TaiChi: Efficient Execution for Multi-DNNs Using Graph-based Scheduling

Xilang Zhou, Haodong Lu, Tianchen Wang, Zhuoheng Wan, Jianli Chen, Jun Yu and Kun Wang†
State Key Lab of Integrated Chips & Systems, and School of Microelectronics, Fudan University, Shanghai, China
†kun.wang@ieee.org

***Abstract*—Applications constructed with multiple Deep Neural Networks (multi-DNNs) are growing rapidly in edge and data center. However, executing multi-DNNs efficiently remains challenging because multi-DNNs are inherently heterogeneous. The diverse operators, dependencies and performance requirements of multi-DNNs lead to high costs of encoding and generalization. We introduce Taichi, a graph-based framework for efficiently scheduling multi-DNNs on multi-core accelerators. Specifically, Taichi consists of two phases: (1) a graph neural network (GNN) is utilized to automatically capture the features from the graph structure of multi-DNNs and (2) reinforcement learning (RL) is employed to find an optimal online scheduling strategy. Evaluation results show that TaiChi reduces latency by 1.1-2.4× and 1.1-1.6× compared to SJF and MAGMA, and improves throughput by 26.4-63.7% and 18.6-33.7%, respectively. Moreover, TaiChi achieves an average speedup of 779× in scheduling runtime compared to MAGMA.**

I. Introduction

Multiple deep neural networks (multi-DNNs) have pervaded across a wide range of applications. In data center, multi-tenancy tasks [1], [6], [17] require the concurrent execution of diverse DNNs with different functions. In edge, multiple DNNs are involved to achieve accurate perception [5], [8] or complete complex tasks [14], [20]. To support efficient execution of these applications, multi-core accelerators [7], [13], [15], [18], [21], [24] have been proposed recently. These specific accelerators offer advantages such as low latency and high energy efficiency compared to general-purpose processors (e.g., CPU, GPU).

However, executing multi-DNNs on multi-core accelerators efficiently is challenging. First, multi-DNNs are inherently heterogeneous. As illustrated in Fig. 1, the heterogeneity of multi-DNNs is exhibited in two aspects: different types of operators (e.g., convolution, attention) and different dependencies of models (e.g., pipeline, concurrent, independent). Capturing these heterogeneous features results in a high cost of encoding. Second, the scheduling space is extremely massive, which is the combination of the choices of operators execution orders and accelerators selection. It is hard to achieve high scheduling quality with low scheduling runtime.

Up to now, several works have been proposed to address scheduling problem for multi-DNNs as listed in Table I. For data center, AI-MT [1] is manually designed to schedule multi-DNNs at the layer level. MAGMA [9] and COMB [23] both use genetic algorithm to enhance the search efficiency, additionally, COMB considers memory optimization. MoCA [10] utilizes dynamic memory management for multi-DNNs to achieve performance improvement. For edge, Herald [13] and DREAM [11] are both designed for real-time multi-DNNs. Herald proposes a heterogeneous dataflow architecture, while DREAM introduces a dynamic scheduler that adapts to dynamic workloads. DySta [4] mainly focuses on sparse multi-DNN workloads.

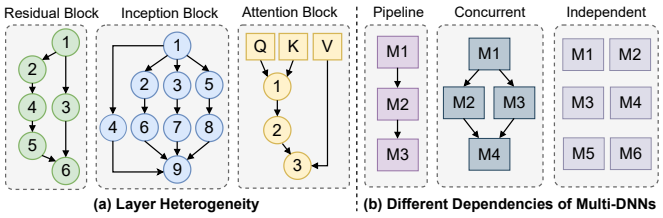

Fig. 1. The characteristics of multi-DNNs. (a) The layers of different models are heterogeneous and they expose different parallelisms. (b) In edge, multi-DNNs perform complex tasks by cascading or partial paralleling networks and in data center multi-DNNs come from different users with no dependencies between DNNs.

In the above scheduling frameworks, however, three deficiencies exist: 1) The graph structure of multi-DNNs is ignored; 2). The supported scenarios are limited; 3). The scheduling quality and time is unbalanced. To bridge the gap, we propose TaiChi, a graph-based scheduling framework that can efficiently execute multi-DNNs on multi-core accelerators. Specifically, we model the scheduling problem as a Markov decision process and train a lightweight neural network to solve the problem. The model consists two parts: graph neural network (GNN) and reinforcement learning (RL). GNN is used to automatically encoding the graph structure of multi-DNNs and capture the heterogeneous features of operators and dependencies. RL is employed to learn the excellent scheduling strategies in the huge search space. After training, the integrated GNN-RL model can obtain the high-quality scheduling results through a singe inference.

Compared with the existing scheduling frameworks, TaiChi has the following advanced features:

1) TaiChi is the first graph-based scheduling framework for multi-DNNs that supports automated encoding.
2) TaiChi adapts different scheduling scenarios at data center and edge without the cost of manual redesign.
3) TaiChi is a lightweight scheduler that can reduce the scheduling runtime significantly with high performance.

