POS: An Operator Scheduling Framework for Multi-model Inference on Edge Intelligent Computing

Ziyang Zhang
Harbin Institute of Technology, Harbin
Harbin, Heilongjiang, China
zhangzy@stu.hit.edu.cn

Huan Li
Harbin Institute of Technology, Shenzhen
Shenzhen, Guangdong, China
huanli@hit.edu.cn

Yang Zhao
Harbin Institute of Technology, Shenzhen
Shenzhen, Guangdong, China
yang.zhao@hit.edu.cn

Changyao Lin
Harbin Institute of Technology, Harbin
Harbin, Heilongjiang, China
lincy@stu.hit.edu.cn

Jie Liu
Harbin Institute of Technology, Shenzhen
Shenzhen, Guangdong, China
jieliu@hit.edu.cn

ABSTRACT

Edge intelligent applications, such as autonomous driving usually deploy multiple inference models on resource-constrained edge devices to execute a diverse range of concurrent tasks, given large amounts of input data. One challenge is that these tasks need to produce reliable inference results simultaneously with millisecond-level latency to achieve real-time performance and high quality of service (QoS). However, most of the existing deep learning frameworks only focus on optimizing a single inference model on an edge device. To accelerate multi-model inference on a resource-constrained edge device, in this paper we propose POS, a novel operator-level scheduling framework that combines four operator scheduling strategies. The key to POS is a maximum entropy reinforcement learning-based operator scheduling algorithm *MEOS*, which generates an optimal schedule automatically. Extensive experiments show that POS outperforms five state-of-the-art inference frameworks: TensorFlow, PyTorch, TensorRT, TVM, and IOS, by up to 1.2×~3.9× inference speedup consistently, with 40% improvement on GPU utilization. Meanwhile, MEOS reduces the scheduling overhead by 37% on average, compared to five baseline methods including sequential execution, dynamic programming, greedy scheduling, actor-critic, and coordinate descent search algorithms.

CCS CONCEPTS

• **Human-centered computing** → *Ubiquitous and mobile computing*; • **Computing methodologies** → *Planning and scheduling*.

KEYWORDS

edge computing, multi-model inference, operator scheduling, deep reinforcement earning

ACM Reference Format:
Ziyang Zhang, Huan Li, Yang Zhao, Changyao Lin, and Jie Liu. 2023. POS: An Operator Scheduling Framework for Multi-model Inference on Edge Intelligent Computing. In *The 22nd International Conference on Information Processing in Sensor Networks (IPSN '23), May 09–12, 2023, San Antonio, TX, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3583120.3586953

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

IPSN '23, May 09–12, 2023, San Antonio, TX, USA

© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 979-8-4007-0118-4/23/05…$15.00
https://doi.org/10.1145/3583120.3586953

1 INTRODUCTION

As we all know, AI accelerators and lightweight deep neural network (DNN) have boosted rapid expansion of edge intelligence applications [1, 10], from intelligent inspection in drones [14] to crop growth monitoring in smart agriculture [39], from voice assistants [36] and recommendation systems [4] in smartphones to visual detection in autonomous driving [8]. Recent research on edge intelligent systems focuses on efficient multi-task DNN inference, i.e., how to deploy multiple DNN models on a single edge device with better resource utilization and lower cost [34]. For instance, for a scene perception system in autonomous driving, multiple DNN models are deployed on a single edge device to take full advantage of the compute capability of accelerators to perform real-time inference tasks, such as vehicle detection [24], traffic light recognition [17] and lane tracking [16], *etc.* Such multi-model inference requires not only high accuracy, but also millisecond-level latency. However, current accelerators with GPUs struggle to achieve high throughput and low latency at the same time for multi-model inference [2, 3].

One of the most important factors affecting the performance is operator scheduling, *i.e.*, deciding the order to perform operators in the computation graph abstracted from the DNN model. The operators here represent computation units such as matrix multiplication, e.g., convolution operations [7, 22]. We briefly describe how deep learning models execute on GPUs in a fine-grained manner. Existing deep learning frameworks (*e.g.*, TensorFlow, PyTorch, *etc.*) first abstract DNN models into a computation graph, *i.e.*, a directed acyclic graph (DAG) constructed from operators and their dependencies.

To carry out model inference, the scheduler of deep learning framework must go through the following process: ❶ choose an operator from the ready queue that is waiting to be dispatched, and send it to an appropriate worker thread [18]; ❷ check the type and shape of input tensors, and compute these parameters of output tensors; ❸ allocate free GPU kernels to operators with different tensors types and shapes, and allocate GPU memory for these tensors at the same time; ❹ prepare to submit function parameters related to GPU kernel to accelerator, and dispatch operator one by one according to the topological order.

To achieve high inference performance with low overhead, recent works have leveraged various operator scheduling strategies to accelerate inference. For accelerating single-model inference, Ding et al. [7] developed an operator-level scheduler, namely IOS, that considers two operator scheduling strategies, *i.e.*, operator fusion [23] and inter-operator parallelism [22]. IOS chooses the optimal approach at different stages of inference based on dynamic programming scheduling algorithm. However, IOS is only suitable for accelerating a single inference model.

Compared with single-model inference, multi-model inference involves resource competition between models and complex schedule [37, 38], and thus is more challenging to achieve low latency and high throughput. To illustrate the challenge, we show the multi-model scheduling problem of current deep learning frameworks, e.g., TensorFlow and PyTorch, running on GPUs in Fig. 1. We utilize a series of single models (i.e., ResNet18/34/50 [12], Inception-v1 [31], and MobileNet-v1 [13]) to assemble homogeneous (R18+R34+R50) and heterogeneous (R18+Iv1+Mv1) multi-model. We leverage the concurrent execution strategy(i.e., CUDA streams) from native GPU multi-stream support to evaluate the inference performance of different multi-model combinations on a single edge device. It is clearly seen from Fig. 1 that the current deep learning frameworks provided by vendors have not enough support for multi-model inference yet. To be more specific, the average GPU utilization of Tensorflow for homogeneous and heterogeneous multi-model inference is only 34% and 43%, respectively. The result under PyTorch is even lower. Moreover, the overall inference latency of multi-model under TensorFlow/PyTorch unable to satisfy the real-time demands of edge intelligence applications. The reason for the low utilization and high latency is that the schedule in deep learning framework are either sequential or limited execution in parallel for computation graphs, which leads to the interference mainly caused by the potential GPU kernel queuing delay and resource contention when inferring multi-model on a single GPU. Therefore, such inefficient schedule would cause huge GPU idle in multi-model inference.

In order to utilize the resource of accelerator efficiently to speed up multi-model inference concurrently, Yu et al. [37] first abstracted multi-tenant inference scheduling into fine-grained concurrency control, and then proposed the scheduling algorithm based on machine learning to automatically find the optimal amount of parallelism for each stage in computation graph composed of multi-model. However, it only considers inter-operator parallelism and has not considered other scheduling strategies yet, *e.g.*, intra-operator parallelism, operator fusion, subgraph reuse, *etc.* Accordingly, a framework based on that still would not utilize the full compute capability and parallel advantage of edge accelerators.

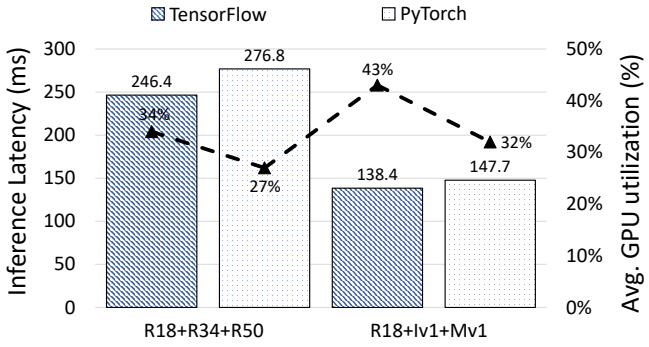


Figure 1: The overall inference latency and average GPU utilization from different multi-model combinations: ResNet18+ResNet34+ResNet50 (homogeneous [12]), ResNet18+Inception-v1+MobileNet-v1 (heterogeneous [13, 31]), running on NVIDIA Xavier NX GPU with 8 GB memory.

In this paper, we propose a novel operator-level scheduling framework, namely POS, to provide efficient multi-model inference service. The key to POS is a novel operator-level scheduling algorithm, maximum entropy reinforcement learning-based operator scheduling, namely MEOS. MEOS first searches for optimal schedule in the training stage offline, and then deploys the generated optimal schedule to edge devices for online inference in real-time. POS targets to efficiently utilize the resources of edge accelerators through fine-grained scheduling at the operator level, accordingly reducing the overall inference latency of multi-model. The proposed MEOS algorithm can schedule multi-model intelligently with four operator scheduling strategies: operator fusion [23], subgraph reuse [7], inter-operator and intra-operator parallelism [22]. Specifically, operator fusion and subgraph reuse can significantly reduce memory access and kernel scheduling overhead. Inter-operator and intra-operator parallelism can improve the parallelism of model inference in coarse-grained and fine-grained modes, respectively. The efficacy of four operator scheduling strategies mentioned above has been demonstrated for a single model [7, 22], in which one or two operator scheduling strategies are used in inference acceleration.

