CS249r: Model Optimizations

Oct. 2, 2023

Course Logistics

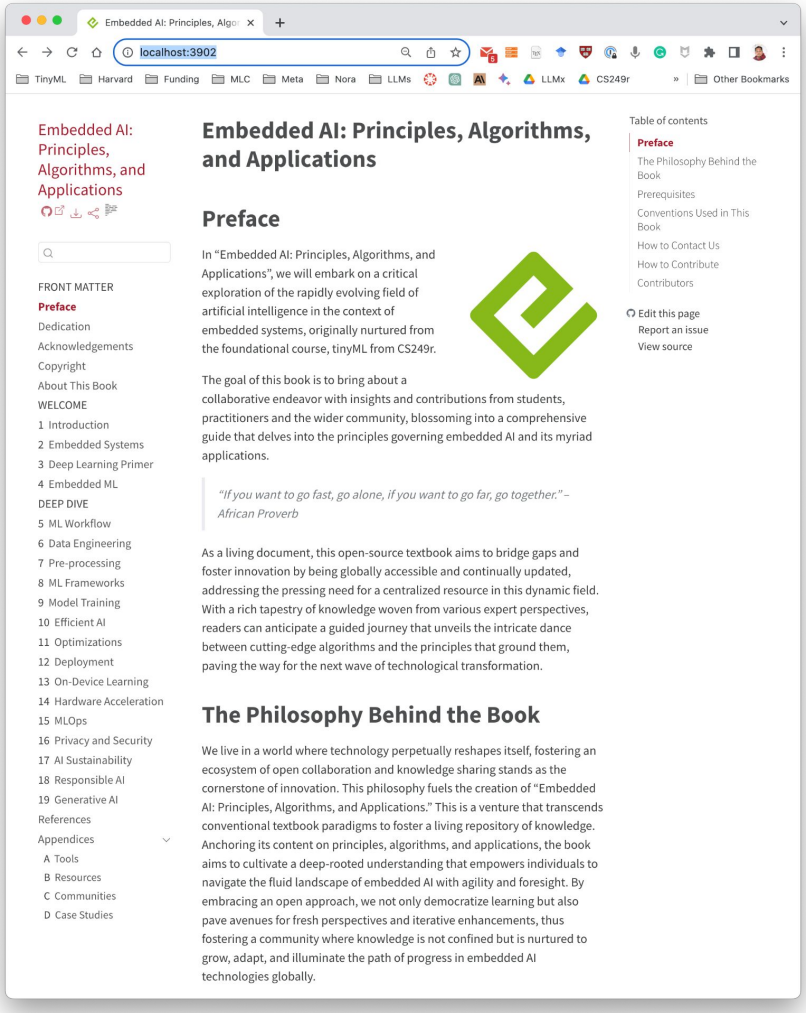
Assignment #1

- Due Dates:
  - **Part 1:** Oct 2nd (TODAY)
  - **Part 2:** Oct 10th

- Questions/Concerns?

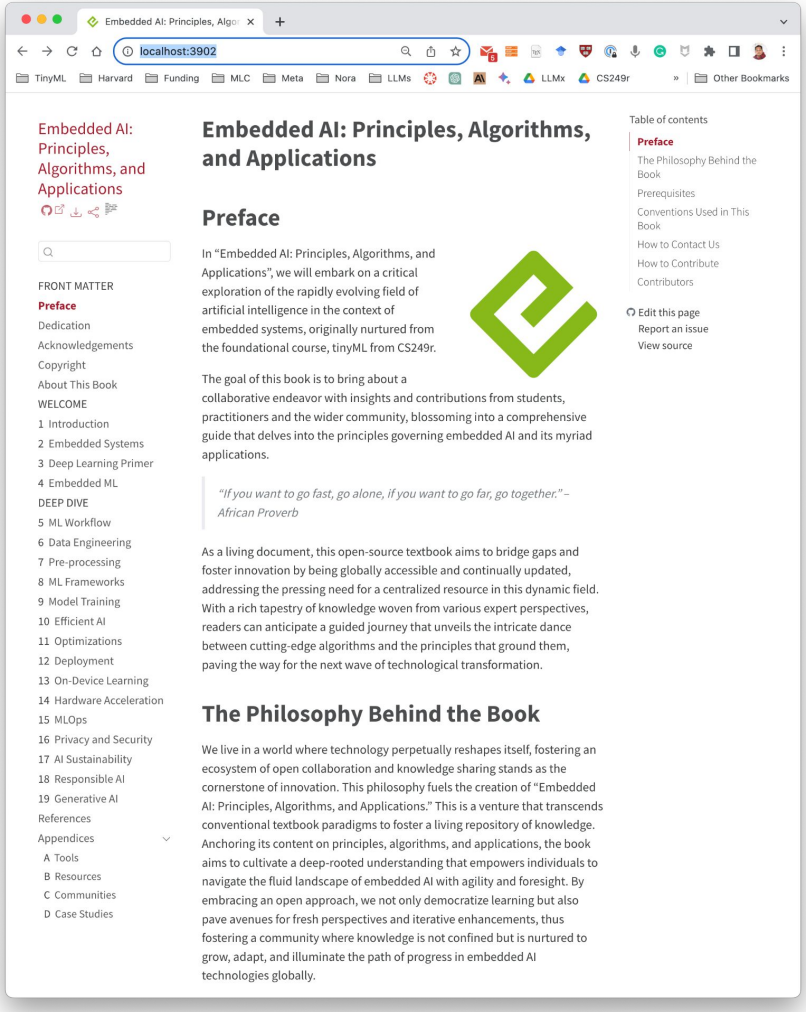
Scribing

- Week of topic
  - Meet with Matthew & VJ **(schedule via Slack)**
  - Create a google doc
  - Share with staff
  - Iterate on rough outline with VJ/Matt
  - After 👍from staff, put changes in a **single forked repo**
  - Create **one PR** with the entire chapter
  - Classmates will peer review using the PR



Peer Reviewing

- This week **(10/2 - 10/8)**
  - Navigate to the PR for **Data Engineering**: [Will Update Link to PR]
  - Render the chapter using Quarto
  - Read over the chapter and make comments directly on the GitHub PR
  - See scribing instructions



Peer Reviewing - Quick Demo

1. Scribe: Jessica
2. Peer Reviewer: Ike
3. Scribe makes PR from their group's forked repo
4. Staff will provide classmates with link to the group's repo, and a link to the PR
5. Use the GitHub interface to comment directly on the PR
    a. To render the book locally, pull the group's forked repo and run "quarto publish"

Project Sign-Ups due by **11:59pm today!**

- Join the #projects channel and post your interests
- Once you find a group, sign up on the spreadsheet

Lecture

Model Optimization Use Cases

- **Reducing latency and cost** for inference for both cloud and edge devices (e.g. mobile, IoT)

Model Optimization Use Cases

- **Reducing latency and cost** for inference for both cloud and edge devices (e.g. mobile, IoT)
- **Deploying models on edge devices** with restrictions on processing, memory and/or power-consumption

Model Optimization Use Cases

- **Reducing latency and cost** for inference for both cloud and edge devices (e.g. mobile, IoT)
- **Deploying models on edge devices** with restrictions on processing, memory and/or power-consumption
- **Reducing payload size** for over-the-air model updates

Model Optimization Use Cases

- **Reducing latency and cost** for inference for both cloud and edge devices (e.g. mobile, IoT)
- **Deploying models on edge devices** with restrictions on processing, memory and/or power-consumption
- **Reducing payload size** for over-the-air model updates
- Enabling execution on hardware restricted-to or optimized-for fixed-point operations
- Optimizing models for special purpose hardware accelerators.

