domain. The remaining paper is organized as follows: Section II introduces the novel challenges and opportunities in multi-tenant GPU computing stack and a high-level overview of current vendor GPU support. Section III summarizes recent research works for multi-tenant computing in detail. We then give our vision and insights in Section IV. Section V concludes this paper.

II. Challenges & Opportunities for Multi-Tenant Computing on GPU

In this section, we first characterize the major differences between single- vs. multi-tenant DL computing optimization through the full DL computing stack. We then introduce the recently-released GPU features, such as Stream, MPS, MIG, which provide important fundamental backend support for multi-tenant computing optimization.

A. *Challenges for Multi-Tenant DL Computing*

Traditional DL computing optimization in full stack often expands in ❶ *service-level* orchestration [41], ❷ *graph-level* optimization [15, 49], ❸ *runtime-level* scheduling [9], ❹ *kernel-level* tuning [2, 45] and ❺ *resource-level* management [8, 19]. Although there are many previous works for computing optimization in these difference levels, multi-tenant computing shows dramatic characteristics that make these methods ill-fitted. According to the same optimization stack, we summarize the major differences in Table I and analyze the computing challenges in Figure 2.

❶ **Service-level:** AI-centric cloud services handle millions of service queries simultaneously [47]. With the massive computing capacity of GPUs, multiple DL services could be *strategically co-located for efficient concurrent execution*, which is one key difference between multi-tenant GPU computing versus traditional CPU multi-tasking. By allowing the resource sharing among concurrent DL workloads, the service providers could potentially improve the GPU resource utilization and reduce cost of ownership (COO) like infrastructure and power cost especially for large-scale data centers [26].

However, the challenges remains for strategic co-location like that the *inter-tenant interference* [19] could happen and degrade the quality of service such as service-level objectives (SLA) of tail latency and throughput. This could become worse with increased number of co-located workloads. Therefore, there are many recent *service-level orchestration* works [4, 19,

TABLE I
Challenges for Multi-Tenant Optimization.

Full Optimization Stack		Single-Tenant	Multi-Tenant
❶ Service-level	Co-location	No	Yes
	Interference	No	High Interference
❷ Graph-level	DAG(s)	Mostly Seq.	Seq. + Parallel
❸ Runtime-level	Parallelism	Limited	Extensive
	Complexity	Low	High
❹ Kernel-level	Resource Usage	Exclusive	Shared
	Tuning Objective	100% util.	x% partial util.
❺ Resource-level	Management	No	Resource Partition

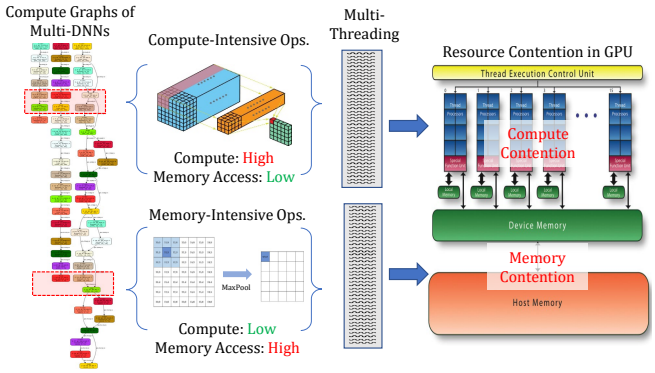


Fig. 2. Multi-Tenant DL Computing Challenges.

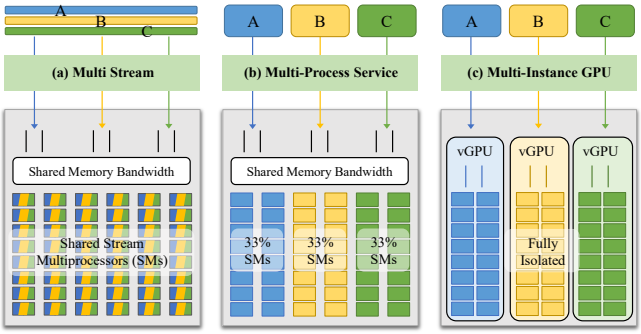


Fig. 3. Multi-Stream, MPS and MIG Illustration.

48] that design different heuristic-based, modeling-based or prediction-based mechanisms to conduct strategic co-location for efficient multi-tenant computing on GPUs.