Different from the previous single-model acceleration work based on operator scheduling, we utilize multiple operator scheduling strategies to accelerate multi-model inference concurrently. To the best of our knowledge, we are the first to combines four operator scheduling strategies to accelerate multi-model inference. To summarize, we design a novel operator scheduling framework, namely POS, to support efficient multi-model inference and have the following three major contributions:

- We abstract the multi-model inference into a computation graph-based unified intermediate representation (IR), thus transforming the overall latency problem of minimizing multi-model inference into a fine-grained operator scheduling problem.
- Based on four operator scheduling strategies, we propose a novel learning-based operator scheduling algorithm, namely MEOS, to find optimal scheduling strategies for operators in computation graph automatically.

- We conduct extensive experiments in diverse multi-models. The results show that POS can consistently achieve 1.2×~3.9× speedup compared to state-of-the-art deep learning inference frameworks, with 40% improvement on GPU utilization. Meanwhile, MEOS reduces the average scheduling overhead by 37%.

2 MOTIVATION

In this section, we explain the motivation of this work, and the reason why this work outperforms current sequential and parallel scheduling algorithms in multi-model inference. As shown in Fig. 2, for sequential scheduling, since a single operator takes over the entire computing resources, the sequential execution for a single operator significantly increases kernel dispatch overhead and memory access, resulting in low resource utilization and high inference latency. For scheduling in parallel, the scheduler abstracts each model into computation graph-based unified intermediate representation (IR). In particular, the nodes and edges in the computation graph represent operators and the dependencies between operators, respectively. The computation graph is divided into multiple stages, and the operators in the same stage can be executed in parallel, while the operators in different stages are executed sequentially. Although this approach improves resource utilization, the performance improvement is limited, because there is no more effective parallel schedule to reduce kernel dispatch overhead and memory access, which inevitably increases inference latency.

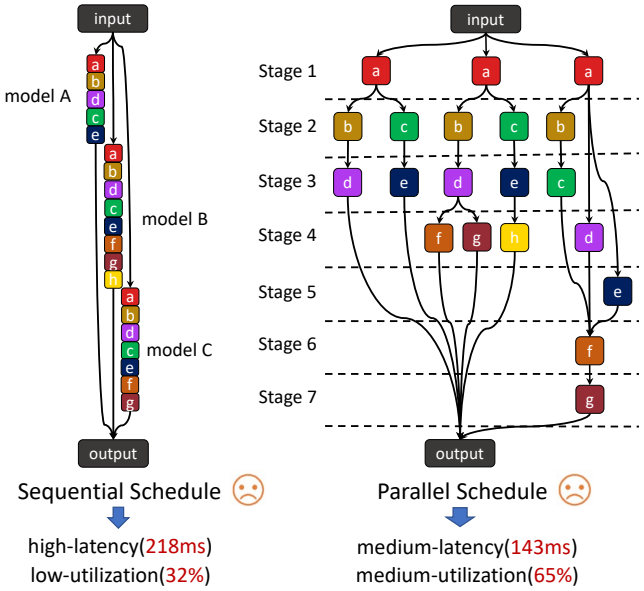


Figure 2: The performance of multi-model inference with different schedules.

The existing scheduling algorithms essentially optimizes the inference of a single-model. For more challenging multi-model inference, we need more advanced schedule. To this end, we combine intra-operator parallelism, operator fusion, and subgraph reuse, in addition to inter-operator parallelism, in order to improve parallelism by significantly reducing kernel dispatch overhead and memory access. Fig. 2 illustrates the inference latency and resource utilization of various scheduling algorithms in multi-model inference. We can clearly see our method outperforms sequential and scheduling in parallel. We explain the four operator scheduling strategies mentioned above in more details in Section 4.

Once we define the computation graph-based search space, we need to answer another question: how to find an optimal schedule with minimal overhead in large search space composed of multi-model. Existing work has demonstrated that the number of schedules is exponential in the number of operators [7]. Our work is inspired by recent advantages of deep reinforcement learning in dealing with complex policy decision, so that we investigate how to apply it to our computation graphs-based operator scheduling problem. As described in Section 4.2, we reformulate our scheduling problem, and propose an deep reinforcement learning-based operator scheduling framework POS. POS targets to find optimal schedule automatically for operators in different stages in order to significantly reduce inference latency.

3 PROBLEM FORMULATION

We first abstract multi-model into a directed acyclic graph (DAG)-based unified intermediate representation (IR) [37]. The computation graph can be expressed as $G = (V, E)$. Where V is the set of operators, a vertex represents an operator. E is the set of edges with dependencies between operators. Afterward, we split the computation graph into multiple stages, denoted as $G = \{stage_1, stage_2, ..., stage_n\}$. The stages are executed sequentially according to First-In-First-Out (FIFO), and operators in the same stage can be executed in parallel with different scheduling strategies. In addition, we define different pieces in the same stage as a group, denoted as $stage = \{g_1, g_2, ..., g_m\}$. The operators in the same group are executed sequentially, while operators in different groups in the same stage can be executed in parallel.

According to the computation graph obtained by the abstraction, we can transform a minimization problem of the overall latency of multi-model inference into a fine-grained operator scheduling problem.

The schedule of a computation graph G, i.e., a policy Φ, which can be formulated as:

$$\Phi = \{s_i(P_i, L_i)\} \tag{1}$$

where s_i is the ith stage, L_i is the running latency for the ith stage, and P_i is the corresponding operator scheduling strategy for the ith stage.

We utilize the cost model in [2] to measure the inference latency of the overall computation graph in order to find the optimal schedule. Our objective function is:

$$\Phi^* = argmin_\Phi f(G, \Phi), for\ \Phi \in R_\Phi \tag{2}$$

where f is the cost model that is used to directly measure the running latency of the computation graph G generated by using the schedule Φ on device. R_Φ is all potential search spaces. We use deep reinforcement learning to generate the optimal schedule (Section 4).

The definitions and descriptions of the major symbols used in the rest of this paper are listed in Table 1.

Table 1: Symbol Table and Description.

Notation	Description
G	A computation graph
V	The set of operators
E	The edge set of dependencies between operators
Φ	Operator scheduling policy
s_i	The ith stage
g_i	The ith group
L_i	The running latency of the ith stage
P_i	Operator scheduling strategy for the ith stage
f	Cost model
R_Φ	Search space
s_t	The state of agent at timestep t
a_t	The action of agent at timestep t
r_t	The reward of agent at timestep t
c_u	CPU utilization
g_u	GPU utilization
m_u	Memory utilization
e_u	Power consumption
α	Temperature parameter
$\mathcal{H}$	Entropy

4 SYSTEM DESIGN

Based on the problem formulated above, we design a novel operator scheduling framework, namely POS, to choose the optimal scheduling strategies for operators so that it can provide efficient multi-model inference services. We further propose a novel entropy-based deep reinforcement learning algorithm namely MEOS to efficiently dispatch operators, so that the scheduling overhead is dramatically reduced in huge spaces that is composed of computation graphs. The overview framework of POS is shown in Fig. 3, and we will describe four scheduling strategies in more details next.

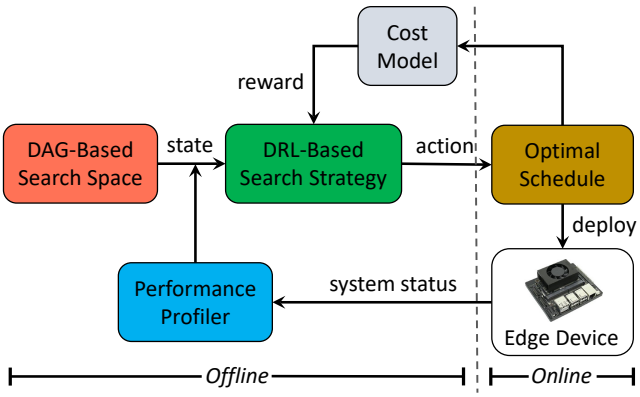


Figure 3: Overview of our proposed operator-level scheduling framework POS.

4.1 Operator scheduling strategy

The scheduler is the core of POS. We take use of the following four different operator scheduling strategies to take full advantage of the parallel advantages of GPU: operator fusion, subgraph reuse, inter-operator parallelism, and intra-operator parallelism. To be more specific, operator fusion refers to merge multiple operators of the same or different types into a large operator [23]. Subgraph reuse multiplexes subgraphs with the same input and output shape in computation graph [7]. Instead of using sequential orders, inter-operator parallelism changes the topology of operators in computation graph from sequential to parallel [22]. Finally, intra-operator parallelism means performing arithmetic operations in parallel within a single operator [22]. We next describe these scheduling strategies in detail.

