Hindawi
Security and Communication Networks
Volume 2020, Article ID 8831341, 10 pages
https://doi.org/10.1155/2020/8831341

Research Article

Collaborative Intelligence: Accelerating Deep Neural Network Inference via Device-Edge Synergy

Nanliang Shan, Zecong Ye, and Xiaolong Cui

College of Information Engineering, Engineering University of PAP, Xi'an 710086, China

Correspondence should be addressed to Xiaolong Cui; 18182437082@163.com

Received 31 May 2020; Revised 11 July 2020; Accepted 31 July 2020; Published 7 September 2020

Academic Editor: Xiaolong Xu

With the development of mobile edge computing (MEC), more and more intelligent services and applications based on deep neural networks are deployed on mobile devices to meet the diverse and personalized needs of users. Unfortunately, deploying and inferencing deep learning models on resource-constrained devices are challenging. The traditional cloud-based method usually runs the deep learning model on the cloud server. Since a large amount of input data needs to be transmitted to the server through WAN, it will cause a large service latency. This is unacceptable for most current latency-sensitive and computation-intensive applications. In this paper, we propose Cogent, an execution framework that accelerates deep neural network inference through device-edge synergy. In the Cogent framework, it is divided into two operation stages, including the automatic pruning and partition stage and the containerized deployment stage. Cogent uses reinforcement learning (RL) to automatically predict pruning and partition strategies based on feedback from the hardware configuration and system conditions so that the pruned and partitioned model can better adapt to the system environment and user hardware configuration. Then through containerized deployment to the device and the edge server to accelerate model inference, experiments show that the learning-based hardware-aware automatic pruning and partition scheme can significantly reduce the service latency, and it accelerates the overall model inference process while maintaining accuracy. Using this method can accelerate up to 8.89× without loss of accuracy of more than 7%.

1. Introduction

As the backbone technology to support modern intelligent services and applications, deep neural network (DNN) has become more and more popular due to their superior performance in computer vision [1], speech recognition [2], natural language processing [3], and big data analysis [4]. With the development of mobile edge computing (MEC), more and more intelligent services and applications based on DNN are deployed on mobile terminal devices to meet the diverse and personalized needs of users. Unfortunately, today's mobile devices cannot support these DNN-based intelligent services and applications well because these intelligent models usually require a lot of computing resources.

To solve a large number of service resource requirements of the DNN model, the traditional method relies on powerful cloud servers to provide rich computing power. In this case, the training and inference data generated on the mobile device are transmitted to the remote cloud server, and the inference result is returned after the execution is completed. However, many intelligent applications require frequent information interaction, which makes large amounts of data to be transmitted between mobile devices and remote cloud server through a long wide area network. This will bring intolerable communication latency, making cloud-based computing impractical. To solve this problem, we explored the emerging computing architecture of device-edge synergy. By adding an edge service node composed of an edge server and an edge microbase station near the terminal device, the cloud computing capability is sunk from the core network to the edge network closer to the user. This new computing architecture provides sufficient computing power support for DNN-based computing-intensive and latency-sensitive applications at the edge network. So how to

effectively deploy the intelligent model based on DNN to the edge and make full use of the rich computing resources of the edge server to perform model inference (i.e., edge intelligence) [5] to minimize the service latency will be the main consideration in this paper.

In response to the above problems, the predecessors have made many efforts. These include collaborative computing between terminal devices and cloud servers [6–8], model compression and parameter pruning [9–12], or customized mobile implementation [13–15]. Despite all these efforts made by the predecessors, on the premise of ensuring the accuracy of the model required by the user, the service latency is minimized, and the user's hardware configuration and system status can be sensed to implement automatic model pruning and partition. The current edge intelligence architecture still has major defects.

On this issue, this paper proposes a device-edge synergy framework Cogent, which uses reinforcement learning (RL) to achieve automatic pruning and partition of models. And through the container technology, the divided model blocks are packaged and deployed on the edge server and the terminal device, and the rich computing resources of the edge server are used to accelerate the model inference collaboratively. The Cogent framework is a latency-sensitive collaborative intelligent design. It is mainly divided into two operation stages, namely, automatic pruning and partition stage and containerized deployment stage. In the automated pruning and partition phase, Cogent uses RL to observe hardware accelerators and system status (including network bandwidth and edge server load) and provides model pruning and partition strategies. We observe that the accuracy of the compression model is very sensitive to the sparsity of each layer and requires a fine-grained action space. Therefore, instead of searching in discrete space, we propose a continuous compression ratio control strategy with DDPG [16] agent, which learns through trial and error and penalizes loss of accuracy, while encouraging model acceleration and reduction. Specifically, our DDPG agent handles the network model in an integrated and layered approach. For the overall network model, the agent receives network structure information of the entire model, system network bandwidth B, hardware accelerator information A, and edge server load information E, and then it outputs the model partition point. For each layer, the agent receives the state information S_t and hardware accelerator information A, and then it outputs the precise pruning ratio of each layer. Our Cogent framework automates this process through learning-based strategies rather than relying on rule-based strategies and experienced engineers. In the containerized deployment phase, we use Docker and Kubernetes to dynamically package model blocks and assign containers to one or more available devices to complete a DNN task, which greatly increases the flexibility and reliability of Cogent. Cogent makes full use of device-edge synergy to achieve collaborative intelligence, which can minimize inference latency while meeting user accuracy requirements.

To summarize, we present the contribution of this paper as follows:

(i) We propose Cogent, an execution framework that accelerates deep neural network inference through device-edge synergy. We use Cogent automated pruning and partition to jointly optimize DNN model inference to minimize service latency while ensuring user accuracy requirements.

(ii) We propose an automated DNN model pruning and partition algorithm, which uses reinforcement learning to determine pruning and partition strategies automatically. At the same time, we receive the feedback of the hardware accelerator and system state in the design cycle, so that the pruned and partitioned models can better adapt to different hardware architectures and system conditions, greatly reducing service latency.