❷ **Graph-level:** DNNs with many operators are commonly represented as directed acyclic graphs (DAGs), which use nodes to represent operators and edges to represent the data flow and dependency [15]. Single-model DAGs are usually *sequential with limited parallelism* like VGG [39], ResNets [14], MobileNets [35] and EfficietNets [44], which have only one or two branches and thus exposes small scheduling space [21].

By contrast, multi-tenant DL computing with multiple parallel DAGs usually have *extensive inter-operator parallelism*, as shown in Figure 2 (left), which enables more flexible inter-operator scheduling [50]. Certain challenges exist in such graph scheduling such as the increased complexity with larger number of operators and search space, and more complex GPU resource contention analysis, as shown in Figue 2 (right).

❸ **Runtime-level:** Previously due to the limited inter-operator parallelism, only a few works [1, 9] touch upon runtime-level scheduling such as certain multi-branch models like Inception, NasNets and Transformers. These works leverage certain GPU runtime primitives (*e.g.*, Nvidia multi-stream [24]) for concurrent operator scheduling, many of which however incur large runtime overheads. For example, multi-stream synchronization forces all streams to wait/stall until the last stream finishes its workloads [24]. Multi-tenant scheduling tends to suffer more from such overheads with the increased number of operators and scheduling complexity. Due to the increased attention in GPU multi-tenant scheduling, GPU vendors have recently released a series of important features such as CUDA graphs [29] to address such scheduling overheads.

❹ **Kernel-level:** Kernel configurations such as loop tiling, thread blocking, memory colasing, *etc.* could significantly influence each operator's computing efficiency. Previous single-tenant kernel-level works like TVM [2] and TF-XLA [45] try to find the best configuration that can saturate the GPU resource, i.e., *exclusive resource usage*. However, as multi-tenant DNNs *share the underlying resource*, kernels optimized for single-tenant settings can easily become sub-optimal for multi-tenant scenarios. Recently, there are certain works that show multi-tenant DL computing should optimize kernel configurations according to its available resource ratio during practical execution [7], which shows a 5× throughput difference.

❺ **Resource-level:** To achieve *adaptive multi-tenant resource partitioning and provisioning*, it asks for both strategic design and hardware support. The first challenge for such adaptive resource provisioning lies in the *DL workload dynamics* [50]. Multiple DL models with different deep structures have highly diverse and non-stable computing/memory requirements, making the inter-model resource sharing and competition highly dynamic and thus hard to determine the optimal resource partitioning. On the other hand, adaptive resource management requires highly *flexible GPU partitioning and reconfiguration capability*. Although recently there emerges certain adaptive resource provisioning features (*e.g.*, Nvidia multi-process service, multi-instance GPU [26, 27]) that supports resource partitioning, the reconfiguration takes non-negligible time (*e.g.*, several ms), which is a major limitation of the recent resource scheduling works [8, 19].

B. Emerging Multi-Tenant Computing Opportunities

Previously, one important reason that hinders the adoption and development of multi-tenant DL computing on GPU is the insufficient hardware scheduling mechanisms. But with the increasing attention in this topic, GPU vendors like Nvidia release certain new GPU multi-tenant features to support multi-tenant DL inference, which provides *great opportunities for multi-tenant scheduling*. The multi-tenant GPU scheduling features could be categorized into two major types: software-level and hardware-level support.

❶ **Software Approach:** The very first GPU multi-tasking feature is the ***Multi-Stream*** mechanism [24] supported in the Fermi GPU architecture (Figure 3 (a)). As a software-based programming model, a stream can contain a sequence of issued operations that execute on the GPU. Operations in different streams could run concurrently and share the underlying GPU resources like SMs [24]. The similar feature ***Hyper-Q*** [23] is introduced in the Kepler GPU architecture (2013) that expands previous 16-way to 32-way hardware kernel queues for higher concurrency support. Along with the concurrency support, the CUDA library also releases certain scheduling APIs like DeviceSync, StreamWait, etc. to support more fine-grained

TABLE II
GPU SUPPORT FOR MULTI-TENANT COMPUTING.

Mechanisms	Stream	MPS	MIG
Partition Type	No	Logical	Physical
Max Partition	Unlimited	48	7
SM Isolation	No	By Percentage	Yes
Mem BW Isolation	No	No	Yes
Performance QoS	No	Partial	Yes
Reconfiguration	Dynamic	Process Launch	When Idle

scheduling capability [25]. These software-level APIs provide valuable multi-tenant GPU scheduling mechanisms, based on which many recent works have started to explore the fine-grained DL operator-level scheduling techniques [9, 50].