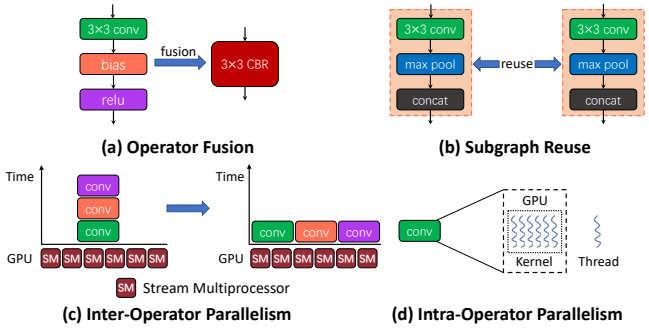


Figure 4: Illustration of four operator scheduling strategies.

4.1.1 Operator Fusion. Operator fusion mainly merges producer-consumer operators with the data-dependent on the computation graph. This approach reduces additional memory accesses by executing multiple computations at once, rather than writing data to global memory due to repeated reads. For instance, in Fig. 4(a), after we merge convolution (conv), bias and rectified linear unit (relu) operators, we only need to leverage one CBR (conv+bias+relu) block to replace the original three operator calculations, accordingly reducing two memory access operations. Note that only child nodes with the same parent node can be merged.

4.1.2 Subgraph Reuse. Similar to the purpose of operator fusion, subgraph reuse also accelerates inference by reducing memory access and kernel dispatch overhead. As shown in Fig. 4(b), since there are many redundant subgraphs in the computation graph composed of multi-model, we can reuse these subgraphs to reduce the dispatch frequency of GPU kernel. Due to the limitation of tensor types and shapes for operators in computation graph, we only perform subgraph reuse at the beginning of the computation graph. This is motivated by the fact that the first few layers of the network structure for multi-model are highly redundant. Another benefit of using subgraph reuse is that it can also shrink the search space of the schedule.

4.1.3 Inter-Operator parallelism. In spite of current deep learning frameworks utilize intra-operator parallelism, the benefit of inter-operator parallelism has not been explored to make GPU resource utilization more efficient. Considering that inter-operator and intra-operator parallelism are orthogonal, we introduce the inter-operator parallelism into the scheduling of the operator in computation graph. As shown in Fig. 4(c), on the left in Fig. 4(c) illustrates the operator scheduling mechanism of the popular deep

learning framework. Model inference is accomplished by sequentially scheduling an operator one by one on multiple stream multiprocessor. On the right in Fig. 4(c) explains the inter-operator parallelism. We divide the computation graph into multiple stages, and operators in the same stage can be executed in parallel to improve resource utilization and reducing latency.

4.1.4 Intra-Operator parallelism. As shown in Fig. 4(d), after the scheduler in the deep learning framework dispatches operators to accelerator one by one, the scheduler in accelerator will enable multi-threading to divide a single operator into finer-grained scheduling units, and map them to multi-threading to take full advantage of the parallelism of accelerator. Here we apply TVM-cuDNN [2] to achieve intra-operator parallelism. TVM-cuDNN compiles convolutional neural networks in TVM using the cuDNN library, which would use the convolution kernel provided by cuDNN to efficiently execute convolutions.

4.2 Operator Scheduling Problem

For single-model operator scheduling, IOS [7] utilizes dynamic programming to generate optimal schedule for a single model. Although this optimization method can be applied to schedule, the complexity suffers from the curse of dimensionality as the size of the problem increases, especially for multi-model operator scheduling. In comparison, deep reinforcement learning (DRL) combines the advantages of efficient decision of reinforcement learning and the powerful representation of deep learning, and has proved to Go [25] and protein prediction [15], et al. Since the computation graph we abstract from multi-model is extremely complex and has high-dimensional information. We can naturally treat the operator scheduling problem as a sequence decision problem, and utilize DRL to search for the optimal schedule automatically. Therefore, we convert the function in Eq.(2) into a reward in DRL and model it as a markov decision process (MDP) that can be described by a four-tuple: ($\mathcal{S}$, $\mathcal{A}$, π, r). Each parameter is specifically defined as follows:

4.2.1 State. $\mathcal{S}$ is the state space. At each scheduling timeslot t, the agent in DRL constructs a state $s_t = \{G, U\}$, ($s_t \in S$) that consists of two parts: (I) the computation graph-based search space $G = (V, E)$. (II) the system information periodically collected on edge devices, which include currently available CPU/GPU/memory utilization and power consumption, denoted as c_u, g_u, m_u, and e_u, respectively.

4.2.2 Action. $\mathcal{A}$ is the action space that is used to find the optimal schedule for the operators in each stage in the computation graph. Therefore, the schedule at timeslot t can be expressed as $a_t = s_t^i(P_t^i, L_t^i)$. Where s_t^i is the ith stage in timeslot t for the computation graph, and P_t^i and L_t^i are the operator scheduling strategy selected by the agent and the corresponding execution latency, respectively.

4.2.3 Reward. r_t is the immediate reward obtained by the agent when the agent executes the action at timeslot t. To be more specific, in this paper r_t refers to the execution latency obtained by the agent after choosing an appropriate operator scheduling strategy for each stage at each scheduling timeslot t. The agent targets to maximize the accumulated expected reward $\mathbb{E}\left[\sum_{t=0}^{T} \gamma^t r_t\right]$, while our objective is to minimize the overall inference latency of multi-model. Therefore, the reward in DRL is:

$$r_t = -f(G, \Phi). \tag{3}$$

4.2.4 Policy. The policy $\pi(a_t|s_t)$ refers to the function related to agent, which is used to decide the next action $a_t (a_t \in \mathcal{A})$ according to the state of the environment s_t at timeslot t. We treat the scheduling policy Φ of operator as the policy function of agent in DRL. The optimal policy π^* can be defined as follows:

$$\Phi^* = \pi^* = \text{argmax}_\pi \sum_{t=0}^{T} E_{(s_t, a_t)\sim\rho_\pi}\left[\gamma^t r\left(s_t, a_t\right)\right] \tag{4}$$

where $\gamma \in [0, 1]$ is a discount factor and ρ_π is the trajectory distribution produced by policy π.

4.3 Maximum Entropy DRL-based Operator Scheduling Algorithm

4.3.1 Methodology Overview. In general, the conventional DRL-based approaches have the following weaknesses when applied to complex real-world tasks: ***(1) Inefficient sampling***: with regard to on-policy approaches such as PPO [27], TRPO [26], *etc*, each round of updates needs to resample sufficient samples and discard the previously samples completely. Therefore, diversity of numerous samples are required to ensure the algorithm converges. ***(2) Extremely sensitive to hyperparameters***: with regard to the off-policy approaches such as DDPG [21], although replay buffer is used to relieve sampling inefficiency, these approaches are extremely coupled with Q-value (used for policy evaluation) that may lead to system instability and affected by hyperparameters readily.

In this work, we introduce entropy into DRL [9] to maximize the reward while maximizing the entropy of the visited states. As we all know, the entropy is a measure that describes the indeterminacy for random variables. Apparently, the higher the uncertainty of random events, the larger the entropy. The entropy can be formulated as:

$$\mathcal{H}(\mathrm{X}) = -\sum_{x_i \in X} P\left(x_i\right) \log P\left(x_i\right). \tag{5}$$

Introducing entropy in DRL has the following benefits:

- ***Accelerate convergence***. The entropy enables the policy learned by the agent to be used as initialization for more complex tasks. The reason is the entropy allows the policy to learn not only one approach to solve the task, but all of them, so that the agent is able to learn more near-optimal actions to accelerate the learning process.
- ***Encourage exploration***. Apparently, the entropy will make the distribution of actions more uniform, and find a better schedule faster to avoid falling into a local optimum.
- ***Robustness improvement***. Since the agent can potentially explore the possibility of diverse optimal solutions with different schedules, therefore the agent is better to make adjustments when encountering disturbances.

Because the action in DRL is discrete, we propose a maximum entropy reinforcement learning-based operator scheduling algorithm namely MEOS. We introduce entropy into reward [9] in MEOS, and

Eq.(4) is converted into the following form:

$$\Phi^* = \pi^* = \text{argmax}_\pi \sum_{t=0}^{T} E_{(s_t,a_t)\sim\rho_\pi}[\gamma^t r(s_t,a_t) + \alpha\mathcal{H}(\pi(\cdot \mid s_t))] \tag{6}$$

where α is the temperature parameter used to balance reward and entropy. $\mathcal{H}(\pi(\cdot \mid s_t)) = -\log\pi(\cdot \mid s_t)$ is the entropy of policy π in state s_t. We next describe the proposed MEOS algorithm in detail.

4.3.2 Details of Algorithm. Our proposed MEOS algorithm is summarized in Algorithm 1. MEOS is based on the discrete Soft Actor-Critic (SAC) [9][5] framework. A dictionary is constructed to record the execution latency for each stage according to operator scheduling strategy selected by the agent. The dictionary is initialize as ∅. Afterward, we initialize all networks and corresponding parameters in MEOS. We use two Q networks and take the one with the smallest Q-value as the target Q-value to avoid the overestimation of the Q-value and achieve stable training. In particular, there are five networks in MEOS, including one policy network as actor and four Q networks (two Q networks and two target Q networks) as critics. We finally initialize an empty replay buffer to store historical experience.

