All Roads Lead to TinyML: The Rome of Efficient Machine Learning in Engineering

Session 2: Model Compression Techniques

Dr. Dinuka Sahabandu

Assistant Teaching Professor

Department of Electrical and Computer Engineering

University of Washington

ELECTRICAL & COMPUTER ENGINEERING

UNIVERSITY of WASHINGTON

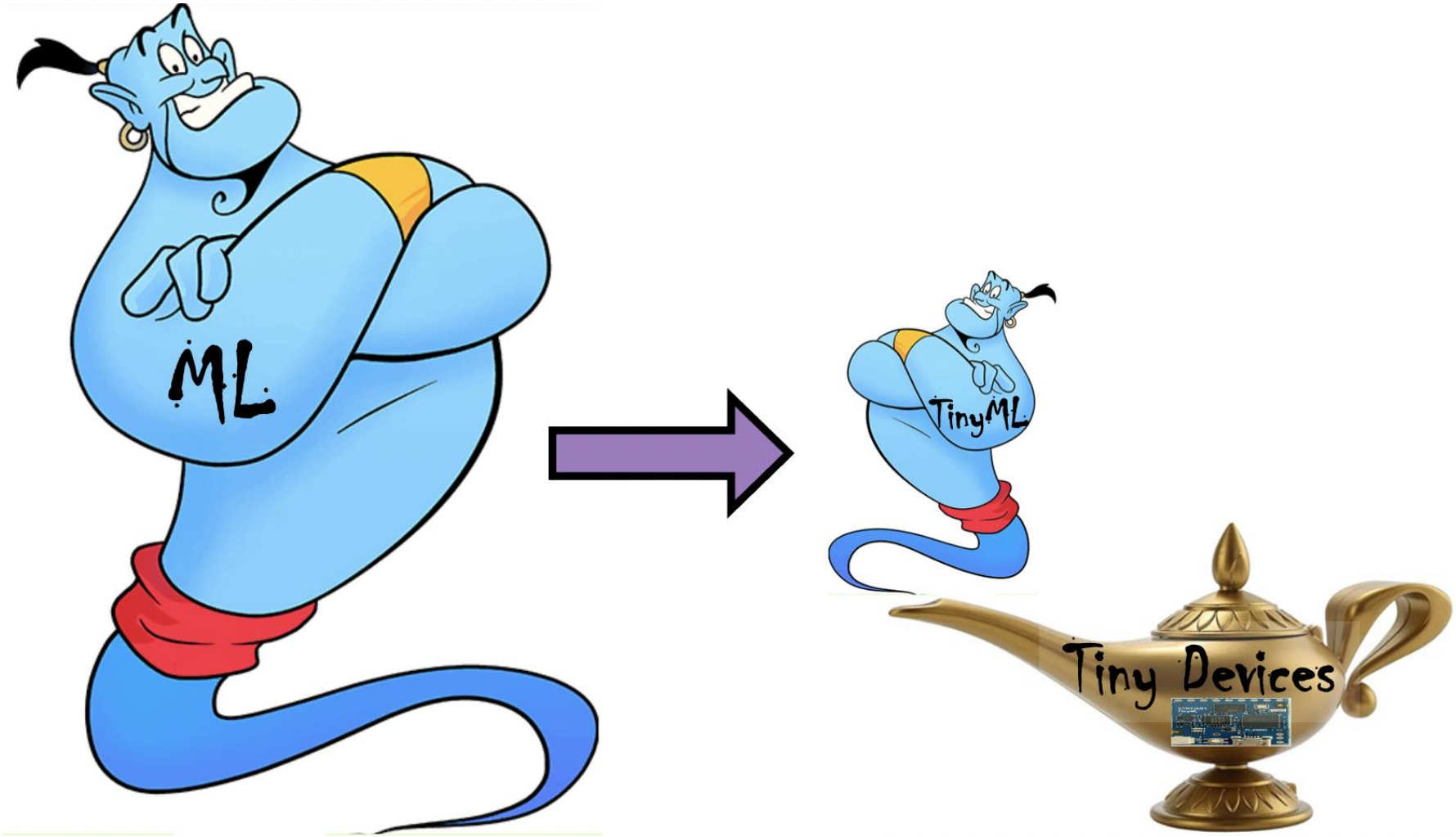
BRAIN LABS

Brain-Inspired AI & Neuroinformatics Research Group

Topics Covered

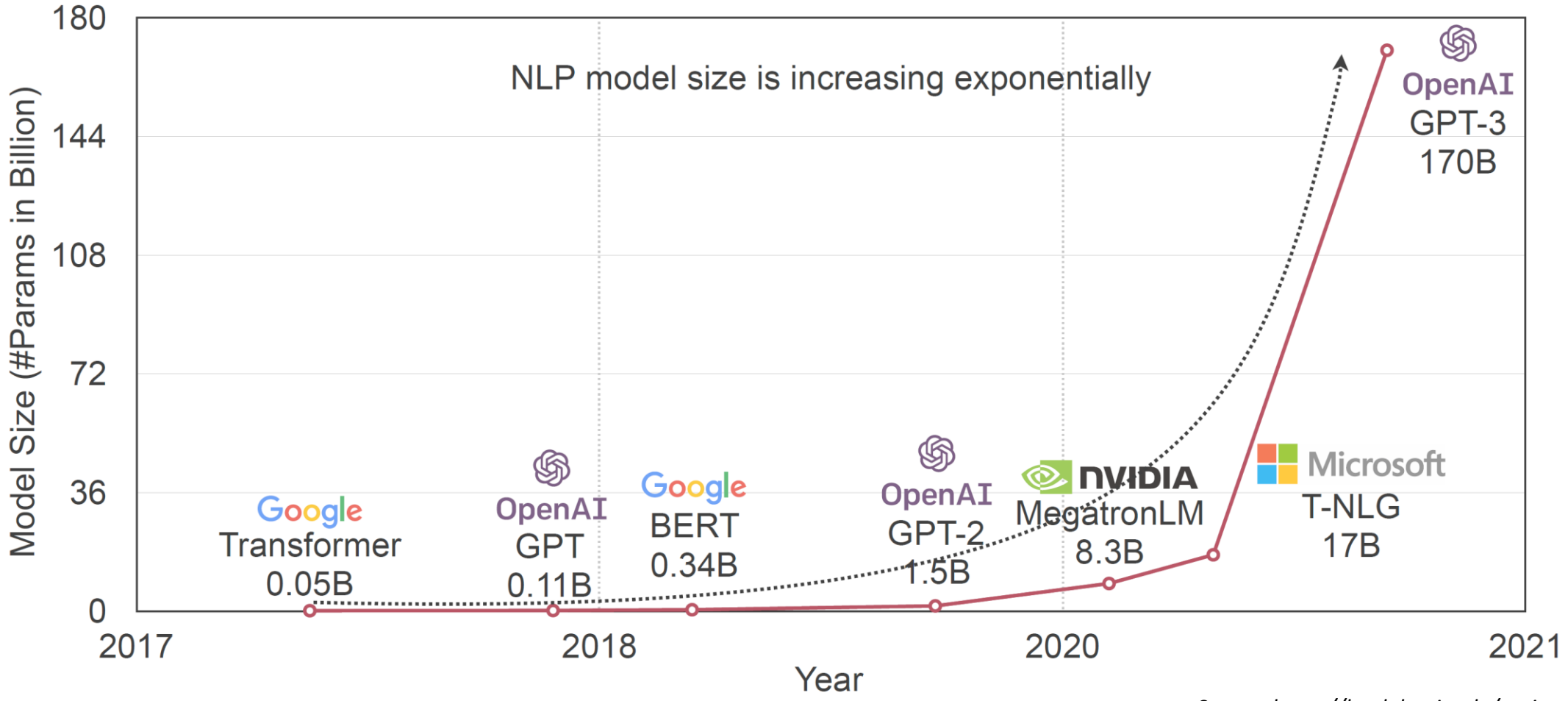
- Introduction of Different Model Compression Methods
- Model Quantization Techniques
- Introduction to TensorFlow Lite
- Model Pruning Techniques
- Knowledge Distillation (KD) for Model Compression
- **Coding session**: Quantization, Pruning, and KD using TensorFlow / TensorFlow Lite
- Emerging Applications and Future of TinyML

Overview of TinyML



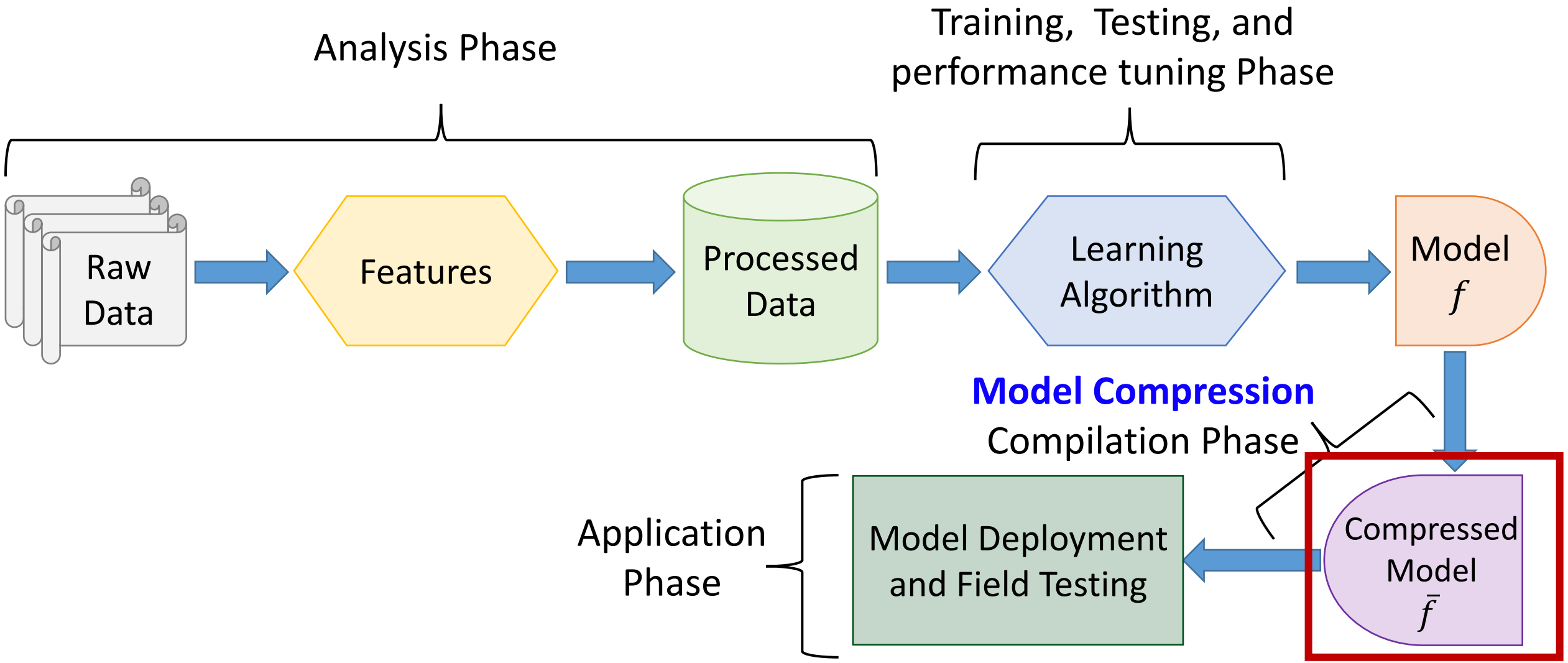
Trend of AI Model Size

- **Model size and parameters of AI continue to growing fast!**

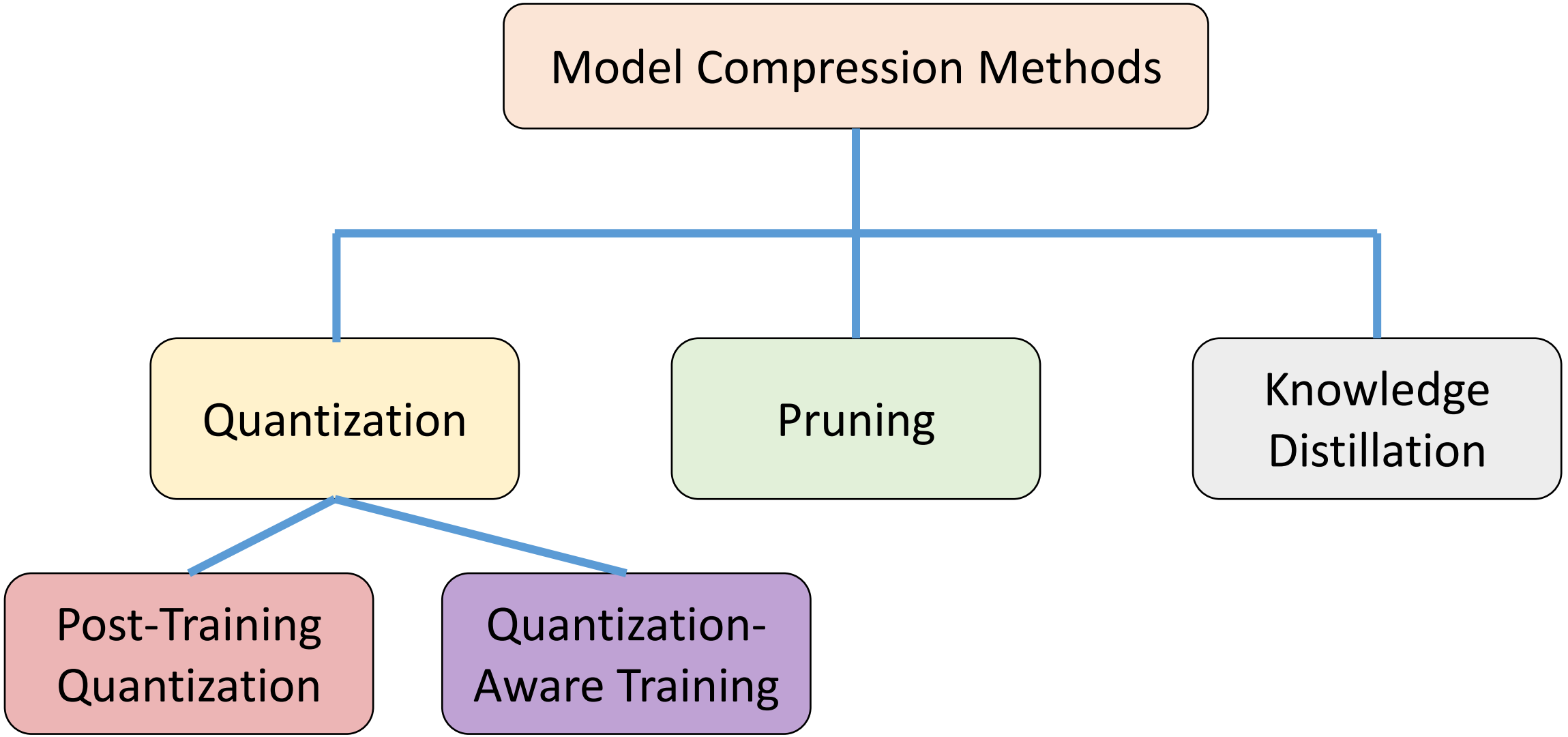


Source: https://hanlab.mit.edu/projects/efficientnlp_old/

TinyML Model Compression Phase

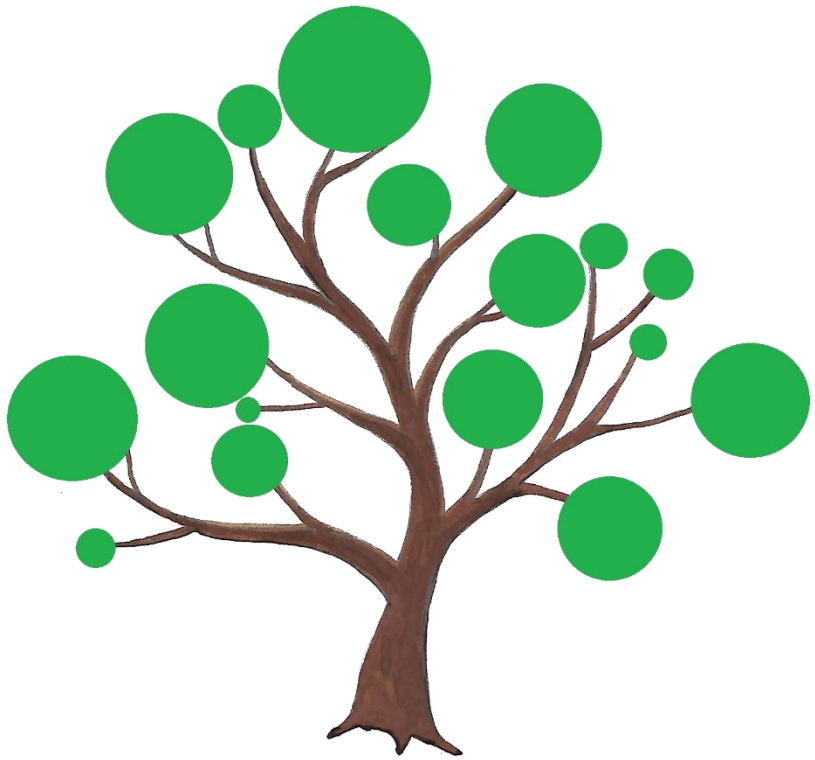


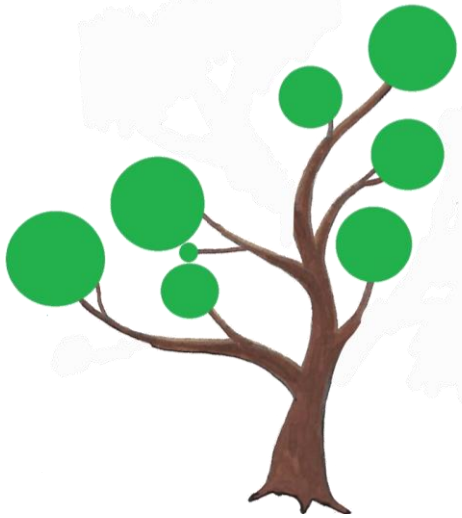
Key Model Compression Techniques



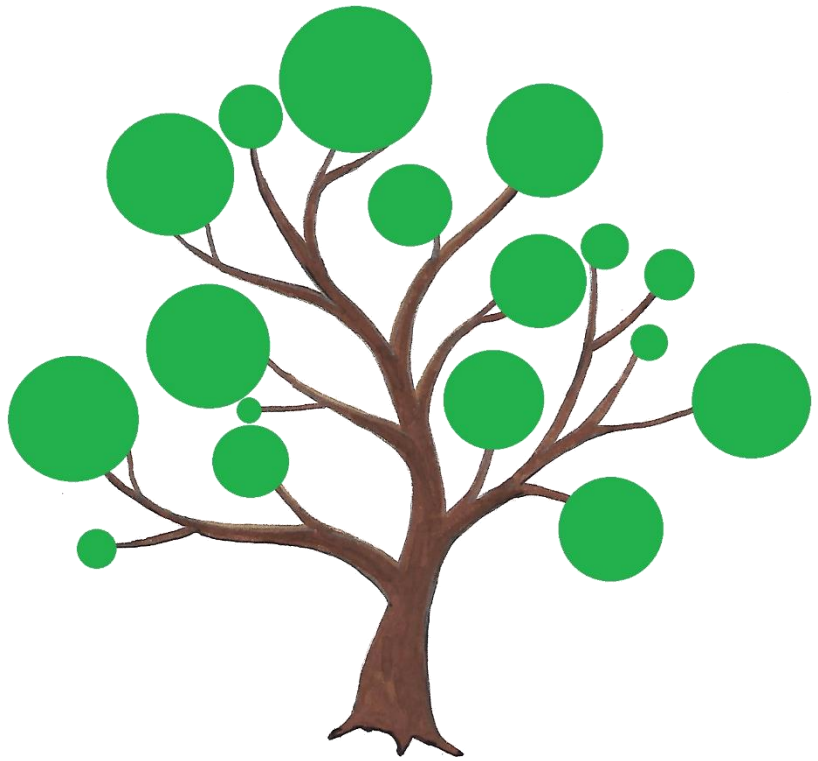
Quantization Vs. Pruning Discussion

1. What is quantization?
2. What is pruning?

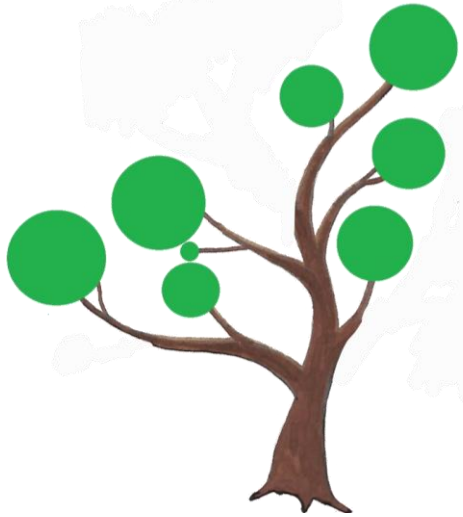




Quantization and Pruning



A Tree



A Pruned Tree

A Quantized Tree

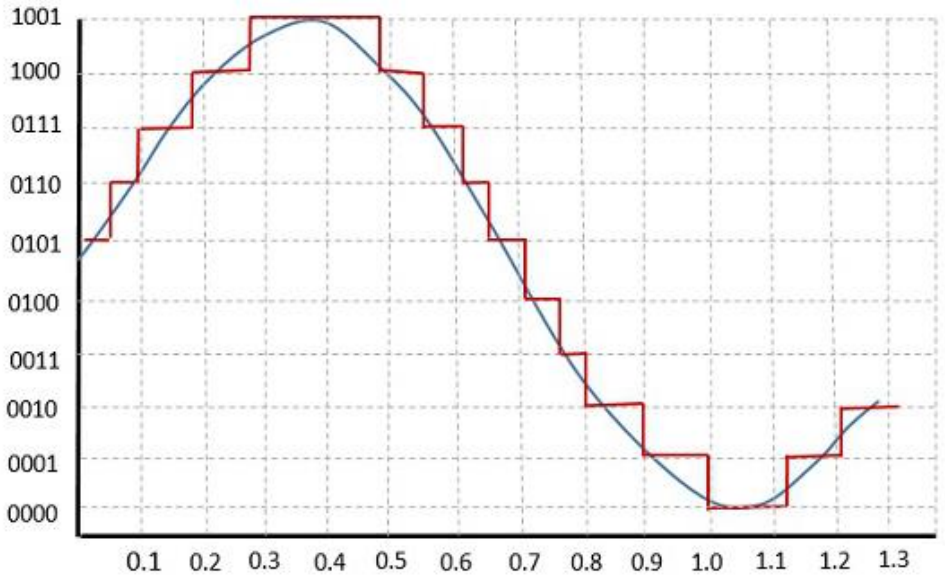
A Pruned and Quantized Tree

Quantizing Image and Signal Inputs

Original Image Of Billion Colors

Quantized image with four color Image

RGB (24 bits, 2^{24} colors) **2 bits, $2^2 = 4$ colors**



Quantization in ML

Quantization of ML offers significant benefits to TinyML!

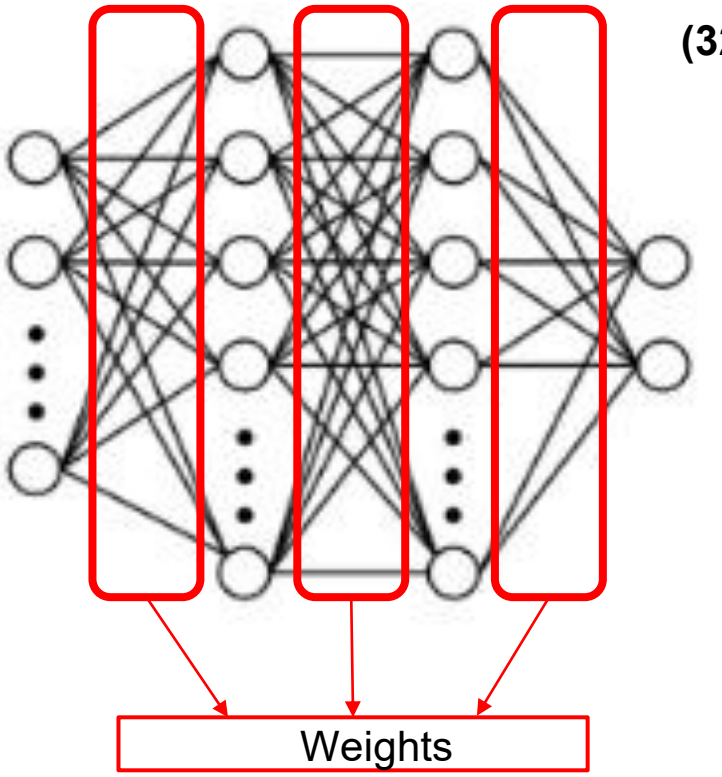
Where and how?