❷ **Hardware Approach:** Besides the software support, NVIDIA recently also releases advanced hardware-level resource management mechanisms to support *flexible resource allocation, isolation and virtualization*. These resource management methods can be categorized into two types: *logical* and *physical*. ***Multi-Process Service (MPS)*** [27] is a logical resource partitioning mechanism (Figure 3 (b)) that allows user to partition the streaming multi-processors (SMs) and allocate them to different processes, for example, 30%, 70% to two concurrent processes. Such partition is done by the software-based process-to-SM mapping scheduling and thus considered logical. Notably, although MPS enables logical SM partitioning, other GPU resources like memory bandwidth are not partitioned and thus MPS cannot fully avoid the inter-process resource competition and interference. To address this, the recently introduced ***Multi-Instance GPU (MIG)*** [26] on Ampere architecture enables physical partitioning of both SMs and memory bandwidths through dedicated GPU architecture design (Figure 3 (c)). Such physical partition ensures fully isolated resources, and thus no interference can happen between different processes. MIG support splitting one A100 GPU into seven fully isolated GPU instances. Meanwhile, it provides certain *reconfiguration capability* when the GPU is fully or partial idle. For example, one A100 could be split into three instances with the ratio of 4:2:1 and then reconfigured to be 3:3:1, etc [26]. More detailed comparison of Stream, MPS, MIG could be found in Table II.

III. MULTI-TENANT COMPUTING OPTIMIZATION: DESIGN AND INNOVATIONS

Based on the former challenges and opportunities, we review the emerging works tackling the multi-tenant scheduling optimization from different perspectives. We summarize these works in Table III. From a top-down view, these works are categorized into several levels, *i.e.*, from *DL service-level orchestration*, *graph & runtime-level scheduling*, to *kernel-level auto-tuing* and then *GPU resource-level management*.

A. *Service-level Orchestration*

DL service-level orchestration is an important feature in large-scale data centers to improve the GPU utilization. As the top-most scheduling level, such orchestration usually regards one service query as the basic scheduling unit. This reduces the scheduling complexity as there is no need to consider the intra-DNN model structure details (operators and graphs). One example is the Microsoft Deep Learning Inference Service (DLIS) system [41]. The service orchestrator characterizes different models' resource requirements and then strategically places one or multiple queries onto hosts through the service router. Therefore, it could maximize the served queries per second (QPS) while ensure little inter-query interference so as to maintain similar tail latency.

However, designing a proper co-location strategy or system is a non-trivial task. For example, one challenging factor is the serving dynamics, *i.e.*, undetermined arrival rates and/or distribution of incoming DL queries, different RNN/LSTMs queries with varied inputs and control states. Distinct from static workloads that we can get the full information, such dynamic scenarios require us to either utilize historical data or predict the future workload dynamics. PREMA [4] proposed a predictive multi-task DNN scheduling algorithm that combines off-line records and online token-based job scheduling to determine the best multi-tenant co-location strategy.

Another challenge in multi-tenant co-location is how to accurately predict the inter-model resource interference. This is a critical factor in ensuring QoS such as tail latency. [19] trained a ML-based latency degradation predictor under co-location using offline-profiled hardware-level features such as SM and DRAM usage, PCIe read/write BW, buffer usage, etc. Then the latency degradation predictor is used to evaluate the model placement's potential influence for each query.

However, these works have certain scalability issues as they mostly targeted at static model types, hardware types, etc., which may not be suitable for dynamic workloads. Meanwhile, as each DNN can have many operators (*e.g.*, layers) that have fluctuated resource consumption, such coarse-grained scheduling (with one entire query as the basic unit) may suffer from resource under-utilization/contention occasionally and thus still hinders the QoS.

B. *Graph and Runtime-level Scheduling*

Graph and runtime-level scheduling could help address one of the aforementioned challenge of coarse-grained granularity by enabling more fine-grained scheduling, *e.g.*, the DNN operators. This could be done by leveraging the GPU software-level support such as the multi-stream mechanism and scheduling APIs. For example, [50] propose an ML-based scheduling strategy for multi-tenant DNN exeuction acceleration. It first abstracts multiple DNN's computation graph with all operators into a global intermediate representation (IR), which enables flexible resource sharing between different tenants so as to improve the utilization. To find the optimal concurrent operator execution strategy in the huge scheduling space, they design a ML-based auto-search method by defining three main factors:

TABLE III
RECENT WORKS ON MULTI-TENANT COMPUTING OPTIMIZATION (JCT: JOB COMPLETION TIME, SLA: SERVICE-LEVEL AGREEMENT).

