A Security Analysis of CNN Partitioning Strategies for Distributed Inference at the Edge

Fatemeh Mehrafrooz[1(✉)], Roozbeh Siyadatzadeh[1], Nele Mentens[1,2], and Todor Stefanov[1]

[1] Leiden Institute of Advanced Computer Science (LIACS), Leiden University, Leiden, The Netherlands
f.mehrafrooz.mayvan@liacs.leidenuniv.nl
[2] Department Electrical Engineering Department (ESAT), KU Leuven, Leuven, Belgium

Abstract. The inference of Convolutional Neural Networks (CNNs) at the Edge poses significant challenges due to resource limitations of edge devices. One approach to addressing these challenges is to distribute a CNN model across multiple edge devices. While much attention has been paid to improving the performance, memory utilization, energy efficiency, and robustness of distributed CNN inference at the Edge, security implications of such inference remain largely unexplored. Therefore, in this paper, we investigate the security vulnerabilities of the three main partitioning strategies for distributing CNN models across multiple edge devices, namely vertical partitioning, horizontal partitioning, and data partitioning. More specifically, we assess how accurately an attacker can reconstruct the input image given to a CNN model and predict the image class by eavesdropping on the communication link between two edge devices. We devise a simple, yet realistic attack scenario in which the attacker attempts to reconstruct the input image from intermediate data obtained from the communication link. In order to evaluate the vulnerability of the system, the reconstructed image is fed back into the model to see if its class can be determined. We conduct extensive experiments using different CNN models and datasets. Our results show that data partitioning is less vulnerable to this attack scenario compared to the other partitioning strategies, while vertical partitioning is the most vulnerable.

Keywords: Distributed Edge Computing · Neural Networks · Security

1 Introduction

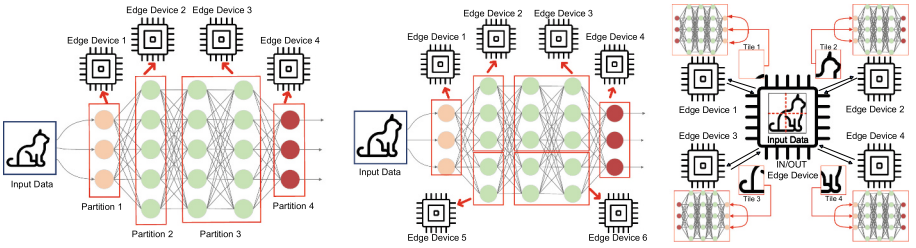
Deep Neural Networks (DNNs) play a crucial role in our daily lives and are used in various applications like healthcare, smart home, and autonomous vehicles. One of the most popular type of DNN models are Convolutional Neural Networks

C. Karfa et al. (Eds.): SPACE 2025, LNCS 16406, pp. 262–281, 2026.
https://doi.org/10.1007/978-3-032-16342-4_15

(CNNs), thanks to their effectiveness in image recognition and classification tasks. Typically, large CNN models are computationally intensive and require a significant amount of memory for storing weights and biases to perform inference. Cloud servers usually have the computational power and sufficient memory to handle the inference of such CNN models. However, for some modern machine learning-based applications like autonomous vehicles, face recognition, and voice assistance, large CNN models need to be deployed on resource-constrained edge devices. This is because edge devices, unlike cloud servers, bring computing power closer to data sources, thereby significantly reducing data communication delays and making such devices suitable for running time-sensitive tasks in the aforementioned applications.

The deployment of CNNs on edge devices poses a significant challenge due to the inherent limitations in computational resources and memory capacity of these devices [1]. One approach to overcome this challenge is to use lightweight CNN models, i.e., a compressed version of larger CNN models, obtained by pruning [2], quantization [3], and knowledge distillation [4] techniques. Although lightweight models are more resource-efficient, they often suffer from reduced accuracy [5]. Another approach is to distribute a CNN model by running a part of it on an edge device and the rest on a cloud server [6]. Such edge-cloud distribution approach, although effective in overcoming resource limitations of edge devices, introduces increased latency and potential security vulnerabilities, as some data has to be transmitted over a communication network to cloud servers for processing [7].

An alternative approach to tackling the aforementioned challenge involves distributing entirely a CNN model across multiple edge devices [8]. Such an approach uses the collective processing power and resources of several edge devices while it maintains the original CNN model accuracy instead of using a lightweight version of the CNN model. Additionally, this approach mitigates the delay issue associated with the edge-cloud distribution approach.



(a) Vertical Partitioning (b) Horizontal Partitioning (c) Data Partitioning

Fig. 1. Partitioning strategies for distributing a CNN model across multiple edge devices.

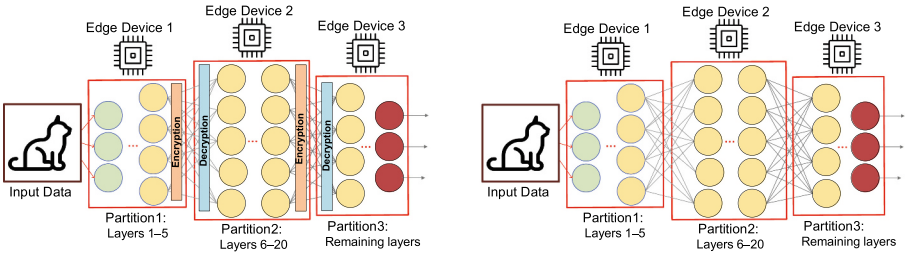
Entirely distributing a CNN model for inference on multiple edge devices requires to split the model into several partitions and map each model parti-

tion onto an edge device. There exist three main strategies for splitting a CNN model into partitions: *vertical partitioning* [9], *horizontal partitioning* [10], and *data partitioning* [7]. In vertical partitioning, a layer or a group of CNN layers comprises a partition as illustrated in Fig. 1(a). Horizontal partitioning involves splitting every CNN layer into segments, where a segment or a group of segments from different CNN layers comprises a partition (Fig. 1(b)). In data partitioning, the input data to a CNN model is split into tiles, and every tile together with a copy of the whole CNN model comprises a partition (Fig. 1(c)).

Several studies have been conducted to examine the overall system performance [7–14], memory requirements [7–9,11,13,15], energy consumption [8,11,16], and robustness [17] when distributing CNN models at the edge using vertical, horizontal and data partitioning strategies. However, to the best of our knowledge, there are no research studies addressing security implications when these different partitioning strategies are used to distribute CNNs. Studying such implications is important because CNN partitions run on different edge devices and these devices need to communicate data in order to collaboratively execute the CNN model. Typically, the communication links between resource-constrained edge devices, running the CNN partitions, are not protected by data encryption. This is because the very large time and energy overhead due to running cryptographic algorithms on such devices is unacceptable. The unprotected communication links increase the vulnerability of the system. For example, an attacker might eavesdrop on the communication link between two edge devices and obtain intermediate data in transit. One action the attacker might take with this intermediate data is to attempt reconstructing the input data provided to the CNN model for processing. In certain scenarios, such as sensitive healthcare or virtual assistants applications, reconstructing the input data can compromise users' privacy.