- Memory usage:
  - 8-bit versus 32-bit weights and activations stored in memory
- Power consumption:
  - Significant reduction in energy for both computations and memory access
- Latency:
  - With less memory access and simpler computations, latency can be reduced
- Silicon area:
  - Integer math or fewer bits requires less silicon area compared to floating point math and more bits

Quantization in ML

- **Why do we want to quantize a DNN?**
  To reduce the model size by reducing the complexity of representing model parameters

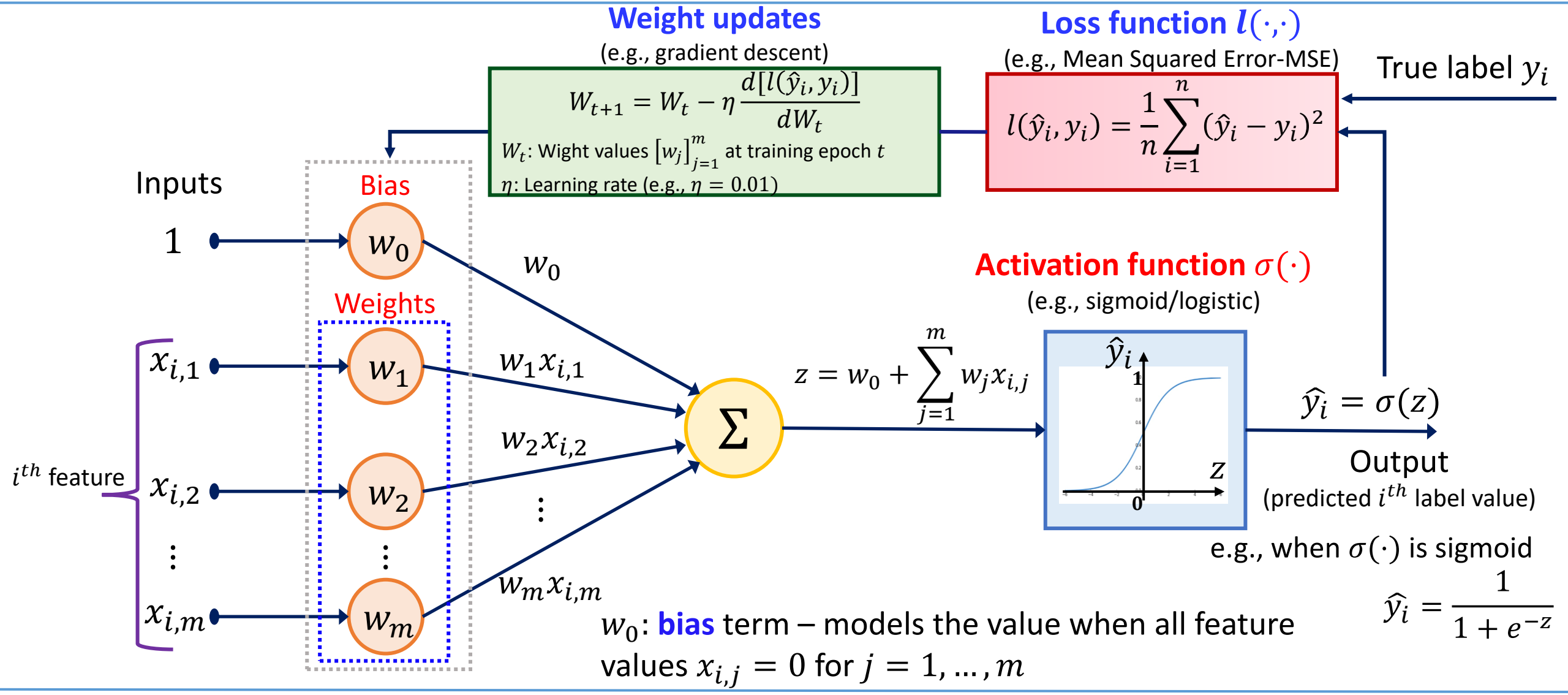
- **What parameters are we quantizing in DNN?**
  Model weights (bias , activation outputs)



Example of weights
(32-bit float)

2.09
-0.98
1.48
0.09
0.05
-0.14
-1.08
2.12
-0.91
1.92
0
-1.03
1.87
0
1.53

Structure of a Single Layer of Neural Network



Quantization

Quantization in Neural Networks

1. Theory in Quantization

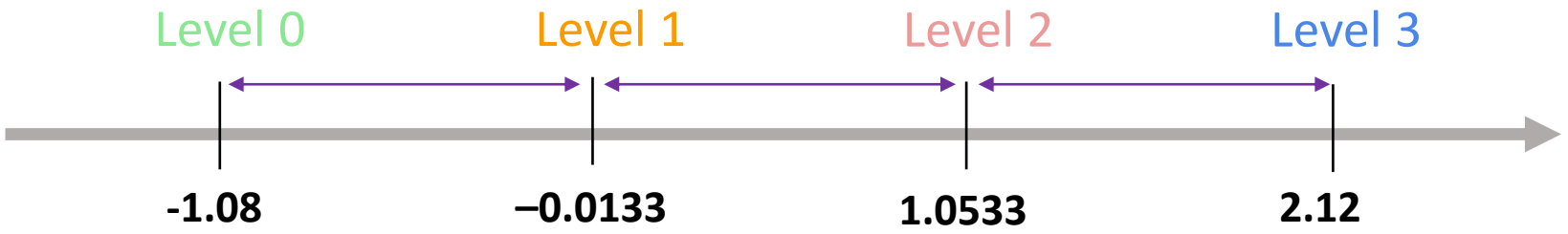
- Linear quantization

2. Post-Training Quantization (PTQ)
    - Weight quantization
    - Activation quantization
    - Bias quantization

3. Quantization-Aware Training (QAT)

Linear Quantization: Example

- **-1.08, -1.03, -0.98, -0.91, -0.41, 0, 0, 0.05, 0.09, 1.48, 1.49, 1.53, 1.87, 1.92, 2.09, 2.12**
- **Find the range:** Determine the minimum and maximum values. In this case, the minimum is -1.08 and the maximum is 2.12.
  Range = 2.12 – (-1.08) = 3.2
- **Determine the scale:** With 4 quantization levels (e.g., 2-bit), the range (-1.08 to 2.12) is divided into 3 $(2^4 - 1)$ intervals.
  The interval size would be (2.12−(−1.08))/3 = 3.2/3 = 1.0667

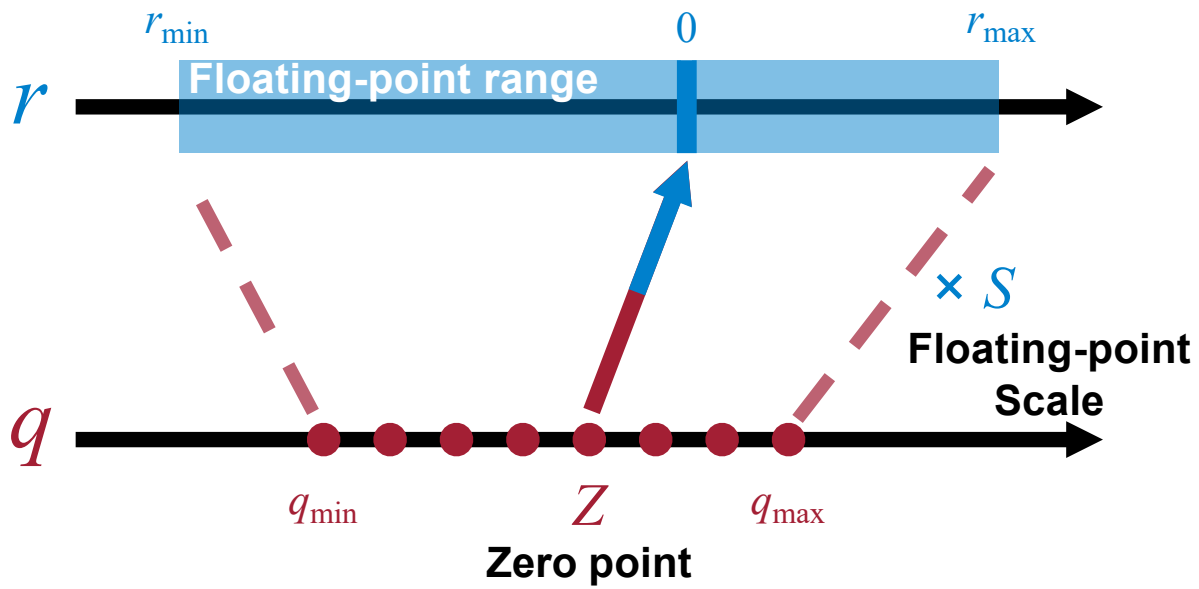


- **Quantize:** Assign each number to the nearest quantization level based on these intervals.

-1.08, -1.03, -0.98, -0.91, -0.41, 0, 0, 0.05, 0.09, 1.48, 1.49, 1.53, 1.87, 1.92, 2.09, 2.12

Linear Quantization: Formal Definition

An affine mapping of integers to real numbers $r = S(q - Z)$



$$r_{max} = S(q_{max} - Z)$$

$$r_{min} = S(q_{min} - Z)$$

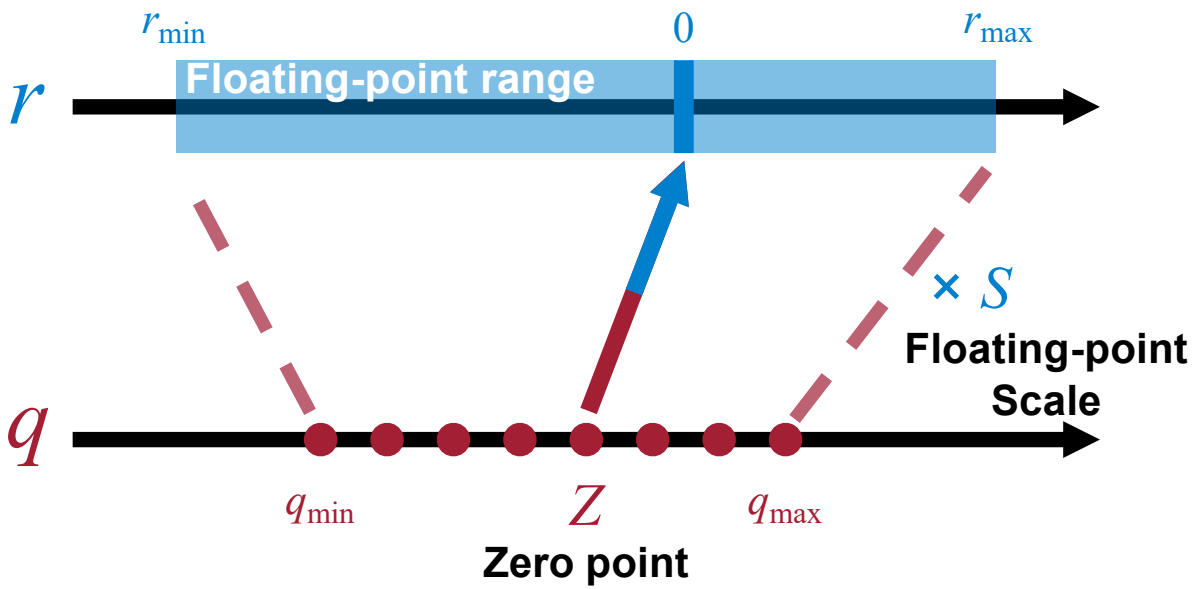
$$r_{max} - r_{min} = S(q_{max} - q_{min})$$

$$S = \frac{r_{max} - r_{min}}{q_{max} - q_{min}}$$

Source: Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference [Jacob et al., CVPR 2018]

Linear Quantization : Formal Definition

An affine mapping of integers to real numbers $r = S(q - Z)$



$$r_{\min} = S(q_{\min} - Z)$$

$$\downarrow$$

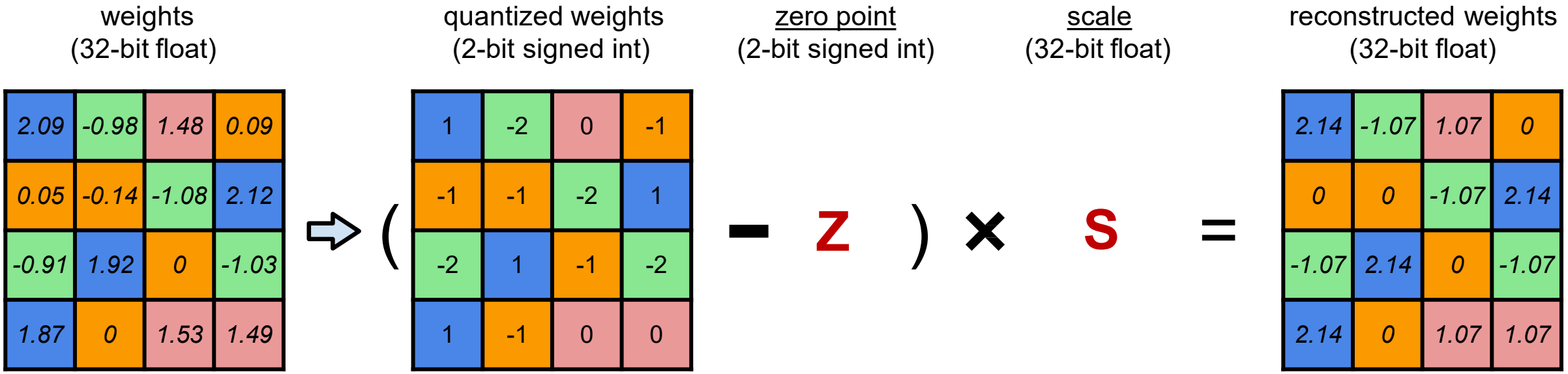
$$Z = q_{\min} - \frac{r_{\min}}{S}$$

$$\downarrow$$

$$Z = round(q_{min} - \frac{r_{min}}{S})$$

Source: Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference [Jacob et al., CVPR 2018]

Linear Quantization



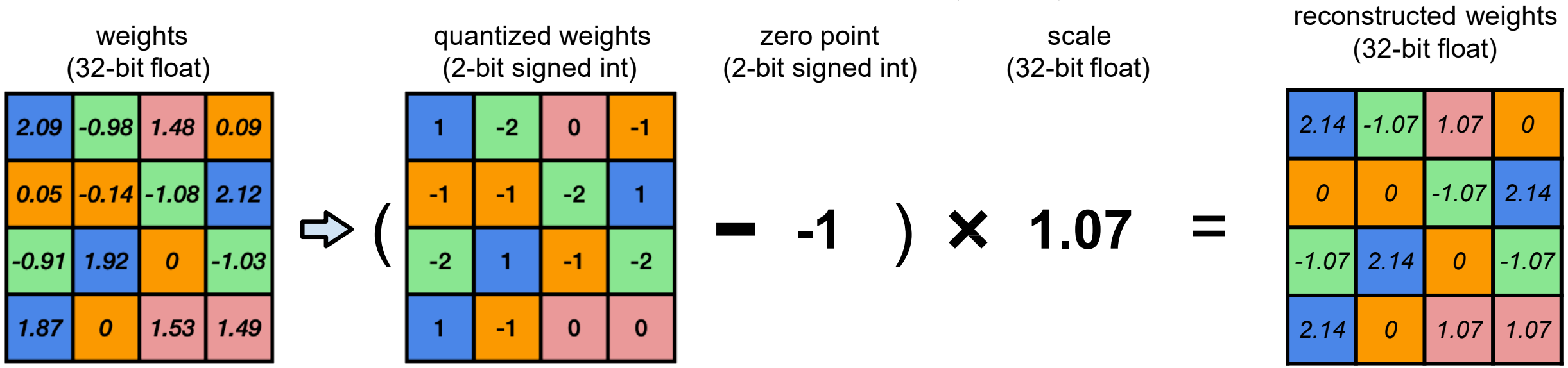
2-bit signed int to quantization weight mapping

Binary	Decimal
01	1
00	0
11	-1
10	-2

Source: Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference [Jacob et al., CVPR 2018]

Linear Quantization

An affine mapping of integers to real numbers $r = S(q - Z)$



r = (q − Z) × S

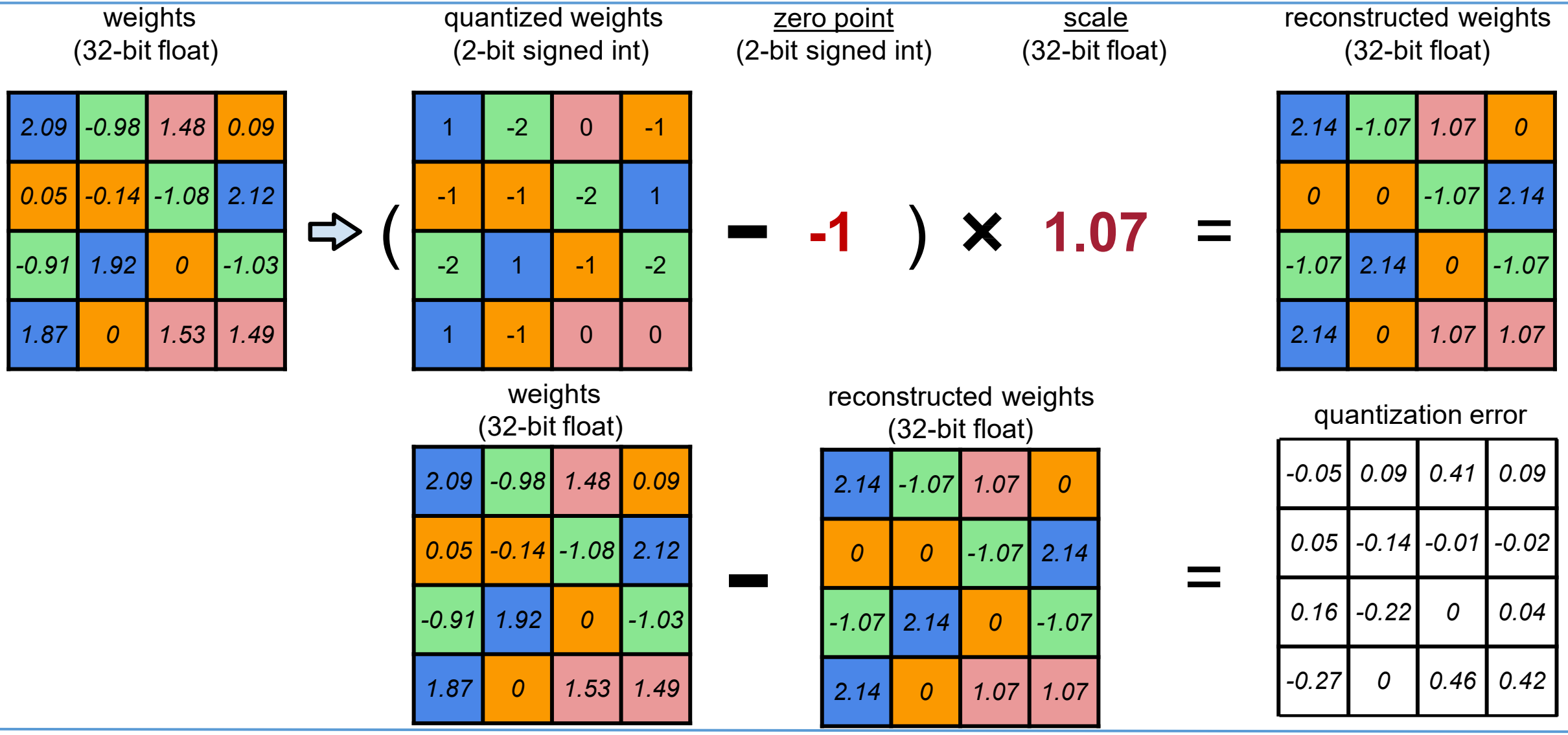
Floating-Point = (**Integer** − **Integer**) × **Floating-Point**

- quantization parameter
- allow real number *r=0* be exactly representable by a quantized integer

- quantization parameter

Source: Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference [Jacob et al., CVPR 2018]

Linear Quantization



Source: Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference [Jacob et al., CVPR 2018]

Quantization

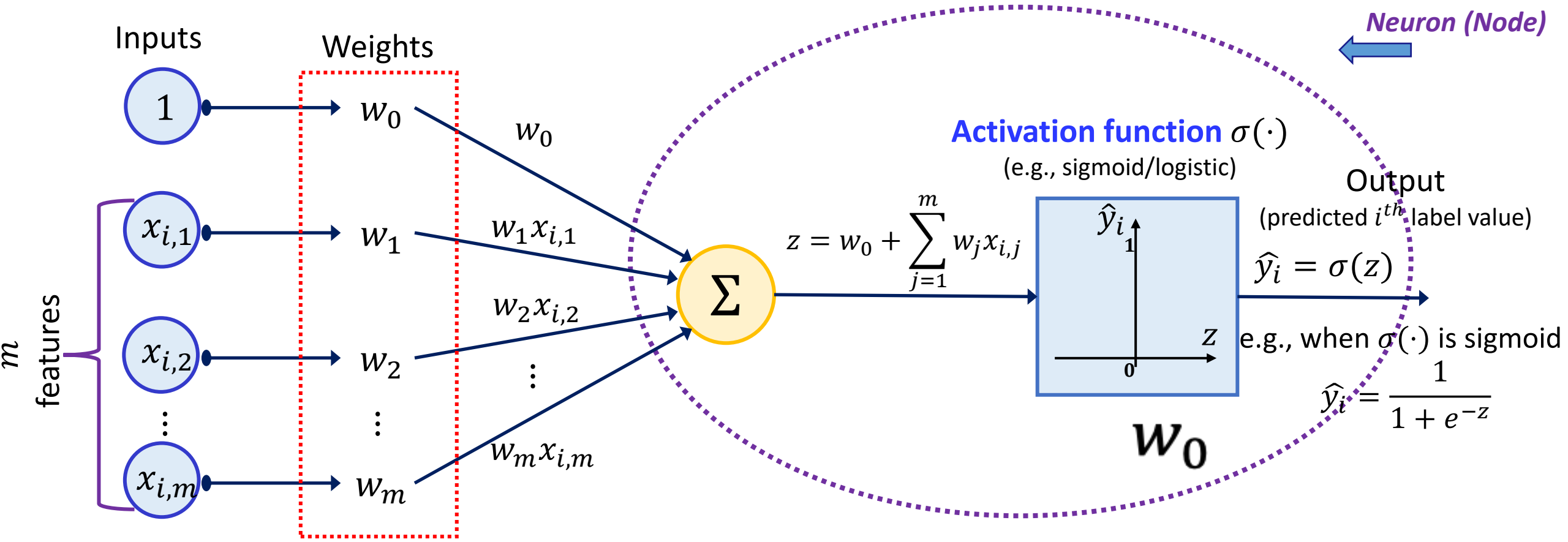
Quantization in Neural Networks

1. Theory in Quantization
    - Linear quantization

2. Post-Training Quantization (PTQ)
– How can we get optimal linear quantization parameters (S, Z)?
    - Weight quantization
    - Activation quantization
    - Bias quantization

3. Quantization-Aware Training

Post Training Quantization (PTQ)



w_0: **bias term** – models the value when all feature values $x_{i,j} = 0$ for $j = 1, \ldots, m$

Quantization

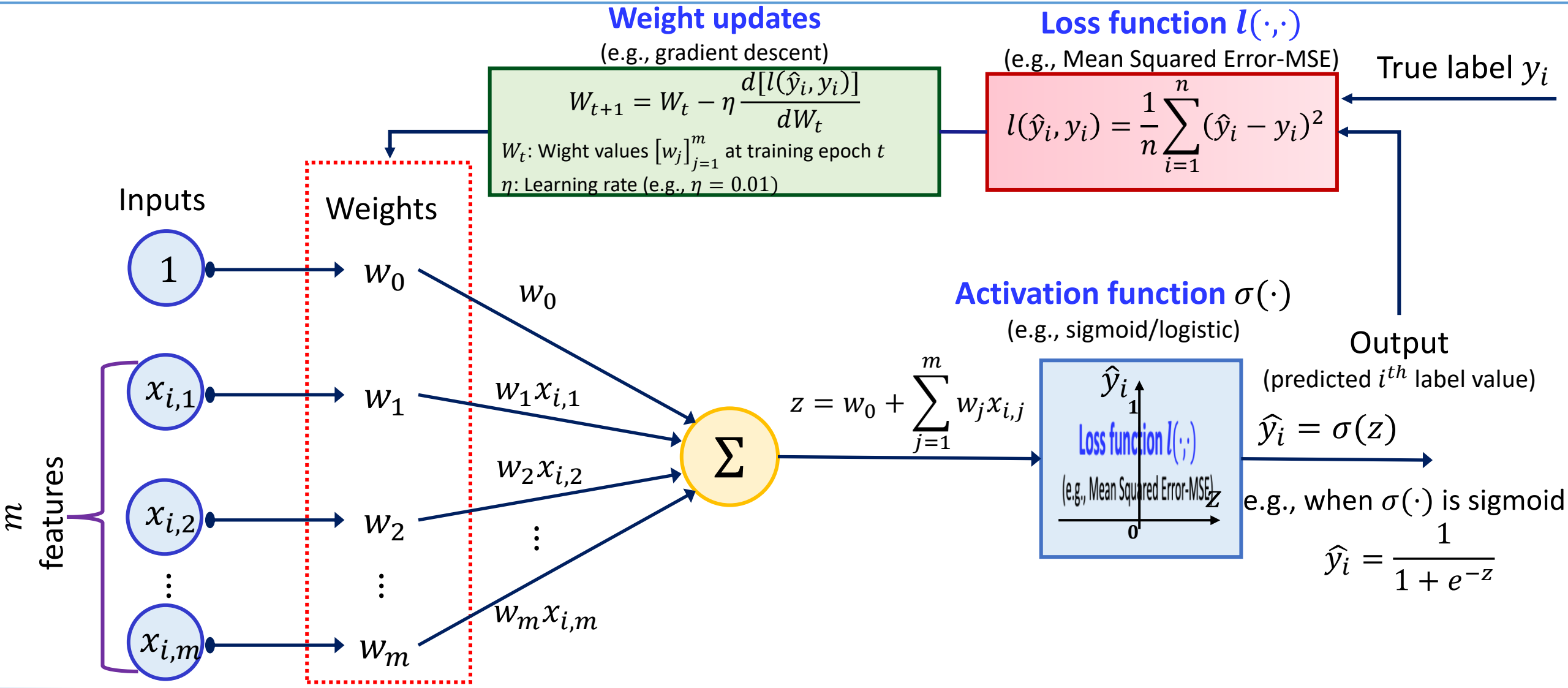
Quantization in Neural Networks

1. Theory in Quantization
    - K-mean based quantization

2. Post-Training Quantization (PTQ)
    - Weight quantization
    - Activation quantization
    - Bias quantization

3. Quantization-Aware Training (QAT)
– How should we improve the performance of quantized models?

Quantization-Aware Training (QAT)



Quantization-Aware Training (QAT)

- Quantization-Aware Training = **simulate int8 quantization during training**
- Model **learns to adapt to quantization (rounding/clipping) errors**
- Leads to high-accuracy int8 models

1. Start with float32 model
2. Insert fake quant ops
3. Train with QAT
4. Convert to real int8 model for deployment

QAT Workflow

Quantization

What algorithms to choose to improve accuracy?