Ref.	Hardware	Perspective	Algorithm/Strategy	Improvement/Achievement
Inter-Aware [19]	GPU	DL Service-level Orchestration	• ML-based Interference Predictor • Proactive Query Scheduler	• Reducing Job Interference • Enhancing Serving Throughout
Irina [48]	GPU	DL Service-level Orchestration	• Online Query Scheduler • Heuristic-based Preemption • Concurrent Execution & Batching	• Reducing Client-Side JCT
PREMA [4]	NPU	DL Service-level Orchestration	• Online Query Scheduler • Heuristic-based Preemption	• Reduced High-Priority Job JCT • Maintaining Low-Priority SLA
Runtime-Aware [50]	GPU	Graph & Runtime-level Scheduling	• Multi-Model DAG Rewriting • ML-based Scheduling Search • Multi-Stream Runtime Scheduling	• Reduced Inference Latency
Spatial-Tune [7]	GPU	Kernel-level Auto-Tuning	• MPS-based Resource Allocation • Partial-Resource Kernel Tuning	• Enhanced Kernel Performance • Reduced Inter-kernel Interference
GSlice [8]	GPU	Resource-level Management	• MPS-based Resource Partitioning • Adaptive Batching	• Enhanced Serving Throughput • Maintaining SLA
Spatial-Partition [3]	GPU	Resource-level Management	• MPS-based Resource Partitioning • Interference-aware Scheduling	• Enhanced Serving Throughput • Maintaining SLA
MIG-Serving [43]	GPU	Resource-level Management	• MIG-based Resource Reconfiguration • Fast & Slow Query Scheduling	• Enhanced Serving Throughput • Maintaining SLA
Planaria [13]	Systolic Arrays	Resource-level Management	• Architecture Reconfiguration	• Enhanced Serving Throughput • Reduced Energy Consumption

scheduling search space, profiling-guided latency cost model, and the ML search algorithm. Based on offline profiling records, the search algorithm could find the best scheduling for optimal GPU utilization and throughput.

Such graph and runtime-level operator scheduling could usually achieve better performance due to the fine-grained design, but they also face more scalability issues, *e.g.*, when the number of co-located workloads increase to very large. Meanwhile, it also applies to static or known multi-tenant workload only, which cannot address dynamic model types.

C. *Resource-Level Management*

Besides the aforementioned works, another optimization perspective to solve the inter-tenant inference is to conduct fine-grained resource managing [8, 13]. For example, spatial partitioning and allocation of GPU resources to different DL workloads could isolate different jobs' resource (*e.g.*, stream multiprocessors (SMs), memory bandwidths), thus avoiding the job interference in the hardware resource level. However, as we introduced before, achieving fine-grained resource partitioning is non-achievable until recently GPU vendors release a series of resource sharing and partitioning support like multi-streams, multi-process services (MPS [27]) and multi-instance GPU (MIG [26]). Most recent resource-level management works are built upon these technologies.

For example, GSlice [8] uses MPS to conduct adaptive SM partitioning for different DNN jobs. They design a self-learning method to dynamically adjust the GPU resource allocation ratio for each workload and thus avoid interference among co-located DL workloads and maximize the throughput. [3] utilizes similar spatial partitioning mechanism by MPS while additionally combining temporal scheduling strategies. MIG-Serving [43] is the most recent work that adopts the newly-released MIG feature on A100 to achieve spatial resource management for multi-tenant scheduling.

However, such spatial resource partitioning solutions also have an intrinsic limitation that is the *costly re-configuration* when the workloads change and requires resource partitioning adjustment. For GPUs, re-configuring the resource partitioning requires certain amount of time (*e.g.*, tens of ms or more), which can be even larger than one DL inference workloads' processing time. Therefore, re-configuring frequently is not practical and thus limits such solutions' performance when facing dynamic workloads. [8] tries to reduce the stall caused by reconfiguration time of MPS by utilizing a standby/shadow process. However, the minimum time for switching one partitioning configuration to another one still cost several seconds, which is still non-negligible in online serving.

