RobustDiCE: Robust and Distributed CNN Inference at the Edge

Xiaotian Guo
Univ. of Amsterdam, Leiden Univ.
Amsterdam, Netherlands
x.guo3@uva.nl

Quan Jiang
Nanjing Agricultural Univ.
Nanjing, China
aapool@outlook.com

Andy D. Pimentel
Univ. of Amsterdam
Amsterdam, Netherlands
a.d.pimentel@uva.nl

Todor Stefanov
Leiden Univ.
Leiden, Netherlands
t.p.stefanov@liacs.leidenuniv.nl

2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC) | 979-8-3503-9354-5/24/$31.00 ©2024 IEEE | DOI: 10.1109/ASP-DAC58780.2024.10473970

Abstract— Prevalent large CNN models pose a significant challenge in terms of computing resources for resource-constrained devices at the Edge. Distributing the computations and coefficients over multiple edge devices collaboratively has been well studied but these works generally do not consider the presence of device failures (e.g., due to temporary connectivity issues, overload, discharged battery, etc. of edge devices). Such unpredictable failures can compromise the reliability of edge devices, inhibiting the proper execution of distributed CNN inference. In this paper, we present a novel partitioning method, called RobustDiCE, for robust distribution and inference of CNN models over multiple edge devices. Our method can tolerate intermittent and permanent device failures in a distributed system at the Edge, offering a tunable trade-off between robustness (i.e., retaining model accuracy after failures) and resource utilization. We evaluate RobustDiCE using the ImageNet-1K dataset on several representative CNN models under various device failure scenarios and compare it with several state-of-the-art partitioning methods as well as an optimal robustness approach (i.e., full neuron replication). In addition, we demonstrate RobustDiCE's advantages in terms of memory usage and energy consumption per device, and system throughput for various system set-ups with different device counts.

I. Introduction

As Artificial Intelligence (AI) continues its rapid evolution, convolutional neural networks (CNNs) are becoming increasingly prevalent across a variety of applications [1]. And the surge of Internet-of-Things (IoT) devices has also elevated the deployment requirements of CNNs at the Edge. However, the growing complexity and size of CNN models, such as VGG-16 [2], and CoAtNet-6 [3], pose a significant challenge in terms of computing resources for resource-constrained edge devices. One approach to address this challenge is to construct a lightweight CNN model from a large CNN model utilizing model compression [4] or neural architecture search [5] which may decrease accuracy. Another approach is to distribute inference models between edge devices and cloud servers [6], but this approach introduces unpredictable inference latency and raises trustworthiness, security, and privacy concerns.

To address these issues, studies on fully distributing the CNN inference over multiple edge devices have been proposed without the need for model compression and cloud servers. In such a *horizontal CNN distribution paradigm*, model partitioning [7], [8] and data partitioning [9], [10] methods are typically applied to alleviate the discrepancy between the constrained resources of edge devices and the huge requirements of deploying large CNN models. However, these partitioning methods assume continuous availability of all involved edge devices that cannot be always guaranteed because an edge device could be temporarily unreachable (especially when edge devices are mobile and use low-power short distance radios for communication) or a device could experience a temporary failure (e.g., due to a discharged battery). Therefore, it is imperative to devise and utilize partitioning methods for distributed CNN inference with robustness in mind.

In this paper, we present a novel partitioning method, called **RobustDiCE**, for robust distribution and inference of CNN models over multiple edge devices. RobustDiCE features both *system robustness*, i.e., CNN inference can continue execution even if one or more edge devices fail to function properly, and *model robustness*, i.e. preserving the inference accuracy of the CNN model as much as possible when some of the intermediate CNN inference results are lost due to failed devices. In this paper, however, *we solely focus on RobustDiCE's model robustness*. We address this model robustness challenge by evaluating the relative importance of each neuron in the CNN model and then partitioning these different neurons of each CNN layer into different groups (to be mapped to the various edge devices) as 'evenly' as possible. Our main novel contributions can be summarized as follows:

- Based on the importance criterion of different neurons in each CNN layer, a new partitioning method is proposed to preserve the model accuracy as much as possible against device failures. This new method combines *partial* neuron replication and *importance-aware* neuron clustering to achieve CNN model robustness. It also provides a tunable trade-off between robustness (i.e., retaining model accuracy after failures) and resource utilization.
- We evaluate our novel partitioning method using the ImageNet-1K dataset on several representative CNN models under pessimistic device failure scenarios. We compare it with a number of state-of-the-art (partitioning) approaches, including the CDC method [11] leveraging neuron replication to increase robustness and an ideal robustness approach utilizing full neuron replication.
- We demonstrate our method's superiority in terms of memory usage and energy consumption per device, and system throughput under different system configurations.