(iii) We use Docker and Kubernetes to dynamically package model blocks and assign containers to the edge server and terminal devices to complete a DNN task cooperatively. It not only makes full use of the rich computing resources of the edge server to accelerate the inference process but also greatly increases the flexibility and reliability of Cogent.

The rest of this paper is organized as follows. First, we review the related work in Section 2. The proposed overall framework of Cogent is introduced in Section 3. The results of the performance evaluation are shown in Section 4 to demonstrate the effectiveness of Cogent. Finally, the paper is concluded in Section 5.

2. Related Work

The rapid development of DNN makes it quickly become one of the most important components of artificial intelligence technology today. DNN consists of a series of network layers, and each layer of network consists of a group of neurons. DNN is widely used in the field of computer vision and natural language processing, including image classification, target detection, video recognition, and text processing. At present, edge intelligence technology has attracted the attention of researchers. To implement artificial intelligence at the edge, edge intelligence technology deploys DNN models on mobile devices that are closer to users to enable more flexible and safe interaction between users and smart models. However, due to the resource limitations of terminal devices, it has become very challenging to directly deploy and infer computation-intensive DNN models on edge devices. On this issue, existing efforts are devoted to optimizing DNN calculations on edge devices. There are three main areas worthy of attention here.

2.1. Optimize DNN Model. DNN model optimization is used to include model structure optimization and hardware acceleration. In terms of model structure optimization, some researchers tried to develop new DNN structures to achieve the desired accuracy under moderate calculations, such as DNN models that were much smaller than normal network models without sacrificing excessive accuracy [13]. Also, in

order to reduce the amount of data transmission during DNN inference, the DNN model was compressed by model pruning [17–19]. Others were focused on reducing the redundancy in the original model by the model compression techniques [20–22] to obtain an effective model. Recent advances in this optimization had turned to network architecture search (NAS) [23–25]. In terms of hardware acceleration, mobile devices could be embedded with deep learning inference chips to improve latency and energy efficiency with the help of architectural acceleration technology [26, 27]. Other works were aimed at optimizing the use of existing resources [28–30] and improving service performance [31–34].

2.2. Device-Edge Synergy. The most involved in the mobile device and edge server collaboration were model partition and related technologies. DNN model partition technology referred to partitioning a specific DNN model into some continuous parts and deploying these parts on multiple participating devices. The goal of model partition technology was similar to computation offloading and aimed to maximize the use of external computing resources to accelerate mobile edge computing. For example, some frameworks used DNN partition to optimize computation offloading between mobile devices and the cloud, while other frameworks aimed to distribute computing workload among mobile devices [35–37]. The most critical technique of model partition lied in the choice of partition point. In the work [38], the DNN partitioning problem was transformed into the shortest path problem, and the approximate solution was used to solve the problem. At the same time, they also used PNG encoding to reduce the amount of intermediate data transmission. To study the influence of network conditions and server load during the partition process, Kang [7] studied the hierarchical partition of the DNN model, and the variation of latency with server load changed under three typical wireless communication conditions. Besides, an improved DNN structure had been proposed for device-edge synergy [8, 39], where early exit network branches were added to the original network. Their evaluation proved the effectiveness of the improved DNN structure in low-latency inference and accuracy assurance.

2.3. Automatic Machine Learning (AutoML). Besides, many research efforts aimed to improve the performance of DNNs through an automated search of network structures: NAS [40] aimed to search for transferable network blocks whose performance exceeds many manually designed architectures. Progressive NAS [24] adopted sequential model-based optimization methods to accelerate architecture search by 5×. Pham et al. [25] introduced efficient NAS used parameter sharing to accelerate the speed of architecture exploration by 1000×. Cai et al. [41] introduced path-level network transformation to efficiently search the tree structure space. Driven by these AutoML frameworks, He et al. [42] leveraged reinforcement learning to automatically prune the convolution channel.

Compared with the current work, the Cogent framework designed in this paper makes a good combination of DNN model optimization, device-edge synergy, and AutoML. Cogent leverages reinforcement learning to automatically predict the pruning ratio of each layer and the partition point of the model. At the same time, it takes into account the hardware architecture and system state. Finally, through containerized deployment, the flexibility and reliability of the Cogent framework are greatly improved. Cogent can speed up model inference as much as possible while ensuring user accuracy requirements and at the same time has better adaptability to different hardware devices and system status. This provides a good choice for latency-sensitive service requests on mobile devices.

3. Overall Framework Design

3.1. Framework Overview. As shown in Figure 1, we proposed the design of the Cogent framework, which includes two operational stages, namely, automated pruning and partition stage and containerized deployment stage. First of all, the Cogent framework uses reinforcement learning to automatically search the huge pruning design space in the loop. Its RL agent integrates hardware accelerators and system status (including network bandwidth and edge server load) into the detection loop so that it can obtain direct feedback from the hardware and system status. Then, the agent proposes an optimal model partitioning and pruning strategy under the given amount of computing resources and network bandwidth. The automatic model pruning and partition algorithm on Cogent will perform model pruning and partition according to these strategies. Finally, the divided model blocks are packaged and delivered. The Cogent framework automates the pruning process and model partition process by using learning-based methods that take hardware- and system-state-specific metrics as direct rewards to meet the requirements of service request accuracy while minimizing service latency. We use the actor-critic model with the deep deterministic policy gradient (DDPG) agent to give actions: the pruning ratio of each layer and the partition point of the model. We collect hardware counters as constraints and use latency as a reward to search for optimal pruning and partition strategies. We have two hardware environments, including terminal device accelerators and edge server accelerators. The following describes the details of each element of reinforcement learning.