For each episode with the state s_t of current timestep t, the agent chooses an optimal scheduling strategy for each stage according to the policy in MEOS and evaluate the corresponding latency simultaneously. We put the result of each scheduling strategy into the dictionary τ and returns the operator scheduling strategy with the lowest latency for each stage. Simultaneously, the agent receives immediate reward and update the state according to Eq.(3), and put the current information into the replay buffer as historical data for next learning step. We utilize soft policy iteration [9] with policy evaluation and policy improvement to maximize the accumulated expected reward. The following are the steps in detail:

We first compute the soft state value of policy π in policy evaluation at each gradient step:

$$V(s_t) := \pi(s_t)^T [Q(s_t) - \alpha\log(\pi(s_t))]. \tag{7}$$

We then utilize the modified bellman backup operator to calculate the soft Q-function y_t:

$$y_t = r(s_t,a_t) + \gamma E_{s_{t+1}\sim p(s_t,a_t)}[V(s_{t+1})]. \tag{8}$$

We next update the parameters for all networks.

- ***Update soft Q-function (Critic network)***. We train soft Q-function by minimizing the soft bellman residual, the loss function of the critic network is:

$$J_Q(\theta) = E_{(s_t,a_t)\sim D}[\frac{1}{2}(Q_\theta(s_t,a_t) - (r(s_t,a_t) + \gamma E_{s_{t+1}\sim p(s_t,a_t)}[V_{\bar{\theta}}(s_{t+1})]))^2] \tag{9}$$

  where $V_{\bar{\theta}}(s_{t+1})$ is obtained by sampling from the replay buffer using monte carlo estimation in target network.
- ***Update policy (Actor network)***. We update the policy network to maximize reward in policy improvement, the loss function of the actor network is:

$$J_\pi(\phi) = E_{s_t\sim D}\left[\pi_t(s_t)^T\left[\alpha\log\left(\pi_\phi(s_t)\right) - Q_\theta(s_t)\right]\right] \tag{10}$$

  where D represents Kullback-Leibler (KL) divergence, and α is a temperature parameter [9].
- ***Update temperature parameters***. We utilize the approach proposed by Haarnoja et al. [9] to update the temperature parameter α automatically (see [9] for details), which can be defined as follows:

$$J(\alpha) = E_{a_t\sim\pi_t}\left[-\alpha\left(\log\pi_t(a_t \mid s_t) + \bar{H}\right)\right]. \tag{11}$$

- ***Update target network***. We finally update the target network with the soft update to stabilize the training:

$$\bar{\theta}_i \leftarrow \eta\theta_i + (1-\eta)\bar{\theta}_i, \text{ for } i \in \{1,2\}. \tag{12}$$

Algorithm 1: MEOS operator-level scheduling algorithm

Input : a computation graph $G = (V,E)$ of M models, the number of stages s_i in each model, and current system status $u_t = \{c_t, g_t, m_t, e_t\}$
Output : an optimal schedule Φ^*

Initialization a dictionary $\tau\{s_i(P_i,L_i)\} = \varnothing$ to record the latency of each stage L_i according to the scheduling strategy selected by agent P_i;
Randomly initialize network parameters $\theta_1, \theta_2, \phi, \alpha$;
Initialization the weights of actor network $\pi(s|\phi)$ with ϕ and critic network $Q(s,a \mid \theta_1)$,$Q(s,a \mid \theta_2)$ with θ_1 and θ_2, respectively;
Initialization the weights of target network $Q' : \bar{\theta_1} \leftarrow \theta_1, \bar{\theta_2} \leftarrow \theta_2$;
Initialize an empty replay buffer $\mathcal{D} \leftarrow \varnothing$;
for *episode* $e \leftarrow 1$ **to** E **do**
  for *environment step* $t \leftarrow 1$ **to** T **do**
    for *the ith stage* $i \leftarrow 1$ **to** N **do**
      Choose action a_t based on policy $\phi(a_t \mid s_t)$;
      $\tau \leftarrow s_t^i(P_t^i, L_t^i)$;
    end
    return P_t^i with the lowest latency L_t^i;
    Obtain instant reward $r_t(a_t \mid s_t)$ using Eq.(3);
    $s_{t+1} \sim p(s_{t+1} \mid s_t, a_t)$;
    $\mathcal{D} \leftarrow (s_t, a_t, r(s_t,a_t), s_{t+1})$;
  end
  for *each gradient step* $g \leftarrow 1$ **to** G **do**
    Calculate $V(s_t)$ for policy π using Eq.(6);
    Obtain the soft Q-function y_t using Eq.(7);
    Update critic network θ_i using Eq.(8);
    Update actor network ϕ using Eq.(9);
    Update temperature parameter α using Eq.(10);
    Update target network $\bar{\theta_i}$ using Eq.(11);
  end
end

Case: Fig. 5 illustrates the scheduling process from POS for the multi-model inference case shown in Fig. 2. The parallel scheduling in Fig. 2 divides the computation graph composed of multiple models into seven stages, while POS shortens the number of stages to three. In addition, the four scheduling strategies used by POS can

effectively improve resource utilization to increase the concurrency of multi-model inference. Specifically, the optimal schedule found by POS reduces latency by 31% and 55% compared to sequential and concurrent, respectively.

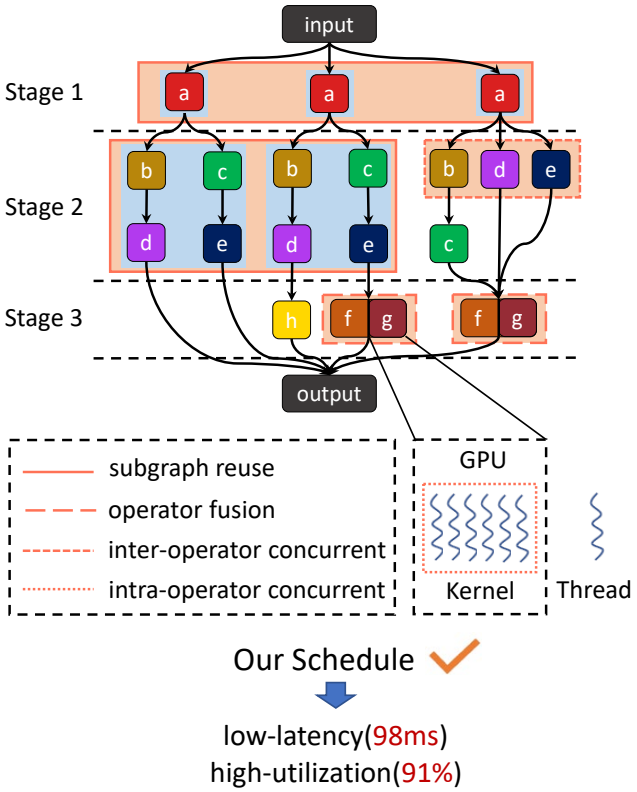


Figure 5: The optimal schedule from POS for the multi-model inference case illustrated in Fig. 2.

5 PERFORMANCE EVALUATION

In this section, we report comprehensive evaluation results to illustrate the efficiency of POS with comparison with other state-of-the-art deep learning frameworks and different scheduling algorithms.

5.1 Experimental Setup

5.1.1 Benchmark Models. We benchmark nine modern CNNs in our experiment: ResNet-18/34/50 (R18/34/50) [12], Inception-v1/v2/v3 (Iv1/v2/v3) [31] and MobileNet-v1/v2/v3 (Mv1/v2/v3) [13]. Table 2 shows different combinations of nine models with a homogeneous or heterogeneous manner. Note that these models have different types and numbers of operators, therefore, these models have different requirements for computing and memory resources.

5.1.2 Baselines. We compare POS with the following five state-of-the-art deep learning inference frameworks:

- ***TensorFlow:*** As a deep learning framework with static graphs, TensorFlow constructs a tensor-based static computation graph before execution, and then dispatches operators one by one sequentially.

Table 2: Homogeneous/Heterogeneous Combinations of Multi-model.

Combination Type	Multi-model
Homogeneous	ResNet-18+ResNet-34+ResNet-50
	Inception-v1+Inception-v2+Inception-v3
	MobileNet-v1+ MobileNet-v2+ MobileNet-v3
Heterogeneous	ResNet-18+ Inception-v1+ MobileNet-v1
	ResNet-34+ Inception-v2+ MobileNet-v2
	ResNet-50+ Inception-v3+ MobileNet-v3

- ***PyTorch:*** Unlike TensorFlow, PyTorch is a deep learning framework with dynamic graph, which means that DNN models unnecessary generate computation graphs before compile phase. Similarly, the scheduler in PyTorch also dispatches operators sequentially.
- ***TensorRT:*** As an popular deep learning inference engine, TensorRT accelerates inference through a series of lightweight technologies (*e.g.*, weight pruning, operator fusion, and precision quantization *etc.*), which schedules the computation graph after the automatic fine-tuning of the kernel to the multi-stream in GPUs to execution in parallel.
- ***TVM:*** An machine learning compiler framework for inference optimization. TVM abstracts the DNN model into a computation graph-based unified intermediate representation, and utilizes AutoTVM [3] and AutoScheduler [41] to generate optimal schedule for scheduling primitives [2] (*e.g.*, loop transformations, inlining, vectorization, *etc.*) on GPUs.
- ***IOS:*** IOS considers operator fusion and inter-operator parallelism, and utilizes dynamic programming-based scheduler to accelerate model inference.

