Embedded Machine Learning for Edge Computing

Model Optimization

Damith Kawshan

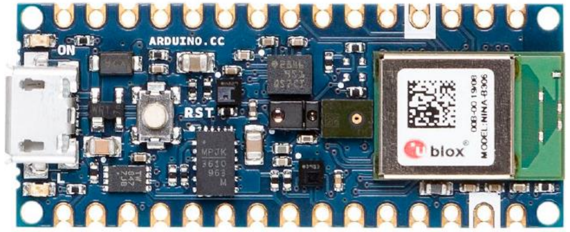
Aug 2024

About Myself

- Researcher at Nanyang Technological University, Singapore
- Specialized in FPGA based embedded system design.
- B.Sc. in Electronic & Telecom. Eng. from the University of Moratuwa, Sri Lanka (2021)
- Former Engineer – Accelerated Systems at London Stock Exchange Group
- Former Electronic Engineer at Sifive Inc.

Optimization

Year	Model name	Accuracy (%)	Size (MB)	#Params (M)	Time (ms)
2012	AlexNet	56.48	237.9	61.1	0.46
2014	VGG19	72.38	461.1	20.08	1.12
2014	Inception_v3	79.35	5212.6	6.25	5.8
2016	ResNet18	76.82	721.6	11.22	1.28
2016	ResNet50	78.07	4233.6	23.7	4.56
2016	ResNet101	77.35	6409.6	42.7	7.38
2016	ResNet152	78.28	9097.6	58.34	10.54
2018	DenseNet121	78.46	5025.6	7.05	5.23
2018	DenseNet169	75.56	6111.6	12.64	6.75
2018	DenseNet201	76.87	7947.6	18.28	9.24
2020	COVID-Net-Large*	-na-	270.88	33.85	-na-