Model Pruning

The MLOps **Personas**

ML Engineer

ML Researcher

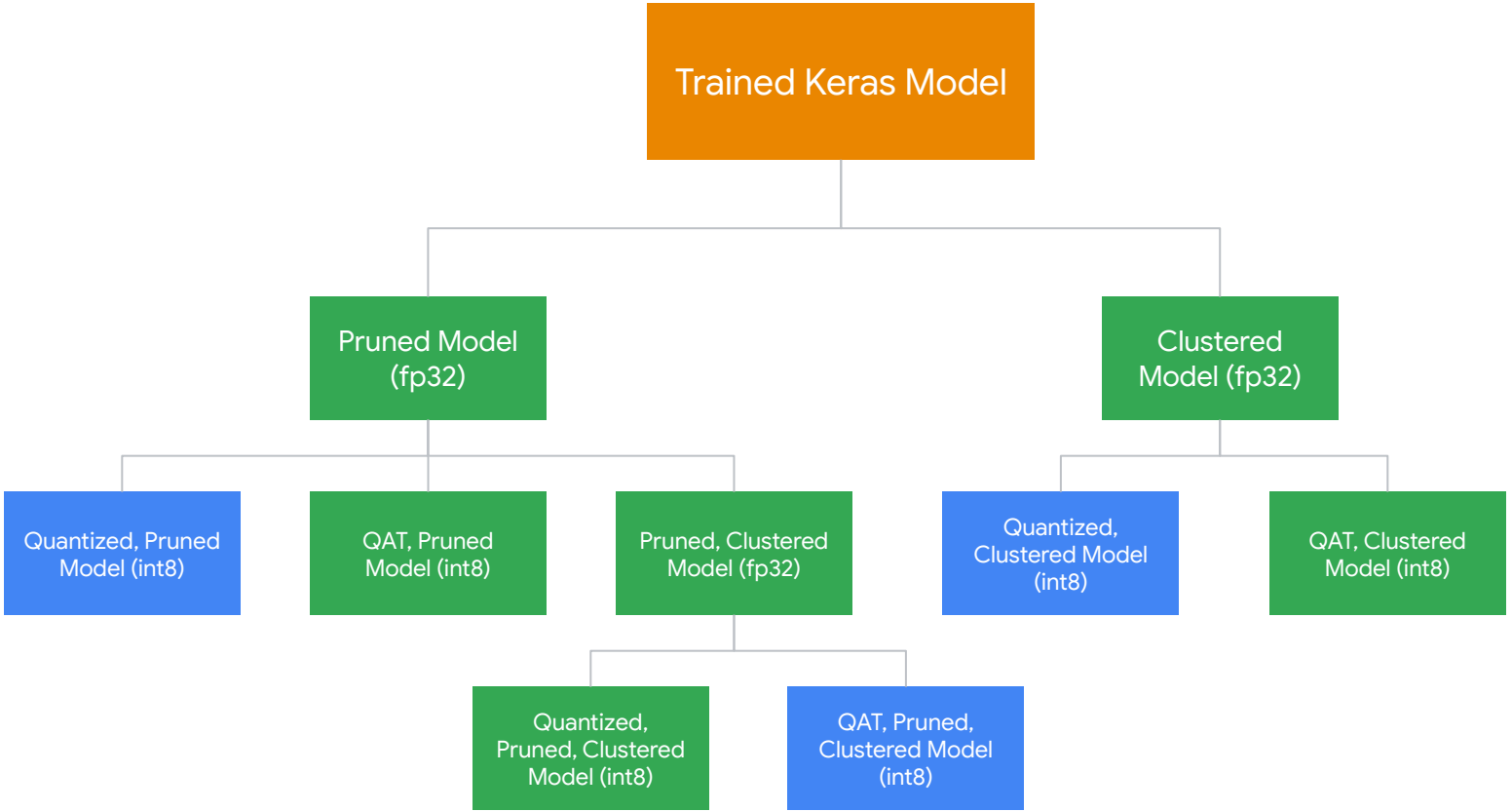
Data Scientist

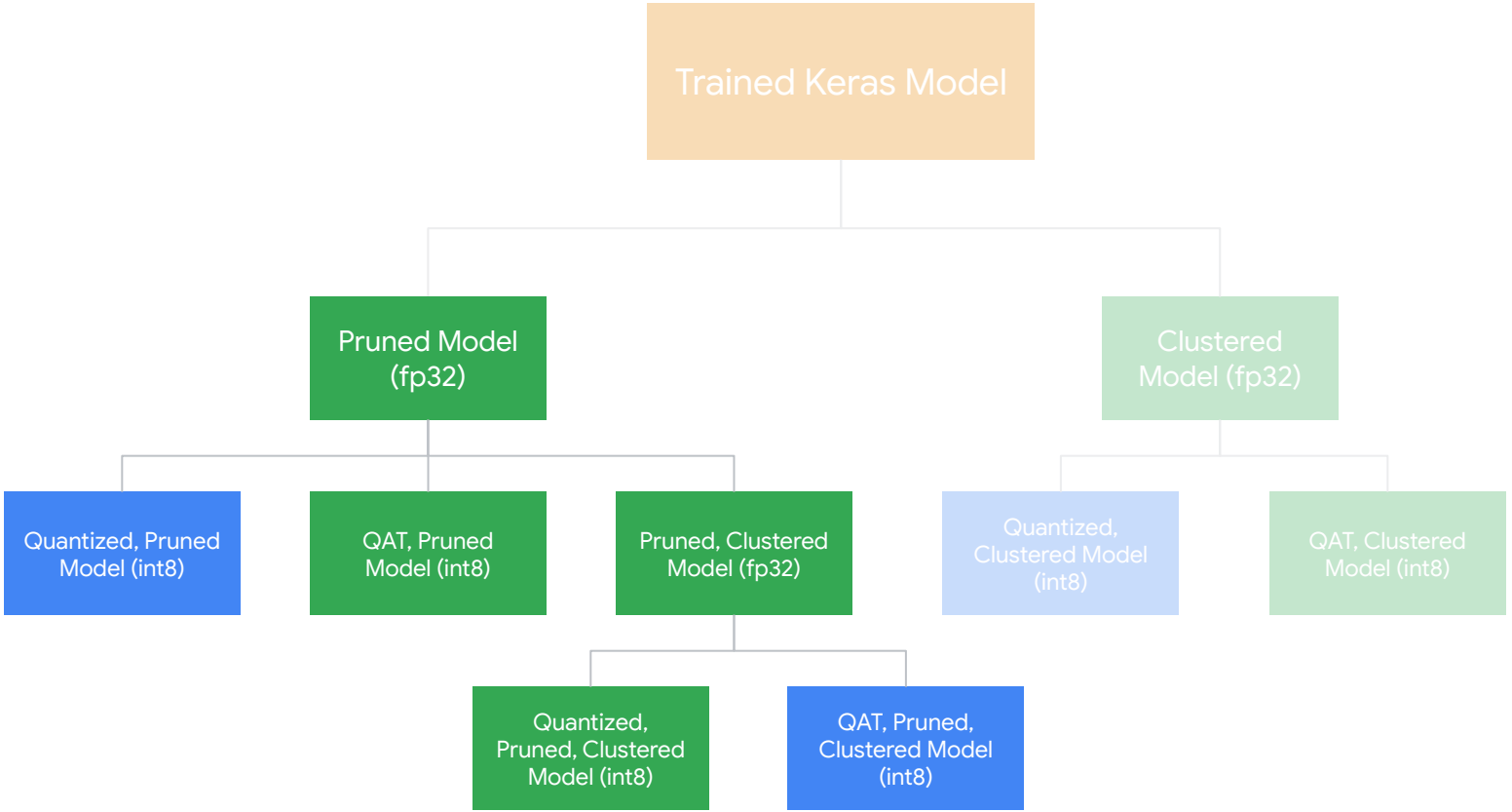
Data Engineer

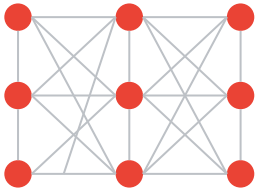
Software Engineer

DevOps

Business Analyst

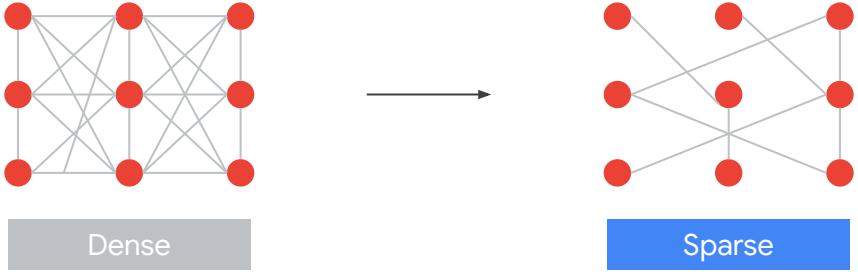

Trained Keras Model
Pruned Model (fp32)
Clustered Model (fp32)
Quantized, Pruned Model (int8)
QAT, Pruned Model (int8)
Pruned, Clustered Model (fp32)
Quantized, Clustered Model (int8)
QAT, Clustered Model (int8)
Quantized, Pruned, Clustered Model (int8)
QAT, Pruned, Clustered Model (int8)

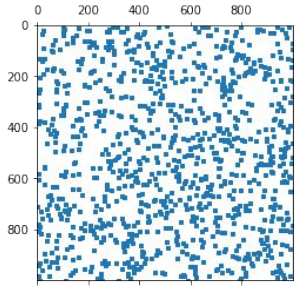

Trained Keras Model
Pruned Model (fp32)
Clustered Model (fp32)
Quantized, Pruned Model (int8)
QAT, Pruned Model (int8)
Pruned, Clustered Model (fp32)
Quantized, Clustered Model (int8)
QAT, Clustered Model (int8)
Quantized, Pruned, Clustered Model (int8)
QAT, Pruned, Clustered Model (int8)



