Creating Robust Deep Neural Networks With Coded Distributed Computing for IoT

IEEE Edge'23
July 2023

Ramyad Hadidi[§]
Rain AI

Jiashen Cao
Georgia Tech

Bahar Asgari[§]
University of Maryland

Hyesoon Kim
Georgia Tech

[§] This work was done when these authors were affiliated with Georgia Tech.

Georgia Tech

Internet of Things (IoT) Devices

Record an abundance of various raw data types and must act in real-time on this data

- Therefore, they must understand it with their limited resources of power and compute

Emergence of DNN on IoT

With deep neural networks (DNNs) IoT devices can

- Process several new data types and
- Understand behaviors

However, DNNs are resource hungry

- Cannot met real-time constraints
- Several DNNs cannot even be executed

Solutions – Cloud/Fog

Offload to cloud/fog, but this is not always a solution

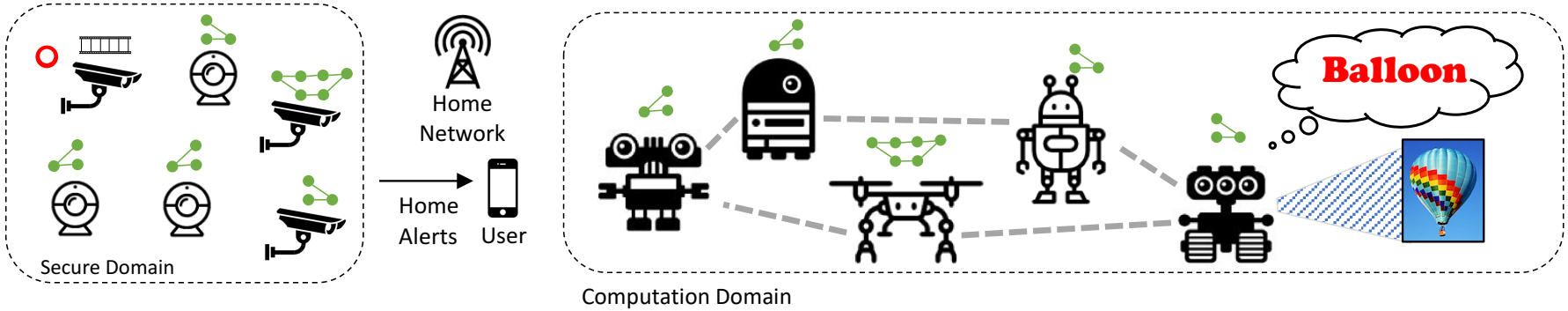
- Unreliable connections to the cloud
  - What happens when device is disconnected
- Low bandwidth and high latency
  - No real-time processing
- Privacy concerns of personal data
  - Hard to guarantee privacy of users
- New features such personalization and domain adaptations are hard to implement with this model

Solution – IoT Collaboration

Distribute computations of a single inference*

- Deployed after DNN optimizations for embedded devices such as compression and quantization
- Achieves linear speed up with number of devices

Not Dependent on Cloud | Privacy Preserving



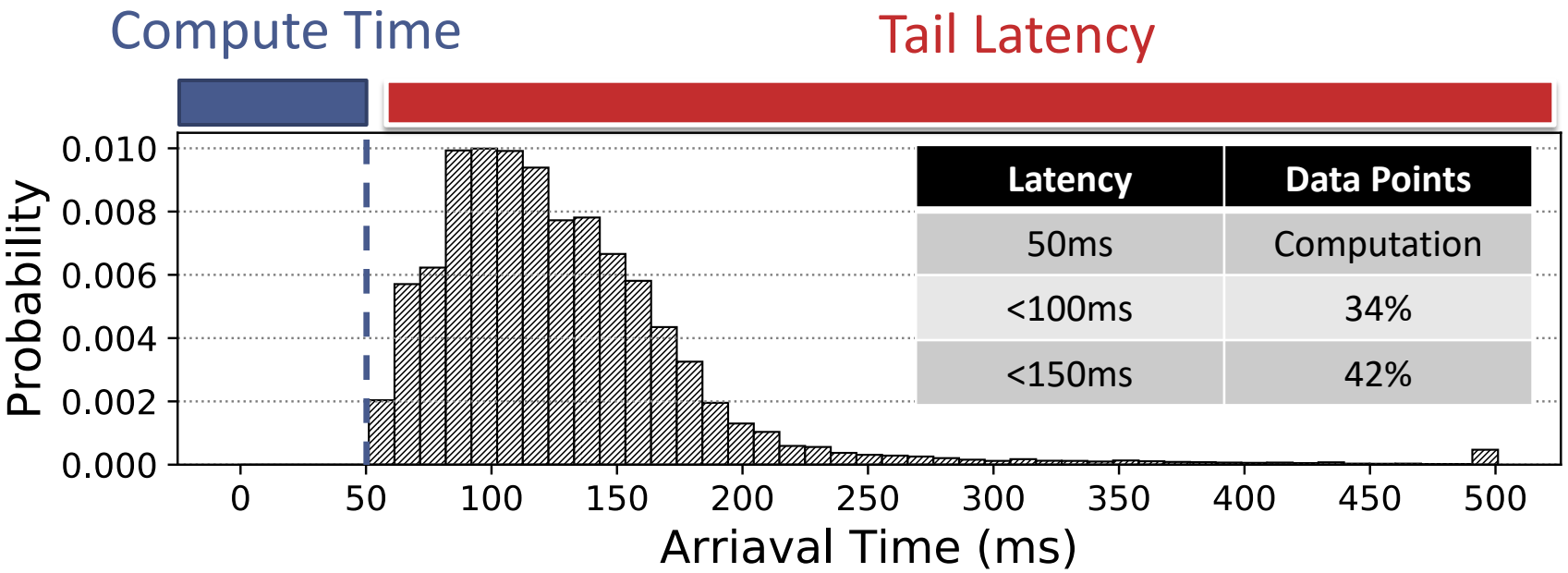
* R. Hadidi, Deploying Deep Neural Networks in Edge with Distribution, PhD Dissertation, Georgia Tech Library

IoT Collaboration Challenges

- Susceptible to unstable latency and straggler problem
- Intermittent device failures
- Susceptible to losing part/all of data
- Devices may become busy with other tasks, such as user interaction
- High recovery overheads with traditional methods