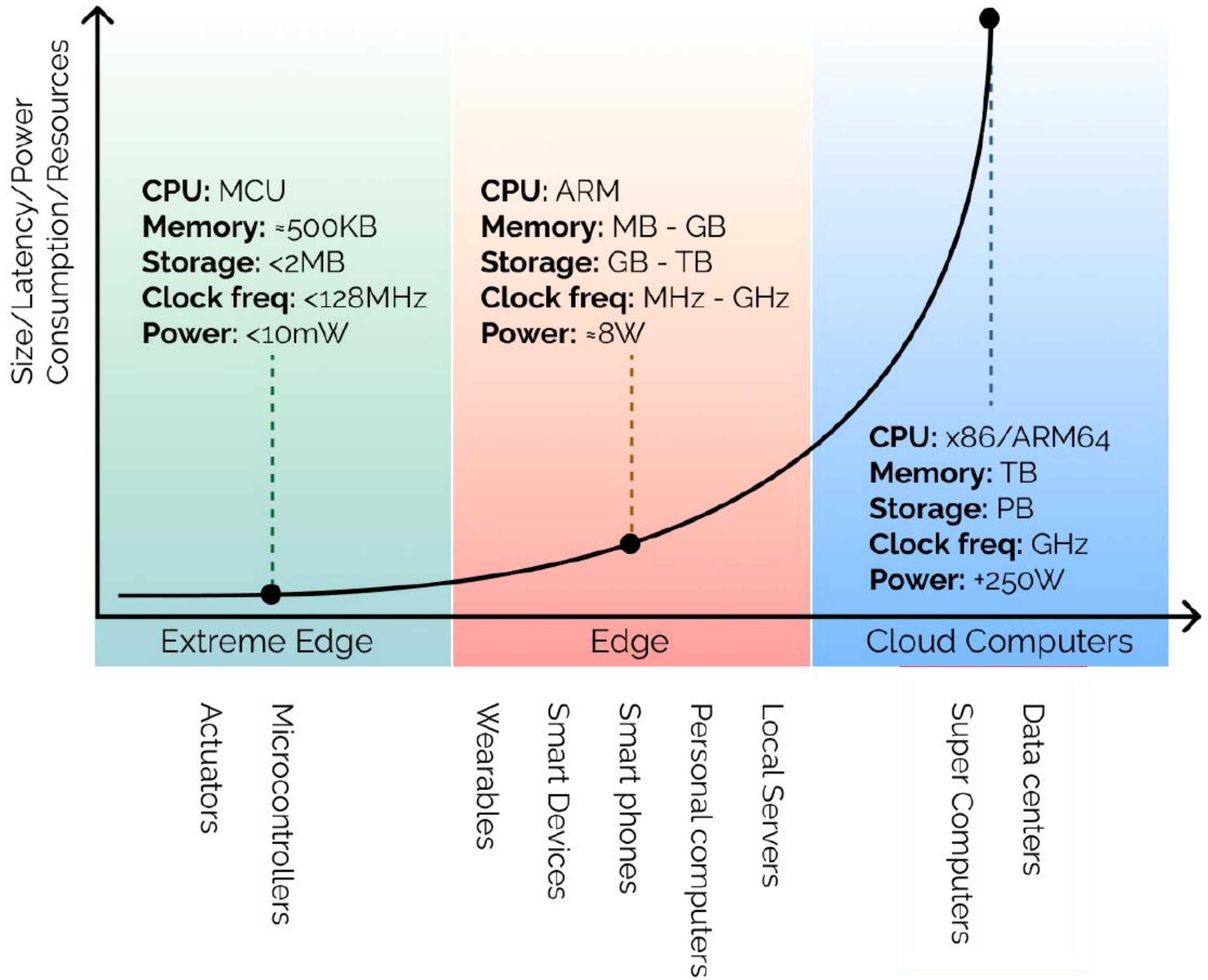


FLASH Memory size : 1MB

Smallest Model x237.9 larger !

Optimization

- **State of the Art : YOLO, ResNet, RCNN**
  - Higher Accuracy
  - But they are Massive!
- You need powerful hardware to run the inference.
- Cloud based services,
  - Amazon Web Services (AWS), Azure ..
  - Communication bandwidth can affect performance.
- Edge / Extreme Edge,
  - Limited computational resources
  - Low power requirement
  - Real time constraints



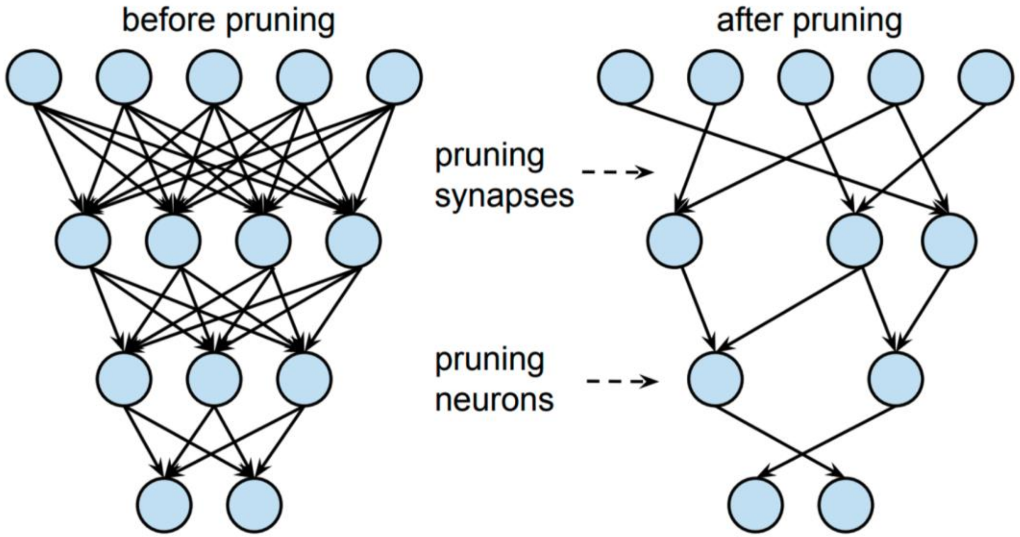
Optimization

- **Optimization :** The process of adapting complex ML models to run efficiently on resource-constrained hardware platforms.
- Trade off between Accuracy vs. Size / Performance / Speed / Power

- Today we will discuss about :
  - Pruning
  - Knowledge Distillation
  - Quantization

Optimization - I

- **Pruning :** Iterative process of removing parameters (Turning them into 0)



Optimization - I

Why use pruning?

To increase speed and reduce model size.