Post-Training Quantization vs Quantization-Aware Training

✓ : Pros

✗ :Cons

Post-Training Quantization (PTQ)

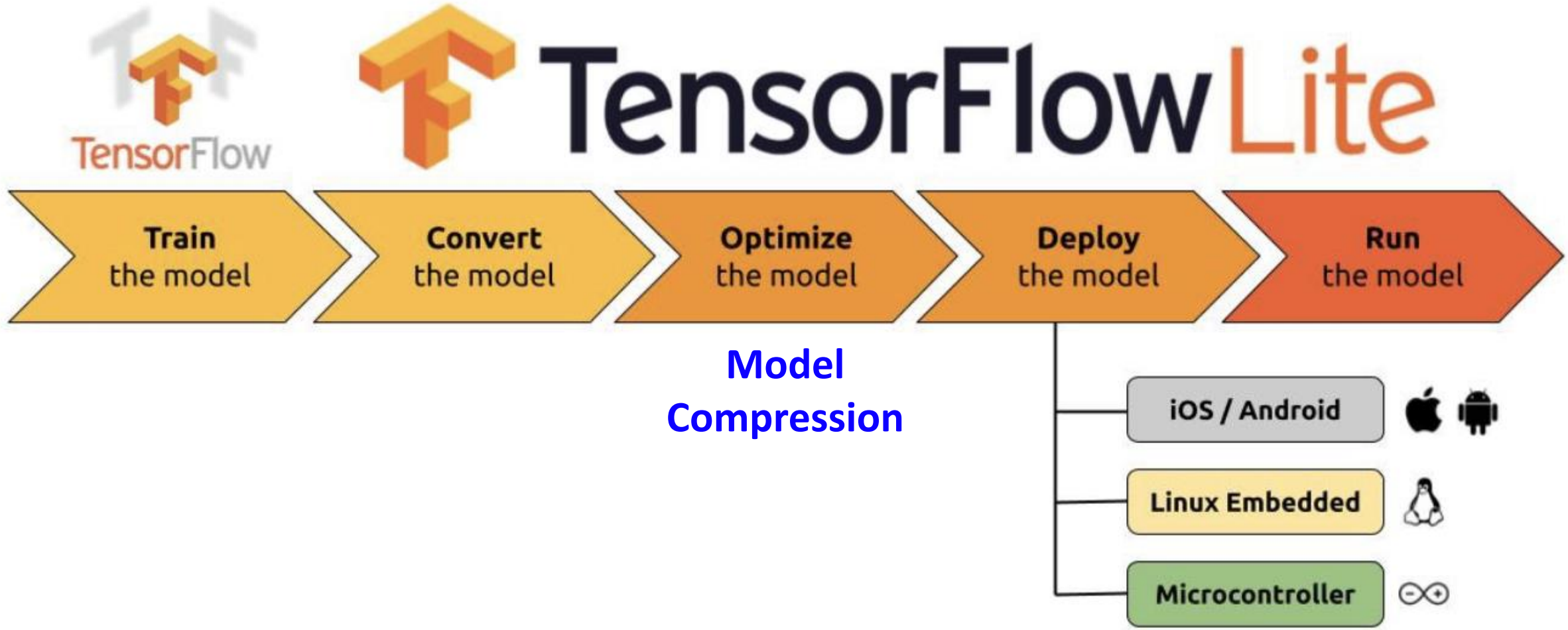
- ✓ Takes a pre-trained network and converts it to a fixed-point network without access to the training pipeline
- ✓ Data-free or small calibration set needed
- ✓ Use though single API call
- ✗ Lower accuracy at lower bit-widths

Quantization-Aware Training (QAT)

- ✗ Requires access to training pipeline and labelled data
- ✗ Longer training times
- ✗ Hyper-parameter tuning
- ✓ Achieves higher accuracy

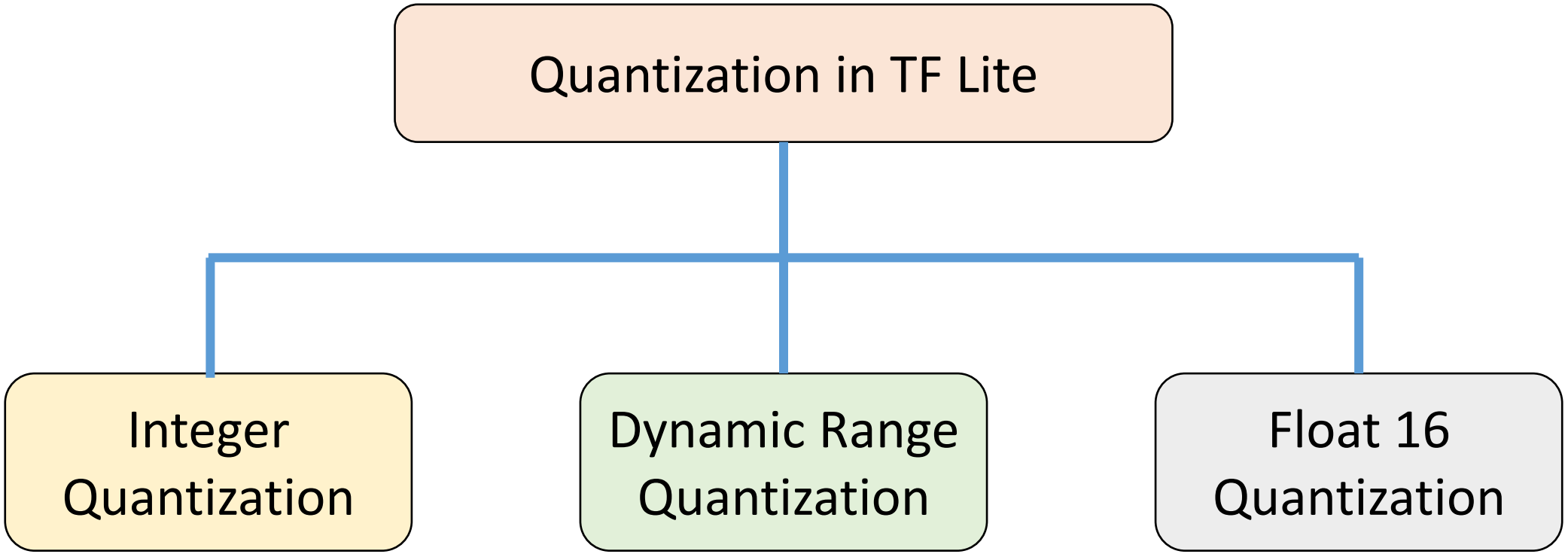
Source: https://www.tinyml.org/event/tinyml-talks-a-practical-guide-to-neural-network-quantization/

TensorFlow (TF) and TFLite Workflow for TinyML



Source: https://leonardocavagnis.medium.com/tinyml-machine-learning-for-embedded-system-part-i-92a34529e899

Quantization in TensorFlow Lite



Integer Quantization

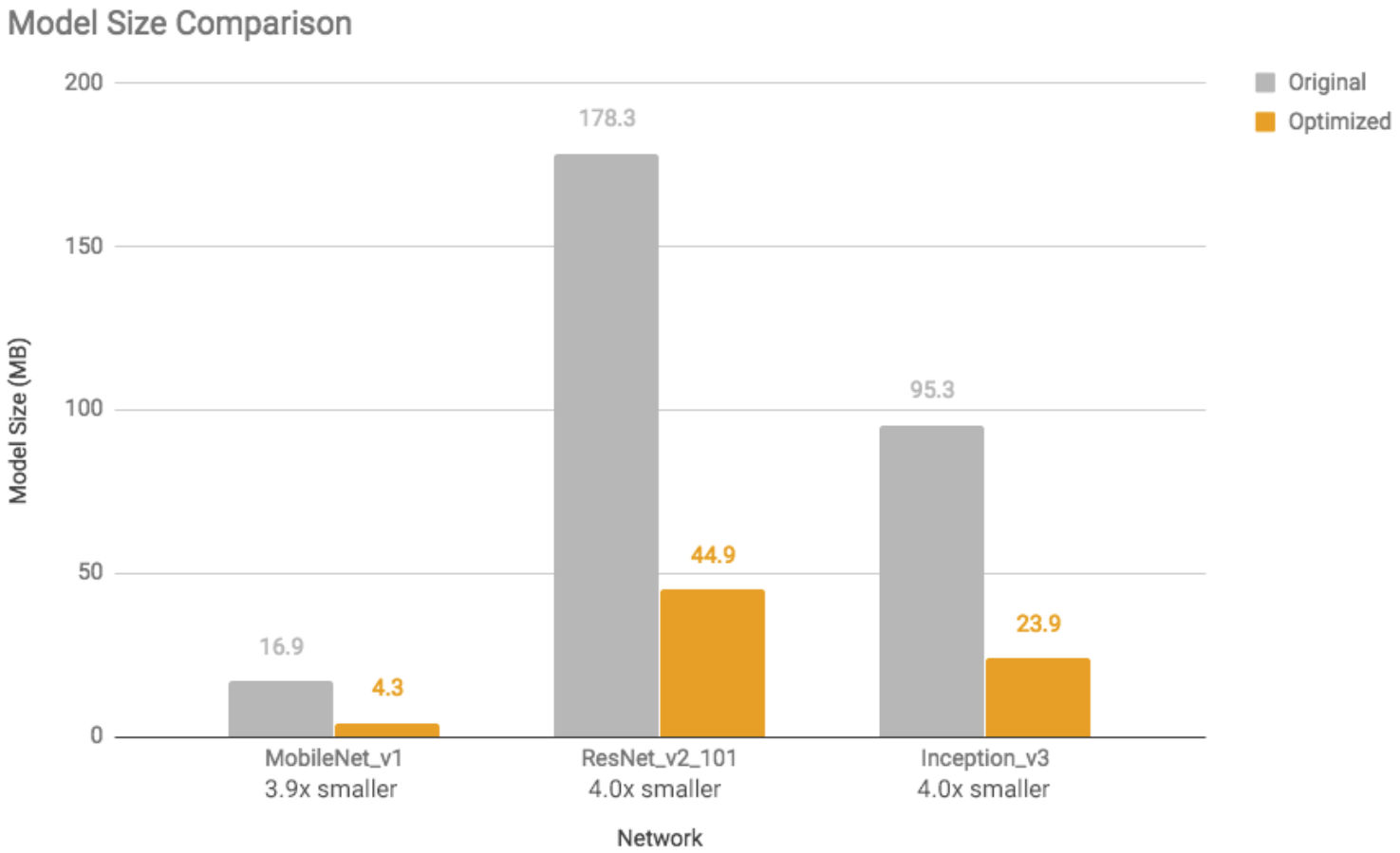
- Full integer quantization means **convert all weights and activation outputs into 8-bit integer data**—whereas other strategies may leave some amount of data in floating-point.

- **Requires a calibration dataset**: A small, representative dataset is used to determine the range of activations for accurate quantization.

- Results in a **smaller model and increased inferencing speed**, which is valuable for low-power devices such as microcontrollers.

Integer Quantization

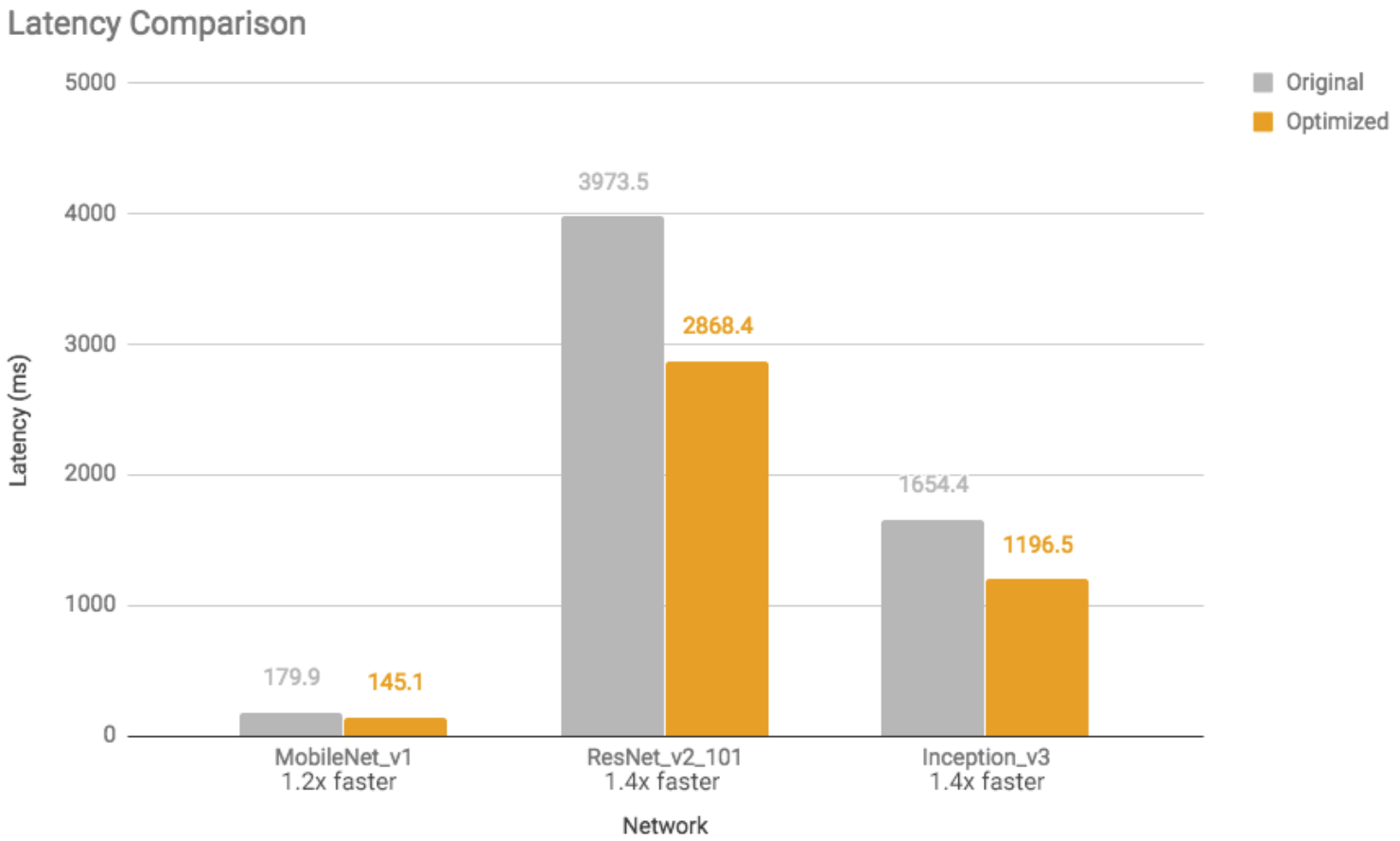
Performance: Model Size



Source: https://blog.tensorflow.org/2018/09/introducing-model-optimization-toolkit.html

Integer Quantization

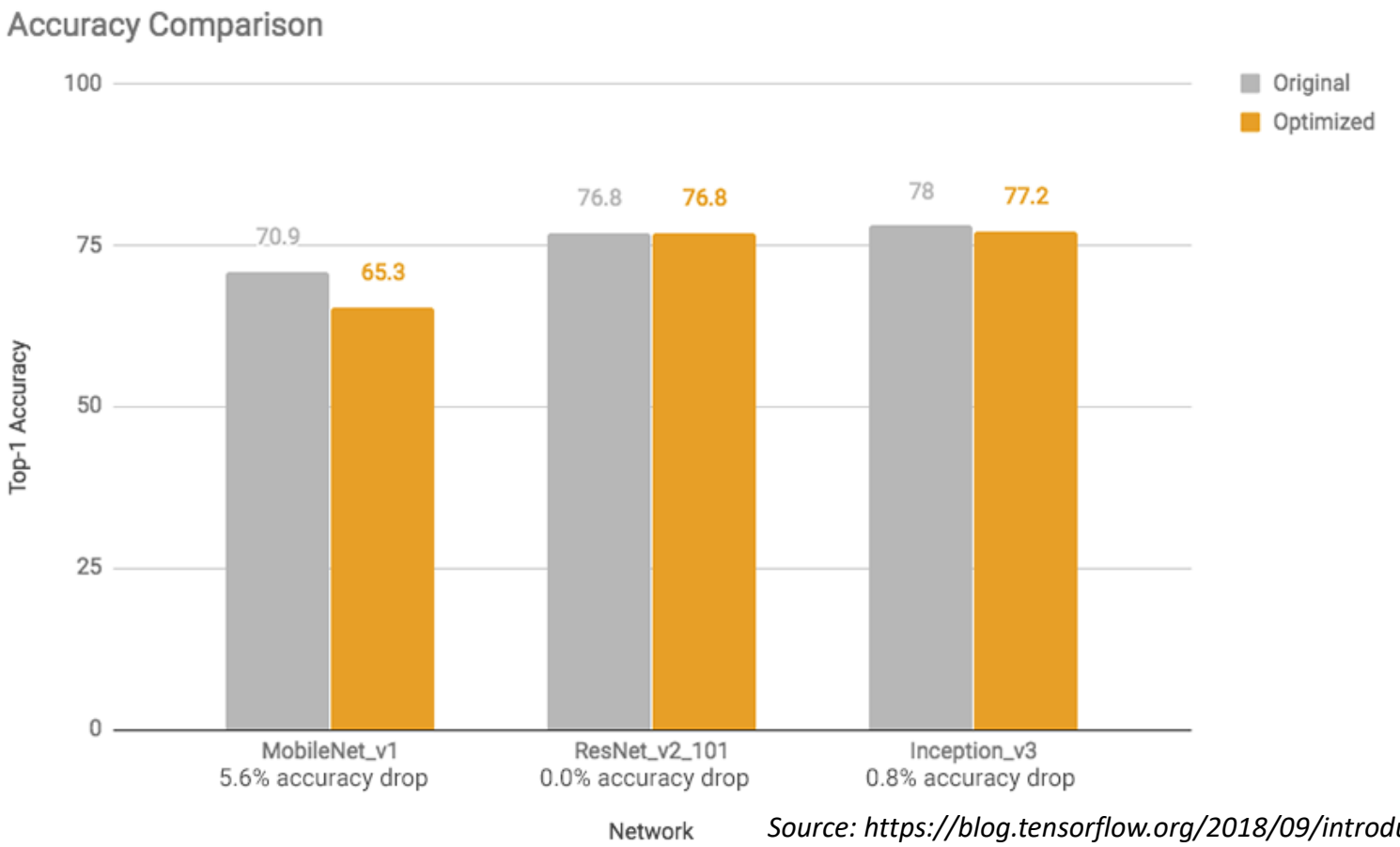
Performance: Latency



Source: https://blog.tensorflow.org/2018/09/introducing-model-optimization-toolkit.html

Integer Quantization

Performance: Accuracy



Source: https://blog.tensorflow.org/2018/09/introducing-model-optimization-toolkit.html

Dynamic Range Quantization

- **Converts only weights to 8-bit precision**.
- **Keep activations in float32 during inference**. Unlike full int8 quantization, activations are **not quantized.**
- **No calibration data needed**: Since activations are not quantized, there's **no need to determine activation ranges or scale them**.
- **Inference computation uses float32 activations and dequantized weights**: Quantized weights are **converted back to float32** before computations.
- **Achieves a 4× reduction in model size** (for the weights only), while maintaining nearly the same accuracy as the original float32 model.

Float-16 Quantization

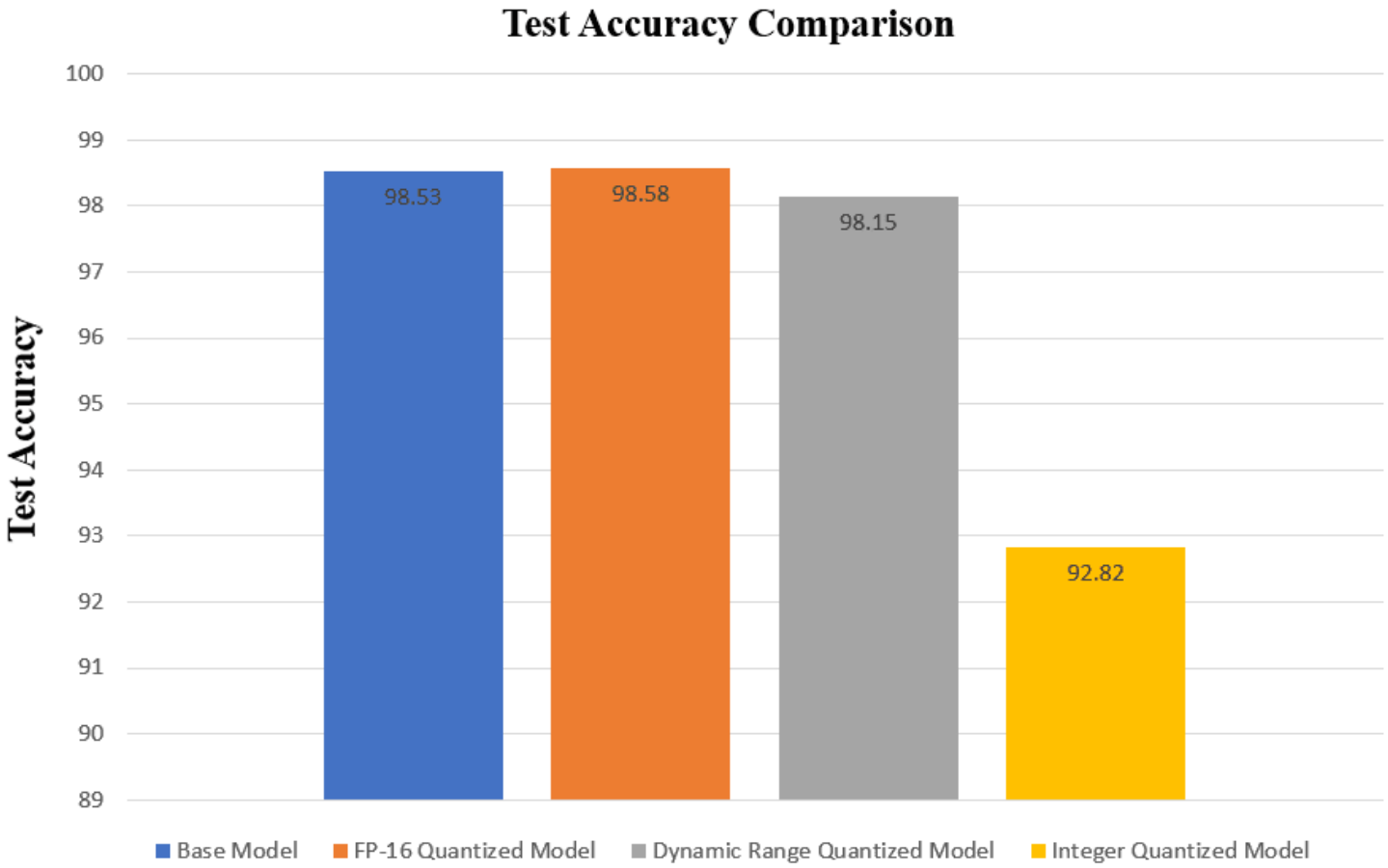
- **Convert weights to 16-bit floating point values** during model conversion from TensorFlow to TF Lite's flat buffer format.

Original float32 Weights		Applying Float16 Quantization
[0.25890732, -1.4732046, 0.9910725]		[0.2589, -1.473, 0.9911]

- Results in a 2x reduction in model size for a minimal impacts on latency and accuracy.
- GPUs, can compute natively in this reduced precision arithmetic, realizing a speedup over traditional floating point execution.