3.2. State Space. Our agent deals with neural networks in a combination of whole and layer. For the overall model, the agent needs to determine the most appropriate partition point. For each layer, the agent needs to determine the proportion of pruning for each layer. In this paper, we introduced an 11-dimensional feature vector as our state value S_t:

$$\left\{B, E, A_{m,c}, L_t\left(t, n, c, \text{FLOPs, reduced}, \text{rest}, a_{t-1}, time\right)\right\}, \tag{1}$$

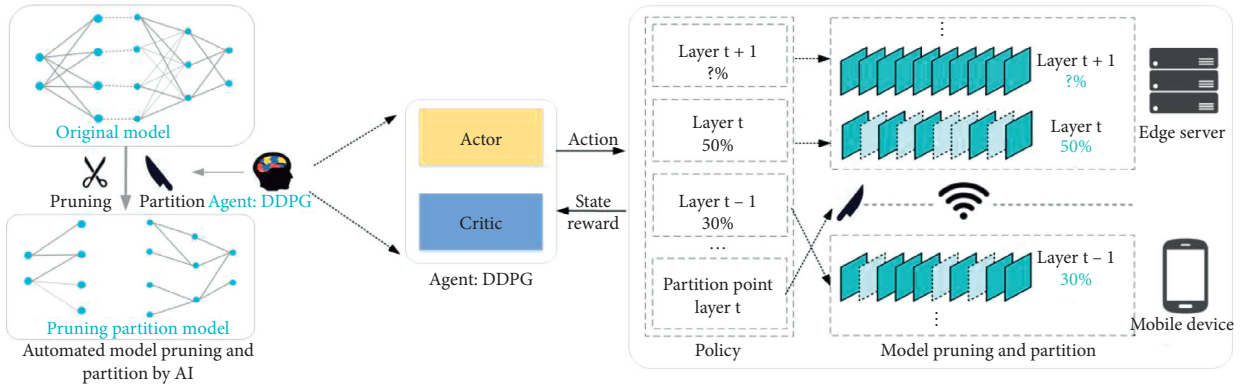

Figure 1: Overview of Cogent framework for model pruning and partition.

where B is the network bandwidth, E is the edge server load, and $A_{m,c}$ is the hardware accelerator configuration, which usually refers to the CPU speed of the mobile terminal device and the CPU speed of the edge server. t is the layer index, n is the dimension of the core, c is the input of this layer, FLOPs is the FLOPs calculation of the layer, reduced is the total number of calculations reduced in previous layers, rest is the number of calculations remaining in the subsequent layer, a_{t-1} is the pruning ratio selected by the upper layer, and *time* is the inference time spent on this layer. Before being passed to the agent, they are scaled within [0, 1]. These features are essential for the agent to distinguish one network layer from another.

3.3. Action Space. For the partition point p, we use the discrete space as the action space, because the choice of p is fixed and limited. For the pruning ratio of each layer, most of the existing works use discrete space as a coarse-grained action space. For high-accuracy model architecture search, coarse-grained action space may not be a problem. However, we observe that model compression is very sensitive to the sparse ratio, which leads to a surge in the number of discrete actions, so we need a more fine-grained action space. Otherwise, such a large action space will be difficult to effectively explore [16]. At the same time, discretization will also make the action selection jumpy and may miss the optimal sparse ratio. Therefore, we suggest using the continuous action space $a \in (0, 1]$, which can achieve more fine-grained and more accurate model pruning.

3.4. DDPG Agent. As shown in Figure 1, we first select the partition point p before the agent starts to determine the pruning rate for each layer. The choice of the partition point is mainly affected by the network architecture. Of course, the agent will also adjust the decision based on the hardware accelerator and system status. After determining the partition point, there is no need to partition immediately, but first determine the pruning ratio of each layer. The agent receives an embedding state S_t of layer L_t from the environment and then outputs a sparse rate as action a_t. The agent then moves to the next layer L_{t+1} and receives state S_{t+1}. After the final layer of the decision is made, the accuracy of the model is evaluated on the validation set and returned to the agent. On the premise of ensuring the accuracy requirements of users, according to the decision set, the specified compression algorithm (e.g., channel pruning) is used to compress the model. To improve the speed of exploration, we only evaluate the accuracy without fine-tuning, which is a good method to approximate the precision of fine-tuning. At this time, we will get the optimal pruning ratio of each layer for the hardware characteristics and system status when the current partition point is p.

For agent decision a_t, we use DDPG to continuously control the compression ratio. For the noise distribution during exploration, we use a truncated normal distribution. The noise σ is initialized to 0.5 and decays exponentially after each episode:

$$\mu'(s_t) \sim TN\left(\mu\left(s_t \mid \theta_t^{\mu}\right), \sigma^2, 0, 1\right). \tag{2}$$

The design of the DDPG agent follows Block-QNN [43], applying a variant of the Bellman equation [44]. Each state input of each episode is (S_t, a_t, R, S_{t+1}), where R is the reward after pruning the network. In the update process, to reduce the gradient loss, the gradient estimate needs to subtract the baseline reward b, which is equivalent to the exponential moving average of the previous reward [40]:

$$\begin{aligned} \text{Loss} &= \frac{1}{N}\sum_i \left(y_i - Q\left(s_i, a_i \mid \theta^Q\right)\right)^2, \\ y_i &= r_i - b + \gamma Q\left(s_{i+1}, \mu\left(s_{i+1}\right) \mid \theta^Q\right). \end{aligned} \tag{3}$$

The discount factor γ is set to 1 to avoid overprioritizing short-term rewards.

3.5. Reward Function. By adjusting the reward function, we can accurately find the limit of compression and minimize service latency. Using the reward function to limit the action space (sparse rate of each layer), we can accurately obtain the target accuracy. Take fine-grained pruning to reduce inference latency as an example: we allow arbitrary operations in the first few layers. When we find that we are close to the target accuracy, we begin to limit action a, pruning all the following layers with the most conservative strategy. Our reward function is as follows:

$$R = \lambda \times \Delta\text{latency}, \tag{4}$$

where Δlatency is the latency difference before and after pruning and λ is a scaling factor, which is set to 0.1 in our experiment.

