Collaborative DNNs Inference with Joint Model Partition and Compression in Mobile Edge-Cloud Computing Networks

Yaxin Tang[1], Xiuhua Li[1*], Hui Li[1], Zhengyi Yang[1], Xiaofei Wang[2], and Victor C. M. Leung[3, 4]
[1]School of Big Data & Software Engineering, Chongqing University, Chongqing, China
[2]TKLAN, College of Intelligence & Computing, Tianjin University, Tianjin, China
[3]College of Computer Science & Software Engineering, Shenzhen University, Shenzhen, China
[4]Department of Electrical & Computer Engineering, The University of British Columbia, Vancouver, Canada
*Corresponding author: Xiuhua Li
Email:yaxintang@stu.cqu.edu.cn, {lixiuhua, h.li, zyyang}@cqu.edu.cn, xiaofeiwang@tju.edu.cn, vleung@ieee.org

***Abstract*—Mobile edge-cloud computing utilizes the computing resources of edge devices and cloud servers to execute complex deep neural networks (DNNs) for collaborative inference. However, many existing collaborative inference methods do not fully consider the limited resources of edge devices, resulting in high inference latency. In this paper, we design an integrated computational framework that combines model partition and compression to reduce inference latency. Specifically, we partition a DNN model at the middle layer and deploy the previous layer on the edge device and the subsequent layer on the cloud server respectively. We propose a collaborative dual-agent reinforcement learning algorithm called CPCDRL to determine partition point and compression ratios. It enables adaptive adjustments of compression ratios based on various partition points, with the overarching goal of minimizing the inference latency across the entire DNN model. The proposed algorithm can significantly reduce computational latency while minimizing accuracy loss compared to the baseline schemes.**

I. Introduction

Recently, artificial intelligence (AI) has made significant advancements in domains such as computer vision and speech recognition, owing to its precise predictive abilities [1]. However, AI applications frequently demand the substantial computational capabilities provided by cloud servers [2]. Concurrently, the widespread adoption of the internet-of-things has led to an exponential increase in data generated by edge devices [3]. The conventional approach of uploading data to the cloud consumes substantial bandwidth, resulting in unacceptable transmission delays for edge devices [4].

To address this formidable challenge, edge intelligence (EI) [5] has emerged as a viable solution. In general, EI relocates deep neural networks(DNNs) inference tasks closer to the data source [6], [7]. A specific form of EI leverages the combined computational capabilities of both edge and cloud environments for executing DNNs inference [8]. In this collaborative approach, a DNN model is partitioned at an intermediate layer and deployed separately on edge devices and cloud servers [9]. Compared to traditional single-end approaches, collaborative inference offers lower latency and energy consumption, enhancing the overall performance of AI applications. It is crucial to determine how to partition and deploy DNN models, achieving a balance between preserving model accuracy and minimizing inference latency.

Several researches have focused on collaborative inference in edge-cloud computing. *Neurosurgeon* marked a significant milestone as the pioneers in achieving granular layer-level automatic partition of DNNs [10]. However, it failed to account for the limited resources of edge devices. Edgent [11], proposed adaptive partition of DNNs combined with joint early exits [12] to reduce inference latency. This approach maximizes inference accuracy through adaptive adjustments. Nevertheless, it fails to address the challenge of high-latency computation and data transmission. Recent works like [13], [14], [15] aim to integrate model compression with partition. Yang *et al.* [13] proposed an algorithm, which used identical channel pruning to compress the model and explored all partition points to determine the best strategy. The authors in [14] employed an attention mechanism to select critical model channels, thereby enhancing the efficiency of inference computation. Zhang *et al.* [15] introduced an approach to accelerate DNNs using acceleration techniques to create candidate partition layers. They then employed a fine-grained prediction model to estimate the execution latency of each DNN layer and determine the optimal partition. Nonetheless, these methods do not fully consider how to generate appropriate compression rates based on different partition points. This remains an open issue within the field.