Quantization in TinyML: Comparison

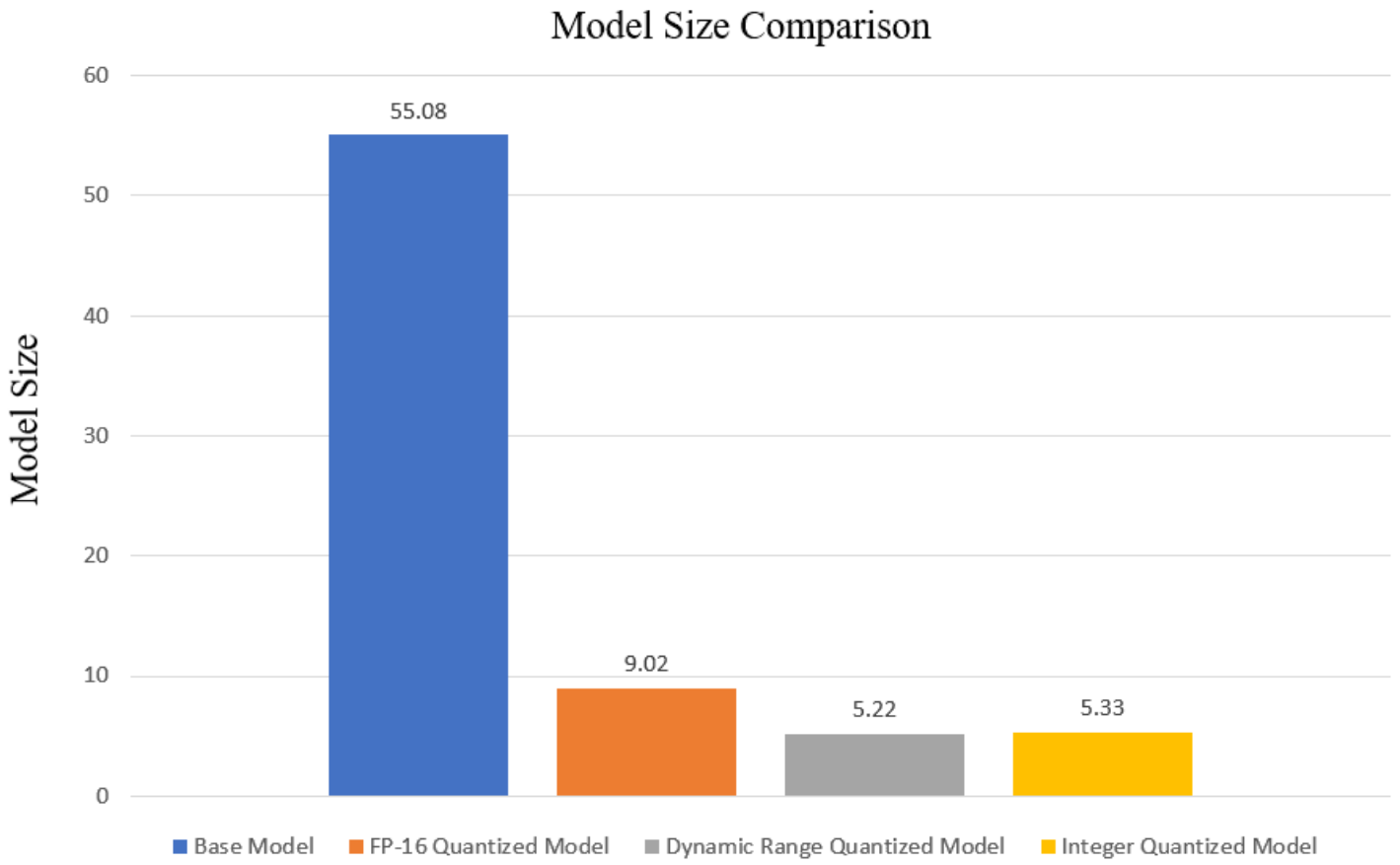
Among Integer & Dynamic Range & Float 16 Quantization



Source: https://learnopencv.com/tensorflow-lite-model-optimization-for-on-device-machine-learning/

Quantization in TinyML: Comparison

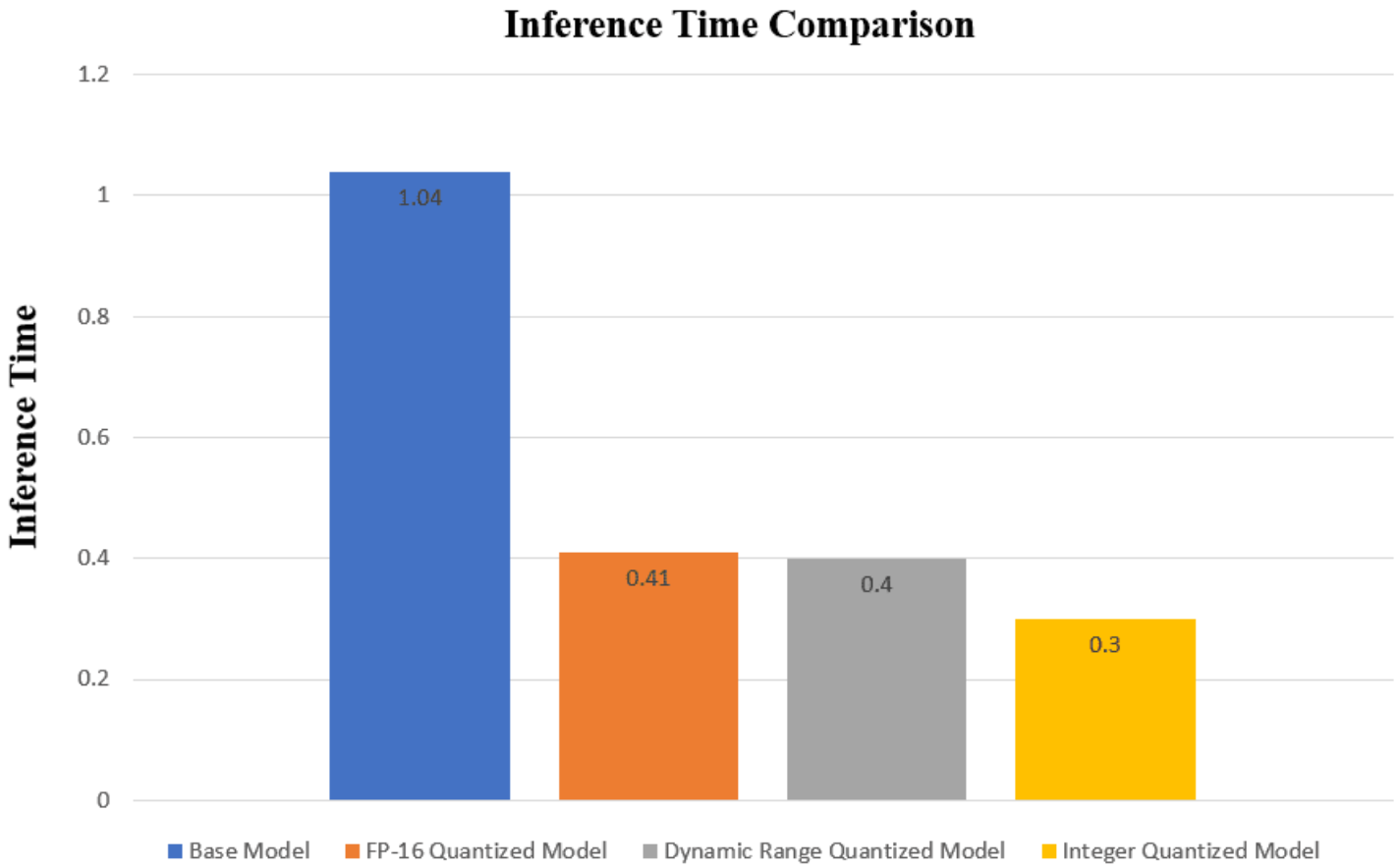
Among Integer & Dynamic Range & Float 16 Quantization



Source: https://learnopencv.com/tensorflow-lite-model-optimization-for-on-device-machine-learning/

Quantization in TinyML: Comparison

Among Integer & Dynamic Range & Float 16 Quantization



Source: https://learnopencv.com/tensorflow-lite-model-optimization-for-on-device-machine-learning/

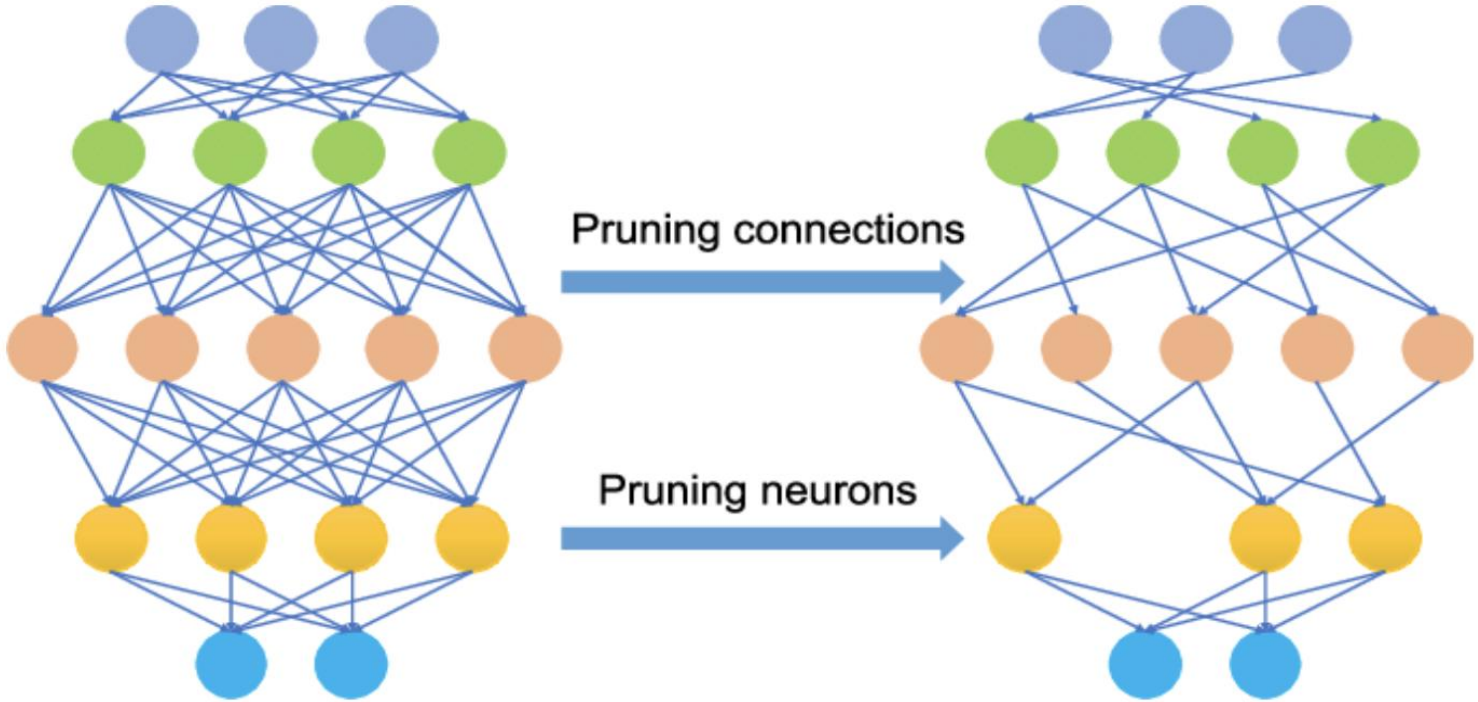
Quantization in TinyML: Summary

Summary table of the choices and the benefits:

Technique	Benefits	Hardware
Dynamic range quantization	4x smaller, 2x-3x speedup	CPU
Full integer quantization	4x smaller, 3x+ speedup	CPU, Edge TPU, Microcontrollers
Float16 quantization	2x smaller, GPU acceleration	CPU, GPU

Model Compression: Pruning

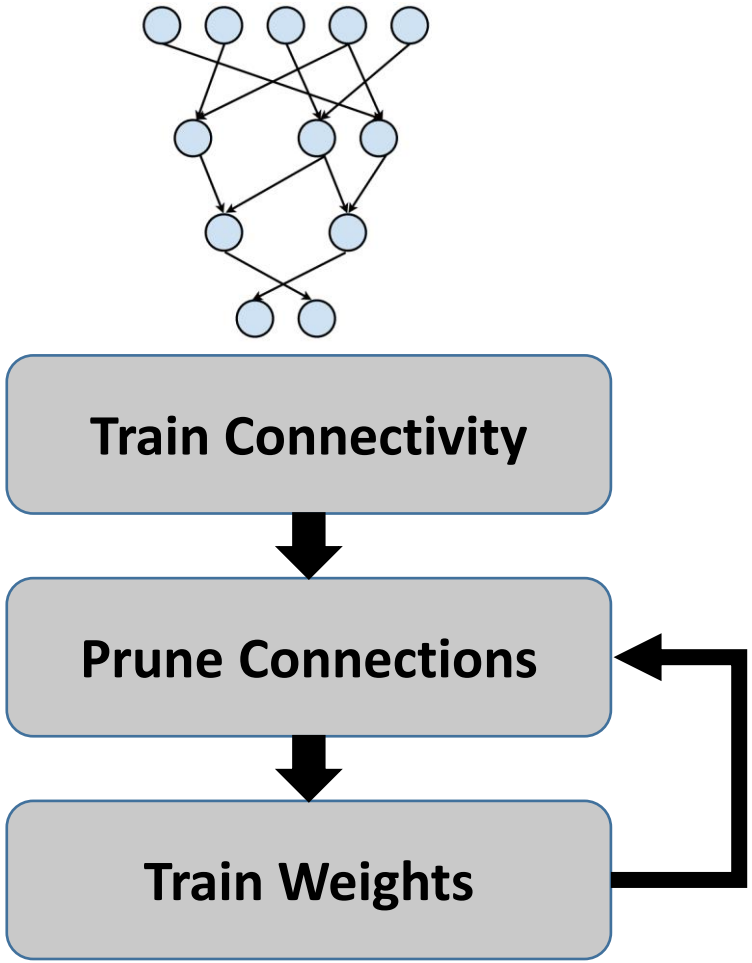
- First creates a **large model** that achieves the **required accuracy**
- Then try and **decrease the size** (prune it) **while trying to maintain the accuracy**
- Once the model has been pruned, this **new pruned model** is **trained again** on the dataset and this is known as **iterative pruning**



Source: https://heartbeat.comet.ml/neural-network-pruning-research-review-2020-bc21a77f0295

Pruning Method

Pruning makes neural network smaller by removing synapses and neurons.



Baseline: a man is riding a surfboard on a wave.

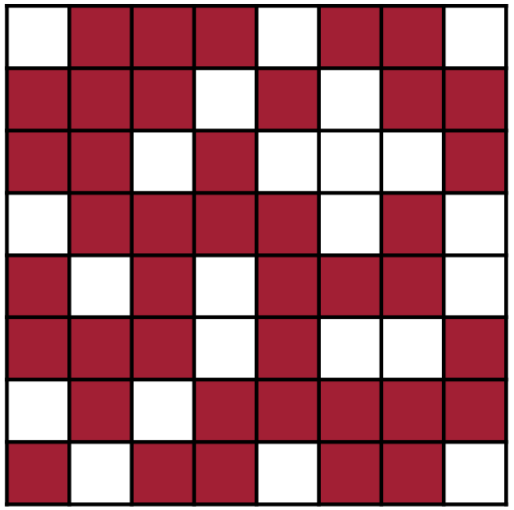
Pruned 90%: a man in a wetsuit is riding a wave on a beach

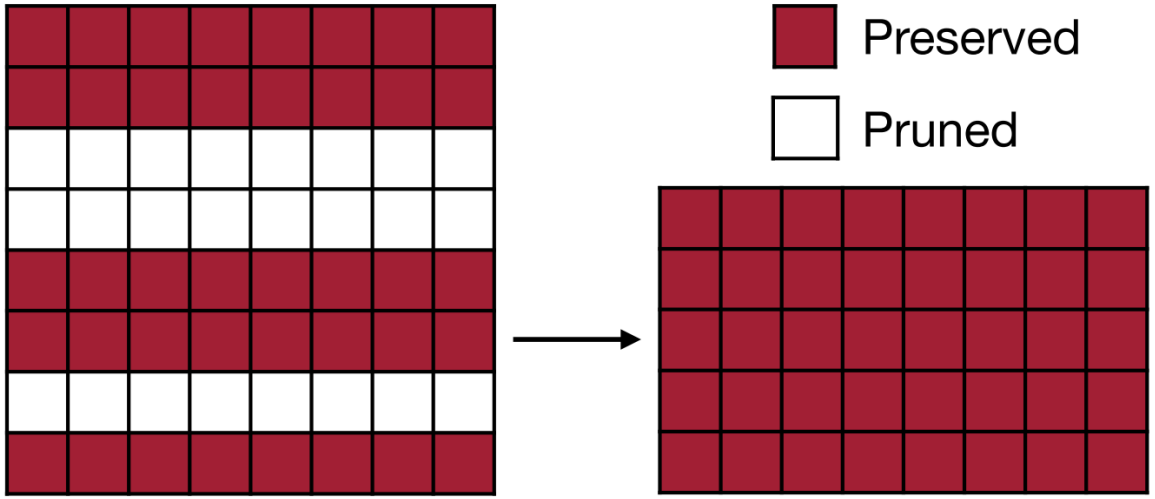
Baseline: a soccer player in red is running in the field.

Pruned 95%: a man in red shirt and black and white black shirt is running through a field.

Pruning Techniques

Pruning can be classified as structured and unstructured based on the granularity.





Fine Grained/ Unstructured Pruning

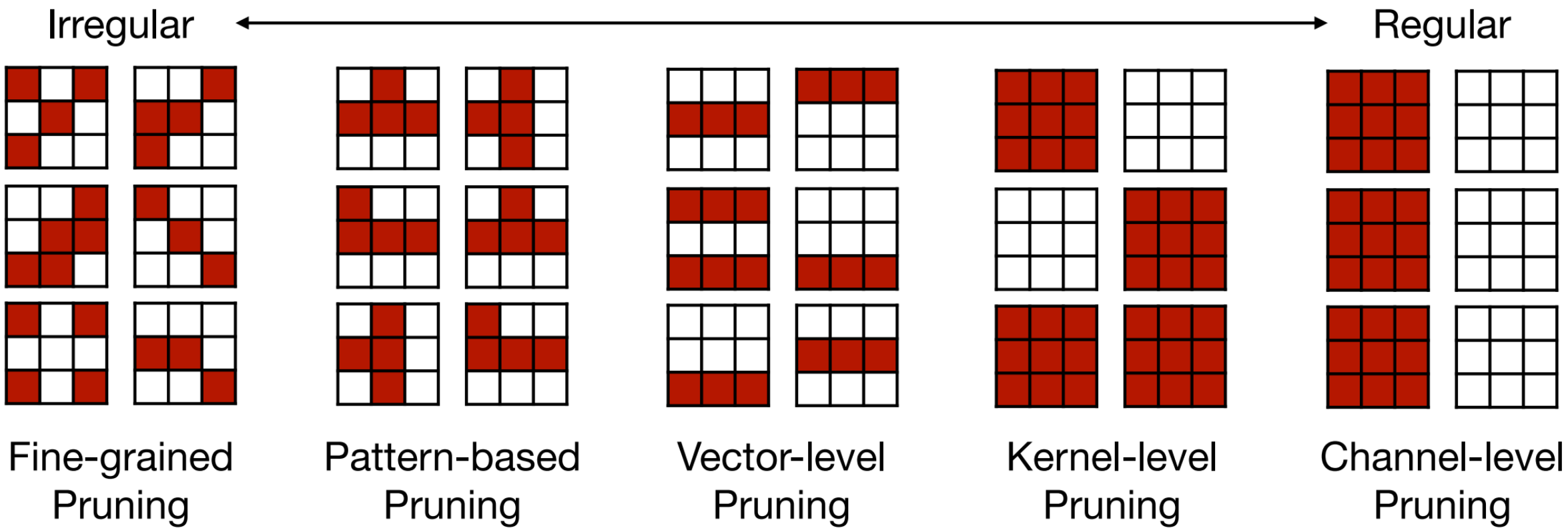
- More flexible pruning index choice
- Hard to accelerate (irregular data expression)

Coarse Grained/ Structured Pruning

- Less flexible pruning index choice.
- Easy to accelerate (since it is a smaller matrix)

Source: https://hanlab.mit.edu/files/course/slides/MIT-TinyML-Lec03-Pruning-I.pdf

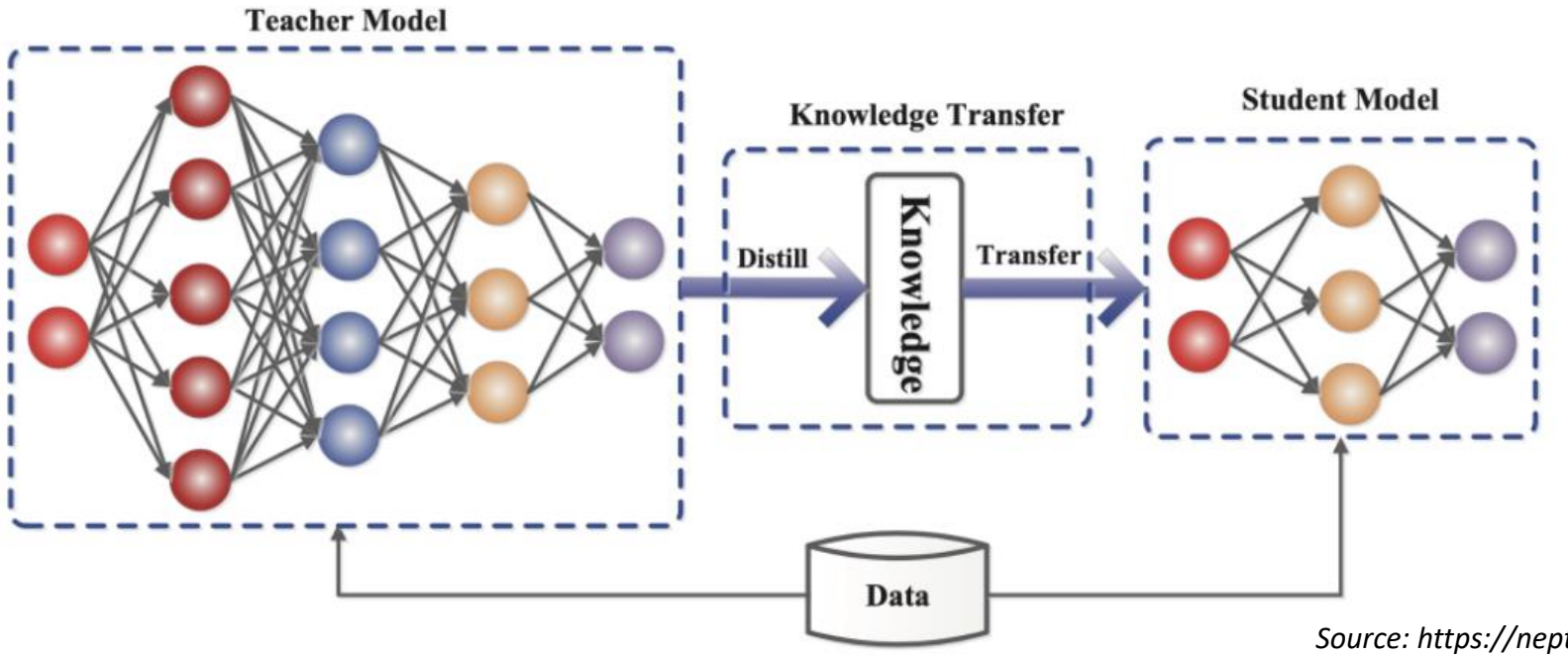
Pruning Types: Granularity Based



Source: https://hanlab.mit.edu/files/course/slides/MIT-TinyML-Lec03-Pruning-I.pdf

Model Compression: Knowledge Distillation

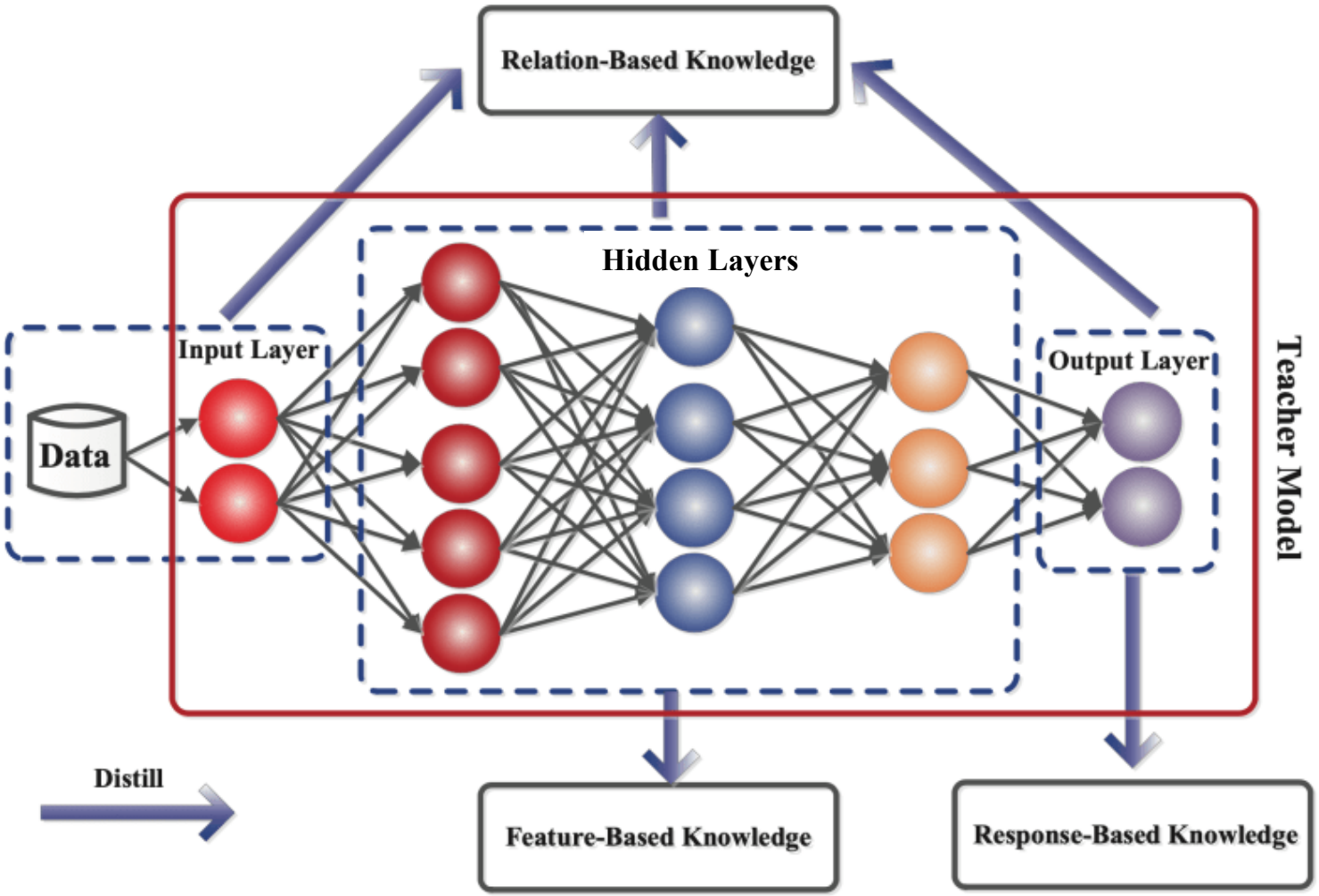
- Effectively training a **bigger ML model**, or the **TEACHER**, with a given dataset.
- The model would generate predictions based on the given data.
- A **new dataset** would be constructed which would **combine the old data and the information generated from the bigger model**.
- This new dataset would be then fed to the **smaller ML model**, or the **STUDENT**.



