A Survey of Multi-Tenant Deep Learning Inference on GPU

Fuxun Yu†, Di Wang‡, Longfei Shangguan†, Minjia Zhang‡, Chenchen Liu§, 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. 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 lights on new optimization perspectives and motivate novel works in future large-scale DL system optimization.

I. INTRODUCTION

Deep Learning (DL) models have achieved superior performance in cognitive tasks like vision, speech and language domain, and been adopted in medical analysis, machine translation, product recommendation, etc. This motivates great amounts of DL services, millions of users, and significant DL serving traffics to the computing system. For example, FB with 1.82 billions of daily active users can generate ads recommendation queries with 10M queries/second [13] that are served simultaneously by large-scale data centers.

The huge growth in customer-generated needs of DL services propels the demand for AI-centric data centers with the increased adoption of DL accelerators. Especially, GPU due to its massive parallel computing capacity has become the major type and maintains continuously increase in the cloud infrastructure. According to the report [40], GPU has accounted for the major share of 85% with 2983M USD in the global data center accelerator market in 2018. And this product segment is poised to reach 29819M USD by 2025.

GPU Capacity Scaling: With the strong market needs, the capacity of recent generations of GPUs also shows exponential growing, which demonstrates the overwhelming computing power compared to the common model workload. A concrete computing scaling summarization is shown in Figure 1(a). More detailed performance results could be found in [40]. From K80, P40, and P100 to recent T40, V100, A100, GPUs maintain the trend of doubling in performance. The last generation of V100 [7] 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 newly released A100 [6], which reports the ResNet50 [16] inference speed of 36,436 images/second and meanwhile provides continuously increasing energy efficiency.

Multi Tenant DL (MT-DL) Computing: Multi-tenant computing is motivated by such exponential computing scaling trends of GPUs. With the large computing capacity, surging amounts of DL users and service traffics, multi-tenant DL computing on GPUs is proposed by running mixed DL workloads simultaneously on the GPU. For example, multiple users could run varied DL workloads (inference, training, eager programming) by different DL frameworks (TensorFlow [1], PyTorch [32], TensorRT [43], etc). Since executing single DNN instance on such GPUs usually incurs resource under-utilization [50], multi-tenant computing mode could greatly improve the utilization, serving throughput and the energy efficiency (Perf/Watt), shining lights on both energy-constrained edge applications like autonomous driving system and AI-centric cloud services with large-scale data centers like Amazon AWS [21] and Microsoft Azure [8].

Although theoretically promising, achieving highly efficient Multi-tenant DL computing on GPU faces many challenges. As shown in Figure 1(b), the increased number of models brings more operator parallelism, larger scheduling complexity, and also higher chances of computing interference and mem contention, etc., which requires a thorough full-stack software/hardware (SW/HW) co-optimization.

MT-DL Computing Stack: Although traditional DL compilers (Figure 1(c)) already include multi-level optimization like compression [33, 48, 49], graph rewriting [17, 19, 46, 52], runtime scheduling [11, 14, 37] and kernel optimization [8, 12, 14, 18, 34, 43], they mainly target at single-tenant settings, which are ill-fitted for multi-tenant. One concrete example is that, TVM [8], as one of the most competitive kernel auto-tuning framework, comes with a built-in assumption of single-tenant execution setting, the tuning configuration of which aims to saturate all SMs and memory bandwidth of the GPU. Such assumption and the single-tenant targeted configuration however becomes unsuitable for concurrent multi-kernel execution since each kernel only has partial resource available. According to [9], the maximum throughput gap comparing single-tenant vs. multi-tenant tuned configurations for the same computing kernel could reach 5× difference.
dedicated GPU hardware primitives for resource partitioning, isolation and allocation, etc. To meet the increasing market needs of such multi-tenant applications, GPU vendors like NVIDIA have recently released many new features including multi-stream [24], multi-process service (MPS) [28], multi-instance GPU (MIG) [27] and virtual GPUs (vCS) [29] to support both runtime scheduling and resource management.

In this work, we aim to provide a comprehensive survey on emerging challenges, opportunities, and research works on multi-tenant DL computing on the GPU. We also hope this survey could motivates 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 1 service-level orchestration [39], 2 graph-level optimization [17, 45], 3 runtime-level scheduling [11], 4 kernel-level tuning [3, 42] and 5 resource-level management [10, 22]. 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 the the same optimization stack, we summarize the major differences in Table I.

1 Service-level: AI-centric cloud services handle millions of service queries simultaneously [44]. With the massive computing capacity of GPUs, multiple DL queries 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 [27].

However, the challenges remains for strategic co-location like that the inter-tenant interference [22] 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 and degrade the overall serving throughput. Therefore, there are many recent service-level orchestration works [5, 22, 45] that design different heuristic-based, modeling-based or prediction-based mechanisms to conduct strategic co-location for efficient multi-tenant computing on GPUs.