In this paper, our motivation is to design an algorithm capable of automatically determining partition point and compression rates, while adaptively adjusting compression rates based on different partition points to minimize the inference latency of the entire DNN model. Our primary contributions are as follows:

- We investigate the joint DNN partition and compression in edge-cloud networks to achieve a trade-off between the latency and accuracy of collaborative DNN inference.
- We propose a dual-agent deep reinforcement learning (DRL) algorithm CPCDRL. The two agents collaborate to autonomously determine the location of partition point and the corresponding compression rates.

- We conduct extensive simulation experiments to evaluate the performance of the proposed algorithm. The results demonstrate that our proposed algorithm significantly reduces inference latency compared to baseline schemes by 10% to 67% while incurring minimal accuracy loss.

The remainder of this paper is organized as follows. Section II outlines the system model and frames the optimization problem. Our proposed algorithm is detailed in Section III, and an assessment of its performance is presented in Section IV. Finally, Section V provides the conclusion for this paper.

II. System Model and Problem Foumulation

As shown in Fig. 1, edge-cloud computing networks consist of edge devices e and cloud servers c. To fully exploit the heterogeneous resources available in edge-cloud computing networks, a pre-trained DNN model is deployed on both edge and cloud. The DNN model $\mathcal{N}$ in consideration consists of N prunable layers, represented as $\mathcal{N} = \{L_1, L_2, ..., L_N\}$. Each layer is amenable to channel pruning or neuron pruning, enabling the reduction of the model's size while preserving critical features. To accelerate the execution process of a DNN inference task, the DNN model is decoupled at an intermediate layer and separately deployed on edge device and cloud server. Given the constrained nature of edge resources, network compression is applied to the edge device to adapt to their computational capabilities.

A. DNN Partition

Typically, edge devices generate a large amount of raw data. Uploading this data directly to cloud server can result in substantial transmission latency. However, opting to upload intermediate data holds the promise of reducing inference latency. We assume $\mathbf{l} = \{l_0, l_1, l_2, ..., l_N\}$ to represent the partition strategy of $\mathcal{N}$, where $l_i \in \{0, 1\}$, and $\sum_{i=0}^{N} l_i = 1$. If $l_i = 1 (i \in (0, N))$, it signifies that edge device performs calculations for DNN layers from L_1 to L_i and then transmits the output of layer L_i (denoted as S_i) to cloud server. The cloud server begins computation from layer L_{i+1} and continues until the DNN's output layer, ultimately transmitting the inference results back to the edge device. Specifically, if $l_N = 1$, the entire task is computed locally, and if $l_0 = 1$, the entire task is offloaded for execution on cloud server.

B. Channel Pruning

Channel pruning is a technique used in DNNs to reduce computational complexity by removing less important channels, enhancing inference efficiency without compromising accuracy. We employ channel pruning during the deployment of model on the edge device, effectively reducing computational workload and enhancing inference speed. It evaluates channel importance in a DNN model and removes less important ones, resulting in a more compact and efficient model. Specifically, it can be formulated as:

$$\begin{aligned} &\arg\min_{\beta, W} \frac{1}{2N} \|Y - \sum_{i=1}^{c} \beta_i X_i W_i^T\|_F^2, \\ &\textbf{s.t. } \|\boldsymbol{\beta}\|_0 \leq c', \end{aligned} \tag{1}$$

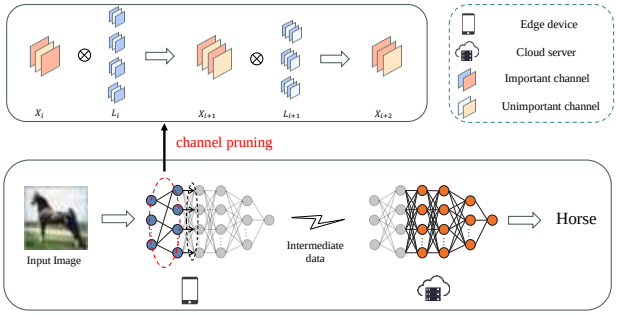


Fig. 1. Illustration of DNN partition and compression at mobile edge-cloud computing networks.

$\|\bullet\|_F$ is Frobenius norm. Y and X_i represent output feature maps and the ith input feature map channel for N samples. N, c, and c' represent the number of test samples, total available channels for pruning, and retained channels post-pruning, respectively. $\boldsymbol{\beta}$ is a c-dimensional vector of channel coefficients, where $\beta_i \in \{0, 1\}$ indicates the ith entry of $\boldsymbol{\beta}$. If $\beta_i = 0$, it signifies that the ith channel has a negligible impact on the output and can be pruned. W_i represents the ith entry of the layer weight W.