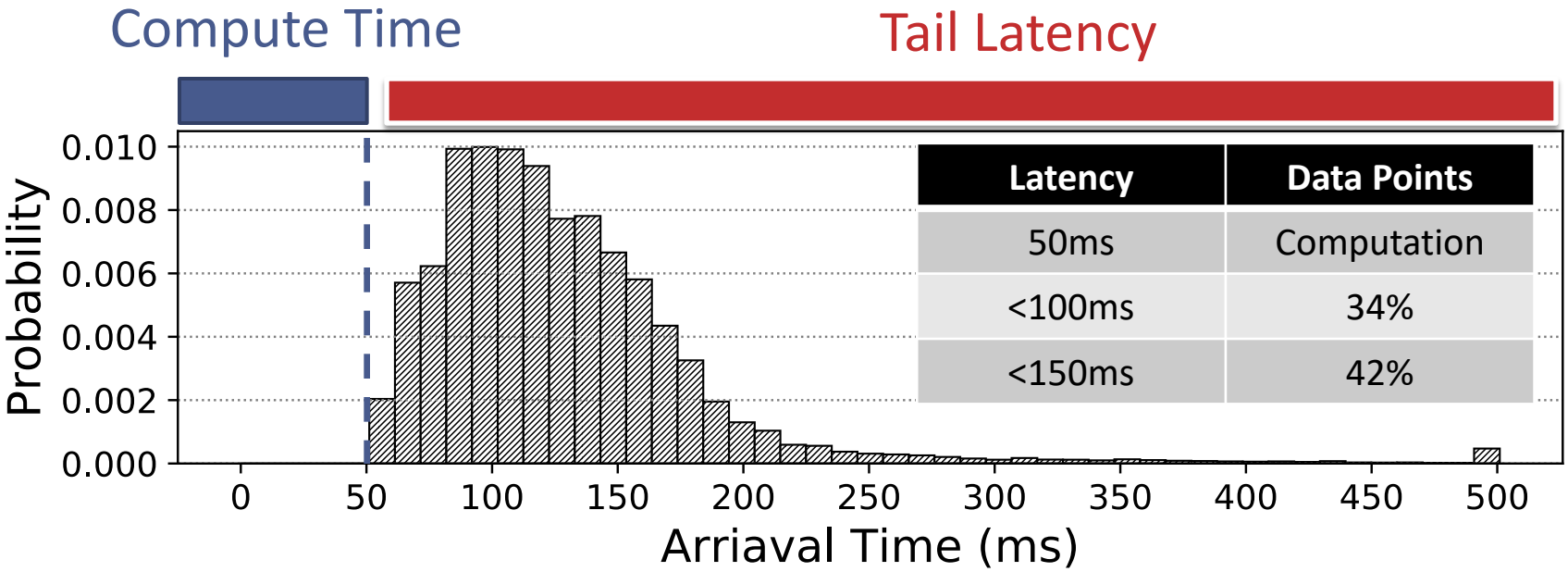
Challenges: Unreliable Latencies

Histogram distributed version of AlexNet's final fully-connection layer in 4-node system



Challenges: Unreliable Latencies

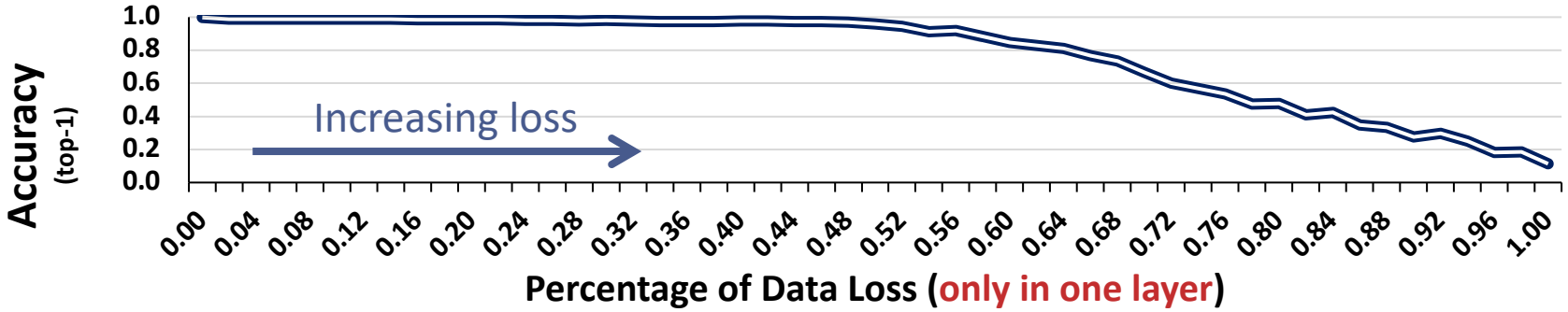
Histogram distributed version of AlexNet's final fully-connection layer in 4-node system



Long Tail and Max Latency → Straggler Problem

Challenges: Accuracy Drop

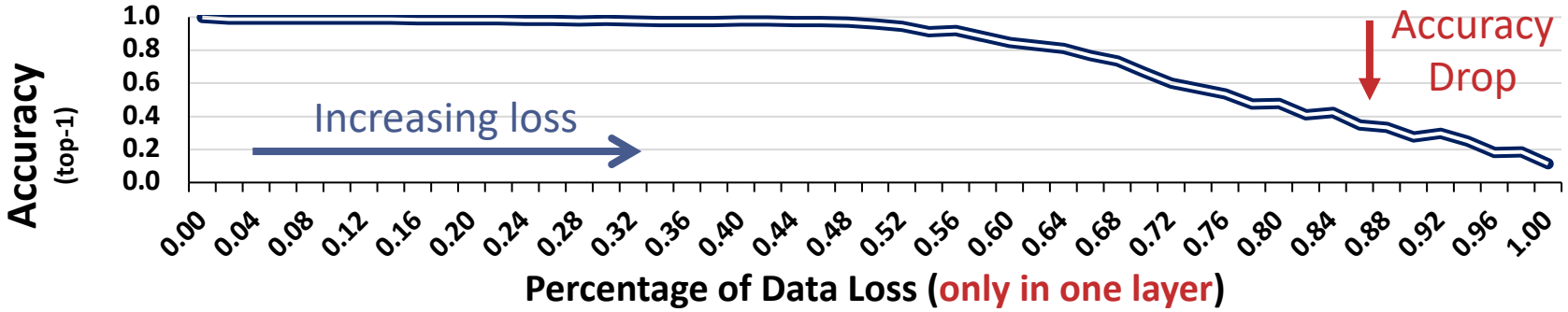
Even small packet drops are destructive for DNNs



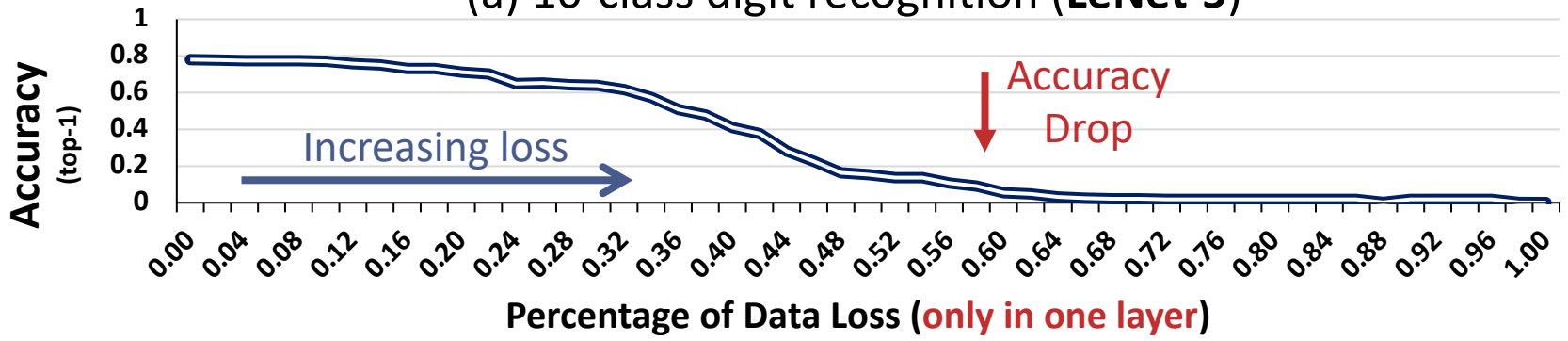
(a) 10-class digit recognition (**LeNet-5**)