Therefore, in this paper, we investigate and examine how each strategy for partitioning a CNN model across edge devices affects the vulnerability of the system to attacks attempting to reconstruct the CNN input data from intermediate data communicated between two edge devices. To assess the vulnerability, we devise and utilize a machine learning method to reconstruct, as much as possible, the CNN input data from intermediate data exchanged between layers in the CNN model. Then, to examine the vulnerability, we classify this reconstructed input data with the same CNN model to determine whether the inferred class is the same as the class inferred by utilizing the original input data. Our main novel contributions can be summarized as follows:

- To the best of our knowledge, this is the first paper to examine the security aspect of distributed CNN inference across multiple edge devices. We analyze the three main existing strategies for distributing a CNN model, namely the vertical, horizontal and data partitioning strategies. Our analysis provides important insights from a security point of view and offers guidance for selecting the most appropriate partitioning strategy based on specific application requirements with security in mind.



(a) MobileNet with Encryption–Decryption (b) MobileNet without Encryption–Decryption

Fig. 2. Illustration of two scenarios for distributing a CNN model vertically across three edge devices with and without encryption–decryption.

- We propose a machine learning based attack scenario in which an attacker can obtain intermediate data exchanged between layers in the CNN model by eavesdropping on the communication between two edge devices. Using machine learning, the attacker attempts to reconstruct the input image from this intermediate data and then classify the reconstructed input. Through this approach, we aim to examine the vulnerability of the three partitioning strategies.
- We evaluate the three partitioning strategies using different CNN models and datasets in terms of the aforementioned vulnerability. Our findings indicate that the data partitioning strategy is significantly less vulnerable compared to the vertical and horizontal partitioning strategies.

The remainder of this paper is organized as follows. In Sect. 2, we motivate, in more detail, why unsecured communication links between resource-constrained edge devices are typically utilized. Section 3 provides a brief overview of related studies on distributed CNN inference across edge devices. In Sect. 4, we provide a detailed description of our method for analysis and evaluation of the vulnerability of the existing CNN partitioning strategies. Section 5 presents and discusses the evaluation results. Finally, the paper is concluded in Sect. 6.

2 Motivation

In this section, we give a quantitative example of the overhead in terms of delay and performance of a distributed CNN system in which the communicated data are encrypted and authenticated. This serves as a motivation why a network with secured communication links is often not practical and why unsecured communication links between resource-constrained edge devices are typically utilized when a trained CNN model is distributed across multiple edge devices for inference. As mentioned earlier, in such case, an attacker with non-authorized access to a communication link between two edge devices can eavesdrop on transmitted data. This allows the attacker to obtain intermediate data, which is closely

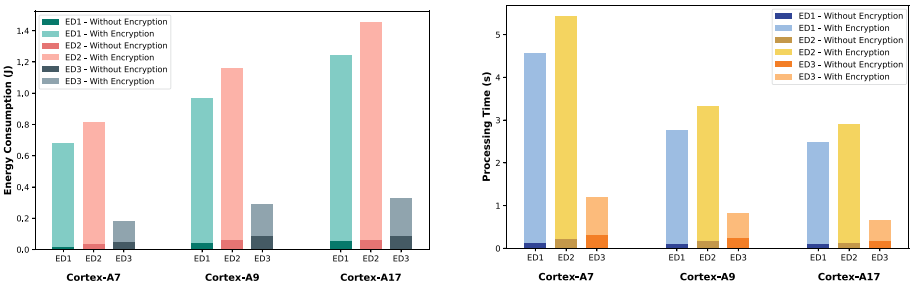
related to the private and potentially sensitive input data being processed by the CNN model.

Although authentication encryption can protect the transmitted intermediate data against potential attacks, deploying even the most lightweight existing cryptographic algorithms on resource-constrained edge devices presents significant challenges. These devices often have limited computational resources and energy budgets, and they are commonly employed in time-sensitive applications where rapid and efficient processing is crucial. Therefore, in some cases, the large time and energy overhead caused by authentication encryption and decryption is not practically acceptable for applications involving distributed CNN inference on resource-constrained edge devices. To support this statement, we present and compare two scenarios that reveal the large time and energy overhead caused by deploying a lightweight cryptographic algorithm to protect data communication links between such devices.

Figure 2 illustrates these two scenarios where in one of the scenarios authenticated encryption-decryption algorithm is applied to protect the data transmitted between edge devices. In both scenarios, we consider the MobileNet CNN model which is distributed by vertical partitioning across three edge devices. In Fig. 2(a), Edge Device 1 receives the input data and processes it up to Layer 5. The output of Layer 5 is encrypted using the ASCON authenticated encryption algorithm, which is currently regarded as the most lightweight cryptographic algorithm, offering state-of-the-art efficiency and security for resource-constrained devices [18]. After encryption, the authenticated and encrypted data is transmitted to Edge Device 2. This device decrypts the data and verifies the authenticity using ASCON, processes it up to Layer 20, authenticates and encrypts the results, and sends the result to Edge Device 3. This device decrypts and verifies the received data and completes the remaining layers to produce the final classification result. For comparison, Fig. 2(b) shows the same CNN model and vertical partitioning across three edge devices without applying any encryption-decryption on the data transmitted between the devices.

Figure 3 shows the processing time (Fig. 3(b)) [19] and energy consumption (Fig. 3(a)) per edge device for the two scenarios mentioned above. The processing time and energy consumption are evaluated for edge devices utilizing three different ARM microprocessors: Cortex-A7 [20], Cortex-A9 [21], and Cortex-A17 [22]. The x-axis indicates the three edge devices (ED1, ED2, and ED3) together with the microprocessor within each device. The bars represent the corresponding metric, i.e., the processing time (in seconds) in Fig. 3(b) and the energy consumption (in joules) in Fig. 3(a). Each bar is divided into two sections where the dark-colored section corresponds to the scenario without data encryption-decryption and the bright-colored section corresponds to the scenario with data encryption-decryption.

For example, in Fig. 3(b), the dark-blue and bright-blue bar sections denote the processing time for Edge Device 1 without and with data encryption, respectively. Similarly, in Fig. 3(a), the dark-green and bright-green sections indicate



(a) Energy consumption

(b) Processing time

Fig. 3. Comparison of processing time and energy consumption for edge devices with and without encryption–decryption.

the energy consumption for Edge Device 1 without and with data encryption, respectively.

The results, shown in Fig. 3, clearly demonstrate that applying even the most lightweight cryptographic algorithm (ASCON) on resource-constrained edge devices, when running a partitioned CNN model, imposes significant time and energy overheads. As can be seen in Fig. 3, these overheads range from at least 3 times up to as much as 36 times, significantly affecting both the processing time and energy consumption. For example, looking at the results for device ED1 with Cortex-A7 in Fig. 3(b), the processing time without data encryption (the dark-blue bar) is approximately 0.122 s, whereas applying data encryption raises this value to 4.432 s (the bright-blue bar), which is an increase of nearly 36.3 times. Similar excessive overhead but in terms of energy consumption can be seen for device ED2 with Cortex-A17 in Fig. 3(a). Such significant/excessive overheads are generally not acceptable for low-power, time-sensitive applications, where rapid inference and efficient energy usage are crucial. Consequently, to maintain acceptable performance levels, many system implementations rely on unsecured communication links between resource-constrained edge devices, thereby leaving data transmissions vulnerable to potential adversaries.