Besides, we also compare the proposed MEOS scheduling algorithm with five different scheduling algorithms, *i.e.*, sequential execution, dynamic programming (DP), greedy scheduling, Actor-Critic (AC) and coordinate descent search (CDS) in [37] with the same inference framework. We discuss in more details in Section 5.3.

5.1.3 Implementation. We implement the prototype of POS with a popular edge device and a camera to evaluate the performance of multi-model inference. Specifically, we take NVIDIA Xavier NX as an edge device and deploy it on autonomous vehicle. NVIDIA Xavier NX is an embedded edge computing platform to provide inference services, which installed Ubuntu 18.04 with cuDNN 7.6.5 and CUDA 10.2. We use the Intel RealSense D435 to capture the video in real-time that can generat video frames with 1920×1080 resolution at 30 frames per second (FPS). Note that POS can be suitable for high-resolution images if the computing power of the edge device is abundant or there is no strict real-time constraint. Due to limited computing power of edge devices, we downsample the resolution of video to 224×224×3, and set the batch size to 1. Furthermore, we report the average inference latency of 5 experiments.

We use PyTorch to implement our proposed MEOS algorithm in Algorithm 1. All networks are trained with the Adam optimizer and leveraged a three-layer relu activation function with 128, 64, and 32 hidden units in each layer. In addition, we fixed the learning rate to $1e^{-4}$. The buffer size and batch are set to $1e^{6}$ and 512, respectively.

5.2 Comparison of State-of-the-art Deep Learning Frameworks

We evaluate the inference performance of the proposed POS scheduling framework on three homogeneous models. All inference experiments are performed on batch size one unless stated otherwise. We compare it to five state-of-the-art deep learning frameworks. The performance is normalized according to the optimal inference latency to compare the relative speedup. Fig. 6 indicates that POS consistently outperforms all benchmark frameworks for three homogeneous multi-models. In particular, POS has a significant improvement compared with TensorFlow and PyTorch with 2.6×~3.9× speedup. The reasons are as follows:

On one hand, the built-in scheduler of baseline frameworks utilize sequential execution, which require frequent dispatch kernel and access memory. On the other hand, the baseline frameworks does not consider execution in parallel to fully exploit the potential of GPUs, which lead to inefficiency.

Furthermore, compared to state-of-the-art libraries like TVM, TensorRT and IOS, POS can also achieve 1.2×~1.5× speedup. Although these libraries considered operator parallelism to accelerate inference, they either only utilize inter-operator parallelism or operator fusion, ignoring the cooperative scheduling of inter-operator and intra-operator parallelism. Therefore, the degree of operator parallelism is still limited.

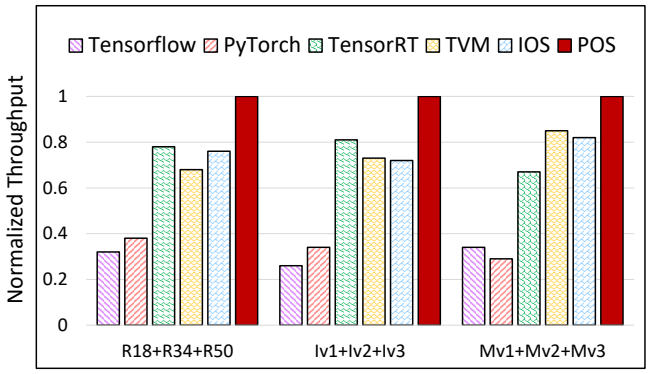


Figure 6: The inference performance of diverse homogeneous models on different deep learning frameworks. The throughput is normalized to the highest one for each model.

In addition to homogeneous multi-model, we also investigate the inference performance of all framework under heterogeneous multi-model. As shown in Fig. 7, POS can achieve 1.2×~2.8× relative speedup. We observe that the outperformance is lower than the homogeneous model. The reason for this phenomenon is that there is less similarity between the structures of heterogeneous models that reduces the redundancy of operators. Since there are fewer opportunities to apply operator fusion and subgraph reuse, the kernel dispatch overhead and frequency of accessing memory is increased. Moreover, the computation graph abstracted from heterogeneous multi-model is more complex, this also affects the operator-inter parallelism to a certain degree.

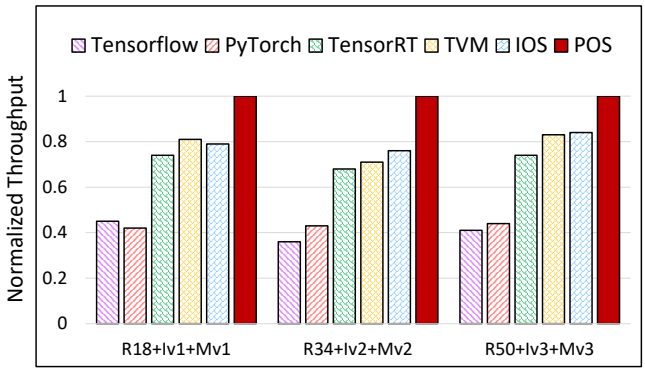


Figure 7: The inference performance of diverse heterogeneous models on different deep learning frameworks. The throughput is normalized to the highest one for each model.

5.3 Comparison of Different Scheduling Algorithms

In this subsection, we report the results of the comparison of various schedules. The baselines include five scheduling algorithms: sequential execution, dynamic programming (DP), greedy scheduling, Actor-Critic (AC), and coordinate descent search (CDS). We execute all operator scheduling strategies on IOS's inference engine [7] for fair comparison. Fig. 8 shows the inference latency of all homogeneous multi-models, it demonstrates that POS consistently outperforms all baselines with 1.2×~2.8× relative speedup.

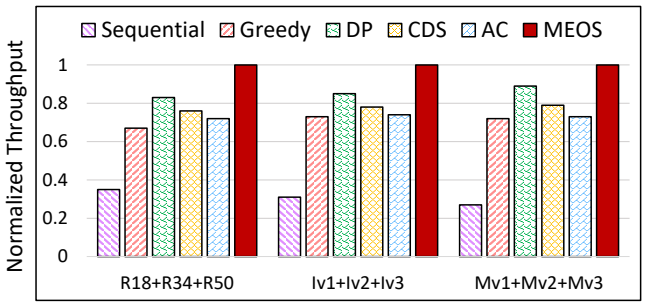


Figure 8: The inference performance of diverse homogeneous models on different scheduling algorithms. The throughput is normalized to the highest one for each model.

We explain the results from the following aspects: (I) The sequential execution only individually dispatches operators according to the topology in the computation graph. Because it ignores the benefits of parallelism, the inference performance is inefficient. (II) The greedy scheduling puts all operators in the same stage, and repeats this process until all operators have been dispatched. Although the performance is improved to a certain degree, it will cause excessive operators in the same stage, which results in resource saturation or even out of memory that extremely affects the inference performance. (III) The dynamic programming in IOS [7] selecting the best one between operator fusion and operator parallelism in the last stage for a computation graph. It utilizes two operator scheduling

strategies and that is why the performance is better than the previous two approaches. (IV) Coordinate descent search performs a one-dimensional search along a coordinate direction at the current point to find a local minimum of a function, which is easy to fall into a local optimum. (V) Compared with Actor-Critic, MEOS with entropy can find a better schedule to avoid falling into local optimum. (VI) MEOS considers more operator scheduling strategies: operator fusion, subgraph reuse, inter-operator and intra-operator parallelism, which can fully exploit the parallelism of operators to significantly reduce kernel dispatch overhead and memory access. Therefore, the acceleration performance of MEOS is more significant than all baselines.

Fig. 9 shows the inference performance of various scheduling algorithms on heterogeneous multi-model. We also have the following observations: (I) Similar to the performance of deep learning frameworks in heterogeneous models, the schedule are also affected by model heterogeneity, which means fewer opportunities for operator fusion and subgraph reuse. Nonetheless, POS still has 1.2×~2.9× speedup even compared with the baselines. The reason is that POS can consider more operator scheduling strategies and maximize operator parallelism to match the computing advantages for GPU. (II) Since MEOS is more suitable for efficiently dealing with complex computation graphs, the advantages of POS are gradually increasing when heterogeneous multi-model changes from simple to complex, such as from R18+Iv1+Mv1 to R50+Iv3+Mv3. In addition, MEOS can enable the agent to obtain stronger exploration to find a better schedule by introducing entropy.

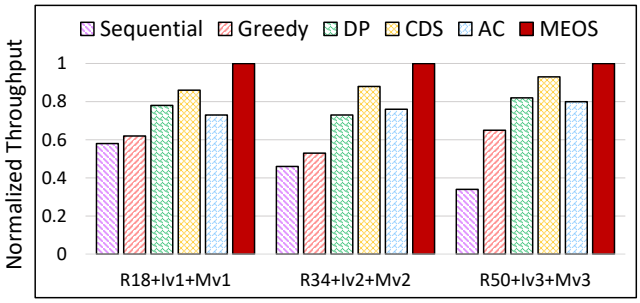


Figure 9: The inference performance of diverse heterogeneous models on different scheduling algorithms. The throughput is normalized to the highest one for each model.