In evaluation, TaiChi achieves 1.11-2.37× and 1.11-1.63×

† Corresponding author

978-3-9826741-0-0/DATE25/© 2025 EDAA

TABLE I
A COMPARISON OF TAICHI AND EXSITING SCHEDULERS.

Works	Scheduling Algorithm	Scenario Support	Automatic Encoding	Online Scheduling
AI-MT [1]	Manual	Data Center	✗	✓
Herald [13]	Manual	Edge	✗	✓
MAGMA [9]	GA	Data Center	✗	✗
MoCA [10]	Manual	Data Center	✗	✓
H3M [22]	CMA-ES	Data Center	✗	✗
COMB [23]	GA	Data Center	✗	✗
DREAM [11]	Manual	Edge	✗	✓
DySta [4]	Manual	Edge	✗	✓
TaiChi	**GNN+RL**	**Edge & Data Center**	✓	✓

latency reduction and 26.4-63.7% and 18.6-33.7% throughput improvement compared to SJF and MAGMA, respectively. Moreover, TaiChi obtains an average speedup of 779 × in scheduling runtime compared to MAGMA.

II. PROBLEM FORMULATION

A. Scheduling Goals

In our design, three aspects are explored, including automatic encoding, multiple scenarios support, and online scheduling. Specifically, automatic encoding is used to automatically extract the raw information of multi-DNNs form ONNX while satisfying model and layer dependency; multi-scenario support aims to recast the scheduling problem as a sequential decision process, so as to adapt different scenarios by simply switching the objective function; and online scheduling is designed to explore a large scheduling space and realize low scheduling delay for real-time tasks. To realize the three goals, GNN and RL are jointly adopted, considering that GNN can capture graph structure and RL can solve sequential decision problems in uncertain environments with low scheduling overhead.

B. Input and Output

In scheduling problem, the input can be represented as a directed acyclic graph (DAG) $G = (V, E)$ to represent the multi-DNNs that need to be performed, where V is composed of all the layers in the multi-DNN and E represents connections between layers or models. Each node has a feature vector: $F(f_1, f_2, \ldots, f_N)$, where N is the number of features. In actual code, the DAG is transformed as a relational matrice and feature matrice for efficient computation. The output is a scheduling table $T_{\text{core}_i} = [M_1L_1, M_1L_2, ..., M_iL_j]$, where M_iL_j represents the j-th layer of the i-th model. The table records which node executed on $core_i$ and the execution order.

C. Scheduling Framework

Fig. 2 depicts the overall framework of TaiChi. In TaiChi, the input is a DAG of multi-DNN workloads, while the output result is a scheduling table. Between the input and the output, a scheduling agent with GNN and RL integrated is used to find an excellent scheduling strategy. The implementation of scheduling agent consists of two components. In the first one, GNN is utilized to perform automatic feature extraction from the DAG by aggregating nodes features. In the second one, RL is trained to continually interact with the environment to update the policy network. The environment is determined by the hardware configuration of multi-core accelerators includes the number of cores, the dimension of processing elements (PEs), on-chip memory, and bandwidth. When scheduling, the first output of scheduling agent is the scheduling scheme.

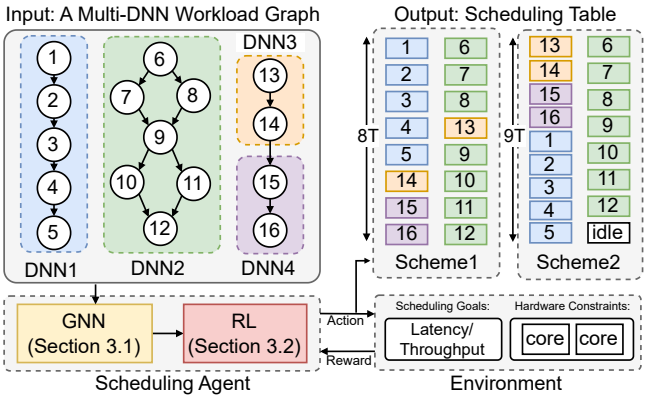

Fig. 2. An illustration of scheduling process in TaiChi. The input is a multi-DNN workloads contains four DNN models (DNN3 and DNN4 are cascaded), where latency of each layer is 1T. On the 2-core accelerator, the scheduling results for TaiChi and SJF are 8T and 9T, respectively.

III. METHOD

In this section, we present our scheduling algorithm in details. First, the raw information of multi-DNNs are mapped into one-dimensional vectors using GNN. Second, the scheduling process is modeled as a Markov decision process (MDP) and RL is employed to obtain an optimal scheduling policy. Third, a dataset is set up for training the proposed GNN-RL network architecture.