3.6. Automated Model Pruning and Partition. Our goal is to find the optimal partition point of the model first and then find the redundancy of each layer according to the hardware environment that the preparatition part will configure in the future. The optimal partition point of a DNN model depends on the topology of the DNN, which is reflected in the change of the amount of calculation and data of each layer. Besides, even for the same DNN structure, dynamic factors such as wireless network bandwidth and edge server load will affect the choice of optimal partition point. For example, the instability of the wireless network bandwidth will directly affect the transmission latency between the mobile device and the edge server, and the load change of the edge server will directly affect the queuing latency or calculation latency of the application request at the edge server. We train an RL agent to predict the partition point and pruning ratio and then perform pruning and partition. We quickly assess the accuracy after pruning before fine-tuning, as an effective representation of the final accuracy. Then, we update the agent by rewarding faster, smaller, and more accurate models. We introduced the AutoML process of the Cogent framework in Algorithm 1.

Cogent first analyzes the constituent layers of the target DNN model and extracts the type and configuration (L_t) of each layer and then uses the RL agent to predict the latency T_m and T_c of the current layer on mobile devices and the edge server, respectively. At the same time, the current network bandwidth B and edge server load E should be considered. Line 10 of Algorithm 1 uses the agent to predict the output parameter amount D_t of the execution layer L_t on the mobile device. Line 13 of Algorithm 1 calculates the transmission latency T_t under the current wireless network bandwidth. Line 15 of Algorithm 1 evaluates the inference latency and inference accuracy of each candidate partition point and selects the partition point with the lowest inference latency under the premise of meeting the user's accuracy requirements. After determining the partition point, there is no need to partition immediately, because this will affect the subsequent pruning process. The pruning ratio needs to be comprehensively decided according to the hardware environment and system status of the model block to be deployed in the future. Also, the agent must ensure the accuracy required by the user when making decisions, which is a prerequisite for Cogent to perform pruning and partition. Finally, the Cogent framework will automatically perform model pruning and partition according to the agent's decision.

3.7. Containerized Deployment. We containerized each model block after partitioning and deployed the model by launching pods on the edge server and mobile devices through Kubernetes. Please note that the general model is configured to work together on one mobile device and one edge server. In this case, only two pods will be used for deployment. If multiple mobile devices request an application at the same time, multiple pods can be deployed. If the system status changes, Cogent will periodically recalculate the optimal partition point. Once the model execution graph changes, we will adjust the pod configuration and reschedule them. Besides, application service requests may fail in mobile edge networks. To quickly restore services without affecting the normal operation of the pods, Kubernetes assigned a static virtual IP to each pod. Each pod communicates with its upstream and downstream pods via virtual IP. The association between the virtual IP and the pod is based on the position of the pod in the model execution graph. If the service request fails, we can easily launch a new pod from the edge server and associate the new pod with the virtual IP. In this way, the application services can be kept in the normal operation of the mobile device without being affected. By containerizing each model block, we use Docker and Kubernetes to simplify model update and deployment and efficiently handle runtime resource management and scheduling of containers.

4. Evaluation

4.1. Experimental Setup. We use Xilinx Zynq-7020 FPGA [45] as our terminal device and Xilinx VU9P [46] as our edge server and prove the feasibility and efficiency of Cogent through design experiments. Table 1 shows our experimental configuration on both platforms and the resources available to them. Configure the inbound and outbound network bandwidth of each terminal device to connect to the edge server through the traffic control (TC) infrastructure. Besides, all physical servers run Ubuntu 18.04 and deploy the Kubernetes (Release-1.7) cluster. VGG19 [47] is a state-of-the-art image classification DNN, which serves as the target network for device-edge synergy inference in this paper. Our dataset is CIFAR-10 [48], a widely used image classification dataset with 10 classes of objects. Our network of actors μ has two hidden layers, each with 300 units. The final output layer is a Sigmoid layer that binds the action within (0, 1). Our critic network Q also has two hidden layers, each with 300 units. We set the learning rate to 0.01, the batch size to 64, and the replay memory capacity to 2000. Our agent first explores 100 episodes with constant noise $\sigma = 0.5$ and then explores 300 episodes with exponential attenuation noise σ, with an attenuation coefficient of 0.99. We set the cloud-based computation offloading method [49] as the baseline and added the status quo method HierTrain [50] as a comparison. We compared the performance of the Cogent architecture with the baseline and status quo in terms of inference latency (Section 4.2). We also assessed the robustness of Cogent to the variation in wireless network bandwidth (Section 4.3) and server load (Section 4.4), demonstrating the importance of a dynamic runtime architecture for collaborative inference speedup. Finally, we verified the performance of the Cogent framework in terms of hardware awareness (Section 4.5).

4.2. Latency Improvement. In this section, we examine the latency improvement that can be brought about by using the Cogent collaborative intelligence framework proposed in this paper. Figure 2 shows per-layer execution latency and output data size after each layer's execution (input for next layer) of the baseline approach, the status quo approach, and Cogent

(1) **Input**:
(2) N: number of layers in the DNN
(3) $\{L_t \mid t = 1, \ldots, N\}$: layers in the DNN
(4) *Agent* (L_t): reinforcement learning agent predicting the output parameters and latency of executing L_t
(5) B: current wireless network uplink bandwidth
(6) E: current edge server load
(7) H: hardware accelerator's feedback
(8) **procedure** THE FIRST STEP
(9) **for each** t *in* $1 \ldots N$ **do**
(10) $D_t \longleftarrow \text{Agent}_{\text{moblie}}(L_t)$
(11) $T_m \xleftarrow{H_m} \text{Agent}_{\text{moblie}}(L_t)$
(12) $T_c \xleftarrow{H_c, E} \text{Agent}_{\text{cloud}}(L_t)$
(13) $T_t \; T_t \longleftarrow D_t / B$
(14) **end for**
(15) **return** Partition point $\longrightarrow$ $\boldsymbol{Action_p}$

$$j = \text{argmin}_{j=1\ldots N}\left(\sum_{k=1}^{j} Tm_k + \sum_{k=j+1}^{N} Tc_k + Tt_j\right)$$