Based on these findings, the following sections present a specific attack scenario aimed at reconstructing input data from unprotected intermediate data transmitted between edge devices running a partitioned CNN model for inference. Our objective is to use this attack scenario in order to analyze how different CNN partitioning strategies for distributed CNN inference are vulnerable to such a scenario as well as to offer guidance for selecting the most appropriate partitioning strategy that inherently enhances the security of distributed CNN inference against the aforementioned attack scenario.

3 Related Work

In this section, we focus on related studies that cover three relevant aspects, namely distributed CNN inference on multiple edge devices, machine learning for data reconstruction and security attacks in distributed CNN models.

3.1 Distributing CNN Inference Across Multiple Edge Devices

Numerous research studies have proposed approaches to distribute and optimize CNN model inference across multiple edge devices to improve the overall system performance [7–14], to reduce memory requirements [7–9,11,13,15] and energy consumption [8,11,16], and to increase the system robustness [17]. For example, MoDNN [14] and DeepThings [7] improve CNN inference on mobile and IoT devices by leveraging parallel execution and efficient task distribution methods. MoDNN uses partitioning schemes to balance the workload and optimize data delivery, while DeepThings employs a novel partitioning and layer fusion method, called Fused Tile Partitioning (FTP), and a distributed work-stealing runtime to reduce the CNN memory footprint and communication overhead between CNN partitions. CoEdge [13] focuses on optimizing memory, computation, and communication. It dynamically partitions workloads based on device capabilities and network conditions, thereby optimizing workload allocation without altering DNN structures. AutoDiCE [8,11] and RobustDiCE [17] introduce frameworks for automated multi-objective exploration and implementation of distributed CNN inference at the edge by focusing on system performance, memory, energy, and robustness optimizations.

The main focus of the aforementioned studies has been on optimizing objectives such as performance, memory, energy, and robustness when distributing CNN models across multiple edge devices. However, security implications have not been addressed in these studies. In contrast, our work focuses on examining a specific security aspect, namely eavesdropping attacks on the communication between edge devices. By analyzing the three main existing strategies for distributing a CNN model, namely the vertical, horizontal, and data partitioning strategies, we provide important insights from a security point of view for selecting the most resilient partitioning strategy against such attacks. Thus, our work significantly complements the aforementioned studies.

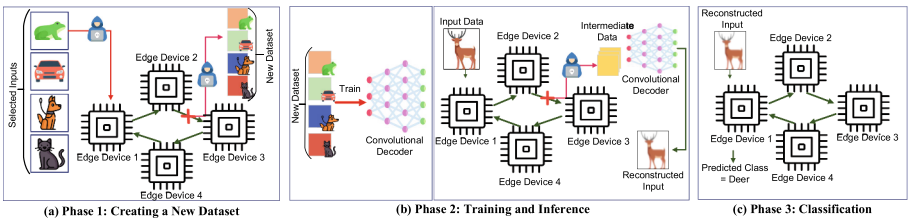


Fig. 4. Overall workflow of our simple attack to evaluate CNN partitioning strategies in terms of security.

3.2 Data Reconstruction Using Machine Learning

Several studies introduce deep learning approaches for reconstructing images from noisy or incomplete data using CNNs. One approach [23] leverages a CNN within an iterative framework to reduce artifacts and improve image quality, outperforming traditional methods. Another method [24], called RPGD, iteratively refines intermediate data through gradient descent and CNN projections to achieve high-quality image reconstruction. In [25], an approach utilizing deep CNNs performs missing data reconstruction tasks in remote sensing, such as dead lines and cloud removal, by integrating spatial, temporal, and spectral information.

In the domain of medical imaging, iCT-Net [26] employs CNNs to refine initial CT reconstructions from specific kernels, enhancing accuracy across various conditions. Another approach [27] uses DNNs to processes limited view projection data to produce high-quality CT images, addressing challenges in applications such as CT scans. A cascaded architecture of multiple CNNs is utilized in [28] to reconstruct ultrasound images, correcting artifacts and enhancing resolution. In [29], an autoencoder processes noisy Monte Carlo rendered images to produce high-quality, stable image sequences.

Beyond image reconstruction, deep learning techniques have also been applied to audio restoration. For example, [30] demonstrates the use of deep learning to restore audio files affected by background noise and signal loss. Another paper [31] proposes a novel method for reconstructing lost audio signals by combining steganography, halftoning, and deep learning models like LSTM and Random Forest.

As described above, many studies utilize machine learning techniques to reconstruct data from noisy or incomplete sources. Inspired by the strong performance of the aforementioned approaches, we devise and apply a specific approach targeting a CNN input data reconstruction from intermediate data processed within a CNN model and obtained in an attack scenario. Our approach involves training and using a convolutional decoder network to reconstruct the input given to the CNN model. The model is distributed across multiple edge devices and its input is reconstructed from intermediate data, exchanged between CNN layers mapped on different devices, in order to evaluate the vulnerability of different partitioning strategies used to distribute the CNN model.

3.3 Security Attacks in Distributed CNN Models

Recent studies have highlighted vulnerabilities in split learning frameworks, where neural networks are partitioned between clients and servers to preserve data privacy. These works demonstrate that intermediate representations exchanged during training or inference can leak sensitive information. Pasquini et al. [32] introduce the Feature-Space Hijacking Attack (FSHA), showing that a malicious server can reconstruct clients' private training data by manipulating the learning process. Erdoğan et al. [33] present UnSplit, a suite of attacks including model inversion, model stealing, and label inference, demonstrating

that even with limited knowledge, a server can recover input data and model parameters. Liu et al. [34] propose similarity-based label inference attacks that exploit gradients and intermediate representations to infer private labels during both training and inference phases. Yu et al. [35] develop SIA, a sustainable inference attack framework that reconstructs client data while evading detection mechanisms.

While the aforementioned studies focus on split learning scenarios, our work examines the security implications of distributing CNN inference across multiple edge devices. Specifically, we analyze the vulnerability of different partitioning strategies (vertical, horizontal, and data partitioning) to eavesdropping attacks by reconstructing input data from intermediate representations exchanged between devices. Our approach provides insights into selecting partitioning strategies that enhance resilience against such attacks, thereby complementing the findings of the aforementioned studies.

