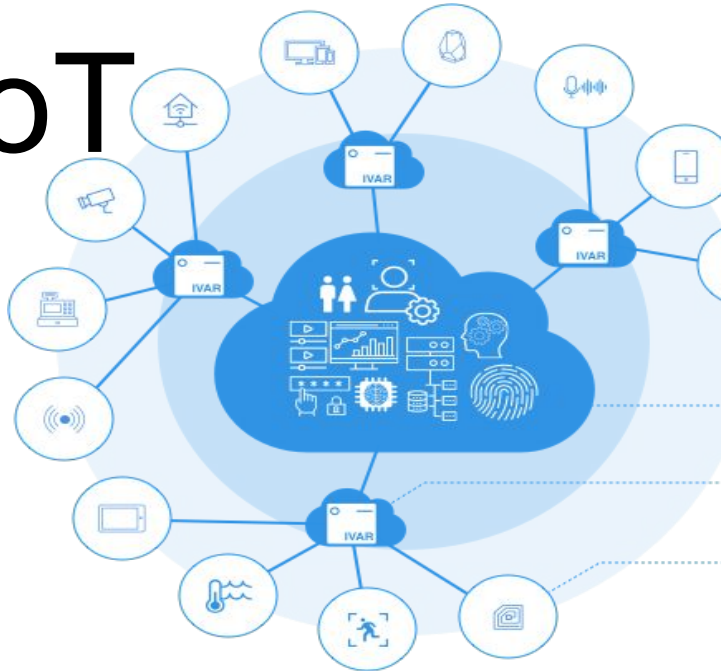
Real-Time Data Processing and AI for Distributed IoT



Group 15

Rajitha Opanayaka (E/15/246)
Rashmi Thilakarathne (E/15/363)
Amila Weerasinghe (E/15/385)

Problem

- Resource Limited IoT devices
- Latency and network congestion
- Lack of support for real time decision making on promises
- Inefficient use of cloud resources
- Privacy issues

Current Trends

- Cloud based processing
- Hardware accelerators
- Libraries ex - tensorflow

Why not tensorflow ?

- Tensorflow lite runs specific models.
- Tensorflow terminates in run time.

Tensorflow for gpu

- Tensorflow is built for CNN using cuDNN
- cuDNN is a library for deep neural nets built using CUDA
- CUDA is NVIDIA’s language/API for programming on the gpu enabled card

CPU	GPU
Around 4 cores (Raspberry Pi 3B)	128 cores(Jetson nano board)
Powerful less number of cores	Large number of weak cores
Less cost	High cost

Performance Comparison for a CNN trained on 2080 rtx GPU vs Ryzen 2700x CPU only

Package:	tensorflow 2.0	tensorflow-gpu 2.0
Total Time [sec]:	4787	745
Seconds / Epoch:	480	75
Seconds / Step:	3	0.5
CPU Utilization:	80%	60%
GPU Utilization:	1%	11%
GPU Memory Used:	0.5GB	8GB (full)

Things to consider.

Due to lesser resource utilization of cpu only tensorflow the performance was more than 6x slower.

This results were tested on Ryzen 2700x CPU which if far more capable than a single raspberry pi quad core processor which even can’t run the large CNN

Our goal

To enable real time computations at the edge,
By utilizing the low cost devices (non gpu based) in a distributed architecture using FOG and ROOF.