A. Graph Embedding Generation

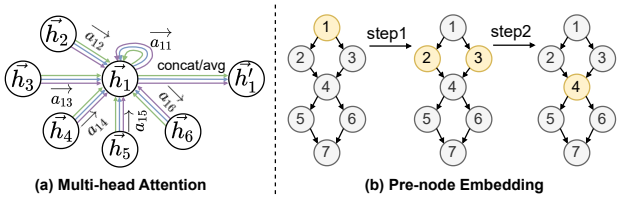

Fig. 3. (a) An illustration of multi-head attention (with $K = 3$ heads) by node 1 on its neighborhood. Different arrow styles and colors denote independent attention computations. The aggregated features from each head are concatenated or averaged to obtain $\vec{h}$. (b) An illustration of message passing.

GNN is used to generate graph embedding from multi-DNN workloads, which is divided into two steps. **Step-1**) The plain node and graph features are selected to form a set of feature vectors, where the choice way of node feature information has an important effect on the agent learning environment. The node features are divided into two parts: the in/out degree of nodes and the mapped node parameters such as latency, power consumption, throughput, and bandwidth. The Graph feature is the total GOPs of a DNN. **Step-2**) Graph attention network (GAT) [19] is utilized to accomplish graph embedding. In TaiChi, GNN is utilized to encode the graph structure information of multi-DNNs into new features. Moreover, the encoding process is learnable. By training on a large amount

of data, the GNN can learn which features are influencing the decision. In this work, GAT, as a kind of GNN model with inductive properties, is employed to perform the feature extraction process.

Owing to the utilization of multi-head attention mechanism, GAT can generalize the aggregation capability to unseen topologies or nodes, which makes TaiChi scalable. The specific procedure for GAT aggregation of node information is as follows.

Given node i, the attention coefficient of it and its neighbor node j is given by

$$\alpha_{ij} = \frac{\exp\left(\text{LeakyReLU}\left(e_{ij}\right)\right)}{\sum_{k \in N_i} \exp\left(\text{LeakyReLU}\left(e_{ik}\right)\right)}, \tag{1}$$

where

$$e_{ij} = a(\mathbf{W}\vec{h}_i \parallel \mathbf{W}\vec{h}_j). \tag{2}$$

$\vec{h}_i$ and $\vec{h'}_i$ represent the feature vectors of node i and a new set of features after feature extraction, respectively. N_i is the set of neighbor nodes for node i. $\parallel$ represents concatenation. $\mathbf{W}$ is a shared linear transformation, which is parametrized by a weight matrix.

For nodes i and j, their transformed feature vectors are concatenated and normalized by the softmax function to calculate their attention coefficients. Based on this, the normalized attention coefficients and the corresponding linear combination of features are used to aggregate the node information to obtain the final node feature $\vec{h'}_i$, which is given by

$$h'_i = \|_{k=1}^{K} \sigma\left(\sum_{j \in N(i)} \alpha_{ij}^k \mathbf{W}^k \vec{h_j}\right), \tag{3}$$

where α_{ij}^k is the normalized attention coefficient. k is the number of attention heads. σ is a nonlinear function. The aggregation process of a multi-head graph attentional layer is illustrated by Fig. 3(a).

Through the above procedure, one message passing, i.e., aggregating the feature information of the first-order neighbors during the feature extraction is completed. In TaiChi, we use multi-step message passing, as shown in Fig. 3(b). In the first step message passing, the features of $node_1$ are passed to $node_2$ and $node_3$ through the graph attentional layer. In the next message passing, the features of $node_2$ and $node_3$ are passed to $node_4$. In this step, $node_2$ and $node_3$ have already included the features of $node_1$. Therefore, $node_4$ can perceive the information of more distant nodes with dependencies. Based on the node features, dependency-aware is realized and better scheduling decisions are obtained in the subsequent decision making of RL.

B. RL-based Scheduling Design

In this subsection, we firstly model the scheduling problem as a Markov Decision Process (MDP). Then, we design RL corresponding modules in solving MDP in a complex, uncertain environment to obtain excellent scheduling results.

In the scheduling problem, the action refers to the assignment of nodes to different cores, while the state refers to the remaining nodes available for scheduling. Due to the random nature of the action, the state is also random. In essence, the scheduling problem of multi-DNN workloads on multi-core accelerators is about how to balance the load of multi-core accelerators and reduce their idle time by leveraging the parallelism. However, inter-layer dependency has a negative impact on the parallel processing. Define the state as the set of nodes that can be executed in parallel at time t. To satisfy the inter-layer dependency, we schedule nodes without dependencies at time t and their children at time $t+1$. The scheduling of the entire multi-DNN workload is done using dynamic sequential decision making. Based on this, the scheduling process is modeled as an MDP.

A Case of RL-based Scheduling. RL is an optimization method based on Markov decision process in complex and uncertain environments, which is often modelled as an agent-environment framework. As shown in Fig. 2, RL is divided into two parts: scheduling agent and environment. By continuously interacting with the environment, the agent adjusts the action based on the returned reward. Moreover, a set of effective strategies for solving the scheduling problem is obtained after multiple rounds of iteration. In TaiChi, the agent and the environment are represented by a scheduling decision maker and a multi-core accelerator, respectively.