II. Related Work

Model and data partitioning methods [7], [12] can be exploited to distribute the workload of a large CNN model inference along the edge-cloud continuum or fully among multiple edge devices, thus reducing the required computation resources of edge devices [8]. However, these partitioning methods assume that the involved computing devices/servers (and communication links) between them are always available and work properly. Our partitioning method is designed to be robust against temporary or permanent failures of devices.

The robustness of distributed CNN inference concerns the property of a model of being resilient in terms of inference accuracy to the failure of physical computing nodes due to power outages, unstable inter-node connections, other hardware/software failures, etc. In distributed CNN inference, the missing neurons on those failed nodes may result in a significant accuracy drop of a CNN model [11]. The code distributed computing (CDC) method in [11] utilizes one additional, presumed functional device to back up the summation of partitioned neurons of other distributed devices and use that to recover the output of missing neurons in the event of a single node failure. Our method, on the other hand, can cope with multiple node failures without integrating additional devices and computations.

To minimize the influence of node failures on the CNN inference accuracy, several failure-aware retraining methods [13], [14] for CNNs also have been developed. For example, DeepFogGuard [13] utilizes retraining to add hyperconnections that can skip certain failed physical nodes in a pipe-lined distributed inference. These retrained models are designed to be aware of only specific failures such as communication failures between two CNN layers, certain node failures in a pipeline multi-node inference, etc. Moreover, CNN retraining requires a large amount of data that may not be always accessible for an end user of a pre-trained CNN model to perform retraining before the deployment in an unreliable environment. As most pre-trained models directly available to an end user for deployment are not failure-aware, our RobustDiCE method can be easily applied to partition these pre-trained models to achieve model robustness without any retraining, without assuming specific types of failures, and without suffering from model accuracy degradation due to parameter changes (e.g., by retraining). Moreover, RobustDiCE can be seen as complementary to these retraining approaches, i.e., if we would apply our method to the aforementioned retrained models, we can further improve their robustness against node failures.

To summarize, performing robust inference on distributed edge devices is vital. Existing robustness methods suffer from extra computing resource requirements, time-consuming retraining, etc. In contrast, our method RobustDiCE is designed to guarantee robustness against device failures while accounting for the limited resources of edge devices.

III. Background and Motivation

In this section, we provide some background information and a motivational example to understand our novel CNN partitioning method for robustness.

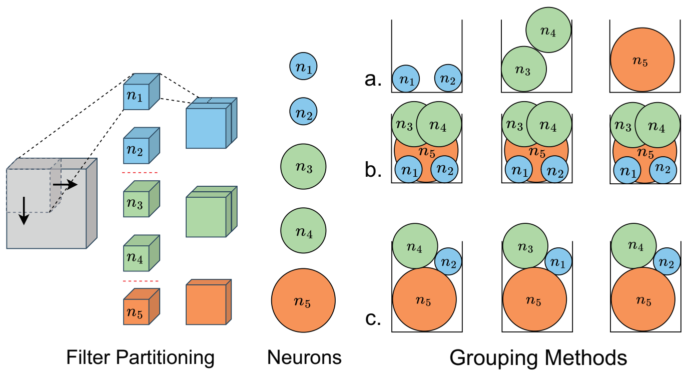


Fig. 1: Typical vs. Robust Partitioning

Generally, state-of-the-art partitioning methods, such as discussed in [7], do not consider robustness as they do not consider the fact that different neurons/filters in CNN layers have different *importance*, thereby causing various effects on the inference accuracy of a CNN model, particularly those neurons with larger values [15]. The relative *importance* of a neuron in a CNN layer can be measured by calculating metrics such as the l_1-norm [16], l_2-norm [17], etc. To partition a CNN layer with robustness in mind, it is essential to find an effective way to group and distribute its neurons/filters over computing nodes as evenly as possible in terms of *importance*.

To clarify this statement, we use the simple example, shown in Figure 1, where we consider a convolution layer with five filters/neurons denoted as n_1 to n_5. We want to partition these neurons over three computing nodes. In this example, the importance score s_j of each neuron n_j is measured by calculating the l_1-norm, i.e., taking the filter corresponding to neuron n_j with shape $C_{in} \times k \times k$ (where k denotes the kernel size of the filter and C_{in} the number of input channels), we calculate the sum of absolute values of all the weights in the filter and its bias as shown in Line 7 of Algorithm 1. In Line 7, $W_j^{c,h,w}$ denotes a particular weight value in the j_{th} filter corresponding to neuron n_j, and b_j its bias. In the middle and the right part of Figure 1, we visualize the importance s_j of each neuron n_j by the size of the circle representing the neuron, i.e., neuron n_5 has the highest importance whereas n_1 and n_2 have the lowest importance.