Challenges: Accuracy Drop

Even small packet drops are destructive for DNNs



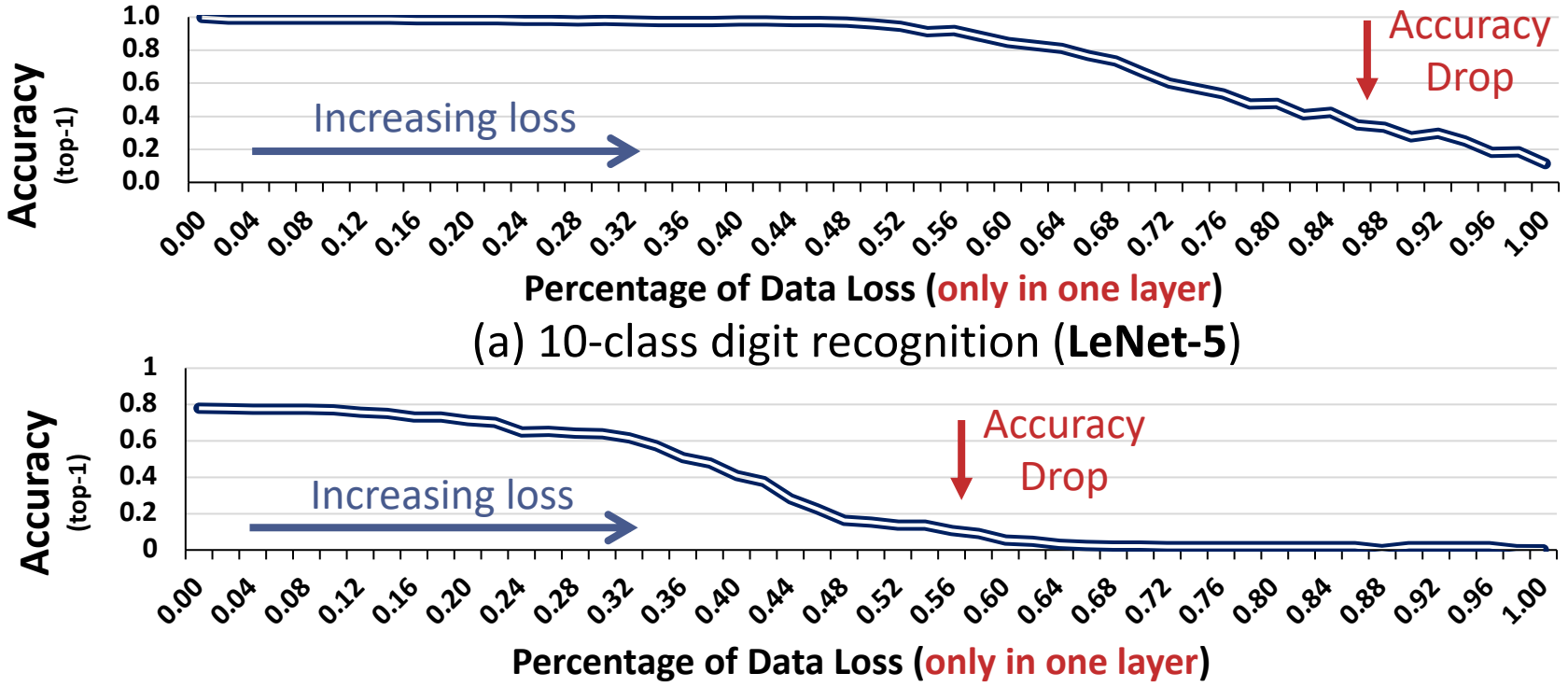
(a) 10-class digit recognition (**LeNet-5**)



(b) 1000-class image recognition (**Inception-v3**)

Challenges: Accuracy Drop

Even small packet drops are destructive for DNNs



High accuracy drop for complex data

There is a need for distribution methods that are more **efficient** and **robust** for DNNs

Our Solution

We propose a novel robustness method repurposing **Coded Distributed Computing (CDC)***

- Close-to-zero recovery latency for DNN computations
  (Never spending time to recover from a failure)
- Achieves lower latency by removing stragglers
- Minimal changes to the program
- The cost remains constant even as the number of devices to cover increases

* Li, Songze, et al. "A fundamental tradeoff between computation and communication in distributed computing." *IEEE Transactions on Information Theory* 64.1 (2018): 109-128.

Steps to Reach to Our Solution

- How DNN computations are transformed to matrix-matrix multiplications?
- How does distribution affect this matrix-matrix multiplications performed on each device?
- What is coded distributed computing (CDC)?
- How to apply CDC to matrix-matrix multiplications?
- How does this solution achieve better latency and recovery times?

Steps to Reach to Our Solution

- How DNN computations are transformed to matrix-matrix multiplications?
- How does distribution affect this matrix-matrix multiplications performed on each device?
- What is coded distributed computing (CDC)?
- **How to apply CDC to matrix-matrix multiplications?**
- **How does this solution achieve better latency and recovery times?**

Using CDC for Robustness

A simple example to showing the main insight

- Add column-wise summation of the weights:

$$\begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{11}+w_{21} & w_{12}+w_{22} \end{bmatrix} \times \begin{bmatrix} a'_1 \\ a'_2 \end{bmatrix} = \begin{bmatrix} a_1 \\ a_2 \\ a_1+a_2 \end{bmatrix}$$

- The new weights are constant, so added offline

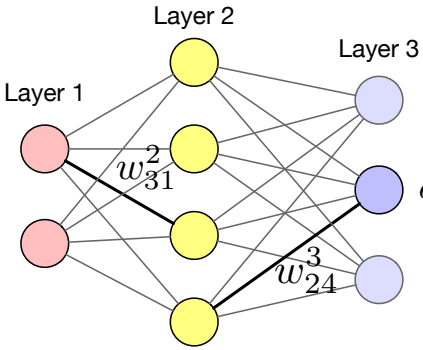
$$\begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{:1}^{cdc} & w_{:2}^{cdc} \end{bmatrix} \times \begin{bmatrix} a'_1 \\ a'_2 \end{bmatrix} = \begin{bmatrix} a_1 \\ a_2 \\ a^{cdc} \end{bmatrix}$$

- We can recover one failure since we have the summation (i.e., performing one substruction)

Distributed DNNs

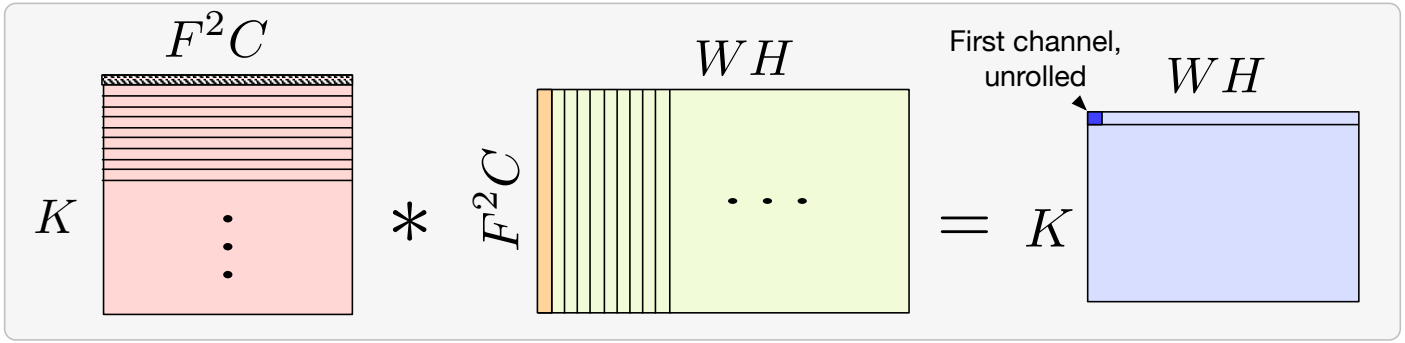
Each layer's computations can be represented as matrix-matrix multiplication (*GEMM* kernels).

Fully-connected layer:



$$a^3_2 = \sigma\Big(\sum_{k=1}^{4} w^2_{3k} a^{l-1}_k + b^l_2\Big)$$

Conv. layer:



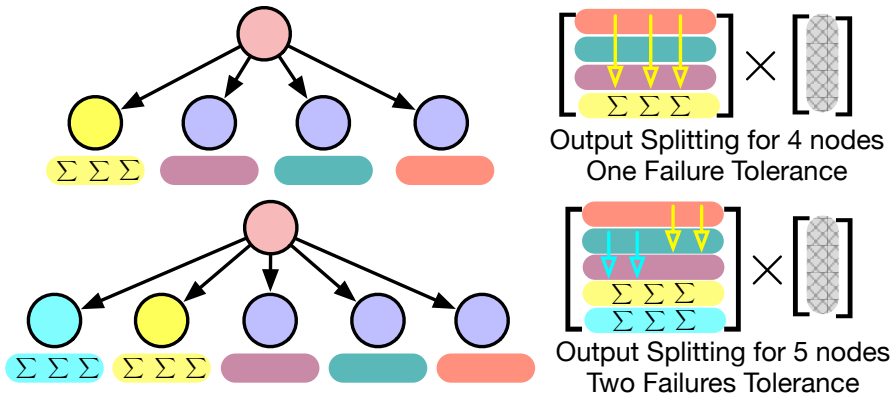
$$\mathrm{W}_{k \times F^2C} \times \mathrm{I}_{F^2C \times WH} = O_{K \times WH}$$

How to Distribute CDC and Benefits

Add column-wise summation of the weights.