To clarify the scheduling decision process based on RL, we take the example described in section 2, as shown in Fig. 4. In this example, it is set that the multi-DNN workload consists of 4 DNN models and the accelerator has 2 cores. The state is characterized by the set of nodes executed in parallel at the moment t, which is called scheduling pool. For example, the three nodes (1, 6, 13) are executed in parallel in state 1. In this state, the agent takes a scheduling action: 1 node is assigned to the core 1; 6 nodes are assigned to core 2; and 13 nodes are unassigned. Thus, the action space is a discrete sequence [1, 1, 0]. Once the first action decision is made, the scheduling pool is updated to produce state 2, in which 4 nodes (2, 7, 8, 13) are executed in parallel and an action sequence [1, 1, 0, 0] is generated. After the agent executes an action, a reward is returned to evaluate the performance of the action. After decision making is repeated n times, the scheduling pool becomes empty. Thus, the multi-DNN scheduling is completed and a final reward is output. A more detailed description is as follows.

Action Space. In RL, the design of the action space affects the speed of convergence of the model. We can take the approach of outputting all actions at once, which has to choose actions from an exponentially large set of combinations. On the other extreme, We can take the method of sequentially output the actions of each node, which requires long sequences of actions to schedule a given set of DNNs. In RL, both large action spaces and long action sequences increase sample complexity and slowdown training.

In our design, each action is the number of nodes that can

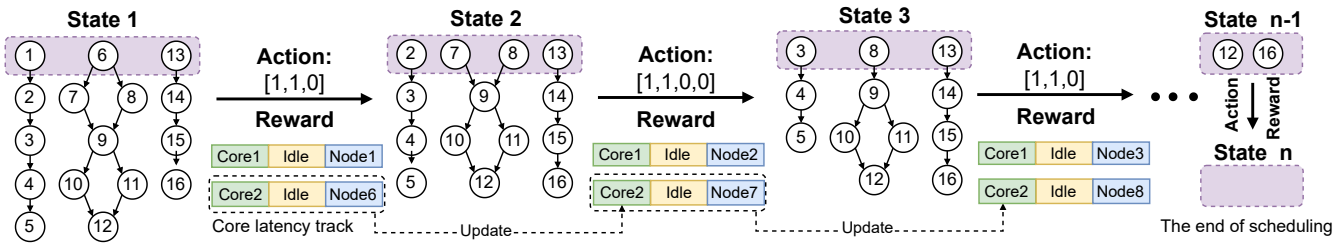

Fig. 4. An illustration for RL-based scheduling process.

be parallelized in a single stage. This treatment balances the size of the action space and the number of actions, placing the decision space too large for the model not to converge or compensating for the short adoption time. Specifically, we take the features h_i' of node i extracted by GAT and through a MLP to obtain a score q_i. The set of parallelizable nodes under the nth Stage is S_n, which is normalized by softmax to obtain the probability of a node. A node with high probability indicates that it needs to be scheduled. Mathematically, q_i and $action_i$ are respectively given by

$$q_i = MLP\left(h_i'\right) \tag{4}$$

$$action_i = \text{softmax}_j(q_i) = \frac{\exp(q_i)}{\sum_{m \in S_n} \exp(q_m)}. \tag{5}$$

Reward. After agent takes a scheduling decision, it will have to obtain a reward. A single-step decision affects the reward of subsequent decisions, so subsequent payoffs are assigned to moment t through a discount factor γ. The objective of RL is to obtain an optimal strategy network π^* that maximizes the value function $v_\pi(s)$. The formulation is as follows

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}, \tag{6}$$

$$v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s], \tag{7}$$

$$v^* = \max_a \mathbb{E}_{\pi^*}[G_t | S_t = s, A_t = a], \tag{8}$$

$$= \max_a \mathbb{E}[R_{t+1} + \gamma v^*(S_{t+1}) | S_t = s, A_t = a], \tag{9}$$

where G_t is the cumulative reward of the action at moment t, π is the scheduling policy and $\mathbb{E}$ denotes the expected value of a random variable under policy π.

In our design, the reward is divided into two parts single-step reward and global reward. The result of the single-step reward is the time that a single action results in a accelerator idle. Specifically, idle time refers to when a node is dispatched to a core that will be idle due to the parent node not finishing execution. The purpose of the single-step decision is to ensure that the multi-DNN accelerator can be load balanced as much as possible for that decision, and the idle time of the core is as small as possible. The global reward is the final execution time of the multi-DNN. Idle time and task time are calculated as follows

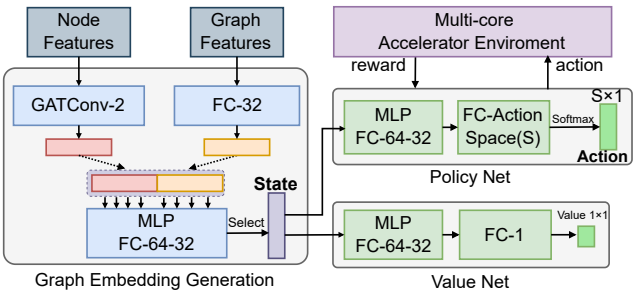

Fig. 5. The neural network architecture of TaiChi.