As shown in Figure 1(a), a partitioning method without robustness in mind (i.e., no consideration of the neurons' importance s_j) splits the five neurons into three groups (visualized by the three colors in Figure 1) and the groups are distributed over the three nodes. Such distribution reduces computational resources per node because the layer workload is split over the nodes. However, this distribution is not robust at all because if, for example, the third node fails, which runs the most important neuron n_5, then the inference accuracy will decrease significantly.

To maximize the robustness, well-known modular redundancy methods can be applied as shown in Figure 1(b). Here, we replicate all neurons over the three nodes, thereby achieving maximum robustness against failures because even if one or two nodes fail then the remaining available node will run all the neurons without a decrease in inference accuracy. However, this significantly increases the resource requirements (e.g., memory and energy consumption) for each node. Moreover, this full replication approach might be infeasible for resource-constrained nodes due to the limitations with respect to their computational/memory resources and the possible energy budget of an edge device.

The two example scenarios, illustrated in Figure 1(a) and (b), clearly show that using existing, robustness-unaware partitioning methods or modular redundancy methods in isolation cannot provide efficient, robust distributed CNN inference on multiple resource-constrained edge devices. Therefore, in this paper, we propose a novel method, explained in detail in Section IV, which *combines replication and importance-aware partitioning* to achieve high and tunable robustness in an efficient way for distributed CNN inference. The result of applying our method to our simple example is illustrated in Figure 1(c). The basic idea is that some (not all) neurons in a CNN layer are replicated and all neurons (initial and replicas) are partitioned into groups and distributed evenly over the nodes based on their *importance*.

The advantage of this partitioning method is that if either the first or third node fails, the remaining nodes can still run all the neurons, preserving inference accuracy. If the second node fails, the critical neuron n_5 still remains, limiting the accuracy degradation. Therefore, we can achieve comparable robustness to the scenario in Figure 1(b), but with reduced computational resource requirements, as not all neurons are replicated or run on each node.

IV. The RobustDiCE Method

In this section, we present our new partitioning method which achieves CNN model robustness by combining importance-aware neuron grouping and clustering with partial neuron replication in order to evenly distribute the neurons in a CNN model over multiple nodes. The partitioning method is applied layer-wise on every layer until the whole CNN model is partitioned. The general layer-wise partitioning procedure is outlined in Algorithm 1. It accepts as inputs a set of computational layers L from the CNN model and their coefficients W as well as the total number of computing nodes ND across which the CNN model will be distributed. Additionally, a set T of threshold values corresponding to layers in L is provided as another input. The threshold values serve as specific criteria for identifying similar neurons in terms of importance, and subsequently making neuron grouping decisions based on the similarity. The output of Algorithm 1 is set P of neuron partitions. Every partition $P_i = \{p^1, ..., p^{ND}\} \in P$ determines how the neurons in layer $l_i \in L$ are distributed across the specified number of computing nodes ND.

The goal of Algorithm 1 is to evenly distribute the neurons n_j of every layer $l_i \in L$ over ND nodes (i.e., devices) in terms of importance. For example, applying Algorithm 1 (Lines 3–29) to the convolution layer with the five neurons n_1 to n_5 shown in Figure 1 and setting $ND = 3$, the output P of the algorithm is the partition illustrated in Figure 1(c). Algorithm 1 consists of three main steps performed on each layer $l_i \in L$.

In Step 1 (Lines 3–9), we first include each neuron $n_j \in l_i$ into a separate group G_j which is stored in the set of groups G (Line 5). Then, we calculate three importance scores for