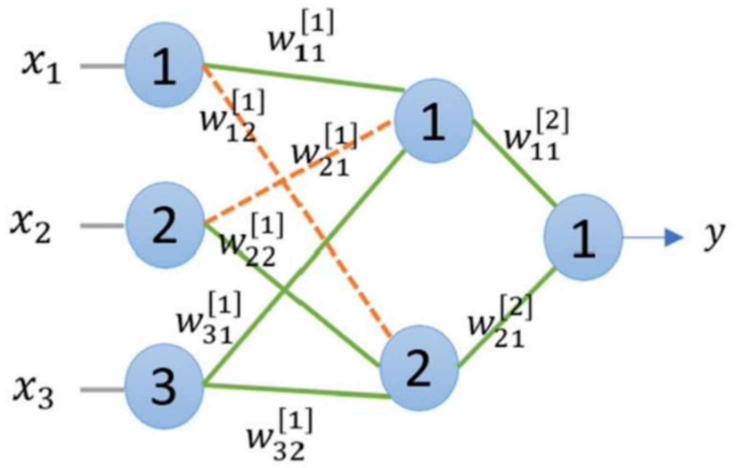
BUT,

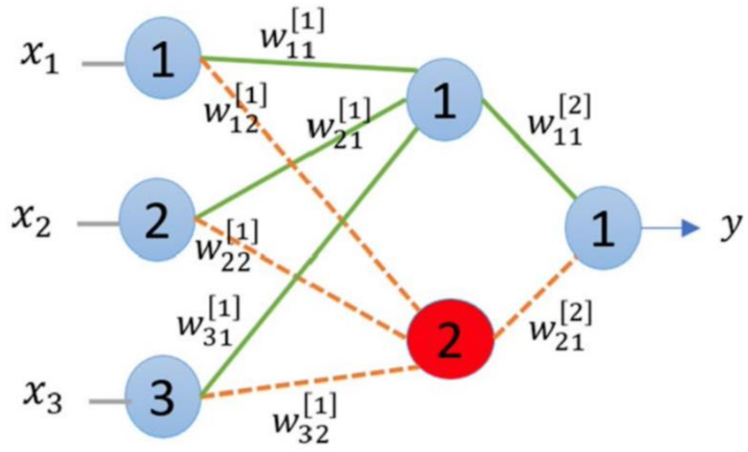
Pruning can cause loss of Accuracy

$$n_1 = w^{[1]}_{11} x_1 + \cancelto{0}{w^{[1]}_{21}} x_2 + w^{[1]}_{31} x_3$$

$$n_2 = \cancelto{0}{w^{[1]}_{12}} x_1 + w^{[1]}_{22} x_2 + w^{[1]}_{32} x_3$$

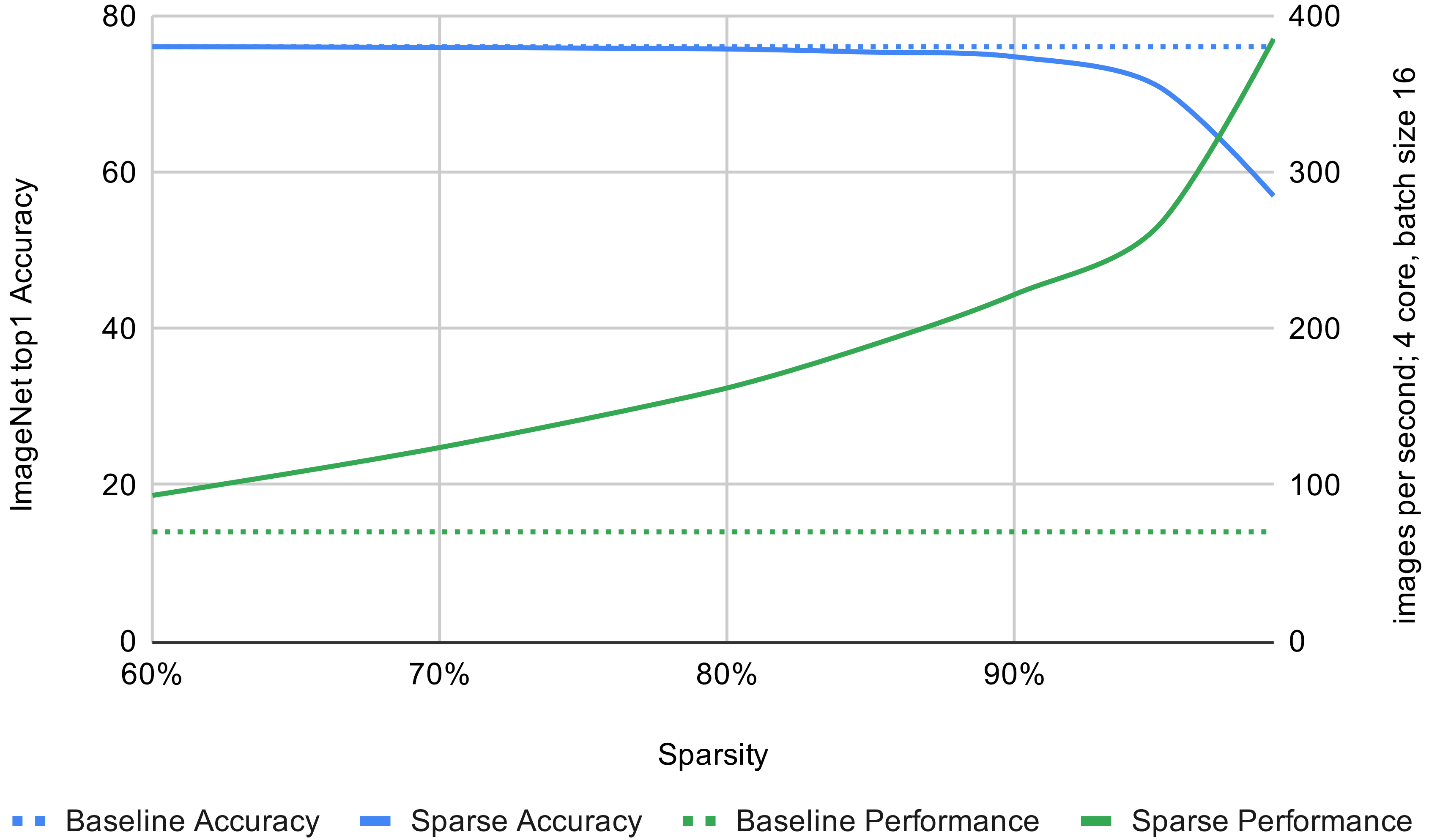
$$y = n_1 + n_2$$





Optimization - I