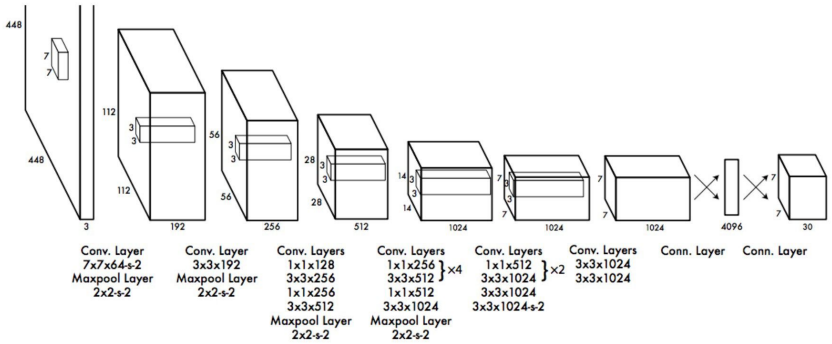
YOLO

Why

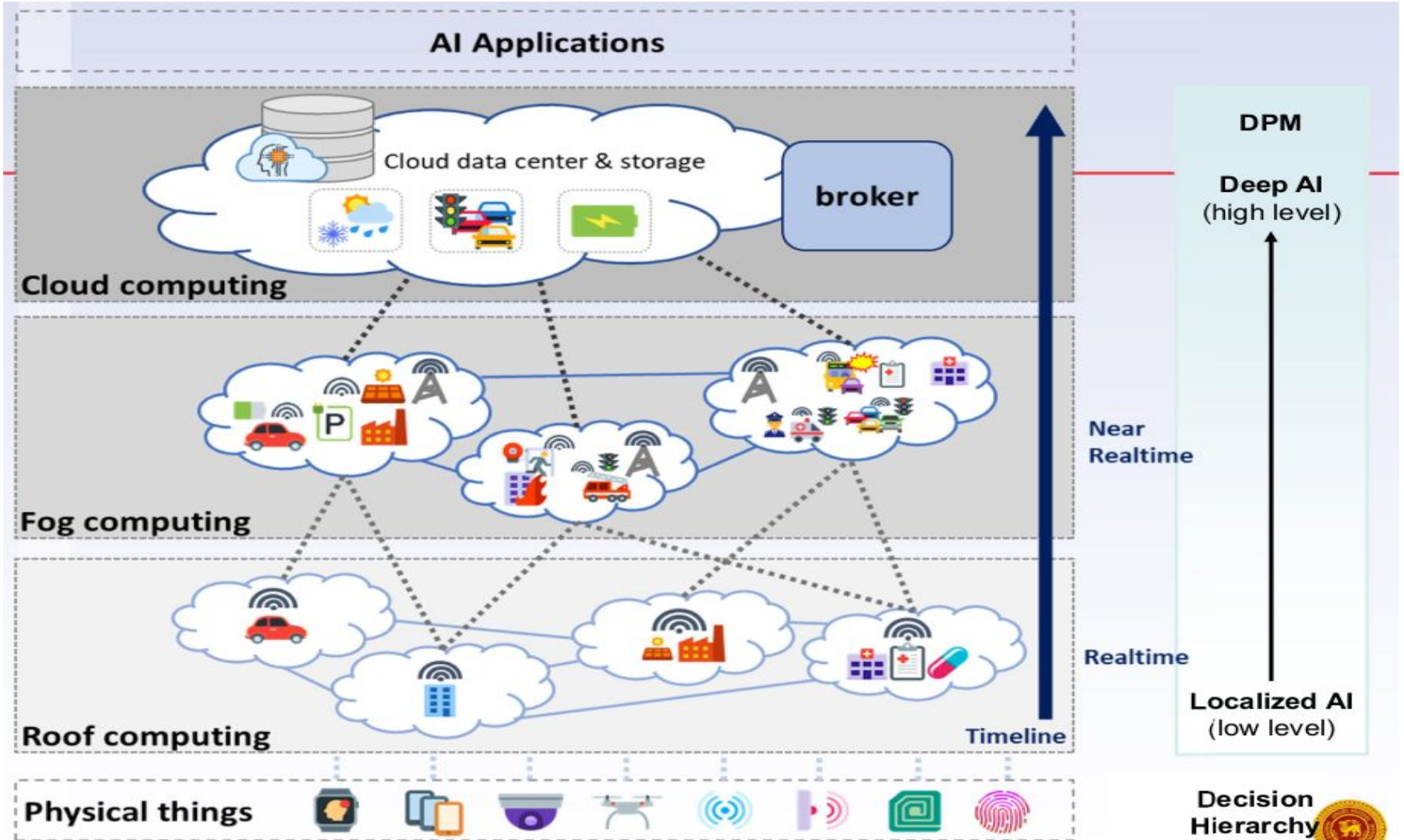
- Fast
- High accuracy

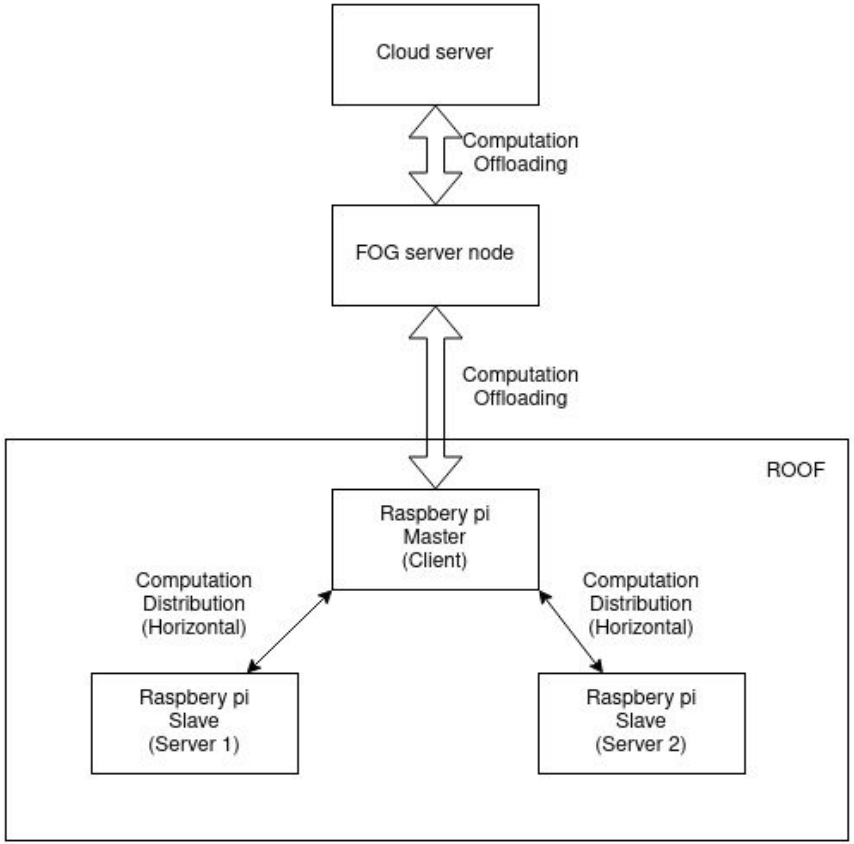
Features

- CNN
- IoU
- Non-max Suppression



Solution

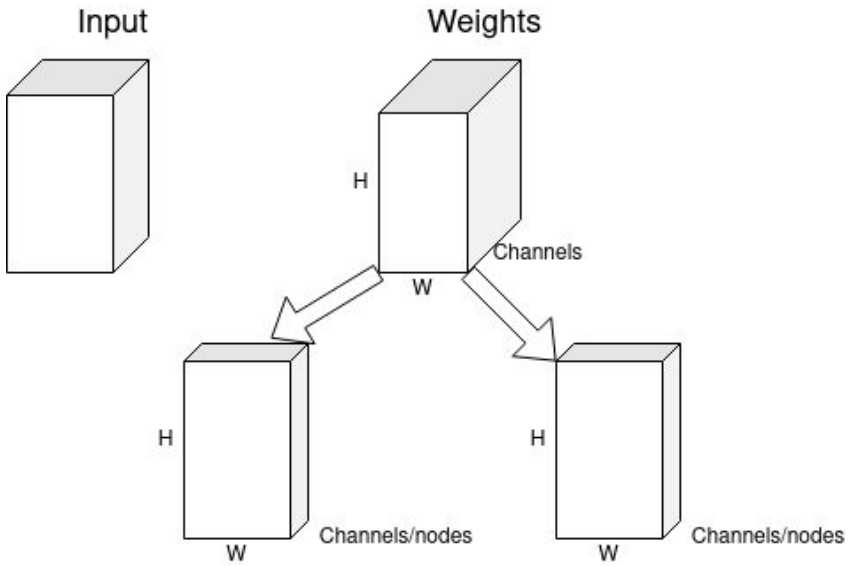

AI Applications