Algorithm 1: Robust Partitioning

```
   Input : Set of layers $L$; Number of nodes $ND$;
           Set of layer coefficients $W = \{W_1, ..., W_{|L|}\}$;
           Set of threshold values $T = \{t_1, ..., t_{|L|}\}$;
   Output: Set of neuron partitions $P = \{P_1, ..., P_{|L|}\}$;
1  $P \leftarrow \emptyset$
2  for $l_i \in L$ do
       // Step 1: neuron importance scores
3      $G \leftarrow \emptyset$
4      for $n_j \in l_i$ do
5          Create $G_j$; $G_j \leftarrow G_j + n_j$; $G \leftarrow G + G_j$
6          Create $S_j = \{s_j^1, s_j^2, s_j^3\}$
7          $s_j^1 = \sum_{h=1}^{k_h}\sum_{w=1}^{k_w}\sum_{c=1}^{C_{in}} |W_j^{c,h,w}| + |b_j|$
8          $s_j^2 = \sum_{h=1}^{k_h}\sum_{w=1}^{k_w}\sum_{c=1}^{C_{in}} |\frac{\partial \mathbf{y}}{\partial W_j^{c,h,w}} \cdot W_j^{c,h,w}| + |\frac{\partial \mathbf{y}}{\partial b_j} \cdot b_j|$
9          $s_j^3$ =JSD($\mathbf{y}_{\text{complete}} \,||\, \mathbf{y}_{\text{removing neuron } n_j}$)
       // Step 2: neuron clustering
10     for $G_z \in G$ do
11         for $G_q \in G - G_z$ do
12             $d_{max} = 0$
13             for $n_j \in G_z$ do
14                 for $n_o \in G_q$ do
15                     $d(n_j, n_o) = \sqrt{\sum_{a=1}^{3} (s_j^a - s_o^a)^2}$
16                     if $d(n_j, n_o) > d_{max}$ then
17                         $d_{max} = d(n_j, n_o)$
18             if $d_{max} < t_i$ then
19                 $G_z \leftarrow G_z + G_q$
20                 $G \leftarrow G - G_q$
       // Step 3: round-robin distribution
21     Create $P_i = \{p^1, ..., p^{ND}\}$; $p^1 \leftarrow \emptyset, ..., p^{ND} \leftarrow \emptyset$
22     for $G_o \in G$ do
23         if $(|G_o| \mod ND) \neq 0$ then
24             for $j \in [1, ND - (|G_o| \mod ND)]$ do
25                 Create $n_{|G_o|+j}$ = REPLICA($n_j \in G_o$)
26                 $G_o \leftarrow G_o + n_{|G_o|+j}$
27         for $n_j \in G_o$ do
28             $r = (j \mod ND) + 1$; $p^r \leftarrow p^r + n_j$
29     $P \leftarrow P + P_i$
30 return $P$
```

n_j from three different perspectives. The first score s_j^1 (Line 7) is the l_1-norm [16] which is a magnitude-based approach, widely used in CNN pruning techniques, to compute neuron importance based on the sum of its absolute weights and bias. The second importance score s_j^2 (Line 8) of n_j is computed by summing the sensitivity scores of all its connections with other neurons. We use the Taylor expansion approach [18] to obtain the connection sensitivity scores through the gradient in the propagation process [19]. The third score s_j^3 (Line 9) assesses the neuron importance by employing the Jensen-Shannon divergence [20] denoted as JSD. A larger change in the CNN output probability distributions $\mathbf{y}$, induced by removing neuron $n_j \in l_i$, indicates that n_j is more important. Instead of using a single importance score only, set $S_j = \{s_j^1, s_j^2, s_j^3\}$ of the three different scores enables a more comprehensive evaluation of the neuron importance because it performs a three-dimensional assessment of the importance, thereby facilitating a more effective clustering of neurons (see Table I).

In Step 2 (Lines 10–20), Algorithm 1 takes the initial set of groups G created in Line 5, where each group contains only one neuron $n_j \in l_i$, and clusters these 1-neuron groups into a *new* set of groups G where any group may contain multiple neurons with similar importance. To this end, the following two actions are performed iteratively for every two groups $G_z \in G$ and $G_q \in G - G_z$. First, the largest distance d_{max} between the neurons in G_z and G_q is determined in Lines 12–17. Initially, d_{max} is set to zero. Then, for every pair of neurons $n_j \in G_z$ and $n_o \in G_q$, the Euclidean distance $d(n_j, n_o)$ between n_j and n_o in the three-dimensional importance score space (s^1, s^2, s^3) is computed in Line 15. If $d(n_j, n_o)$ is greater than d_{max}, then d_{max} is updated with $d(n_j, n_o)$ in Line 17.