Performance and Accuracy numbers for a ResNet 50 model trained on ImageNet as compared to uniform sparsity levels.



Source : https://opendatascience.com/what-is-pruning-in-machine-learning/

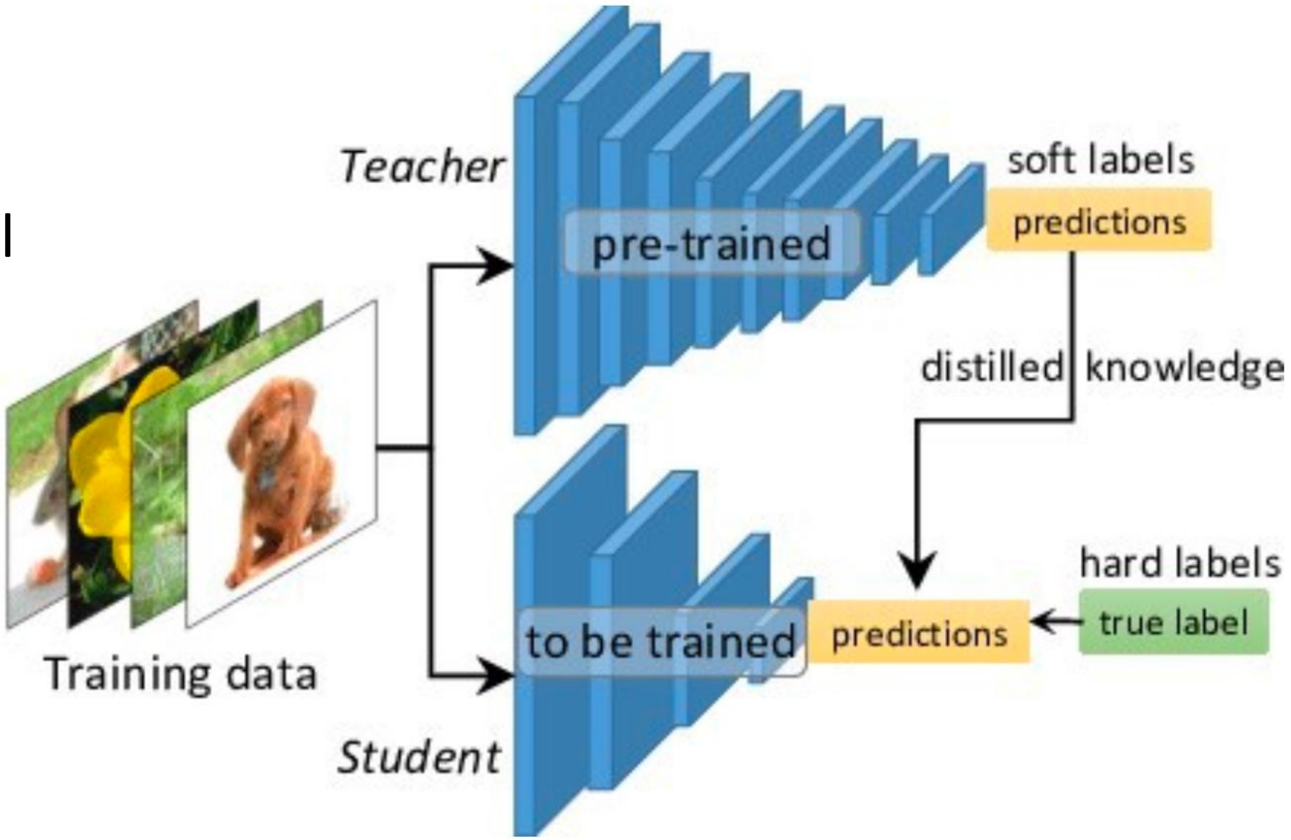
Optimization - II

- **What is Knowledge Distillation?**
  - Takes the larger network ***(N1)*** and…
  - Trains a smaller model ***(N2)*** that…
  - Attempts to mimic its ***(N1's)*** behavior