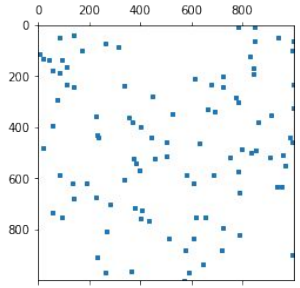
Dense

Sparse

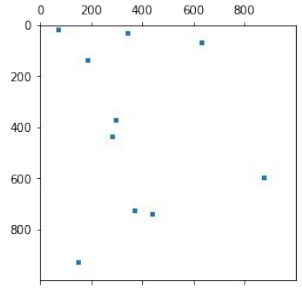




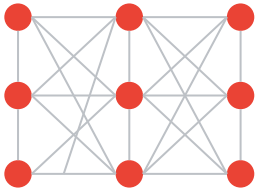
Dense



Sparse-ish

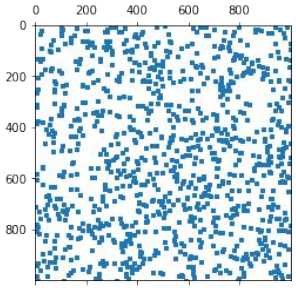


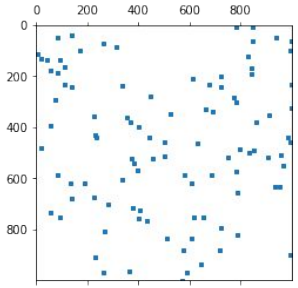
Sparse

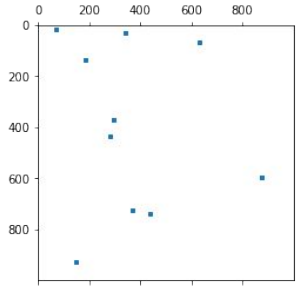


Dense

Sparse





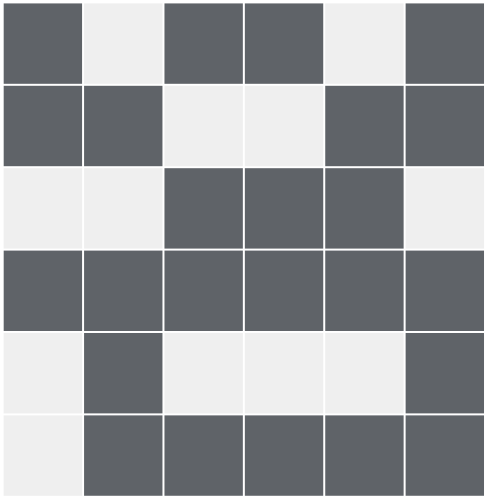


3	4	8
7	6	3
5	5	1

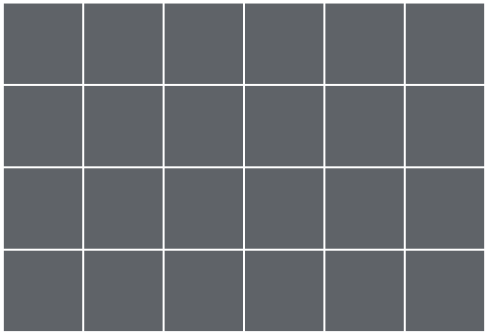
0	4	8
7	6	3
0	5	0

0	4	0
7	0	3
0	5	0

Sparse



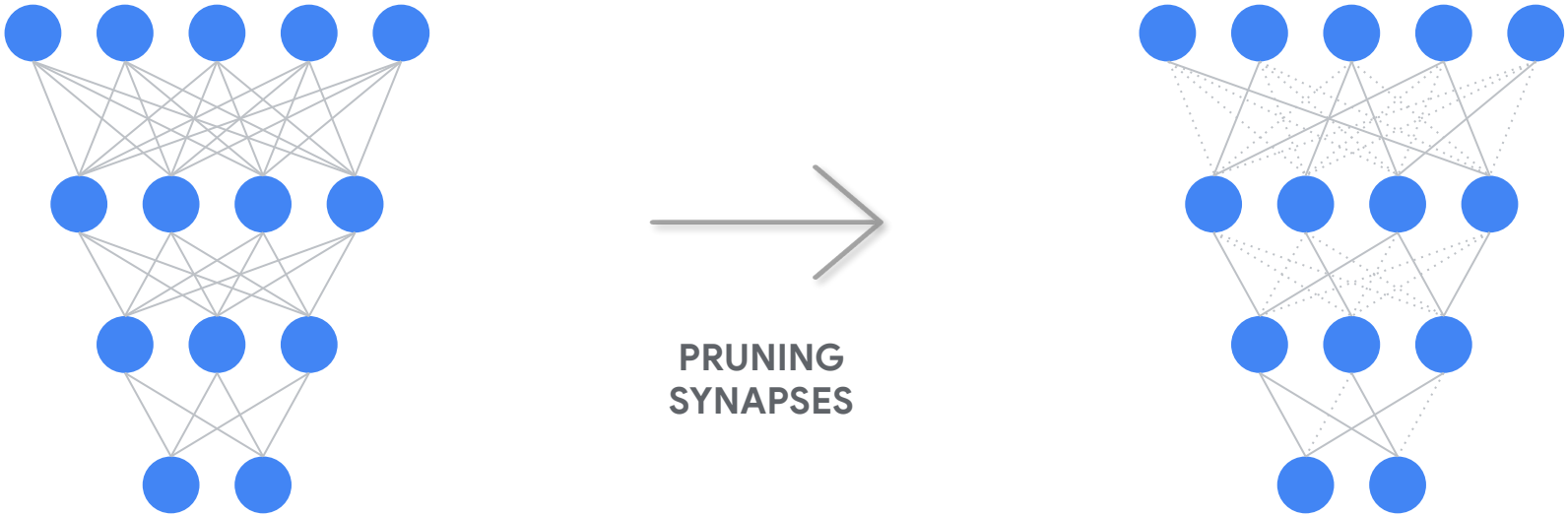
Packed

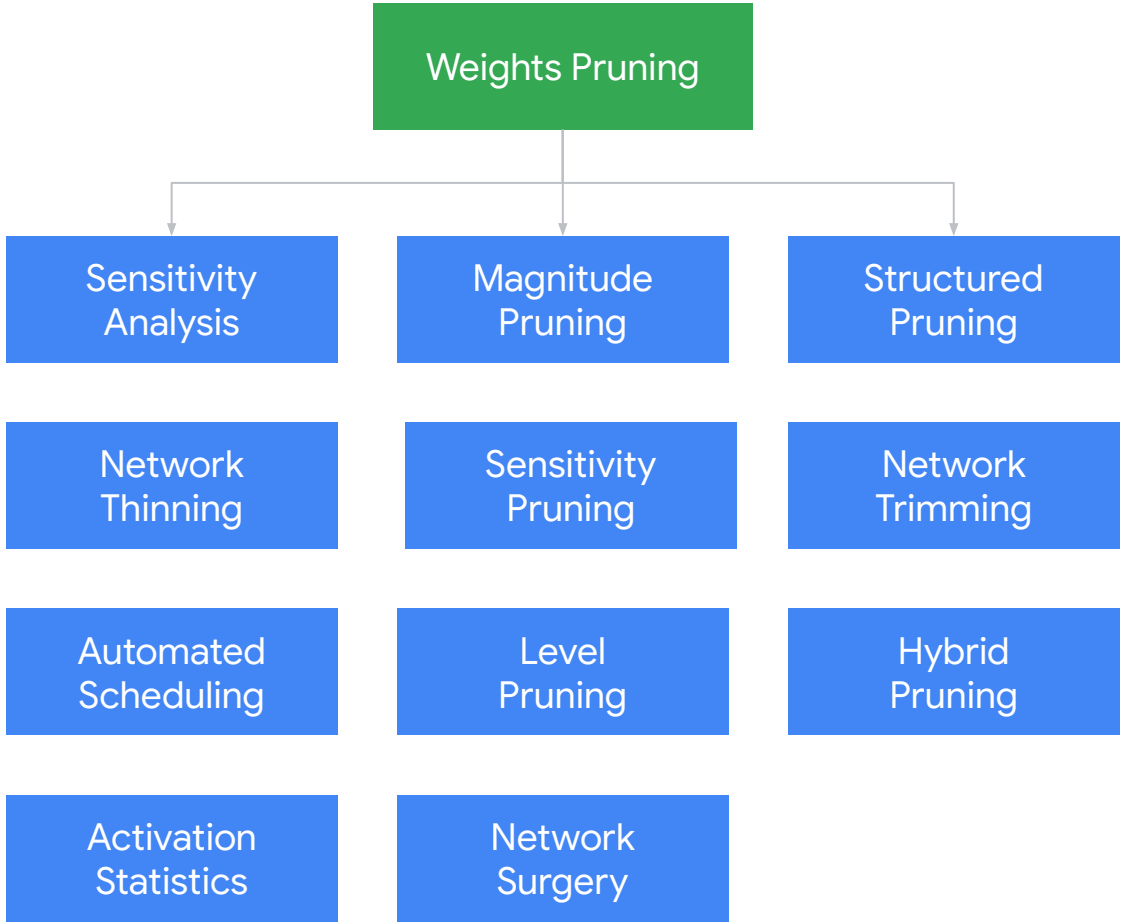


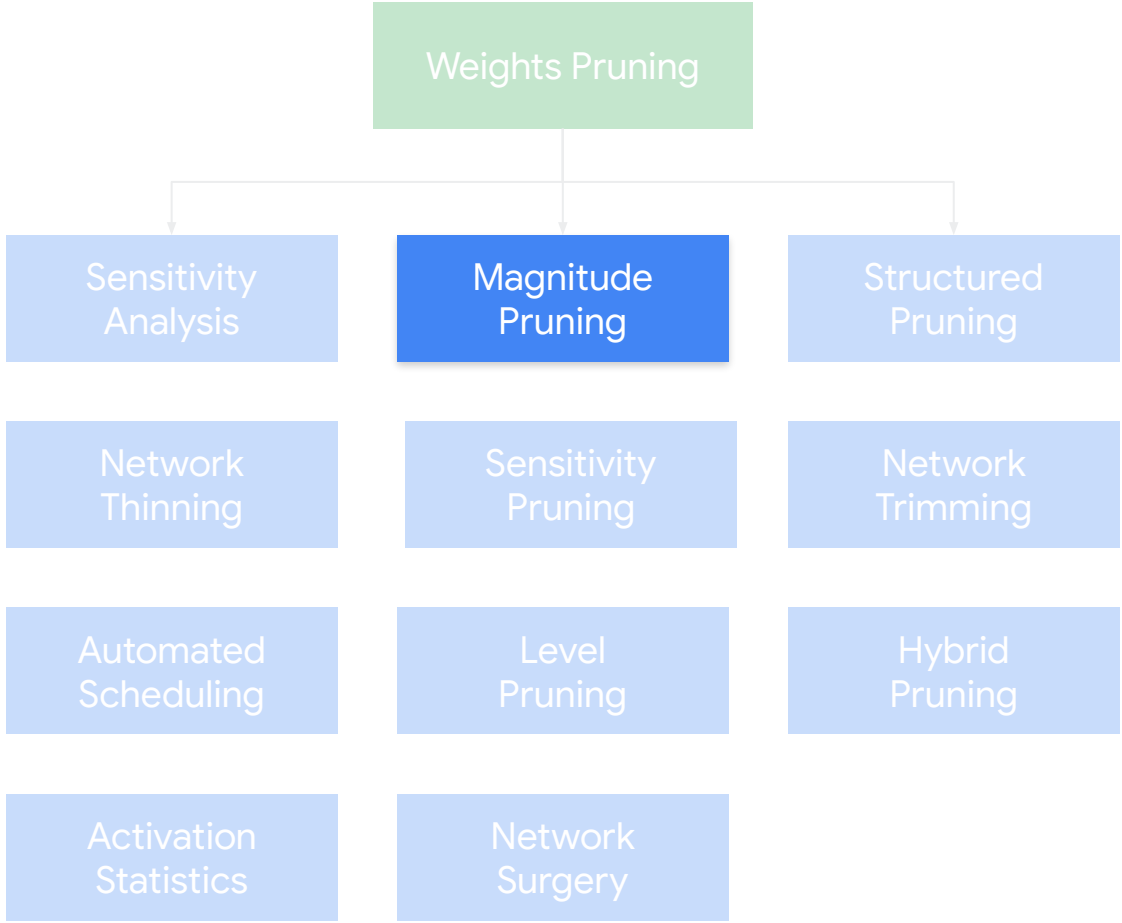
CPU

2X Faster Execution

Pruning




Weights Pruning
Sensitivity Analysis
Magnitude Pruning
Structured Pruning
Network Thinning
Sensitivity Pruning
Network Trimming
Automated Scheduling
Level Pruning
Hybrid Pruning
Activation Statistics
Network Surgery


Weights Pruning
Sensitivity Analysis
Magnitude Pruning
Structured Pruning
Network Thinning
Sensitivity Pruning
Network Trimming
Automated Scheduling
Level Pruning
Hybrid Pruning
Activation Statistics
Network Surgery

Magnitude Pruning

- Sparse models are **easier to compress**

$$thresh(w_i) = \left\{ \begin{array}{ll} w_i : & if \ |w_i| > \lambda \\ 0 : & if \ |w_i| \leq \lambda \end{array} \right\}$$

Magnitude Pruning

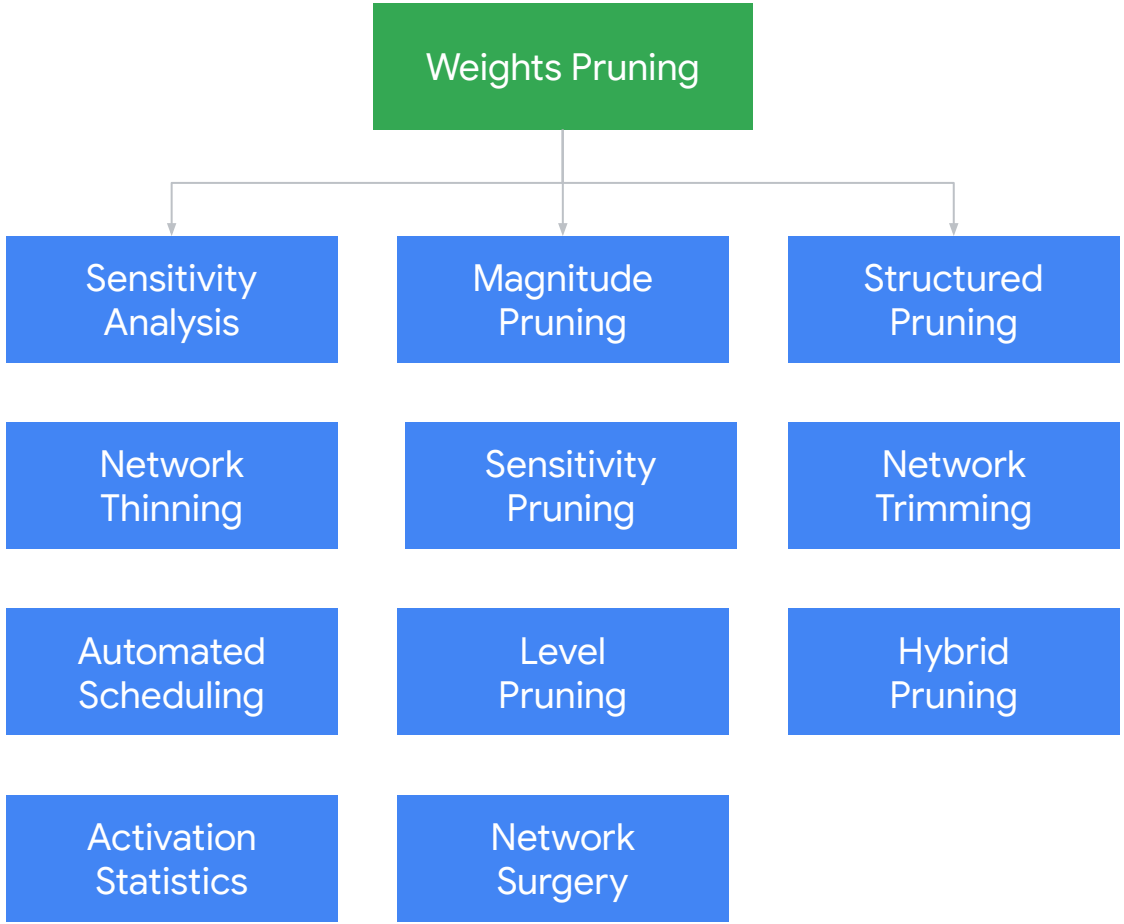
- Sparse models are **easier to compress**
- We can **skip the zeroes during inference for latency** improvements

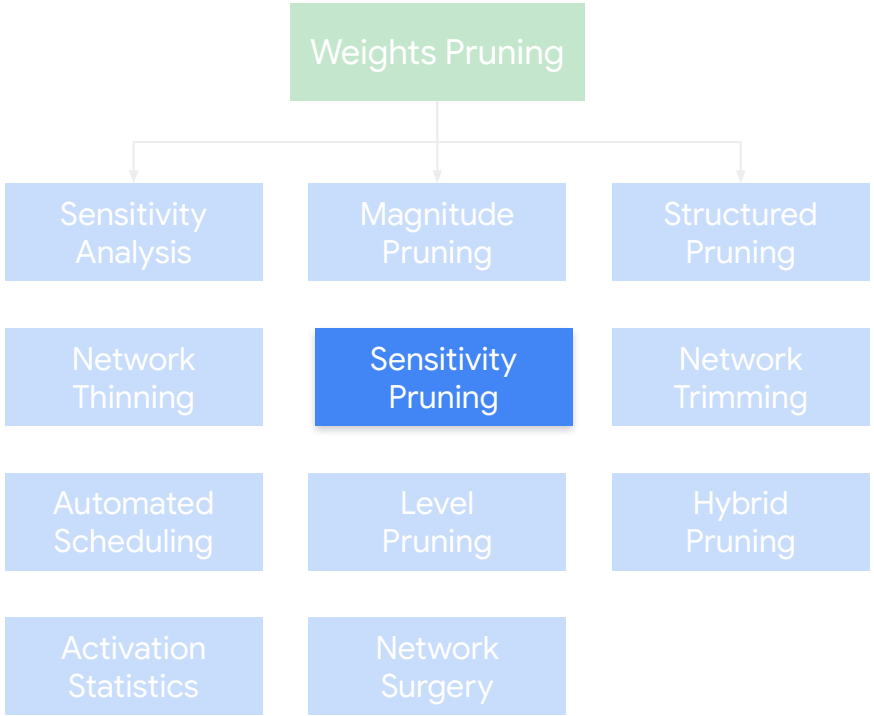
$$thresh(w_i) = \left\{ \begin{array}{ll} w_i : & if \ |w_i| > \lambda \\ 0 : & if \ |w_i| \leq \lambda \end{array} \right\}$$

Magnitude Pruning

- Sparse models are **easier to compress**
- We can **skip the zeroes during inference for latency** improvements
- Up to **6x improvement**

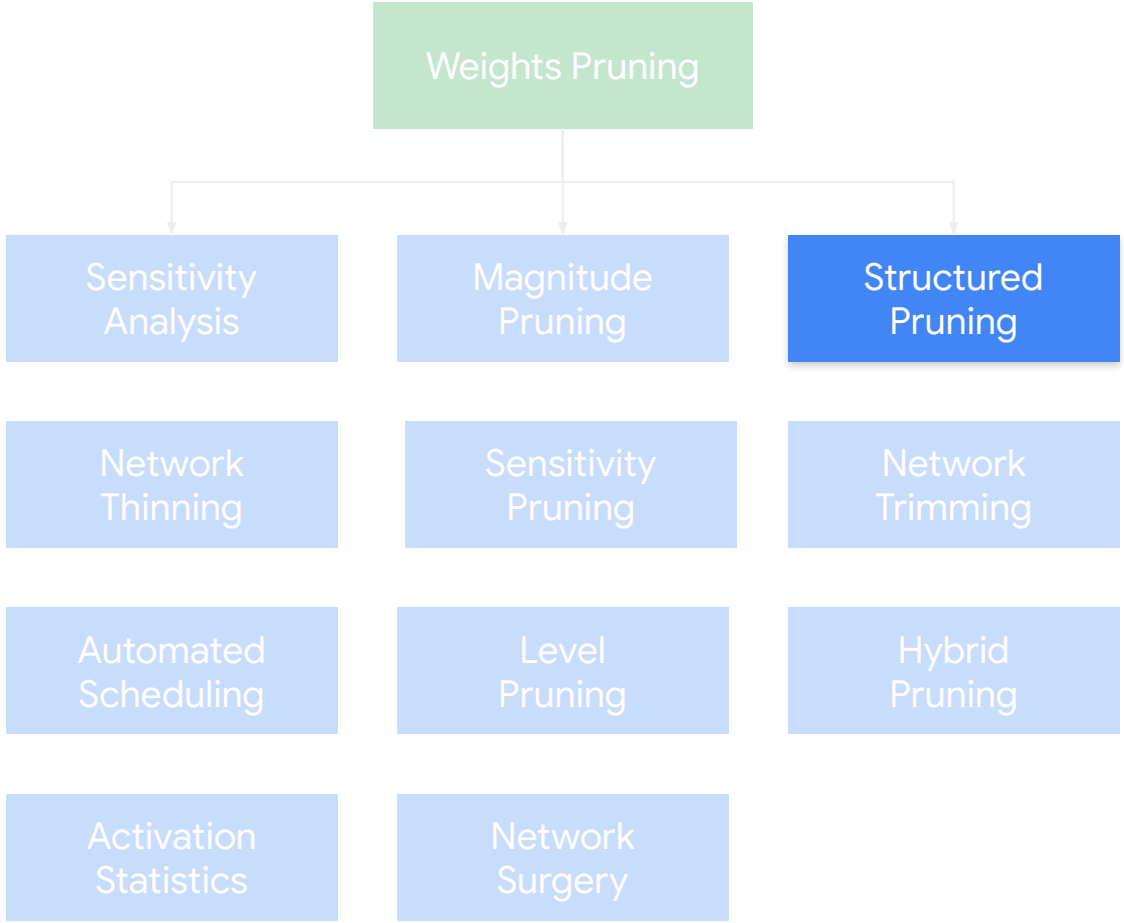
$$thresh(w_i) = \left\{ \begin{array}{ll} w_i : & if \ |w_i| > \lambda \\ 0 : & if \ |w_i| \leq \lambda \end{array} \right\}$$


Weights Pruning
Sensitivity Analysis
Magnitude Pruning
Structured Pruning
Network Thinning
Sensitivity Pruning
Network Trimming
Automated Scheduling
Level Pruning
Hybrid Pruning
Activation Statistics
Network Surgery

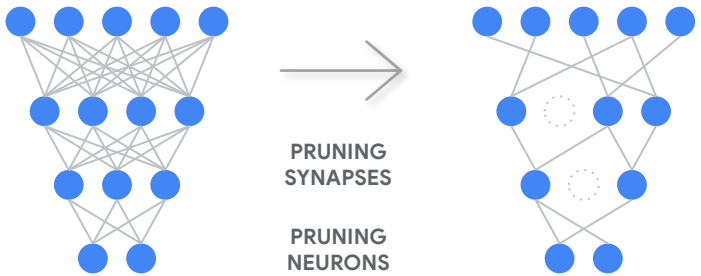


$$thresh(w_i) = \left\{ \begin{array}{l} w_i : \ if \ |w_i| > \lambda \\ 0 : \ if \ |w_i| \leq \lambda \end{array} \right\}$$

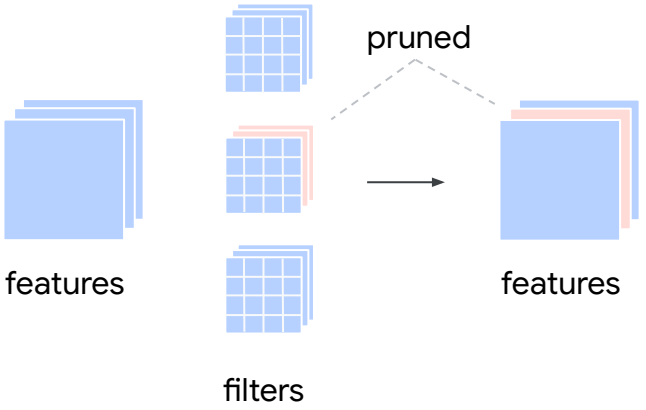
$\lambda = s * \sigma_l$ *where* σ_l *is the std of layer l as measured on the dense model*