$$L_{\text{core}_i} = \sum (node_i + idle_i), \tag{10}$$

$$idle_i = \max\left(0, L_{\text{parents}} - L_{\text{core}_i}\right), \tag{11}$$

$$L_{\text{task}} = \max\left(L_{\text{core}_i}\right), \tag{12}$$

where L_{core_i} refers to the moment when all the parents of node i finish scheduling, and $L_parents$ refers to the current execution latency of the core that node i is assigned. The single step reward is the idle time, i.e., $reward_{\text{step}} = idle$. Global rewards are also divided into two categories to accommodate data center and edge applications. The formulation for calculating the global reward is as follows

$$\text{Reward1} = \frac{1}{\alpha L_{\text{task}} + \beta reward_{\text{step}}}, \tag{13}$$

$$\text{Reward2} = \sum \frac{\sum MACs_{\text{node}_j}}{(\alpha L_{\text{ccore}} + \beta reward_{\text{step}})}. \tag{14}$$

In the global reward we mix single-step rewards with the aim of speeding up the RL convergence process. To prevent RL from falling into local optimum, discount factors α and β are set to 0.8 and 0.2, respectively.

C. Training

Dataset. We have selected popular DNN models that cover a wide range of tasks in vision (VGG16, ResNet50, MobileNetV1), speech (GNMT), and language (Transformer). To improve the extensiveness of the dataset, we randomly change three parameters: batch size, number of models, and model dependencies include pipeline and concurrent. Finally, we get a dataset of 500 DAGs.

Neural Network Architecture. The node embedding is completed through a two GATConv layer. The Graph features are embedded through a FC layer. We further pass the aggregated node features and graph features through a two-layer MLP, and this setup can learn the critical path [16]. The output features are selected and fed to RL. The policy network of RL

TABLE II
BENCHMARK OF MULTI-DNN WORKLOADS.

Workloads	DNN Models	#Node
Vision-Light	2 (SqueezeNet → MobileNetV2), 2 ResNet18)	204
Vision-Heavy	2 VGG16, 4 GoogleNet, 4 ResNet50	546
Language	4 GNMT, 2 Transformer	612
Mixed-Light	1 VGG16, 1 GoogleNet, 2 Transformer, 2 (ResNet50 → MobileNetV2)	897
Mixed-Heavy	2 VGG16, 4 GoogleNet, 4 ResNet50 4 MobileNetV2, 4 Transformer	1922

consists of a two-layer MLP. Each node gets the corresponding probability (scheduling score) through the same MLP. Multiple node probabilities are normalized by softmax. Specific details are shown in Fig. 5.

Training strategies. The policy and value outputs of the network reflect the decision-making capability of the RL agent. The agent learns to maximize the objective function gradually via training. In TaiChi, the action space is dynamic with different action lengths for each sample. Therefore, the feature extraction network and the policy network need to be trained separately. At each step, the quadruple is collected (s_t, a_t, r_t, s_{t+1}). For the policy network, once a batch of data has been collected, the weights of the network are updated through stochastic gradient descent. For the GAT network, we need to wait until the scheduling is completely finished before updating the network.

IV. EVALUATION

A. Experiment Setup

Multi-DNN Workloads. We follow the workloads selection in previous work [9], [13]. We use three types of Multi-DNN workloads: Vision (VGG16, SqueezeNet, ResNet50, MobileNetV2, GoogleNet), Language (GNMT, Transformer) and Mixed. By changing the batch size of models and topology in each type, we get totally of five different workloads as shown in Table II, where → indicates that the two networks are connected.

Multi-core Accelerators. The settings of multi-core accelerators are motivated by MAGMA [9] and the simulator is modified by Maestro [12]. To demonstrate TaiChi can support edge and data-center scenarios, we set up two types of accelerators with different sizes of PE dimensions, core number, on-chip storage, and bandwidth, respectively. For each type, we consider three different dataflows: NVDLA-like, Eyeriss-like [2], and ShiDianNao-like [3]. The specific configurations are shown in the Table III.

Baseline. We compare against two scheduling approaches Shortest-Job First (SJF) and MAGMA [9] in evaluation. All scheduling algorithms are deployed on an AMD CPU (EPYC 7352).

Metrics. To assess the performance of different scenarios remand, we utilize two metrics: latency and throughput. Latency is used to evaluate the need for real-time in edge. Throughput is used to evaluate the number of tasks the system can handle currently in data center and is given by $Throughput = \sum \frac{\sum_{j \in N_i} MACs_j}{L_{\text{core}_i}}$ in our work, where $MACs_j$ is the multiplication accumulation number of node N_i is the set of nodes executing on $core_i$.

TABLE III
CONFIGURATIONS OF MULTI-CORE ACCELERATORS.