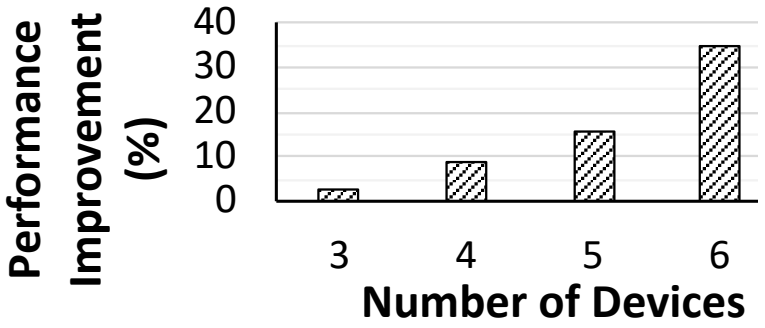
Benefits:

- Recovery
  Local Subtraction vs. (Transmit + Multiplication)
- Addition of one device covers all computations
- Introduced computations are similar on nature to DNNs



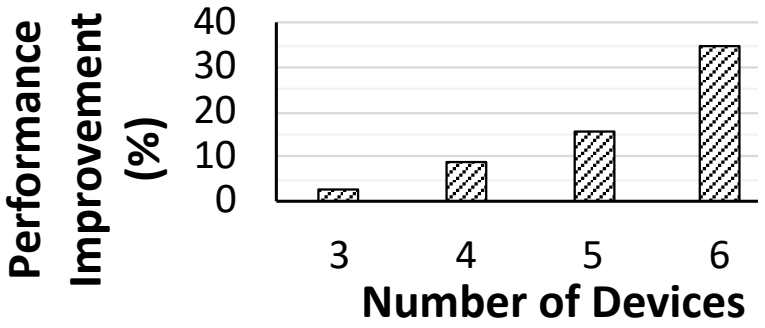
Straggler Mitigation & Failure Coverage

Do not need to wait for all devices to send data: (AlexNet)

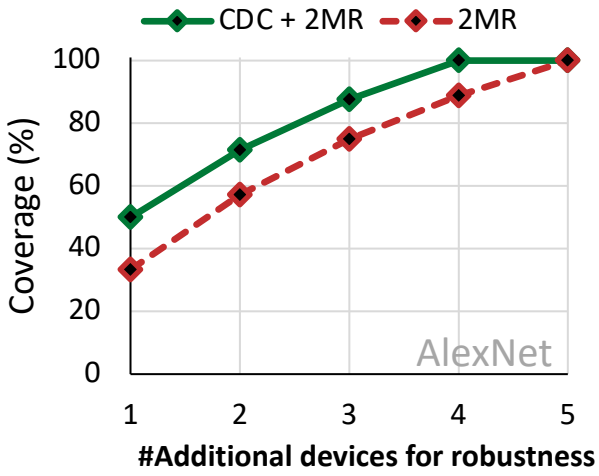


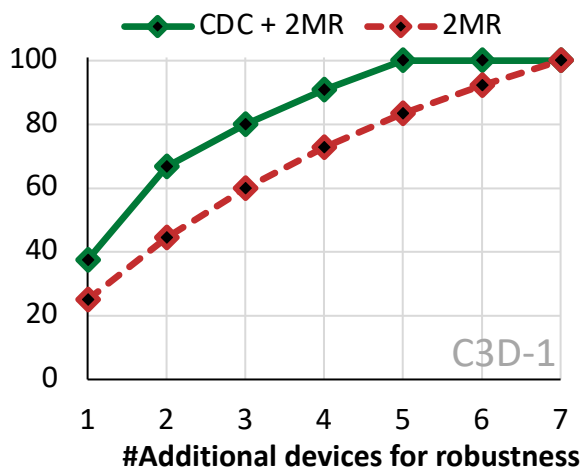
Straggler Mitigation & Failure Coverage

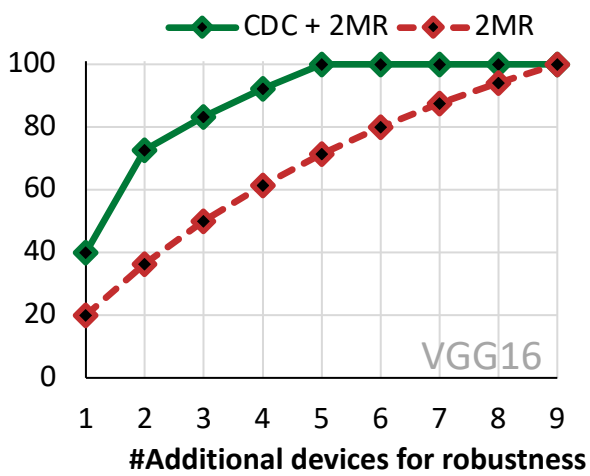
Do not need to wait for all devices to send data: (AlexNet)



Better Coverage versus with 2-modular redundancy (2MR)







Please check the paper for more details on

- Distribution methods
- Distribution methods that CDC would work on
- Formulization
- Introduced computations
- Extending to more failures

Creating Robust Deep Neural Networks With Coded Distributed Computing for IoT

Ramyad Hadidi[§] — *Rain AI* — ramyad@rain.ai

Jiashen Cao — *Georgia Tech* — jiashenc@gatech.edu

Bahar Asgari[§] — *University of Maryland* — bahar@umd.edu

Hyesoon Kim — *Georgia Tech* — hyesoon.kim@gatech.edu

***Abstract*—The increasing interest in serverless computation and ubiquitous wireless networks has led to numerous connected devices in our surroundings. Such IoT devices have access to an abundance of raw data, but their inadequate resources in computing limit their capabilities. With the emergence of deep neural networks (DNNs), the demand for the computing power of IoT devices is increasing. To overcome inadequate resources, several studies have proposed distribution methods for IoT devices that harvest the aggregated computing power of idle IoT devices in an environment. However, since such a distributed system strongly relies on each device, unstable latency, and intermittent failures, the common characteristics of IoT devices and wireless networks, cause high recovery overheads. To reduce this overhead, we propose a novel robustness method with a close-to-zero recovery latency for DNN computations. Our solution never loses a request or spends time recovering from a failure. To do so, first, we analyze how matrix computations in DNNs are affected by distribution. Then, we introduce a novel coded distributed computing (CDC) method, the cost of which, unlike that of modular redundancies, is constant when the number of devices increases. Our method is applied at the library level, without requiring extensive changes to the program, while still ensuring a balanced work assignment during distribution.**

***Index Terms*—Edge AI, Reliability, IoT, Edge, Distributed Computing, Collaborative Edge & Robotics**

I. Introduction

Recent years have witnessed the emergence of deep neural network (DNN) applications. Additionally, with the proliferation of Internet-of-Things (IoT) devices, they became inseparable from our daily lives. The conventional methods to process raw IoT data are to offload them to cloud services. However, moving such a tremendous amount of data incurs a high amount of monetary cost and delay, besides creating a major concern of privacy leakages. Therefore, serverless and edge computation paradigms are recognized as promising solutions. As a result, pushing the frontier of DNNs computations to the edge is receiving a tremendous amount of interest both from academia [1]–[9] and from the industry with commercial edge-tailored hardware accelerators such as NVIDIA Jetson Nano, edge TPU, and Intel Movidius.