Source: https://neptune.ai/blog/knowledge-distillation

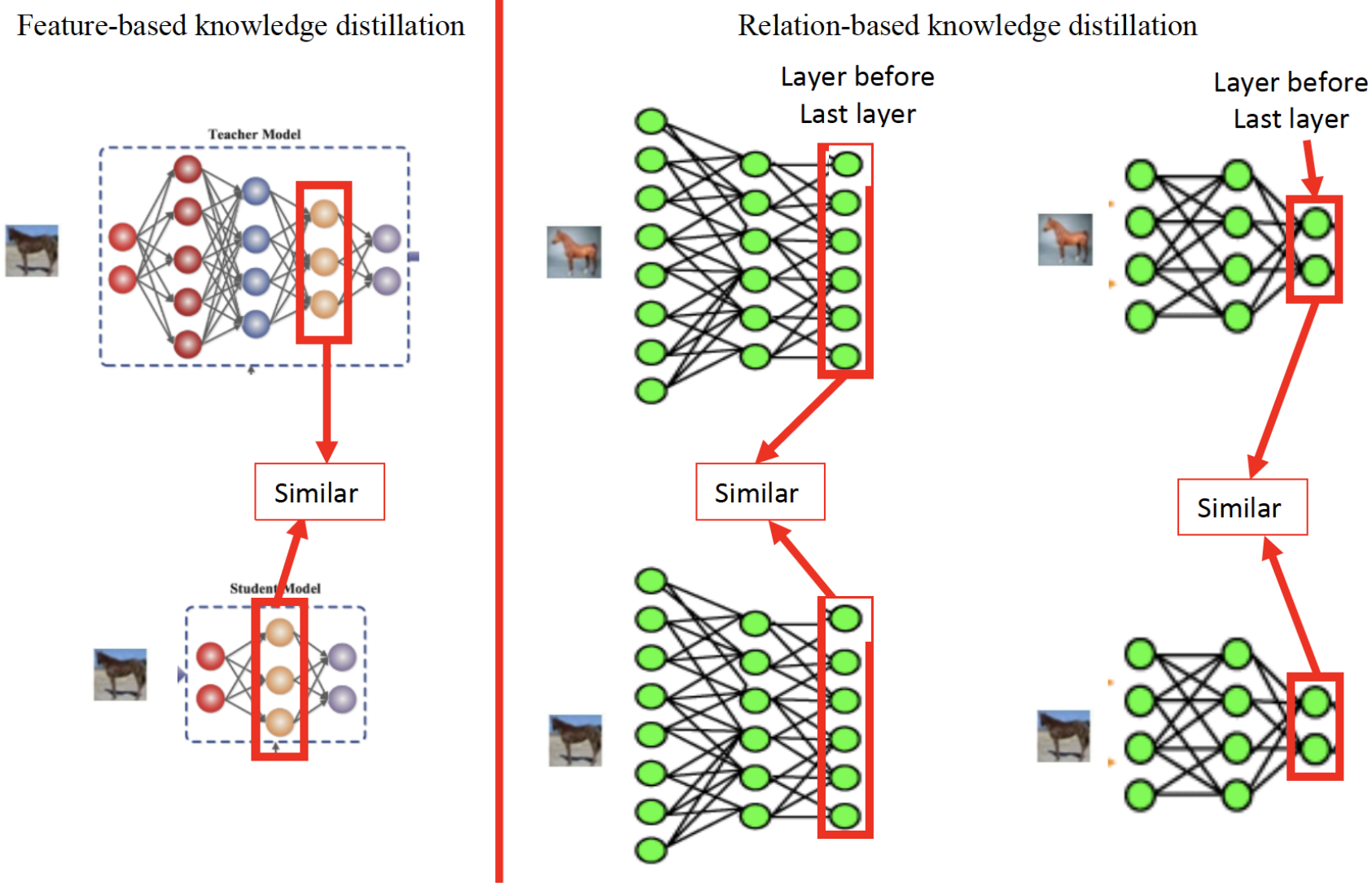
Knowledge Distillation Types

- **Feature-Based Knowledge**: Matching intermediate representations/features
- **Response-Based Knowledge**: Matching output logits or probabilities
- **Relation-Based Knowledge**: Matching the relational structure between data points or features.
  - i.e., if two inputs are deemed similar by the teacher model, the student model should also learn to recognize this similarity
  - ***Applications***: Computer Vision, Natural Language Processing

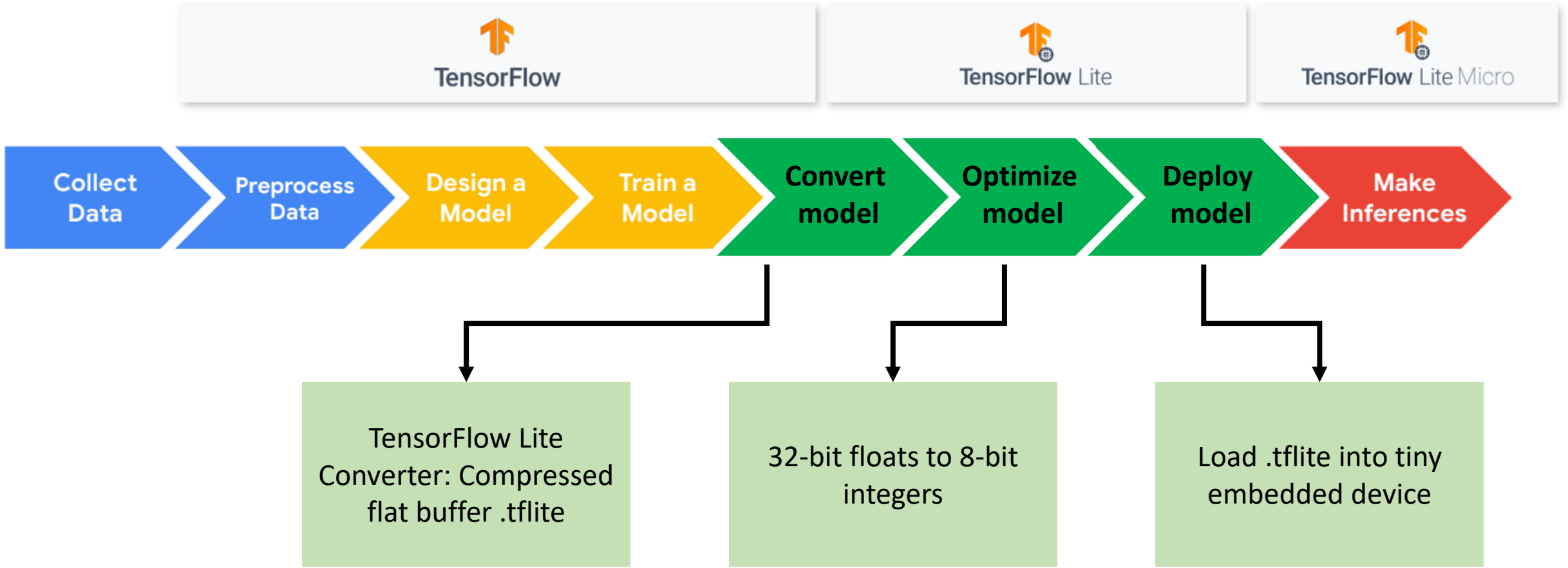


Source: https://neptune.ai/blog/knowledge-distillation

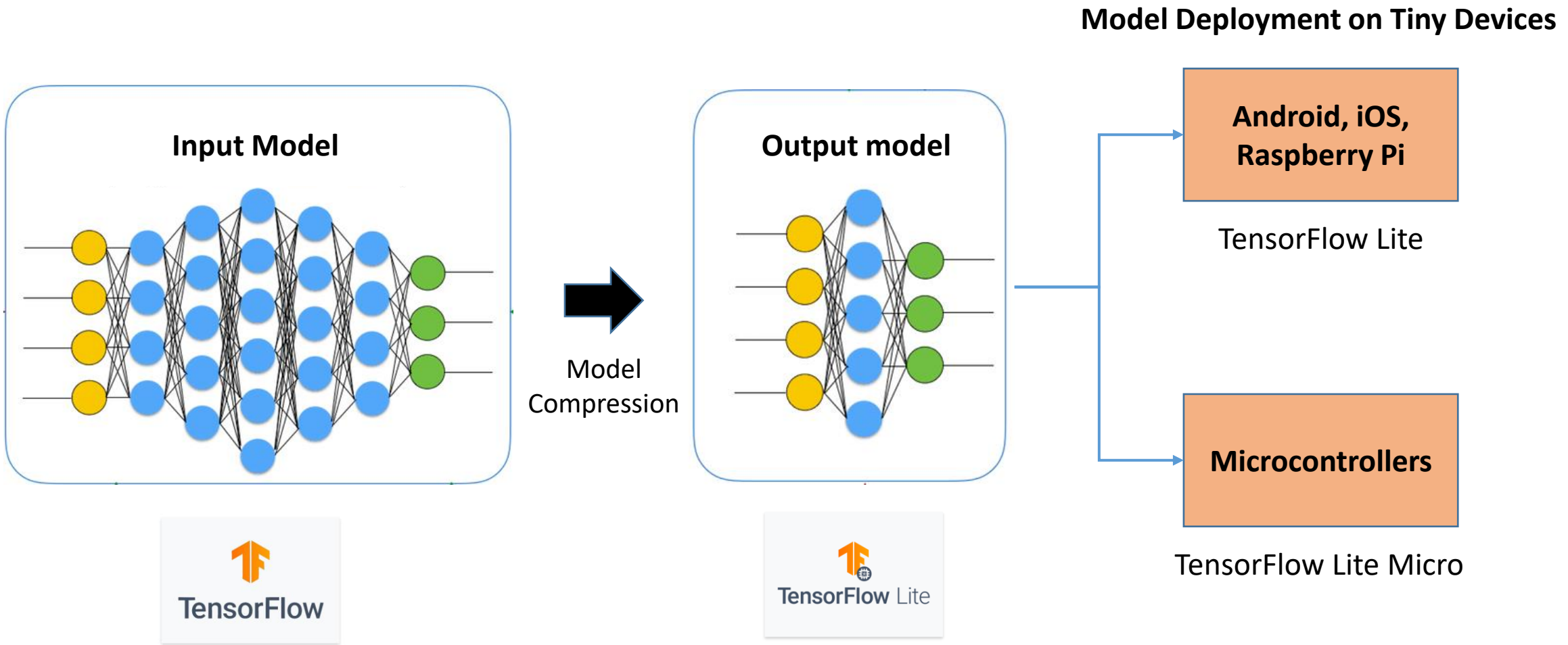
Knowledge Distillation: Feature-based Vs. Relation-based



Putting it all together – TF -> TF Lite for TinyML

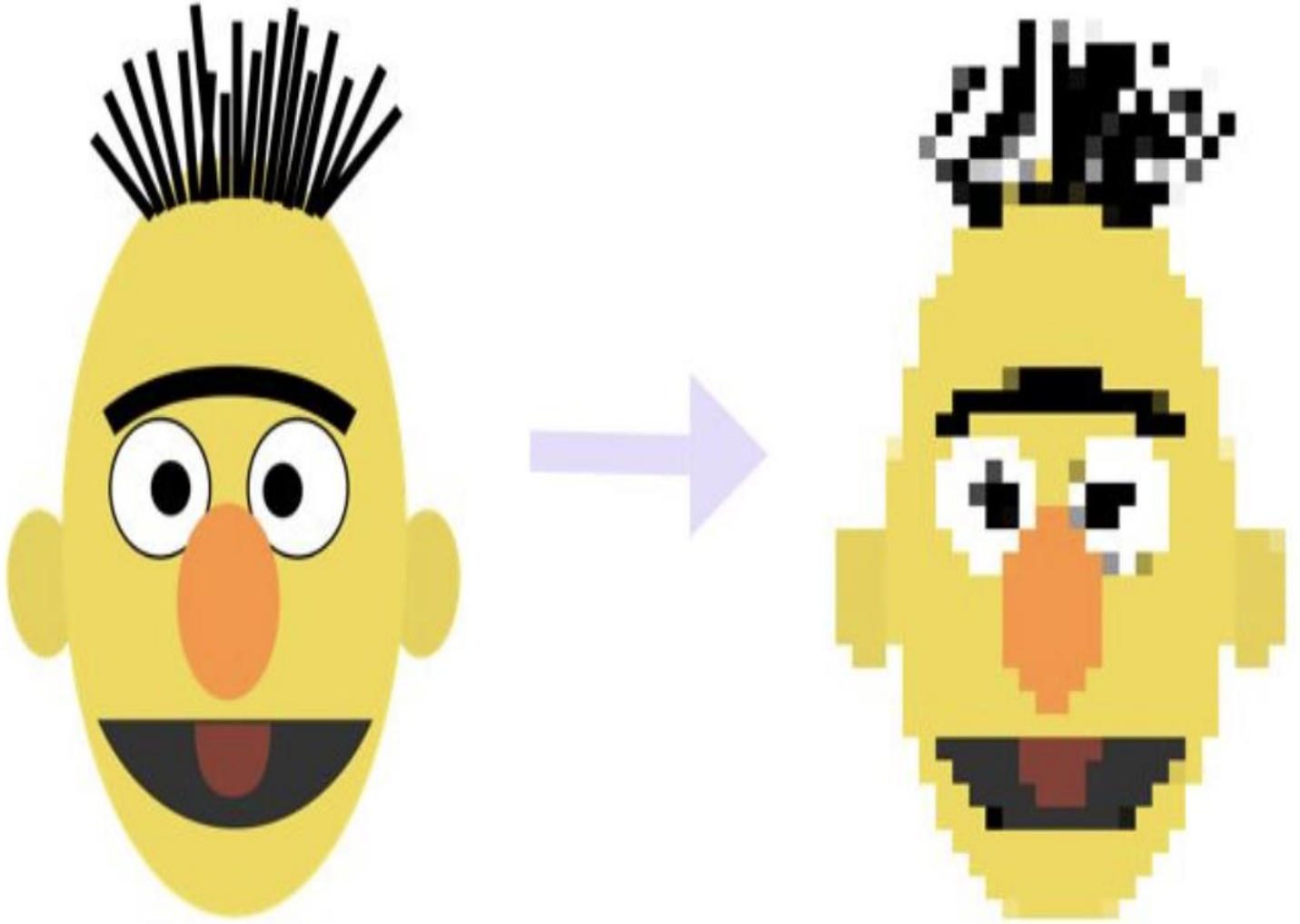


TFLite: Model Compression and Deployment



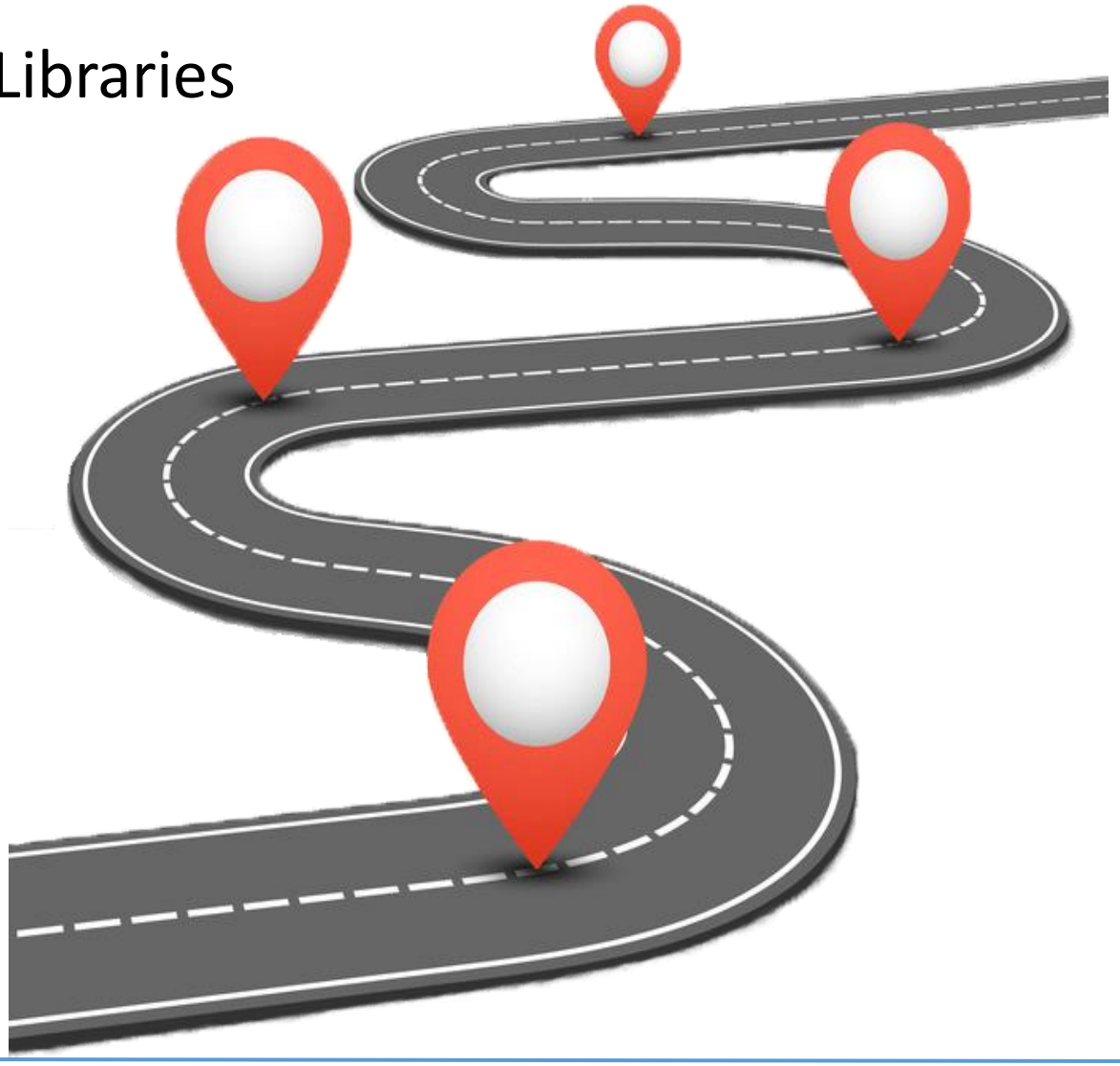
Coding Session

- Training a simple CNN model using MNIST Dataset
- Quantizing CNN model
- Pruning CNN model
- Applying KD on CNN
- Evaluating the performance of compressed CNN Models



Road Map

1. Installing and Configuring Needed Python Libraries
2. Preparing MNIST Dataset
3. Creating a simple CNN model
4. Compiling the CNN Model
5. Training the CNN Model
6. Evaluating the Trained CNN Model



Road Map

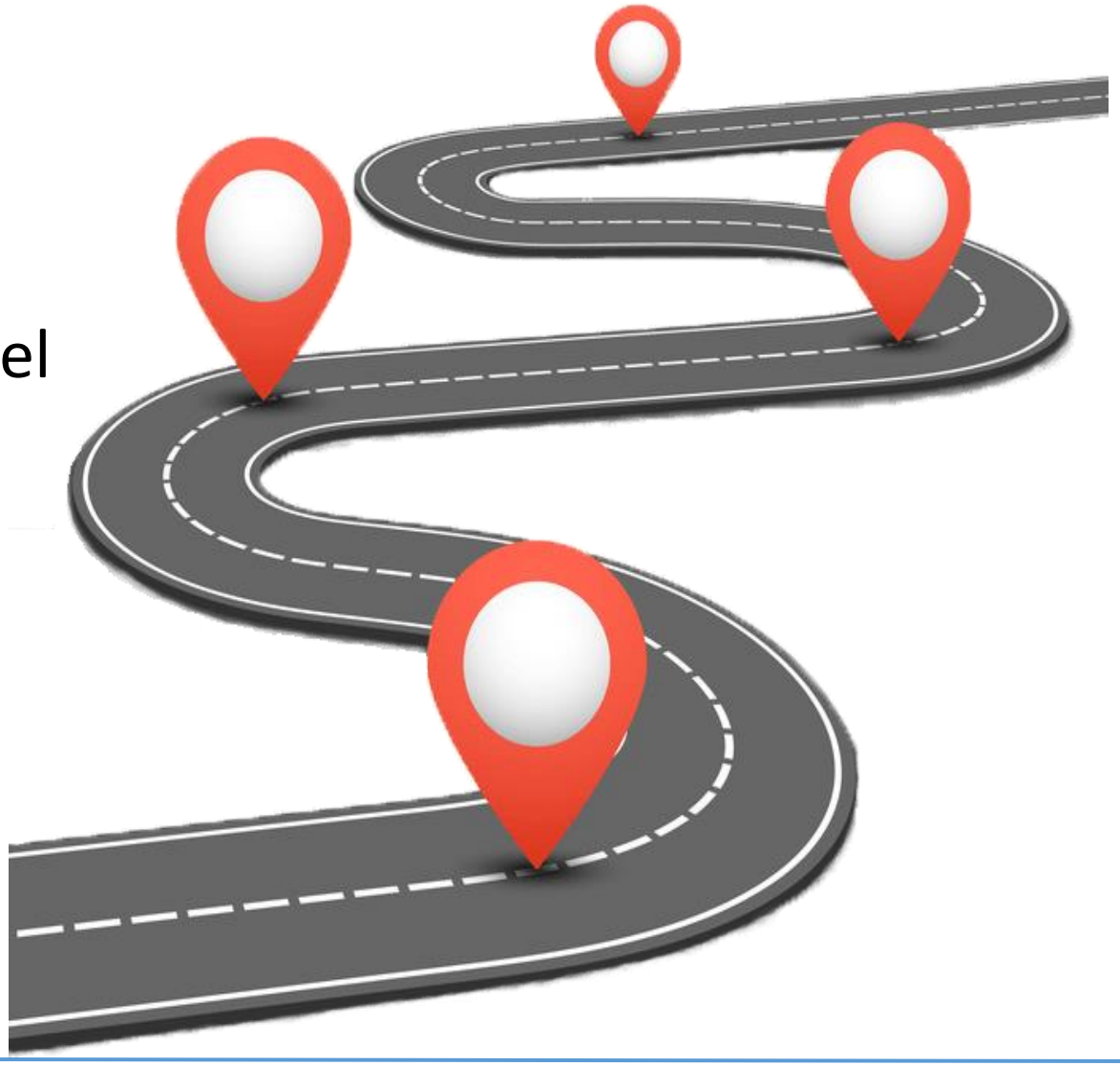
7. Integer Quantization on CNN Model

8. Dynamic Range Quantization on CNN Model

9. Float-16 Quantization on CNN Model

10. Weight Pruning on CNN Model

11. Knowledge Distillation on CNN Model



Broader Applications of TinyML

- **Model Compression for Resource-Constrained Deployment**
  Reduce model size using techniques like quantization, pruning, and knowledge distillation — enabling ML on microcontrollers and edge devices.

- **Reducing Energy and Latency for Real-Time Systems**
  Compressed models running on-device consume less power and eliminate round-trip delays — crucial for IoT, wearables, and robotics.

