LCP: A Low-Communication Parallelization Method for Fast Neural Network Inference for IoT

CSCE
July 2023

Ramyad Hadidi*
Rain AI

Bahar Asgari*
University of Maryland

Jiashen Cao
Georgia Tech

Younmin Bae
Georgia Tech

Da Eun Shim
Georgia Tech

Hyojong Kim
Georgia Tech

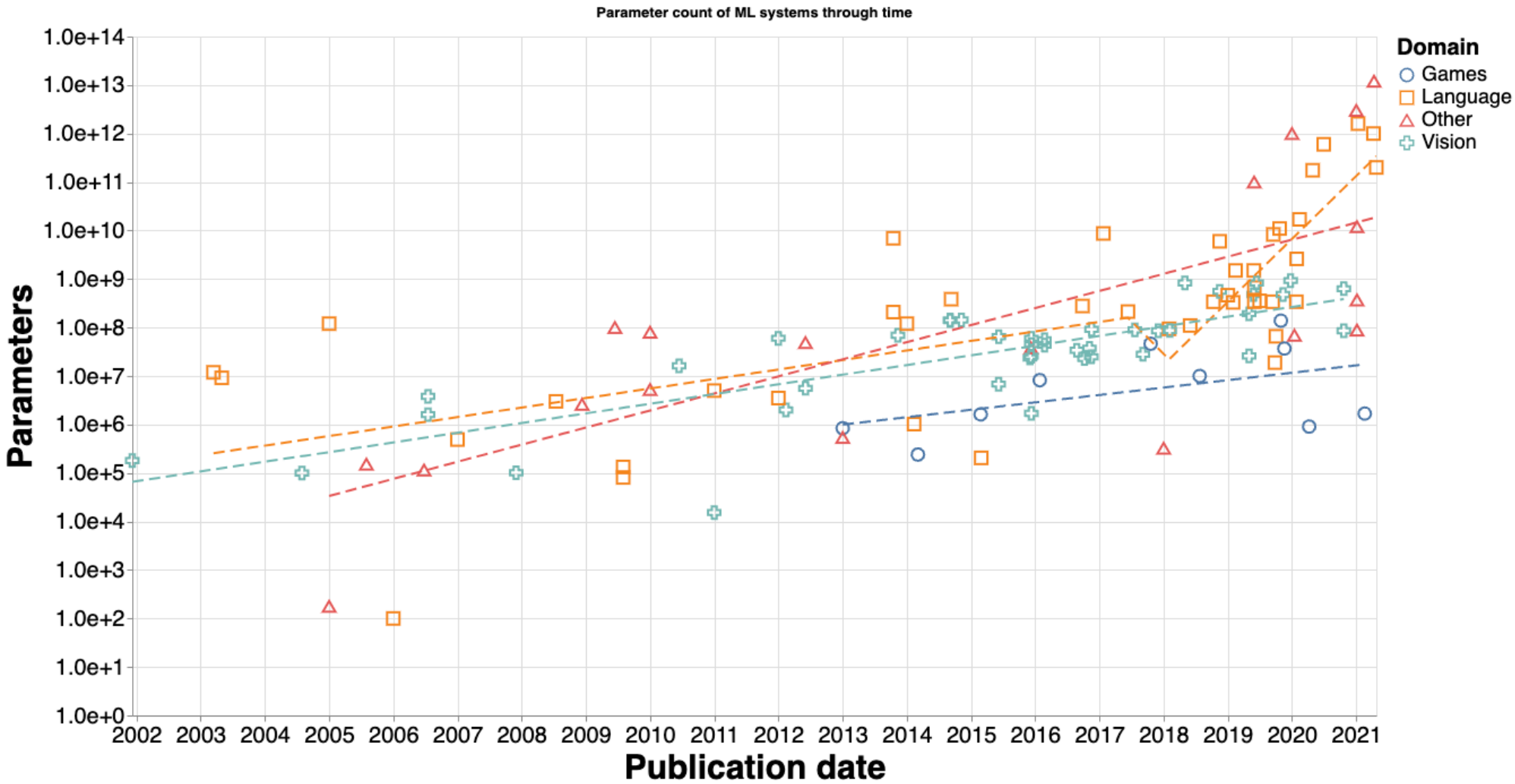
Sung-Kyu Lim
Georgia Tech

Michael S. Ryoo
Stony Brook University & Google

Hyesoon Kim
Georgia Tech

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

Modern AI



Parameter counts in Machine Learning, Jaime Sevilla,
Published in Towards Data Science Jul 2, 2021

Artificial Intelligence

GatesNotes THE BLOG OF BILL GATES

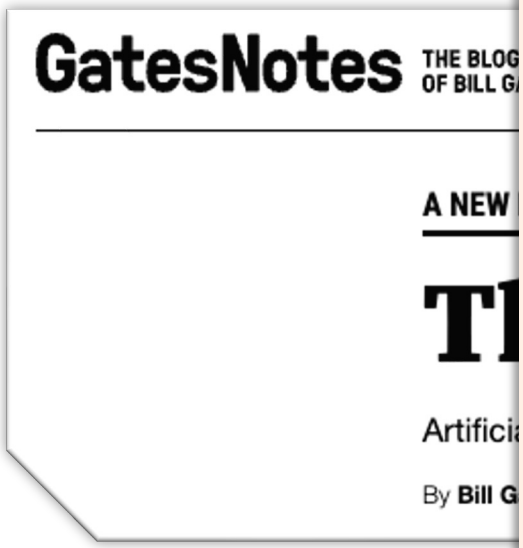
A NEW ERA

The Age of AI has begun

Artificial intelligence is as revolutionary as mobile phones and the Internet.

By **Bill Gates** | March 21, 2023 • 14 minute read

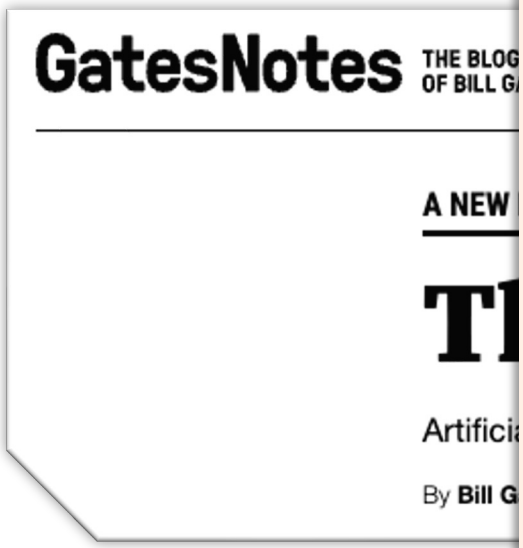
Artificial Intelligence



	2012	2022
Compute used to train largest AI model	1e+16 FLOPS (10,000,000,000,000,000)	1e+24 FLOPS (1,000,000,000,000,000,000,000,000)
Data consumed by largest AI model	Imagenet: a dataset of 15mn labelled images (150GB)	The entire internet (45,000GB)
Capabilities of largest AI models	Can recognise images at “beginner human” level Superhuman at chess	Superhuman or high-human at a wide variety of games (Go, Diplomacy, Starcraft II, poker etc) Human-level at 150 reasoning & knowledge tasks Passes US Medical Licensing Exam, passes the Bar Exam Displays complex capabilities like power-seeking, deceiving humans Can self-improve by “reasoning” out loud Can write 40 per cent of the code for a software engineer

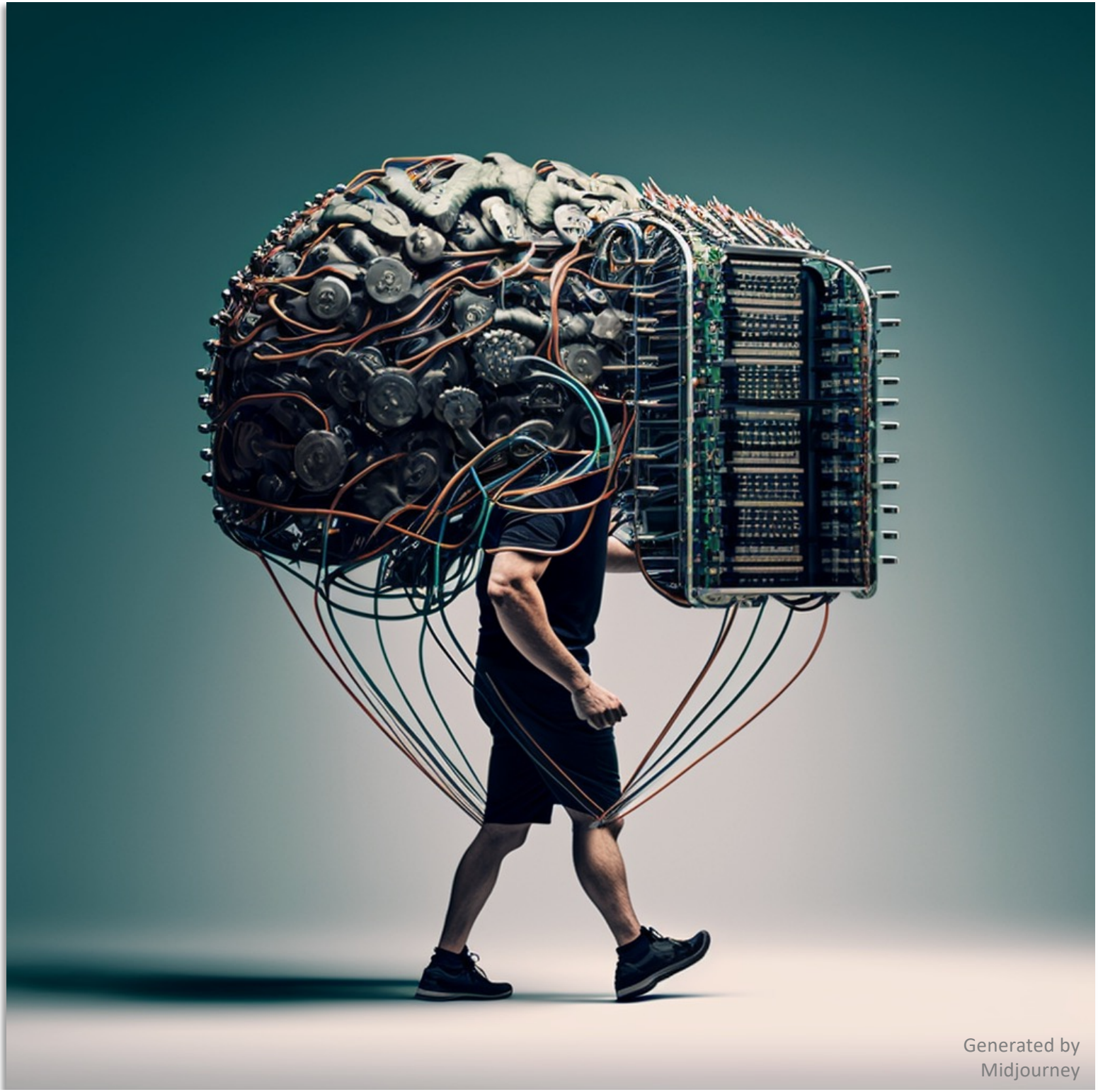
Hogarth, Ian. "We must slow down the race to God-like AI." Financial Times, 12 Apr. 2023

Artificial Intelligence



	2012	2022
Compute used to train largest AI model	1e+16 FLOPS (10,000,000,000,000,000)	1e+24 FLOPS (1,000,000,000,000,000,000,000,000)
Data consumed by largest AI model	Imagenet: a dataset of 15mn labelled images (150GB)	The entire internet (45,000GB)
Capabilities of largest AI models	Can recognise images at "beginner human" level Superhuman at chess	Superhuman or high-human at a wide variety of games (Go, Diplomacy, Starcraft II, poker etc) Human-level at 150 reasoning & knowledge tasks Passes US Medical Licensing Exam, passes the Bar Exam Displays complex capabilities like power-seeking, deceiving humans Can self-improve by "reasoning" out loud Can write 40 per cent of the code for a software engineer

Hogarth, Ian. "We must slow down the race to God-like AI." Financial Times, 12 Apr. 2023


Generated by
Midjourney

The Challenge

DNNs are increasingly deeper and wider models with higher computational demands

Training is Hard

Training GPT-3 on
3584x H100s would take 46h*

* Nvidia, NVIDIA H100 GPUs Set Standard for Generative AI in Debut MLPerf Benchmark, 27 Jun. 2023

The Challenge

DNNs are increasingly deeper and wider models with higher computational demands

Training is Hard

Training GPT-3 on
3584x H100s would take 46h*

Fast Inference is Even Harder

Requiring optimizations at various levels of HW-SW and next-level of efficient production toolsets

* Nvidia, NVIDIA H100 GPUs Set Standard for Generative AI in Debut MLPerf Benchmark, 27 Jun. 2023

Speeding up Inference

Several techniques are employed for a faster inference:

- Reducing parameters/computation
  - Model compression (pruning)
  - Post-Training Quantization (PTQ)
  - Quantization-aware training (QAT)

- Exploiting **parallelism** in computation
  - Happens at several levels with several assumptions and end goals

* This is not a complete list
* Several LLM-related techniques are omitted related to reducing the computation and memory footprints

Speeding up Inference

Several techniques are employed for a faster inference:

Orthogonal Directions

- Reducing parameters/computation
  - Model compression (pruning)
  - Post-Training Quantization (PTQ)
  - Quantization-aware training (QAT)
- Exploiting **parallelism** in computation
  - Happens at several levels with several assumptions and end goals

Speeding up Inference

Several techniques are employed for a faster inference:

Orthogonal Directions

- Reducing parameter size
  - Model compression (pruning)
  - Post-Training Quantization (PTQ)
  - Quantization-aware training (QAT)
- Exploiting **parallelism** in computation
  - Happens at several levels with several assumptions and end goals

Our focus

Other optimizations can be applied after extracting parallelism

Parallelizing Computations

Inside Model

Model Parallelism
Layer Scheduling
Per Layer
Per Tensor
Per Operation

Outside Model

Multiple Models
Data Parallelism

Parallelizing Computations

Inside Model

Model Parallelism
Layer Scheduling
Per Layer
Per Tensor
Per Operation

Model Itself!