(16) **procedure** THE SECOND STEP
(17) **for each** t *in* $1 \ldots N$ **do**
if OptTarget $\geq$ $_{\min}$(Accuracy) **then**
(18) **return** Pruning ratio $\longrightarrow$ $Action_t \; P_t \xleftarrow{H_{m,c}} \text{Agent}(L_t)$
(19) **end if**
(20) **end for**
(21) **return** NULL

Algorithm 1: Automated model pruning and partition algorithm.

executing VGG model on the mobile device, respectively. The histogram in Figure 2 shows the per-layer execution latency, which shows that a fully cloud-based baseline approach has significantly more execution latency than the other two model partition methods due to network conditions and edge server load. The status quo method is mainly used to prune to the network layer at the back end of the model, so that the network layer at the front of the execution model will also face a large execution latency. Cogent architecture minimizes the latency of the network layer's execution on mobile devices by automatically pruning each layer of the VGG model. The broken line in Figure 2 shows the size of output data after each layer's execution. It can be seen that the network layer parameter output through the Cogent framework has been significantly reduced, which can fully reduce the transmission latency from the terminal device to the edge server. Combining the results of the per-layer execution latency and size of output data after each layer's execution, Cogent predicts that the best latency optimization can be obtained by partitioning the VGG model in the pool3 layer.

We show a comparison of the inference results of running the VGG model through three different methods in Table 2. From the second column of the table, we can see that Cogent can almost guarantee the inference accuracy of the VGG model. The third and fourth columns represent the proportion of the pruning parameters and the number of parameters remaining in the model, respectively. The fifth and sixth columns represent the proportion of the pruning calculation and the number of

Table 1: The configurations of device and edge accelerators.

	Hardware	Batch	PE array	AXI port	Block RAM
Device	Zynq-7020	1	8 × 8	4 × 64b	140 × 36 Kb
Edge server	VU9P	16	16 × 16	4 × 256b	2160 × 36 Kb

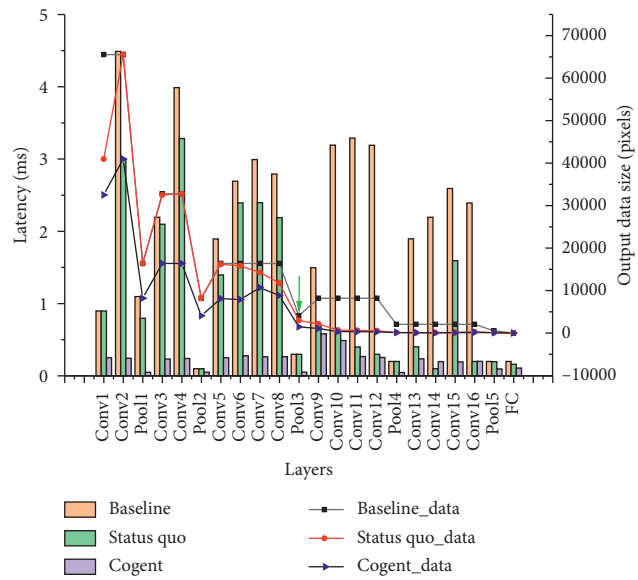

Figure 2: Comparisons of per-layer execution latency and output data size of different strategies.

Table 2: Comparisons of pruned parameter ratio and latency speedup of different strategies.

Prune process	Accuracy (%)	Pruned (%)	Parameter (M)	Pruned (%)	Mult-Adds (M)	Time-32
VGGNET (baseline)	94.64	—	20.3	—	398.14	69.0641 ms
VGGNET (status)	93.34	67.5	6.51	37.2	250.03	24.8501 ms (2.78×)
VGGNET (cogent)	93.82	88.5	2.31	50.8	195.87	7.7703 ms (8.89×)

residual multiplier calculations, respectively. The last column shows the average inference time when the input test image set is $32 * 32$. From the table, the inference latency acceleration of the Cogent framework reached a maximum of 8.89×.

Since the loss of information in the feature mapping pruning may affect the accuracy of the model, we study the trade-off between the percentage of pruning parameters and the accuracy of the model. Figure 3 shows the trade-off between the accuracy loss of Cogent and the percentage of pruning parameters. This curve represents the accuracy loss threshold of the model implemented on the CIFAR-10 dataset, which corresponds to the percentage of model pruning parameters by the Cogent architecture. We observed that, for VGG19 networks, the percentage of pruning parameters less than 90% can guarantee the accuracy loss less than 5%.

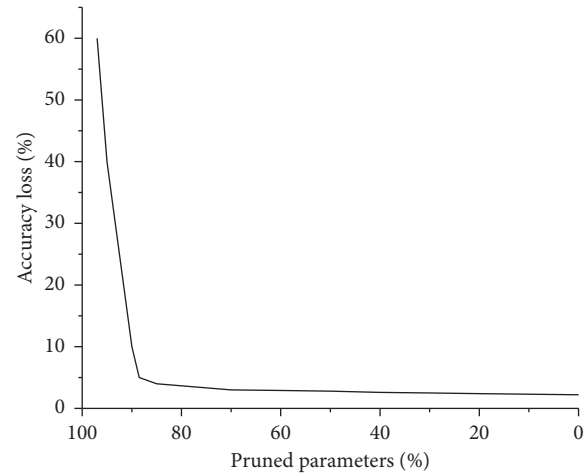

Figure 3: Accuracy loss versus the percentage of the pruned parameter with Cogent.

4.3. Impact of Network Bandwidth Variation. In this section, we evaluate the resilience of Cogent to the variation in wireless network bandwidth between terminal devices and the edge server. In Figure 4, the purple line shows the wireless bandwidth we configure through the traffic control (TC) infrastructure. The green and orange curves show the end-to-end latency of the status quo method and Cogent performing VGG19 on the mobile device platform, respectively. As you can see, the status quo approach is easily affected by network bandwidth variation, so application latency increases significantly during the low bandwidth phase. In contrast, Cogent can effectively adapt to variation in network bandwidth and provide consistent low latency. The main reason is that Cogent can dynamically adjust the partition point and pruning ratio based on the available bandwidth to change the amount of data transmission, thereby minimizing the impact of network bandwidth variation.