4 Our Method for Vulnerability Analysis

In this section, we sketch a realistic machine learning based attack scenario which we use as a method for evaluating the three main CNN partitioning strategies, introduced in Sect. 1 and illustrated in Fig. 1, with respect to security. First, we assume that a trained CNN model is distributed across multiple edge devices for inference and the model inference is provided as a service to users. By booking a time slot, any user can exclusively reserve the service to classify own private input data. Thus, an attacker, acting as a legitimate user, can input data into the distributed CNN model for classification. Second, we assume that the attacker has access to a communication link between two edge devices and can eavesdrop on the communicated data. By accessing the communication link, the attacker can obtain intermediate data related to private input data provided by any user and currently processed by the CNN model.

The aforementioned assumptions are realistic in many practical edge computing setups. For example, edge devices might communicate over a shared but physically segmented (wireless) network. Suppose some devices use local 5G connectivity while others use Wi-Fi or a dedicated Ethernet channel. An attacker physically close to a Wi-Fi router (or using a compromised intermediate network device) may be able to eavesdrop on one link, such as the communication between Device 1 and Device 2, without having access to links involving other devices that are routed through separate, isolated network paths or encrypted channels. This partial access could result from exploiting a vulnerability in a specific router or switch, from being in range of an unencrypted wireless transmission, or from compromising a device connected to a specific network segment. Importantly, we assume that it is not possible for the attacker to access all communication links due to network segmentation, physical isolation of devices, and the possible use of secure protocols on some links.

A third assumption is that the attacker does not have access to the internals of any edge device, and does not know the structure of the CNN model,

its architecture, parameters, or layer mappings. Their only access point is the intercepted intermediate data flowing through a single link between two edge devices. This reflects a black-box, passive attacker model in which the attacker's objective is to reconstruct private input data from intermediate data and infer the corresponding label.

Considering the above realistic assumptions, we can setup and perform an attack, which utilizes and trains a convolutional decoder network [36]. The purpose of this convolutional decoder is to reconstruct input data provided to the distributed CNN model from intermediate data communicated between two edge devices. Figure 4 shows the overall workflow of our simple attack, which consists of three phases. Phase 1 involves the creation of a dataset that is used to train the convolutional decoder. Phase 2 focuses on the training of the convolutional decoder with the dataset created in Phase 1 and uses the trained decoder to reconstruct the input from intermediate data. Phase 3 determines the class of the reconstructed input obtained in Phase 2. In the rest of this section, we explain the three phases of our attack approach and how we use it to evaluate the vulnerability of distributed CNN inference in more detail.

4.1 Phase 1: Creating a New Dataset

As mentioned above, the goal of this phase is to generate a dataset D used for training the convolutional decoder. This dataset is based on intermediate data that the attacker can obtain from the communication link between two edge devices. Since the attacker has access to the distributed CNN model by acting as a legitimate user, the attacker can input any data into the model for inference. Therefore, the process of generating the training dataset D begins by giving the CNN model a set of selected input data samples $I = \{x_1, x_2, ..., x_n\}$ illustrated by the Frog, Car, Dog, and Cat images in Fig. 4a. For each selected input sample $x_i \in I$, the attacker collects the corresponding intermediate data z_i exchanged between two edge devices via the eavesdropped communication link, e.g., the link between Edge Device 2 and 3 in Fig. 4a.

Since the attacker does not know the internal structure or the mapping of layers of the distributed CNN model, the z_i is collected as a bitstream and stored into a vector shape. Then, the attacker creates a pair (z_i, x_i) and stores it as a new labeled data sample in dataset D where z_i is the new training data sample and x_i is its label. By repeating the above steps for every $x_i \in I$, the new training dataset $D = \{(z_1, x_1), (z_2, x_2), \ldots, (z_n, x_n)\}$ is created.

4.2 Phase 2: Training and Inference

The goal of this phase is to prepare and perform the attack using dataset D created in Phase 1. Phase 2 consists of two steps: training and inference. In the training step, illustrated in the left part of Fig. 4b, any available training algorithm [37] can be used to train a convolutional decoder with dataset D. Based on the length and complexity of the intermediate data samples, the attacker designs

a decoder to reconstruct the original input. The attacker can iteratively evaluate the output of the decoder and modify its structure if necessary to improve reconstruction quality.

Once the convolutional decoder is trained, it is ready to reconstruct original input data samples from intermediate data obtained by eavesdropping on the communication link between two edge devices. The inference step, illustrated in the right part of Fig. 4b, is the actual attack where the user provides an input data sample to the distributed CNN model and the attacker collects the corresponding intermediate data. Then, this intermediate data is given as input to the convolutional decoder which outputs the reconstructed input data sample.

4.3 Phase 3: Classification

In the third phase, illustrated in Fig. 4c, the goal is to determine the class of the data reconstructed in Phase 2. The attacker can use the distributed CNN model as a legitimate user for this purpose. Therefore, the reconstructed data is given to the distributed model as input, and the model predicts the class of the reconstructed data.

We evaluate the vulnerability of the distributed CNN inference by performing the attack, described above and illustrated in Fig. 4, with the following modifications. In Phase 2, we play the role of the user and the attacker, i.e., we provide user input data to the distributed CNN model instead of the user, and we collect the corresponding intermediate data and obtain reconstructed input data as the attacker does. In addition, we use Phase 3 to assess whether the attack is successful or not by classifying the reconstructed input data using the distributed CNN model for inference. If the inferred class is the same as the class inferred by utilizing the user input data, then the attack is successful, indicating a vulnerability.

Table 1. Characteristics of the Datasets

Dataset	Number of Images	Image Size	Categories
MNIST	70,000	$28 \times 28 \times 1$	10
CIFAR-10	60,000	$32 \times 32 \times 3$	10
Oxford Flowers	8,189	Variable	102

5 Experimental Evaluation

In this section, we examine the vulnerability of the three main partitioning strategies (Fig. 1) used to distribute a CNN model across multiple edge devices. The goal of this examination is to demonstrate and compare the vulnerabilities of these partitioning strategies to the attack scenario described in Sect. 4. First, we explain the experimental setup in Sect. 5.1. Then, in Sect. 5.2, we present the experimental results and discuss their implications with respect to the vulnerability of the different partitioning strategies.

5.1 Experimental Setup

In this study, we investigate the vulnerability by distributing three distinct CNN models across multiple edge devices, namely LeNet-5 [38], MobileNet [39], and ResNet-50 [40]. These models are selected because of their varied complexity and characteristics, making them ideal for representative evaluation of vulnerabilities when distributing CNN models. LeNet-5 represents CNNs with a small and simple architecture. MobileNet is well known for its efficiency on mobile devices. ResNet-50 is a representative of deep and more complex CNN models.

To train the above CNN models, three datasets are used: MNIST for LeNet-5; CIFAR-10 for LeNet-5, MobileNet and ResNet-50; and Oxford Flowers- for MobileNet and ResNet-50. Table 1 characterizes these datasets by three attributes. For every dataset, the attribute `Number of Images` shows the total image count in the dataset, `Image Size` indicates the dimensions of the images in the dataset, and `Categories` shows the number of classes within the dataset. MNIST consists of grayscale images of handwritten digits, CIFAR-10 includes color images, and the Oxford Flowers dataset contains high-resolution color images. The described datasets vary in the input image size, type, and number of classes, thereby ensuring a robust and trustworthy evaluation. Each dataset is split as follows: 80% of the images are used for training, 10% for validation, and 10% for testing. The CNN models are trained for 10 epochs on each dataset.