Cloud data center & storage
broker
Cloud computing
Fog computing
Roof computing
Physical things
Timeline
DPM
Deep AI
(high level)
Near
Realtime
Realtime
Localized AI
(low level)
Decision
Hierarchy



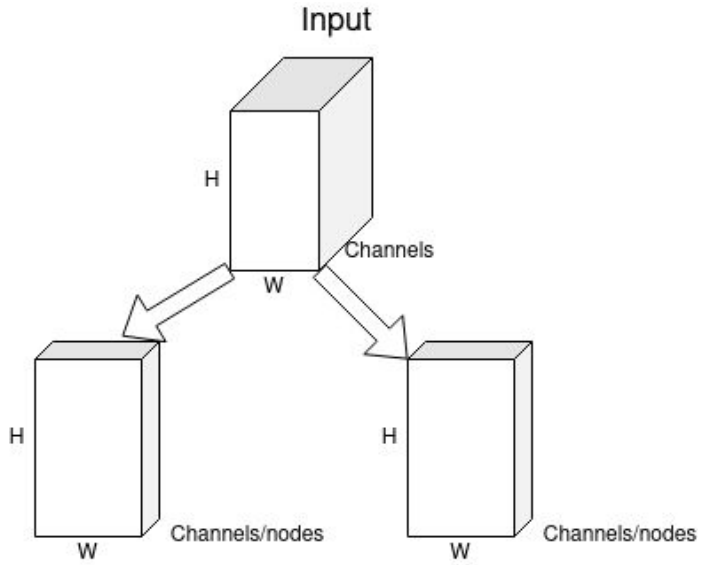
Architecture of the solution.