- **Enabling Always-On Sensing in Low-Power Environments**
  TinyML allows deployment of continuous monitoring systems (e.g., keyword spotting, gesture recognition) on devices powered by coin-cell batteries or solar.

Broader Applications of TinyML

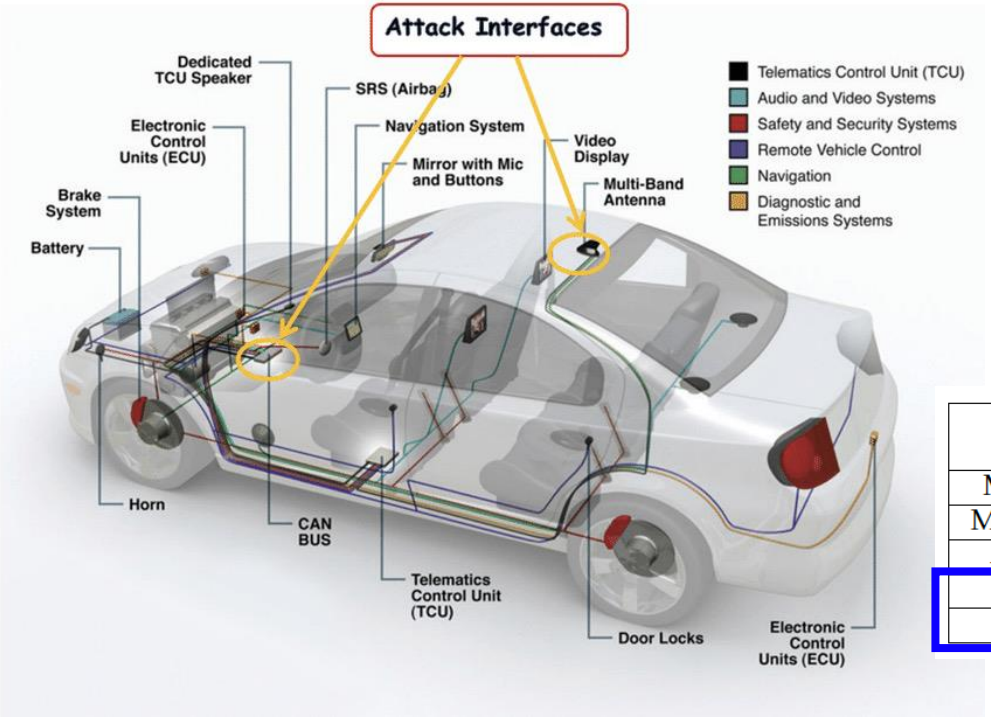
- **Preserving Privacy Through On-Device Inference**
  Keeping inference local avoids transmission of sensitive data to the cloud, supporting privacy-critical applications like health monitoring and smart home systems.

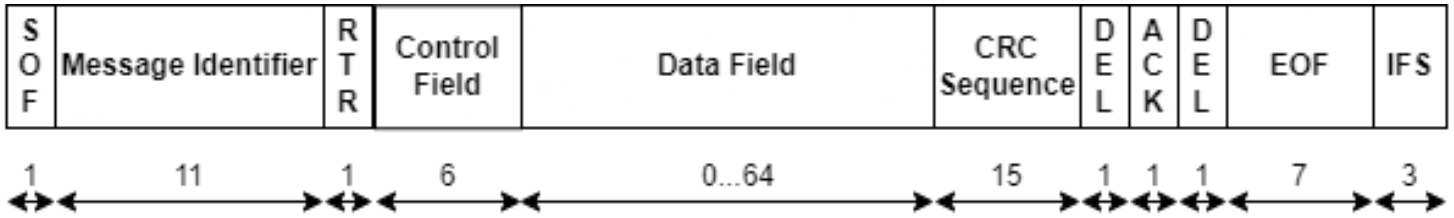
- **Mitigating Overfitting on Small Datasets**
  Smaller, regularized TinyML models generalize better when training data is limited — which is common in embedded applications.

- **Pruning as a Defense Against Adversarial Attacks**
  By removing redundant or low-salience connections, pruning can increase model robustness and reduce susceptibility to adversarial perturbations.

CANLP: Quantized ML Model Designed for Intrusion Detection in CAN Bus Systems

- The CAN protocol is one of the most widely used in-vehicle networking standard but lacks security features leaving it vulnerable to cyber attacks: DoS, Spoofing, Replay, Fuzzing, and Masquerade





Method	Hardware Platform	CPU	F1-Score	Latency	
				Mean	Std
MTH-IDS [16]	Raspberry Pi 3 Model B	1.2 GHz Broadcom BCM2837	0.999	1.722 ms	NR
MA-QCNN [12]	Zynq FPGA	1.3 GHz Arm Cortex-A53 MPCore	0.996	1.290 ms	NR
ACGAN [17]	Raspberry Pi 4 Model B	1.8 GHz ARM Cortex-A72	0.992	0.538 ms	0.030 ms
CANLP	**Raspberry Pi 3 Model B**	**1.2 GHz Broadcom BCM2837**	**0.997**	**0.213 ms**	**0.030 ms**
CANLP	**Raspberry Pi 4 Model B**	**1.8 GHz ARM Cortex-A72**	**0.997**	**0.049 ms**	**0.011 ms**

Balasubramanian, Kavya, Adithya Gowda Baragur, Denis Donadel, Dinuka Sahabandu, Alessandro Brighente, Bhaskar Ramasubramanian, Mauro Conti, and Radha Poovendran. "CANLP: NLP-Based Intrusion Detection System for CAN." In Proceedings of the 39th ACM/SIGAPP Symposium on Applied Computing, pp. 212-214. 2024.

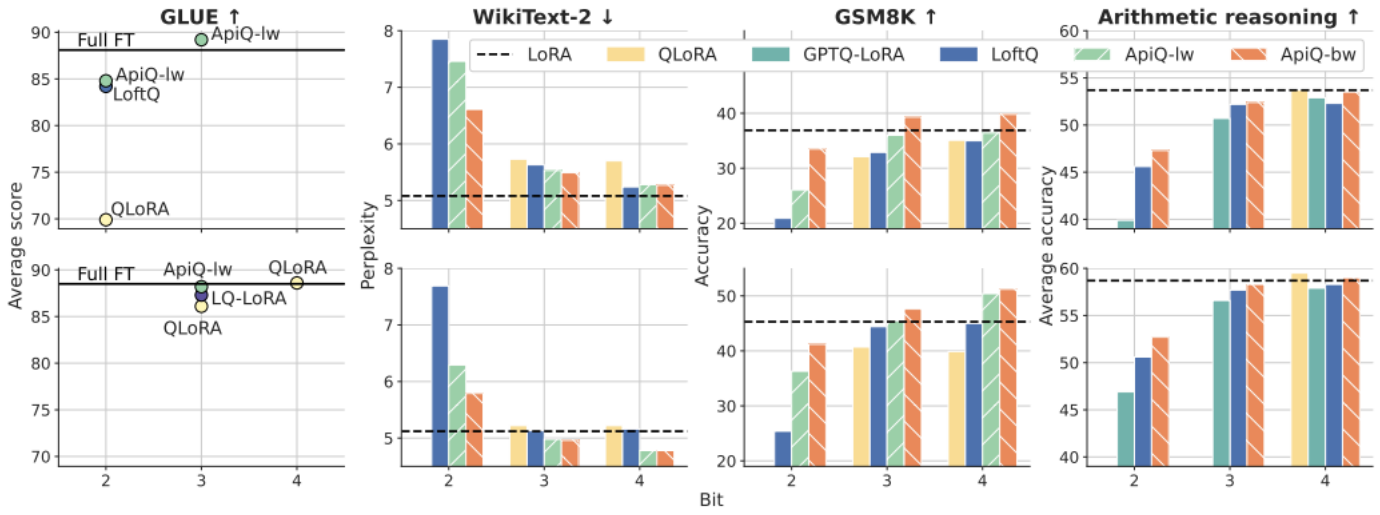
2-bit Large Language Models (LLMs)

ApiQ: Finetuning of 2-Bit Quantized Large Language Model

Baohao Liao[1,2*] **Christian Herold**[2] **Shahram Khadivi**[2] **Christof Monz**[1]
[1]Language Technology Lab, University of Amsterdam
[2]eBay Inc., Aachen, Germany
Code: https://github.com/BaohaoLiao/ApiQ

Abstract

Memory-efficient finetuning of large language models (LLMs) has recently attracted huge attention with the increasing size of LLMs, primarily due to the constraints posed by GPU memory limitations and the effectiveness of these methods compared to full finetuning. Despite the advancements, current strategies for memory-efficient finetuning, such as QLoRA, exhibit inconsistent performance across diverse bit-width quantizations and multifaceted tasks. This inconsistency largely stems from the detrimental impact of the quantization process on preserved knowledge, leading to catastrophic forgetting and undermining the utilization of pretrained models for finetuning purposes. In this work, we introduce a novel quantization framework, *ApiQ*, designed to restore the lost information from quantization by concurrently initializing the LoRA components and quantizing the weights of LLMs. This approach ensures the maintenance of the original LLM's activation precision while mitigating the error propagation from shallower into deeper layers. Through comprehensive evaluations conducted on a spectrum of language tasks with various LLMs, ApiQ demonstrably minimizes activation error during quantization. Consequently, it consistently achieves superior finetuning results across various bit-widths.



Pruning Large Language Models (LLMs)

The Unreasonable Ineffectiveness of the Deeper Layers

Andrey Gromov* Meta FAIR & UMD

Kushal Tirumala* Meta FAIR

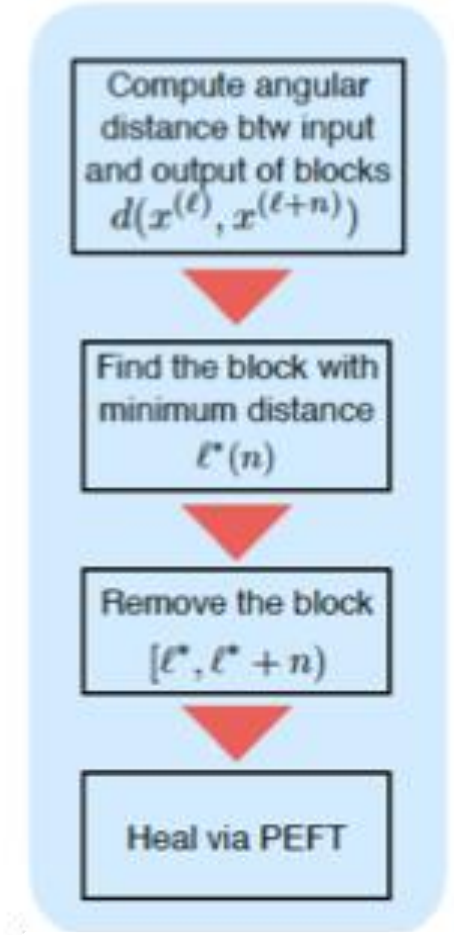
Hassan Shapourian Cisco

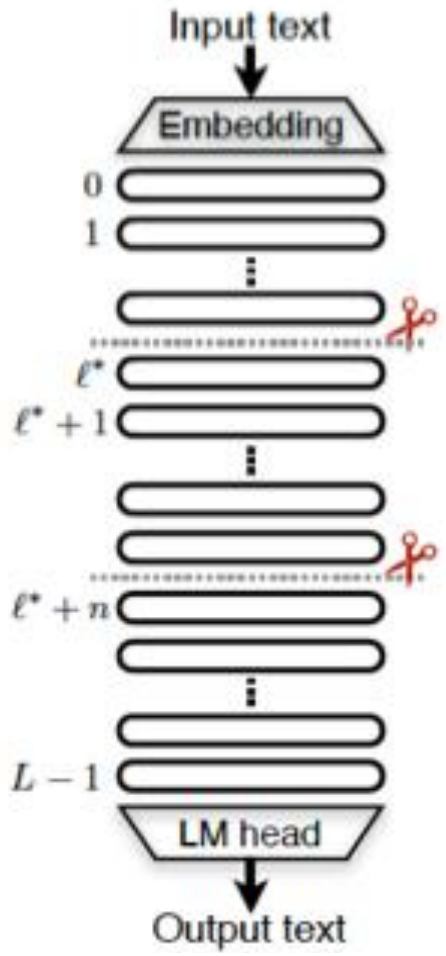
Paolo Glorioso Zyphra

Daniel A. Roberts MIT & Sequoia Capital

Abstract

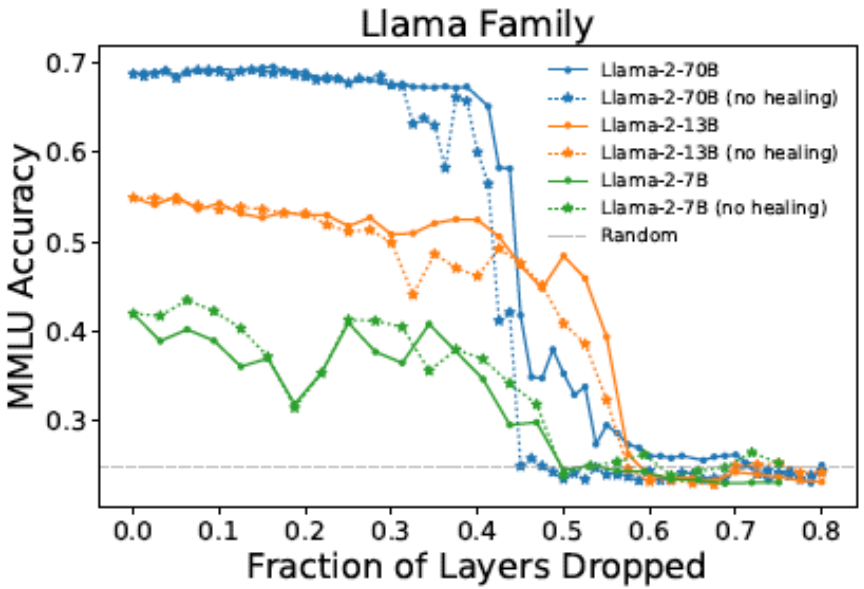
We empirically study a simple layer-pruning strategy for popular families of open-weight pretrained LLMs, finding minimal degradation of performance on different question-answering benchmarks until after a large fraction (up to half) of the layers are removed. To prune these models, we identify the optimal block of layers to prune by considering similarity across layers; then, to "heal" the damage, we perform a small amount of finetuning. In particular, we use parameter-efficient finetuning (PEFT) methods, specifically quantization and Low Rank Adapters (QLoRA), such that each of our experiments can be performed on a single A100 GPU. From a practical perspective, these results suggest that layer pruning methods can complement other PEFT strategies to further reduce computational resources of finetuning on the one hand, and can improve the memory and latency of inference on the other hand. From a scientific perspective, the robustness of these LLMs to the deletion of layers implies either that current pretraining methods are not properly leveraging the parameters in the deeper layers of the network or that the shallow layers play a critical role in storing knowledge.

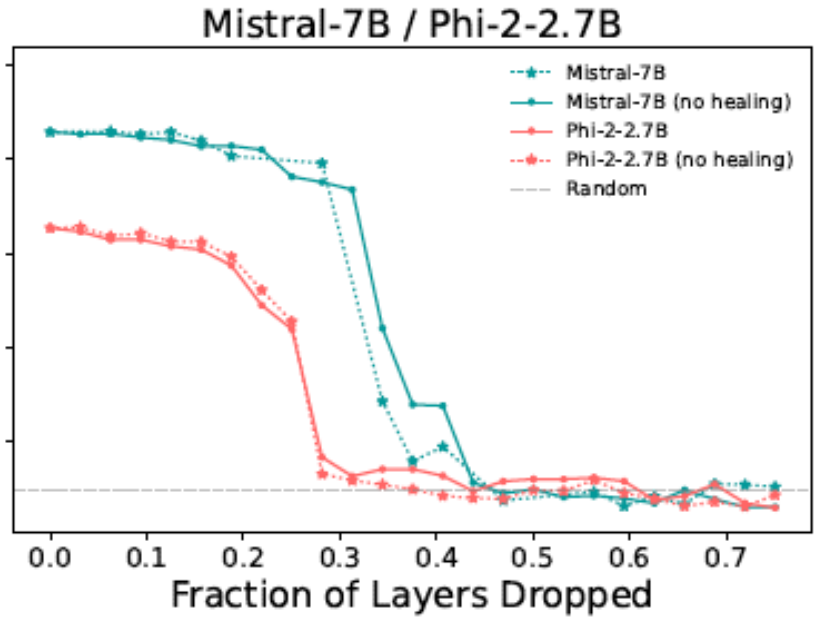


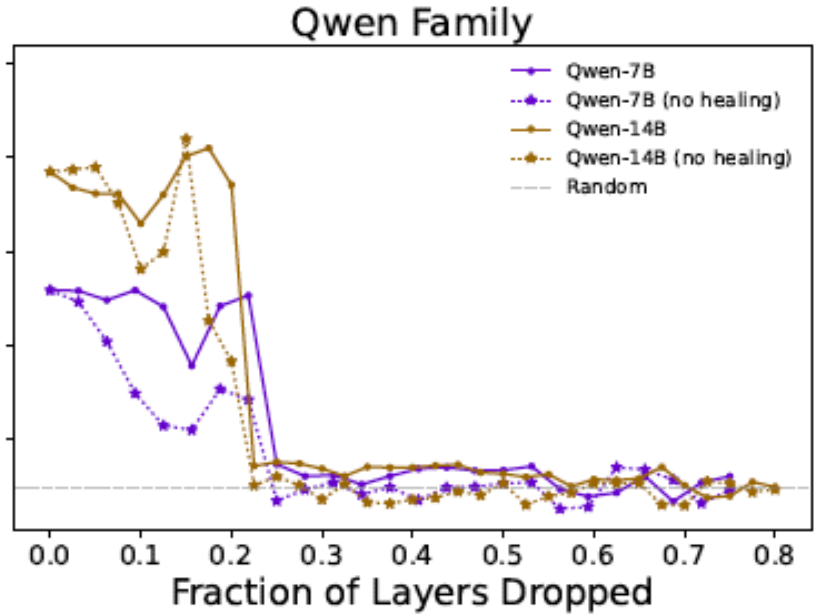


Pruning Large Language Models (LLMs)

- Gromov, Andrey, Kushal Tirumala, Hassan Shapourian, Paolo Glorioso, and Daniel A. Roberts. "The unreasonable ineffectiveness of the deeper layers." arXiv preprint arXiv:2403.17887 (2024).

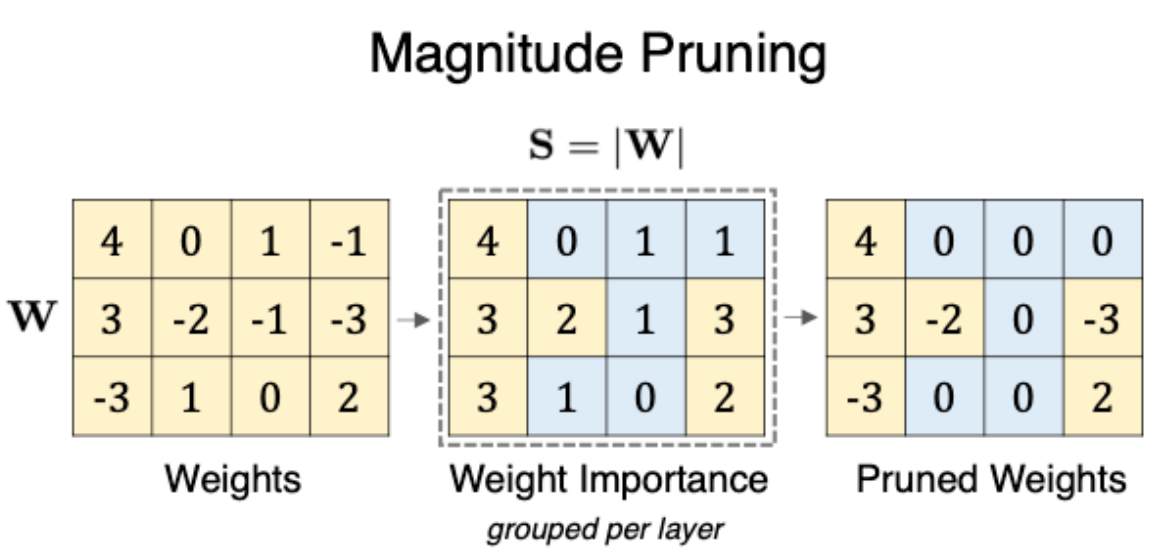


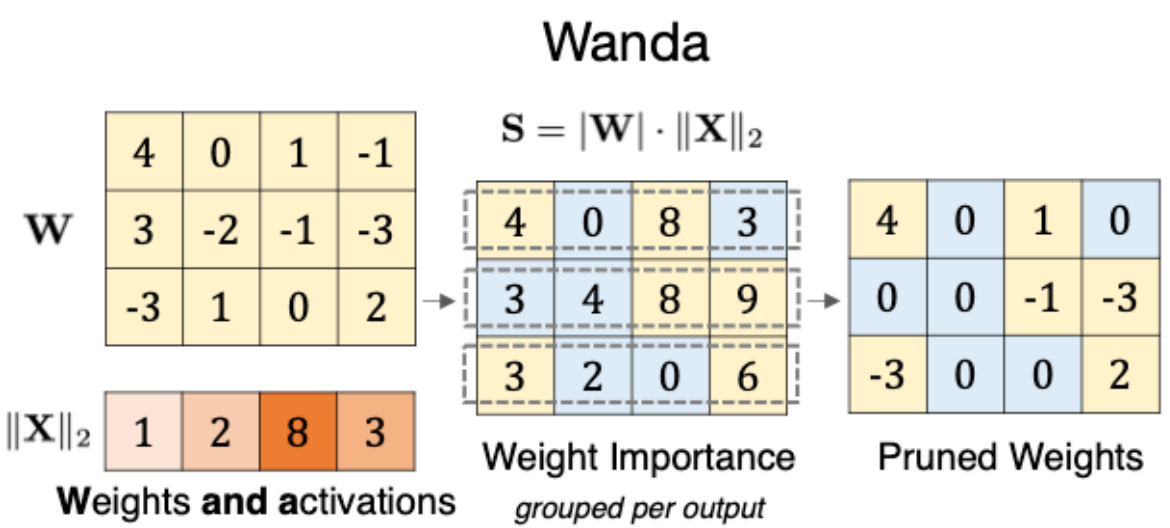


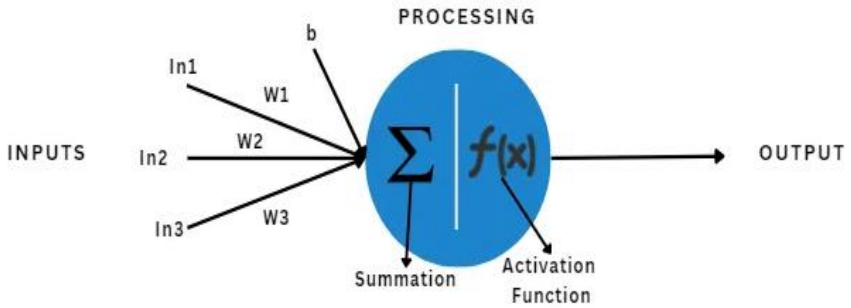


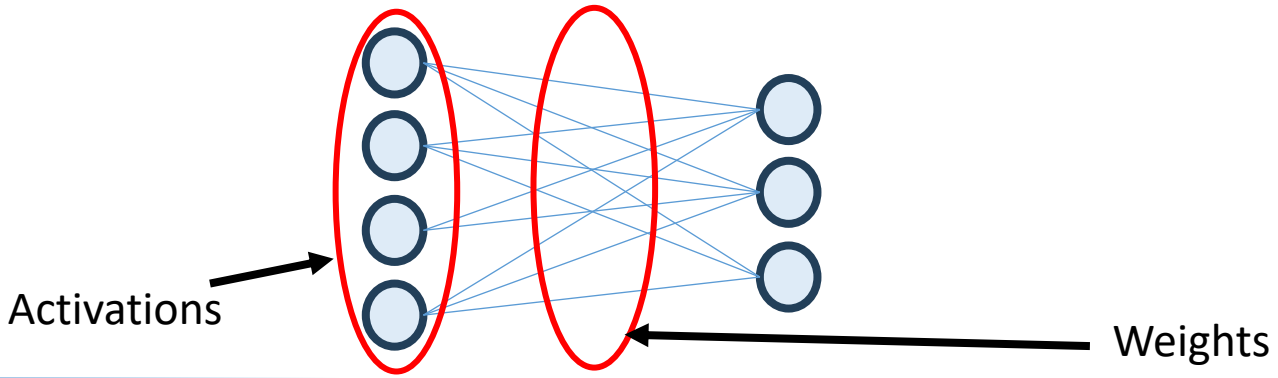
WANDA Pruning for Large Language Models (LLMs)

- Sun, Mingjie, Zhuang Liu, Anna Bair, and J. Zico Kolter. "A simple and effective pruning approach for large language models." arXiv preprint arXiv:2306.11695 (2023). – ICLR 2024









WANDA Pruning for Large Language Models (LLMs)