Weights Pruning
Sensitivity Analysis
Magnitude Pruning
Structured Pruning
Network Thinning
Sensitivity Pruning
Network Trimming
Automated Scheduling
Level Pruning
Hybrid Pruning
Activation Statistics
Network Surgery

Unstructured Pruning



Structured Pruning



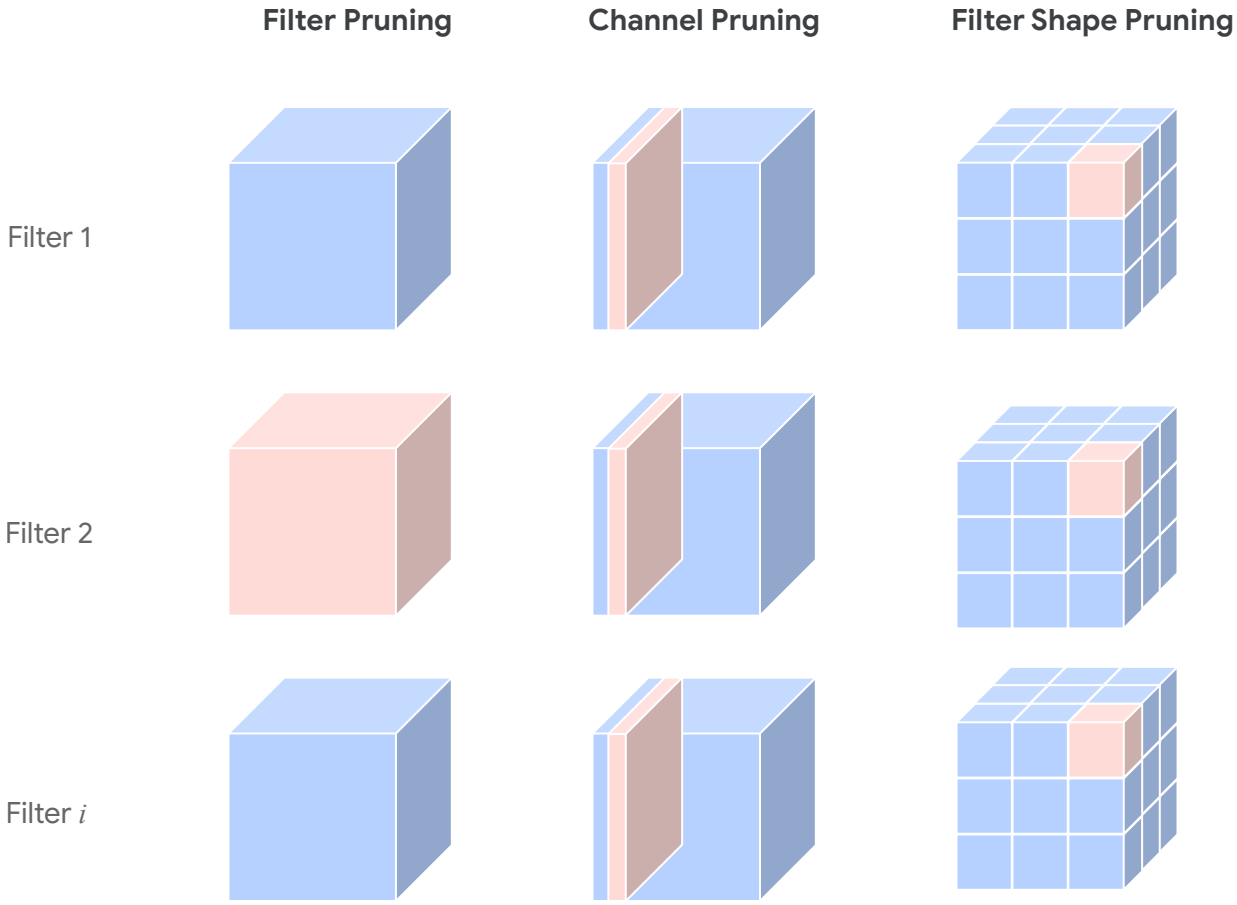

Filter Pruning
Channel Pruning
Filter Shape Pruning
Filter 1
Filter 2
Filter i

Image Classification

Model	Non-sparse Top-1 Accuracy	Sparse Accuracy	Sparsity
InceptionV3	78.1%	78.0%	50%
		76.1%	75%
		74.6%	87.5%
MobilenetV1 224	71.04%	70.84%	50%

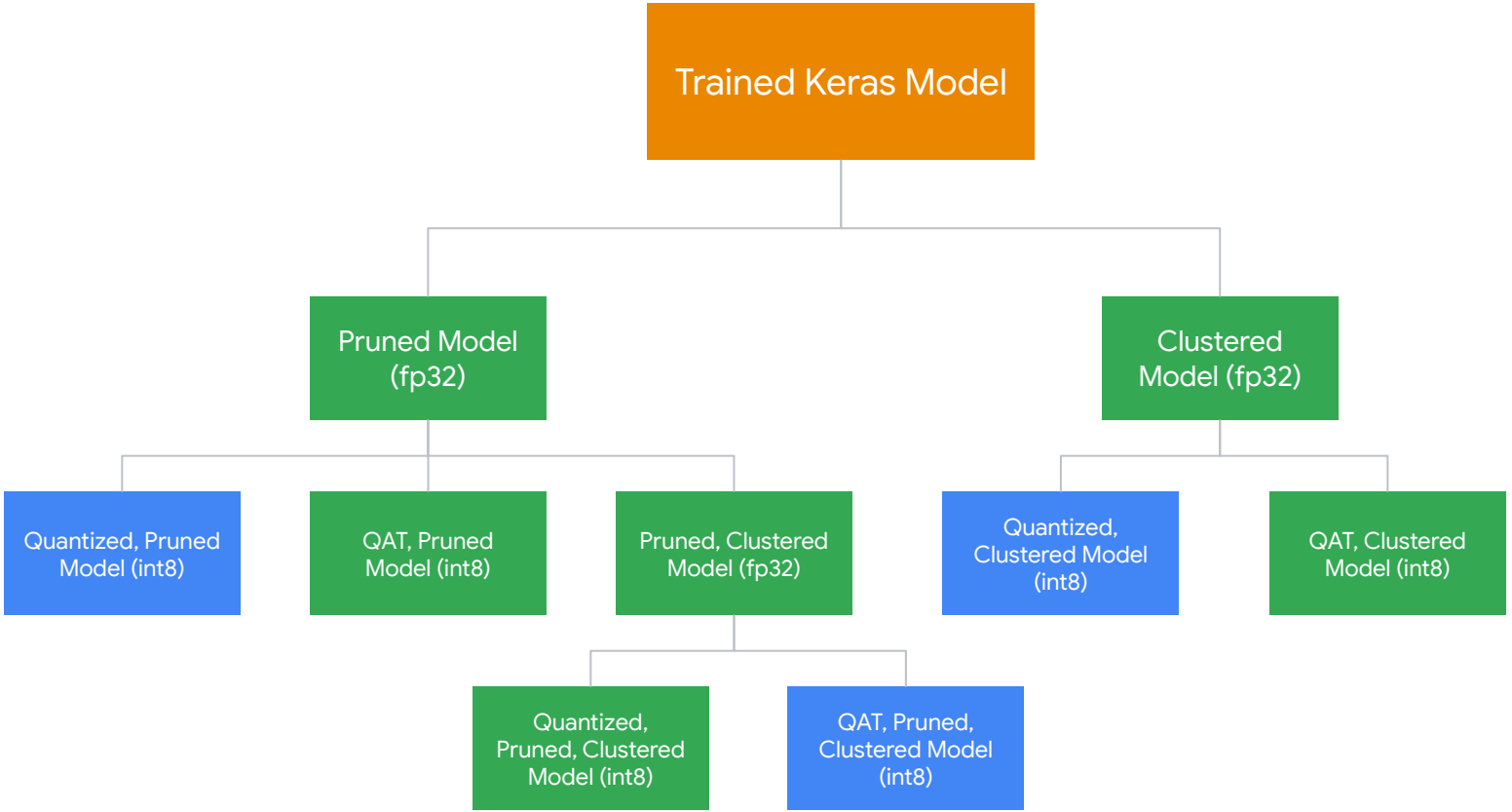
The models were tested on Imagenet.

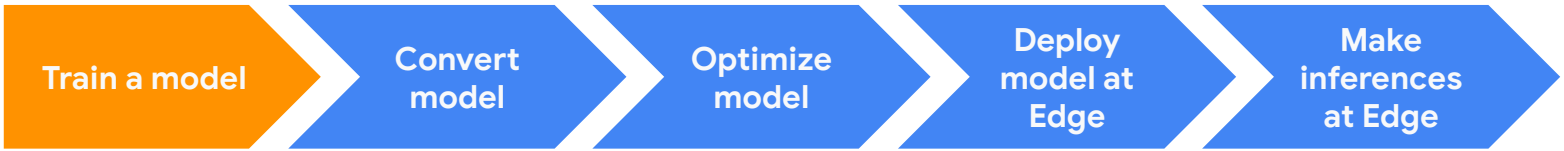
Language Translation

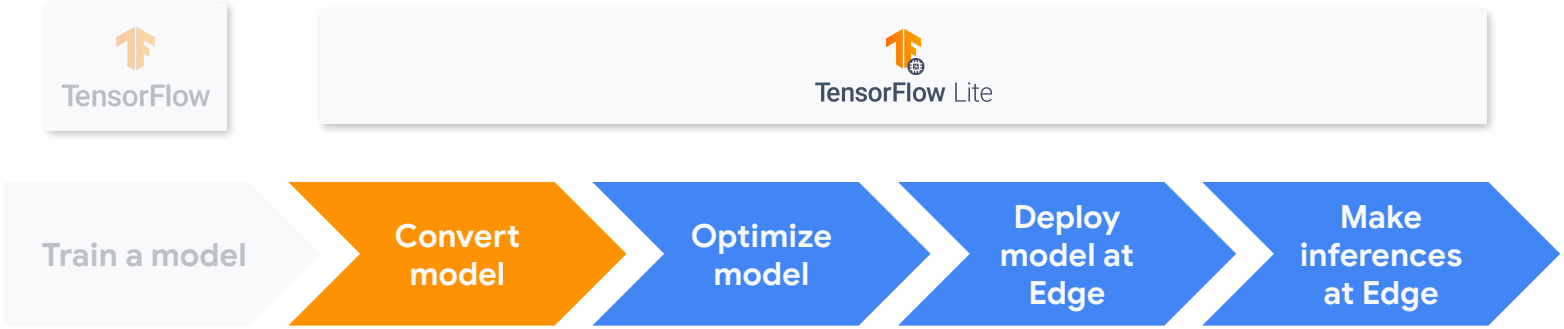
Model	Non-sparse BLEU	Sparse BLEU	Sparsity
GNMT EN-DE	26.77	26.86	80%
		26.52	85%
		26.19	90%
GNMT DE-EN	29.47	29.50	80%
		29.24	85%
		28.81	90%

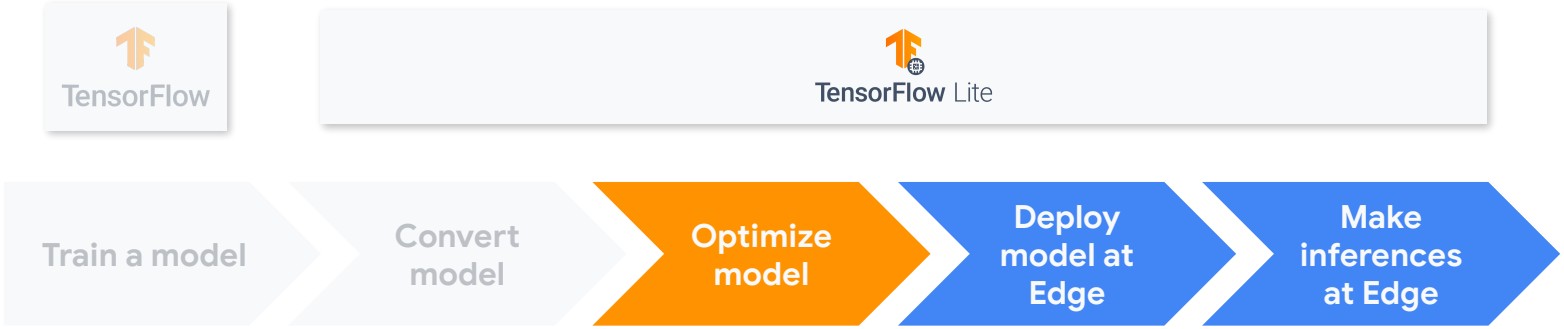
Keyword Spotting

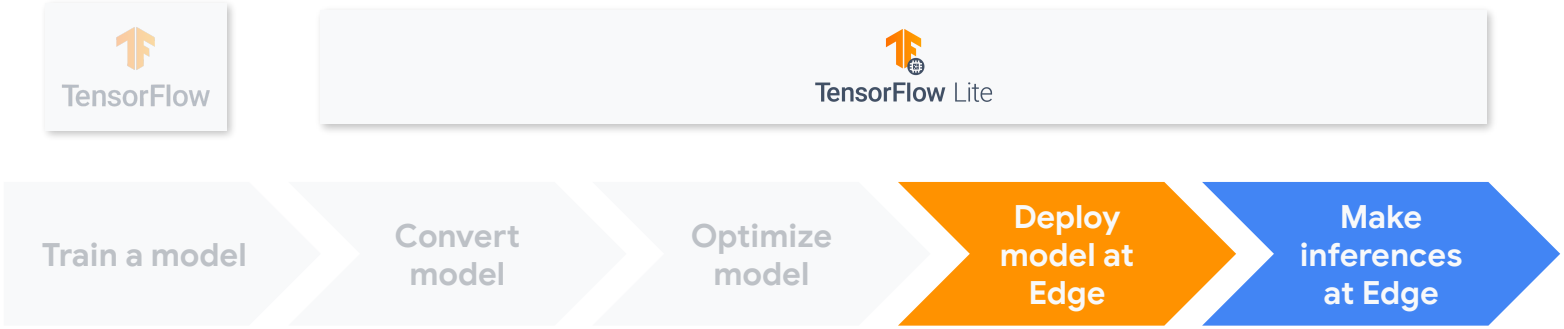
Model	Non-sparse Accuracy	Structured Sparse Accuracy (2 by 4 pattern)	Random Sparse Accuracy (target sparsity 50%)
DS-CNN-L	95.23	94.33	94.84