Computation Distribution

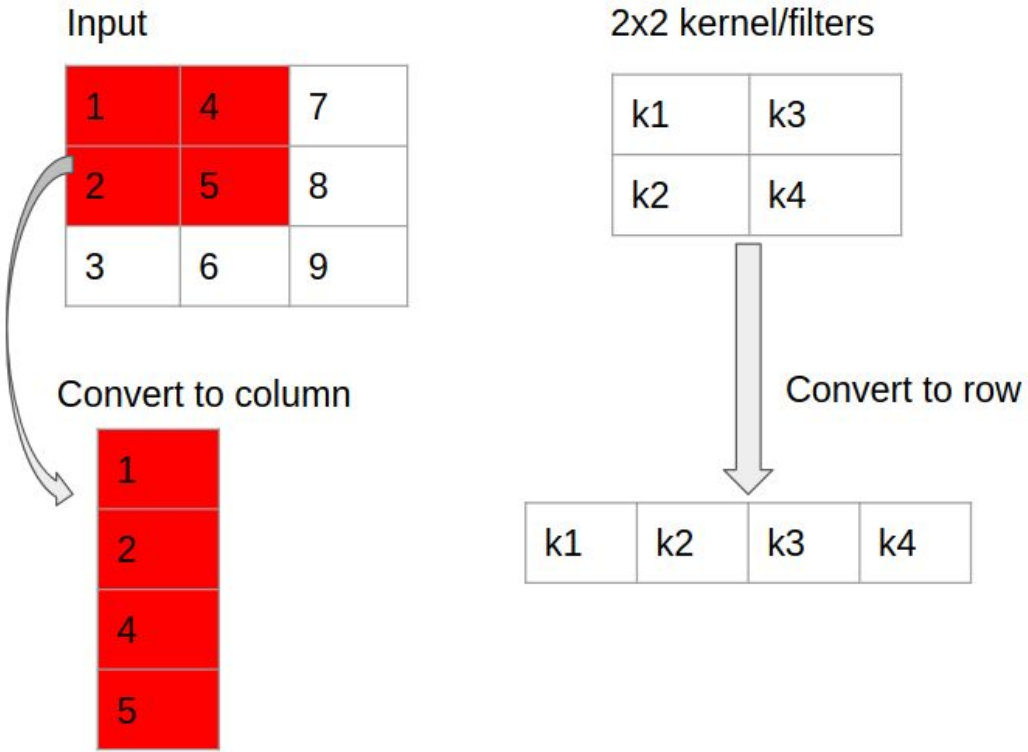
Convolution



Pooling



Im2Col Technique



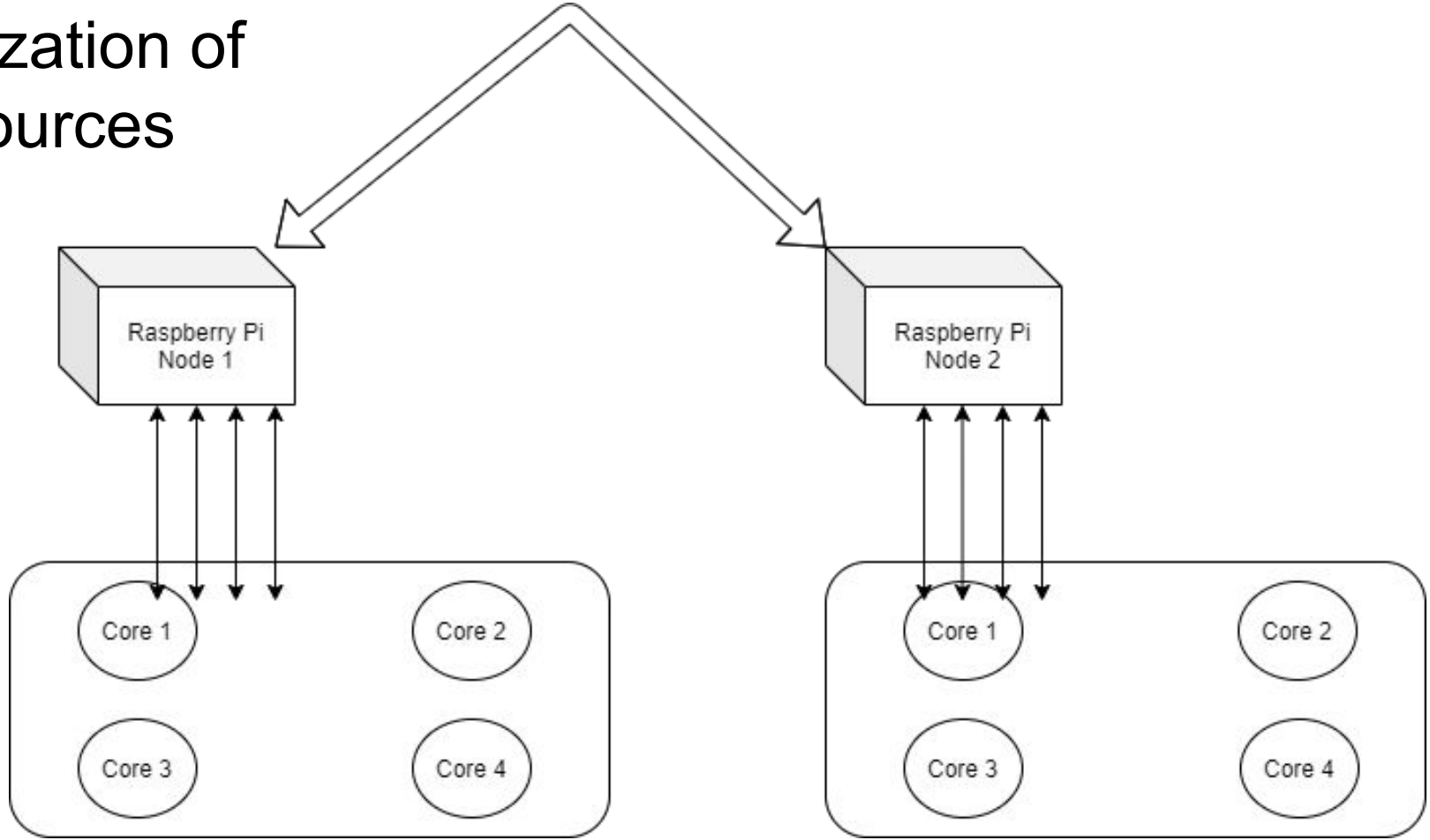
Computation offloading

ω-amount of computation
S_m-CPU speed raspberry pi
S_s-CPU speed of the server
d_s- sending data