To evaluate and compare the vulnerability of the three partitioning strategies, introduced in Sect. 1 and illustrated in Fig. 1, we partition each trained CNN model, using the three strategies, and then deploy the model on a distributed system with multiple edge devices for inference, using Python and the TensorFlow Keras library. After the distributed deployment of a CNN model, we collect intermediate data from the output of specific layers. This data is used to create dataset D as described in Sect. 4.1. Any compatible input data can be used for creating dataset D. Therefore, we use a part of the ImageNet dataset [41], consisting of 40,000 image samples, as the set of input data samples I. These samples are unseen during the training process of the distributed CNN models. Each sample $x_i \in I$ is processed by a distributed CNN model, and the corresponding intermediate data z_i is collected during this processing.

Then, we build a convolutional decoder network to reconstruct the original input of the distributed CNN model using the intermediate data, as described in Sect. 4.2. Table 2 provides a detailed overview of the decoder architectures used for reconstructing the original input from different intermediate representations. Each row in the table corresponds to a specific combination of a model and a layer index, characterized by the input feature map size at that point in the network. Layer 1 in each model is used specifically for our data partitioning strategy. In this approach, we divide the original input image into four equal parts and assign each part to the decoder. As a result, the input size shown in the table for Layer 1 reflects the size of each partition.

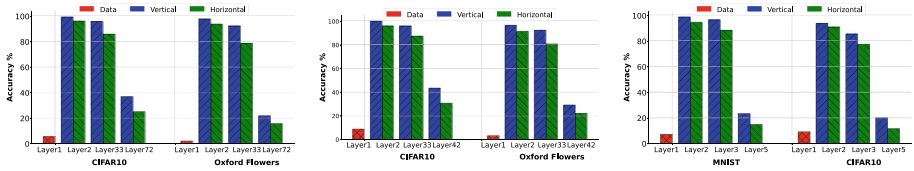
In this table, the `Vector Size` column shows the length of the intermediate data received by the attacker. This vector is reshaped into a set of 2D matrices to enable the application of convolutional operations for reconstruction. As

Table 2. Decoder structures used to reconstruct the original input from intermediate data of different models and layers

Model	Layer	Vector Size	Input Size	Output Size	Decoder Structure
MobileNet	Layer 1	37632	(112,112,3)	(224,224,3)	$\text{Up}(224) \rightarrow \text{Conv}(3,16,3) \rightarrow \text{ReLU} \rightarrow \text{Conv}(16,3,3)$
	Layer 2	401408	(112,112,32)	(224,224,3)	$\text{Up}(224) \rightarrow \text{Conv}(32,64,3) \rightarrow \text{ReLU} \rightarrow \text{Conv}(64,32,3) \rightarrow \text{ReLU} \rightarrow \text{Conv}(32,3,3)$
	Layer 33	200704	(28,28,256)	(224,224,3)	$\text{ConvT}(256,128,3,2) \rightarrow \text{ReLU} \rightarrow \text{ConvT}(128,64,3,2) \rightarrow \text{ReLU} \rightarrow \text{ConvT}(64,32,3,2) \rightarrow \text{ReLU} \rightarrow \text{Conv}(32,3,3)$
	Layer 72	100352	(14,14,512)	(224,224,3)	$\text{ConvT}(512,256,3,2) \rightarrow \text{ReLU} \rightarrow \text{ConvT}(256,128,3,2) \rightarrow \text{ReLU} \rightarrow \text{ConvT}(128,64,3,2) \rightarrow \text{ReLU} \rightarrow \text{ConvT}(64,32,3,2) \rightarrow \text{ReLU} \rightarrow \text{Conv}(32,3,3)$
ResNet-50	Layer 1	37632	(112,112,3)	(224,224,3)	same as MobileNet Layer 1
	Layer 2	802816	(112,112,64)	(224,224,3)	$\text{Up}(224) \rightarrow \text{Conv}(64,32,3) \rightarrow \text{ReLU} \rightarrow \text{Conv}(32,3,3)$
	Layer 33	200704	(56,56,64)	(224,224,3)	$\text{ConvT}(64,32,3,2) \rightarrow \text{ReLU} \rightarrow \text{ConvT}(32,16,3,2) \rightarrow \text{ReLU} \rightarrow \text{Conv}(16,3,3)$
	Layer 42	100352	(28,28,128)	(224,224,3)	$\text{ConvT}(128,64,3,2) \rightarrow \text{ReLU} \rightarrow \text{ConvT}(64,32,3,2) \rightarrow \text{ReLU} \rightarrow \text{ConvT}(32,16,3,2) \rightarrow \text{ReLU} \rightarrow \text{Conv}(16,3,3)$
LeNet-5 (CIFAR-10)	Layer 1	768	(16,16,3)	(32,32,3)	$\text{Up}(32) \rightarrow \text{Conv}(3,16,3) \rightarrow \text{ReLU} \rightarrow \text{Conv}(16,3,3)$
	Layer 2	2304	(12,12,16)	(32,32,3)	$\text{Up}(32) \rightarrow \text{Conv}(16,8,3) \rightarrow \text{ReLU} \rightarrow \text{Conv}(8,3,3)$
	Layer 3	576	(6,6,16)	(32,32,3)	$\text{ConvT}(16,16,3,2) \rightarrow \text{ReLU} \rightarrow \text{Up}(32) \rightarrow \text{Conv}(16,8,3) \rightarrow \text{ReLU} \rightarrow \text{Conv}(8,3,3)$
	Layer 5	480	(2,2,120)	(32,32,3)	$\text{ConvT}(120,60,3,2) \rightarrow \text{ReLU} \rightarrow \text{ConvT}(60,30,3,2) \rightarrow \text{ReLU} \rightarrow \text{ConvT}(30,15,3,2) \rightarrow \text{ReLU} \rightarrow \text{Conv}(8,3,3)$
LeNet-5 (MNIST)	Layer 1	196	(14,14,1)	(28,28,1)	$\text{Up}(28) \rightarrow \text{Conv}(1,8,3) \rightarrow \text{ReLU} \rightarrow \text{Conv}(8,1,3)$
	Layer 2	1600	(10,10,16)	(28,28,1)	$\text{Up}(28) \rightarrow \text{Conv}(16,8,3) \rightarrow \text{ReLU} \rightarrow \text{Conv}(8,1,3)$
	Layer 3	400	(5,5,16)	(28,28,1)	$\text{ConvT}(16,16,3,2) \rightarrow \text{ReLU} \rightarrow \text{Up}(28) \rightarrow \text{Conv}(16,8,3) \rightarrow \text{ReLU} \rightarrow \text{Conv}(8,1,3)$
	Layer 5	120	(1,1,120)	(28,28,1)	$\text{ConvT}(120,64,4,2,1) \rightarrow \text{ReLU} \rightarrow \text{ConvT}(64,32,3,2) \rightarrow \text{ReLU} \rightarrow \text{ConvT}(32,16,3,2) \rightarrow \text{ReLU} \rightarrow \text{ConvT}(16,8,3,2) \rightarrow \text{ReLU} \rightarrow \text{Up}(28) \rightarrow \text{Conv}(8,1,3)$