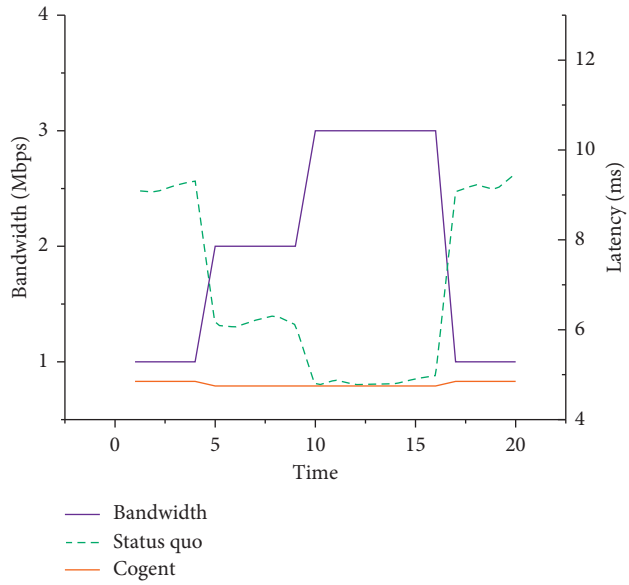

Figure 4: The effect of bandwidth variation on end-to-end latency.

4.4. Impact of Server Load Variation. In this section, we evaluate how Cogent makes dynamic decisions to the variation in the edge server load, so we assume that the network bandwidth is sufficient. Servers typically have high and low traffic queries, and a high server load will increase in the service time for DNN queries. Cogent makes the best decision to meet the current load state by periodically sending query information to the edge server to get the server's occupancy status. Figure 5 shows the end-to-end inference latency of VGG19, which is implemented by the status quo method and Cogent as the edge server load increases. The status quo method does not dynamically adapt to different server loads and therefore suffers significant performance degradation as the server load increases. On the other hand, by considering the server load, Cogent dynamically selects the partition point and pruning ratio to adapt well to the variation. In Figure 5, two vertical dashed lines indicate that Cogent has changed its computing strategy: from completely edge server execution at low load to partitioning the DNN between the mobile device and edge server at medium load and eventually completely local execution on the mobile device when the load is above 80%. Cogent keeps the end-to-end latency of performing image classification below 10 ms regardless of the server load. By considering the server load and its impact on server performance, Cogent always provides the best latency regardless of the variation in server load.

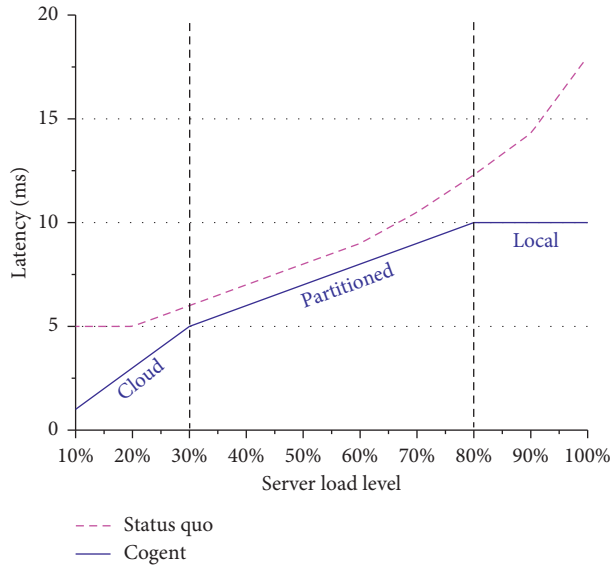

Figure 5: Cogent adjusts its partitioned execution as the result of varying edge server load.

Table 3: Inference latency of VGGNET on four hardware architectures under different strategies.

Inference latency on	HW1 (ms)	HW2 (ms)	HW3 (ms)	HW4 (ms)
VGGNET (baseline)	69.0641	69.1523	64.2927	16.2941
VGGNET (status)	24.8501	25.1915	19.9543	19.9781
VGGNET (cogent)	7.7703	12.1214	11.4689	3.7743

4.5. Impact of Hardware Architecture. In this section, we evaluate the adaptability of Cogent to different hardware accelerators. Because the behavior of different hardware is very different, the performance of the model on the hardware is not always accurately reflected by the proxy signal. Therefore, receiving performance feedback directly from the hardware architecture is important to adapt to the operating environment of the model. The experiment set up four different hardware architectures: HW1: device accelerator1, HW2: device accelerator2, HW3: device accelerator3, and HW4: edge server accelerator. HW1 and HW4 are already described in Section 4.1, HW2 is a Raspberry Pi 3 with a quad-core 1.2 GHz ARM processor and 1 GB RAM, and HW3 is a mobile device with 1 NVIDIA Quadro K620 GPU. As can be seen from Table 3, a solution usually can only achieve optimal performance on hardware architecture. The Cogent framework we proposed can use reinforcement learning to automatically predict pruning and partition strategies based on given hardware feedback so that it can adapt to different hardware architectures. From the comparison results in Figure 6, it can be seen that Cogent running the same intelligent model can obtain better inference acceleration on different hardware architectures, reaching a maximum

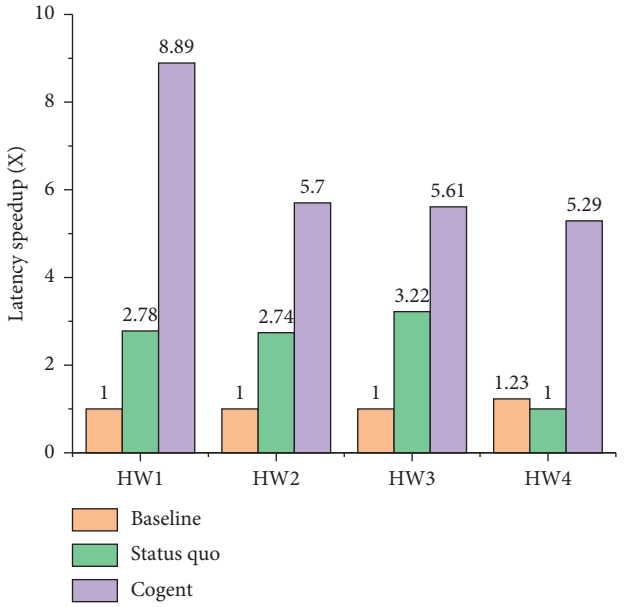

Figure 6: Comparison of latency speedup on four hardware architectures under different strategies.

of 8.89×. Besides, the reason why the baseline method performs better than the status quo method on HW4 is that the baseline method has better adaptability to cloud-based hardware architectures.