Processing IoT data locally in the edge may suffer from poor performance and energy efficiency because the computational demand from DNNs outweighs the computation capacity and energy constraints of IoT devices. Furthermore, the computational demands are escalated because these devices have to meet real-time constraints. Even for edge-tailored hardware accelerators, the real timeliness of applications is not guaranteed [10], [11]. Nevertheless, privacy concerns, unreliable connection to the cloud, tight real-time requirements, and personalization are still pushing inferencing to the edge. To address the resource constraint challenges, a solution is to distribute heavy computations among idle devices [1], [2], [4], [12] because the state-of-the-art IoT networks are formed with various IoT sensors and recording agents, such as HD cameras and temperature sensors, many of which are capable of performing computations. However, such a distribution is susceptible to failures, from short disconnectivity and user interaction to losing a device. This fact necessitates developing a robust method for tolerating these failures. Additionally, since IoT networks use wireless technology, unreliability and variability in their networks are much higher than acceptable limits to ensure a robust system.

We extend studies that enable distributed single-batch inference of DNNs in the edge [1], [2], [4], [12] to tolerate failures with close-to-zero recovery latency. We first analyze general methods of distributing the computations of DNNs and how their underlying general matrix-matrix multiplication (GEMM) is affected by distribution. Such a detailed study is necessary to introduce a general seamless method within the underlying library or machine learning framework. Then, we propose a new recovery method based on coded distributed computing (CDC) that enables distributed DNN models on IoT devices to tolerate failures. Our method is inspired by CDC applications in big data analytics [13], and speeding up distributed learning using codes [14].

To enable robustness in distributed IoT, we introduce an extra coded computation per device. We propose a novel fault recovery method based on CDC that has close-to-zero recovery latency, does not disturb the balanced work assignment in distribution, requires minimal changes to the program, and has a constant cost with the increasing number of devices. Our introduced extra computations are derived by thoroughly analyzing how general methods of distributing the computation of DNNs affect their underlying GEMM. The added computations are similar in nature to those of DNNs, which eases balancing the work among IoT devices and reduces the deployment cost. Balanced distribution is essential in attaining the expected performance. Additionally, since our method is implemented at the library level, it does not require changes to the program. Moreover, unlike approaches that sacrifice latency for robustness to recompute the missing part of the data, our

This work was supported in part by the NSF grant number 2103951.

§ This work was done when the authors were affiliated with Georgia Tech.

Future Work

Same concept for robustness and speedup is also applicable for distributed commutating between

- Chip to chip (PCIe)
- Die to Die (UCIe)
- Processing Elements (Network on Chip)

Coded Distributed Computing (CDC)

Designed for MapReduce workloads (2018)

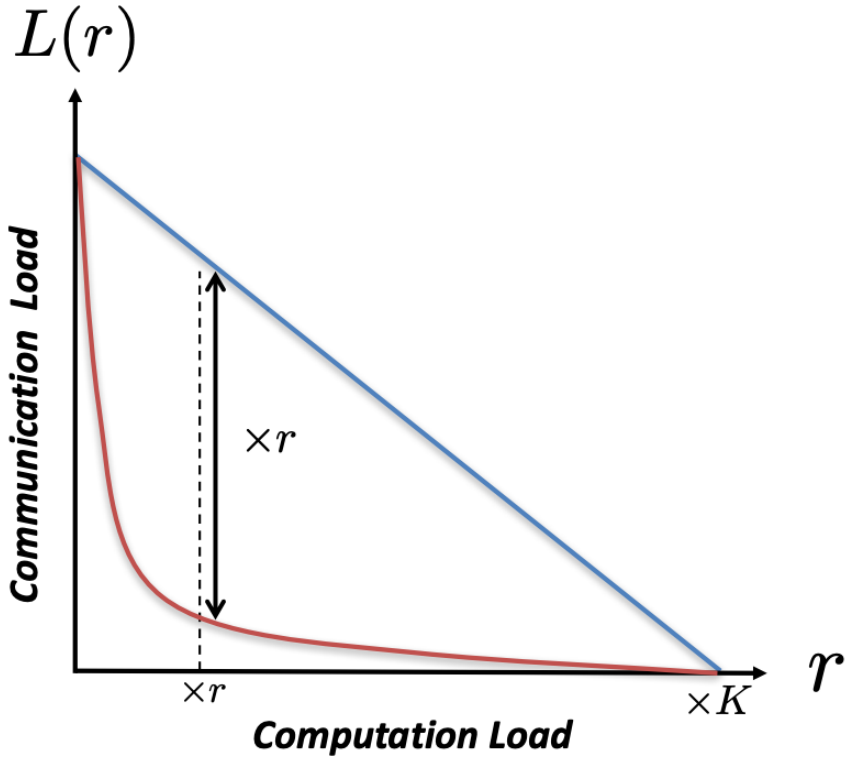
Preforming redundant or coded computer per node to reduce communication

This work: **DNNs on IoT**

More Compute / Node

=

More Reliability



CDC for Distributed DNNs (FC)

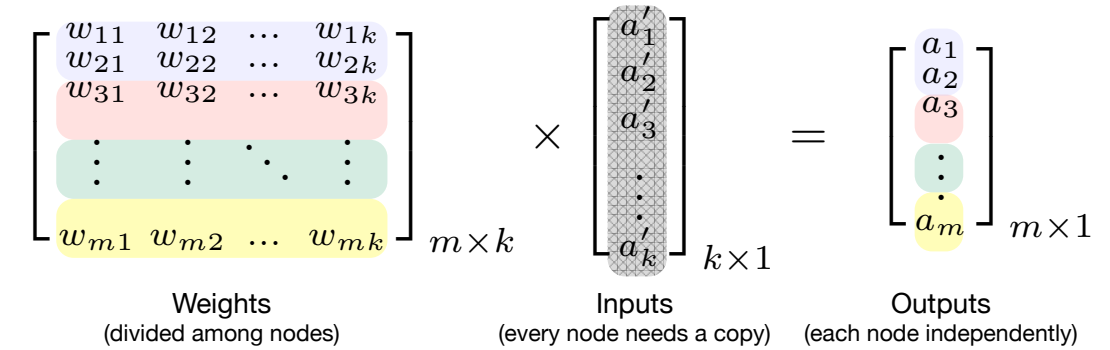
Methods distributing computation of a model*

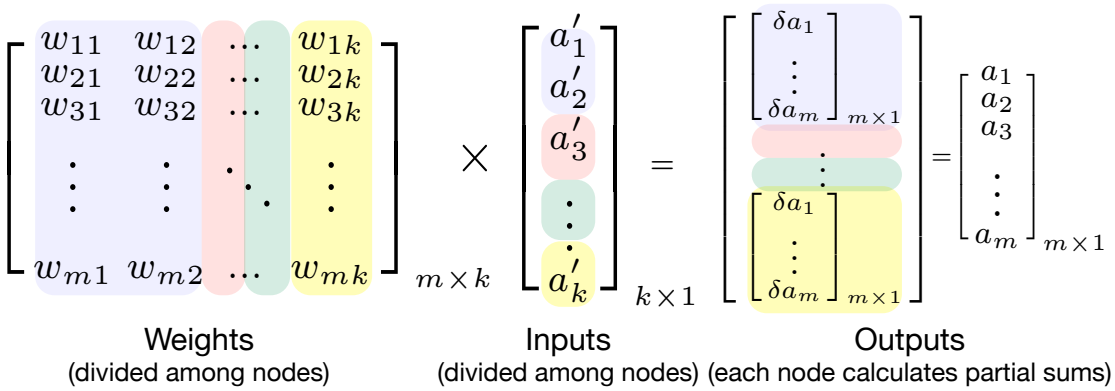
Fully-connected Layers

Output splitting:

[Several in-between variants]

Input splitting:



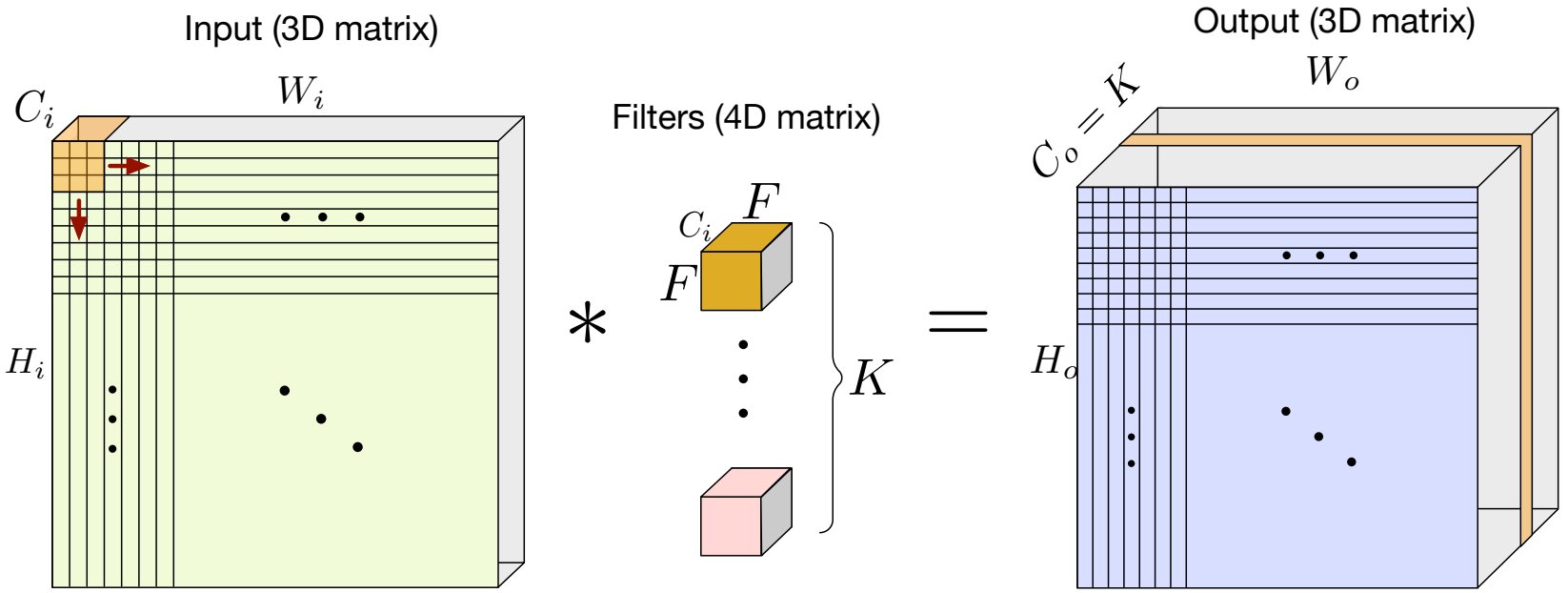


Same can be applied on convolution layers*

* R. Hadidi, J. Cao, M. S. Ryoo and H. Kim, "Toward Collaborative Inferencing of Deep Neural Networks on Internet-of-Things Devices," in *IEEE Internet of Things Journal*,

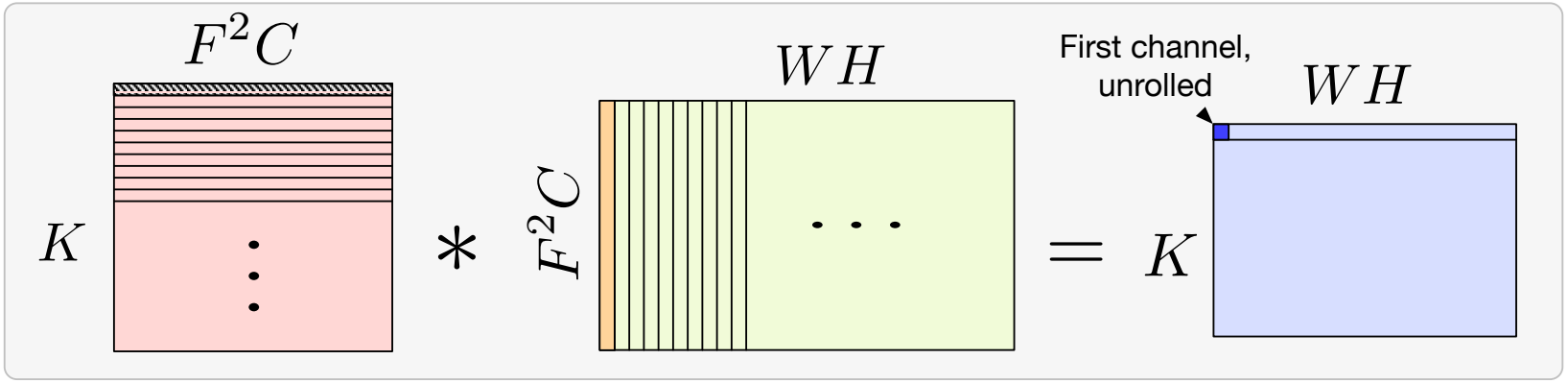
CDC for Distributed DNNs (conv)

Same can be applied on convolution layers

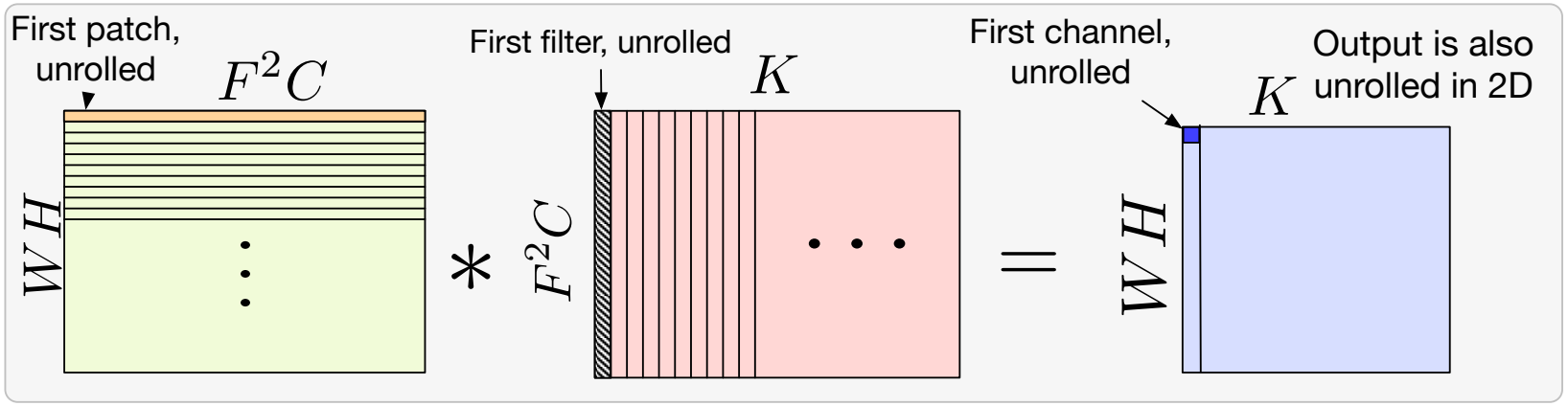


CDC for Distributed DNNs (conv)