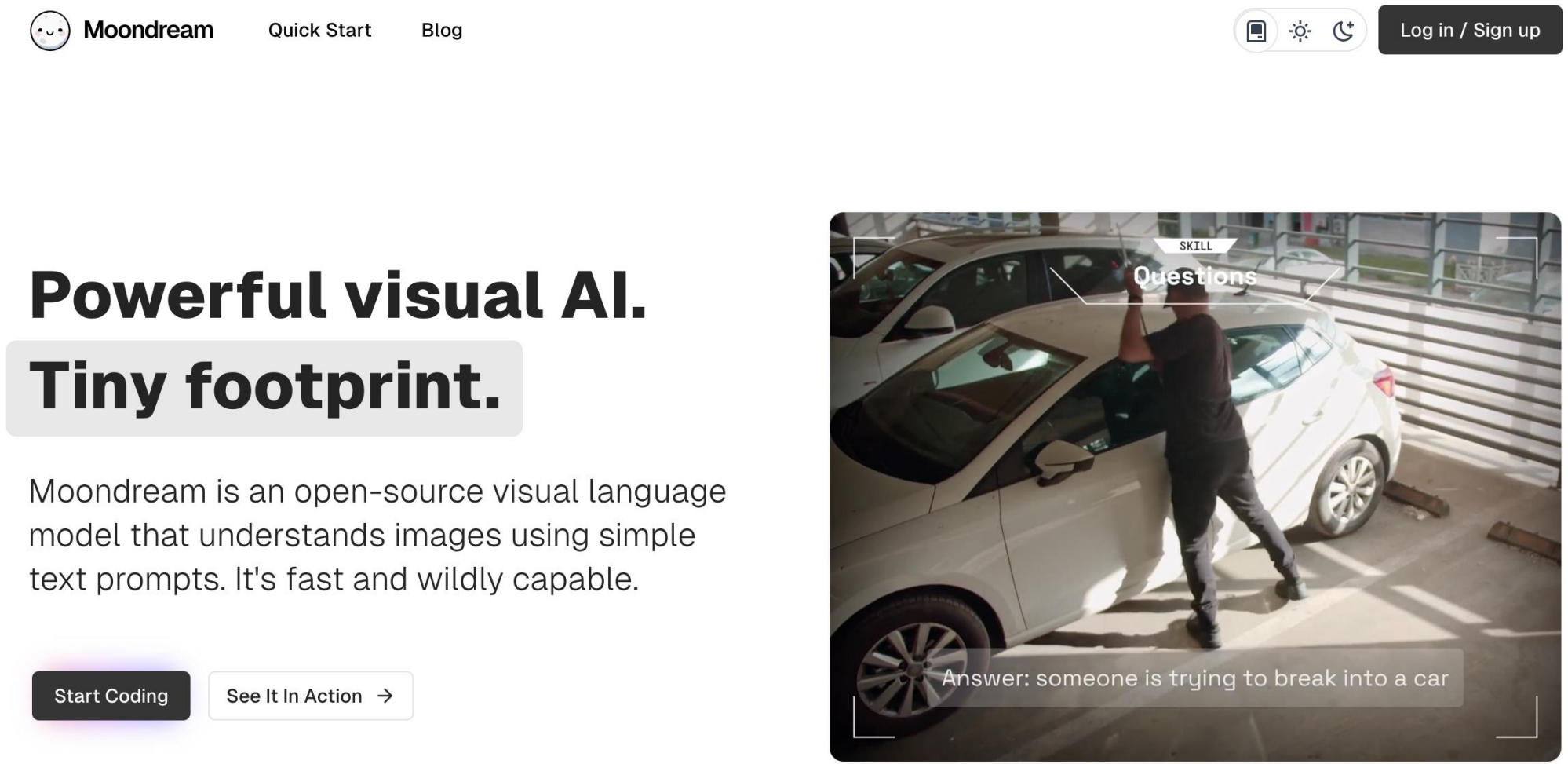
- Sun, Mingjie, Zhuang Liu, Anna Bair, and J. Zico Kolter. "A simple and effective pruning approach for large language models." arXiv preprint arXiv:2306.11695 (2023). – ICLR 2024

Mean zero-shot accuracies (%) of pruned LLaMA and LLaMA-2 models

			LLaMA				LLaMA-2		
Method	Weight Update	Sparsity	7B	13B	30B	65B	7B	13B	70B
Dense	-	0%	59.99	62.59	65.38	66.97	59.71	63.03	67.08
Magnitude	✗	50%	46.94	47.61	53.83	62.74	51.14	52.85	60.93
SparseGPT	✓	50%	**54.94**	58.61	63.09	66.30	**56.24**	60.72	**67.28**
Wanda	✗	50%	54.21	**59.33**	**63.60**	**66.67**	**56.24**	**60.83**	67.03
Magnitude	✗	4:8	46.03	50.53	53.53	62.17	50.64	52.81	60.28
SparseGPT	✓	4:8	**52.80**	55.99	60.79	64.87	**53.80**	**59.15**	65.84
Wanda	✗	4:8	52.76	**56.09**	**61.00**	**64.97**	52.49	58.75	**66.06**
Magnitude	✗	2:4	44.73	48.00	53.16	61.28	45.58	49.89	59.95
SparseGPT	✓	2:4	**50.60**	**53.22**	58.91	62.57	**50.94**	54.86	63.89
Wanda	✗	2:4	48.53	52.30	**59.21**	**62.84**	48.75	**55.03**	**64.14**

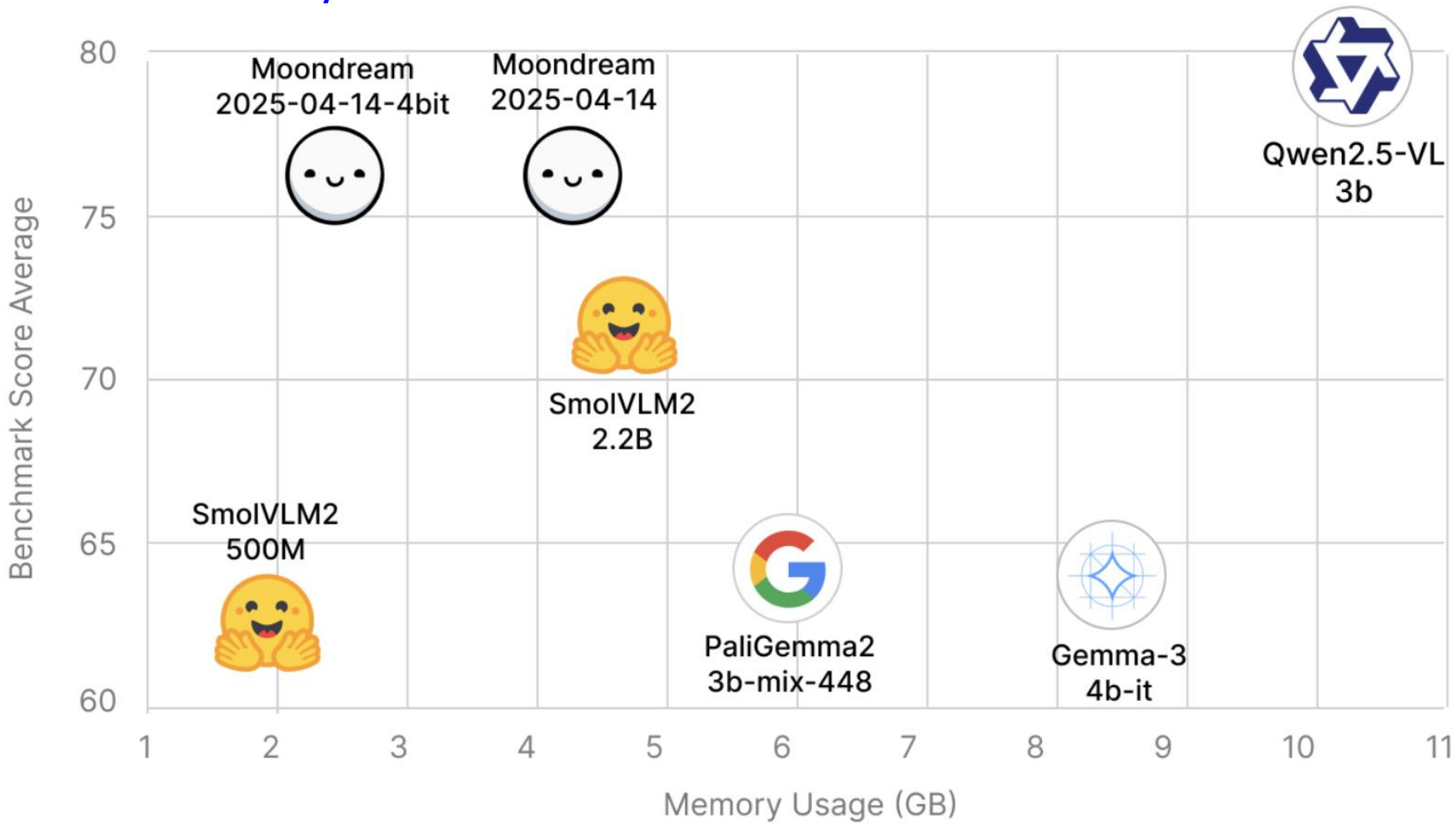
Moondream – Tiny Vision-Language Model

Link: https://moondream.ai/



Moondream – Tiny Vision-Language Model

Link: https://moondream.ai/



Pruning for Secure ML Models: Backdoor Attacks

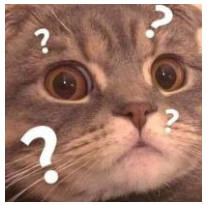
Poisoned training data

Training Phase

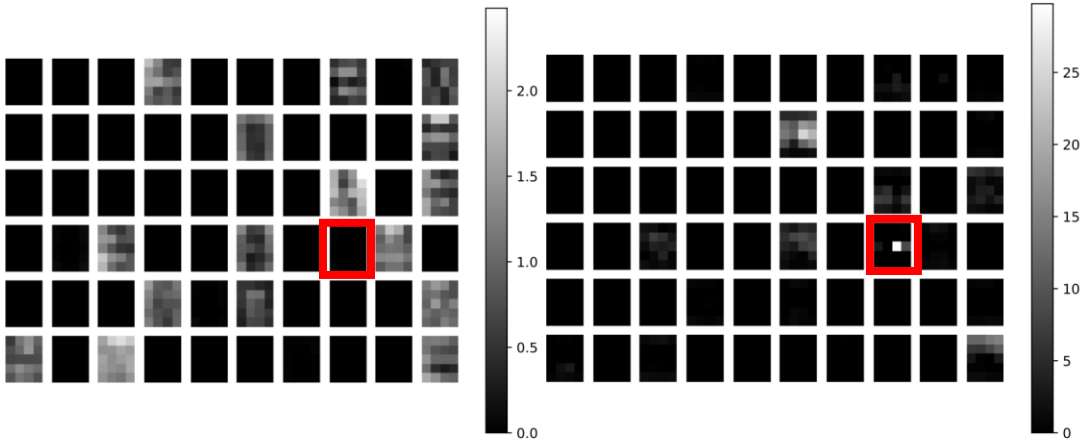
Features → Processed Data → Learning Algorithm → Model f → = Dog

Decision Phase

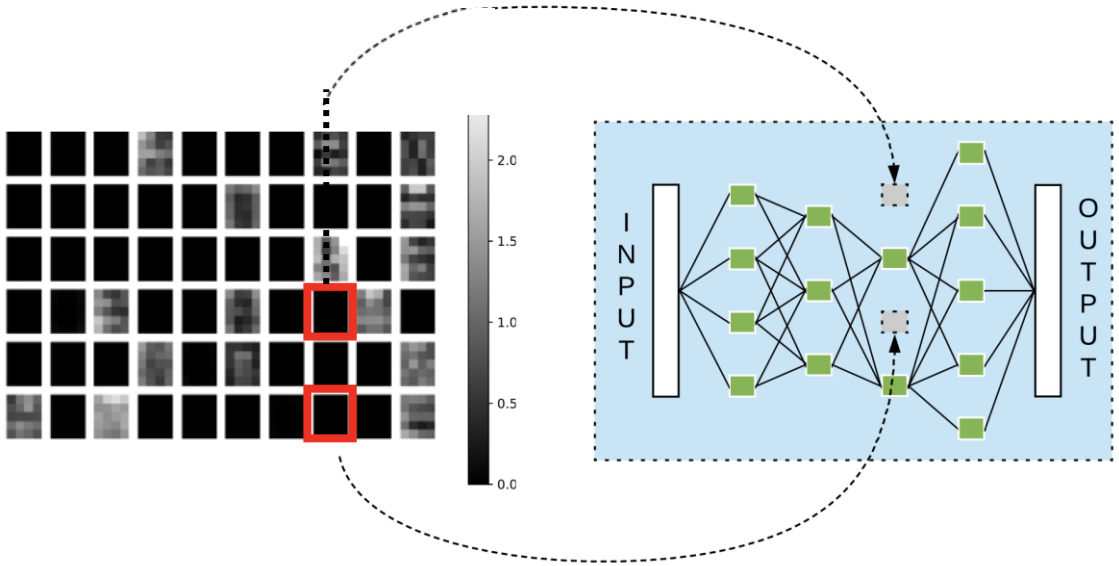
Features → Processed Data → Model f → ***ML Prediction*** Dog



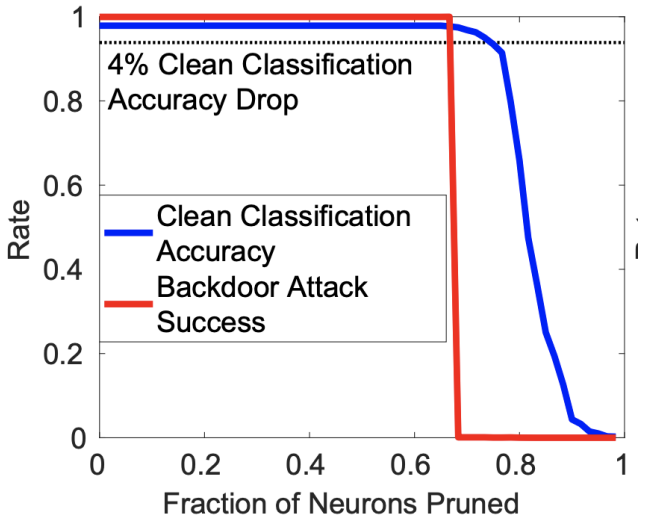
Pruning for Secure ML Models: Activation Pruning



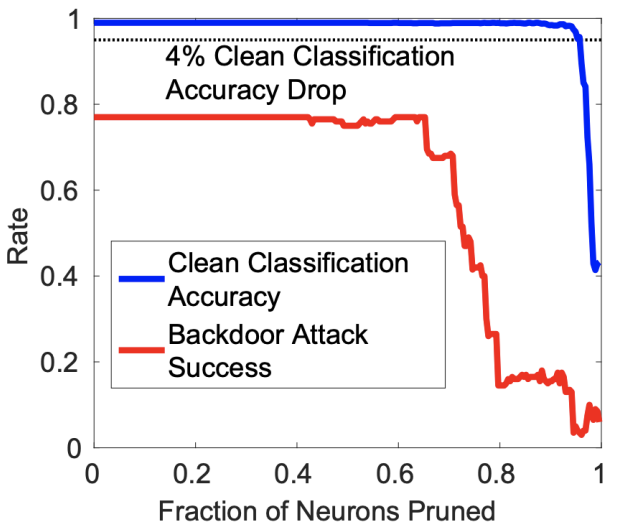
(a) Clean Activations (baseline attack) (b) Backdoor Activations (baseline attack)



- Liu, Kang, Brendan Dolan-Gavitt, and Siddharth Garg. "Fine-pruning: Defending against backdooring attacks on deep neural networks." In International symposium on research in attacks, intrusions, and defenses, pp. 273-294. Cham: Springer International Publishing, 2018.



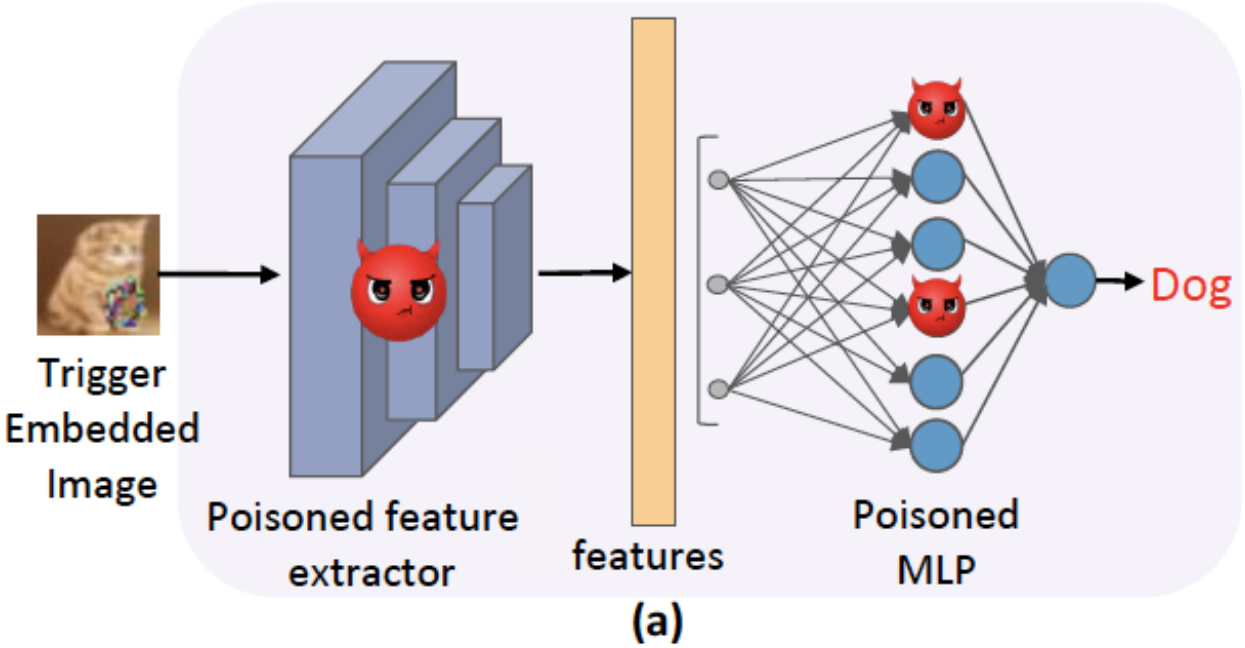
(a) Baseline Attack (Face)



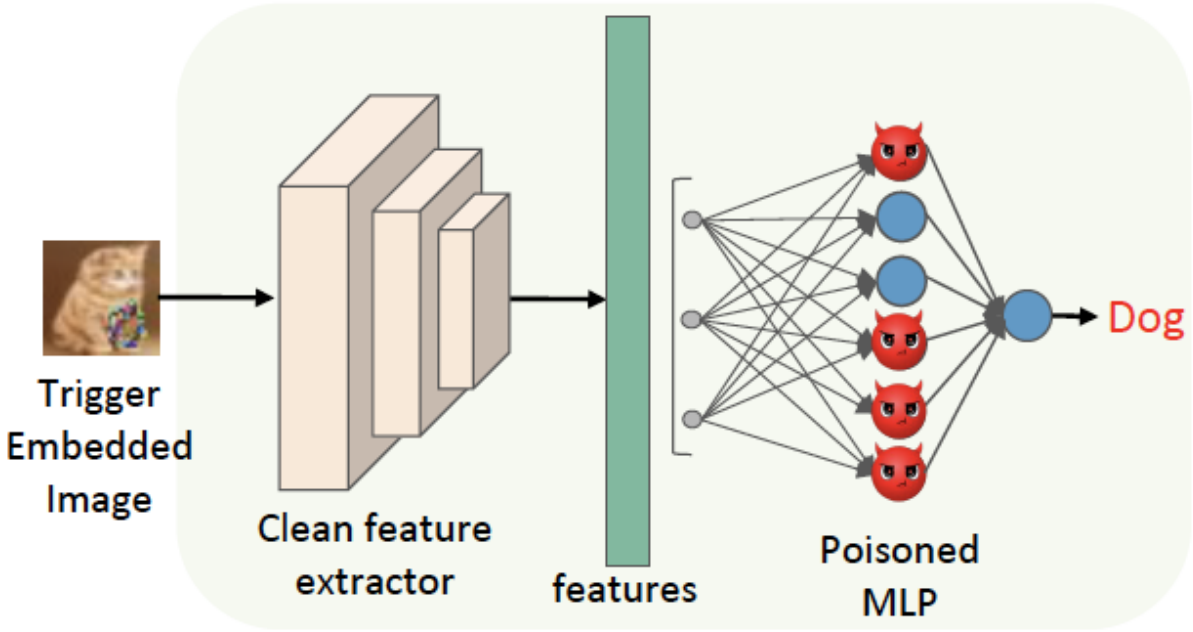
(c) Baseline Attack (Speech)

Pruning for Secure ML Models: Holder Pruning

- Wu, Yuchen, Yifei Zhao, Kyle Zheng, Dinuka Sahabandu, Bhaskar Ramasubramanian, and Radha Poovendran. "Holder Pruning: A Localized Pruning Strategy for Backdoor Removal in Deep Neural Networks."



Joint Training of Feature Extractor and Classifier

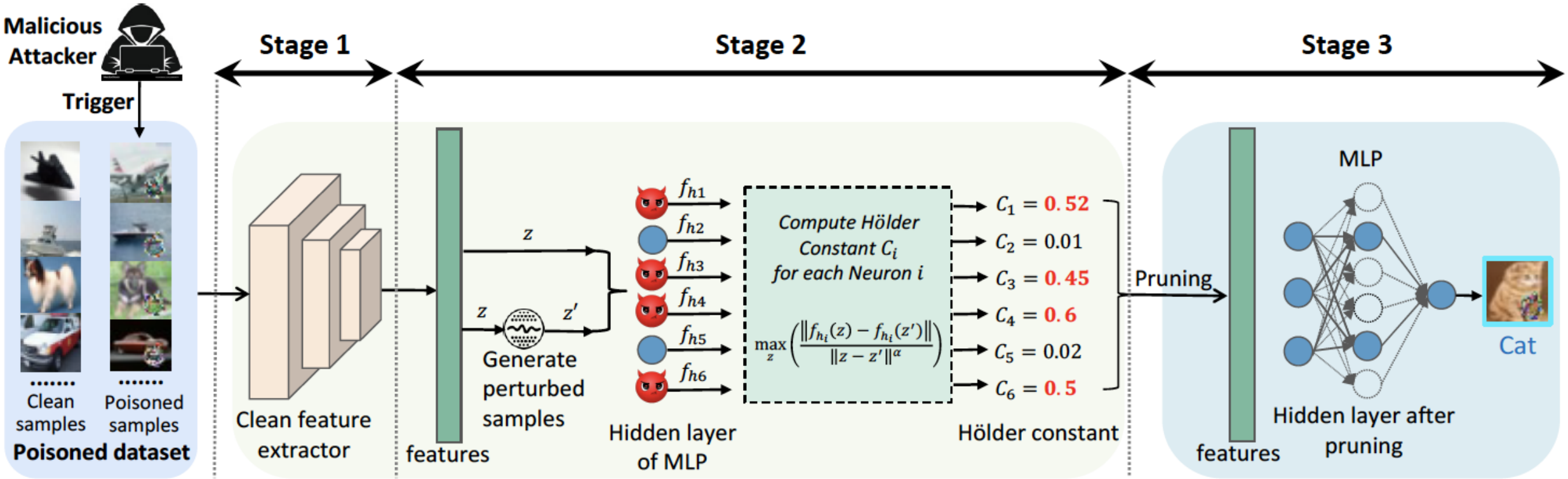


E.g., Pre-trained Vision Transformers such as CLIP, ViT, etc.

Only Classifier is Trained

Pruning for Secure ML Models: Holder Pruning

- Wu, Yuchen, Yifei Zhao, Kyle Zheng, Dinuka Sahabandu, Bhaskar Ramasubramanian, and Radha Poovendran. "Holder Pruning: A Localized Pruning Strategy for Backdoor Removal in Deep Neural Networks."



Pruning for Secure ML Models: Holder Pruning

- Wu, Yuchen, Yifei Zhao, Kyle Zheng, Dinuka Sahabandu, Bhaskar Ramasubramanian, and Radha Poovendran. "Holder Pruning: A Localized Pruning Strategy for Backdoor Removal in Deep Neural Networks."