In our model, we employ RL to determine compression rates for each DNN layer, creating an overall compression strategy $\mathbf{r}$. We then calculate $\boldsymbol{\beta}$ based on channel importance. Selected channels are used to minimize reconstruction error through least squares:

$$\arg\min_{W'} \frac{1}{2N} \|Y - X'(W')^T\|_F^2, \tag{2}$$

where $X' = [\beta_1 X_1 \quad \beta_2 X_2 \quad ... \quad \beta_c X_c]$ represents the pruned input feature maps and $W' = [W_1^{'} \; W_2^{'} \; ... \; W_c^{'}]$ represents the corresponding convolution(Conv) kernel.

C. DNN Execution Latency Model

Before assessing DNN execution latency, it's crucial to determine the computational load of each layer. We use multiply-accumulate operations (MACs) as a metric, encompassing multiplication and accumulation operations in the DNN inference. We specifically calculate MACs for Conv and fully-connected(FC) layers, as the impact of other layers on runtime is minimal. With the reference to [16], the calculation for MACs in each type of layer is:

$$MACs_{Conv} = K \times K \times C_{in} \times C_{out} \times H_{out} \times W_{out}, \tag{3}$$

$$MACs_{FC} = C_{in} \times C_{out}, \tag{4}$$

here, K is the kernel size, H_{out} and W_{out} are the output feature's height and width. C_{in} and C_{out} denote the number of input and output channels.

After DNN partition and compression decisions are made, the inference execution time can be divided into three components, which are described as follows.

1) Execution Time on Edge: C_{ei} denotes the computation at the edge device when the model is partitioned at layer L_i. Denote the computing capacity of edge device as F_e. The local execution time is:

$$f^e(i) = \frac{C_{ei}}{F_e} = \frac{\sum_{k=1}^{i} MACs_k}{F_e}, \tag{5}$$

where $MACs_k$ represents the MACs of layer L_k. The term $\sum_{k=1}^{i} \text{MACs}_k$ signifies the cumulative computation of all layers from the initial layer to the partition layer L_i within the model deployed on the edge device.

2) Transmission Time: The edge device generates intermediate data of size S_i and uploads it to cloud server with a bandwidth of B. The required transmission time can be expressed as:

$$f^{e\to c}(i) = \frac{S_i}{B}. \tag{6}$$

3) Execution Time on Cloud: C_{ci} denotes the computation at the cloud server when the model is partitioned at layer L_i. Denote the computational capacity of the cloud server as F_c. The cloud execution time is:

$$f^c(i) = \frac{C_{ci}}{F_c} = \frac{\sum_{k=i+1}^{N} MACs_k}{F_c}, \tag{7}$$

the term $\sum_{k=i+1}^{N} MACs_k$ signifies the cumulative computation of all layers between the layer L_{i+1} and the final layer of the model deployed on the cloud server.

D. Problem Foumulation

In this paper, our goal is to minimize the inference latency by determining the location of the partition point and the compression rate of each layer of the model while maintaining accuracy. Hence, the problem can be formulated as:

$$\begin{aligned} &\min_{\mathbf{l},\mathbf{r}} T(\mathbf{l},\mathbf{r}), \\ &\textbf{s.t. } A(\mathbf{l},\mathbf{r}) \geq A_0, \end{aligned} \tag{8}$$

$A(\mathbf{l},\mathbf{r})$ is evaluated based on the given partition strategy $\mathbf{l}$ and compression strategy $\mathbf{r}$. It is required to achieve the desired accuracy A_0. This involves a dynamic decision problem since different partition points correspond to different compression rates, making solving this problem quite challenging.

III. Proposed Algorithm Design

To dynamically select partition points and their corresponding compression rates, we design two agents, A_p and A_c, specializing in partition and compression tasks. Fig. 2 illustrates our proposed algorithm, collaborative partition and compression using deep reinforcement learning(CPCDRL). In CPCDRL, A_c employs DDPG for model compression, while A_p uses DQN for DNN partition. The agents collaborate through the exchange of information to make decisions.