Conv to GEMM



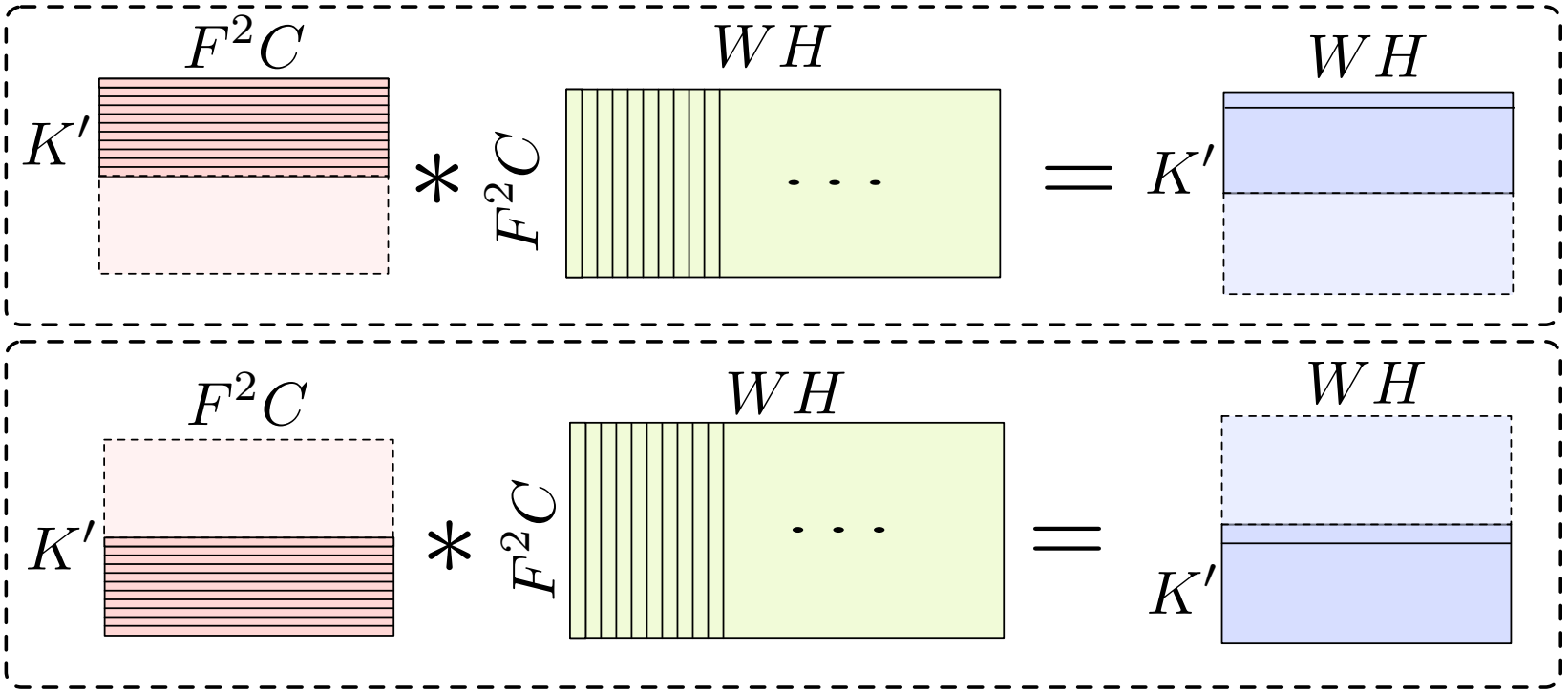
(a) $\mathrm{W}_{k\times F^2C} \times \mathrm{I}_{F^2C\times WH} = O_{K\times WH}$



(b) $\mathrm{I}_{WH\times F^2C} \times \mathrm{W}_{F^2C\times K} = O_{WH\times K}$

CDC for Distributed DNNs (conv)

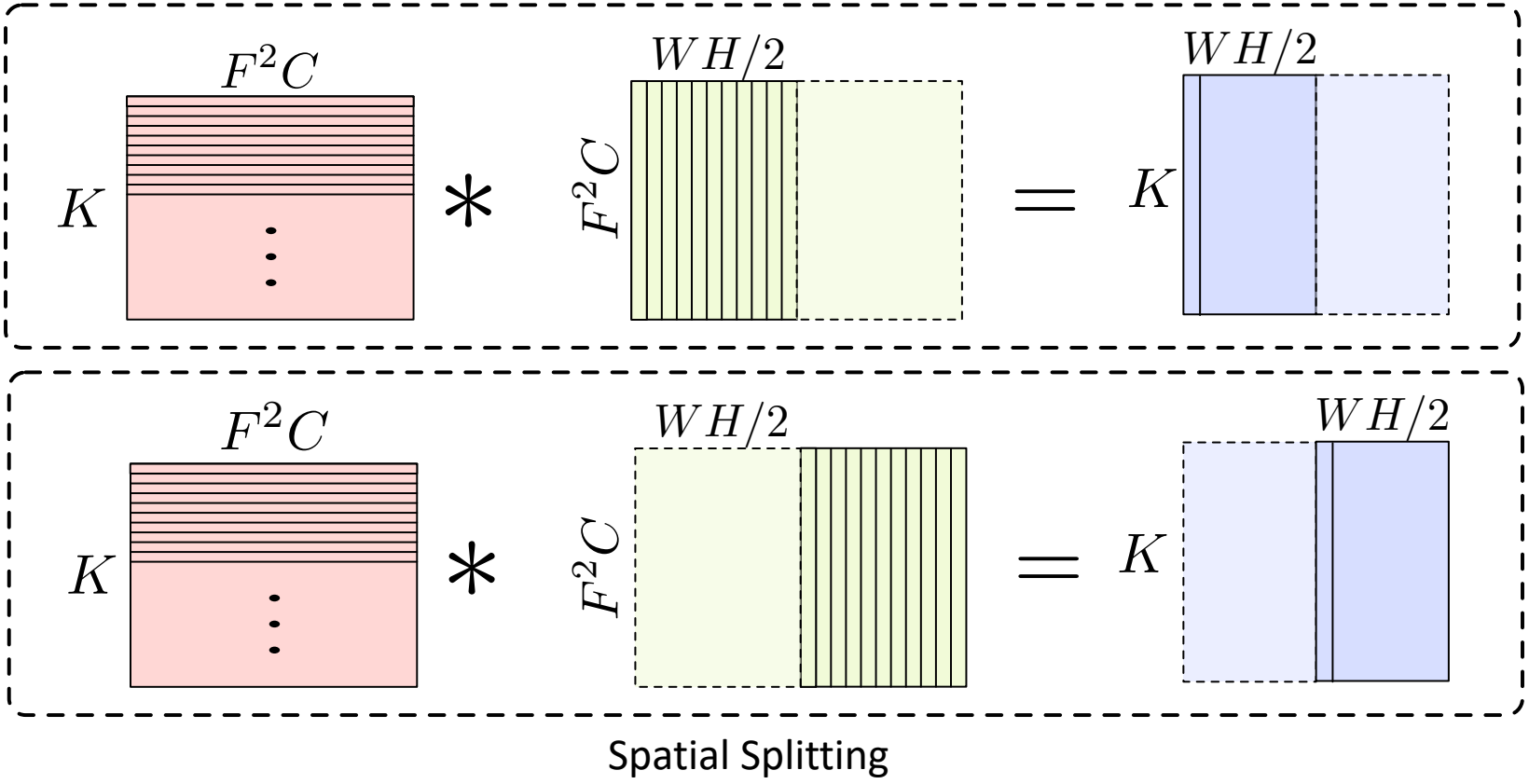
Same can be applied on convolution layers*



Channel Splitting

CDC for Distributed DNNs (conv)