- **Relationship between a real-world student and teacher**
- **Similarly,**
  - Compressed model is the Student network
  - Pre-trained model is the Teacher network

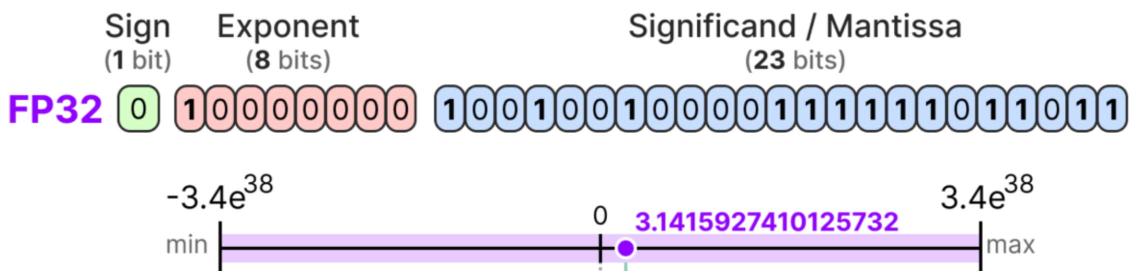
Optimization - II

- **Hard Labels**
  - True labels
- **Soft Labels**
  - Labels generated by teacher model



Optimization - III

- **Quantization :** Technique to reduce the precision of the numbers used to represent a model's parameters while keeping desired accuracy.
- Usually, 32-bit floating point is used to store model parameters.
  - Float32 mapped into INT8 (8bit integer)
  - 4 billion numbers in the interval [$-3.4e^{38}$, $3.4e^{38}$] → mapped into 256 values (2^8)
  - Reducing the precision of activation and parameter data from 32-bit floats to 8-bit integers results in 4x data reduction.
- This results in a smaller model size and faster computation.



Optimization - III

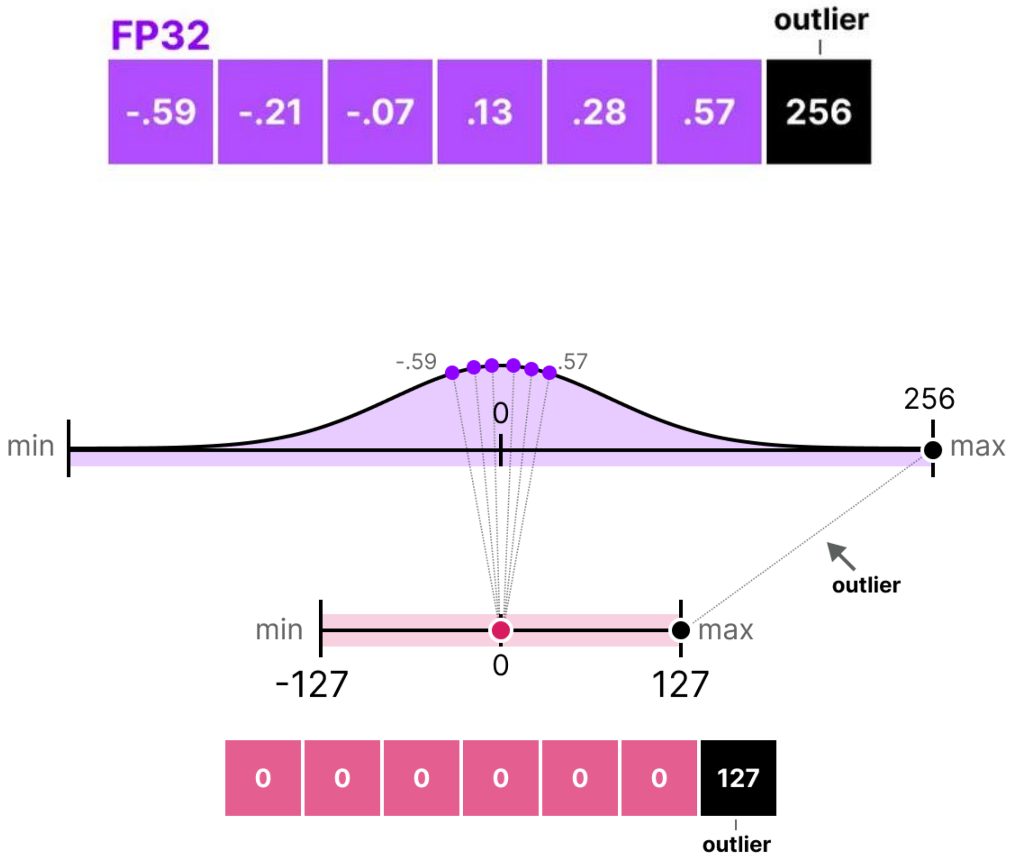
$$W = \begin{bmatrix} 0.97 & 0.64 & 0.74 & 1.00 \\ 0.58 & 0.84 & 0.84 & 0.81 \\ 0.00 & 0.18 & 0.90 & 0.28 \\ 0.57 & 0.96 & 0.80 & 0.81 \end{bmatrix} \approx \frac{1}{255} \begin{bmatrix} 247 & 163 & 109 & 255 \\ 148 & 214 & 214 & 207 \\ 0 & 46 & 229 & 71 \\ 145 & 245 & 204 & 207 \end{bmatrix} = w_t W_{int8}$$