5. Conclusion

In this paper, we propose a device-edge synergy intelligent acceleration framework Cogent based on reinforcement learning. The framework receives hardware configuration and system status feedback in a learning-based manner, uses

DDPG agents to automatically predict model pruning and partition strategies, and uses container technology to flexibly deploy partitioned model blocks. The Cogent framework includes two operational phases: automated pruning and partition phase and containerized deployment phase. Through the automatic pruning and partition stage of Cogent, the amount of computation and data transmission of intelligent model inference can be greatly reduced. Through the containerized deployment stage of Cogent, the flexibility and reliability of the system can be greatly improved. Our simulation results show that the Cogent acceleration framework has a significantly latency improvement compared to the completely cloud-based method and the representative partition synergy method when meeting user accuracy requirements. Besides, the Cogent framework also has better adaptability to network bandwidth, server load, and different hardware architectures. In future work, we hope that Cogent has user memory for the data cache and resource requirements of service requests. In the stage of pruning and partition of the model, Cogent can adjust the model according to the user's request habits to make the service more suitable for the user's personalization. User's personalization is a characteristic of the development of artificial intelligence services. The future service framework will only be recognized by users if it develops in a direction that better suits the needs of users.

Data Availability

No data were used to support this study.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

References

[1] C. Szegedy, W. Liu, and Y. Jia, "Going deeper with convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1–9, Boston, MA, USA, June 2015.

[2] A. v. d. Oord, S. Dieleman, and H. Zen, "Wavenet: A generative model for raw audio," 2016, https://arxiv.org/abs/1609.03499.

[3] X. Chi, C. Yan, H. Wang, W. Rafique, and L. Qi, "Amplified LSH-based recommender systems with privacy protection," *Concurrency and Computation: Practice and Experience*, vol. 1, 2020.

[4] A. Almomani, M. Alauthman, F. Albalas, O. Dorgham, and A. Obeidat, "An online intrusion detection system to cloud computing based on NeuCube algorithms," in *Cognitive Analytics*, pp. 1042–1059, IGI Global, PA,USA, 2020.

[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," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1738–1762, 2019.

[6] R. Hadidi, J. Cao, M. Woodward, M. Ryoo, and H. Kim, "Distributed perception by collaborative robots," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, p. 1, 2018.

[7] Y. Kang, J. Hauswald, C. Gao et al., "Neurosurgeon: collaborative intelligence between the cloud and mobile edge," in *ACM SIGARCH Computer Architecture News*, vol. 45, no. 1, pp. 615–629, 2017.

[8] S. Teerapittayanon, B. McDanel, and H.-T. Kung, "Distributed deep neural networks over the cloud, the edge and end devices," in *Proceedings of the IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pp. 328–339, Atlanta, GA, USA, June 2017.

[9] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 6869–6898, 2017.

[10] F. Tung and G. Mori, "Deep neural network compression by in-parallel pruning-quantization," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, no. 3, pp. 568–579, 2020.

[11] T. Paine, H. Jin, J. Yang et al., "GPU asynchronous stochastic gradient descent to speed up neural network training," 2013, https://arxiv.org/abs/1312.6186.

[12] J. Yu, A. Lukefahr, D. Palframan, G. Dasika, R. Das, and S. Mahlke, "Scalpel," *ACM SIGARCH Computer Architecture News*, vol. 45, no. 2, pp. 548–560, 2017.

[13] A. G. Howard, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," 2017, https://arxiv.org/abs/1704.04861.

[14] H. Halawa, H. A. Abdelhafez, A. Boktor, and M. Ripeanu, "NVIDIA jetson platform characterization," in *Proceedings of the European Conference on Parallel Processing*, pp. 92–105, Santiago de Compostela, Spain, August 2017.

[15] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 6848–6856, Salt Lake UT, USA, June 2018.

[16] L. Qi, X. Wang, X. Xu, W. Dou, and S. Li, "Privacy-aware cross-platform service recommendation based on enhanced locality-sensitive hashing," *IEEE Transactions on Network Science and Engineering*, vol. 42, no. 1 2020.

[17] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," 2015, https://arxiv.org/abs/1510.00149.

[18] Y.-D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin, "Compression of deep convolutional neural networks for fast and low power mobile applications," 2015, https://arxiv.org/abs/1511.06530.

[19] N. D. Lane, S. Bhattacharya, A. Mathur, C. Forlivesi, and F. Kawsar, "DXTK: Enabling resource-efficient deep learning on mobile and embedded devices with the DeepX toolkit," in *Proceedings of the 8th EAI International Conference on Mobile Computing, Applications and Services*, pp. 98–107, Cambridge, UK, November 2016.

[20] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient inference," 2016, https://arxiv.org/abs/1611.06440.

[21] J.-H. Luo, J. Wu, and W. Lin, "Thinet: A filter level pruning method for deep neural network compression," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pp. 5058–5066, Venice, Italy, October 2017.

[22] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *Proceedings of the IEEE*

International Conference on Computer Vision (ICCV), pp. 1389–1397, Venice, Italy, October 2017.