A. Markov Decision Process Formulation

RL aims to find the optimal policy for an agent by interacting with the environment. In RL, the problems are usually described as Markov Decision Processes(MDPs). At each time step, an RL agent selects action a_t based on its current state s_t and receives reward r_t as feedback. Through continuous exploration and learning, the agent gradually improves its policy to maximize cumulative rewards. Since there are two processes, we formulate each MDP separately.

1) DNN Partition Processing:

a) State Space: The state of the DNN model affects the selection of partition points within the DNN. As the model evolves with compression strategies, we thus regard the compression strategy as the state of agent A_p.

b) Action Space: To determine the location within the intermediate layers for model decoupling, we denote the action $a_p^t = n \in \{0, 1, 2, ..., N\}$.

2) DNN Compression Processing:

a) State Space: We consider each layer of the DNN as a state, which enables us to represent the t-th layer as:

$$s_c^t = (t, i, n, c, h, w, s, k, a_{t-1}, p, \text{MACs}), \tag{9}$$

where t denotes the layer index, and i signifies the layer type (FC or Conv). The variables n, c, h, w, s, and k respectively denote the number of output and input channels, the dimensions of the input data in terms of height and width, the stride value, and the dimensions of the Conv kernel. Furthermore, a_{t-1} stands for the previous layer's compression rate, p indicates the partition point's position and MACs denote the number of multiply-accumulate operations at layer L_t.

b) Action Space: The action is represented by a continuous variable $a_c^t \in (0, 1)$. At layer L_t, the agent selects a_c^t as compression rate for the layer based on the current state.

3) Reward: To enhance the efficiency of the DNN inference without significantly compromising accuracy, we design the reward function as follows:

$$R = e^{\eta(A-A_0)} \times (1-\alpha) - \mu T, \tag{10}$$

here, A denotes the model's accuracy, η and μ are constants, A_0 is a threshold indicative of the desired accuracy level, α is the compression ratio, and T denotes the total latency required for DNN inference. By formulating the reward function in this manner, we encourage the agent to adopt strategies that result in high accuracy and reduced latency. When the accuracy falls below the predefined threshold (A_0), the agent incurs a significant penalty. The parameter η and μ is instrumental in regulating the influence of accuracy and latency on the overall reward, and their values are fine-tuned based on empirical experimentation.

B. DQN-Based Partition

We employ the DQN algorithm to train agent A_p for the network partition task, and the detailed network partition algorithm is outlined in Algorithm 1. At the outset, agent A_p initializes the DQN with random parameters. During each time