No Explicit Parallelism

Blackhole of
ML Models

Outside Model

Multiple Models
Data Parallelism

Parallelizing Computations

Inside Model

Model Parallelism
Layer Scheduling
Per Layer
Per Tensor
Per Operation

Model Itself!

No Explicit Parallelism

Blackhole of
ML Models

Outside Model

Multiple Models
Data Parallelism

Current approaches in reducing the inference latency are always applied ***after*** a model architecture is defined

Parallelizing Computations

Inside Model

Model Parallelism

Layer Scheduling

Per Layer

Per Tensor

Per Operation

Model Itself!

No Explicit Parallelism

Blackhole of
ML Models

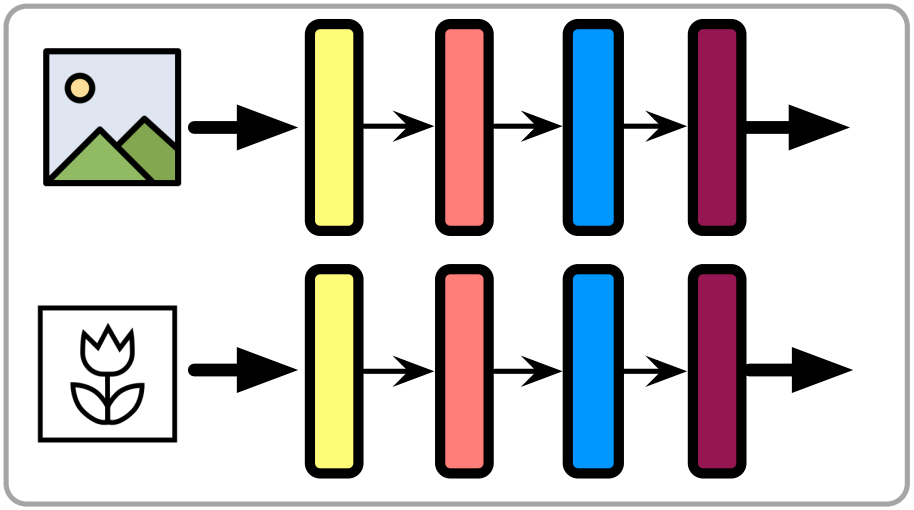
Outside Model

Multiple Models

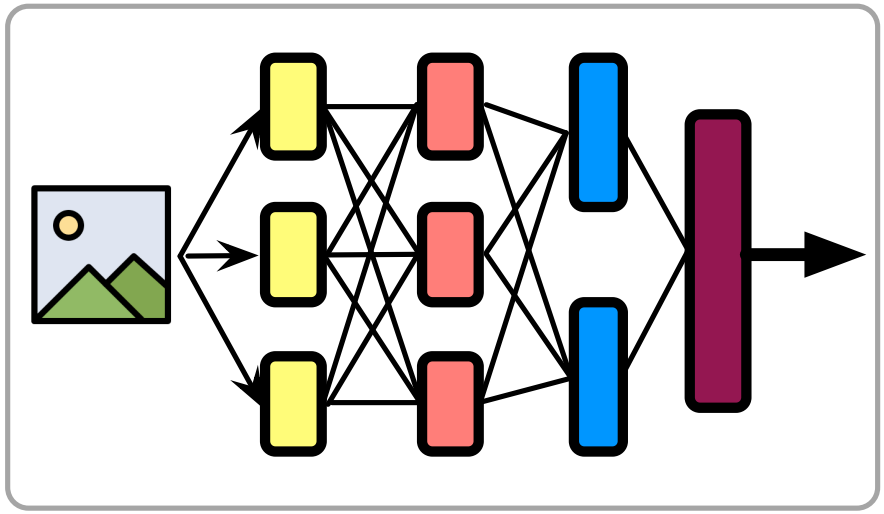
Data Parallelism

Current approaches in reducing the inference latency are always applied ***after*** a model architecture is defined

Data & Model Parallelism



Data Parallelism

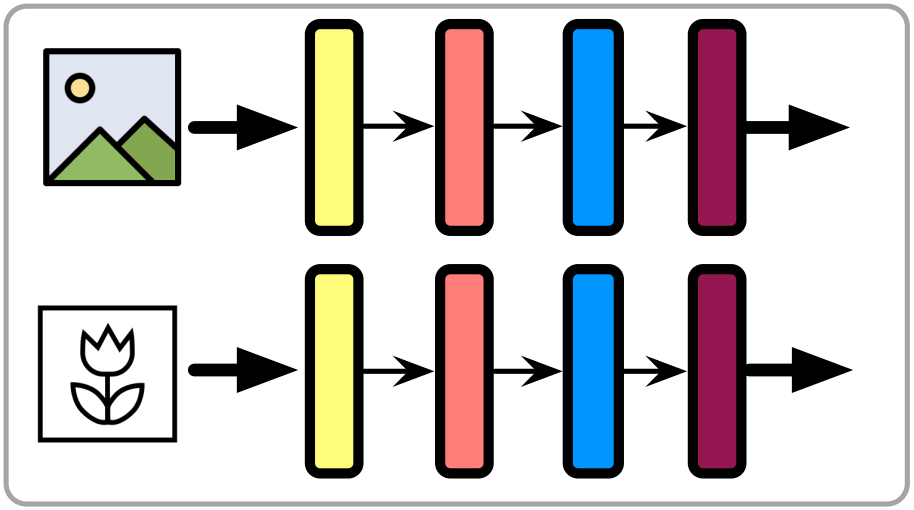


Model Parallelism

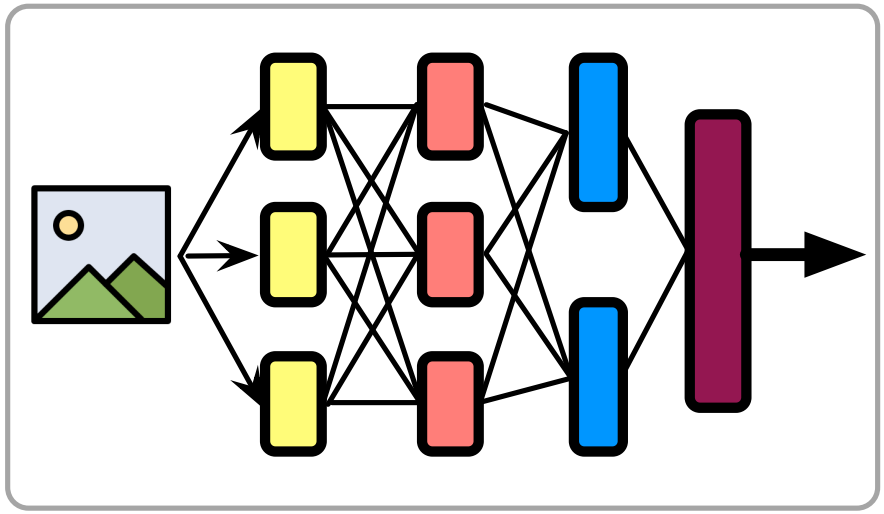
Data parallelism provides the next input to the next device in a network

Model parallelism splits layers over multiple devices, working on the same input

Data & Model Parallelism



Data Parallelism



Model Parallelism

Data parallelism provides the next input to the next device in a network

Model parallelism splits layers over multiple devices, working on the same input

Model Parallelism

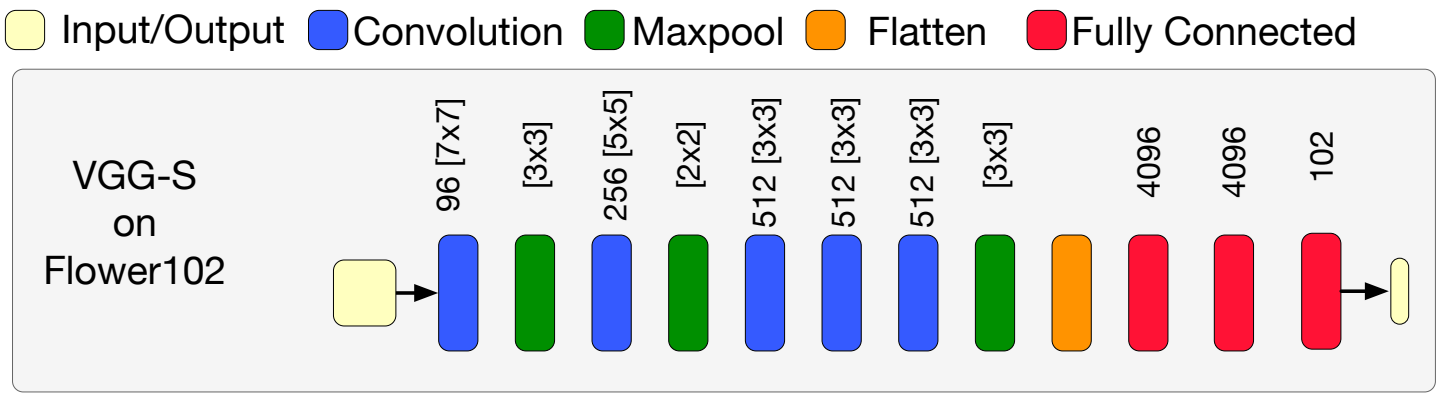
Model parallelism does not change the model

- Synchronization: Difficult to distribute
- Several Connections: High communication overhead

Model Parallelism

Model parallelism does not change the model

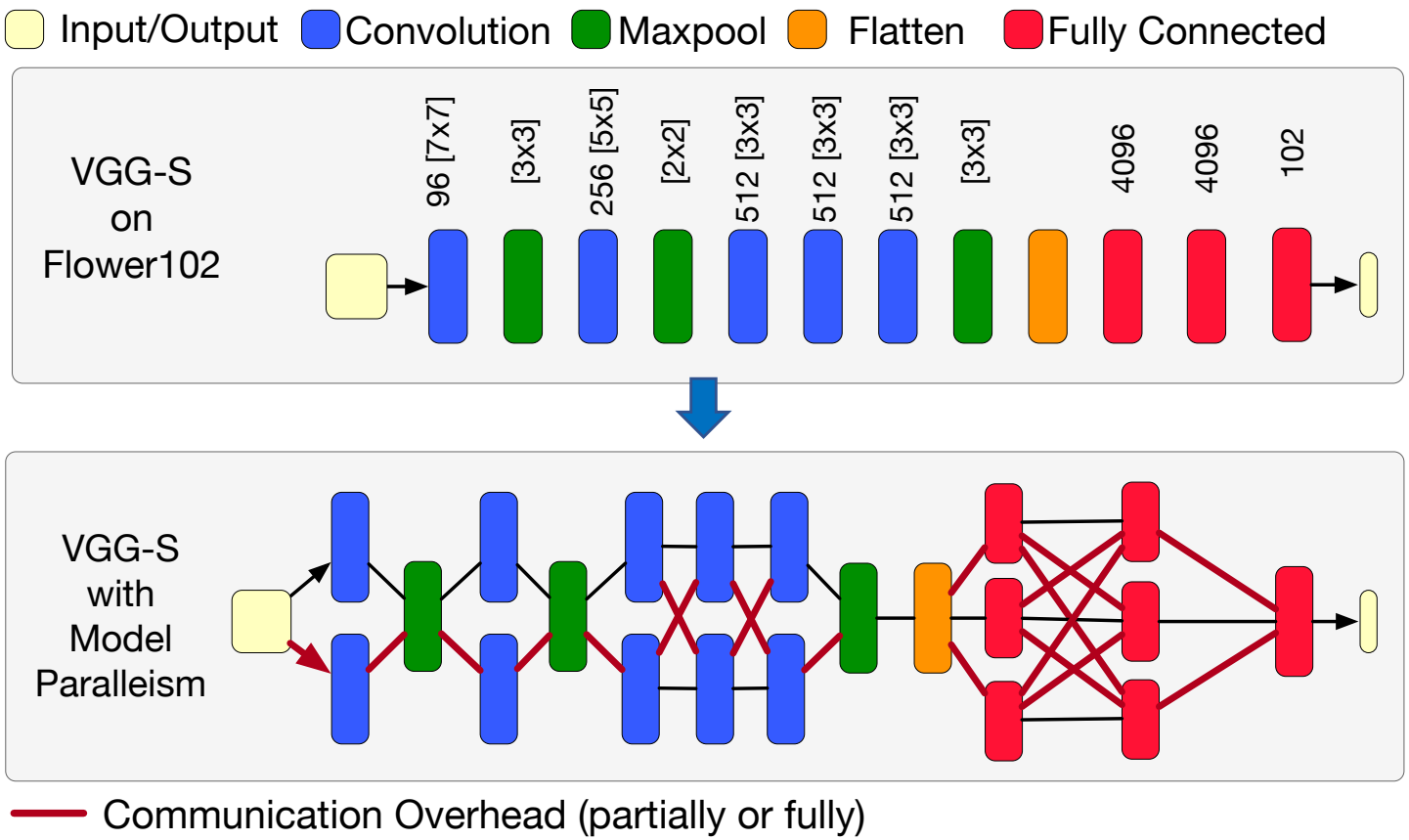
- Synchronization: Difficult to distribute
- Several Connections: High communication overhead