[23] H. Cai, T. Chen, W. Zhang, Y. Yu, and J. Wang, "Efficient architecture search by network transformation," in *Proceedings of the National Conference on Artificial Intelligence*, pp. 2787–2794, New Orleans, LA, USA, Febraury 2018.

[24] C. Liu, "Progressive neural architecture search," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pp. 19–34, Munich, Germany, September 2018.

[25] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, "Efficient neural architecture search via parameter sharing," 2018, https://arxiv.org/abs/1802.03268.

[26] K. Ovtcharov, O. Ruwase, J.-Y. Kim, J. Fowers, K. Strauss, and E. S. Chung, "Accelerating deep convolutional neural networks using specialized hardware," *Microsoft Research Whitepaper*, vol. 2, no. 11, pp. 1–4, 2015.

[27] S. Han, X. Liu, H. Mao et al., "Eie," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 243–254, 2016.

[28] X. Xu, R. Mo, F. Dai, W. Lin, S. Wan, and W. Dou, "Dynamic resource provisioning with fault tolerance for data-intensive meteorological workflows in cloud," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 9, pp. 6172–6181, 2019.

[29] X. Xu, C. He, Z. Xu, L. Qi, S. Wan, and M. Z. A. Bhuiyan, "Joint optimization of offloading utility and privacy for edge computing enabled IoT," *IEEE Internet of Things Journal*, vol. 7, no. 4, pp. 2622–2629, 2020.

[30] W. Zhong, X. Yin, X. Zhang et al., "Multi-dimensional quality-driven service recommendation with privacy-preservation in mobile edge environment," *Computer Communications*, vol. 157, pp. 116–123, 2020.

[31] H. Liu, H. Kou, C. Yan, and L. Qi, "Keywords-driven and popularity-aware paper recommendation based on undirected paper citation graph," *Complexity*, vol. 2020, pp. 1–15, Article ID 2085638, 2020.

[32] H. Liu, H. Kou, C. Yan, and L. Qi, "Link prediction in paper citation network to construct paper correlated graph," *EURASIP Journal on Wireless Communications and Networking*, vol. 2019, no. 1, 2019.

[33] L. Qi, Q. He, F. Chen et al., "Finding all you need: Web APIs recommendation in web of things through keywords search," *IEEE Transactions on Computational Social Systems*, vol. 6, no. 5, pp. 1063–1072, 2019.

[34] Y. Chen, N. Zhang, Y. Zhang, X. Chen, W. Wu, and X. Sherman Shen, "Energy efficient dynamic offloading in mobile edge computing for internet of things," *IEEE Transactions on Cloud Computing*, vol. 13, p. 1, 2019.

[35] J. Zhou, Y. Wang, K. Ota, and M. Dong, "AAIoT: Accelerating artificial intelligence in IoT systems," *IEEE Wireless Communications Letters*, vol. 8, no. 3, pp. 825–828, 2019.

[36] R. Hadidi, J. Cao, M. Woodward, M. S. Ryoo, and H. Kim, "Musical chair: Efficient real-time recognition using collaborative Iot devices," 2018, https://arxiv.org/abs/1802.02138.

[37] X. Xu, S. Fu, W. Li, F. Dai, H. Gao, and V. Chang, "Multi-objective data placement for workflow management in cloud infrastructure using NSGA-II," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 9, pp. 1–11, 2019.

[38] J. Li, T. Cai, K. Deng, X. Wang, T. Sellis, and F. Xia, "Community-diversified influence maximization in social networks," *Information Systems*, vol. 92, pp. 1–12, Article ID 101522, 2020.

[39] E. Li, Z. Zhou, and X. Chen, "Edge intelligence," in *Proceedings of the 2018 Workshop on Mobile Edge Communications*, pp. 31–36, Budapest, Hungary, August 2018.

[40] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," 2016, https://arxiv.org/abs/1611.01578.

[41] H. Cai, J. Yang, W. Zhang, S. Han, and Y. Yu, "Path-level network transformation for efficient architecture search," 2018, https://arxiv.org/abs/1806.02639.

[42] 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 *Proceedings of the European Conference on Computer Vision*, pp. 784–800, Munich, Germany, September 2018.

[43] Z. Zhong, J. Yan, and C.-L. Liu, "Practical network blocks design with *Q*-learning," 2017, https://arxiv.org/abs/1708.05552.

[44] B. J. A. Kröse, "Learning from delayed rewards," *Robotics and Autonomous Systems*, vol. 15, no. 4, pp. 233–235, 1995.

[45] W. Farhat, H. Faiedh, C. Souani, and K. Besbes, "Real-time hardware/software co-design of a traffic sign recognition system using Zynq FPGA," in *Proceedings of the 11th International Design & Test Symposium (IDT)*, Hammamet,Tunisia, December 2016.

[46] X. V. Ultra, "UltraScale architecture and product data sheet: Overview," vol. 2018, p. 28, 2018.

[47] Y. Xu, J. Ren, Y. Zhang, C. Zhang, B. Shen, and Y. Zhang, "Blockchain empowered arbitrable data auditing scheme for network storage as a service," *IEEE Transactions on Services Computing*, vol. 13, no. 2, pp. 289–300, 2020.

[48] Y. Abouelnaga, O. S. Ali, H. Rady, and M. Moustafa, "Cifar-10: knn-based ensemble of classifiers," in *Proceedings of the International Conference on Computational Science and Computational Intelligence (CSCI)*, pp. 1192–1195, Las Vegas, NV, USA, December 2016.

[49] H. Wu, Q. Wang, and K. Wolter, "Methods of cloud-path selection for offloading in mobile cloud computing systems," in *Proceedings of the 4th IEEE International Conference on Cloud Computing Technology and Science*, pp. 443–448, Taipei, Taiwan, December 2012.

[50] D. Liu, X. Chen, Z. Zhou, and Q. Ling, "HierTrain: Fast hierarchical edge AI learning with hybrid parallelism in mobile-edge-cloud computing," *IEEE Open Journal of the Communications Society*, vol. 1, pp. 634–645, 2020.