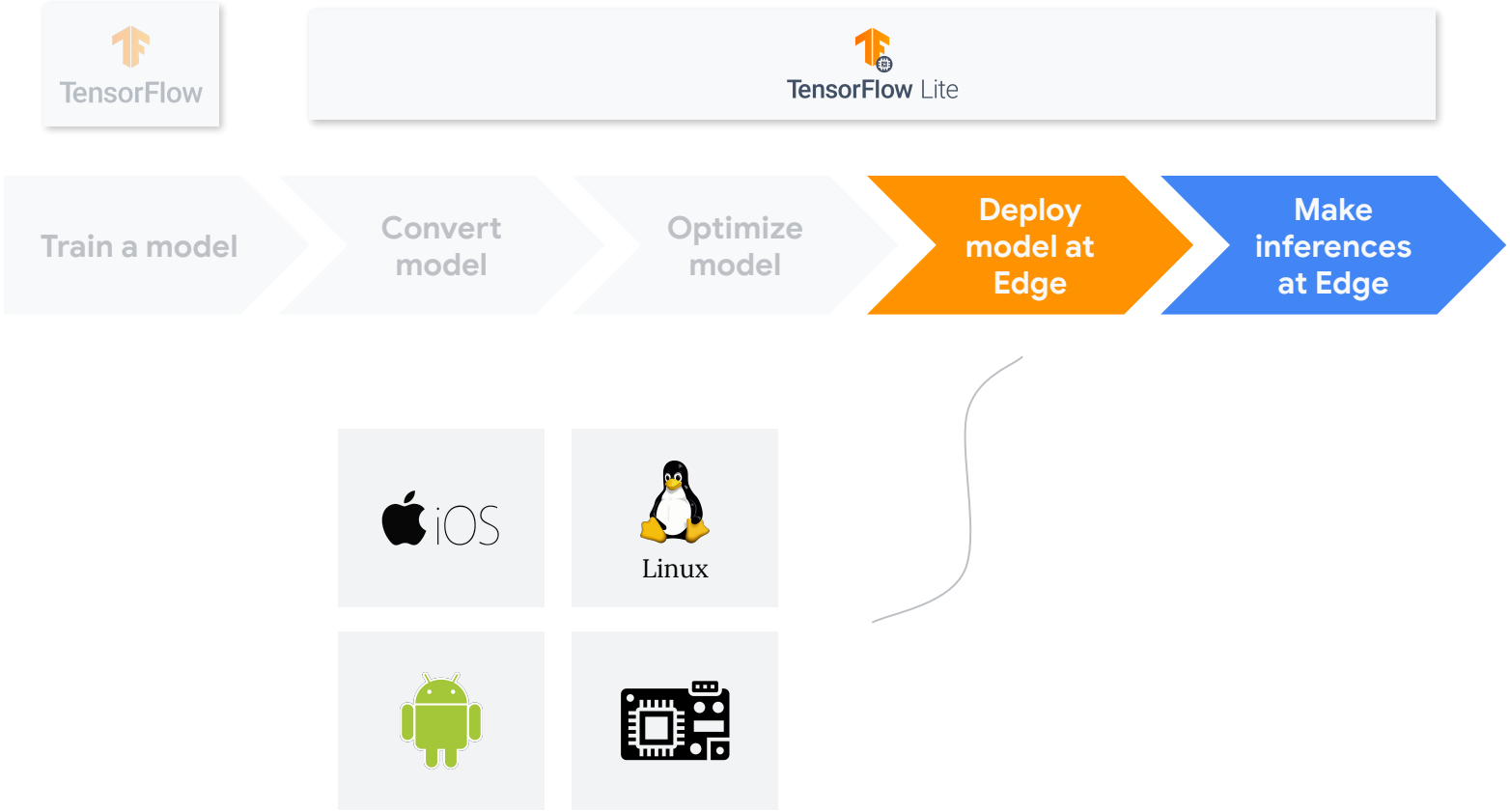

Trained Keras Model
Pruned Model (fp32)
Clustered Model (fp32)
Quantized, Pruned Model (int8)
QAT, Pruned Model (int8)
Pruned, Clustered Model (fp32)
Quantized, Clustered Model (int8)
QAT, Clustered Model (int8)
Quantized, Pruned, Clustered Model (int8)
QAT, Pruned, Clustered Model (int8)

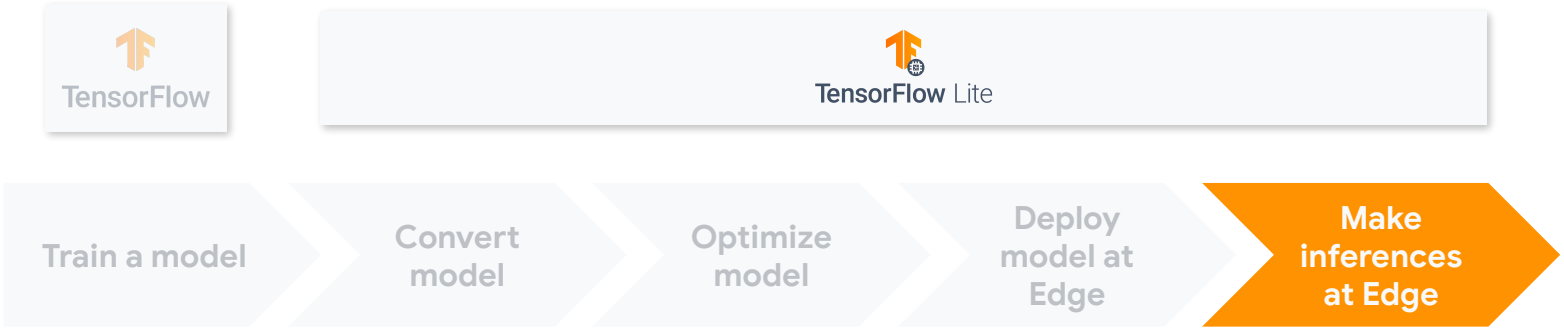

TensorFlow
TensorFlow Lite
Train a model
Convert model
Optimize model
Deploy model at Edge
Make inferences at Edge

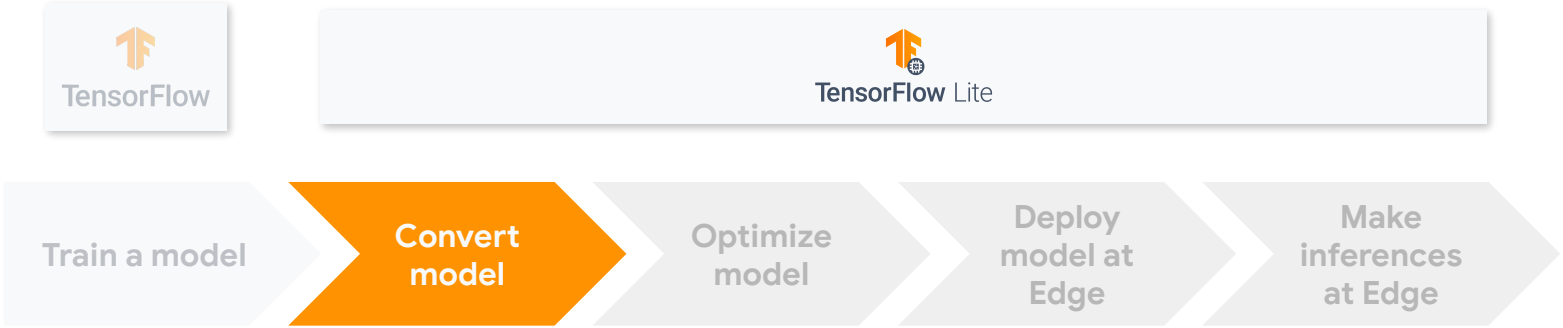

TensorFlow
TensorFlow Lite
Train a model
Convert model
Optimize model
Deploy model at Edge
Make inferences at Edge


TensorFlow
TensorFlow Lite
Train a model
Convert model
Optimize model
Deploy model at Edge
Make inferences at Edge

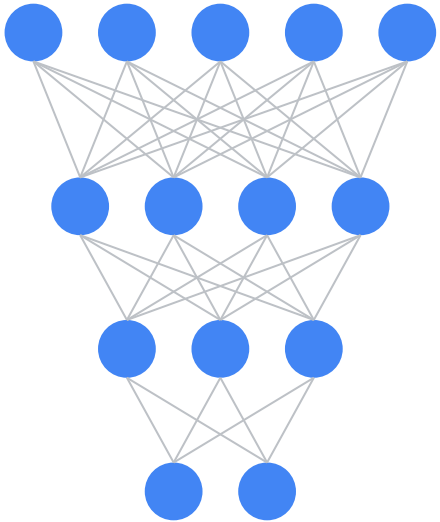

TensorFlow
TensorFlow Lite
Train a model
Convert model
Optimize model
Deploy model at Edge
Make inferences at Edge


TensorFlow
TensorFlow Lite
Train a model
Convert model
Optimize model
Deploy model at Edge
Make inferences at Edge
iOS
Linux

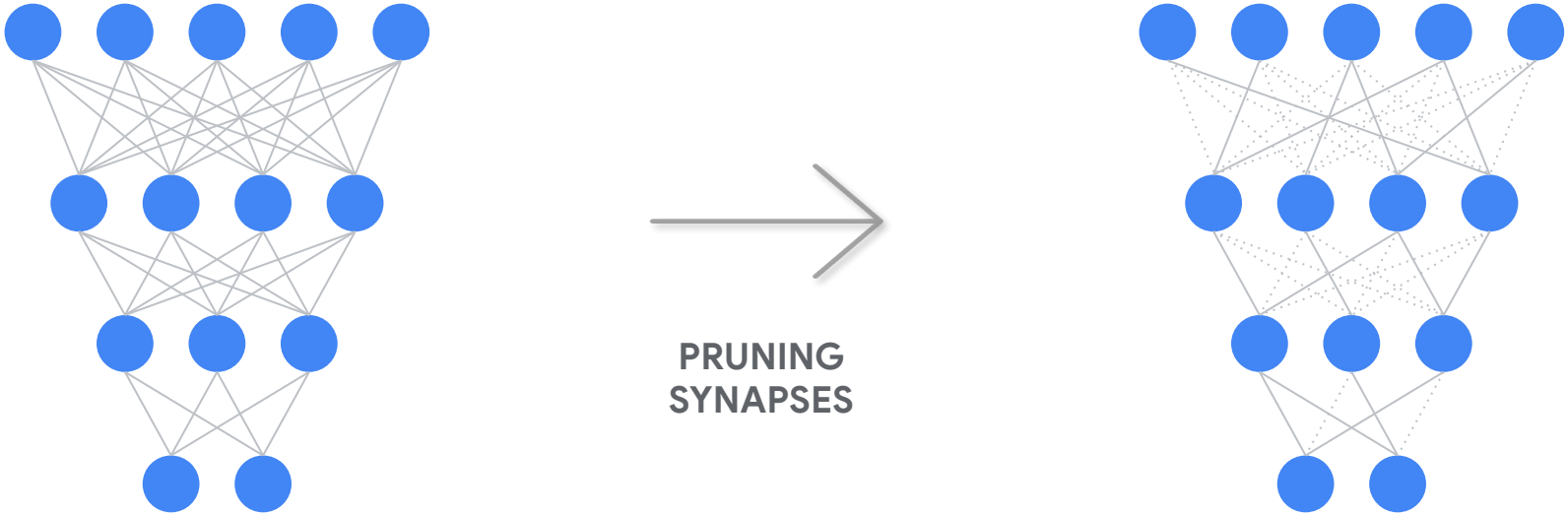

TensorFlow
TensorFlow Lite
Train a model
Convert model
Optimize model
Deploy model at Edge
Make inferences at Edge


TensorFlow
TensorFlow Lite
Train a model
Convert model
Optimize model
Deploy model at Edge
Make inferences at Edge