D. *Potential Directions for Remaining Challenges*

❶ **ML-based Prediction and Online Learning:** To address the problem of service dynamics, using ML-based predictive model (*e.g.*, reinforcement learning, LSTM, *etc.*) is one promising direction, which can potentially predict the future queries trend and guide the overall scheduling. The ML-based model can be initially trained offline by historical serving records. During the online serving process, active learning and continual learning [30, 32] using the latency/throughput as feedback can be potentially utilized to improve the predictive accuracy and the scheduling effectiveness consistently.

Another way of leveraging ML-based prediction is to conduct light-weight modeling to predict the latency degradation under different multi-model and hardware combinations so that the scheduler can make better decision regarding the latency SLA constraints. For example, the work [19] built a ML model to predict the latency of multi-model inference cases on different machines. As the effectiveness of the final scheduling solution highly depends on the modeling accuracy, the scalability and generality issue across hardware/model types needs to be addressed, which can be also very challenging.

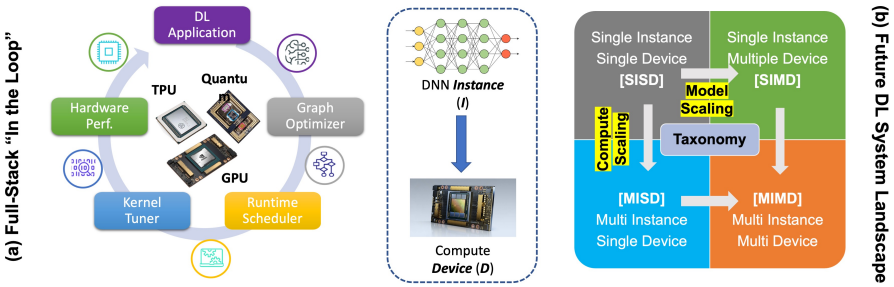


Fig. 4. The future trends to a larger scale DL system.

❷ **Software-Hardware Co-Scheduling:** The software and hardware scheduling could be complementary to provide both high job scheduling flexibility and strict resource isolation. There are some recent works that adopt such a temporal-spatial combined perspective. [3] uses MPS to conduct resource partitioning and then implements a heuristic-based task scheduler to find the appropriate mapping between the DNN queries and gpu partitions. In addition to that, software-hardware scheduling could also be leveraged to alleviate certain reconfiguration overhead. For example, it's potential to conduct software-based scheduling within a partitioned GPU slice, *e.g.*, combining multi-stream with MIG. In this way, fine-grained scheduling could be achieved without re-partitioning the entire GPU, avoiding the reconfiguration overhead.

IV. Towards Large-Scale DL Computing: Vision and Insights

A. *Architecture Design with "Full Stack in the Loop"*

The fast development of multi-tenant DL computing brings many challenges for the system stack optimizations. Besides for the GPU only, this also enlightens the other DL-oriented hardware architecture designers (*e.g.*, TPU, chiplet, neuromorphic and quantum-based accelerators) to optimize for *flexibility and agility* facing a rapidly changing DL application landscape. Specifically, one important future trend is the *"full stack in the loop"*, i.e., to remove the boundaries in the vertical DL system stack and conduct full-stack integration to strive for both optimal performance and flexibility.

One example in this line of efforts is the DL compiler renovation by tvm unity [34], as shown in Figure 4 (a). Current DL computing stack conducts separate layer-wise optimization (graph-runtime-kernel-resource) and single directional *top-down* deployment. This however prohibits feedback loops and cross-layer interactions for SW/HW co-compiling based on model workloads and hardware characteristics. Unifying the abstraction between layers would thus greatly facilitate the new full-stack optimization as a loop, not only for multi-tenant computing, but also for future wider DL application.

Another example is the increasing attention in the versatile and flexible chiplet-based SW/HW co-design [36, 53] that uses multi-chip-modules (MCMs). Compared to traditional large monolithic die, such MCM combines smaller chiplets into a larger system and substantially reduces fabrication and design costs. However, it requires thorough application awareness to optimize the chip design and overall performance. As so, such chiplet modules could also greatly benefit from the full-stack-in-the-loop architecture of DL computing.