5.4 Real-World Performance Evaluation

In Fig. 10, we use YOLOP [35] to evaluate the performance of in real-world low-speed automated driving (LSAD). LSAD is an autonomous driving system with a maximum speed of 8.89 m/s (32 km/h). It is used for last-mile transportation, such as unmanned warehouses, unmanned docks, university campuses, and other applications in low-speed environments. As a composite model of full-scene perception for automatic driving, YOLOP consists of three submodels: vehicle detection, lane line segmentation, and drivable area segmentation. Therefore, YOLOP is a heterogeneous multi-model. Likewise, POS and the baseline deep learning frameworks execute on the inference engine of IOS.

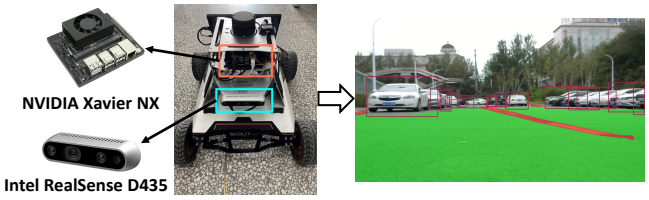


Figure 10: The implementation of POS for multi-model inference in autonomous driving. We use YOLOP [35] as the workload, which consists of three submodels: vehicle detection, lane line segmentation, and drivable area segmentation.

Table 3: The inference latency of different frameworks and scheduling algorithms for YOLOP on NVIDIA Xavier NX.

Framework & Algorithms	FPS (images/s)
TensorFlow	11.2 (**3.16**× ↑)
PyTorch	10.6 (**3.34**× ↑)
TensorRT	28.5 (**1.24**× ↑)
TVM	25.6 (**1.38**× ↑)
IOS	28.1 (**1.26**× ↑)
POS	**35.4**
Sequential	14.9 (**2.13**× ↑)
Greedy	23.7 (**1.34**× ↑)
DP	25.0 (**1.27**× ↑)
CDS	26.9 (**1.18**× ↑)
AC	21.8 (**1.46**× ↑)
MEOS	**31.8**

Table 3 shows the inference latency of different deep learning frameworks and scheduling algorithms for YOLOP. The inference performance of POS is 3.16× and 3.34× compared with TensorFlow and PyTorch, respectively. Obviously, POS considers four operator scheduling strategies that can effectively improve GPU utilization to significantly reduce inference latency. Besides, compared to the state-of-the-art deep learning inference frameworks such as TensorRT, TVM and IOS, POS also has 1.24×~1.38× speedup. Moreover, MEOS has 1.18×~2.13× relative speedup when comparing the baseline scheduling algorithms. All these results demonstrates the efficiency of MEOS.

Resource contention. As shown in Fig. 11, we visualize the memory usage timeline of the scheduling strategy used by POS for the three subtasks in YOLOP. In order to mitigate the impact of contention, POS uses more parallel strategies to increase resource utilization to compensate for its performance loss. The results show that our method can efficiently find better schedules through a reinforcement learning-based automatic search algorithm (i.e., MEOS), thereby achieving higher speedup in multi-model inference.

5.5 Scheduling Algorithm Comparison and Overhead Analysis

5.5.1 Scheduling Algorithm Comparison. Fig. 12 presents the performance of all scheduling algorithms when search for optimal inference latency in the case of R18+R34+R50 and YOLOP. All methods are evaluated on the inference engine of IOS. The red line illustrates the native performance of the scheduling algorithms (sequential) in popular deep learning frameworks. Especially, in

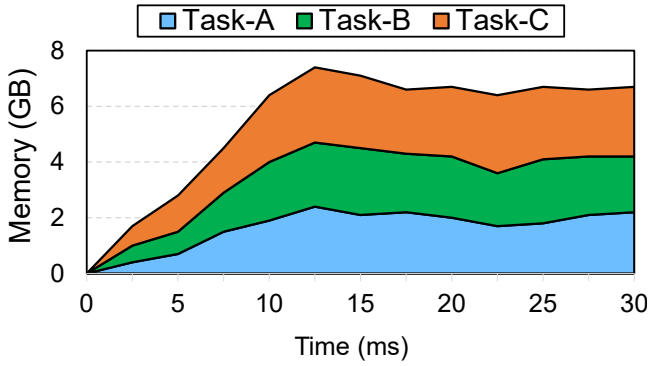


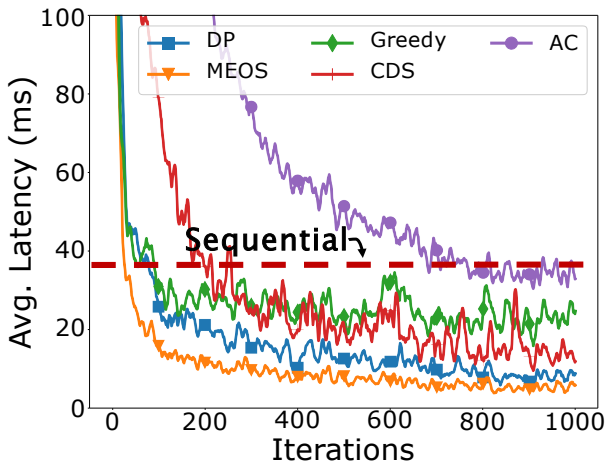
Figure 11: Resource contention for multi-model. POS could find a better schedule to avoid both contention and under-utilization, thus achieving better performance.

complex scenarios like Fig. 12(b), MEOS has faster convergence and lower latency. It is also attributed to the benefits of entropy that can accelerate the learning process and improve stability. In contrast, MEOS has a slight advantage in homogeneous model. Overall speaking, the results indicate that MEOS has both better performance than baseline algorithms in homogeneous (R18+R34+R50) and heterogeneous (YOLOP) multi-model.

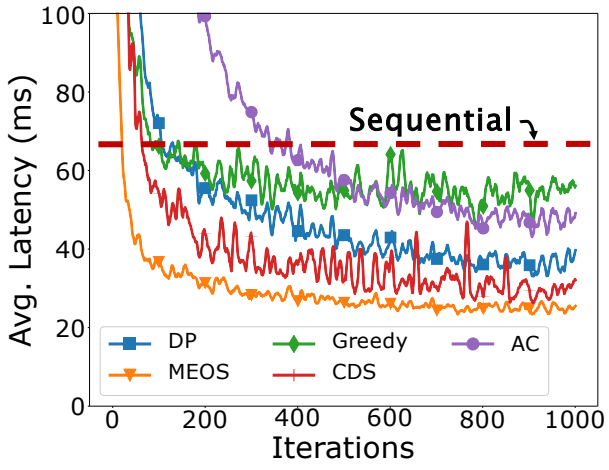
5.5.2 Scheduling Overhead Analysis. We use a homogeneous and two heterogeneous multi-models to evaluate the scheduling overhead of MEOS, greedy scheduling, DP, CDS and AC (sequential execution has no scheduling overhead), respectively. All scheduling algorithms are also executed on the inference engine of IOS. As shown in Fig. 13, POS can generate a near-optimal schedule with less overhead. To be more specific, the average scheduling overhead of POS is reduced by 37% compared to the baselines. It is also reveals that the conventional heuristics-based approaches are inefficient in processing complex computation graph. Because MEOS introduces entropy to enable the agent learn more near-optimal actions in order to accelerates convergence. Note that MEOS is trained offline and deployed online.

5.6 Micro-benchmark Performance

5.6.1 Different GPU platforms. Fig. 14 shows the performance of our scheduling framework POS under four heterogeneous GPU platforms and three model settings. Compared with the baseline inference frameworks, POS has a significant overall performance gain (1.35× to 2.84× average speedup) on different GPU platforms. More importantly, POS also shows a great advantage in real-time performance, which can inference complex model YOLOP at the speed of 15.6*FPS* even on the Jetson Nano that has the most limited resources. The results demonstrate that POS is able to find better schedule than the baseline frameworks. Furthermore, it also reveals the scalability advantage of POS, since it can accelerate the inference of multi-model simultaneously on GPU platforms with different architectures. With the learning-based scheduling algorithm MEOS, POS has the power to find the optimal schedule for different multi-model combinations and GPU accelerators automatically, which significantly relieving the manual tuning efforts.



(a) R18+R34+R50



(b) YOLOP

Figure 12: The Scheduling Algorithm Comparison. This experiment is conducted on NVIDIA RTX 3080 GPU.