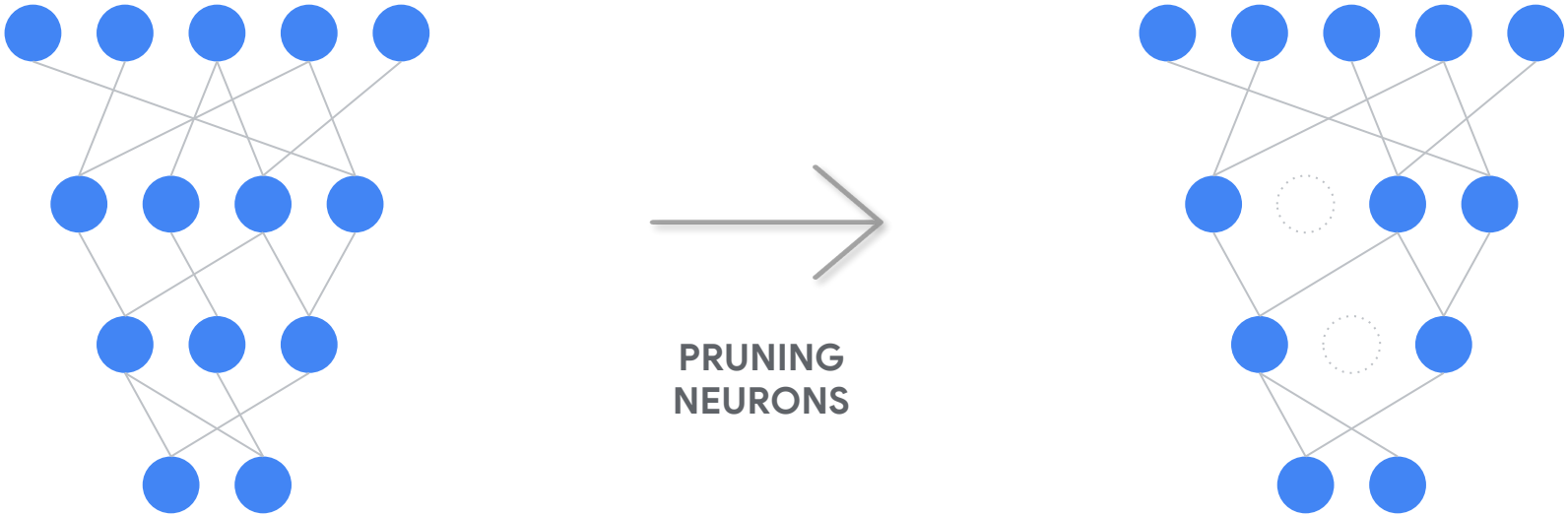
Pruning



Pruning



Pruning

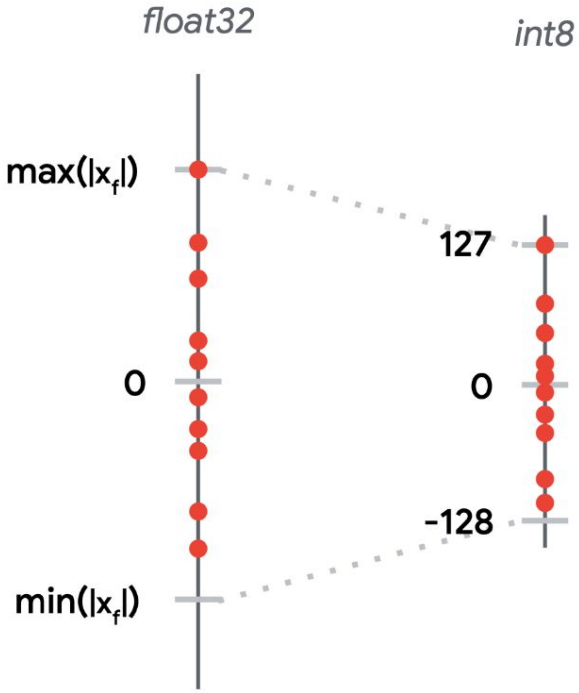


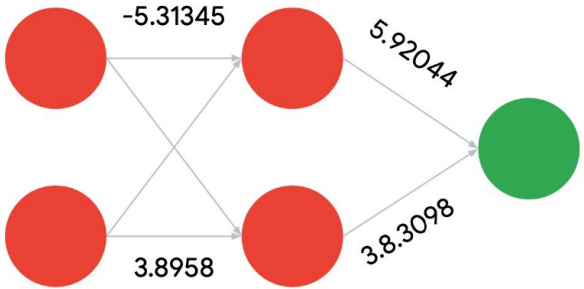
Model Quantization

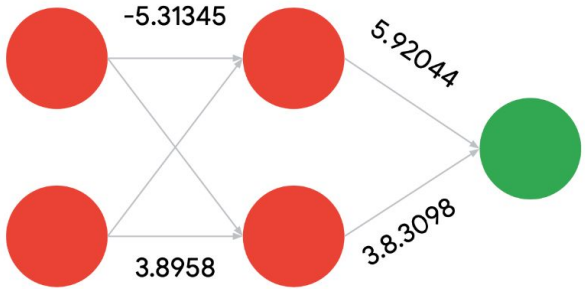
Quantization

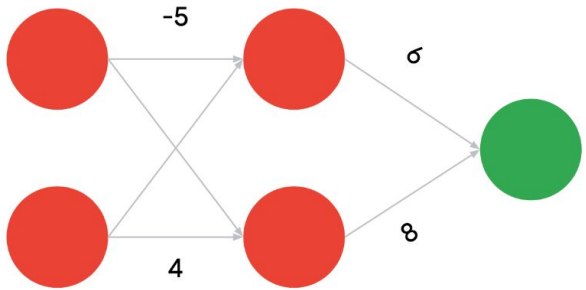
Quantization is the process of transforming an ML program into an approximated representation with available lower precision operations.

Quantization




-5.31345
5.92044
3.8958
3.8.3098


-5.31345
5.92044
3.8958
3.8.3098


-5
6
4
8

Why do we Quantize?

Lower Precision Efficiencies

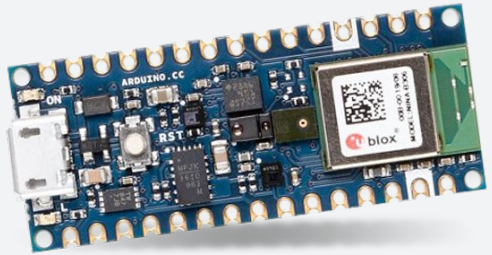
Reduce Memory

8-bit integer parameters means 4x smaller model

Storage & RAM Size

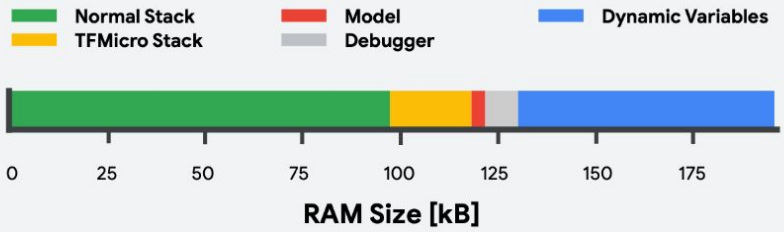
Storage size: Smaller neural network models occupy less storage space on your device, and in moving from 32-bits to 8-bits we readily get **4x** reduction in memory.

The Arduino Nano 33 BLE only has **256KB** of RAM (memory) and **1MB** of Flash (storage)



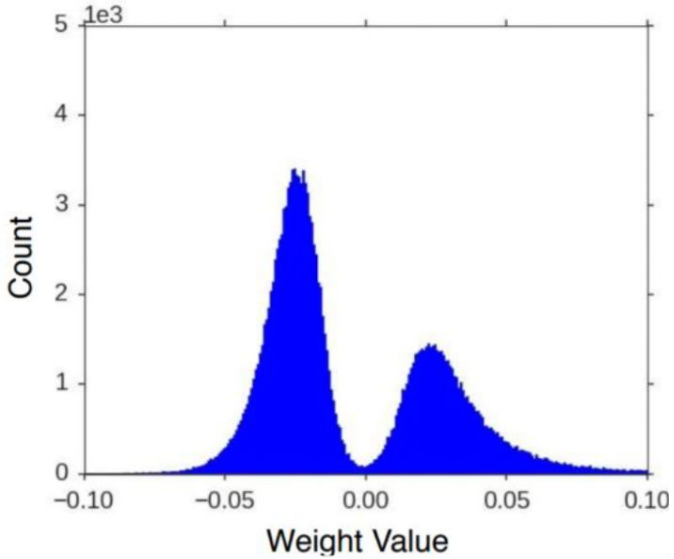
Storage & RAM Size

Less memory usage: Smaller models use less RAM when they are run, which frees up memory for other parts of your application to use, and can translate to better performance and stability.



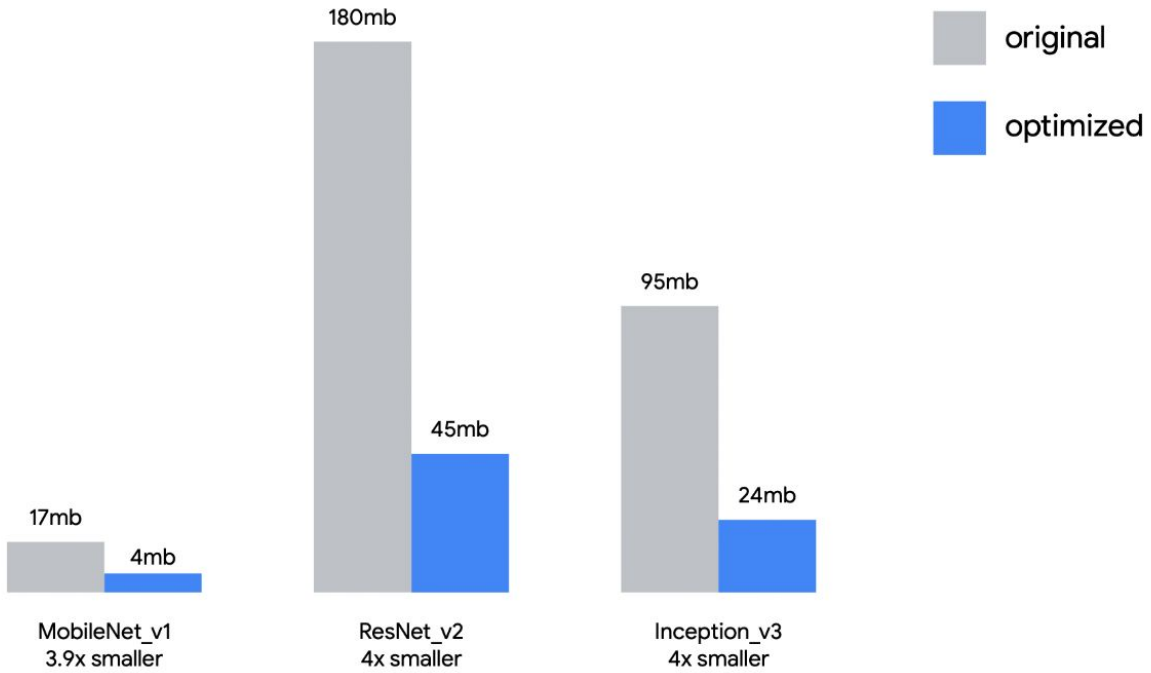
Weight Ranges

Weight distribution for AlexNet shows how most weight values are **concentrated** in a small range.



Source: “Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding” by Song Han, Huizi Mao, William J. Dally

Model Size


180mb
original
optimized
95mb
45mb
24mb
17mb
4mb
MobileNet_v1
3.9x smaller
ResNet_v2
4x smaller
Inception_v3
4x smaller

Lower Precision Efficiencies

Reduce Memory

8-bit integer parameters means 4x smaller model

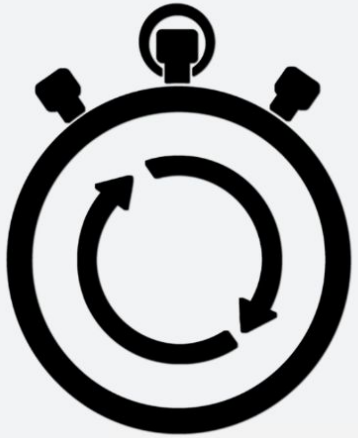
Faster Compute

Integer operations are faster

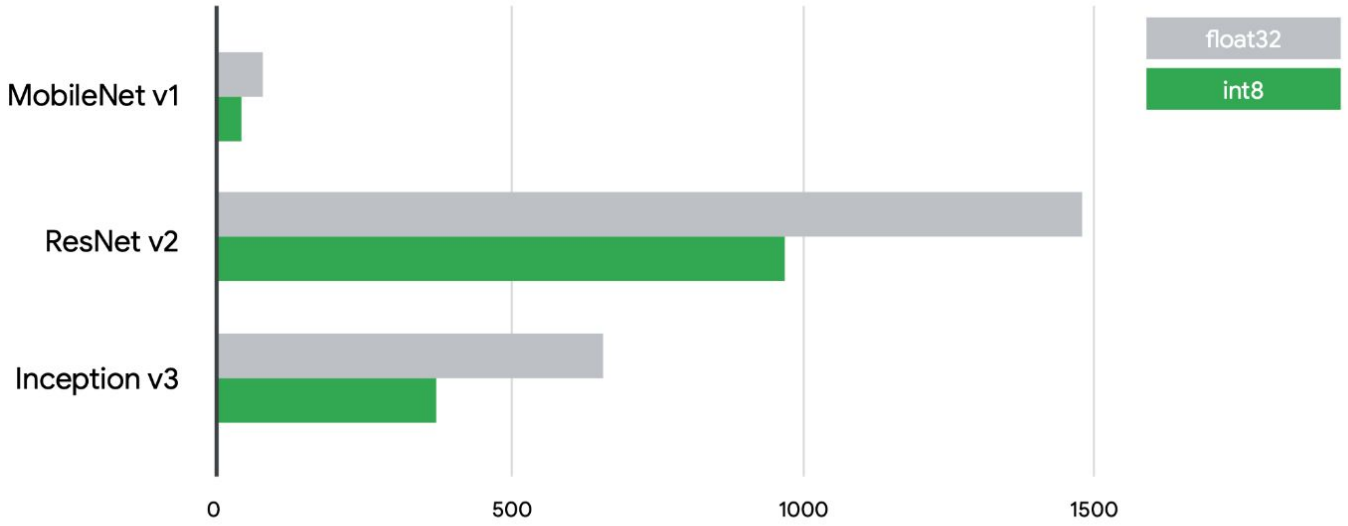
Latency

- ***Int8*** (v. fp32) format severely **reduces the computation** to run inference using a model, resulting in lower latency

- Latency optimizations can also have a notable impact on **power consumption**.



Int8 v. Float (CPU time per inference)



Quantized models are up to 2–4x faster on CPU and 4x smaller.