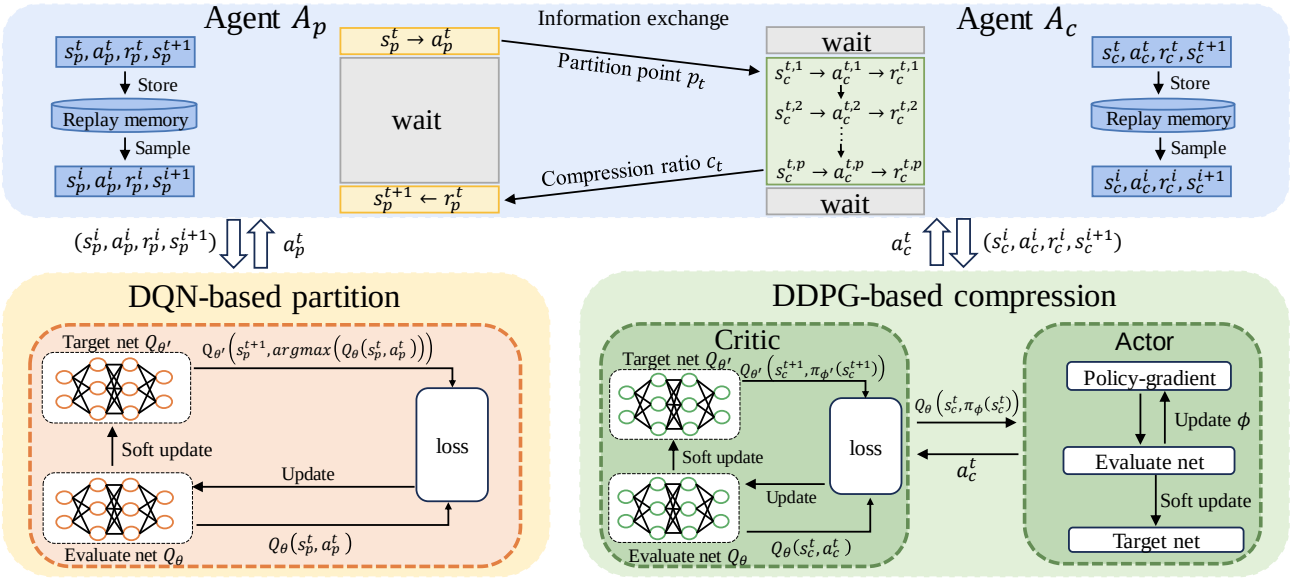


Fig. 2. Framework of the proposed CPCDRL algorithm.

Algorithm 1 DQN-Based Partition

Input: The pre-trained DNN model D, maximum training episodes E, the partitioning agent A_p,
Output: The optimal partition strategy

Initialize network Q_θ with random parameters θ
for $episode = 1$ to E **do**
  Select a random action at with probability ϵ otherwise select $a_p^t = max_a Q_\theta(s_p^t, a_p^t)$
  Send a_p^t to A_c
  Receive compression ratios from A_c
  Calculate r_p^t and observe s_p^{t+1}
  Update Q_θ by (11)
end for

slot, actions are chosen according to the policy Q_θ using an ϵ-greedy strategy. Then A_p sends a_p^t to A_c and receives the compression ratios from A_c. we adopt DQN to train Q_θ as follows:

$$\nabla_\theta J(\theta) = \mathbb{E}\Big[\big(r_p^i + \gamma \max_{a'} Q_\theta(s_p^i, a_p^i) - Q_\theta(s_p^i, a_p^i)\big)\nabla_\theta Q_\theta(s_p^i, a_p^i)\Big]. \tag{11}$$

C. DDPG-Based Compression

The action space of agent A_c is continuous, and thus, we employ the DDPG algorithm to train it. The proposed DNN compression algorithm is illustrated in Algorithm 2. Initially, the compression agent A_c initializes the actor and critic networks with random parameters. In each time slot t, agent A_c generates a compression ratio using the truncated normal distribution:

$$a_c^t \sim TN(\pi_\phi(s_c^t), \sigma^2, 0, 1), \tag{12}$$

Algorithm 2 DDPG-Based Compression

Input: The pre-trained DNN model D, maximum training episodes E, the compression agent A_c
Output: The optimal compression strategy

Initialize critic network Q_θ, actor network π_ϕ
Receive partition point p from A_p
for $episode = 1$ to E **do**
  for $t = 1$ to p **do**
    Obtain state s_c^t based on the t-th layer of D;
    Generate compression action a_c^t, $a_c^t \sim TN(\pi_\phi(s_c^t), \sigma^2, 0, 1)$
    Compress layer L_t
  end for
  Calculate r_c^t according to Equation (10)
  Send compression ratios to A_p
  Update Q_θ, π_ϕ by (13)
  Receive the new partition point p from A_p
end for

when the compression is not yet complete, the reward r_c^t is set to 0 at intermediate time slots. Once compression is finished, the reward for the entire episode is computed based on Equation (10), and the compression information is transmitted to agent A_p. The critic and actor networks are trained by minimizing the loss function

$$Loss(\theta) = \mathbb{E}\Big[(y_c^i - Q_\theta(s_c^i, a_c^i))^2\Big], \tag{13}$$

where y_c^i is the target value computed by the target network $Q_{\theta'}$, defined as

$$y_c^i = r_c^i + \gamma Q_{\theta'}\left(s_c^{i+1}, \pi_{\phi'}(s_c^{i+1})\right). \tag{14}$$

Then A_c receives a new partition point p from A_p, after training A_c E rounds, we get the optimal policy for model compression.