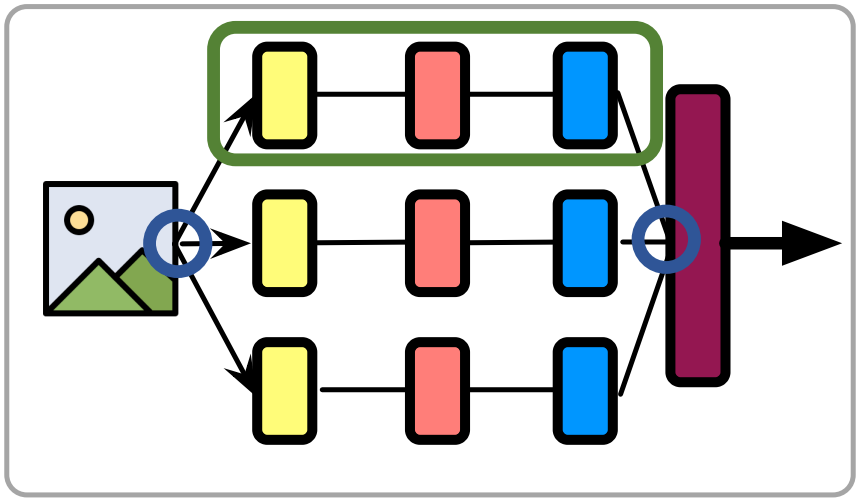
Model Parallelism

Model parallelism does not change the model

- Synchronization: Difficult to distribute
- Several Connections: High communication overhead



Low-Communication Parallelization (LCP)



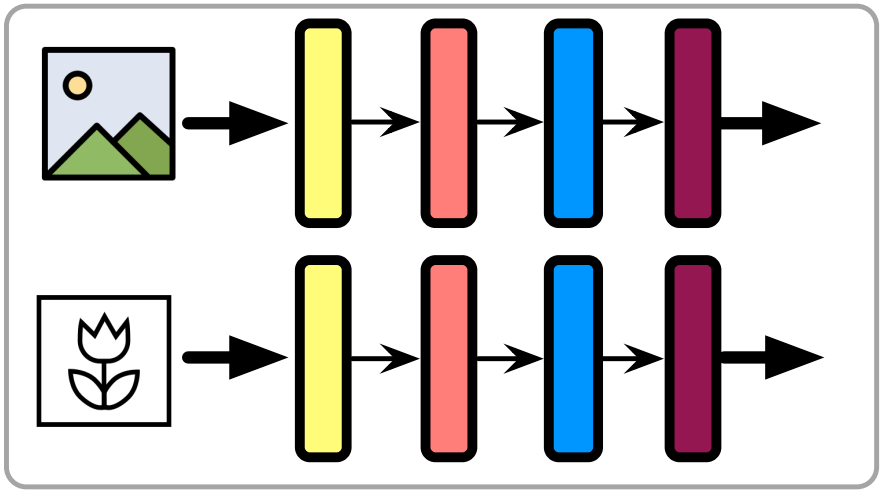
Example with 3 Splits

Two Main Benefits

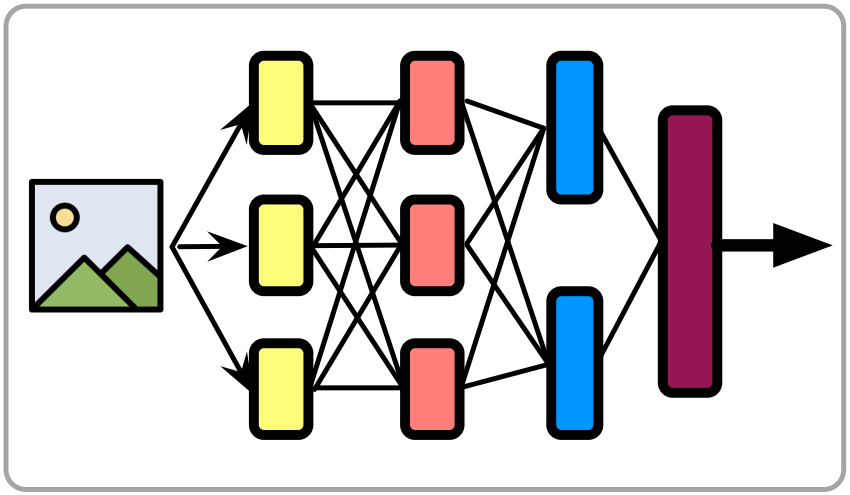
Independent & Balanced Branches → Easier to Allocate On Devices → **Better Parallelization**

Fewer Connections → Less Communication Overhead → **Faster & No Straggler**

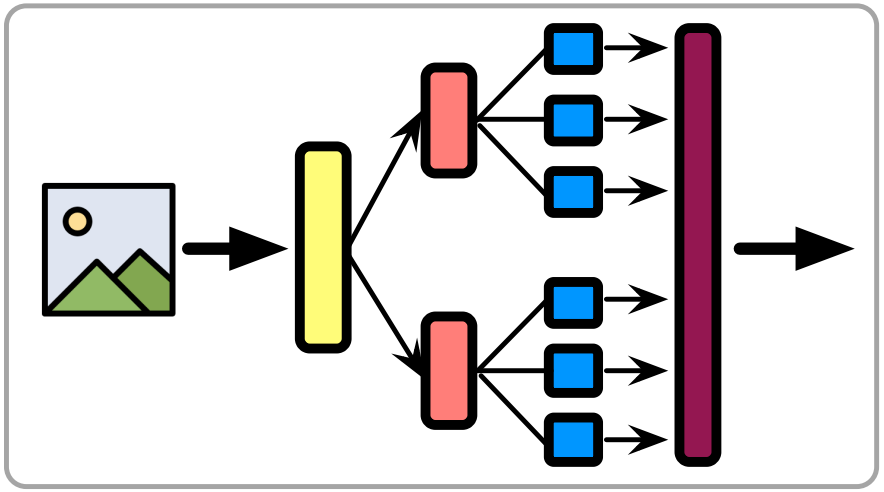
Low-Communication Parallelization (LCP)



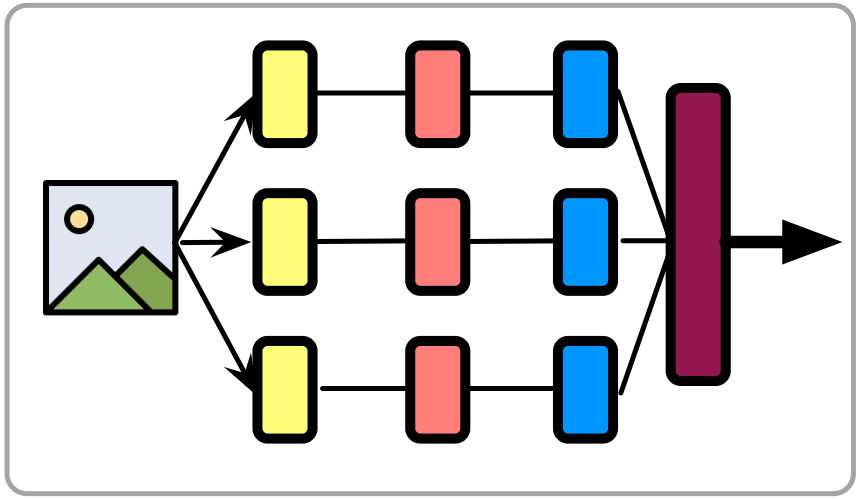
Data Parallelism



Model Parallelism



SplitNet[1]

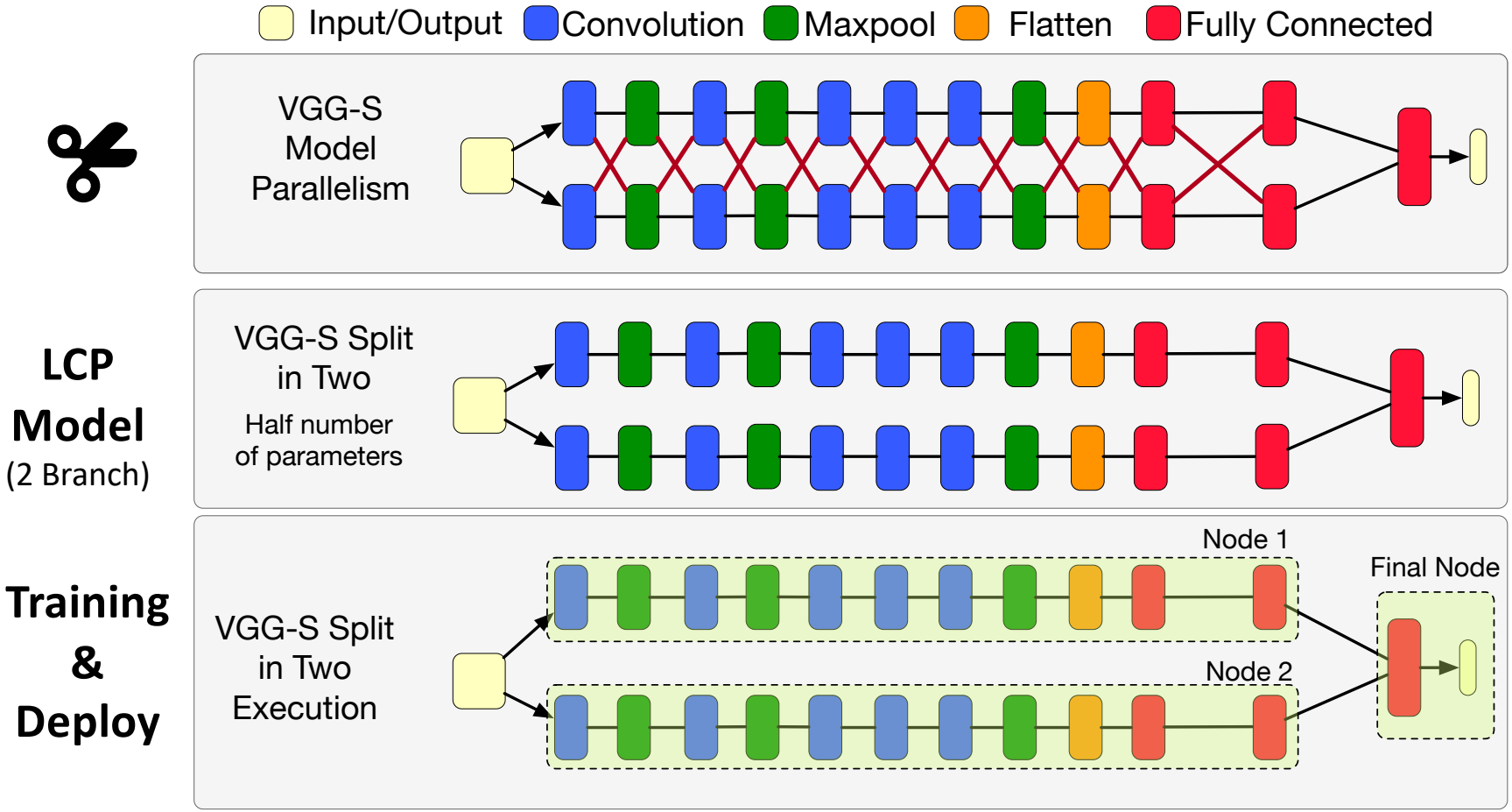


LCP

[1] Kim et al. "Splitnet: Learning to semantically split deep networks for parameter reduction and model parallelization" ICML'17