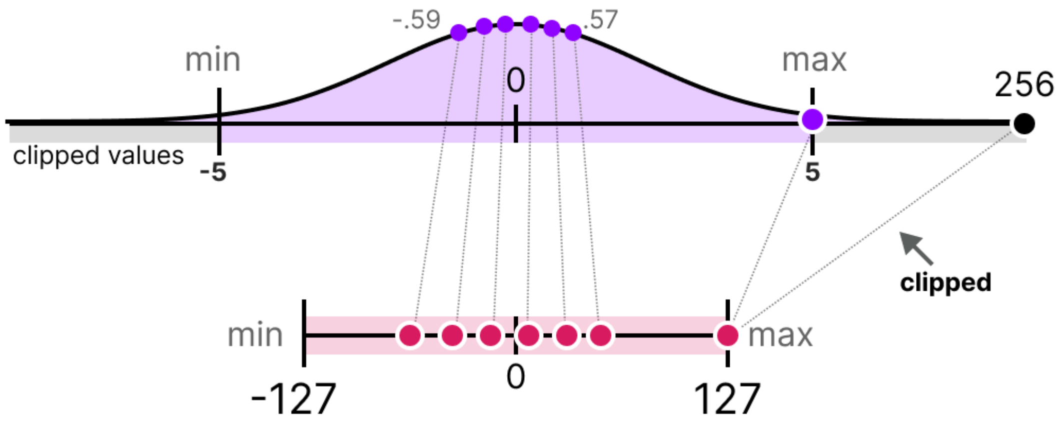
$$\text{Error} = W - w_t W_{int8}$$

$$\text{Error} = \begin{bmatrix} 0.97 & 0.64 & 0.74 & 1.00 \\ 0.58 & 0.84 & 0.84 & 0.81 \\ 0.00 & 0.18 & 0.90 & 0.28 \\ 0.57 & 0.96 & 0.80 & 0.81 \end{bmatrix} - \frac{1}{255} \begin{bmatrix} 247 & 163 & 109 & 255 \\ 148 & 214 & 214 & 207 \\ 0 & 46 & 229 & 71 \\ 145 & 245 & 204 & 207 \end{bmatrix}$$

$$\text{Error} = \begin{bmatrix} 0.0014 & 0.0008 & 0.3125 & 0.00 \\ -0.0004 & 0.0008 & 0.0008 & -0.0018 \\ 0.00 & -0.0004 & 0.0020 & 0.0016 \\ 0.0014 & -0.0008 & 0.00 & -0.0018 \end{bmatrix}$$