Second, if d_{max} is below a given threshold value $t_i \in T$ then the neurons in G_z and G_q are merged (Line 19) into one group G_z because they are considered similar in terms of importance, and group G_q is removed from set G in Line 20. The threshold value t_i affects the result of the neurons clustering in Step 2. For example, a small t_i would result in set G having many groups with a few neurons per group. If t_i is too small then every group in G will contain only one neuron, thereby "forcing" the following Step 3 in Algorithm 1 to perform full replication of all neurons, thus maximizing the robustness at the expense of high resource requirements per node in the distributed system. In contrast, a large t_i would result in a few groups with many neurons per group. If t_i is too large then all neurons would be clustered into one group, thereby "forcing" Step 3 to perform very limited or no replication of neurons which could lead to a significant reduction of the robustness. Recall that a set T of threshold values t_i is given as an input to Algorithm 1, thus an optimal set of such values could be determined by integrating Algorithm 1 in a design space exploration (DSE) procedure with multiple optimization objectives including distributed CNN inference accuracy, energy and resource requirements per node in the distributed system, and system performance.

Finally, in Step 3 (Lines 21–29), Algorithm 1 distributes all neurons n_j in every group $G_o \in G$ across a number of nodes ND in a round-robin fashion (Lines 27–28). If the number of neurons in group G_o is not a multiple of the number of nodes ND then some neurons in the group are replicated (Lines 23–26) in order to increase the neuron number to the closest multiple of the number of nodes before the round-robin distribution. Such round-robin distribution can guarantee that every node runs the same number of similarly important neurons from a group, thereby providing CNN model robustness by reducing the CNN inference accuracy degradation in the event of failures in the distributed system.

V. Evaluation of the RobustDiCE Method

In this section, we present a range of experiments demonstrating the merits of RobustDiCE in terms of achieved robustness and resource utilization per node/device in a distributed system performing CNN inference.

A. Experimental Setup

We implement RobustDiCE and apply it to the following distributed system configurations and real-world CNNs, and considering the following device failure scenarios.

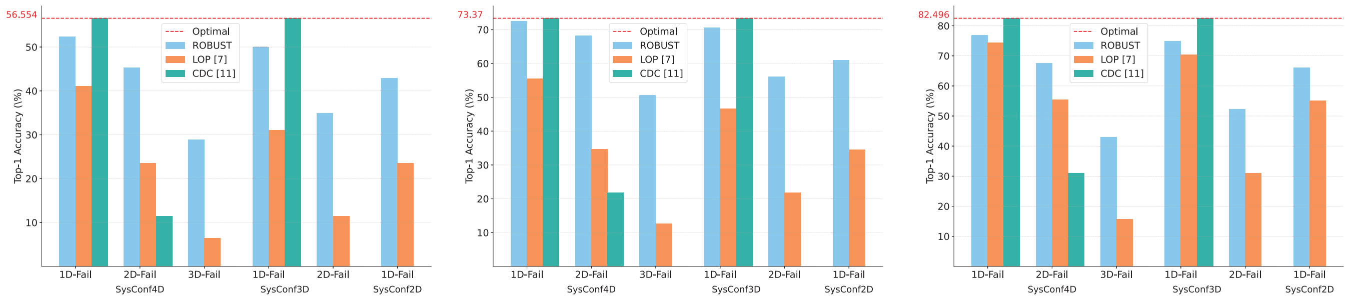
CNNs and System Configurations: We experimented with three CNNs, namely AlexNet [21], VGG16-BN [2], and ConvNext-Tiny [22], taken from the TorchVision library. Given their widespread use in image classification and their diversity in layer types, operation counts, and memory requirements for weights, we consider these CNNs to be representative targets to demonstrate the merits of our method. By applying RobustDiCE, every CNN is distributed for inference on three system configurations: one with four edge devices (**SysConf4D**), one with three devices (**SysConf3D**), and one with two devices (**SysConf2D**). All devices in a system configuration are NVIDIA Jetson Xavier NX boards connected over a Gigabit network switch. Each device has an embedded MPSoC featuring a 6-core Carmel ARMv8.2 CPU, an NVIDIA Volta GPU with 384 CUDA cores, 48 Tensor cores, and 8 GB of LPDDR4x memory.