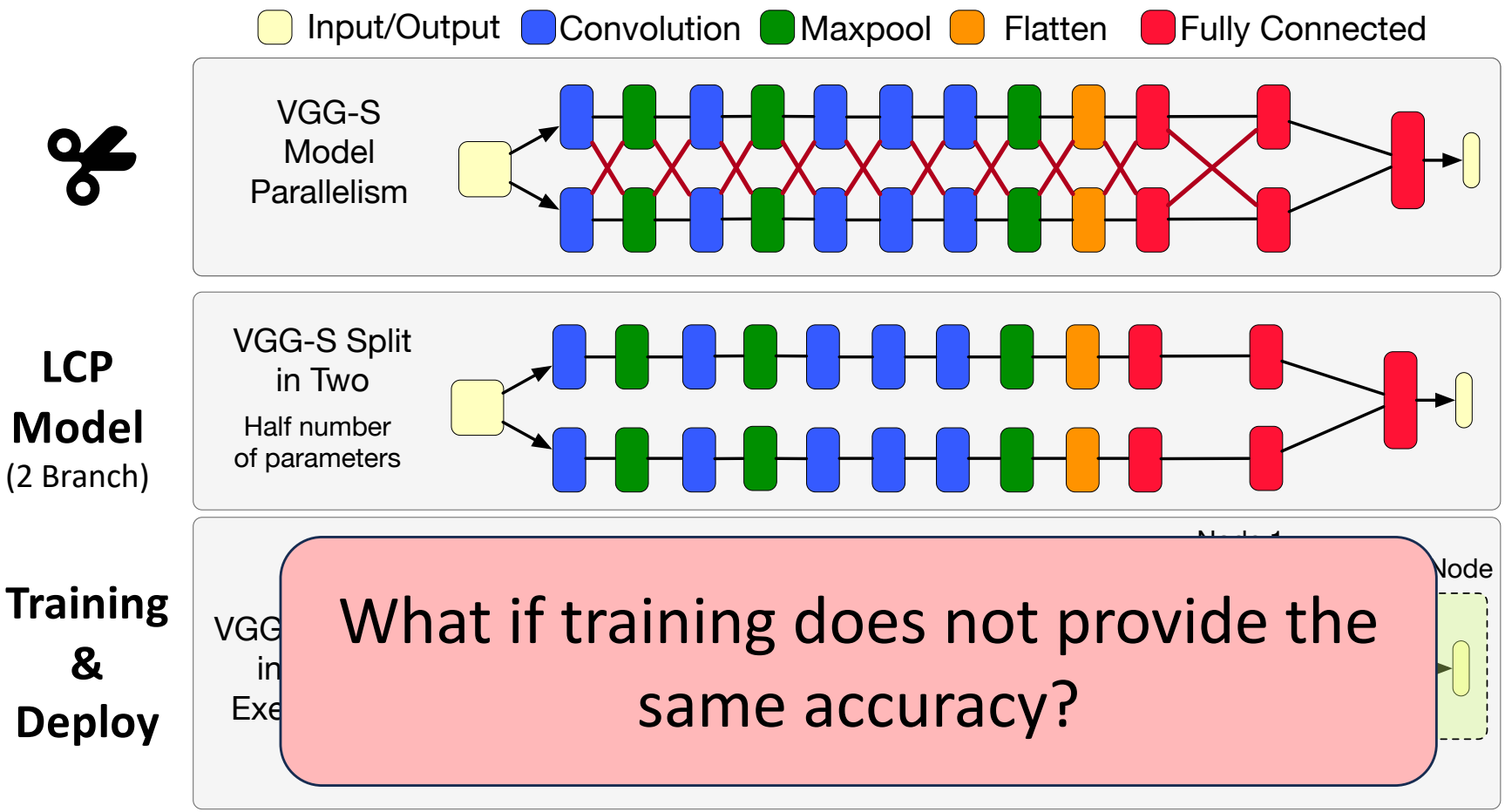
LCP Procedure

Divide a wide DNN and create branches of narrow and independent DNNs

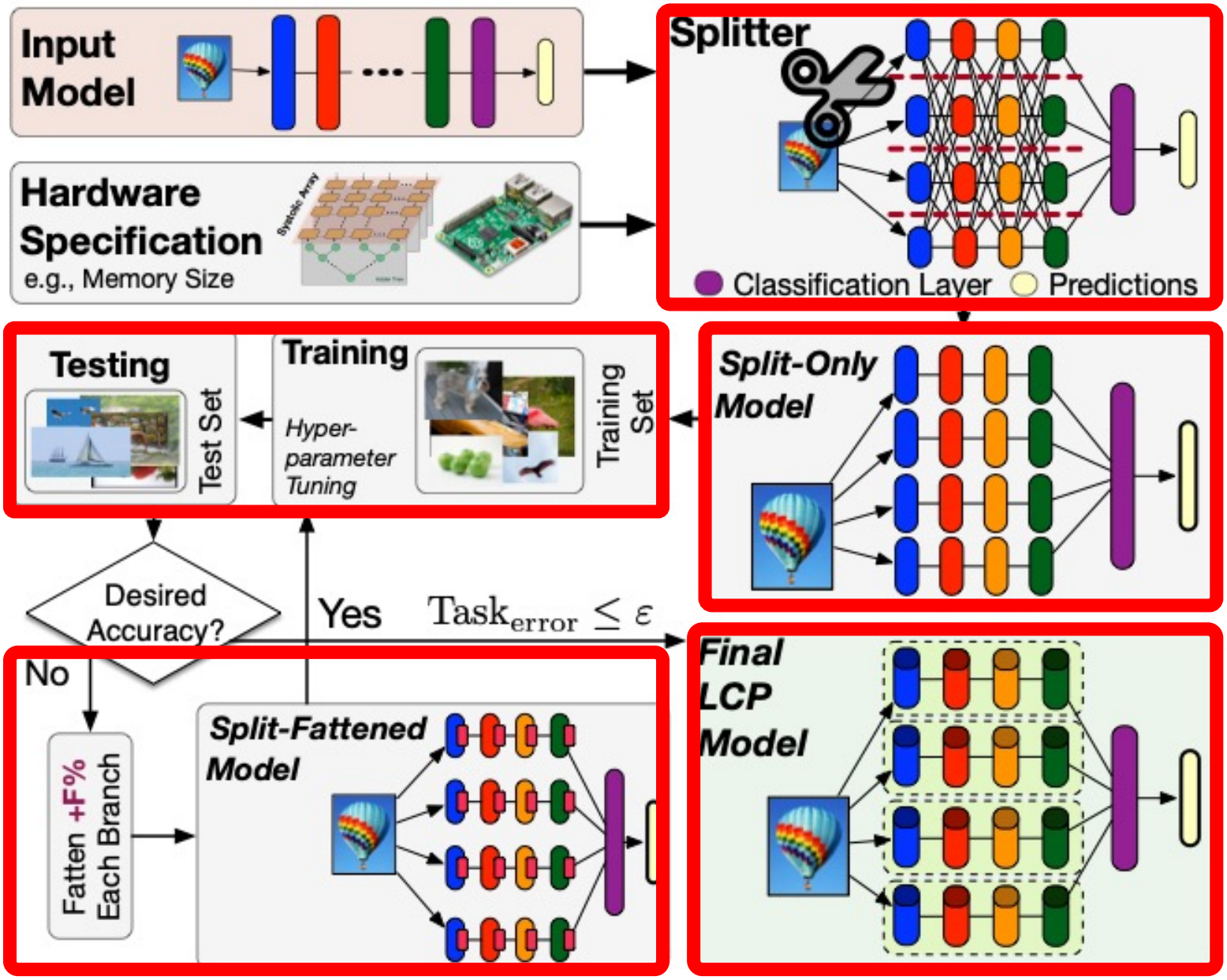


LCP Procedure

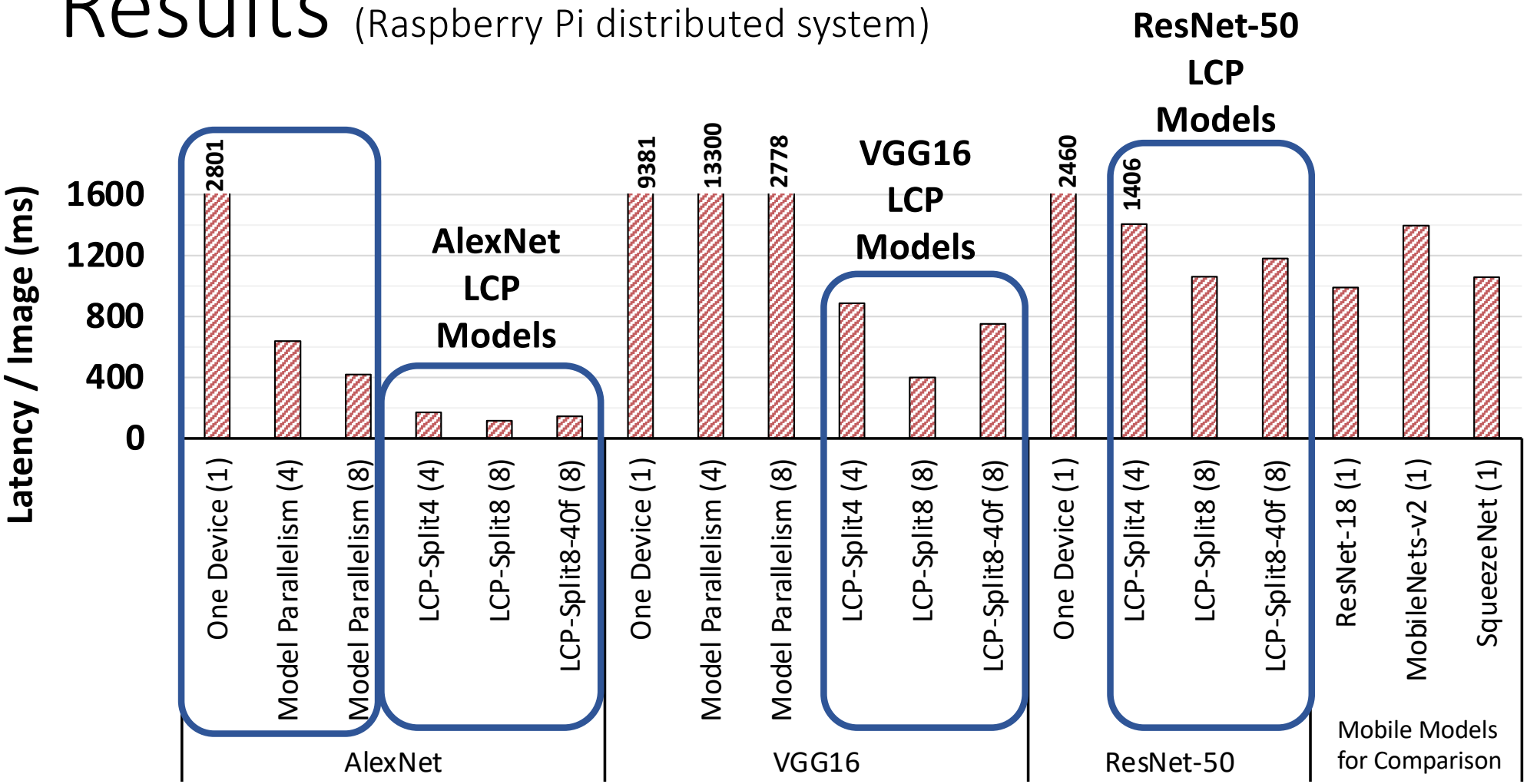
Divide a wide DNN and create branches of narrow and independent DNNs



LCP Procedure



Results (Raspberry Pi distributed system)