explained in Sect. 4.2, the attacker may adjust both the decoder architecture and the input shape based on the vector length and complexity in order to improve the reconstruction quality. The `Input Size` column shows the size of the feature map that is fed into the decoder while the `Output Size` column displays the size of the output produced by the decoder. The `Decoder Structure` column describes the sequence of operations used by the decoder, which typically include upsampling (Up(n)), transposed convolutions (ConvT), standard convolutions (Conv), and ReLU activation functions. For example, in Layer 33 of MobileNet, the decoder receives a feature map of shape (28, 28, 256) and transforms it to an output of shape (224, 224, 3). This transformation is performed through a series of ConvT, each followed by a ReLU activation. The first ConvT layer reduces the number of channels from 256 to 128 and increases the spatial dimensions using a 3×3 kernel and a stride of 2. This is followed by a ReLU and a second ConvT layer that reduces the channels to 64 using the same kernel and stride. A third ConvT layer further reduces the channels to 32. Finally, a standard 3×3 convolution maps the 32-channel output to 3 channels, yielding the final image-shaped output. The decoder is trained for 10 epochs on the dataset D using the Adam optimizer [42] for gradient descent optimization.

Finally, for every distributed CNN model, partitioned using the data, vertical, and horizontal partitioning strategies, the vulnerability of every strategy



(a) MobileNet Model (b) ResNet-50 Model (c) LeNet-5 Model

Fig. 5. Comparison of the Data, Vertical, and Horizontal strategies across different CNN models, datasets, and layers.

is assessed by calculating the accuracy of the reconstructed input images generated by the convolutional decoder. To calculate this accuracy, every image in the aforementioned 10% test data from a dataset in Table 1 is fed into the CNN model for processing, and the predicted class of this original input image is recorded. During the processing, intermediate data produced by a specific layer in the CNN model is collected and given as input to the trained convolutional decoder for reconstruction. The output of the decoder, i.e., the reconstructed image, is then fed back into the CNN model, as explained in Sect. 4.3, and the predicted class is recorded as well. By comparing the predicted class of every reconstructed image with the predicted class of the corresponding original input images, we determine the number of correctly predicted classes of the reconstructed images. The accuracy is calculated by dividing this number by the total number of reconstructed images.

5.2 Experimental Results and Vulnerability Analysis

Figure 5 depicts the results of our experimental evaluation of the three partitioning strategies when they are used to distribute MobileNet, ResNet-50 and LeNet-5 across 4 edge devices, and the attack scenario/method, described in Sect. 4, is applied.

In each bar chart of Fig. 5, the Y-axis shows the accuracy, while the X-axis shows the dataset used to calculate the accuracy as well as the specific layers of the CNN model used to collect the intermediate data. The blue and green bars correspond to the vertical and horizontal partitioning strategies, respectively. The red bars correspond to the data partitioning strategy. For this strategy, we consider only Layer 1 because an attacker, with access to the communication link between two edge devices, can only collect a part of the input image transferred from the IN/OUT edge device to the other device as shown in Fig. 1(c). Therefore, the attacker is limited to the partial input data given to Layer 1 and does not have access to data processed by other layers.

Figure 5(a) plots the results for the MobileNet model, showing the accuracy of the three partitioning strategies for the CIFAR-10 and Oxford Flowers datasets, where each data sample has a size of $224 \times 224 \times 3$. For the vertical and horizontal partitioning, intermediate data is collected at the 2nd, 33rd, and 72nd layers. Similarly, Fig. 5(b) showcases the results for the ResNet-50 model, highlighting

the accuracy differences among the three partitioning strategies for the CIFAR-10 and Oxford Flowers datasets, with a data sample size of $224 \times 224 \times 3$, and intermediate data collected at the 2nd, 33rd, and 42nd layer. Figure 5(c) shows the results for the LeNet-5 model, using the MNIST dataset with a data sample shape of $28 \times 28 \times 1$ and the CIFAR-10 dataset with a data sample size of $32 \times 32 \times 3$. For the vertical and horizontal partitioning strategies, intermediate data is collected at the 2nd, 3rd, and 5th layer.

5.2.1 Partitioning Strategies

First, we analyze how the different CNN partitioning strategies, used for distributed CNN inference across edge devices, affect the input data reconstruction accuracy under our attack scenario. The choice of a partitioning strategy directly influences the amount and nature of the intermediate data available to the attacker, which in turn impacts how effectively the attacker's convolutional decoder can reconstruct the input data. For example, it can be seen in Fig. 5 that if the attacker can collect the output of the second layer (Layer 2), the accuracy is higher in vertical partitioning compared to horizontal and data partitioning. This is because vertical partitioning involves more (intermediate) data exchanged between layers mapped on different devices than the other partitioning strategies. In contrast, horizontal partitioning limits the attacker to only a subset of the data, and data partitioning further restricts the attacker to only a small fraction of the original input. As a result, the information available to the attacker in horizontal and data partitioning scenarios is insufficient to achieve the same level of reconstruction accuracy as in vertical partitioning.

Furthermore, the results in Fig. 5 show that the accuracy in data partitioning is significantly lower compared to the other strategies. This is because, in data partitioning, the input data is divided into smaller parts that are processed separately as illustrated in Fig. 1(c). Consequently, the attacker only gains access to a small part of the input data, making it more difficult for the convolutional decoder to accurately reconstruct the full input data from that small part.

5.2.2 Layer Access

As shown in Fig. 5, the positioning of the accessed layer within the CNN's topology greatly influences the attacker's reconstruction capabilities. In general, when an attacker has access to the output of a higher layer, i.e., a layer that is far away from the input layer in the CNN model topology, the accuracy of the reconstructed input decreases. This trend means that the accuracy is higher when the attacker accesses earlier layers, i.e., layers closer to the input layer in the CNN model topology. This is because earlier layers capture more generalized, low-level features, such as edges and textures, that are more directly linked to the original input data. In contrast, higher layers contain more refined, abstract representations that are crucial for the model's decision-making process but are less directly tied to the original input data. These abstract representations are more challenging to reverse-engineer into the original input data. Consequently, the convolutional decoder requires more detailed features to accurately reconstruct

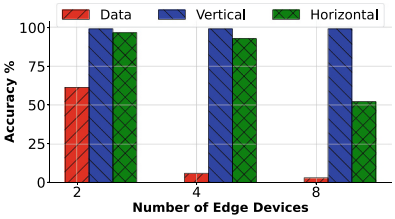


Fig. 6. Comparison of partitioning strategies with respect to the number of devices used to distribute the MobileNet model, based on the CIFAR-10 dataset.

the input data from the intermediate data, that are more readily available in earlier layers.