D. CPCDRL Algorithm

In CPCDRL, agent A_c and agent A_p work together on the complex tasks of model partition and compression. As shown in Fig. 2, agent A_p observes the current state s_p^t and selects a partition point a_p^t. This action is then communicated to agent A_c, which handles model compression up to the defined partition point. Following this, the model's accuracy and execution latency are calculated based on the partition point and compression rates, resulting in a reward that is shared between the two agents. Agent A_p then transitions to the next state s_p^{t+1} after receiving the compression rate from agent A_c. Under close cooperation, two agents collectively engage in the decision-making process. Agent A_p possesses the capability to autonomously select appropriate partition point based on the current model state, while agent A_c can generate corresponding compression rates for different partition points. By updating their respective networks through shared rewards, we can obtain an optimal strategy that minimizes inference latency while ensuring a certain level of accuracy.

IV. Evaluation Results

A. Setup and Baselines

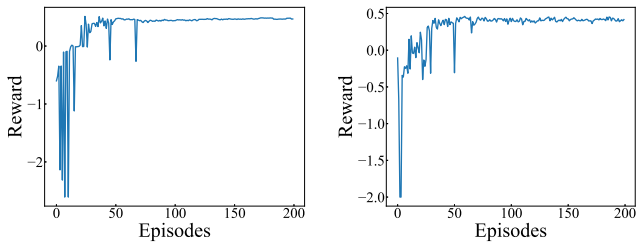
We evaluate CPCDRL using CIFAR-10, focusing on image classification. Two DNN architectures, MobileNet and VGG16, are employed, offering distinct complexities. The CIFAR-10 dataset contains 60,000 images divided into 10 categories, with 50,000 for training and 10,000 for testing. To simulate an edge cloud system realistically, we configure parameters: edge device has computational power (F_e) of 500 MFLOPS, cloud server has computational power (F_c) at 5 GFLOPS and network bandwidth (B) at 200 KBps.

In this paper, we conduct a comparative analysis between our proposed CPCDRL algorithm and three baselines.

- **Edge(Pruned) Deployment** [17]. Deploying all DNN models and compressing them solely on edge devices without the use of cloud resources.
- **Neurosurgeon** [10]. Automating DNN partition across edge devices and central data centers to optimize performance through layer-level considerations.
- **Greedy Partition Strategy**. Select the partition point that minimizes the total latency for a given compression ratio.

B. DRL Training

We conducted a comprehensive analysis of the convergence performance of the MobileNet and VGG16 models, as depicted in Fig. 3(a) and Fig. 3(b). They exhibit remarkably similar trends during the convergence process, reflecting the efficacy of CPCDRL. As the training episodes progress, a distinct pattern in rewards becomes evident. Initially, there is a rapid and substantial increase in reward, indicative of the agents' rapid learning and adaptation to the task. However,



(a) Reward vs. Episodes of MobileNet (b) Reward vs. Episodes of VGG16

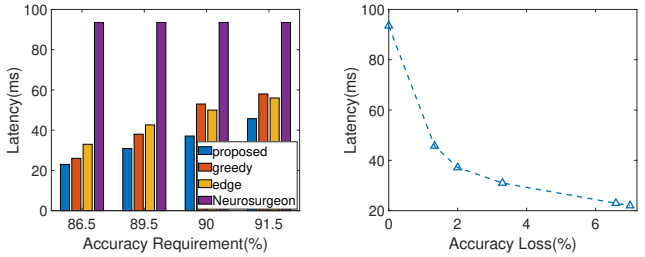
Fig. 3. Training Performance of Different Models.

during the early stages of training, both curves display considerable fluctuations, which are consistent with the agents' exploration of various strategies to optimize their performance. Around the 75th training episode, a noticeable stabilization occurs in both reward curves. This stabilization signifies that the agents have effectively mastered strategies that enable them to make accurate decisions across a spectrum of diverse states.