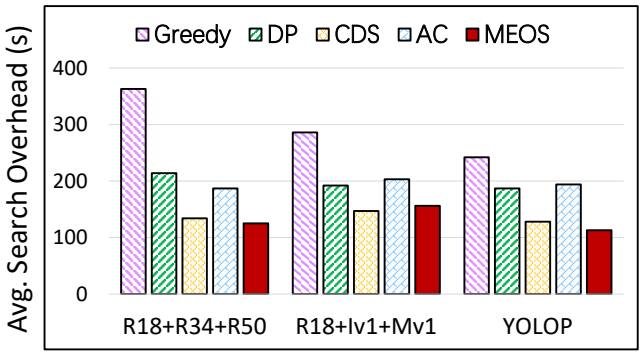


Figure 13: Scheduling overhead for different scheduling algorithms on NVIDIA RTX 3080 GPU.

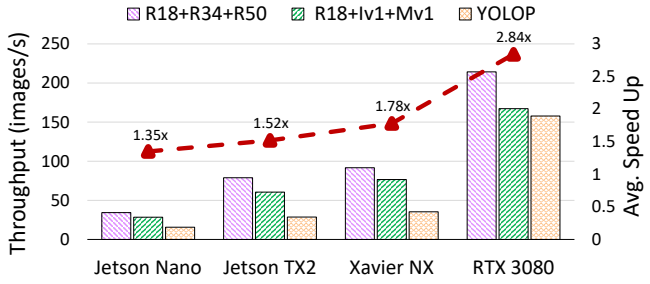


Figure 14: The end-to-end inference performance of POS on heterogeneous GPU platforms.

5.6.2 Different batch sizes. In reality, we normally increase the batch size to infer a series of input data concurrently in order to fully utilize the computing resources of accelerator. However, increasing batch size will reduce inference latency, which is not available for real-time inference with resource-constrained edge devices. In contrast, the larger batch size improves throughput for the device with abundant computing resource. In practice, the above case is more prevalent in cloud computing. In addition, multi-model with different batch sizes requires distinct operator schedules. Here we use a cloud server with NVIDIA RTX 3080 to evaluate end-to-end performance comparison of all frameworks for inferring YOLOP with different batch sizes. We observe that the throughput increases with the batch size in Fig. 15. Similarly, the throughput of POS consistently outperforms all baselines in the case of different batch sizes. The results illustrates that POS can generate optimal schedule automatically for different batch sizes. Note that the throughput tends to saturate or even drop slightly when the batch size is larger than 32. The reason is that the access to shared resources by multi-model will conflict when the batch size is too large, resulting in reduced throughput.

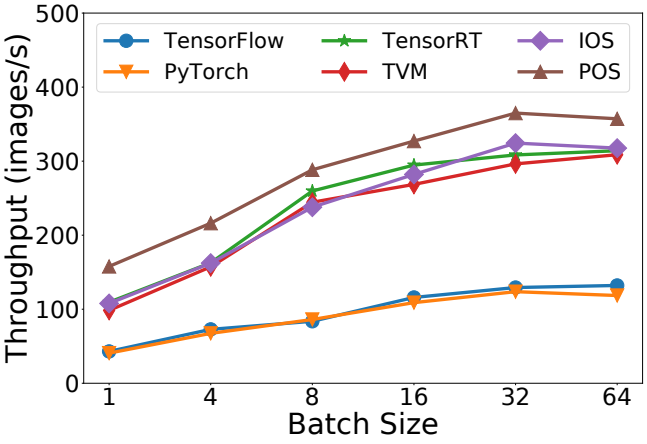


Figure 15: The throughput comparison of different deep learning inference frameworks on batch size 1 to 64 for YOLOP on NVIDIA RTX 3080 GPU.

5.7 Ablation Study

Fig. 16 compares the inference performance of different operator scheduling strategies in POS with three multi-model combinations. We separately evaluate the impact of four strategies, including operator fusion (denoted as POS-Op.Fusion), subgraph reuse (denoted as POS-Reuse), inter-operator parallelism (denoted as POS-Inter-Op. .Para) and intra-operator parallelism (denoted as POS-Intra-Op.Para). Compared with sequential, the results with different strategies of POS all show significant throughput improvement. Moreover, the inference performance increases with more strategies added, which validates the effectiveness of different strategies in POS.

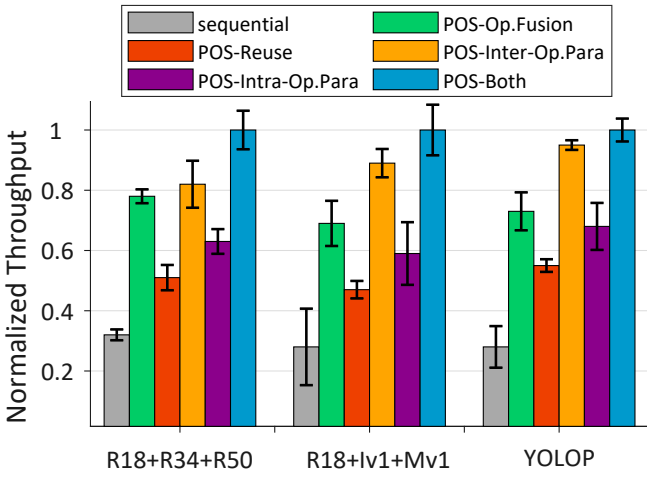


Figure 16: The normalized inference performance of different operator scheduling strategies in POS with three multi-model combinations on NVIDIA Xavier NX GPU.

5.8 GPU Utilization

Warp is the basic execution unit of SM (Streaming Multiprocessor) in GPU. Although GPU utilization can reflect the inference performance of the model, this high-level utilization metric is rough and it cannot judge how many SMs are being used. To this end, we choose achieved occupancy (the ratio of active warps on an SM to the maximum number of active warps supported by the SM) to compare fine-grained low-level GPU utilization with different scheduling algorithms. Fig. 17 illustrates the GPU utilization comparison between Sequential, Greedy, DP, CSD, AC and our scheduling algorithm (MEOS) with different multi-model combinations, sampled using NVIDIA's nvprof profiling tools every 1ms. Specifically, MEOS achieves 1.4× more active warps on average compared to the other scheduling algorithms. The results can also explain the overall performance improvement of POS.

6 RELATED WORK

Single-model acceleration based on operator scheduling. Although model compression [11] (*e.g.*, precision quantization, weight pruning, and model distillation, *etc.*) can significantly accelerate inference, it brings non-negligible information loss that is unacceptable for high-precision scenarios such as autonomous driving. To address the issue, various operator-level schedulers [2] [3] [41] are proposed from the perspective of automating compilation to achieve acceleration without losing accuracy. In addition, the scheduler in

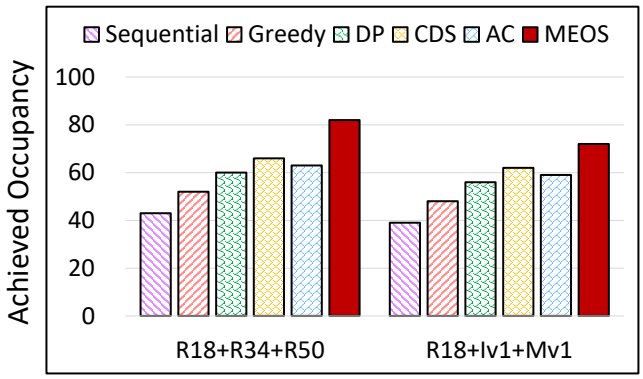


Figure 17: The GPU utilization of different scheduling algorithms for infer YOLOP on NVIDIA Jetson Xavier NX.

deep learning framework only utilize intra-operator parallelism that leads to inefficiency. To this end, some works [22] [18] [42] utilize various operator scheduling strategies to improve deep learning-based model parallelism. However, the above work are only suitable to accelerate single-model, and the heuristic-based scheduling algorithms made search inefficient.

Multi-model acceleration based on operator scheduling. In related work on accelerating multi-model inference, Wang et al. [34] proposed horizontally fused training array (HFTA). HFTA horizontally fuses operators in multi-model in shared accelerator to improve GPU utilization. On the one hand, HFTA is aimed at model training. On the other hand, it only fuse operators with the same type. Wang et al. [33] proposed a scheduling framework AsyMo for on-device inference. AsyMo utilizes an asymmetric-aware task scheduling mechanism to solve the correct partitioning and fair scheduling for multi-model. Since AsyMo only targets mobile CPUs, it's powerless on edge device with accelerators. Han et al. [10] designed an edge inference system REEF, and proposed a dynamic kernel filling-based concurrency mechanism to improve overall system throughput. However, REEF only considered inter-operator parallelism, and the parallelism in operators is still limited.

7 DISCUSSION

Integration with existing technology. Here we discuss the combination of existing model acceleration techniques with POS. First, for a series of model compression paradigms, POS is orthogonal to these techniques, therefore it is natural to seamlessly combine compressed models with POS to further reduce end-to-end latency. Second, POS can also be organically combined with edge computing [29]. Specifically, we can perform POS to the cloud with abundant computing resources to search for the optimal schedule, and the generated schedule is deployed online to resource-constrained edge devices. Moreover, we can offload part of computing tasks from local device to remote server to achieve cloud-edge collaborative inference [19] in order to alleviate the workload on local devices. Overall speaking, POS demonstrates extensive practicality and scalability, and we hope that POS will become a general inference framework for research on efficient intelligence applications.