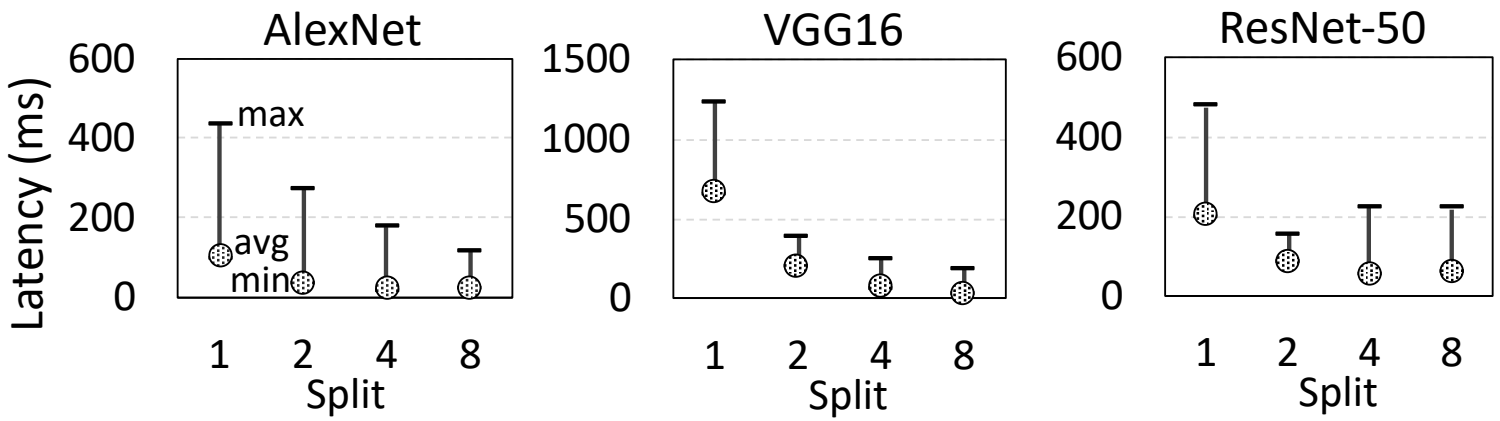


Accuracy (%)	57	55	70	67	75	72	70	72	57

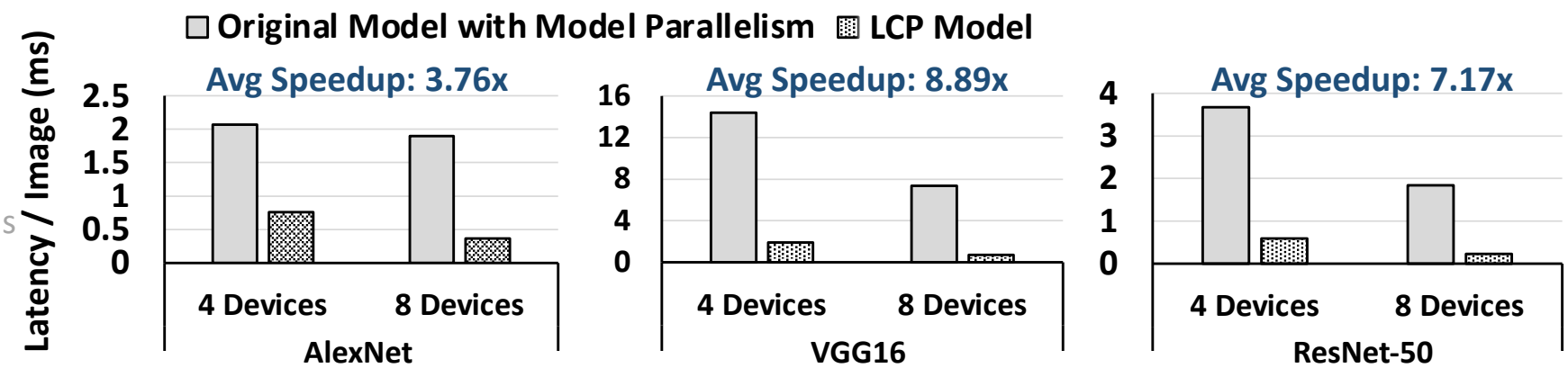
(*) Number in parenthesis shows number of deployed Raspberry Pis

Results (Latency on AWS and FPGAs)

AWS T2.micro Experiments



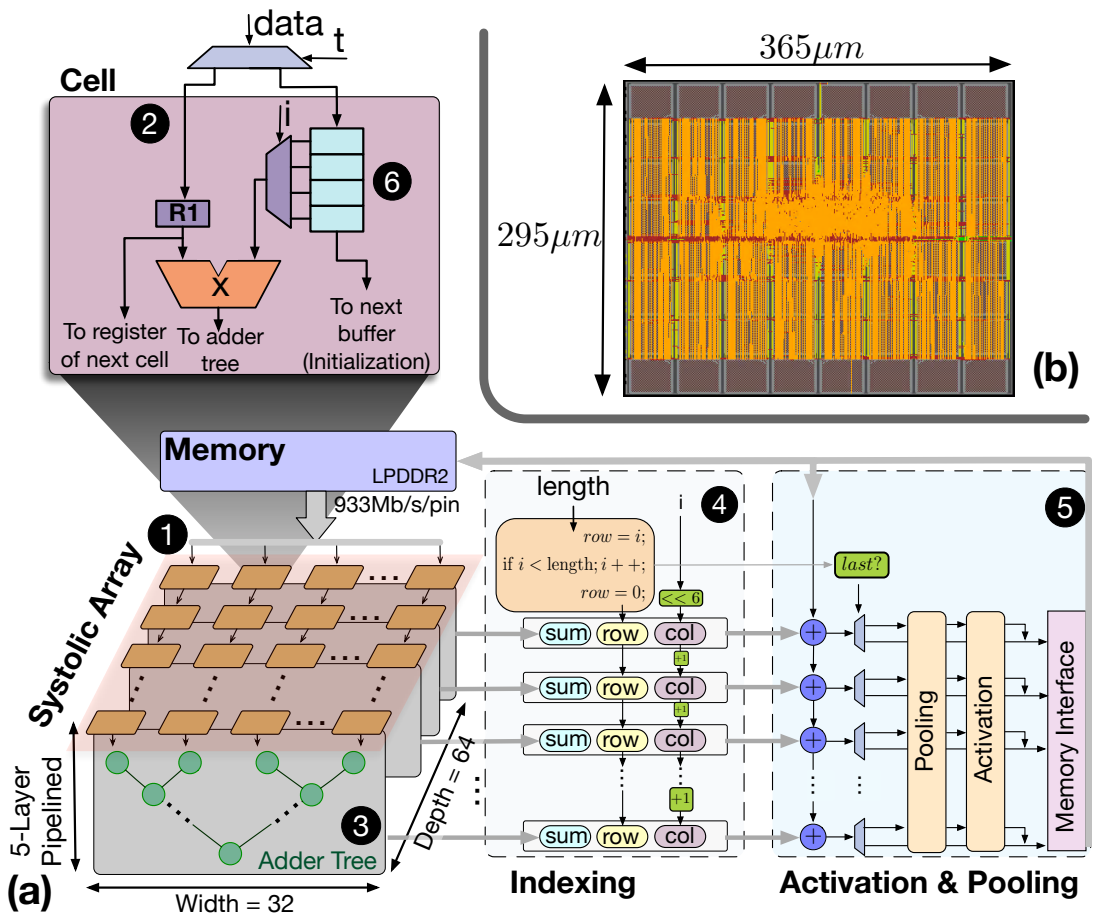
FPGA Implementations



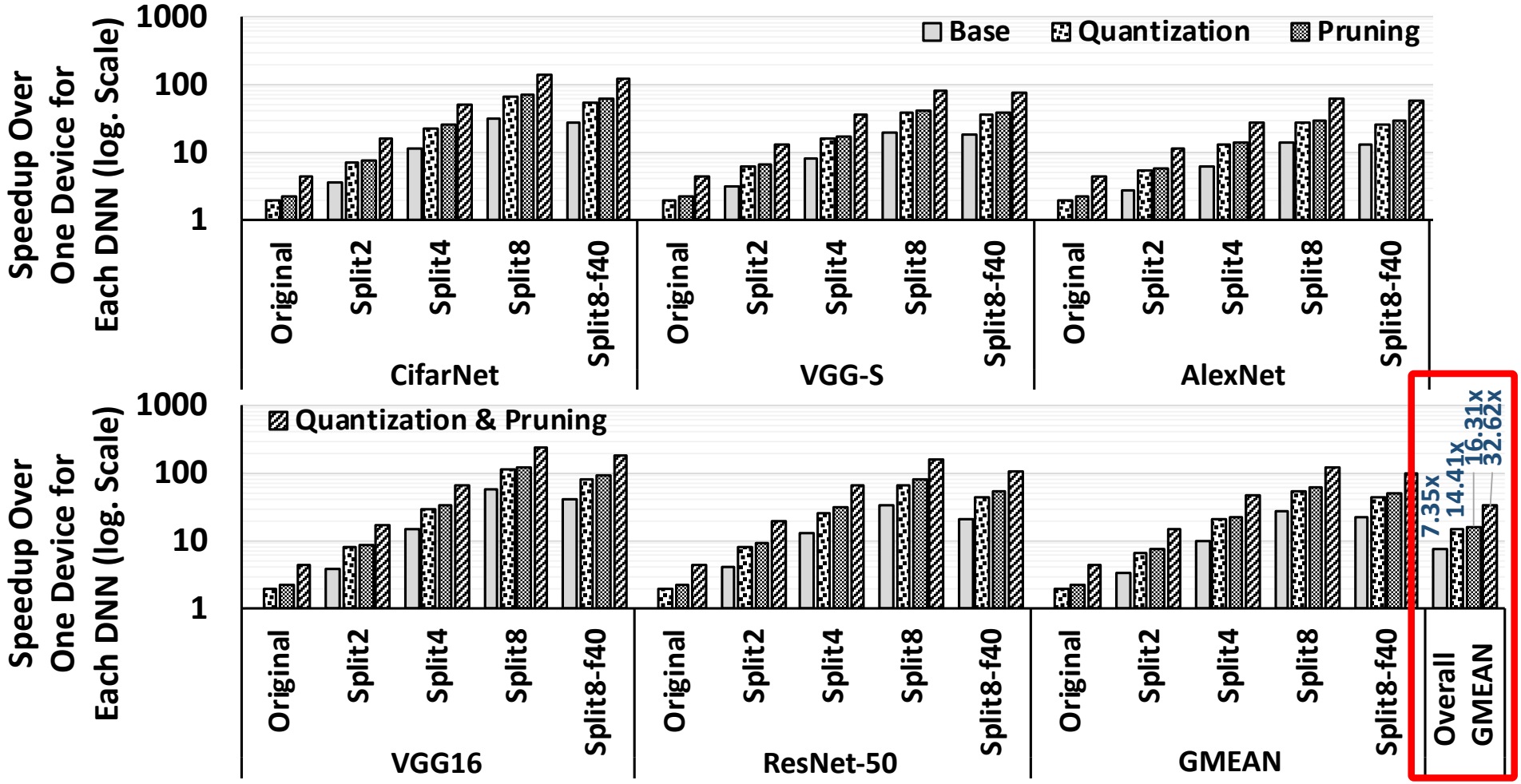
Edge Hardware Accelerator

We complimented LCP with a **16mW 7nm** custom ASIC for low footprint, energy and fast inferencing

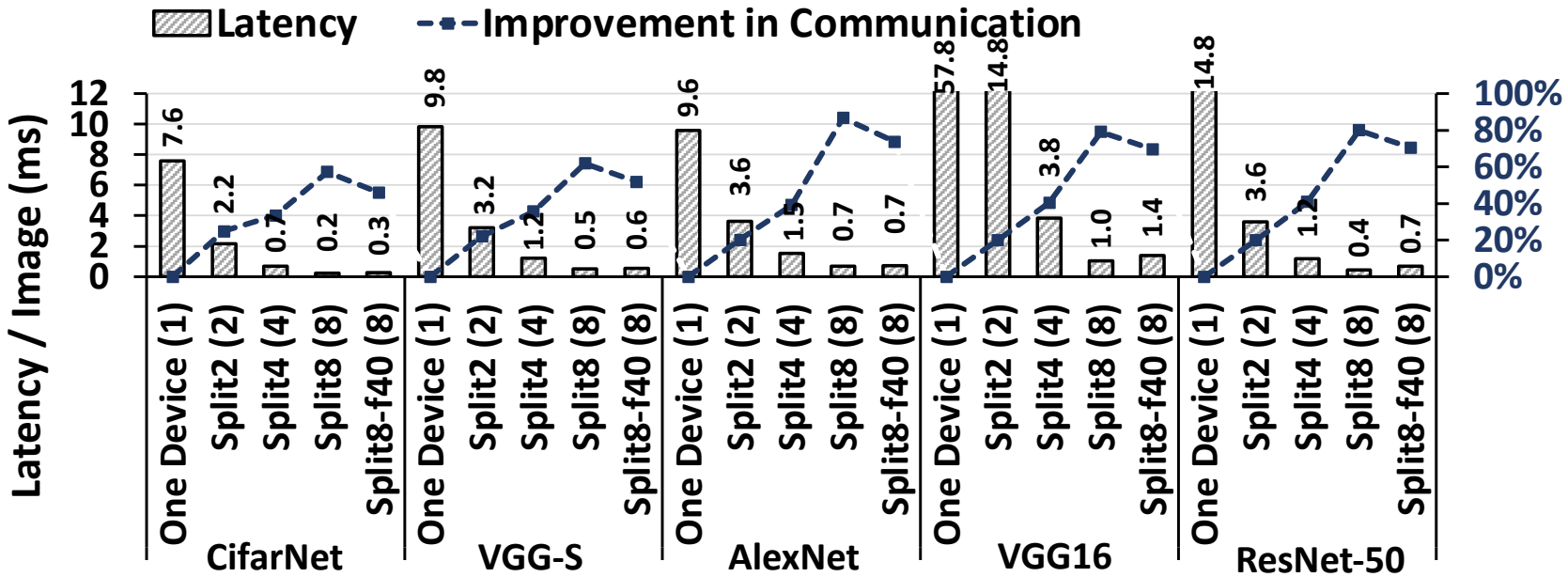
- GEMM based design
- Adder trees instead of complex data routings
- Simple indexing logic instead of complex instruction decoding
- Direct memory streaming instead of levels of caches