Setting	# of cores	Dataflow	PE Dim.	Buffer	Bandwidth
E1	4	NVDLA	32×32	146KB	4GB/s
E2	4	Eyeriss	32×32	146KB	
E3	4	ShiDianNao	32×32	146KB	
D1	8	NVDLA	128×128	434KB	16GB/s
D2	8	Eyeriss	128×128	434KB	
D3	8	ShiDianNao	128×128	434KB	

B. Performance

For all workloads, we evaluate the latency of edge accelerators, the throughput of data center accelerators under SJF, MAGMA and TaiChi respectively. The evaluation results show in Fig. 6.

Latency. As the results shown, TaiChi achieves the best latency for all workloads on edge multi-core accelerators. Specifically, TaiChi achieved 1.11-2.37 × latency reduction compared to SJF and 1.11-1.63 × latency reduction compared to MAGMA. Among the three accelerator types of dataflow, Eyeriss shows the best latency results. In addition, due to the large latency of GNMT, which is the critical path in the entire process, GNMT is a linear model that the inter-layer parallelism cannot be achieved by TaiChi. Therefore, the improvement of TaiChi over SJF and MAGMA under language workloads is limited. And under Mixed-heavy workload, compared to MAGMA, which can only capture inter-model parallelism, TaiChi can utilize both inter-model parallelism and inter-layer parallelism, which reduces the hardware idle time.

Throughput. As for throughput, TaiChi uniformly achieves the maximal throughput for all workloads. Overall, TaiChi can improve from 26.4% to 63.7%, 18.6% to 33.7% throughput compared SJF and MAGMA, respectively. Similar to latency, the throughput improvement of TaiChi is limited under language workloads. And we observe that the improvement gained by TaiChi in throughput is not as marked as latency. In the data center, we focus on the computing power of each core, reducing the idle time of a single core, the throughput of the system can be improved. While in the edge scenario, we focus on the optimization of the longest delay path in the task, which requires multi-core system load balancing, which provides more optimization space for TaiChi.

C. Scheduling Runtime

We evaluate the comparison between TaiChi and MAGMA in terms of scheduling runtime under different task with different node sizes. For MAGMA, we fixed the epoch of GA to 100. As shown in Fig. 8, the scheduling runtime of TaiChi has a 187-1653 × improvement compared to MAGMA for various

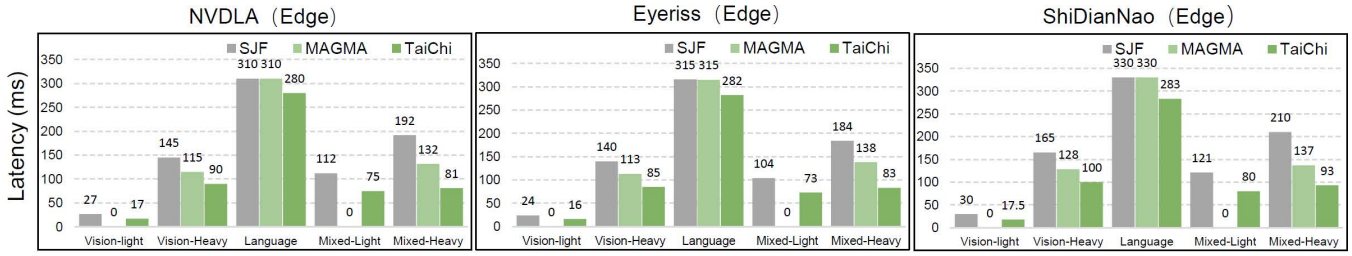

Fig. 6. The latency evaluation on different workloads and edge multi-core accelerators under SJF, MAGMA and TaiChi. The missing data is due to limitations of MAGMA–which dose not accept format with networks dependency.

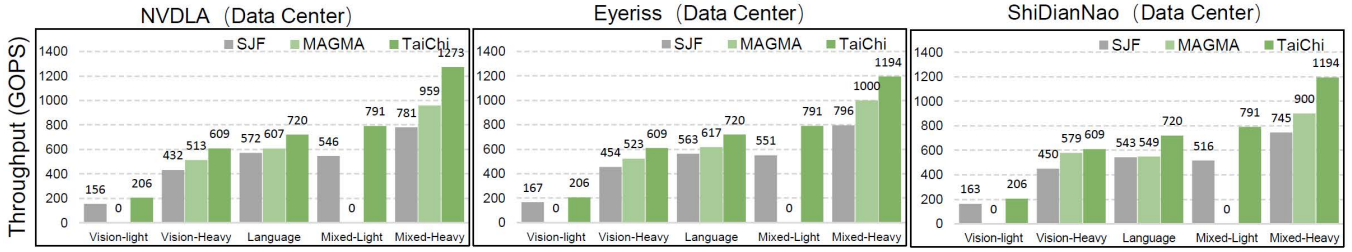

Fig. 7. The throughput evaluation on different workloads and data center multi-core accelerators under SJF, MAGMA and TaiChi. The missing data is due to limitations of MAGMA–which dose not accept format with networks dependency.