5.2.3 Dataset Characteristics

The characteristics of the datasets, processed by a partitioned CNN model during inference, play a role in the accuracy of reconstructing input data from intermediate data. One of the important factors that impacts the accuracy is the number of classes in a dataset. For example, in Fig. 5(a) and 5(b), the accuracy for the Oxford Flowers dataset and a given layer is lower compared to the same layer and the CIFAR-10 dataset. This is because reconstructing the input data from intermediate data can introduce errors, and with a larger number of classes, i.e., 102 in the Oxford Flowers dataset, these errors are more likely to occur and cause a wrong class prediction, resulting in lower accuracy. Additionally, the image size plays a role in reconstructing the input image and achieving better accuracy. If the attacker has access to less (intermediate) data due to a small input image size, it becomes more difficult to reconstruct the original input image and correctly predict the class. This can be seen in Fig. 5(c), where the accuracy for Layer 2 in the CIFAR-10 dataset is lower compared to the accuracy for Layer 2 in Fig. 5(a) and 5(b). This difference is due to the smaller image size ($32 \times 32 \times 3$) in CIFAR-10 with LeNet-5, compared to the larger image sizes ($224 \times 224 \times 3$) in CIFAR-10 and Oxford Flowers used with MobileNet and ResNet-50. Furthermore, it is easier to reconstruct MNIST images than CIFAR-10 images, as shown in Fig. 5(c), because MNIST images are simpler and lack the fine details present in CIFAR-10 images.

5.2.4 Number of Edge Devices

Another important factor for the accuracy is the number of edge devices used to distribute a CNN model. A higher number of edge devices means more partitions. Figure 6 compares the accuracy of the three partitioning strategies when the MobileNet model is distributed across 2, 4, and 8 edge devices, with intermediate data collected from Layer 2 for the vertical and horizontal partitioning, and from Layer 1 for the data partitioning. The model is tested with colour images of size $224 \times 224 \times 3$ from the CIFAR-10 dataset. In Fig. 6, the X-axis shows the

number of edge devices, while the red, blue, and green bars show the accuracy for the data partitioning, vertical partitioning, and horizontal partitioning, respectively. The results indicate that the number of edge devices directly impacts the accuracy, respectively the vulnerability of the data and horizontal partitioning strategies, whereas the vulnerability of the vertical partitioning strategy remains unaffected by the number of edge devices. This is because, with an increasing number of edge devices and partitions, the volume and content of the intermediate data, collected at Layer 2 and Layer 1, are changing in the horizontal and data partitioning strategies. In vertical partitioning, the collected intermediate data at Layer 2 remains unchanged. Moreover, the accuracy for the data and horizontal partitioning strategies tends to be higher when fewer edge devices are used. This is because, by using fewer edge devices, the (intermediate) data is partitioned into larger, more complete segments, thereby allowing the convolutional decoder to reconstruct the original input data more effectively. In contrast, when the number of edge devices increases, each device handles a smaller part of the (intermediate) data. This fragmentation makes it more challenging for the convolutional decoder to accurately reconstruct the input, leading to a decrease in accuracy.

6 Conclusions

In this paper, we examine and compare the vulnerability of the three main strategies for partitioning a CNN model across multiple edge devices, i.e., vertical partitioning, horizontal partitioning, and data partitioning. To assess their vulnerability, we devise a realistic attack scenario in which an attacker eavesdrops on the communication link between two edge devices, obtains intermediate data, and attempts to determine the class of the input data. In this scenario, the attacker tries to reconstruct the original CNN input data using a convolutional decoder and then feeds the reconstructed data back into the model to obtain the class. We evaluate the vulnerability of each partitioning strategy by calculating the data reconstruction accuracy, i.e., comparing the predicted class of the reconstructed data with the original class.

The experimental results show that data partitioning is significantly less vulnerable to the aforementioned attack scenario compared to vertical and horizontal partitioning, with vertical partitioning being the most vulnerable strategy. Moreover, the size of the CNN input data affects vulnerability, as larger input data leads to higher reconstruction accuracy. The number of CNN partitions/edge devices also impacts vulnerability, i.e., when the number of partitions/devices increases, the reconstruction accuracy decreases, thereby reducing the vulnerability.

Based on our experiments, results, and analysis, we can conclude that by employing the data partitioning strategy for distributed CNN inference and increasing the number of edge devices (partitions) as much as possible, or combining it with horizontal partitioning, we can inherently enhance the security of distributed CNN inference against the aforementioned attack scenario. Furthermore, as demonstrated in the experimental section, earlier CNN layers generate

more valuable intermediate data for successful reconstruction of the original input data. Therefore, communicating intermediate data between edge devices, generated by these early CNN layers, must be avoided by running these layers on the same device in order to significantly reduce the overall risk of successful input data reconstruction utilizing the aforementioned attack scenario.

Acknowledgments. This work was supported by European Union (NeuroSoC project - Horizon Europe Grant Agreement n°101070634).

References

1. Yang, L., Zheng, C., Shen, X., Xie, G.: OfpCNN: on-demand fine-grained partitioning for CNN inference acceleration in heterogeneous devices. IEEE Trans. Parallel Distrib. Syst. **34**(12), 3090–3103 (2023)
2. Han, S., Pool, J., Tran, J., Dally, W.J.: Learning both weights and connections for efficient neural networks. In: Proceedings of the 29th International Conference on Neural Information Processing Systems, NIPS 2015, vol. 1, pp. 1135–1143. MIT Press, Cambridge (2015)
3. Nagel, M., Fournarakis, M., Amjad, R.A., Bondarenko, Y., Van Baalen, M., Blankevoort, T.: A white paper on neural network quantization. arXiv preprint arXiv:2106.08295 (2021)
4. Hinton, G., Vinyals, O., Dean, J.: Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531 (2015)
5. Prakash, I., Bansal, A., Verma, R., Shorey, R.: Smartsplit: latency-energy-memory optimisation for CNN splitting on smartphone environment. In: 2022 14th International Conference on COMmunication Systems & NETworkS (COMSNETS), pp. 549–557 (2022)
6. Hu, C., Bao, W., Wang, D., Liu, F.: Dynamic adaptive DNN surgery for inference acceleration on the edge. In: IEEE INFOCOM 2019 - IEEE Conference on Computer Communications, pp. 1423–1431 (2019)
7. Zhao, Z., Barijough, K.M., Gerstlauer, A.: Deepthings: distributed adaptive deep learning inference on resource-constrained IoT edge clusters. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **37**(11), 2348–2359 (2018)
8. Guo, X., Pimentel, A.D., Stefanov, T.: Automated exploration and implementation of distributed CNN inference at the edge. IEEE Internet Things J. **10**(7), 5843–5858 (2023)
9. Tang, E., Stefanov, T.: Low-memory and high-performance CNN inference on distributed systems at the edge. In: Proceedings of the 14th IEEE/ACM International Conference on Utility and Cloud Computing Companion, UCC 2021. Association for Computing Machinery, New York (2022)
10. Stahl, R., Zhao, Z., Mueller-Gritschneder, D., Gerstlauer, A., Schlichtmann, U.: Fully distributed deep learning inference on resource-constrained edge devices. In: Pnevmatikatos, D.N., Pelcat, M., Jung, M. (eds.) SAMOS 2019. LNCS, vol. 11733, pp. 77–90. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-27562-4_6
11. Guo, X., Pimentel, A.D., Stefanov, T.: Hierarchical design space exploration for distributed CNN inference at the edge. In: Machine Learning and Principles and Practice of Knowledge Discovery in Databases, pp. 545–556. Springer, Cham (2023)