Methods→	Benign			FP[38]			ANP[54]			CLP[59]			FMP[20]			**HP(ours)**		
Attacks↓	ACC↑	ASR↓	RA↑	ACC↑	ASR↓	RA↑	ACC↑	ASR↓	RA↑	ACC↑	ASR↓	RA↑	ACC↑	ASR↓	RA↑	ACC↑	ASR↓	RA↑
BadNets[13]	94.5	82.9	-	91.9	3.4	89.4	55.4	99.7	10.2	85.0	6.5	84.4	64.5	86.6	16.6	**93.7**	**0.0**	**93.3**
LC[46]	94.9	90.1	-	88.6	7.9	81.3	93.2	6.8	84.9	93.7	13.0	81.2	91.8	6.3	85.4	**93.8**	**0.0**	**87.8**
SIG[2]	95.0	43.6	-	90.9	15.1	79.6	92.8	41.6	39.7	85.9	50.7	29.5	91.5	14.6	69.6	**93.6**	**4.3**	**90.7**
LF[56]	94.3	87.5	-	93.8	12.5	77.8	83.8	89.4	3.4	91.5	14.3	80.1	73.7	84.8	0.1	**94.3**	**2.5**	**91.1**
WaNet[42]	93.1	20.4	-	92.5	6.9	87.4	21.5	98.2	1.1	86.1	**0.1**	87.2	69.1	10.0	60.4	**94.5**	**0.1**	**94.2**
Input-aware[41]	94.6	96.9	-	89.3	97.4	12.6	92.2	96.2	13.2	88.8	96.7	12.6	87.5	97.4	12.1	**93.1**	**4.5**	**88.4**
SSBA[34]	94.5	91.7	-	88.9	4.9	83.7	75.9	99.5	10.3	93.2	28.2	69.9	84.9	94.3	13.5	**93.8**	**1.1**	**90.3**
Trojan [39]	91.2	100.0	-	88.8	100.0	0.0	93.2	99.9	0.1	91.1	99.9	0.1	86.6	100.0	0.0	**94.4**	**4.7**	**86.9**
BppAttack[51]	94.9	93.9	-	91.1	73.5	21.4	**94.2**	66.1	32.2	82.5	4.1	60.2	91.7	91.4	7.8	92.9	**6.7**	**77.1**
Averages	94.1	78.6	-	90.6	35.7	59.2	78.0	77.4	21.6	88.6	34.8	56.1	82.3	56.1	34.7	**93.8↑**	**2.6↓**	**88.9↑**

CIFAR-10

Methods→	Benign			FP[38]			ANP[54]			CLP[59]			FMP[20]			**HP(ours)**		
Attacks↓	ACC↑	ASR↓	RA↑	ACC↑	ASR↓	RA↑	ACC↑	ASR↓	RA↑	ACC↑	ASR↓	RA↑	ACC↑	ASR↓	RA↑	ACC↑	ASR↓	RA↑
BadNets[13]	87.2	90.1	-	79.6	27.7	57.8	31.1	49.5	3.2	83.9	41.0	49.3	84.1	85.2	12.8	**85.4**	**3.2**	**80.6**
LC[46]	87.2	19.3	-	86.3	**0.0**	75.3	50.6	**0.0**	47.1	83.5	**0.0**	74.3	**86.6**	**0.0**	78.4	85.3	**0.0**	**78.9**
SIG[2]	89.3	44.4	-	**85.4**	28.2	32.2	57.3	26.3	23.7	83.7	37.7	25.9	83.7	37.7	25.9	82.9	**0.0**	**67.1**
LF[56]	88.5	95.0	-	81.3	51.6	38.4	60.0	27.2	42.5	83.3	47.7	44.3	84.2	97.5	2.0	**84.7**	**2.0**	**79.8**
WaNet[42]	84.9	8.6	-	74.9	0.3	65.3	68.8	**0.0**	61.0	82.0	0.3	68.6	83.3	6.2	70.7	**83.7**	0.4	**75.1**
Input-aware[41]	88.4	96.2	-	83.6	50.1	39.7	40.7	99.4	0.0	**83.7**	74.7	20.0	83.4	98.7	1.1	83.0	**6.6**	**68.1**
SSBA[34]	88.3	97.0	-	**85.2**	27.3	54.5	69.5	20.3	50.7	84.3	65.7	28.7	84.1	99.0	0.0	82.8	**2.9**	**75.1**
Trojan [39]	89.4	99.6	-	84.6	99.5	0.0	58.2	65.5	19.4	84.8	98.6	1.2	**85.5**	99.8	0.0	85.2	**4.7**	**77.9**
BppAttack[51]	88.4	91.3	-	**85.2**	9.6	43.2	69.4	22.4	37.9	83.5	47.0	32.4	83.5	47.1	32.4	85.0	**2.4**	**77.7**
Averages	87.9	71.2	-	82.9	32.7	45.1	56.1	34.5	31.7	83.6	45.9	38.3	84.1	63.5	24.8	**84.2↑**	**2.4↓**	**75.6↑**

GTSRB

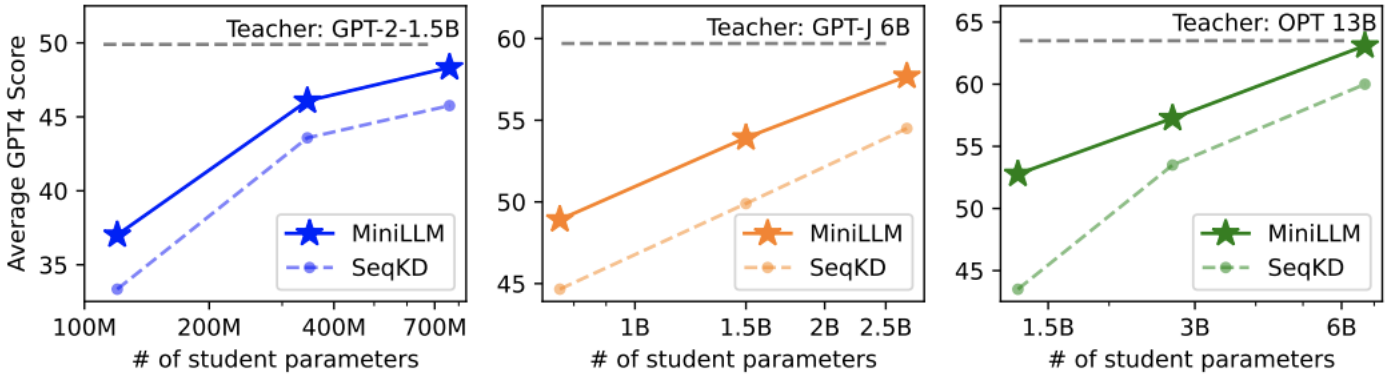
Knowledge Distillation for LLMs

MiniLLM: Knowledge Distillation of Large Language Models

Yuxian Gu[1,2]*; **Li Dong**[2], **Furu Wei**[2], **Minlie Huang**[1†]
[1]The CoAI Group, Tsinghua University
[2]Microsoft Research
guyx21@mails.tsinghua.edu.cn {lidong1,fuwei}@microsoft.com
aihuang@tsinghua.edu.cn

Abstract

Knowledge Distillation (KD) is a promising technique for reducing the high computational demand of large language models (LLMs). However, previous KD methods are primarily applied to white-box classification models or training small models to imitate black-box model APIs like ChatGPT. How to effectively distill the knowledge of white-box LLMs into small models is still under-explored, which becomes more important with the prosperity of open-source LLMs. In this work, we propose a KD approach that distills LLMs into smaller language models. We first replace the *forward* Kullback-Leibler divergence (KLD) objective in the standard KD approaches with *reverse* KLD, which is more suitable for KD on generative language models, to prevent the student model from overestimating the low-probability regions of the teacher distribution. Then, we derive an effective optimization approach to learn this objective. The student models are named **MiniLLM**. Extensive experiments in the instruction-following setting show that MiniLLM generates more precise responses with higher overall quality, lower exposure bias, better calibration, and higher long-text generation performance than the baselines. Our method is scalable for different model families with 120M to 13B parameters. Our code, data, and model checkpoints can be found in https://github.com/microsoft/LMOps/tree/main/minillm.



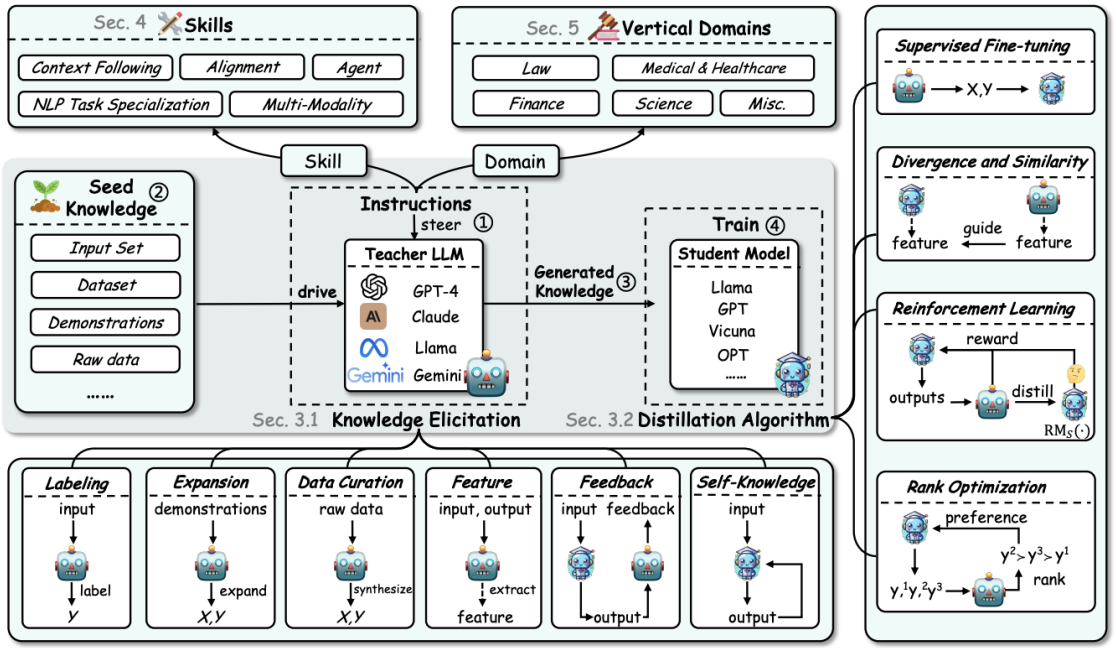
Knowledge Distillation for LLMs

A Survey on Knowledge Distillation of Large Language Models

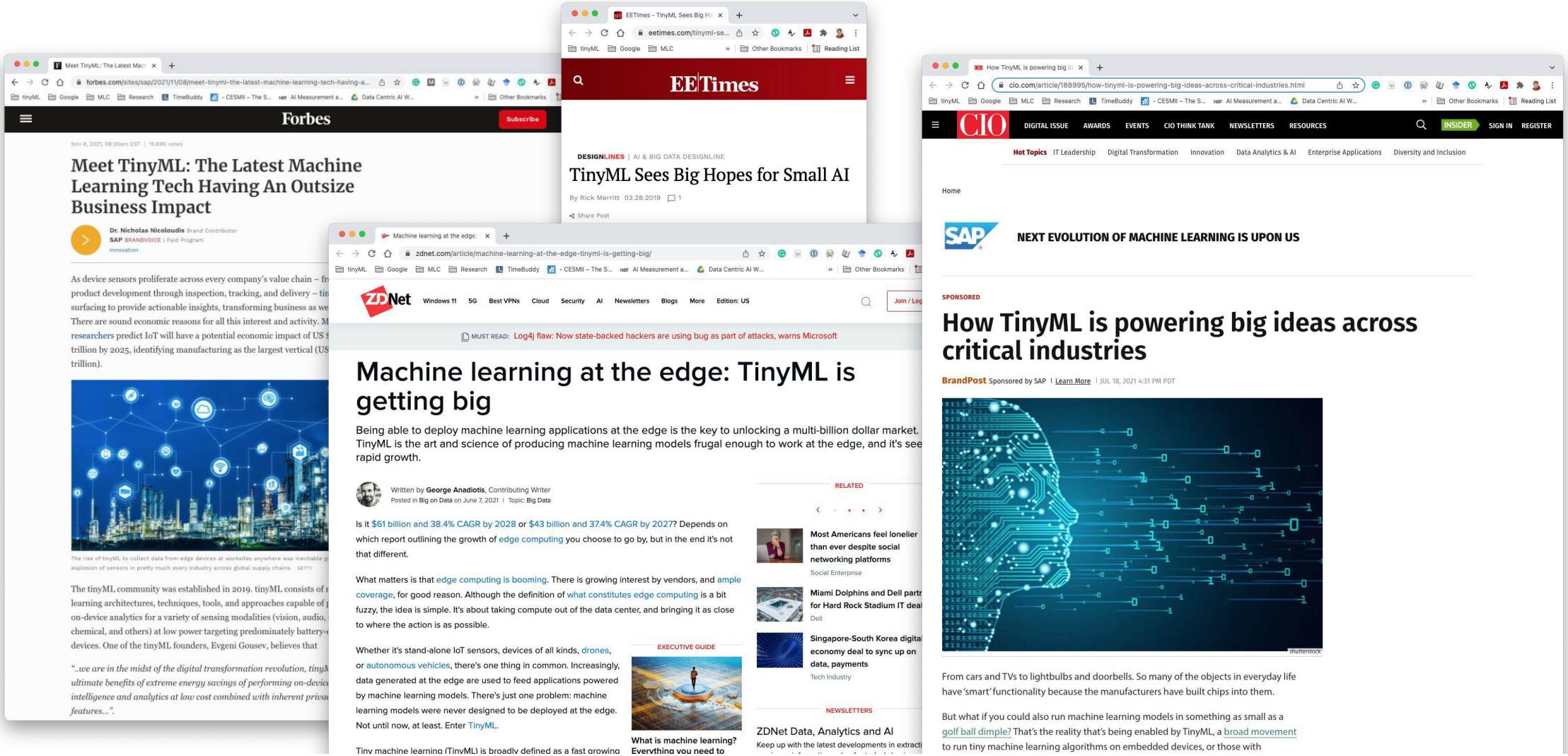
Xiaohan Xu[1], Ming Li[2], Chongyang Tao[3], Tao Shen[4], Reynold Cheng[1], Jinyang Li[1], Can Xu[5], Dacheng Tao[6], Tianyi Zhou[2]

[1]The University of Hong Kong [2]University of Maryland [3]Microsoft
[4]University of Technology Sydney [5]Peking University [6]The University of Sydney
{shawnxxh,chongyangtao,hishentao}@gmail.com {minglii,tianyi}@umd.edu
ckcheng@cs.hku.hk jl0725@connect.hku.hk

Abstract—In the era of Large Language Models (LLMs), Knowledge Distillation (KD) emerges as a pivotal methodology for transferring advanced capabilities from leading proprietary LLMs, such as GPT-4, to their open-source counterparts like LLaMA and Mistral. Additionally, as open-source LLMs flourish, KD plays a crucial role in both compressing these models, and facilitating their self-improvement by employing themselves as teachers. This paper presents a comprehensive survey of KD's role within the realm of LLM, highlighting its critical function in imparting advanced knowledge to smaller models and its utility in model compression and self-improvement. Our survey is meticulously structured around three foundational pillars: *algorithm*, *skill*, and *verticalization* – providing a comprehensive examination of KD mechanisms, the enhancement of specific cognitive abilities, and their practical implications across diverse fields. Crucially, the survey navigates the interaction between data augmentation (DA) and KD, illustrating how DA emerges as a powerful paradigm within the KD framework to bolster LLMs' performance. By leveraging DA to generate context-rich, skill-specific training data, KD transcends traditional boundaries, enabling open-source models to approximate the contextual adeptness, ethical alignment, and deep semantic insights characteristic of their proprietary counterparts. This work aims to provide an insightful guide for researchers and practitioners, offering a detailed overview of current methodologies in knowledge distillation and proposing future research directions. By bridging the gap between proprietary and open-source LLMs, this survey underscores the potential for more accessible, efficient, and powerful AI solutions. Most importantly, we firmly advocate for compliance with the legal terms that regulate the use of LLMs, ensuring ethical and lawful application of KD of LLMs. An associated Github repository is available at https://github.com/Tebmer/Awesome-Knowledge-Distillation-of-LLMs.

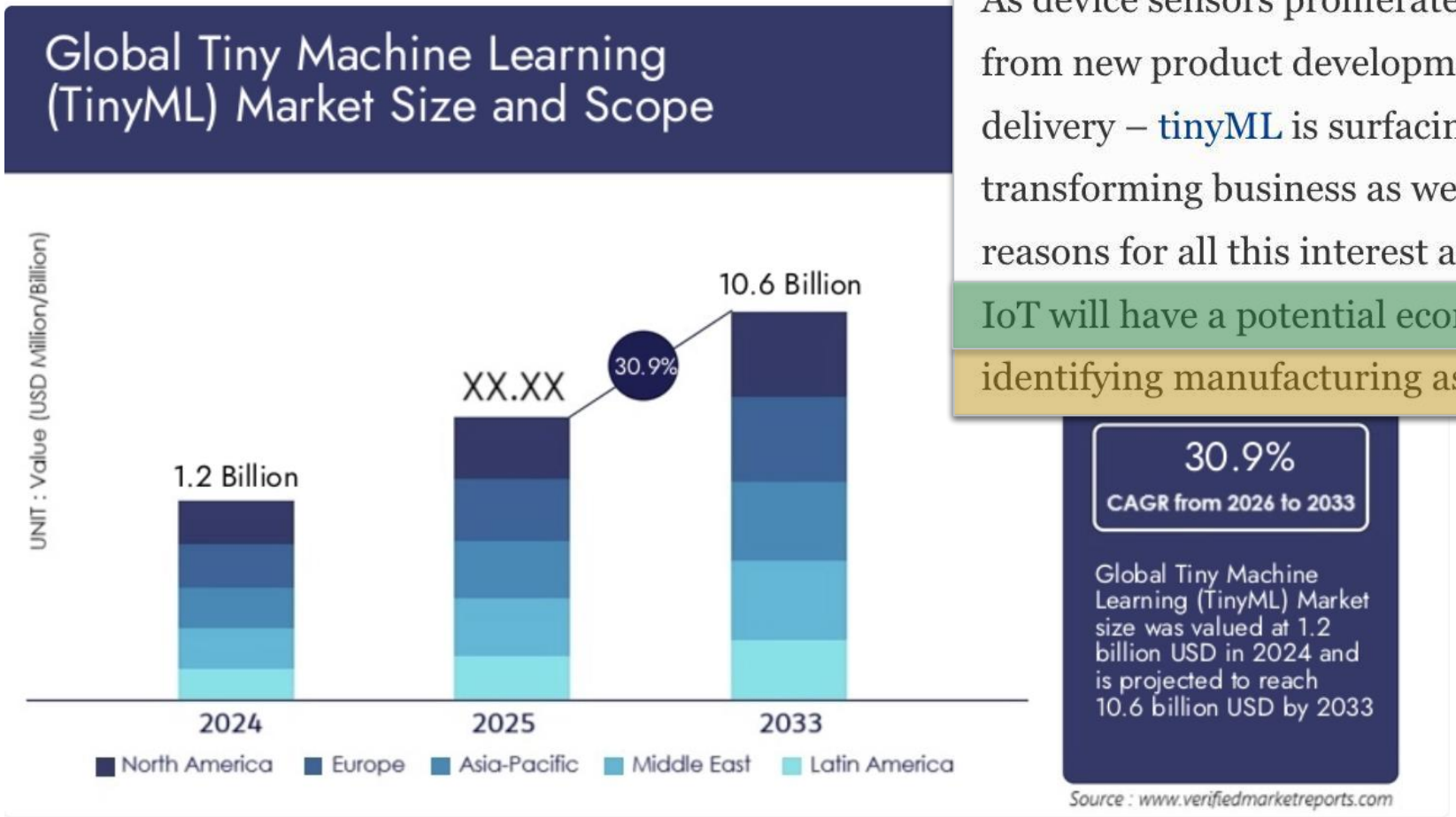


The future of the TinyML



Source: TinyML: The Future of Machine Learning is Tiny and Bright

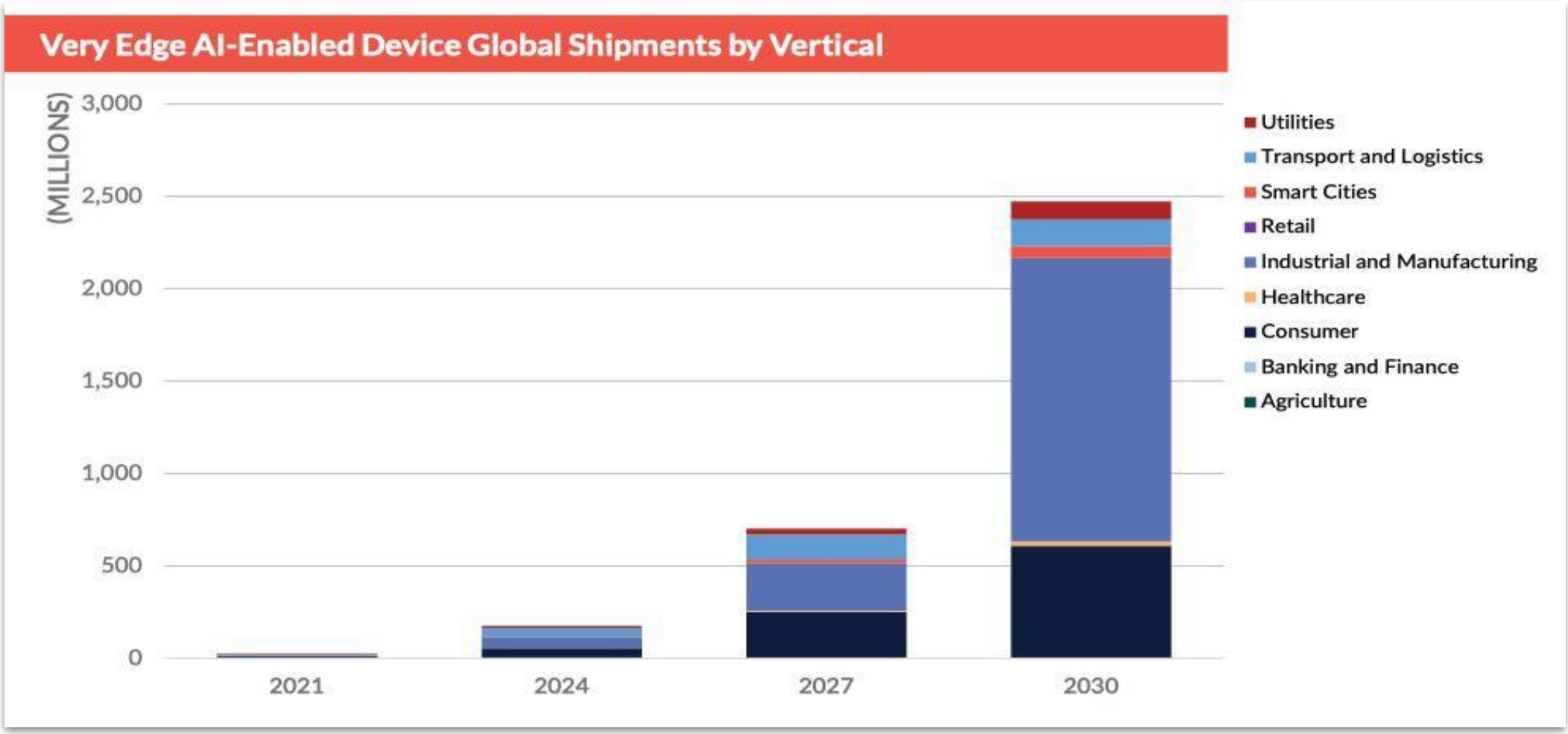
The future of the TinyML



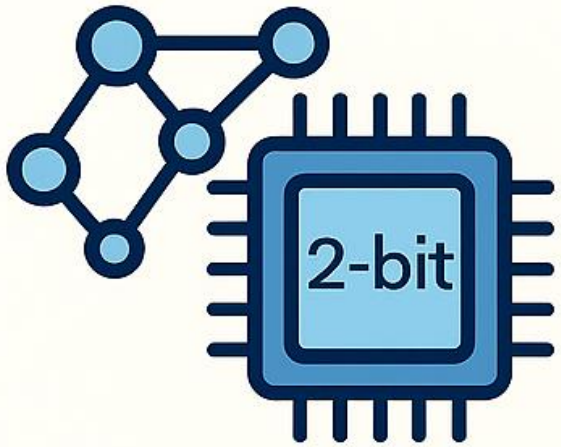
As device sensors proliferate across every company's value chain – from new product development through inspection, tracking, and delivery – tinyML is surfacing to provide actionable insights, transforming business as we know it. There are sound economic reasons for all this interest and activity. McKinsey researchers predict IoT will have a potential economic impact of US $4-11 trillion by 2025, identifying manufacturing as the largest vertical (US $1.2-3.7 trillion).

Source: Forbes

The future of the TinyML

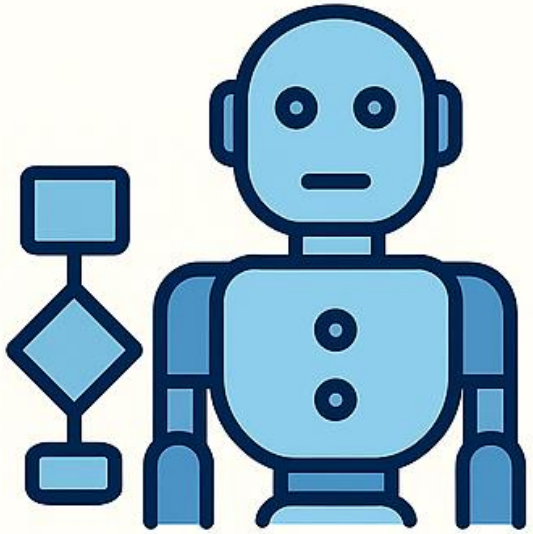


The Future of the TinyML



Ultra-Low-Bit LLMs on FPGAs and Microcontrollers

AI Alignment



Agentic AI

Thank You!

The Future of ML is Tiny and Bright

A Quote from Prof. Vijay Janapa Reddi (Harvard University) and Pete Warden (Google)