TABLE I: Top-1 accuracy (1D-Fail case in SysConf4D)

Importance Scores	AlexNet (%)	VGG16_BN (%)	ConvNext_Tiny (%)
s_1	43.718	60.426	76.618
s_2	43.642	58.920	75.904
s_3	43.432	59.942	76.134
$s_1 + s_2$	51.268	69.152	76.678
$s_1 + s_3$	51.658	71.736	76.580
$s_2 + s_3$	51.250	67.360	76.572
$s_1 + s_2 + s_3$	**52.396**	**72.500**	**76.820**

Device Failure Scenarios: For each of the aforementioned CNNs, we consider three scenarios.
Scenario A: The CNN is distributed for inference on system configuration **SysConf4D** where 1 device fails (**1D-Fail**), 2 devices fail (**2D-Fail**), or 3 devices fail (**3D-Fail**).
Scenario B: CNN on **SysConf3D** where **1D-Fail** or **2D-Fail**.
Scenario C: CNN on **SysConf2D** where **1D-Fail**.
Under every scenario with a different number of failing devices, we evaluate the preserved Top-1 accuracy on the ImageNet-1K test dataset when the CNN is distributed using our RobustDiCE method. We compare RobustDiCE to state-of-the-art robustness-unaware partitioning that performs filter and layer output partitioning, referred to as *LOP* [7], as well as the robustness-aware CDC method from [11]. In addition, we also show the Top-1 CNN accuracy results under an ideal scenario, called `Optimal`. This `Optimal` scenario assumes that in system configurations **SysConf4D**, **SysConf3D**, and **SysConf2D** no devices fail or all CNN neurons are replicated on every device in order to have quadruple (QMR), triple (TMR), and dual (DMR) modular redundancy, thus achieving maximum robustness.

By continuously providing 1000 images as an input data stream for the distributed CNN inference, we measure the system performance in images (frames) per second (FPS), memory usage per device in megabytes (MB), and energy consumption per device in joules per image (J/img) of the distributed CNN inference for the different system configurations. We measure the overall latency in processing the 1000 images and compute averaged FPS as throughput. The energy consumption per device, including CPUs, GPU, communication cost, etc., is obtained through a sampling thread reading power values from the INA3221 monitor on the NVIDIA Jetson Xavier NX board. The memory usage per device is reported directly by the executed CNN code itself during the CNN inference.



(a) AlexNet (b) VGG16-BN (c) ConvNext-Tiny

Fig. 2: CNN Model Robustness under different Device Failure Scenarios

TABLE II: System performance and resource utilization

Network	System Configuration	Max. per-device Energy (J/img)	System Throughput (FPS)	Max. per-device Memory (MB)
AlexNet	QMR/TMR/DMR	0.179	46.255	150.914
	CDC-SysConf3D	0.165	43.670	94.117
	CDC-SysConf4D	0.157	45.587	78.852
	Robust-SysConf2D	0.159	48.214	99.254
	Robust-SysConf3D	0.148	50.045	80.777
	Robust-SysConf4D	0.142	51.219	72.801
VGG16-BN	QMR/TMR/DMR	0.850	10.744	429.215
	CDC-SysConf3D	0.809	10.634	313.688
	CDC-SysConf4D	0.799	10.485	272.293
	Robust-SysConf2D	0.826	10.761	328.426
	Robust-SysConf3D	0.799	10.993	295.086
	Robust-SysConf4D	0.779	11.078	267.395
ConvNext-Tiny	QMR/TMR/DMR	0.308	28.223	88.895
	CDC-SysConf3D	0.307	27.107	69.129
	CDC-SysConf4D	0.297	28.248	59.961
	Robust-SysConf2D	0.301	28.044	76.465
	Robust-SysConf3D	0.296	28.415	65.203
	Robust-SysConf4D	0.288	29.034	58.090

B. Experimental Results

Ablation Study of Importance Scores: To substantiate the efficacy of using multi-dimensional importance evaluation for neuron clustering, we carried out an ablation study with various combinations of importance scores (s_1, s_2, s_3). We only list the top-1 accuracy of the 1D-Fail case for the SysConf4D system configuration in Table I due to paper page limitations but the other failure scenarios show similar results. It is clear that the combination of all three scores preserves top-1 accuracy (model robustness) the best under the 1D-fail scenario for all three models: 52.396% (AlexNet), 72.500% (VGG16_BN), and 76.820% (ConvNext_Tiny). These findings confirm the potential for enhancing model robustness in distributed CNN inference using the combination of multiple importance scores.

Model Robustness Comparison: The results, obtained with the experimental setup described in Section V-A, are presented in Figure 2 and Table II. For every CNN model, we show a graph where the X-axis represents the considered scenarios with a different number of failing devices, and the Y-axis indicates the evaluated Top-1 CNN model accuracy. For every scenario and number of failing devices, we plot a bar for the RobustDiCE results (blue bar), LOP results (orange bar), and CDC results (green bar). In addition, the horizontal dashed (red) line shows the accuracy under the `Optimal` scenario.