C. Latency and Accuracy Performance

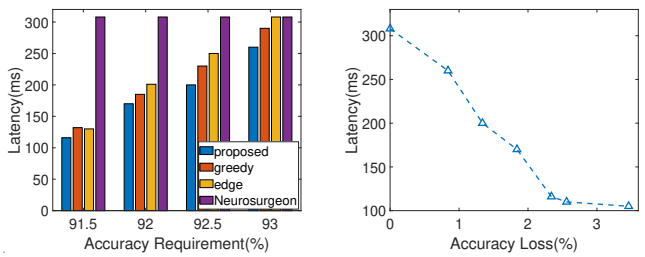
Fig. 4 compares the inference performance of CPCDRL with baselines on MobileNet. Fig. 4(a) demonstrates the inference latency required for the MobileNet at different accuracy requirements. Specifically, when the accuracy requirement is set at 91.5%, CPCDRL exhibits an inference latency 48% lower than that of *Neurosurgeon*. However, when the accuracy requirement is reduced to 89.5%, the latency of CPCDRL is only 33% of the *Neurosurgeon*'s latency. Furthermore, the latency of CPCDRL is significantly lower than the other two methods, indicating that CPCDRL can strike a balance between accuracy and latency. It can effectively utilize resources, and expedite task completion without sacrificing substantial accuracy. In Fig. 4(b), we observe the variation in latency under different accuracy losses(compared with the original model). The inference latency sharply decreases until an accuracy loss of 1% is reached, after which it gradually increases. This phenomenon underscores that CPCDRL can substantially reduce inference latency with minimal accuracy loss.

Fig. 5 shows the inference performance on VGG16. Fig. 5(a) showcases the inference latency for our proposed CPCDRL and baselines under varying accuracy requirements for VGG16. When the accuracy requirement is 91.5%, CPCDRL achieves a latency of only 116 ms, which is 37% of *Neurosurgeon*'s latency, while the latency for the greedy algorithm and the edge-only compression model is 132 ms and 130 ms, respectively. With increasing accuracy requirements, the latency growth rate of CPCDRL remains lower than that of the greedy algorithm and the edge-only compression model. This implies that our proposed CPCDRL can effectively reduce latency and enhance inference speed even when high accuracy requirements are imposed. Fig. 5(b) depicts the variation in latency as accuracy loss increases. Similar to MobileNet, VGG16 experiences a rapid decline in latency with minor losses in accuracy. Beyond an accuracy loss of 3%, there is



(a) Latency under different accuracy requirements (b) Latency vs Accuracy loss

Fig. 4. Inference performance of MobileNet.



(a) Latency under different accuracy requirements (b) Latency vs Accuracy loss

Fig. 5. Inference performance of VGG16.

no significant change in latency. This suggests that further compression of the model at this stage results in substantial accuracy loss without effectively reducing latency.

V. Conclusion

In this paper, we have designed an efficient collaborative DNN inference framework that joint consideration of model partition and compression to reduce inference latency at mobile edge-cloud computing networks. Specifically, we have decoupled the model at intermediate layers and deployed them separately on the edge device and the cloud server. Furthermore, we have introduced model compression for the models deployed on the edge device to accommodate the inherent resource constraints at the edge. We have further proposed CPCDRL for intelligent determination of the partition point and corresponding compression rates. Simulation results have demonstrated that the proposed algorithm can significantly reduce inference latency with minimal precision loss.

Acknowledgment

This work is supported in part by National Key R & D Program of China (Grant No. 2022YFE0125400), National NSFC (Grants No. 62372072, 62102053, 62072060, 92067206, 61972222), Chongqing Research Program of Basic Research and Frontier Technology (Grant No. cstc2022ycjhbgzxm0058), Key Research Program of Chongqing Science & Technology Commission (Grant No. cstc2021jscx-dxwtBX0019), Haihe Lab of ITAI (Grant No. 22HHXCJC00002), the Natural Science Foundation of Chongqing, China (Grant No. CSTB2022NSCQ-MSX1104), the General Program of Chongqing Science & Technology Commission (Grant No. CSTB2023TIAD-STX0035, CSTB2022TIAD-GPX0017, CSTB2022TIAD-STX0006), Regional Innovation Cooperation Project of Sichuan Province (Grants No. 2023YFQ0028, 24QYCX0019), Regional Science and Technology Innovation Cooperation Project of Chengdu City (Grant No. 2023-YF11-00023-HZ), and Guangdong Pearl River Talent Recruitment Program (Grants No. 2019ZT08X603, 2019JC01X235).