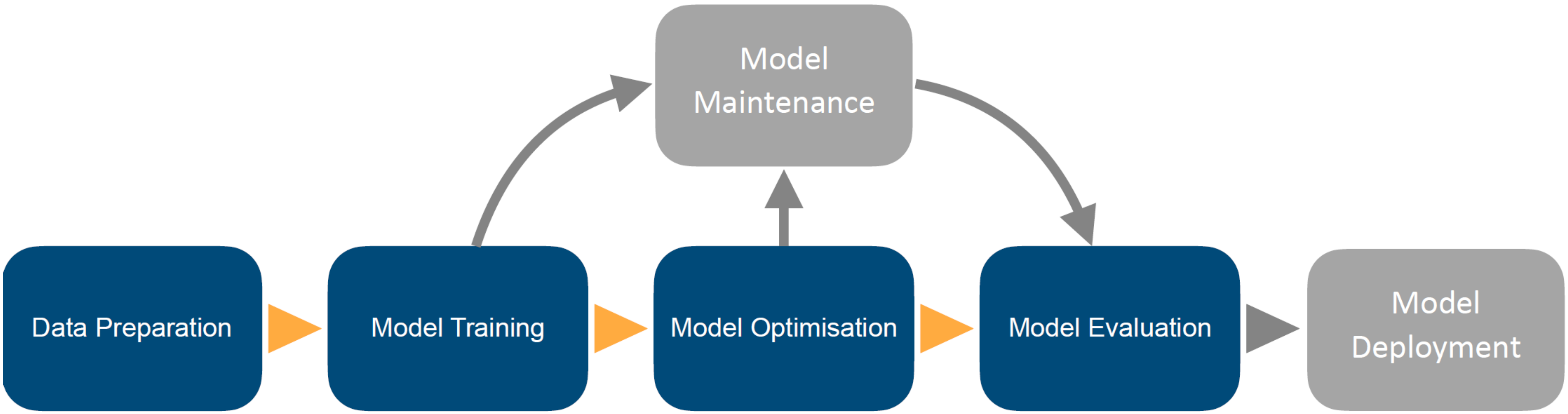
Optimization - III

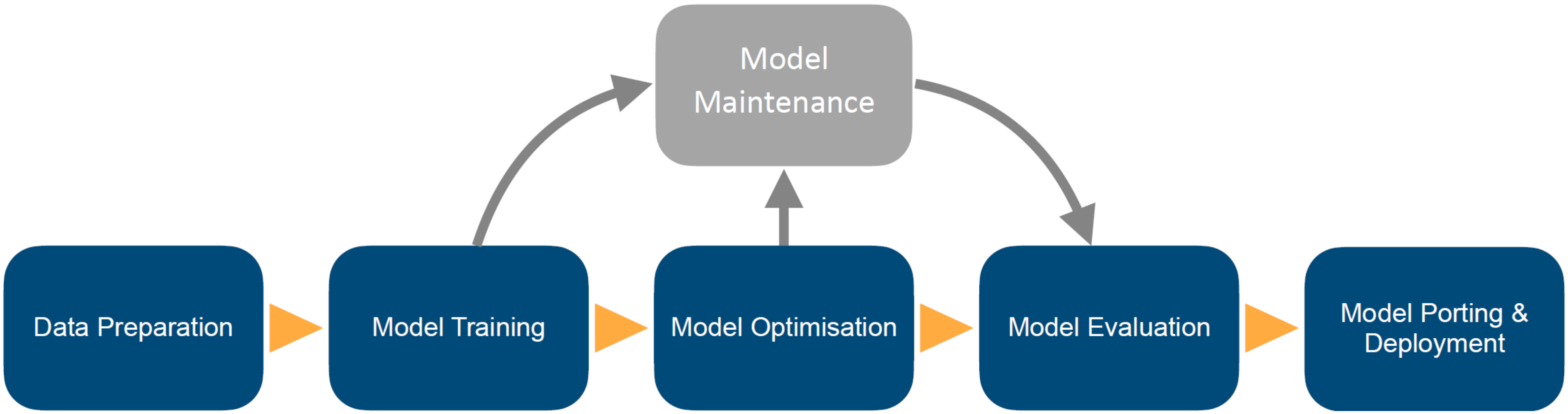




Optimization - III

- Types of Quantization Techniques :
  - Post-training Quantization
    - quantizing a model’s parameters **after** training the model.
  - Quantization-aware training
    - Simulates low precision behavior in the forward pass
  - Float16 quantization


Model
Maintenance
Data Preparation
Model Training
Model Optimisation
Model Evaluation
Model
Deployment


Model
Maintenance
Data Preparation
Model Training
Model Optimisation
Model Evaluation
Model Porting &
Deployment

Porting the Model

- Models cannot be embedded in their raw form
  - Needs to be converted to an MCU understandable format

- 3 identified ways of porting a model to an MCU
  - Manual Programming
  - Code generation tools
  - TinyML Interpreters