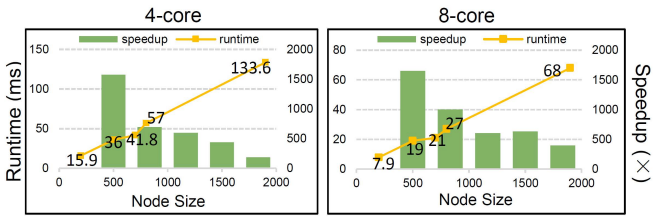

Fig. 8. The scheduling runtime of TaiChi and the speedup of TaiChi compared to MAGMA.

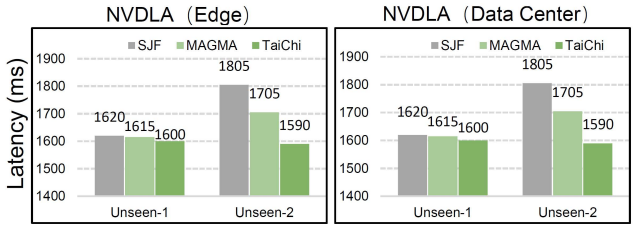

Fig. 9. The performance of TaiChi under unseen workloads.

sizes of workloads when exceeding the MAGMA scheduling quality. Specifically, when scheduling a workload with 1972 nodes, the runtime of TaiChi for the 4-core accelerators and 8-core accelerators are 133.68ms and 68.50ms, respectively. On average, for 4-core accelerators and 8-core accelerators, the runtime is 6.8ms/100 nodes and 4.3ms/100nodes, respectively. Theoretically, the runtime of TaiChi is the inference time of GAT and MLP, which is formulated by $L_{\text{TaiChi}} = L_{\text{GAT}} + L_{\text{MLP}} \times \frac{num\_node}{num\_core}$.

The algorithmic complexity of TaiChi is $O(n)$. Furthermore, as shown in the results, the scheduling runtime of TaiChi increases roughly linearly with node sizs, which is consistent with Eq. (8). While in MAGMA, the number of nodes increases and the scheduling space expands. The number of mutation, crossover and epoch needs to be increased to meet performance demand. Hence the scheduling runtime of MAGMA increases super linearly. Overall, by inferencing through a trained network, TaiChi can satisfy the online scheduling requirement.

D. Generalization

The multi-DNN workloads set up in Table II are avoided to be same as the workloads in dataset by having different batch sizes and connectivity relations. However, to further explore the generalization of TaiChi, we consider completely new workloads, which contain three unseen DNN models: Unet, ResNetX, and Mansnet, whose topologies are unlearned by TaiChi. Specifically, we set up two workloads: unkseen-1 (2 Unet, 2 ResNetX, 2 MansNet) and unseen-2 (4 UNet, 8 ResNetX, 8 MansNet). We also run these workloads on edge and data center accelerators. As shown in Fig. 9, under unseen workloads, the scheduling quality of TaiChi is higher than that of SJF and MAGMA. Moreover, TaiChi guarantees high performance without any extra cost (e.g., encoding, fine tuning). Generalization experiments demonstrate that TaiChi is able to adapt to complex scheduling scenarios of multi-DNN workloads.

V. Conclusion

In this paper, we introduce Taichi, a framework for efficiently executing multi-DNNs on multi-core accelerators via graph-based scheduling. With GNN-RL integration, TaiChi can automatically capture the graph structure information of multi-DNNs and quickly find an optimal solution from a huge scheduling space in most of scheduling scenarios. Evaluation shows Taichi significantly improves the latency and throughput of multi-DNNs while has a low cost in encoding and scheduling runtime. In addition, our observation that TaiChi achieves better performance under vision workloads than language workloads motivates us to explore the parallelism of multiple language models in the future work.

Acknowledgment

This work was financially supported in part by Project of Shanghai EC (24KXZNA12). The computaions in this research were performed using the CFFF platform of Fudan University.

REFERENCES

[1] E. Baek, D. Kwon, and J. Kim, "A multi-neural network acceleration architecture," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020, pp. 940–953.

[2] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE journal of solid-state circuits*, vol. 52, no. 1, pp. 127–138, 2016.

[3] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam, "Shidiannao: Shifting vision processing closer to the sensor," in *Proceedings of the 42nd annual international symposium on computer architecture*, 2015, pp. 92–104.

[4] H. Fan, S. I. Venieris, A. Kouris, and N. Lane, "Sparse-dysta: Sparsity-aware dynamic and static scheduling for sparse multi-dnn workloads," in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, 2023, pp. 353–366.

[5] S. Ghafouri, K. Razavi, M. Salmani, A. Sanaee, T. Lorido-Botran, L. Wang, J. Doyle, and P. Jamshidi, "Ipa: Inference pipeline adaptation to achieve high accuracy and cost-efficiency," Aug. 2023.

[6] S. Ghodrati, B. H. Ahn, J. Kyung Kim, S. Kinzer, B. R. Yatham, N. Alla, H. Sharma, M. Alian, E. Ebrahimi, N. S. Kim, C. Young, and H. Esmaeilzadeh, "Planaria: Dynamic architecture fission for spatial multi-tenant acceleration of deep neural networks," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. Athens, Greece: IEEE, Oct. 2020, pp. 681–697.