d_r- receiving data
B-bandwidth upload

$$d_s/B + d_r/B + \omega/S_s < \omega/S_m$$

Memory usage>Total memory x threshold

Utilization of resources



Logical View of 4 Cores per a node

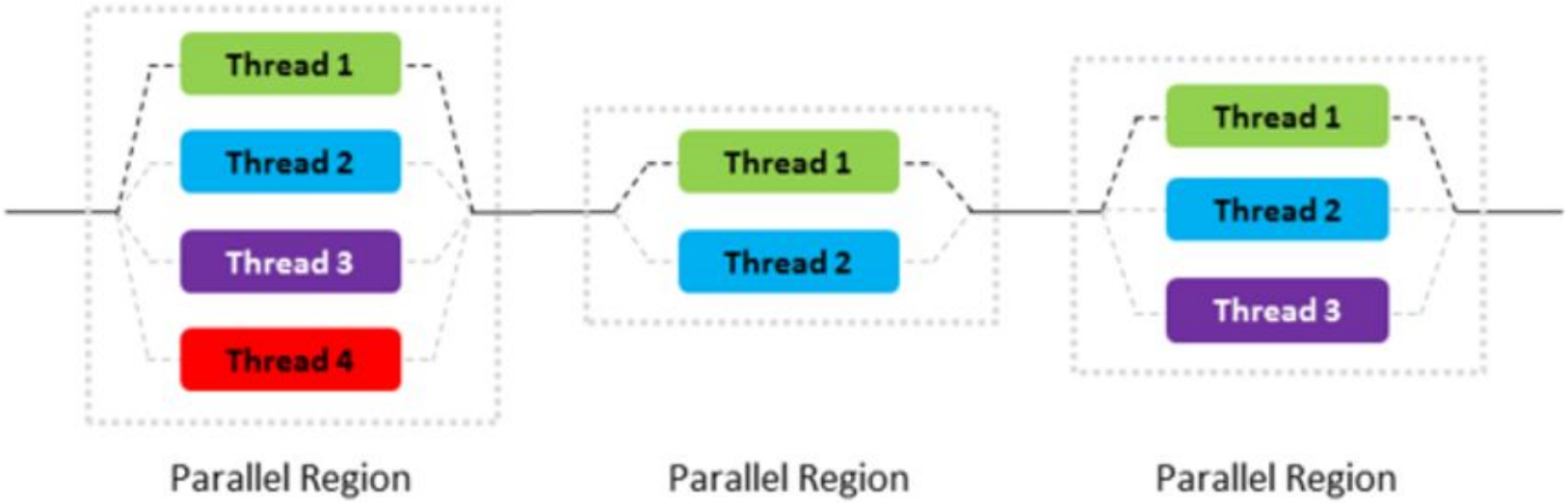
How to apply threads in Python?