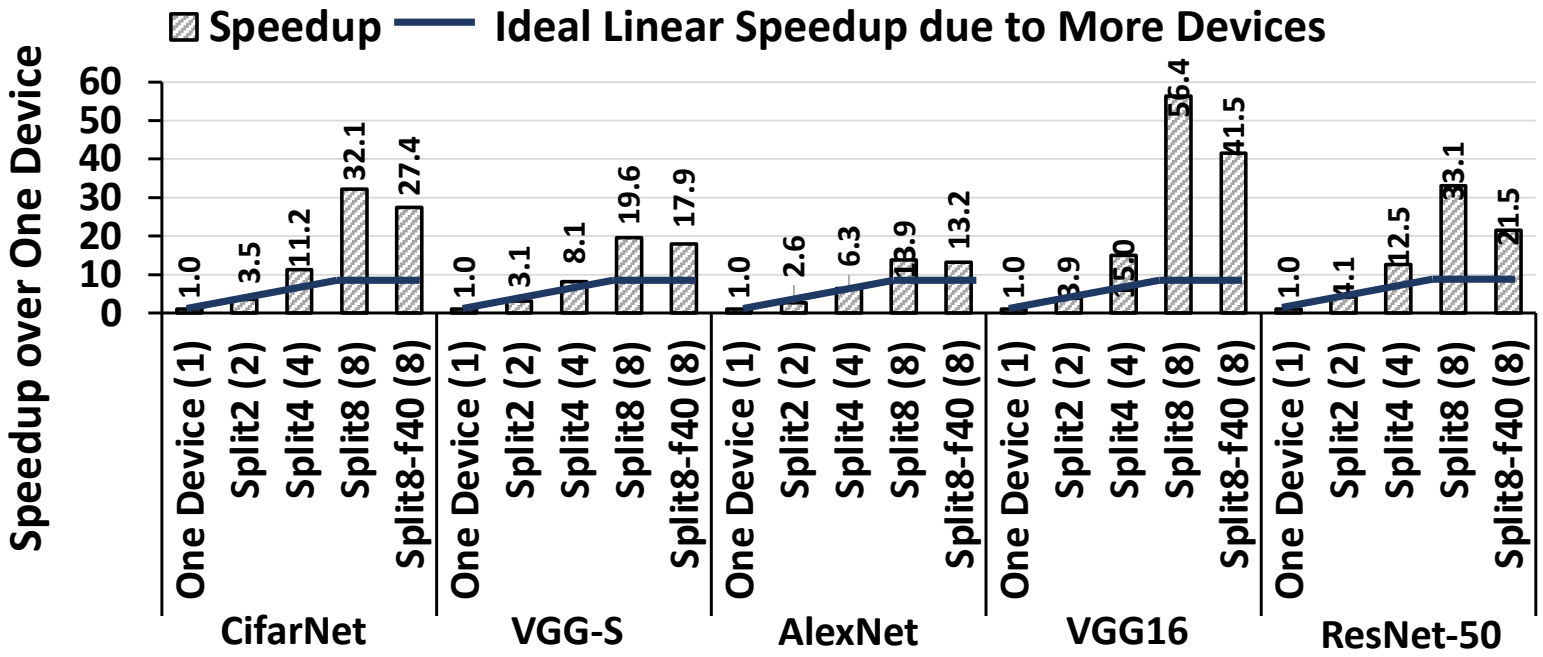


Results (With LCP HW Design on FPGA + Lossless Quantization & Pruning)



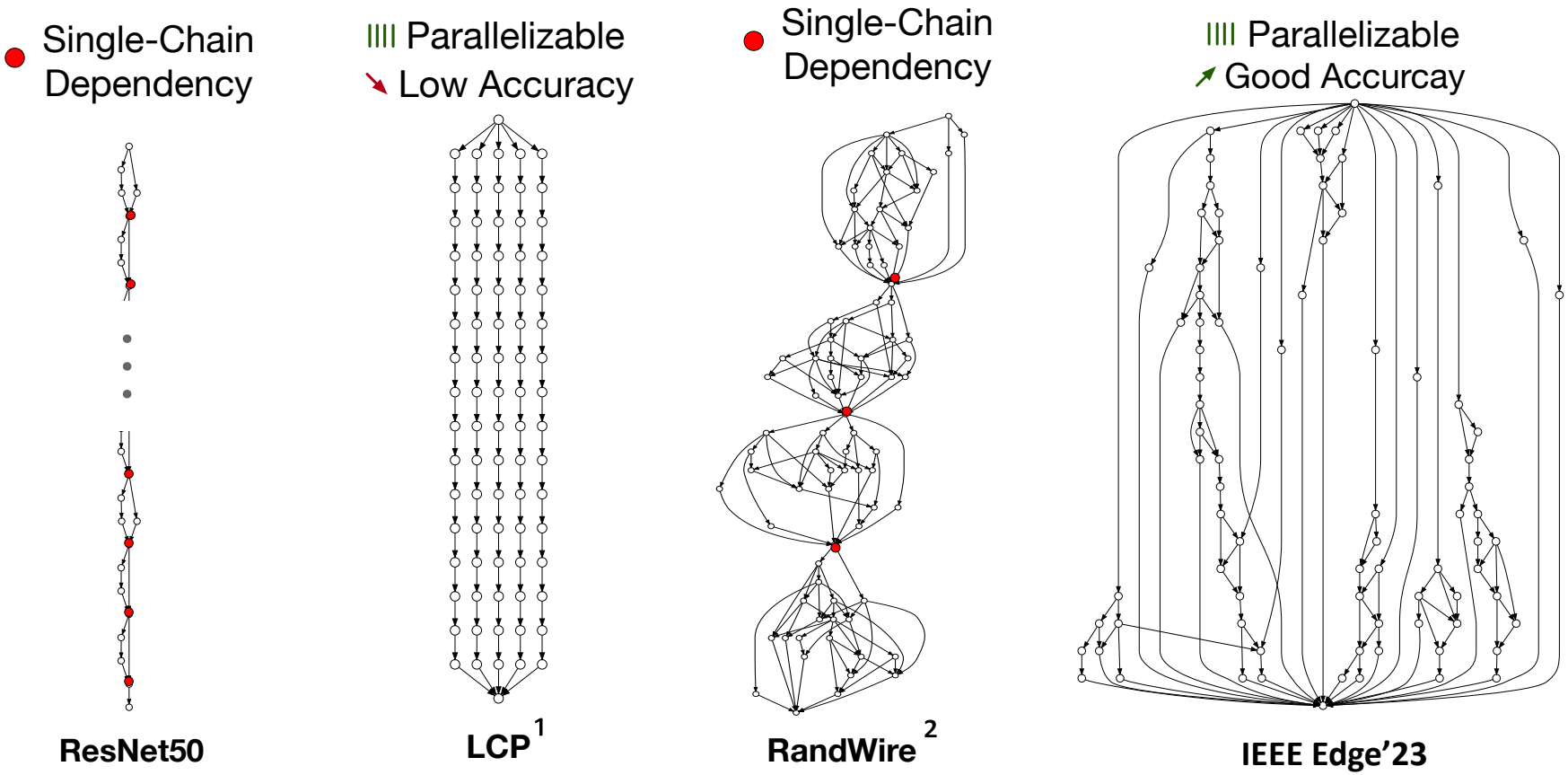
Results (With LCP HW Design on FPGA)





Follow Up Paper

In the next paper we **search** for models efficient for distribution, while providing a good accuracy!



[1] **Hadidi** et al. "LCP: A Low-Communication Parallelization Method for Fast Neural Network Inference in Image Recognition." *This work CSCE 2023*
[2] **Hadidi** et al. "Reducing Inference Latency with Concurrent Architectures for Image Recognition at Edge." IEEE Edge 2023

Please check the follow up paper at IEEE Edge 2023 with new concurrent models

Reducing Inference Latency with Concurrent Architectures for Image Recognition at Edge

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

Jiashen Cao[§] — *Georgia Tech* — jiashenc@gatech.edu

Michael S. Ryoo — *Stony Brook University and Google* — mryoo@cs.stonybrook.edu

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

***Abstract*—Satisfying the high computation demand of modern deep learning architectures is challenging for achieving low inference latency. The current approaches in decreasing latency only increase parallelism within a layer. This is because architectures typically capture a single-chain dependency pattern that prevents efficient distribution with a higher concurrency (*i.e.*, simultaneous execution of one inference among devices). Such single-chain dependencies are so widespread that even implicitly biases recent neural architecture search (NAS) studies. In this visionary paper, we draw attention to an entirely new space of NAS that relaxes the single-chain dependency to provide higher concurrency and distribution opportunities. To quantitatively compare these architectures, we propose a score that encapsulates crucial metrics such as communication, concurrency, and load balancing. Additionally, we propose a new generator and transformation block that consistently deliver superior architectures compared to current state-of-the-art methods. Finally, our preliminary results show that these new architectures reduce the inference latency and deserve more attention.**

***Index Terms*—Edge AI, Neural Architecture Search, Distributed and Collaborative Edge Computing, IoT, Collaborative Edge & Robotics**

I. Introduction & Motivation

Increasingly deeper and wider convolution/deep neural networks (CNN/DNN) [1]–[3] with higher computation demands are continuously attaining higher accuracies. Nevertheless, the high computation and memory demands of these DNNs hinder achieving low inference latency [4]. Although current platforms exploit parallelism, we discover that, since most architectures capture a *single-chain dependency pattern* [5]–[7], shown in Figures 1a & b, we cannot efficiently extend concurrency and distribution beyond current explicit parallelism exposed within intra-layer computations (*i.e.*, matrix-matrix multiplications) to reduce the latency of an inference. In other words, distribution and concurrency, if any, are implemented at data level [8], which only increases the throughput.

The status quo approaches in reducing the inference latency are always applied *after* an architecture is defined (*e.g.*, reducing parameters with weight pruning [9], [10] or reducing computation with quantization or compression [11]–[13]). Additionally, for extremely large architectures, limited model

This work was partially supported by the NSF grant number 2103951 and Institute of Information and Communications Technology Planning and Evaluation grant funded by the Korea government (No. 2021-0-00766).

[§]Equal contribution

[*]This work was done when the author was affiliated with Georgia Tech.

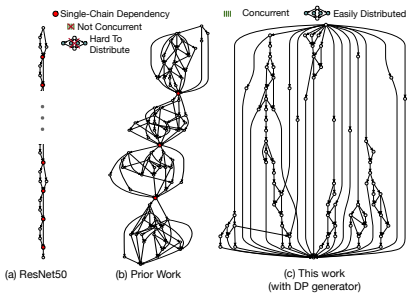


Fig. 1: **Sampled Architectures Overview** – (a) & (b) Limited concurrency and distribution due to single-chain dependency. (c) Improved concurrent architecture.

parallelism is applied on final layers (*i.e.*, large fully-connected layers that do not fit in the memory of edge devices [14]–[16]). However, since model-parallelism methods do not change the architecture, distributing all layers with such methods adds several synchronization/merging points, incurring high communication overheads (Figure 1a & b). We discover that the single-chain inter-layer dependency pattern, common in all the well-known architectures and even in state-of-the-art neural architecture search (NAS) studies [17], prevents the efficient model distribution for reducing inference latency.

This visionary paper addresses the single-chain data dependency in current architecture designs and endeavors to inspire discussion for new concurrent architectures for at-edge distribution. To do so, first, we analyze architectures generated by recent unbiased NAS studies [17] and discover that *scaling/staging* blocks implicitly enforce dependencies. Then, we generate new architectures with prior and our new distance-based network generators using our new probabilistic scaling block. Then, for quantitatively comparing generated architectures, we propose a *concurrency score* that encapsulates important metrics such as communication, load balancing, and overlapped computations, by reformulating the problem as a hypergraph partitioning problem [18], [19]. Based on the scores and experiments, our generated architectures have higher concurrency and are more efficient for distribution

Please check the paper for more details on

- Training
- LCP Splitter Procedure
- Chip layout and design
- More experiment results

LCP: A Low-Communication Parallelization Method for Fast Neural Network Inference for IoT

Ramyad Hadidi*‖, Bahar Asgari‡‖ Jiashen Cao†, Younmin Bae†, Da Eun Shim†, Hyojong Kim†, Sung-Kyu Lim†, Michael S. Ryoo§, Hyesoon Kim†

*Rain AI, †Georgia Tech, ‡University of Maryland College Park, §Google & Stony Brook University

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

***Abstract*—Deep neural networks (DNNs) have stimulated research in diverse edge applications including robotics and Internet-of-Things (IoT) devices. However, IoT-based DNN inference poses significant challenges due to resource constraints. Further, as communication is costly, taking advantage of other available IoT devices by using data- or model-parallelism methods is not an effective solution. We introduce a low-communication parallelization (LCP) method to minimize communication overhead in distributed IoT systems. LCP models consist of multiple, largely-independent, narrow branches, providing enhanced distribution and parallelization opportunities while reducing memory and computational requirements. Implemented on AWS instances, Raspberry Pis, and PYNQ boards, as well as a customized 16mW 0.107mm^2 ASIC @7nm chip, LCP models yield maximum and average speedups of 56x and 7x, compared to original models, which could be improved by incorporating common optimizations such as pruning and quantization.**

***Keywords*– IoT, DNN, Inference, Parallel, Distributed, FPGA**

I. Introduction & Motivation

Deep neural networks (DNNs) have revolutionized many fields including Internet-of-things (IoT) systems. Yet, executing computationally heavy DNN inference locally in isolated networks (*e.g.*, smart homes, drones [1]) remains a challenge [2]. In these cases, acceptable accuracy, standalone operation, and unified ownership are key. The conventional solution to heavy DNN inference computations is cloud-based offloading. However, this approach has limitations: unavailability (*e.g.*, no Internet access), reliance on variable network latency, and scalability issues. Also, privacy concerns and personalization push for local inference. However, local inference demands high resources, clashing with the energy and computational constraints in IoT devices.