Looking at the blue and orange bars in Figure 2, we observe that RobustDiCE consistently delivers higher Top-1 accuracy compared to the state-of-the-art but robustness-unaware LOP partitioning method. This clearly demonstrates the superiority of our method in terms of CNN model robustness. Taking Figure 2(a) as an example, the Top-1 accuracy of AlexNet under the `Optimal` scenario is 56.55% which is our reference point. When a system configuration experiences device failures as in Scenario A, our RobustDiCE method delivers a Top-1 accuracy of 52.40%, 45.28%, and 28.93% for cases 1D-Fail, 2D-Fail, and 3D-Fail, respectively. In contrast, the LOP method exhibits more significant drop in accuracy, namely 41.07%, 23.50%, and 6.42% for the same device failure cases. A similar trend can be observed for VGG-16BN and ConvNext-Tiny in Figure 2(b) and (c), respectively. Here, we have used an optimistic device failure scenario for LOP, i.e., devices with the least important groups of neurons fail.

Comparing our RobustDiCE method (orange bars) with the CDC method (green bars), we see that CDC is capable of perfectly handling a single device failure due to its approach of using actor replication and a spare node. However, the CDC method cannot handle multiple device failures, resulting in very low accuracy (much lower than RobustDiCE) or even complete failure (0% accuracy) when all but one devices fail.

Looking at Figure 2 and comparing the Top-1 accuracy delivered by RobustDiCE with the reference accuracy under the `Optimal` scenario, we observe that our method does not maintain the reference accuracy level in the event of device failures. The reason is that, in this experiment, we set threshold values $t_i \in T$ discussed in Section IV to be greater than 0. Because of this, our method does not replicate all CNN neurons on every device, thereby trading off CNN model robustness (loss of Top-1 accuracy) for reduced system resource utilization. This tradeoff could be tuned by changing the t_i values. Moreover, if all t_i values are set to 0 then our method will maintain Top-1 accuracy at the same level as under the `Optimal` scenario. Under this scenario, all CNN neurons are replicated on every device in order to have quadruple (QMR), triple (TMR), or dual (DMR) modular redundancy, thus achieving maximum robustness. However, achieving this maximum robustness is at the expense of higher memory usage and energy consumption per device compared to the resource utilization, imposed by our method, when trading off robustness against utilization. This statement is supported by the resource utilization results in Table II. In this table, for every CNN, we show the maximum per-device memory usage (Column 5), the maximum per device energy consumption (Column 3), and the overall system throughput (Column 4) for the three system configurations SysConf4D, SysConf3D, and SysConf2D with our RobustDiCE method and the CDC method as well as for the QMR/TMR/DMR configuration associated with the `Optimal` scenario.

System Performance: Considering the memory usage numbers for AlexNet, shown in Column 5, we see that the replication of all neurons on every device in system configuration QMR/TMR/DMR requires about 150 MB of memory per device. In contrast, our RobustDiCE method significantly reduces the required memory per device, i.e., with 51.76% for system configuration SysConf4D, with 46.47% for SysConf3D, and with 34.23% for SysConf2D. Significant memory reduction trends can be observed in Column 5 for VGG16-BN and ConvNext-Tiny as well. The memory usage numbers for CDC show that this method reduces the memory footprint in comparison to the all-neuron replication method (QMR/TMR/DMR) but still has higher memory usage compared to RobustDiCE.

The energy consumption per device is also reduced by RobustDiCE as compared to applying all-neuron replication to achieve CNN model robustness. For example, Column 3 in Table II shows that our method applied on SysConf4D achieves an effective energy reduction over the all-neuron replication method (QMR/TMR/DMR), i.e., 20.67% reduction for AlexNet, 8.35% for VGG16-BN, and 6.49% for ConvNext-Tiny. The CDC energy results again show an improved behavior compared to QMR/TMR/DMR but are inferior to the results from RobustDiCE.

Finally, as shown in Column 4 of Table II, RobustDiCE slightly improves the system throughput for almost all CNNs and system configurations as compared to QMR/TMR/DMR (except for SysConf2D on ConvNext-Tiny). For CDC, on the other hand, the system throughput is generally lower than QMR/TMR/DMR and RobustDiCE.

We note, however, that the system throughput of distributed CNN inference is highly dependent on the quality of the network interconnecting the devices in the system. In our experiments, we have used a Gigabit network switch. Evidently, in other edge/IoT settings, the connectivity between devices may have a lower bandwidth, e.g., using WiFi or other wireless protocols. Thus, our RobustDiCE method cannot always guarantee system throughput improvements but it can guarantee memory usage and energy consumption reductions.