- C language supports OpenMP
- Python GIL lets only on thread at a time

Options

- Cython
- PyMP

Application process

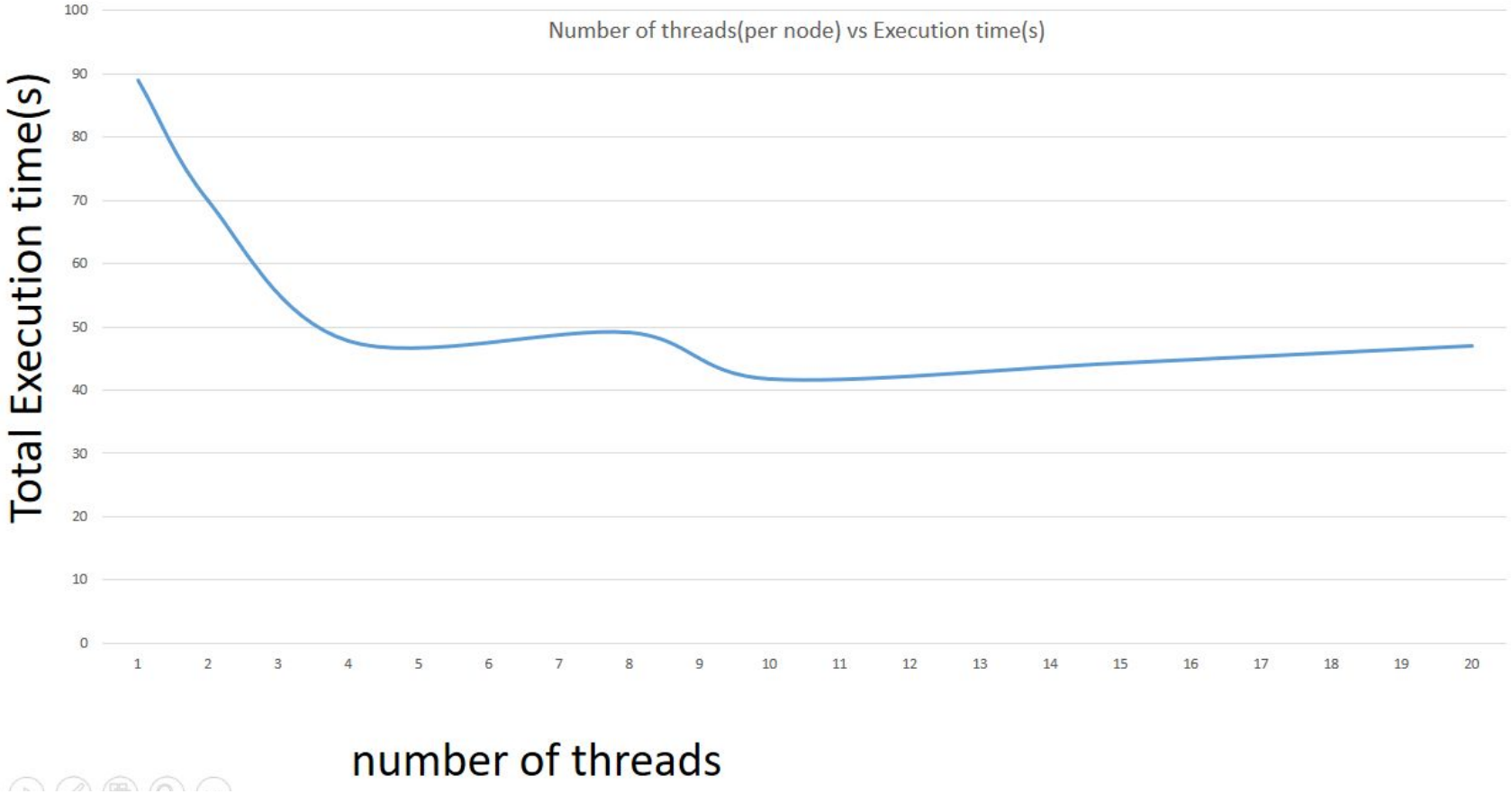


Number of threads vs Total execution time

Number of threads	Node 1 time(s)	Node 2 time(s)	Join(s)	Total(s)
2	38.012	8.4214	31.9943	70.0063
4	24.21	1.9367	23.57	47.78
8	24.12	1.034	25	49.12