[7] P. Houshmand, G. M. Sarda, V. Jain, K. Ueyoshi, I. A. Papistas, M. Shi, Q. Zheng, D. Bhattacharjee, A. Mallik, P. Debacker, D. Verkest, and M. Verhelst, "Diana: An end-to-end hybrid digital and analog neural network soc for the edge," *IEEE Journal of Solid-State Circuits*, vol. 58, no. 1, pp. 203–215, 2023.

[8] Y. Hu, J. Yang, L. Chen, K. Li, C. Sima, X. Zhu, S. Chai, S. Du, T. Lin, W. Wang *et al.*, "Planning-oriented autonomous driving," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 17 853–17 862.

[9] S.-C. Kao and T. Krishna, "Magma: An optimization framework for mapping multiple dnns on multiple accelerator cores," in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2022, pp. 814–830.

[10] S. Kim, H. Genc, V. V. Nikiforov, K. Asanović, B. Nikolić, and Y. S. Shao, "Moca: Memory-centric, adaptive execution for multi-tenant deep neural networks," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2023, pp. 828–841.

[11] S. Kim, H. Kwon, J. Song, J. Jo, Y.-H. Chen, L. Lai, and V. Chandra, "Dream: A dynamic scheduler for dynamic real-time multi-model ml workloads," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 4*, 2023, pp. 73–86.

[12] H. Kwon, P. Chatarasi, M. Pellauer, A. Parashar, V. Sarkar, and T. Krishna, "Understanding reuse, performance, and hardware cost of dnn dataflow: A data-centric approach," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 754–768.

[13] H. Kwon, L. Lai, M. Pellauer, T. Krishna, Y.-H. Chen, and V. Chandra, "Heterogeneous dataflow accelerators for multi-dnn workloads," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2021, pp. 71–83.

[14] H. Kwon, K. Nair, J. Seo, J. Yik, D. Mohapatra, D. Zhan, J. Song, P. Capak, P. Zhang, P. Vajda *et al.*, "Xrbench: An extended reality (xr) machine learning benchmark suite for the metaverse," *Proceedings of Machine Learning and Systems*, vol. 5, 2023.

[15] S. Li, X. Zhou, H. Lu, and K. Wang, "Dnnmapper: An elastic framework for mapping dnns to multi-die fpgas," in *2024 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2024, pp. 1–5.

[16] H. Mao, M. Schwarzkopf, S. B. Venkatakrishnan, Z. Meng, and M. Alizadeh, "Learning scheduling algorithms for data processing clusters," in *SIGCOMM*, 2019, pp. 270–288.

[17] J. Park, M. Naumov, P. Basu, S. Deng, A. Kalaiah, D. Khudia, J. Law, P. Malani, A. Malevich, S. Nadathur, J. Pino, M. Schatz, A. Sidorov, V. Sivakumar, A. Tulloch, X. Wang, Y. Wu, H. Yuen, U. Diril, D. Dzhulgakov, K. Hazelwood, B. Jia, Y. Jia, L. Qiao, V. Rao, N. Rotem, S. Yoo, and M. Smelyanskiy, "Deep learning inference in facebook data centers: Characterization, performance optimizations and hardware implications," Nov. 2018.

[18] Y. S. Shao, J. Clemons, R. Venkatesan, B. Zimmer, M. Fojtik, N. Jiang, B. Keller, A. Klinefelter, N. Pinckney, P. Raina, S. G. Tell, Y. Zhang, W. J. Dally, J. Emer, C. T. Gray, B. Khailany, and S. W. Keckler, "Simba: Scaling deep-learning inference with multi-chip-module-based architecture," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. Columbus OH USA: ACM, Oct. 2019, pp. 14–27.

[19] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[20] S. I. Venieris, C.-S. Bouganis, and N. D. Lane, "Multiple-deep neural network accelerators for next-generation artificial intelligence systems," *Computer*, vol. 56, no. 3, pp. 70–79, Mar. 2023.

[21] M. Verhelst, M. Shi, and L. Mei, "Ml processors are going multi-core: A performance dream or a scheduling nightmare?" *IEEE Solid-State Circuits Magazine*, vol. 14, no. 4, pp. 18–27, 2022.

[22] S. Zeng, G. Dai, N. Zhang, X. Yang, H. Zhang, Z. Zhu, H. Yang, and Y. Wang, "Serving multi-dnn workloads on fpgas: A coordinated architecture, scheduling, and mapping perspective," *IEEE Transactions on Computers*, vol. 72, no. 5, pp. 1314–1328, 2022.

[23] S. Zheng, S. Chen, and Y. Liang, "Memory and computation coordinated mapping of dnns onto complex heterogeneous soc," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–6.

[24] X. Zhou, S. Li, H. Lu, and K. Wang, "Pipefuser: Building flexible pipeline architecture for dnn accelerators via layer fusion," in *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2024, pp. 921–926.