B. *The Future Large-Scale DL System Landscape*

Multi-tenant DL computing is a natural generalization result due to the significant *computing scaling* trend of GPU. However, recently another *model scaling* trend is observed, that is, designing and training super-scale AI models for general intelligence. For example, the recent SOTA giant AI model Megatron-NLG [40] has reached 530 billions of parameters and requires hundreds of GPUs to conduct multi-node distributed inference. If we take such *model scaling* into consideration, even more new DL computing modes could be observed and enrich the future DL & system landscape.

We describe the future large-scale DL system landscape by using a taxonomy in Figure 4 (b). Using Instance (I) to denote one DNN model and Device (D) to denote the hardware, traditional DL system mostly comes within the *Single Instance Single Device (SISD)* domain and only constitute the top-left quarter in the full spectrum. Multi-tenant computing emerges as the *Multiple Instances Single Device (MISD)* with the computing scaling trend, as we summarized in this servery.

Whereas diagonally, with the model scaling trend, the *Single Instance Multiple Devices (SIMD)* interaction mode also emerges and is attracting more attention such as distributed inference for super-scale giant models [11, 18, 20, 38] including language, recommendation models, etc. Finally, *Multiple Instances Multiple Devices (MIMD)* computing would eventually combine all these modes and become a practical needs for future DL-centric data center optimization.

V. Conclusion

DL-based intelligence creates a wide-spectrum of applications featured with substantial complexity like multi-modality and multi-tasking. GPU is one major type of DL accelerators and its gen-by-gen capacity shows exponential scaling. With the double scaling of application complexity and GPU capacity, multi-tenant DL inference emerges as an effective computing paradigm on GPU to enhance the resource utilization, throughput, and power efficiency. This survey categorizes the emerging optimization challenges and opportunities for multi-tenant DL inference on GPU following a hierarchical comparison with traditional single-tenant optimization. We hope that this survey could shed lights on new perspectives and novel works in future large-scale DL system optimization.

REFERENCES

[1] Yang Bai, Xufeng Yao, Qi Sun, and Bei Yu. 2021. AutoGTCO: Graph and Tensor Co-Optimize for Image Recognition with Transformers on GPU. In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*.

[2] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Haichen Shen, Meghan Cowan, Leyuan Wang, Yuwei Hu, Luis Ceze, et al. 2018. {TVM}: An automated end-to-end optimizing compiler for deep learning. In *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*. 578–594.

[3] Seungbeom Choi, Sunho Lee, Yeonjae Kim, Jongse Park, Youngjin Kwon, and Jaehyuk Huh. 2021. Multi-model Machine Learning Inference Serving with GPU Spatial Partitioning. *arXiv preprint arXiv:2109.01611* (2021).

[4] Yujeong Choi and Minsoo Rhu. 2020. Prema: A predictive multi-task scheduling algorithm for preemptible neural processing units. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*.

[5] Jack Choquette, Wishwesh Gandhi, Olivier Giroux, Nick Stam, and Ronny Krashinsky. 2021. Nvidia a100 tensor core gpu: Performance and innovation. *IEEE Micro* 41, 2 (2021), 29–35.

[6] NVIDIA Coorporation. 2017. *NVIDIA Tesla V100 GPU Architecture*. Technical Report. http://www.nvidia.com/object/volta-architecture.

[7] Aditya Dhakal, Junguk Cho, Sameer G Kulkarni, KK Ramakrishnan, and Puneet Sharma. 2020. Spatial Sharing of GPU for Autotuning DNN models. *arXiv preprint arXiv:2008.03602* (2020).

[8] Aditya Dhakal, Sameer G Kulkarni, and KK Ramakrishnan. 2020. Gslice: controlled spatial sharing of gpus for a scalable inference platform. In *Proceedings of the 11th ACM Symposium on Cloud Computing*. 492–506.

[9] Yaoyao Ding, Ligeng Zhu, Zhihao Jia, Gennady Pekhimenko, and Song Han. 2020. IOS: Inter-Operator Scheduler for CNN Acceleration. *arXiv preprint arXiv:2011.01302* (2020).

[10] Vincent Dumoulin and Francesco Visin. 2016. A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285* (2016).

[11] William Fedus, Barret Zoph, and Noam Shazeer. 2021. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *arXiv preprint arXiv:2101.03961* (2021).

[12] Peter Fernandez. 2022. Facebook, Meta, the metaverse and libraries. *Library Hi Tech News* (2022).