10	20.74	1.0015	21.0055	41.74
15	21.9844	0.973084	22.3142	44.29
20	25.9405	1.0015	21.0554	46.9959
50	23.20	1.105	23.2363	46.43

- Single node (without parallelism threads):270+ (s)
- Distributed yolo(without threads) : 73+16=89(s)

Number of threads Vs Total execution time



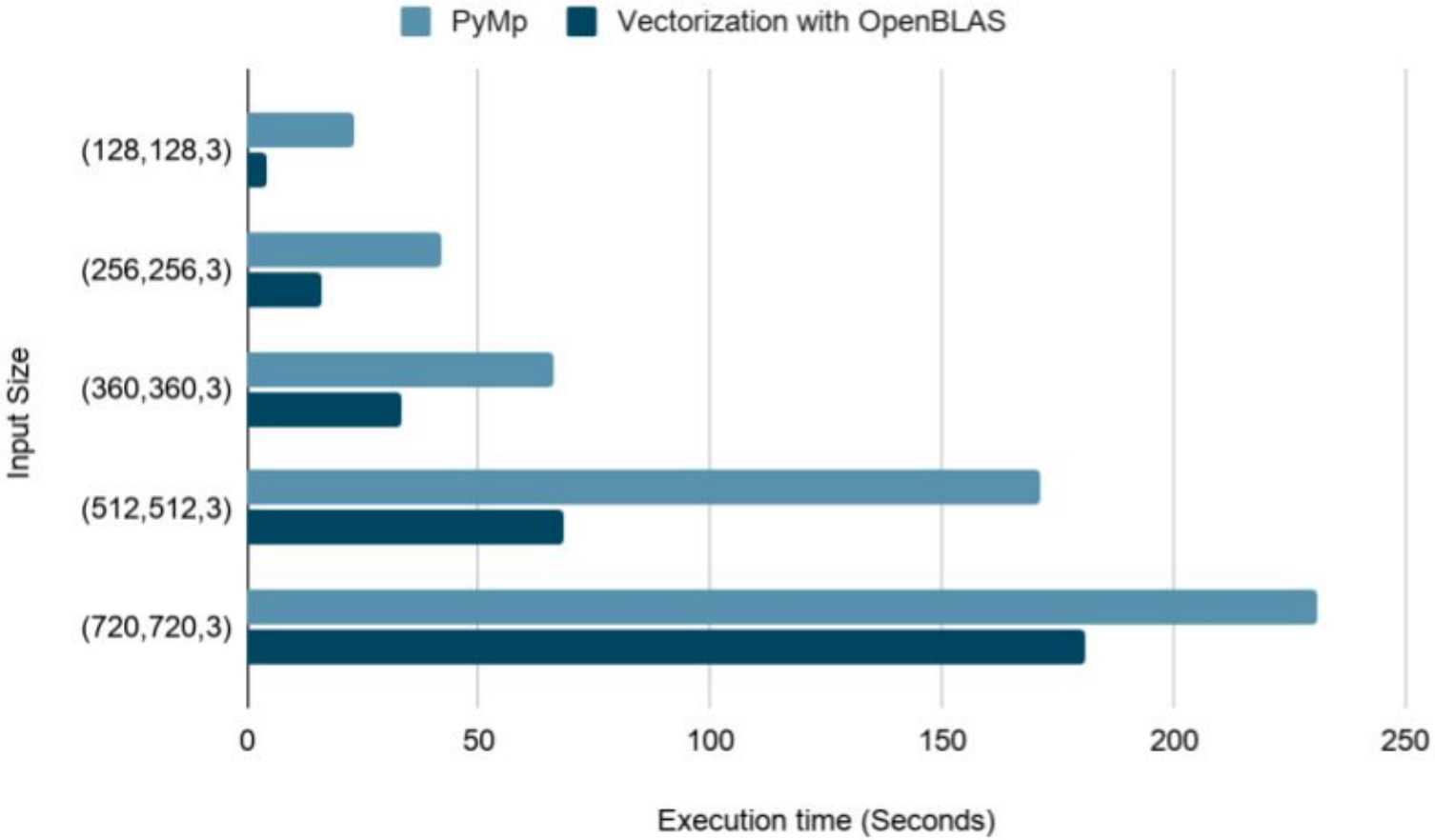
YOLO on Raspberry Pi (Results)