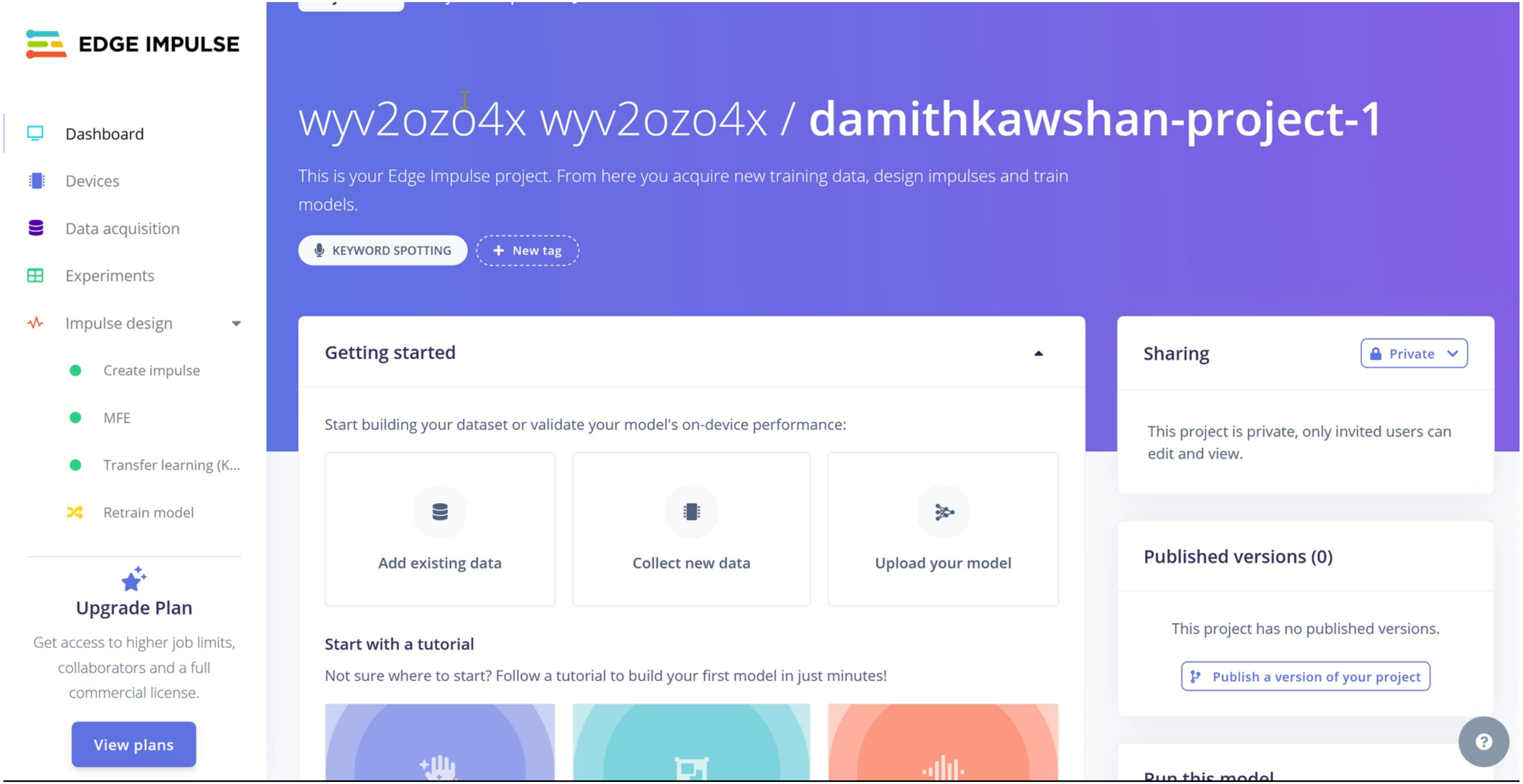
Porting the Model

	Manual Programming	Code generation	TinyML Interpreters
Characteristics	Can tinker all aspects of the model	Convenient to get a solution up and running fast	Superior portability
	Can yield better results	Competing vendors	Model architecture is not coupled to any framework or functionality
	Hardly generalisable	Toolsets/Software are proprietary	Slight overhead in resource requirements
Examples	Primarily research related implementations	Imagimob studio, EdgeImpulse	Tensorflow Lite Micro

Porting the Model

	Manual Programming	**Code generation**	**TinyML Interpreters**
Characteristics	Can tinker all aspects of the model	Convenient to get a solution up and running fast	Superior portability
	Can yield better results	Competing vendors	Model architecture is not coupled to any framework or functionality
	Hardly generalisable	Toolsets/Software are proprietary	Slight overhead in resource requirements
Examples	Primarily research related implementations	Imagimob studio, EdgeImpulse	Tensorflow Lite Micro

Porting the Model



Porting the Model

	Manual Programming	Code generation	TinyML Interpreters
Characteristics	Can tinker all aspects of the model	Convenient to get a solution up and running fast	Superior portability
	Can yield better results	Competing vendors	Model architecture is not coupled to any framework or functionality
	Hardly generalisable	Toolsets/Software are proprietary	Slight overhead in resource requirements
Examples	Primarily research related implementations	Imagimob studio, EdgeImpulse	Tensorflow Lite Micro

Thank you !