12. Hou, X., Guan, Y., Han, T., Zhang, N.: Distredge: speeding up convolutional neural network inference on distributed edge devices. In: 2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS), pp. 1097–1107. IEEE Computer Society, Los Alamitos (2022)
13. Zeng, L., Chen, X., Zhou, Z., Yang, L., Zhang, J.: Coedge: cooperative DNN inference with adaptive workload partitioning over heterogeneous edge devices. IEEE/ACM Trans. Netw. **29**(2), 595–608 (2021)
14. Mao, J., Chen, X., Nixon, K.W., Krieger, C., Chen, Y.: MoDNN: local distributed mobile computing system for deep neural network. In: Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 1396–1401 (2017)
15. Zhang, S., Zhang, S., Qian, Z., Wu, J., Jin, Y., Lu, S.: Deepslicing: collaborative and adaptive CNN inference with low latency. IEEE Trans. Parallel Distrib. Syst. **32**(9), 2175–2187 (2021)
16. Tang, E., Guo, X., Stefanov, T.: The effects of partitioning strategies on energy consumption in distributed CNN inference at the edge. arXiv preprint arXiv:2210.08392 (2022)
17. Guo, X., Jiang, Q., Pimentel, A.D., Stefanov, T.: RobustDiCE: robust and distributed CNN inference at the edge. In: Proceedings of the 29th Asia and South Pacific Design Automation Conference, pp. 26–31 (2024)
18. Dobraunig, C., Eichlseder, M., Mendel, F., Schläffer, M.: Ascon v1.2: lightweight authenticated encryption and hashing. In: International Conference on Selected Areas in Cryptography, pp. 84–115. Springer, Cham (2016)
19. Dobraunig, C., Eichlseder, M., Mendel, F., Schläffer, M.: Implementation comparison: crypto_aead/ascon128v1. https://bench.cr.yp.to/impl-aead/ascon128v1.html. Accessed 11 Dec 2024
20. Arm Limited: ARM Cortex-A7 Processor Datasheet. (2025). Accessed 11 Dec 2024. https://developer.arm.com/documentation/ddi0464/latest/ Accessed 11 Feb 2025
21. Arm Limited: ARM Cortex-A9 Processor Datasheet (2025). Accessed 11 Dec 2024. https://developer.arm.com/documentation/ddi0388/latest/. Accessed 11 Feb 2025
22. Arm Limited: ARM Cortex-A17 MPCore Processor Technical Reference Manual (2025). Accessed 11 Dec 2024. https://developer.arm.com/documentation/ddi0535/latest. Accessed 11 Feb 2025
23. Kelly, B., Matthews, T.P., Anastasio, M.A.: Deep learning-guided image reconstruction from incomplete data. arXiv preprint arXiv:1709.00584 (2017)
24. Gupta, H., Jin, K.H., Nguyen, H.Q., McCann, M.T., Unser, M.: CNN-based projected gradient descent for consistent CT image reconstruction. IEEE Trans. Med. Imaging **37**(6), 1440–1453 (2018)
25. Zhang, Q., Yuan, Q., Zeng, C., Li, X., Wei, Y.: Missing data reconstruction in remote sensing image with a unified spatial–temporal–spectral deep convolutional neural network. IEEE Trans. Geosci. Remote Sens. **56**(8), 4274–4288 (2018)
26. Li, Y., Li, K., Zhang, C., Montoya, J., Chen, G.-H.: Learning to reconstruct computed tomography images directly from sinogram data under a variety of data acquisition conditions. IEEE Trans. Med. Imaging **38**(10), 2469–2481 (2019)
27. Kalare, K.W., Bajpai, M.K.: RecDNN: deep neural network for image reconstruction from limited view projection data. Soft. Comput. **24**(22), 17205–17220 (2020). https://doi.org/10.1007/s00500-020-05013-4
28. Wasih, M., Ahmad, S., Almekkawy, M.: A robust cascaded deep neural network for image reconstruction of single plane wave ultrasound RF data. Ultrasonics **132**, 106981 (2023)

29. Chaitanya, C.R.A., et al.: Interactive reconstruction of Monte Carlo image sequences using a recurrent denoising autoencoder. ACM Trans. Graph. (TOG) **36**(4), 1–12 (2017)
30. Nogales, A., Donaher, S., García-Tejedor, A.: A deep learning framework for audio restoration using convolutional/deconvolutional deep autoencoders. Expert Syst. Appl. **230**, 120586 (2023)
31. Cheddad, Z.A., Cheddad, A.: Active restoration of lost audio signals using machine learning and latent information. In: Arai, K. (ed.) Intelligent Systems and Applications, pp. 1–16. Springer, Cham (2024)
32. Pasquini, D., Ateniese, G., Bernaschi, M.: Unleashing the tiger: inference attacks on split learning. In: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security, CCS 2021, pp. 2113–2129. Association for Computing Machinery, New York (2021)
33. Erdoğan, E., Küpçü, A., Çiçek, A.E.: Unsplit: data-oblivious model inversion, model stealing, and label inference attacks against split learning. In: Proceedings of the 21st Workshop on Privacy in the Electronic Society, WPES 2022, pp. 115–124. Association for Computing Machinery, New York (2022)
34. Liu, J., Lyu, X., Cui, Q., Tao, X.: Similarity-based label inference attack against training and inference of split learning. IEEE Trans. Inf. Forensics Secur. **19**, 2881–2895 (2024)
35. Yu, F., Wang, L., Zeng, B., Zhao, K., Wu, T., Pang, Z.: SIA: a sustainable inference attack framework in split learning. Neural Netw. **171**(C), 396–409 (2024)
36. Ji, Y., Zhang, H., Zhang, Z., Liu, M.: CNN-based encoder-decoder networks for salient object detection: a comprehensive review and recent advances. Inf. Sci. **546**, 835–857 (2021)
37. Sun, S., Cao, Z., Zhu, H., Zhao, J.: A survey of optimization methods from a machine learning perspective. IEEE Trans. Cybern. **50**(8), 3668–3681 (2019)
38. Lecun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proc. IEEE **86**(11), 2278–2324 (1998)
39. Howard, A.G., et al.: Mobilenets: efficient convolutional neural networks for mobile vision applications. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2017)
40. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 770–778 (2016)
41. Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., Fei-Fei, L.: Imagenet: a large-scale hierarchical image database. In: 2009 IEEE Conference on Computer Vision and Pattern Recognition, pp. 248–255 (2009)
42. Kingma, D.P.: Adam: a method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)