- Darknet yolo: Segmentation fault on calculating weights
- Darknet Tiny Yolo: Segmentation fault on cnn
- YOLO on single node 1 core: 270+ (s)
- Distributed yolo : 73+16 = 89(s)
- Distributed with (10 threads): 41.74(s)

- YOLO on Cloud (with GPU enabled): 29.430(s)

Combine vectorization with multi threads optimization

Here we used “OpenBLAS” as an optimized Basic Linear Algebra Subprograms library.

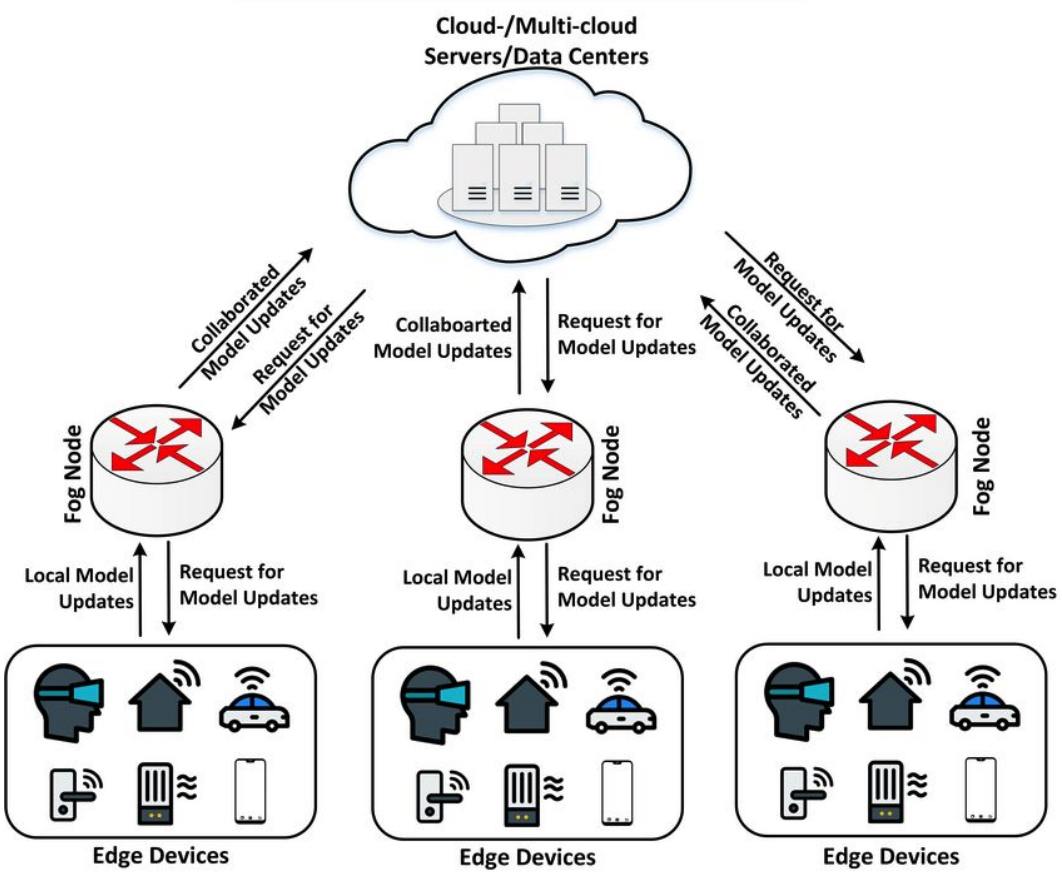

PyMp
Vectorization with OpenBLAS
Input Size
(128,128,3)
(256,256,3)
(360,360,3)
(512,512,3)
(720,720,3)
0
50
100
150
200
250
Execution time (Seconds)

Federated Learning

Importance of Federated Learning ?

Privacy

Quickly development in new device



Problems in FL model,

Our global model was less accurate than edge model

Number of participant for global model will directly impact aggregate model.

Training can be only done when edge devices are idle stage

Demo

THANK YOU


Q&A