2 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 [17]. Single-model DAGs are usually sequential with limited parallelism like VGG, ResNets, MobileNets and EfficientNets, which have only one or two branches and thus exposes small scheduling space [23].

By contrast, multi-tenant DL computing with multiple parallel DAGs usually have extensive inter-operator parallelism for a flexible global-view inter-operator scheduling among tenants [47]. However, certain challenges also emerge such as the increased scheduling complexity due to the larger number of operators and scheduling space.

3 Runtime-level: Previously due to the limited parallelism

| TABLE I | CHALLENGES FOR MULTI-TENANT OPTIMIZATION. |
|--------------------------|----------------------------------|
| Full Optimization Stack | Single-Tenant | Multi-Tenant |
| 1 Service-level          | Co-location | No | Yes |
|                          | Interference | No | High Interference |
| 2 Graph-level            | DAG(s) | Mostly Seq. | Seq. + Parallel |
| 3 Runtime-level          | Parallelism | Limited | Extensive |
|                          | Complexity | Low | High |
| 4 Kernel-level           | Resource Usage | Exclusive | Shared |
|                          | Tuning Objective | 100% util. | ± % partial util. |
| 5 Resource-level         | Management | No | Resource Partition |
and scheduling space, only a few works [2, 11] touch upon runtime-level scheduling for single-tenant scenarios and for certain special models like Inception 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 [30] to address such scheduling overheads.

Kernel-level: Kernel configurations such as loop tiling, thread blocking, memory coalescing, etc. could significantly influence each operator’s computing efficiency. Previous single-tenant kernel-level works like TVM [3] and TF-XLA [42] 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 [9], which shows a 5× throughput difference.

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 [47]. Multiple DL models with different deep structures have highly 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 the flexible GPU reconfiguration capability. Although there are certain adaptive resource provisioning features (e.g., Nvidia multi-process service, multi-instance GPU [27, 28]) that supports resource partitioning, the reconfiguration process requires non-negligible time (e.g., several ms), which is a major limitation of the recent resource scheduling works [10, 24].

B. Vendor GPU Support for Multi-Tenant Scheduling

One important reason that hinders the previous development of multi-tenant DL computing on GPU is the insufficient hardware mechanism support. Unlike CPU multi-tasking, GPU’s SIMD nature and vendor-based blackbox scheduler makes it hard to schedule the workload already, not to mention the DL workloads with even more complex structures. Until recently, with the increasing attention in this topic, GPU vendors like NVIDIA gradually release certain new GPU multi-tasking features to support multi-tenant scheduling. The multi-tenant GPU scheduling features could be categorized into two major types: software-level and hardware-level support.

The very first GPU multi-tasking feature is the Multi-Stream mechanism [24] supported in the Fermi GPU architecture (Figure 2(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 [25] 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 scheduling capability [26]. 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 [11, 47].

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) [28] is a logical resource partitioning mechanism (Figure 2(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) [27] on Ampere architecture enables physical partitioning of both SMs and memory bandwidths through dedicated GPU architecture design (Figure 2(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 [27]. More detailed comparison of Stream, MPS, MIG could be found in Table [I].

### III. Multi-Tenant Computing Optimization: Design and Innovations

Built upon the new features, there are a emerging trend of 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-tuning 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 [39]. 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 [5] 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. [22] 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, [27] propose an ML-based scheduling strategy for multi-tenant DNN execution 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

| Ref.       | Hardware | Perspective                     | Algorithm/Strategy                              | Improvement/Achievement                          |
|------------|----------|---------------------------------|-------------------------------------------------|-------------------------------------------------|
| Inter-Aware [22] | GPU      | DL Service-level Orchestration | • ML-based Interference Predictor               | • Reducing Job Interference                      |
|            |          |                                 | • Proactive Query Scheduler                      | • Enhancing Serving Throughput                   |
| Irina [25] | GPU      | DL Service-level Orchestration | • Online Query Scheduler                         | • Reducing Client-Side ICT                       |
|            |          |                                 | • Heuristic-based Preemption                     |                                                 |
|            |          |                                 | • Concurrent Execution & Batching                |                                                 |
| PREMA [5]  | NPU      | DL Service-level Orchestration | • Online Query Scheduler                         | • Reduced High-Priority Job ICT                  |
|            |          |                                 | • Heuristic-based Preemption                     | • Maintaining Low-Priority SLA                   |
| Runtime-Aware [27] | GPU    | Graph & runtime-level Scheduling | • Multi-Model DAG Rewriting                     | • Reduced Inference Latency                     |
| Spatial-Tune [9] | GPU      | Kernel-level Auto-Tuning      | • MPS-based Resource Allocation                  | • Enhanced Kernel Performance                   |
|            |          |                                 | • Partial-Resource Kernel Tuning                 | • Reduced Inter-kernel Interference             |
| GSlice [14] | GPU      | Resource-level Management      | • MPS-based Resource Partitioning                | • Enhanced Serving Throughput                   |
| Spatial-Partition [4] | GPU  | Resource-level Management      | • MPS-based Resource Partitioning                | • Enhanced Serving Throughput                   |
| Spatial-Partition [3] | GPU  | Resource-level Management      | • Interference-aware Scheduling                  | • Maintaining SLA                               |
| MIG-Serving [41] | GPU    | Resource-level Management      | • MIG-based Resource Reconfiguration             | • Enhanced Serving Throughput                   |
| Planaria [15] | Systolic Arrays | Resource-level Management | • Architecture Reconfiguration                   | • Reduced Energy Consumption                     |