[13] Soroush Ghodrati, Byung Hoon Ahn, Joon Kyung Kim, Sean Kinzer, Brahmendra Reddy Yatham, Navateja Alla, Hardik Sharma, Mohammad Alian, Eiman Ebrahimi, Nam Sung Kim, et al. 2020. Planaria: Dynamic architecture fission for spatial multi-tenant acceleration of deep neural networks. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 681–697.

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

[15] Zhihao Jia, Oded Padon, James Thomas, Todd Warszawski, Matei Zaharia, and Alex Aiken. 2019. TASO: optimizing deep learning computation with automatic generation of graph substitutions. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*. 47–62.

[16] Andrew Lavin and Scott Gray. 2016. Fast algorithms for convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 4013–4021.

[17] Ao Li, Bojian Zheng, Gennady Pekhimenko, and Fan Long. 2020. Automatic horizontal fusion for GPU kernels. *arXiv preprint arXiv:2007.01277* (2020).

[18] Michael Lui, Yavuz Yetim, Özgür Özkan, Zhuoran Zhao, Shin-Yeh Tsai, Carole-Jean Wu, and Mark Hempstead. 2021. Understanding capacity-driven scale-out neural recommendation inference. In *2021 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 162–171.

[19] Daniel Mendoza, Francisco Romero, Qian Li, Neeraja J Yadwadkar, and Christos Kozyrakis. 2021. Interference-Aware Scheduling for Inference Serving. In *Proceedings of the 1st Workshop on Machine Learning and Systems*. 80–88.

[20] Nvidia Microsoft. 2020. *Using DeepSpeed and Megatron to Train Megatron-Turing NLG 530B, the World's Largest and Most Powerful Generative Language Model*. https://developer.nvidia.com/blog/using-deepspeed-and-megatron-to-train-megatron-turing-nlg-530b-the-world

[21] Wei Niu, Jiexiong Guan, Yanzhi Wang, Gagan Agrawal, and Bin Ren. 2021. DNNFusion: accelerating deep neural networks execution with advanced operator fusion. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*. 883–898.

[22] Wei Niu, Xiaolong Ma, Sheng Lin, Shihao Wang, Xuehai Qian, Xue Lin, Yanzhi Wang, and Bin Ren. 2020. Patdnn: Achieving real-time dnn execution on mobile devices with pattern-based weight pruning. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*. 907–922.

[23] NVIDIA. 2013. Hyper-Q. https://developer.download.nvidia.com/compute/DevZone/C/html_x64/6_Advanced/simpleHyperQ/doc/HyperQ.pdf.

[24] NVIDIA. 2015. CUDA Multi-Streams. https://developer.nvidia.com/blog/gpu-pro-tip-cuda-7-streams-simplify-concurrency/.

[25] NVIDIA. 2020. CUDA Programming Guide. https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html.

[26] NVIDIA. 2020. NVIDIA Multi Instance GPU (MIG). https://docs.nvidia.com/datacenter/tesla/mig-user-guide/.

[27] NVIDIA. 2020. NVIDIA Multi Process Service (MPS). https://docs.nvidia.com/deploy/pdf/CUDA-Multi-Process-Service-Overview.pdf.

[28] NVIDIA. 2020. NVIDIA Virtual Compute Server. https://www.nvidia.com/content/dam/en-zz/Solutions/design-visualization/solutions/resources/documents1/Technical-Brief-Multi-Instance-GPU-NVIDIA-Virtual-Compute-Server.pdf.

[29] NVIDIA. 2021. CUDA Graphs. https://developer.nvidia.com/blog/cuda-graphs/.

[30] German I Parisi, Ronald Kemker, Jose L Part, Christopher Kanan, and Stefan Wermter. 2019. Continual lifelong learning with neural networks: A review. *Neural Networks* 113 (2019), 54–71.

[31] Zhuwei Qin, Fuxun Yu, Chenchen Liu, and Xiang Chen. 2018. Functionality-oriented convolutional filter pruning. *arXiv preprint arXiv:1810.07322* (2018).

[32] Pengzhen Ren, Yun Xiao, Xiaojun Chang, Po-Yao Huang, Zhihui Li, Brij B Gupta, Xiaojiang Chen, and Xin Wang. 2021. A survey of deep active learning. *ACM Computing Surveys (CSUR)* 54, 9 (2021), 1–40.

[33] Market Reports. 2021. Global Data Center Accelerator Market Size, Status and Forecast 2020-2025. https://www.mynewsdesk.com/brandessence/pressreleases/data-center-accelerator-market-size-2021-cagr-38-dot-7-percent-3112488.