The Current Approach & Key Challenge: Existing methods enable local DNN inference by distributing computations among idle IoT devices using data- or model-parallelism. Data parallelism improves throughput by duplicating the model on each device for separate inferences, but requires concurrent inputs. Model parallelism distributes the model across devices for the same inference, but is limited by communication overhead and inter-layer data dependencies. An ideal method for IoT devices should minimize communication overhead and memory and computation requirements per node, but no existing distribution methods achieve all these goals.

Our Solution: To address the aforementioned challenge, we propose a low-communication parallelization (LCP) method that enables the following: *(i) Reduces Communication:* LCP models replace a wide, deep model with several narrow ones, reducing communication requirements as they only communicate for input and pre-final activations (see Table I). *(ii) Lowers Compute & Memory Footprints:* Fewer connections in LCP models lead to fewer parameters and lower computational demands compared to model-parallelism counterparts (Table I). *(iii) Enables Inter-Layer Parallelism:* The independent narrow branches in LCP models allow for inter-layer parallelism, unlike model parallelism which is restricted by inter-layer dependencies. *(iv) Recovers Accuracy Without Extra Parameters:* Any potential accuracy loss due to model splitting can be recovered by slightly increasing the branch size, but this still results in fewer overall parameters due to the reduction in unnecessary communication. LCP operates in conjunction with existing techniques like weight pruning and quantization that decrease model computation demands. LCP facilitates model distribution and parallelism in distributed systems, while other techniques implement accuracy/performance tradeoffs on individual nodes. These approaches can be applied to each branch in our method (see §IV-C), meaning LCP complements them.

TABLE I
Methods for distributing inference computations.

	Data Parallelism	Model Parallelism	Target	LCP
Memory Per Device	DNN	$\frac{1}{n}$DNN	$\frac{1}{n}$DNN	$\leq \frac{1}{n}$DNN
Communication Per Inference	IN/OUT	Intermediates +IN/OUT	IN/OUT	$\approx$ IN/OUT
Computation Per Device	DNN	$\frac{1}{n}$DNN	$\frac{1}{n}$DNN	$\leq \frac{1}{n}$DNN

DNN: Metrics associated with the entire model; n: Number of devices.

Experiments Overview: (1) We create and evaluate LCP models using image-recognition DNNs on various datasets (MNIST, CIFAR10/100, Flower102, and ImageNet) including all MLPerf image-recognition models, resulting in a total of 53 training evaluations. **(2)** We implement our method on three different distributed systems: a network of up to 10 Raspberry Pis (RPis), two PYNQ boards, and up to eight AWS instances, using RPis due to their widespread use in IoT applications. **(3)** We assess the performance of LCP on customized hardware. In addition to optimizing models based on hardware constraints, we modify the TPU architecture to be latency-optimized, suitable for IoT applications, and implement it on a Xilinx FPGA. **(4)** We evaluate the area and power efficiency of our tailored hardware using ASAP 7 nm for integration into IoT.

Contributions: Our contributions are as follows:

- We propose the first DNN parallelization to reduce the communication overhead for distributed inference for IoT.
- We generate LCP models, with inter-layer parallelism for fast inference at small memory and computation footprints.
- We investigate the impact of hardware/software co-design on inference performance, by tailoring the hardware of TPU for optimizing single-batch inference latency, and implement it on a small FPGA and as a tiny 0.107mm^2 low-power chip consuming only 16mW.

LCP Splitter Procedure

Procedure 1: LCP Splitter

Input : DNN: Layer configurations $[0 : n]$
DNN_{Mem}, DNN_{MAC}: DNN memory and computational footprints
$Division_{factor}$: Division Factor for splitting
Dev_{Mem}, Dev_{MAC}: Hardware specification

Output: DNN: Layer configurations $[1 : n]$

```
Split(DNN, DNN_Mem, DNN_MAC, Division_factor, Dev_Mem, Dev_MAC)
    Mem_fit ← 0; MAC_Mac ← 0;
    while not Mem_fit and not MAC_Mac do
        Mem_fit ← DNN_Mem < Dev_Mem
        MAC_Mac ← DNN_MAC < Dev_MAC
        for layer [0..n − 1] in DNN do
            layer.width ← layer.width/ Division_factor
        RemoveNonBranchConnections(DNN)
    return <DNN>
```

Real HW Specifications

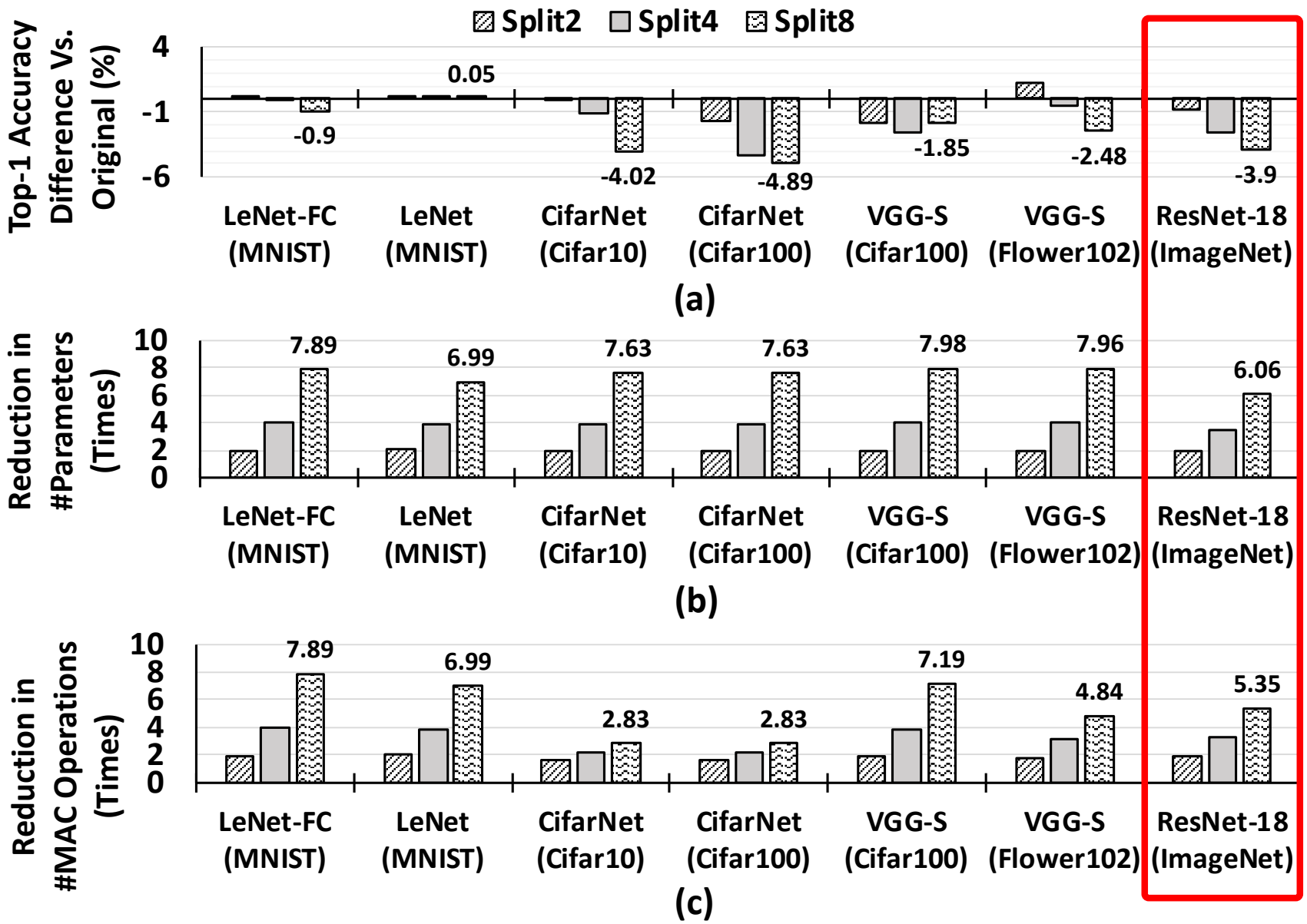
SPECIFICATION OF RPI, PYNQ FPGA, AND AWS.

Raspberry Pi 3B+	
CPU	1.2 GHz Quad Core ARM Cortex-A53
Memory	1 GB LPDDR2 SDRAM @ **933Mb/s/pin**
Die Size	$\approx 196mm^2$ @ **28 nm**

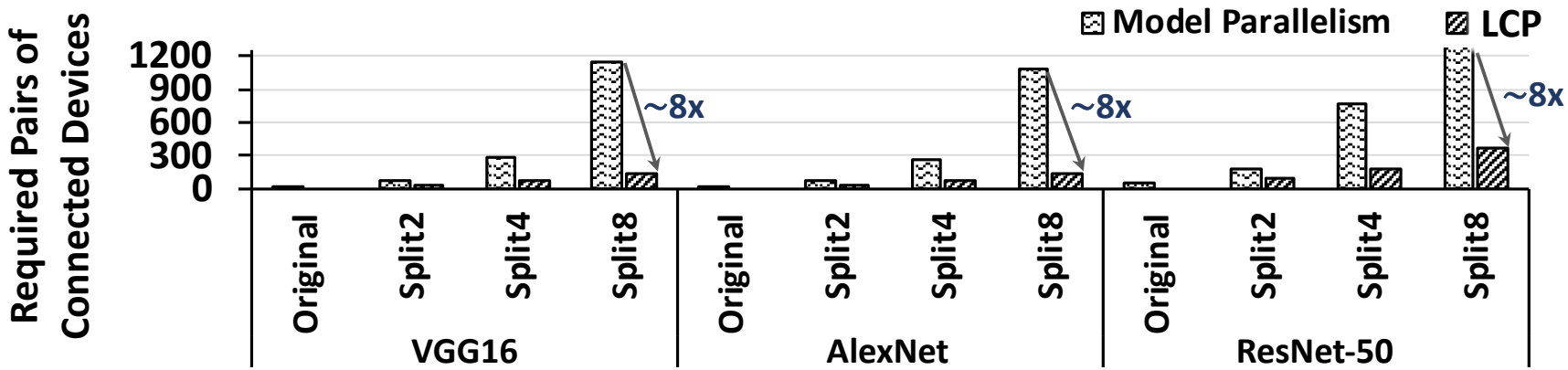
Edge FPGA (Zynq Artix 7 XC7Z020)				
Utilization		DSP48E	FF	LUT
	#Unit	96	5427	2343
	%	44	5	4
Static Power	0.121 W			
Dynamic Power	Signals: 0.009 W		Logic: 0.003 W	

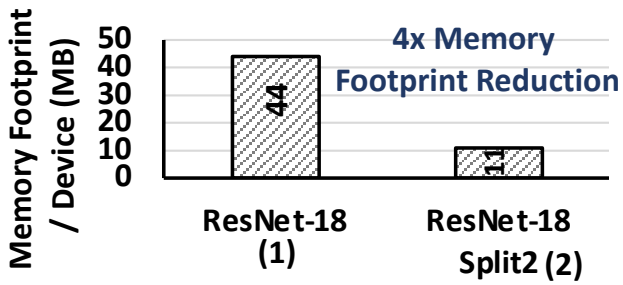
AWS	
AWS Instance Specification	T2.micro 1 vCPU, 1 GB Memory, 64 GB Storage

Results (Training)