VI. Conclusions

This paper presented RobustDiCE, a robust partitioning method for distributed CNN inference at the Edge that preserves the model accuracy as much as possible against device/link failures. Several CNN experiments demonstrated that RobustDiCE can retain the CNN model accuracy after failures much better as compared to the state-of-the-art partitioning methods. We have also shown the advantages of our RobustDiCE method over the optimal robustness approach and CDC method in terms of memory usage per device, energy consumption per device, and system throughput.

Acknowledgments

I am profoundly grateful to Qi Wang for his inspirational guidance and indispensable support for this research. I am also thankful to Yixian Shen, who provided insightful suggestions for the revision.

References

[1] J. Deng *et al.*, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on CVPR*, 2009, pp. 248–255.

[2] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[3] Z. Dai, H. Liu, Q. V. Le, and M. Tan, "Coatnet: Marrying convolution and attention for all data sizes," *Advances in NeurIPS*, vol. 34, pp. 3965–3977, 2021.

[4] Y. Guo, "A survey on methods and theories of quantized neural networks," 2018. [Online]. Available: https://arxiv.org/abs/1808.04752

[5] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *The Journal of Machine Learning Research*, vol. 20, no. 1, pp. 1997–2017.

[6] Y. Kang *et al.*, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," *ACM SIGARCH Computer Architecture News*, vol. 45, no. 1, pp. 615–629, 2017.

[7] R. Stahl, A. Hoffman, D. Mueller-Gritschneder, A. Gerstlauer, and U. Schlichtmann, "Deeperthings: Fully distributed cnn inference on resource-constrained edge devices," *International Journal of Parallel Programming*, vol. 49, no. 4, pp. 600–624, 2021.

[8] X. Guo, A. D. Pimentel, and T. Stefanov, "Automated exploration and implementation of distributed cnn inference at the edge," *IEEE IoT Journal*, vol. 10, no. 7, 2023.

[9] L. Zhou, M. H. Samavatian, A. Bacha, S. Majumdar, and R. Teodorescu, "Adaptive parallel execution of deep neural networks on heterogeneous edge devices," in *SEC*, 2019, pp. 195–208.

[10] J. Mao, X. Chen, K. W. Nixon, C. Krieger, and Y. Chen, "Modnn: Local distributed mobile computing system for deep neural network," in *IEEE DATE*, 2017, pp. 1396–1401.

[11] R. Hadidi, J. Cao, B. Asgari, and H. Kim, "Creating robust deep neural networks with coded distributed computing for iot," in *IEEE International Conference on Edge Computing and Communications*, 2023.

[12] E. Aghapour, D. Sapra, A. Pimentel, and A. Pathania, "Cpu-gpu layer-switched low latency cnn inference," in *2022 25th Euromicro Conference on Digital System Design (DSD)*, 2022, pp. 324–331.

[13] A. Yousefpour *et al.*, "Guardians of the deep fog: Failure-resilient dnn inference from edge to cloud," in *Workshop on challenges in artificial intelligence and machine learning for IoT*, 2019, pp. 25–31.

[14] S. Itahara, T. Nishio, and K. Yamamoto, "Packet-loss-tolerant split inference for delay-sensitive deep learning in lossy wireless networks," in *IEEE GLOBECOM*, 2021, pp. 1–6.

[15] J. L. Bernier, J. Ortega, E. Ros, I. Rojas, and A. Prieto, "A quantitative study of fault tolerance, noise immunity, and generalization ability of mlps," *Neural Computation*, vol. 12, no. 12, pp. 2941–2964, 2000.

[16] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," *arXiv*, 2016. [Online]. Available: https://arxiv.org/abs/1608.08710

[17] Y. He, G. Kang, X. Dong, Y. Fu, and Y. Yang, "Soft filter pruning for accelerating deep convolutional neural networks," *arXiv*, 2018.

[18] N. Lee, T. Ajanthan, and P. H. Torr, "Snip: Single-shot network pruning based on connection sensitivity," *arXiv*, 2018.

[19] S.-K. Yeom *et al.*, "Pruning by explaining: A novel criterion for deep neural network pruning," *Pattern Recognition*, vol. 115, p. 107899, 2021.

[20] B. Fuglede and F. Topsoe, "Jensen-shannon divergence and hilbert space embedding," in *Int. symposium on Information theory*, 2004, p. 31.

[21] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Comm. of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[22] Z. Liu *et al.*, "A convnet for the 2020s," *CVPR*, 2022.