[34] Adrian Sampson, Tianqi Chen, and Jared Roesch. 2022. Apache TVM Unity: a vision for the ML software and hardware ecosystem. https://tvm.apache.org/2021/12/15/tvm-unity.

[35] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 4510–

4520.

[36] Yakun Sophia Shao, Jason Clemons, Rangharajan Venkatesan, Brian Zimmer, Matthew Fojtik, Nan Jiang, Ben Keller, Alicia Klinefelter, Nathaniel Pinckney, Priyanka Raina, et al. 2019. Simba: Scaling deep-learning inference with multi-chip-module-based architecture. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. 14–27.

[37] Haichen Shen, Jared Roesch, Zhi Chen, Wei Chen, Yong Wu, Mu Li, Vin Sharma, Zachary Tatlock, and Yida Wang. 2021. Nimble: Efficiently compiling dynamic neural networks for model inference. *Proceedings of MLSys* (2021).

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

[39] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[40] 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).

[41] Jonathan Soifer, Jason Li, Mingqin Li, Jeffrey Zhu, Yingnan Li, Yuxiong He, Elton Zheng, Adi Oltean, Maya Mosyak, Chris Barnes, et al. 2019. Deep learning inference service at microsoft. In *2019 {USENIX} Conference on Operational Machine Learning (OpML 19)*. 15–17.

[42] Yifan Sun, Nicolas Bohm Agostini, Shi Dong, and David Kaeli. 2019. Summarizing CPU and GPU design trends with product data. *arXiv preprint arXiv:1911.11313* (2019).

[43] Cheng Tan, Zhichao Li, and et al. 2021. Serving DNN Models with Multi-Instance GPUs: A Case of the Reconfigurable Machine Scheduling Problem. *arXiv:2109.11067* (2021).

[44] Mingxing Tan and Quoc V Le. 2019. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946* (2019).

[45] TensorFlow. 2020. TensorFlow XLA (Accelerated Linear Algebra). https://www.tensorflow.org/xla.

[46] Han Vanholder. 2016. Efficient inference with tensorrt. In *GPU Technology Conference*, Vol. 1. 2.

[47] Lukasz Wesolowski, Bilge Acun, Valentin Andrei, Adnan Aziz, Gisle Dankel, Christopher Gregg, Xiaoqiao Meng, Cyril Meurillon, Denis Sheahan, Lei Tian, et al. 2021. Datacenter-Scale Analysis and Optimization of GPU Machine Learning Workloads. *IEEE Micro* 41, 5 (2021), 101–112.

[48] Xiaorui Wu, Hong Xu, and Yi Wang. 2020. Irina: Accelerating DNN Inference with Efficient Online Scheduling. In *4th Asia-Pacific Workshop on Networking*. 36–43.

[49] Yichen Yang, Phitchaya Phothilimthana, Yisu Wang, Max Willsey, Sudip Roy, and Jacques Pienaar. 2021. Equality saturation for tensor graph superoptimization. *Proceedings of Machine Learning and Systems* 3 (2021), 255–268.

[50] Fuxun Yu and et al. 2021. Automated Runtime-Aware Scheduling for Multi-Tenant DNN Inference on GPU. In *Proceedings of the 40th IEEE International Conference on Computer Aided Design (ICCAD)*.

[51] Fuxun Yu, Chenchen Liu, Di Wang, Yanzhi Wang, and Xiang Chen. 2020. AntiDote: Attention-based Dynamic Optimization for Neural Network Runtime Efficiency. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*.

[52] Fuxun Yu, Zhuwei Qin, Di Wang, Ping Xu, Chenchen Liu, Zhi Tian, and Xiang Chen. 2020. DCCNN: computational flow redefinition for efficient cnn through structural decoupling. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1097–1102.

[53] Hao Zheng, Ke Wang, and Ahmed Louri. 2020. A versatile and flexible chiplet-based system design for heterogeneous many-core architectures. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[54] Lianmin Zheng, Chengfan Jia, Minmin Sun, Zhao Wu, Cody Hao Yu, Ameer Haj-Ali, Yida Wang, Jun Yang, Danyang Zhuo, Koushik Sen, et al. 2020. Ansor: Generating high-performance tensor programs for deep learning. In *14th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 20)*. 863–879.