### TABLE III

**Recent Works on Multi-Tenant Computing Optimization (JCT: job completion time, SLA: service-level agreement).**
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: 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. 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) and multi-instance GPU (MIG). Most recent resource-level management works are built upon these technologies.

For example, GSlice 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. utilizes similar spatial partitioning mechanism by MPS while additionally combines temporal scheduling strategies. MIG-Serving 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 a intrinsic limitation that is the inflexible 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. 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

1) 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 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 potential latency degradation performance under different multi-model and hardware combinations so that the scheduler can make better decision regarding the latency SLA constraints. For example, the work 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.

2) 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. is an example that uses MPS to conduct resource partitioning and then implements a heuristic-based task scheduler to find the appropriate one-to-one mapping between the DNN queries and gpu partitions.

In addition to that, software-hardware scheduling could also be leveraged to alleviate certain re-configuration 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 as we introduced before. Besides for the GPU architecture 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 vertical boundaries in the modern DL system stack and conduct full-stack integration to strive for both optimal performance and flexibility. One example of compiler-oriented efforts is the tvm unity as shown in Figure (a). As current system stack conducts separate layer-wise optimization (graph-runtime-kernel-resource) and single-directional deployment, it usually prohibits necessary cross-layer interactions and feedback between different levels. In such cases, unifying the abstraction between layers and
The future trends of technology development towards a larger scale DL system: (a) the full stack in the loop design and (b) the much wider landscape of future DL & system interaction. Automation would greatly facilitate the new full-stack optimization as a loop, not only for multi-tenant computing, but also for future wider DL application.

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 and other types of accelerators. However, if we take into the recent model scaling trend into consideration, a new DL & system interaction mode could be observed, that is to conduct multi-device co-computing for a single model. For example, the recent SOTA giant AI model Megatron-NLG\textsuperscript{36} has reached 500 billions of parameters and requires tens of GPUs to conduct multi-node distributed inference.

We describe the future large-scale DL system landscape by using a taxonomy shown in Figure 3 (b). Using Instance (I) to denote one DNN model and Device (D) to denote the compute hardware, traditional DL system mostly comes within the Single Instance Single Device (SISD) domain and only constitute the top-left quarter of the full-spectrum. Multi-tenant computing emerges as the Multiple Instances Single Device (MISD) with the scaling computing trend, as we summarized in this survey. Whereas diagonally, with the modal scaling trend, the Single Instance Multiple Devices (SIMD) interaction mode also emerges for DL & System and are attracting more and more attention for large model distributed inference\textsuperscript{20} 51 53 such as language models, recommendation models, etc. Finally, Multiple Instances Multiple Devices (MIMD) would eventually combine all these modes together, which can be a practical case for future efficient DL-centric data centers.

V. CONCLUSION

In this survey, we first summarize and categorize the emerging challenges and optimization opportunities for multi-tenant DL inference on GPU. By overviewing the full optimization stack, summarizing the multi-tenant computing innovations, and elaborating the recent technical advances, we hope that this survey could shed lights on new perspectives and novel works in future large-scale DL system optimization.

REFERENCES

[1] Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chong, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard et al. 2016. “TensorFlow: A System for Large-Scale Machine Learning.” In Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16). 265–283.
[2] Yang Bai, Xiaoyi Yao, Qin Bi, and Yu Bu. 2021. AutoTensor: Graph and Tensor Co-Optimization for Image Recognition with Transformers on GPUs. In 2021 IEEE/ACM International Conference on Computer-Aided Design (ICCAD).
[3] Tianyi Chen, Thoery Monara, Zhong Jiang, Liannuo Zhang, Eidyn Tan, Hashem Shen, Mathan Conman, Luyuan Wang, Youwei Liu, Luns Chen, et al. 2014. “TVM: An automatic end-to-end compiler for deep learning.” In In 11th (CIOE) Symposium on Operating Systems Design and Implementation (OSDI’14). 576–594.