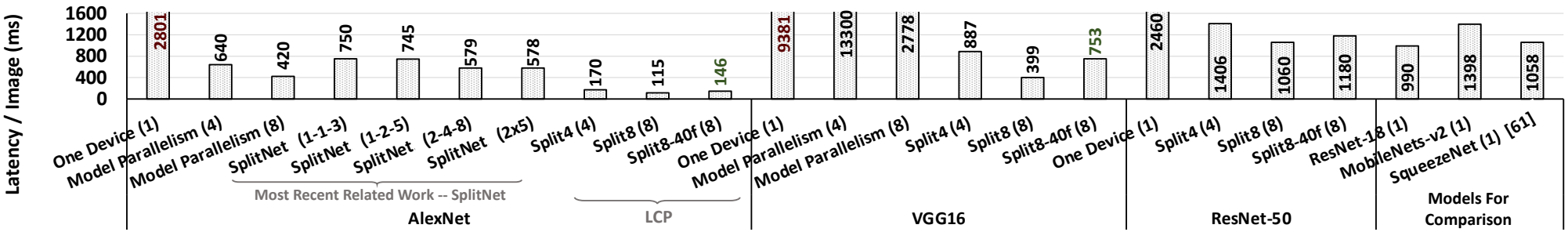
Results (Communication & Parameter)



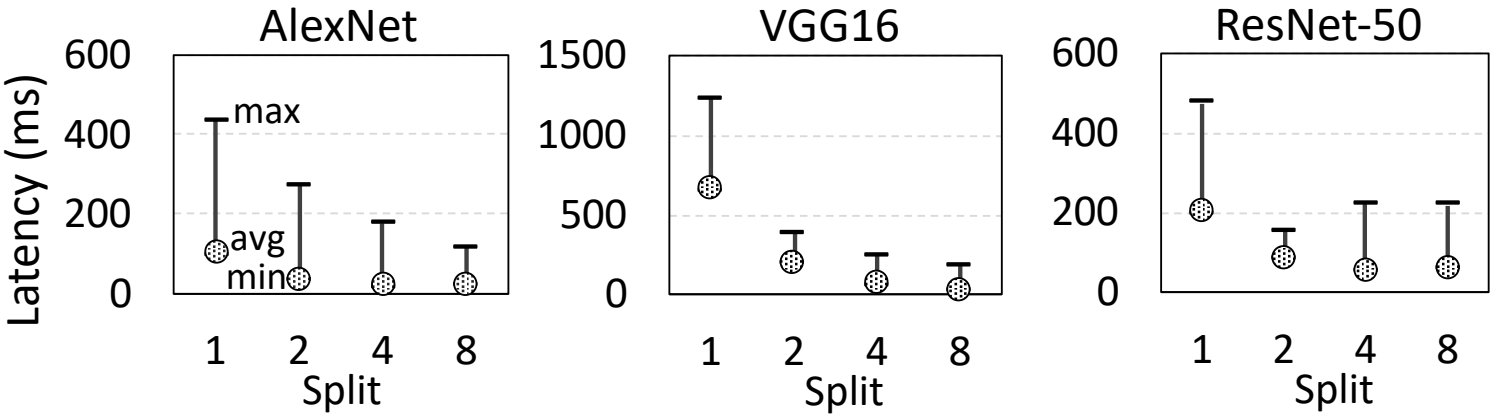


(b)

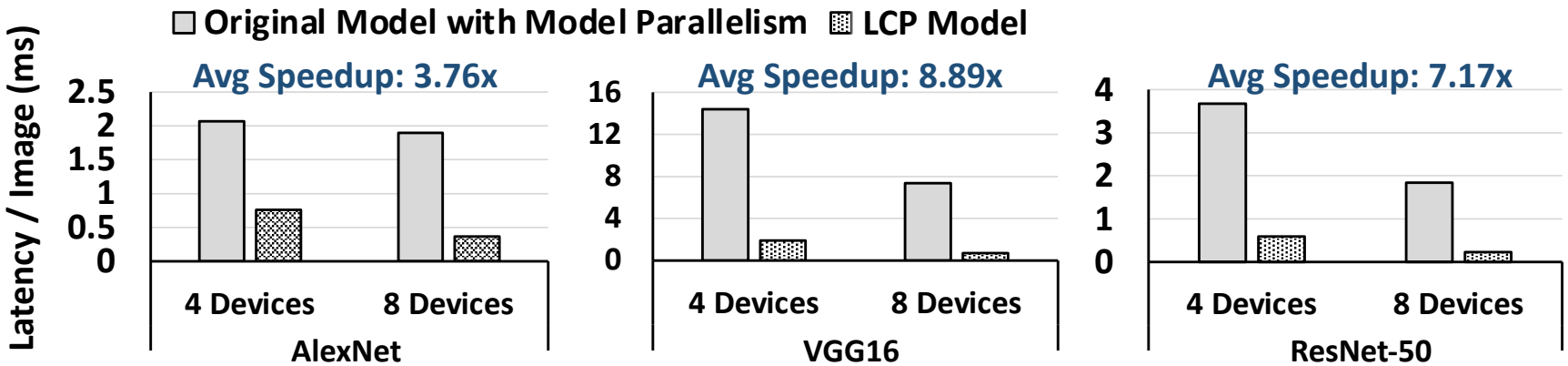
Results (Latency)



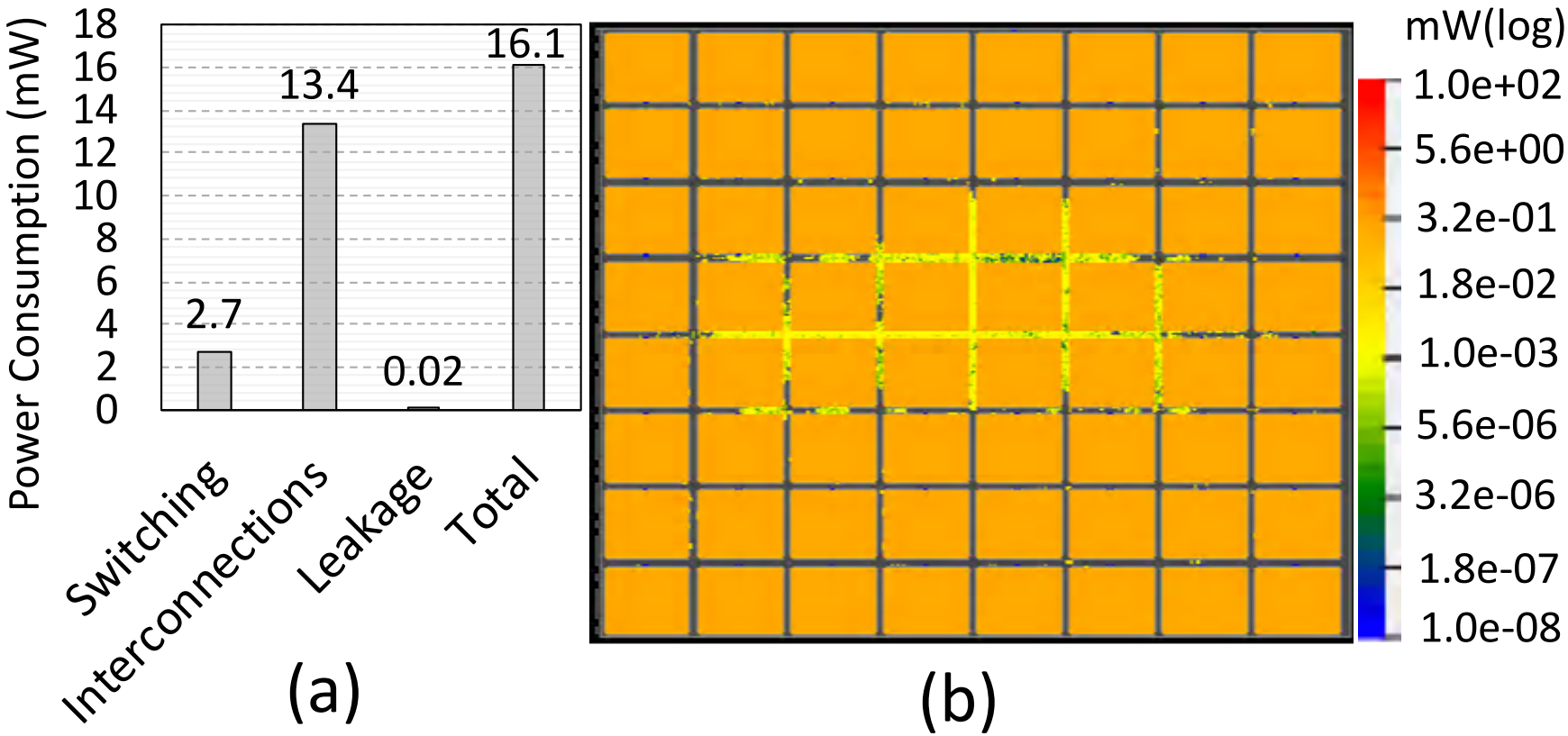
AWS Experiments



FPGA Implementations



Results (Layout & Energy)



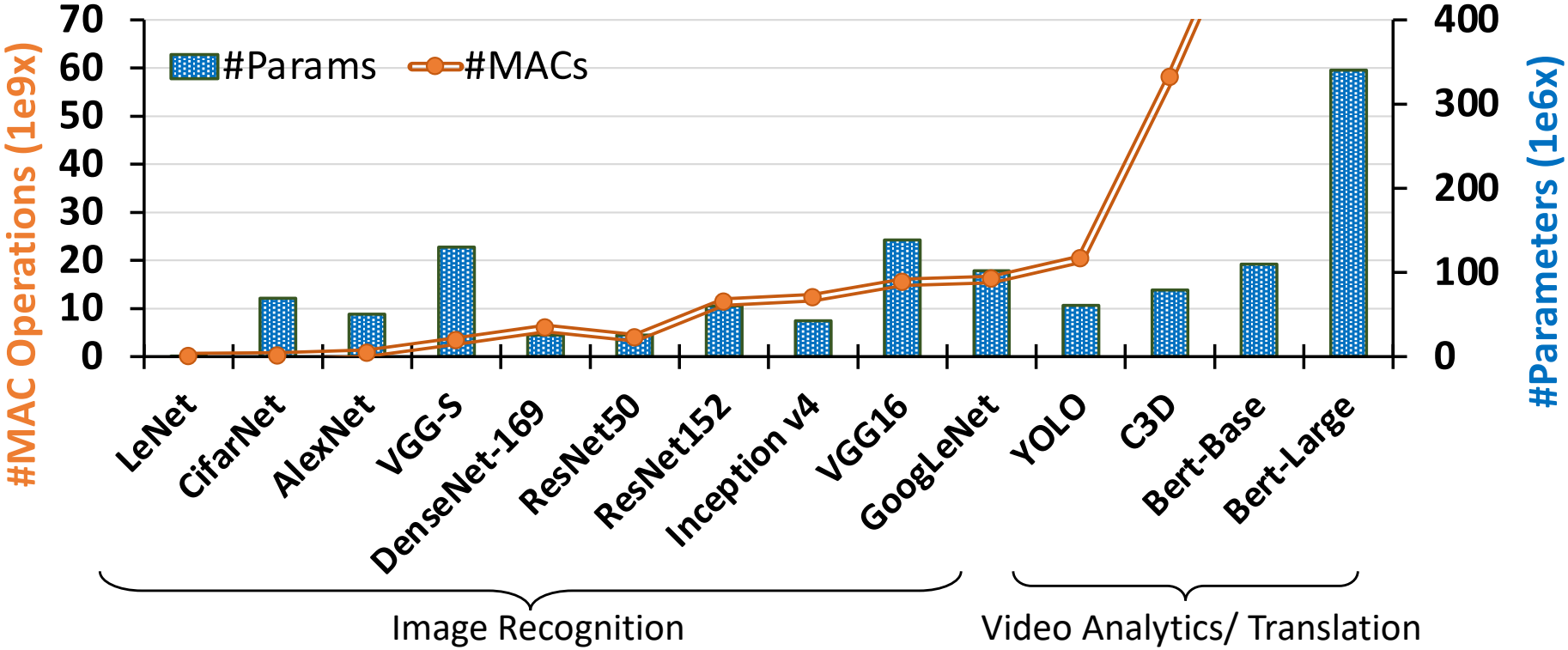
In-the-Edge Inference

- In-the-edge applications
  - Intelligence in self-driving cars, smart homes/cities
- Sometimes is the only option
  - No Internet connectivity
  - Intermittent connectivity
- Privacy preserving
  - Straightforward way to preserve privacy and security
  - Personalization
- Even faster
  - No cost associated with communication latency
- Sometimes cost($) efficient

In-the-Edge Inference (Challenge)

Edge devices cannot handle such heavy computations due to lack of resources

Newer DNNs are heavier for better understanding



Model vs. Data Parallelism

- Data Parallelism – **Throughput Oriented**
  - Requires several input
  - High computation and memory footprints per device
  - Does not break down heavy layers
  - No adjustable work per device
- Model Parallelism – **Latency Oriented**
  - Requires one input
  - Exploits parallelism within a layer
  - Breaks down heavy layers
  - Adjustable work per device