Lower Precision Efficiencies

Reduce Memory

8-bit integer parameters means 4x smaller model

Faster Compute

Integer operations are faster

Reduce Power

Integer operations consume less power

Lower Precision Efficiencies

Reduce Memory

8-bit integer parameters means 4x smaller model

Faster Compute

Integer operations are faster

Reduce Power

Integer operations consume less power

Reduce Bandwidth

Smaller models and dynamic values reduce bandwidth pressure

Lower Precision Efficiencies

Reduce Memory

8-bit integer parameters means 4x smaller model

Faster Compute

Integer operations are faster

Reduce Power

Integer operations consume less power

Reduce Bandwidth

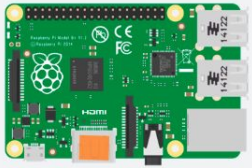
Smaller models and dynamic values reduce bandwidth pressure

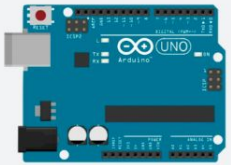
Hardware Compatibility

Integer operations supported across CPU/DSP/NPUs

Portability **Trade-offs**

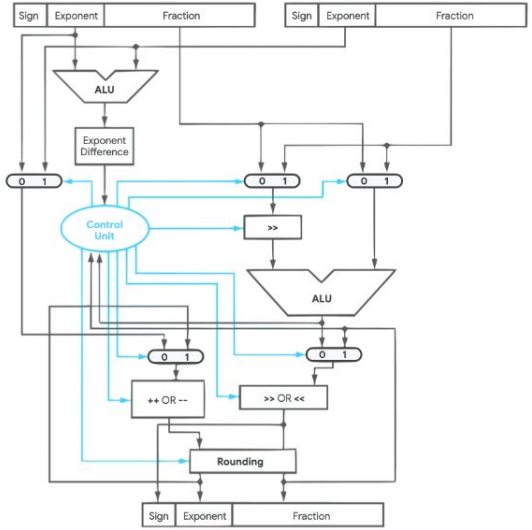
Not all embedded systems are created equal. Sacrifice **portability** across systems for **efficiency**.





✗	✓
Specific HW Implementation of a Library	

Single Precision
IEEE 754 Floating-Point Standard



Option 2

Lower Code Portability	✗
Cost ($)	✓
Power (W)	✓
Eng. Effort	✓

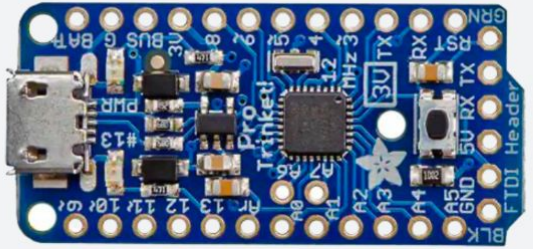
Why Quantization is Necessary

AVR microcontroller

- Manufacturer: Atmel
- ATmega328P-PU MCU

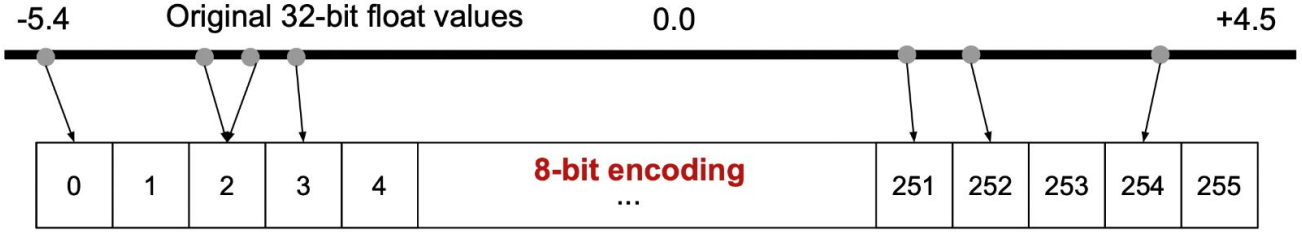
Features

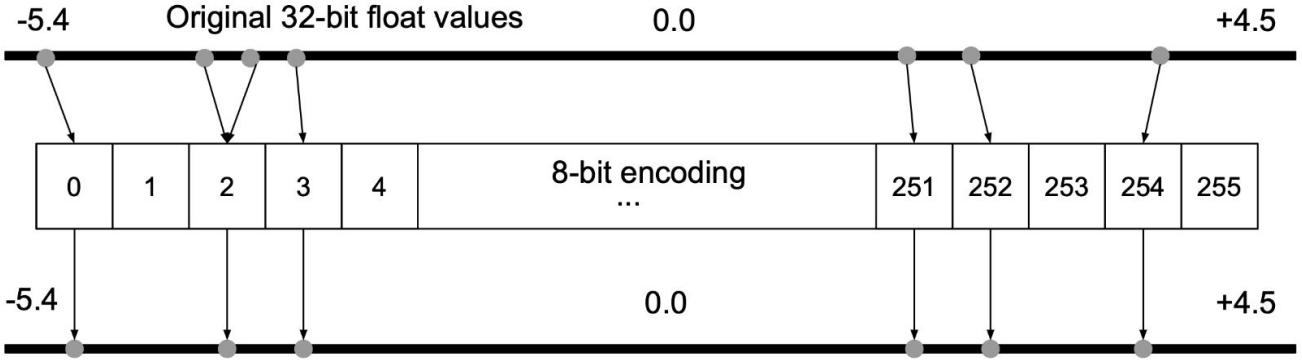
- Core size: 8-bit
- Speed: up to 20MHz
- Flash memory size: 32Kb (16K x 16)
- RAM size: 2K x 8



How do we Quantize?


-5.4
Original 32-bit float values
0.0
+4.5


-5.4
Original 32-bit float values
0.0
+4.5
0
1
2
3
4
8-bit encoding
...
251
252
253
254
255



Reconstructed 32-bit float values

Quantization Types

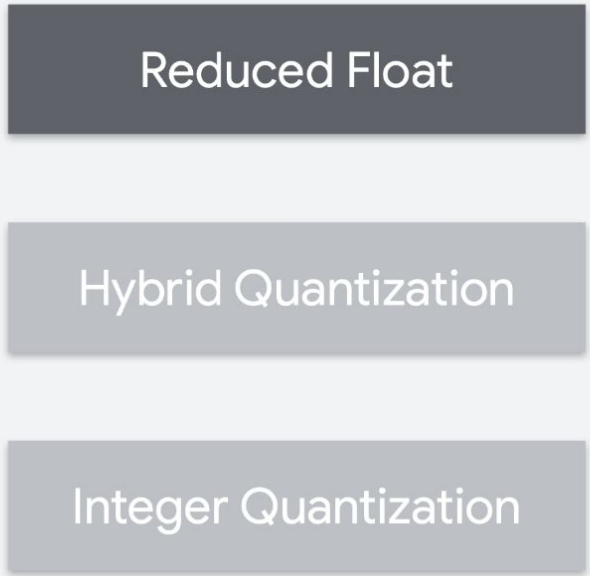
Reduced Float

Hybrid Quantization

Integer Quantization

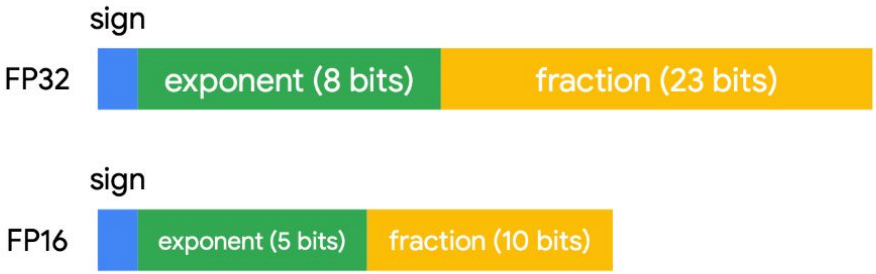
Reduced float

- float16 parameters
- float16 computations



Reduced float

- float16 parameters
- float16 computations



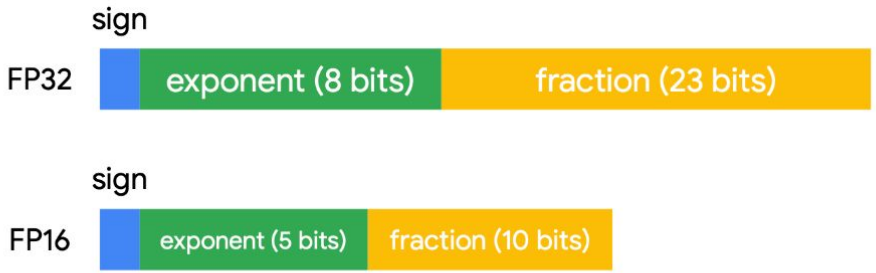
Reduced Float

Hybrid Quantization

Integer Quantization

Reduced float

- float16 parameters
- float16 computations

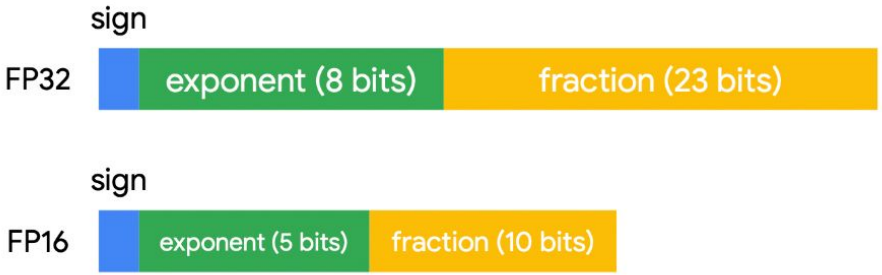


Benefits

2x reduction in model parameters (32 bit → 16 bit)

Reduced float

- float16 parameters
- float16 computations



Benefits

2x reduction in model parameters (32 bit ⟶ 16 bit)

Potential future speed-ups faster as hardware enables optimized operations

Negligible accuracy loss

Quantization Types

Reduced Float

Hybrid Quantization

Integer Quantization

Hybrid quantization

- 8-bit integer weights
- 32-bit float biases & activations
- Integer and floating point computations

Benefits

4x reduction in model parameters (32 bit → 8 bit)

Hybrid quantization

- 8-bit integer weights
- 32-bit float biases & activations
- Integer and floating point computations

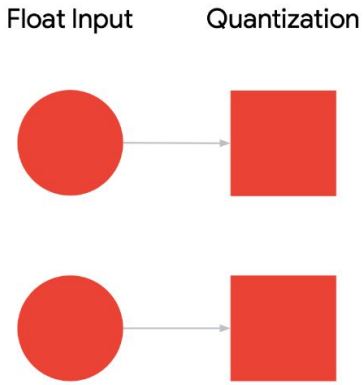
Benefits

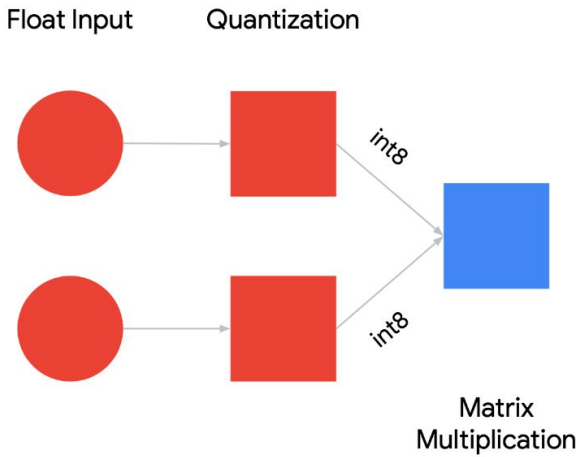
4x reduction in model parameters (32 bit ⟶ 8 bit)

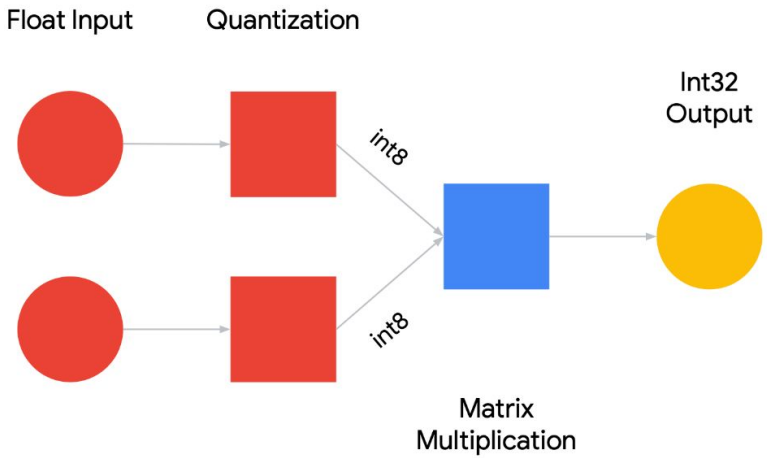
10-50% faster execution for Convolution Models (CPU hybrid v. CPU float)

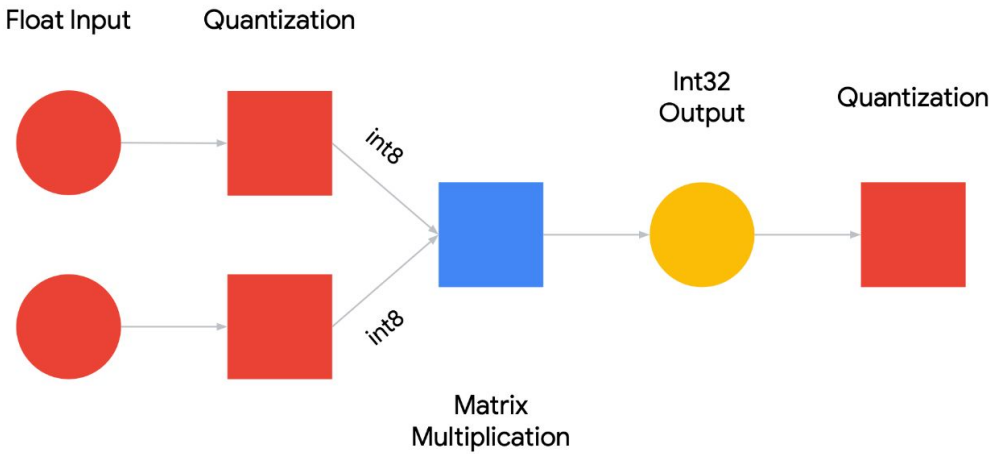
2-3x faster on fully-connected & RNN-based models (CPU quant v. CPU float)