Limitations and future work. Although POS can sufficiently accelerate multi-model inference, our work still has some limitations. The auto-scheduler in POS only support the CNN-based inference model with static shape, which means that all shapes must be known at the compile time, since POS needs these information to construct the scheduling search space. However, modern deep neural networks, especially natural language processing, introduce control flow [30], dynamic data structures [20, 32], and dynamic tensor shapes into the models [6], such as transformer-based language and vision models. Therefore, optimizing dynamic neural networks is more challenging than static neural networks, requiring the operator scheduler to consider all possible tensor shapes. Although some works have investigated the acceleration of dynamic models [28, 40], they are only suitable for model training. With the successful application of dynamic models, in the future, we plan to develop an efficient inference framework for the deep learning models with dynamic shapes to fill this gap.

8 CONCLUSION

In this work, we propose POS, a novel operator-level scheduling framework on GPUs to accelerate multi-model inference concurrently. We first transform the problem of minimizing the overall inference latency of multi-model into a fine-grained operator scheduling problem. Based on the scheduling search space of the computation graph, we develop a novel maximum entropy reinforcement learning-based operator scheduling algorithm, namely MEOS, which leverages four operator scheduling strategies to find the optimal schedule automatically. We conduct extensive experiments on diverse multi-models. The results illustrate that POS consistently achieves 1.2×~3.9× inference speedup compared to five state-of-the-art deep learning inference frameworks, with 40% improvement on GPU utilization. Meanwhile, the average scheduling overhead of MEOS in POS is reduced by 37% compared with other baselines.

ACKNOWLEDGMENTS

We thank our anonymous reviewers and shepherd for the helpful comment and feedback. This work is partly supported by the National Key R&D Program of China under Grant No. 2021ZD0110905, and An Open Competition Project of Heilongjiang Province, China, on Research and Application of Key Technologies for Intelligent Farming Decision Platform, under Grant No. 2021ZXJ05A03.

REFERENCES

[1] Jiasi Chen and Xukan Ran. 2019. Deep learning with edge computing: A review. *Proc. IEEE* 107, 8 (2019), 1655–1674.
[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] Tianqi Chen, Lianmin Zheng, Eddie Yan, Ziheng Jiang, Thierry Moreau, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. 2018. Learning to optimize tensor programs. *Advances in Neural Information Processing Systems* 31 (2018).
[4] Yudong Chen, Xin Wang, Miao Fan, Jizhou Huang, Shengwen Yang, and Wenwu Zhu. 2021. Curriculum meta-learning for next POI recommendation. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 2692–2702.
[5] Petros Christodoulou. 2019. Soft actor-critic for discrete action settings. *arXiv preprint arXiv:1910.07207* (2019).

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[7] Yaoyao Ding, Ligeng Zhu, Zhihao Jia, Gennady Pekhimenko, and Song Han. 2021. Ios: Inter-operator scheduler for cnn acceleration. *Proceedings of Machine Learning and Systems* 3 (2021), 167–180.

[8] Andreas Geiger, Philip Lenz, and Raquel Urtasun. 2012. Are we ready for autonomous driving? the kitti vision benchmark suite. In *2012 IEEE conference on computer vision and pattern recognition*. IEEE, 3354–3361.

[9] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. 2018. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905* (2018).

[10] 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)*. 539–558.

[11] Song Han, Huizi Mao, and William J Dally. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149* (2015).

[12] 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.

[13] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. 2019. Searching for mobilenetv3. In *Proceedings of the IEEE/CVF international conference on computer vision*. 1314–1324.

[14] Siheon Jeong, Donggeun Kim, San Kim, Ji-Wan Ham, Jae-Kyung Lee, and Ki-Yong Oh. 2020. Real-time environmental cognition and sag estimation of transmission lines using UAV equipped with 3-D Lidar system. *IEEE Transactions on Power Delivery* 36, 5 (2020), 2658–2667.

[15] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. 2021. Highly accurate protein structure prediction with AlphaFold. *Nature* 596, 7873 (2021), 583–589.

[16] Yeongmin Ko, Younkwan Lee, Shoaib Azam, Farzeen Munir, Moongu Jeon, and Witold Pedrycz. 2021. Key points estimation and point instance segmentation approach for lane detection. *IEEE Transactions on Intelligent Transportation Systems* (2021).

[17] Neetesh Kumar, Sarthak Mittal, Vaibhav Garg, and Neeraj Kumar. 2021. Deep Reinforcement Learning-Based Traffic Light Scheduling Framework for SDN-Enabled Smart Transportation System. *IEEE Transactions on Intelligent Transportation Systems* 23, 3 (2021), 2411–2421.

[18] Woosuk Kwon, Gyeong-In Yu, Eunji Jeong, and Byung-Gon Chun. 2020. Nimble: Lightweight and parallel gpu task scheduling for deep learning. *Advances in Neural Information Processing Systems* 33 (2020), 8343–8354.

[19] Stefanos Laskaridis, Stylianos I Venieris, Mario Almeida, Ilias Leontiadis, and Nicholas D Lane. 2020. SPINN: synergistic progressive inference of neural networks over device and cloud. In *Proceedings of the 26th annual international conference on mobile computing and networking*. 1–15.

[20] Xiaodan Liang, Xiaohui Shen, Jiashi Feng, Liang Lin, and Shuicheng Yan. 2016. Semantic object parsing with graph lstm. In *European Conference on Computer Vision*. Springer, 125–143.

[21] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2015. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971* (2015).

[22] Lingxiao Ma, Zhiqiang Xie, Zhi Yang, Jilong Xue, Youshan Miao, Wei Cui, Wenxiang Hu, Fan Yang, Lintao Zhang, and Lidong Zhou. 2020. Rammer: Enabling Holistic Deep Learning Compiler Optimizations with {rTasks}. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. 881–897.

[23] 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.

[24] Kun Qian, Shilin Zhu, Xinyu Zhang, and Li Erran Li. 2021. Robust multimodal vehicle detection in foggy weather using complementary lidar and radar signals. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 444–453.

[25] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. 2020. Mastering atari, go, chess and shogi by planning with a learned model. *Nature* 588, 7839 (2020), 604–609.

[26] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. 2015. Trust region policy optimization. In *International conference on machine learning*. PMLR, 1889–1897.

[27] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).

[28] 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 Machine Learning and Systems* 3 (2021), 208–222.

[29] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. 2016. Edge computing: Vision and challenges. *IEEE internet of things journal* 3, 5 (2016), 637–646.

[30] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. *Advances in neural information processing systems* 27 (2014).

[31] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. 2016. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2818–2826.

[32] Kai Sheng Tai, Richard Socher, and Christopher D Manning. 2015. Improved semantic representations from tree-structured long short-term memory networks. *arXiv preprint arXiv:1503.00075* (2015).

[33] Manni Wang, Shaohua Ding, Ting Cao, Yunxin Liu, and Fengyuan Xu. 2021. Asymo: scalable and efficient deep-learning inference on asymmetric mobile cpus. In *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*. 215–228.

[34] Shang Wang, Peiming Yang, Yuxuan Zheng, Xin Li, and Gennady Pekhimenko. 2021. Horizontally Fused Training Array: An Effective Hardware Utilization Squeezer for Training Novel Deep Learning Models. *Proceedings of Machine Learning and Systems* 3 (2021), 599–623.

[35] Dong Wu, Manwen Liao, Weitian Zhang, and Xinggang Wang. 2021. Yolop: You only look once for panoptic driving perception. *arXiv preprint arXiv:2108.11250* (2021).

[36] Lanyu Xu, Arun Iyengar, and Weisong Shi. 2020. CHA: A caching framework for home-based voice assistant systems. In *2020 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 293–306.

[37] Fuxun Yu, Shawn Bray, Di Wang, Longfei Shangguan, Xulong Tang, Chenchen Liu, and Xiang Chen. 2021. Automated Runtime-Aware Scheduling for Multi-Tenant DNN Inference on GPU. In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 1–9.

[38] 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).

[39] Xihai Zhang, Zhanyuan Cao, and Wenbin Dong. 2020. Overview of edge computing in the agricultural internet of things: key technologies, applications, challenges. *Ieee Access* 8 (2020), 141748–141761.

[40] Bojian Zheng, Ziheng Jiang, Cody Hao Yu, Haichen Shen, Joshua Fromm, Yizhi Liu, Yida Wang, Luis Ceze, Tianqi Chen, and Gennady Pekhimenko. 2022. DietCode: Automatic Optimization for Dynamic Tensor Programs. *Proceedings of Machine Learning and Systems* 4 (2022), 848–863.

[41] 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.

[42] Lianmin Zheng, Zhuohan Li, Hao Zhang, Yonghao Zhuang, Zhifeng Chen, Yanping Huang, Yida Wang, Yuanzhong Xu, Danyang Zhuo, Eric P. Xing, Joseph E. Gonzalez, and Ion Stoica. 2022. Alpa: Automating Inter- and Intra-Operator Parallelism for Distributed Deep Learning. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. 559–578.