References

[1] Y. Li, Y. Shi, K. Wang, B. Xi, J. Li, and P. Gamba, "Target detection with unconstrained linear mixture model and hierarchical denoising autoencoder in hyperspectral imagery," *IEEE Trans. Image Process.*, vol. 31, pp. 1418–1432, Jan. 2022.

[2] M. Xu, C. Song, H. Wu, S. S. Gill, K. Ye, and C. Xu, "esDNN: Deep neural network based multivariate workload prediction in cloud computing environments," *ACM Trans. Internet Technol.*, vol. 22, no. 3, pp. 1–24, Aug. 2022.

[3] E. Bout, V. Loscri, and A. Gallais, "Evolution of IoT security: the era of smart attacks," *IEEE Internet Things Mag.*, vol. 5, no. 1, pp. 108–113, Mar. 2022.

[4] M. Khabbaz, K. Shaban, and C. Assi, "Delay-aware flow scheduling in low latency enterprise datacenter networks: Modeling and performance analysis," *IEEE Trans. Commun.*, vol. 65, no. 5, pp. 2078–2090, Feb. 2017.

[5] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," *Proc. IEEE*, vol. 107, no. 8, pp. 1738–1762, Jun. 2019.

[6] F. Hu, Y. Deng, and A. H. Aghvami, "Non-episodic and heterogeneous environment in distributed multi-agent reinforcement learning," in *Proc. IEEE GLOBECOM*, Dec. 2022, pp. 1019–1024.

[7] A. M. Elbir, S. Coleri, A. K. Papazafeiropoulos, P. Kourtessis, and S. Chatzinotas, "A hybrid architecture for federated and centralized learning," *IEEE Trans. Cognit. Commun. Networking*, vol. 8, no. 3, pp. 1529–1542, Sep. 2022.

[8] Z. Gao, D. Miao, L. Zhao, Z. Mo, G. Qi, and L. Yan, "Triple-partition network: collaborative neural network based on the 'end device-edge-cloud'," in *Proc. IEEE WCNC*, May 2021, pp. 1–7.

[9] X. Chen, J. Zhang, B. Lin, Z. Chen, K. Wolter, and G. Min, "Energy-efficient offloading for DNN-based smart IoT systems in cloud-edge environments," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 3, pp. 683–697, Jul. 2021.

[10] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," *ACM SIGARCH Comput. Archit. News*, vol. 45, no. 1, pp. 615–629, Mar. 2017.

[11] E. Li, Z. Zhou, and X. Chen, "Edge intelligence: On-demand deep learning model co-inference with device-edge synergy," in *Proc. Workshop Mobile Edge Commun.*, Aug. 2018, pp. 31–36.

[12] S. Teerapittayanon, B. McDanel, and H.-T. Kung, "Branchynet: Fast inference via early exiting from deep neural networks," in *Proc. IEEE ICPR*, Apr. 2016, pp. 2464–2469.

[13] S. Yang, Z. Zhang, C. Zhao, X. Song, S. Guo, and H. Li, "CNNPC: End-edge-cloud collaborative CNN inference with joint model partition and compression," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 12, pp. 4039–4056, Dec. 2022.

[14] Y. Ding, W. Fang, M. Liu, M. Wang, Y. Cheng, and N. Xiong, "JMDC: A joint model and data compression system for deep neural networks collaborative computing in edge-cloud networks," *J. Parallel Distrib. Comput.*, vol. 173, pp. 83–93, Mar. 2023.

[15] W. Zhang, N. Wang, L. Li, and T. Wei, "Joint compressing and partitioning of CNNs for fast edge-cloud collaborative intelligence for IoT," *J. Syst. Archit.*, vol. 125, p. 102461, Apr. 2022.

[16] L. Wang, L. Xiang, J. Xu, J. Chen, X. Zhao, D. Yao, X. Wang, and B. Li, "Context-aware deep model compression for edge cloud computing," in *Proc. IEEE ICDCS*, Feb. 2021, pp. 787–797.

[17] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han, "Amc: Automl for model compression and acceleration on mobile devices," in *Proc. IEEE ECCV*, Sep. 2018, pp. 784–800.