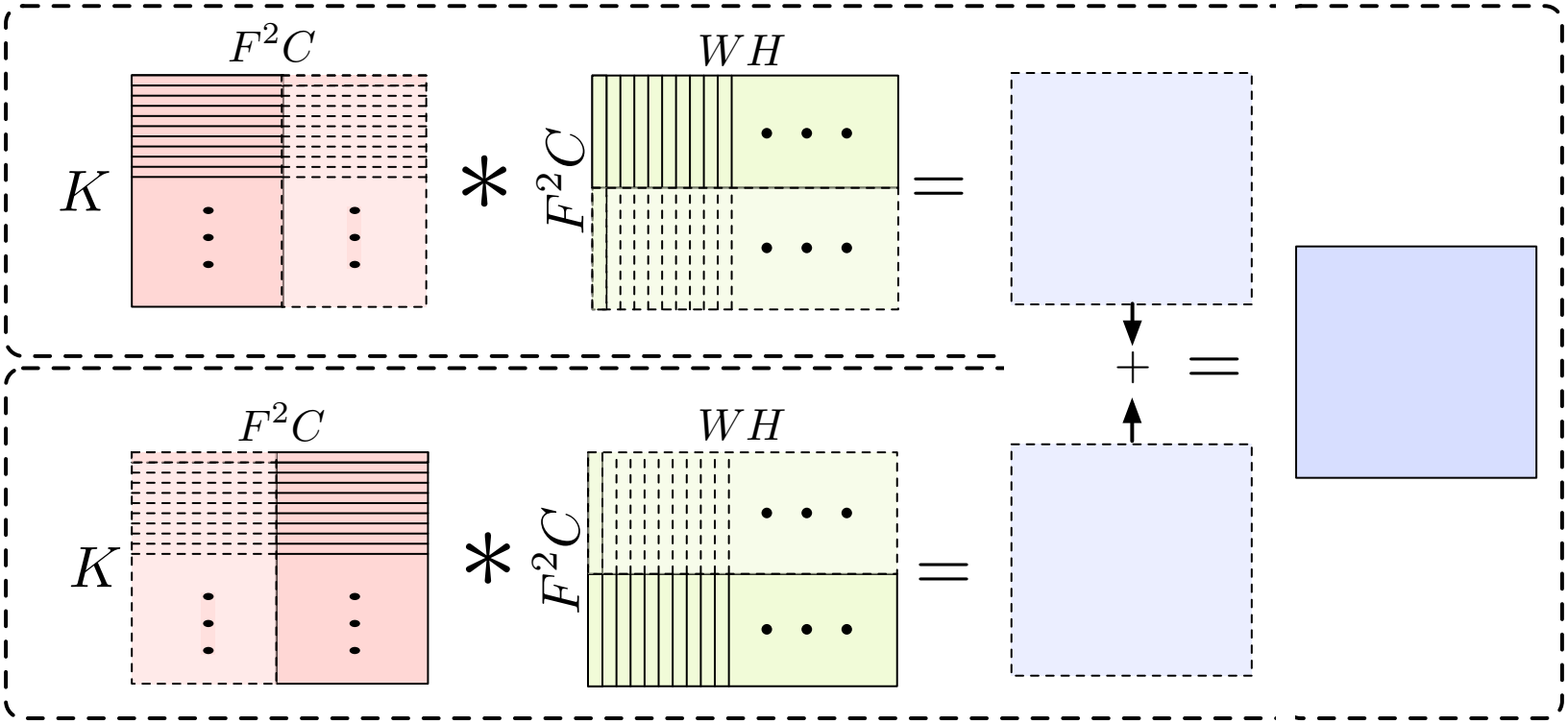
Same can be applied on convolution layers*



Spatial Splitting

CDC for Distributed DNNs (conv)

Same can be applied on convolution layers*



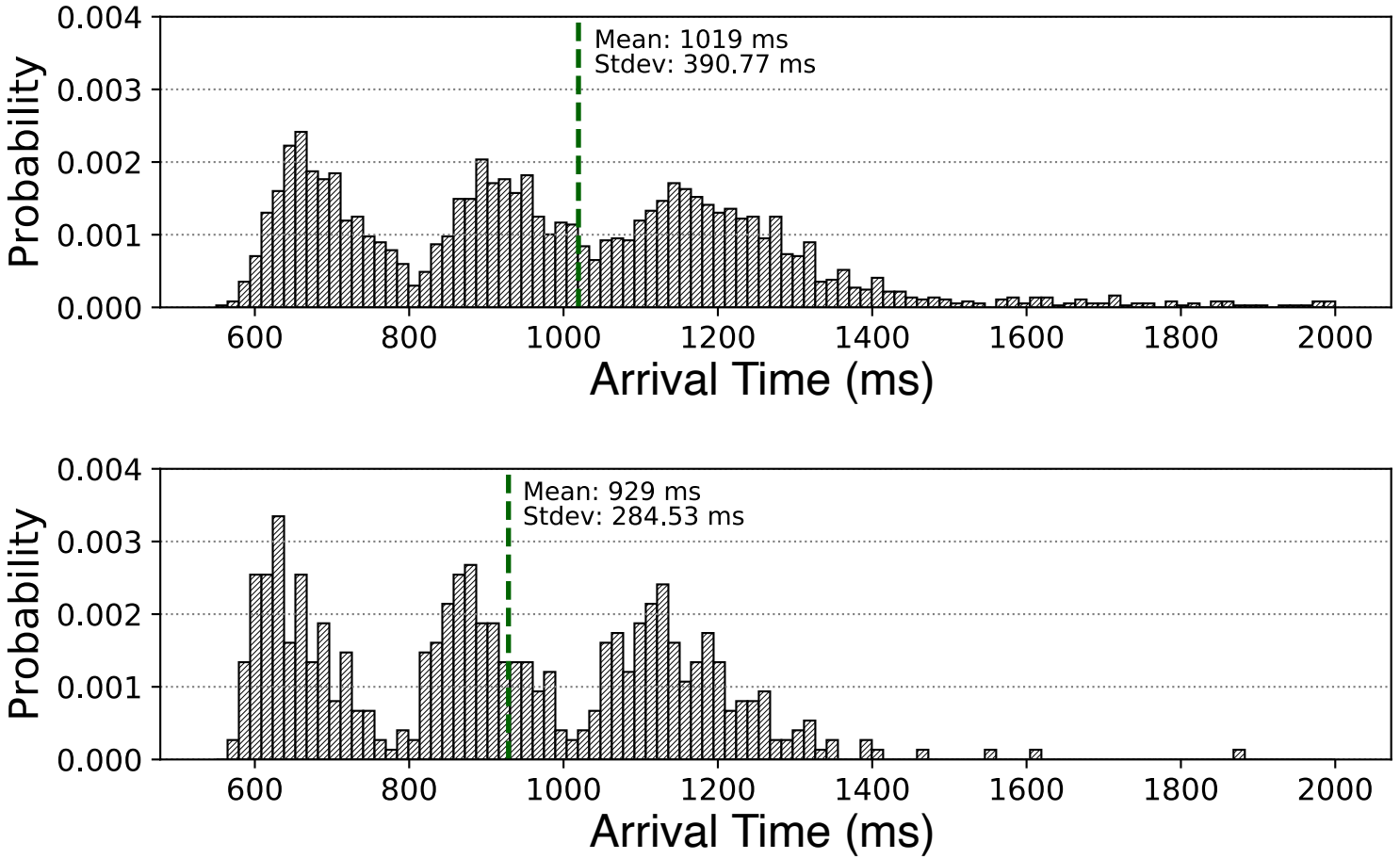
Filter Splitting

Formula

Multiple out/device: Just create a new weight matrix

$$\begin{bmatrix} w_{11}+w_{(\frac{m}{2}+1)1} & w_{12}+w_{(\frac{m}{2}+1)2} & \cdots & w_{1k}+w_{(\frac{m}{2}+1)k} \\ w_{21}+w_{(\frac{m}{2}+2)1} & w_{22}+w_{(\frac{m}{2}+2)2} & \cdots & w_{2k}+w_{(\frac{m}{2}+2)k} \\ \vdots & \vdots & \ddots & \vdots \\ w_{\frac{m}{2}1}+w_{m1} & w_{\frac{m}{2}2}+w_{m2} & \cdots & w_{\frac{m}{2}k}+w_{mk} \end{bmatrix}_{\frac{m}{2}\times k}$$

AlexNet w/o & w Straggler Mitigation



AlexNet w/o & w Recovery