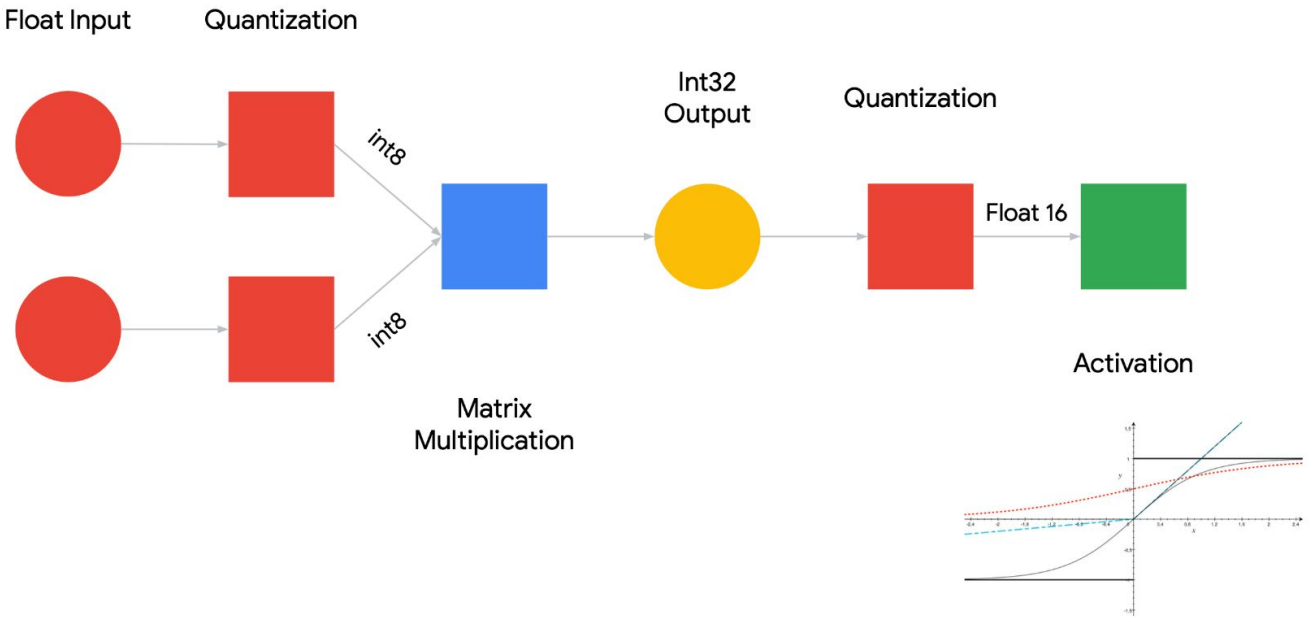
Float Input

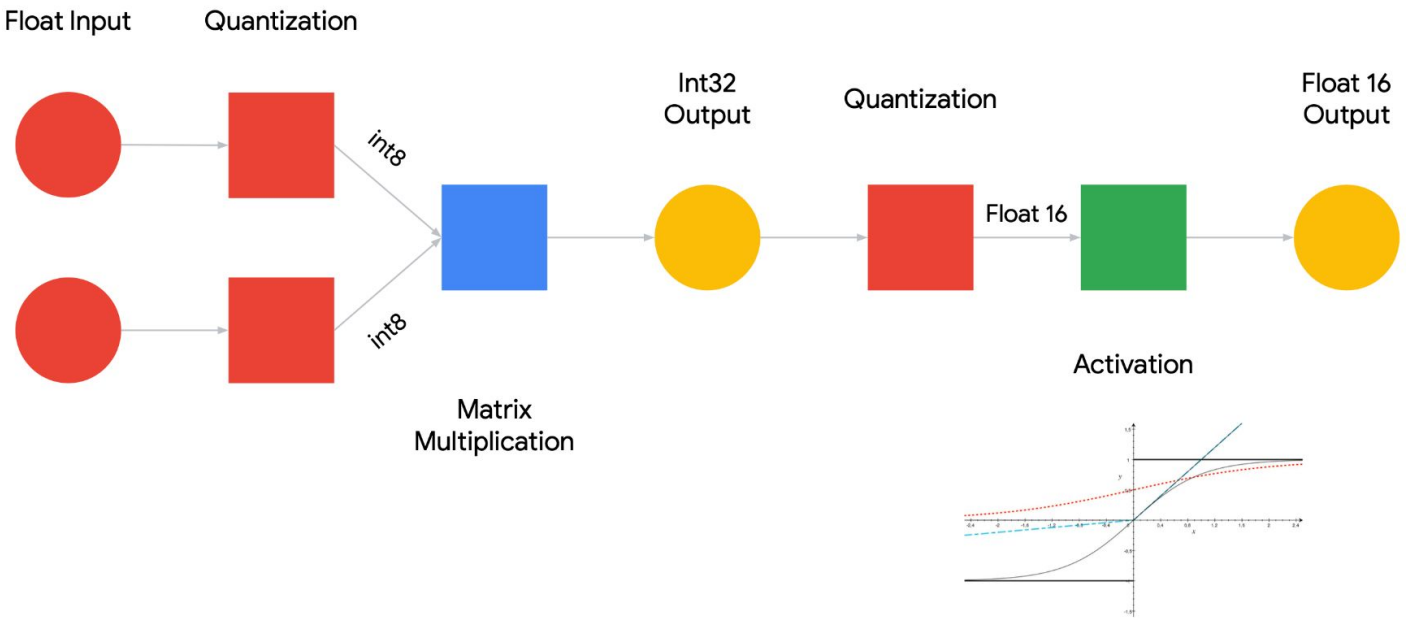

Float Input
Quantization


Float Input
Quantization
int8
int8
Matrix
Multiplication


Float Input
Quantization
Int32
Output
int8
int8
Matrix
Multiplication


Float Input
Quantization
int8
int8
Matrix
Multiplication
Int32
Output
Quantization


Float Input
Quantization
int8
int8
Matrix
Multiplication
Int32
Output
Quantization
Float 16
Activation


Float Input
Quantization
int8
int8
Matrix Multiplication
Int32 Output
Quantization
Float 16
Float 16 Output
Activation

Hybrid quantization

- 8-bit integer weights
- 32-bit float biases & activations
- Integer and floating point computations

Benefits

4x reduction in model parameters (32 bit ⟶ 8 bit)

10-50% faster execution for Convolution Models (CPU hybrid v. CPU float)

2-3x faster on fully-connected & RNN-based models (CPU quant v. CPU float)

Quantization Types

Reduced Float

Hybrid Quantization

Integer Quantization

Integer quantization

- 8-bit integer weights
- 8 and 16-bit biases & activations
- Only integer computations

Benefits

4x reduction in model parameters (32 bit → 8 bit)

1.5x faster execution for Convolution Models (CPU integer v. CPU float)

2-4x faster on fully-connected & RNN-based models (CPU quant v. CPU float)

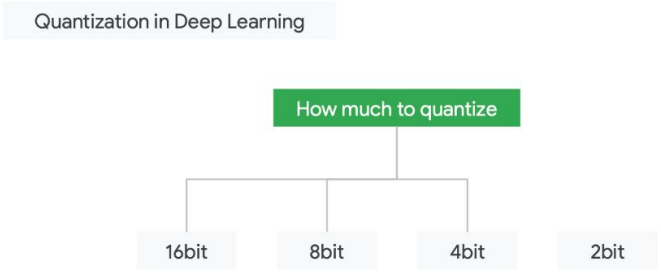
Enables execution on ML accelerators (e.g., Edge-TPU)

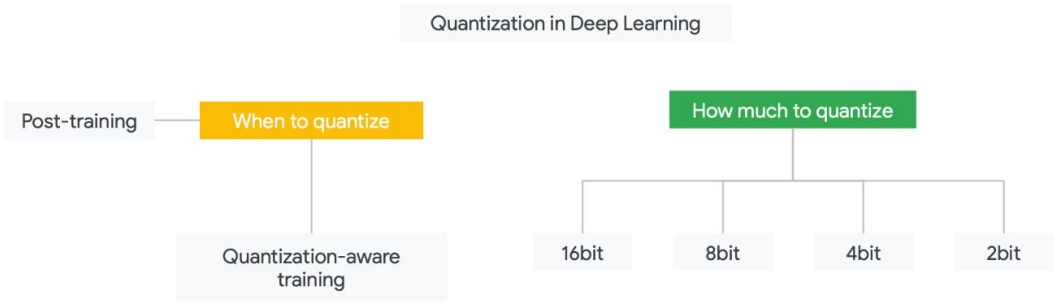
Technique	Ease of use	Accuracy	Latency	Compatibility
Reduced float (post-training)	No data required	Negligible loss	Same or faster than float32	Float16 support or fallback to float32
"Hybrid" quantization (post-training)	No data required	Small loss (≤ float16)	Faster than float	Needs float and integer support
Integer quantization (post-training0	Unlabeled data	Accuracy ≤ hybrid	Fastest	Integer only
Integer quantization (during training)	Labeled training data	Accuracy ≥ integer	Fastest	Integer only

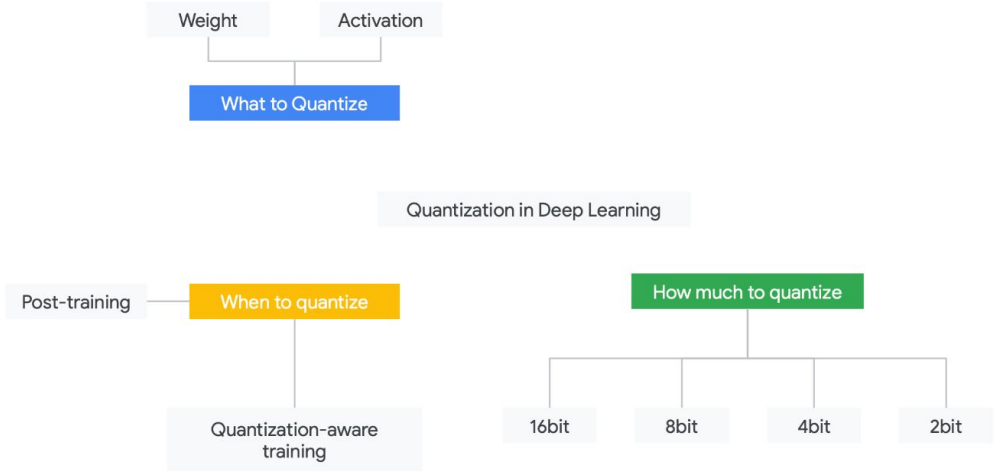
Technique	Ease of use	Accuracy	Latency	Compatibility
Reduced float (post-training)	No data required	Negligible loss	Same or faster than float32	Float16 support or fallback to float32
"Hybrid" quantization (post-training)	No data required	Small loss (≤ float16)	Faster than float	Needs float and integer support
Integer quantization (post-training0	Unlabeled data	Accuracy ≤ hybrid	Fastest	Integer only
Integer quantization (during training)	Labeled training data	Accuracy ≥ integer	Fastest	Integer only

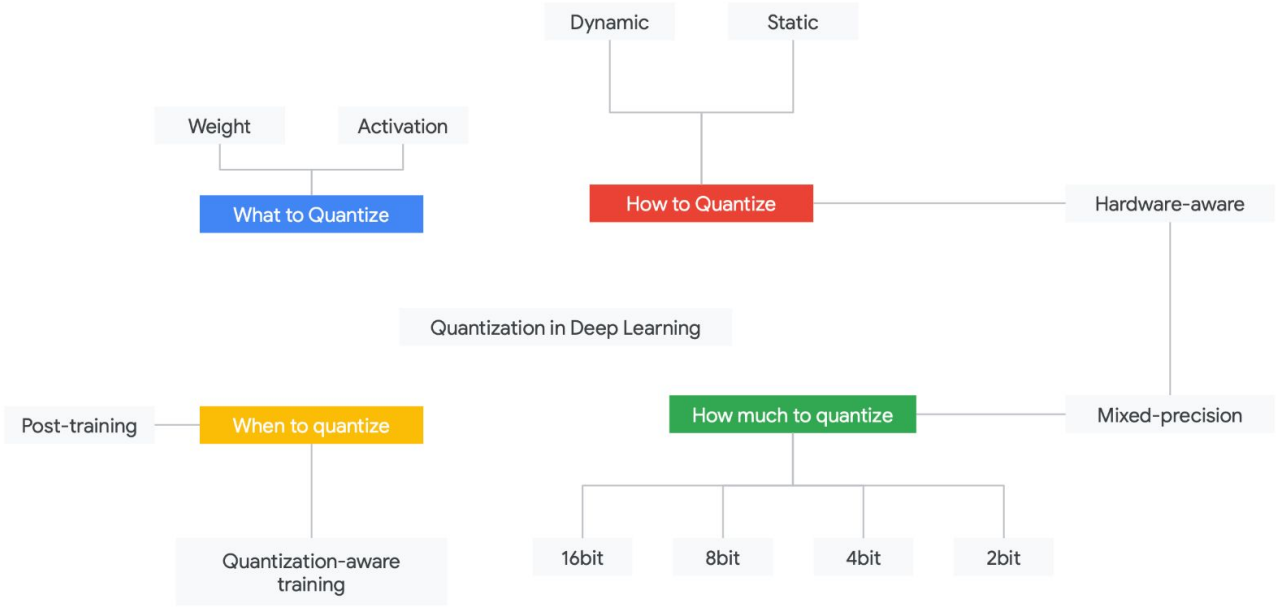
Technique	Ease of use	Accuracy	Latency	Compatibility
Reduced float (post-training)	No data required	Negligible loss	Same or faster than float32	Float16 support or fallback to float32
"Hybrid" quantization (post-training)	No data required	Small loss (≤ float16)	Faster than float	Needs float and integer support
Integer quantization (post-training0	Unlabeled data	Accuracy ≤ hybrid	Fastest	Integer only
Integer quantization (during training)	Labeled training data	Accuracy ≥ integer	Fastest	Integer only

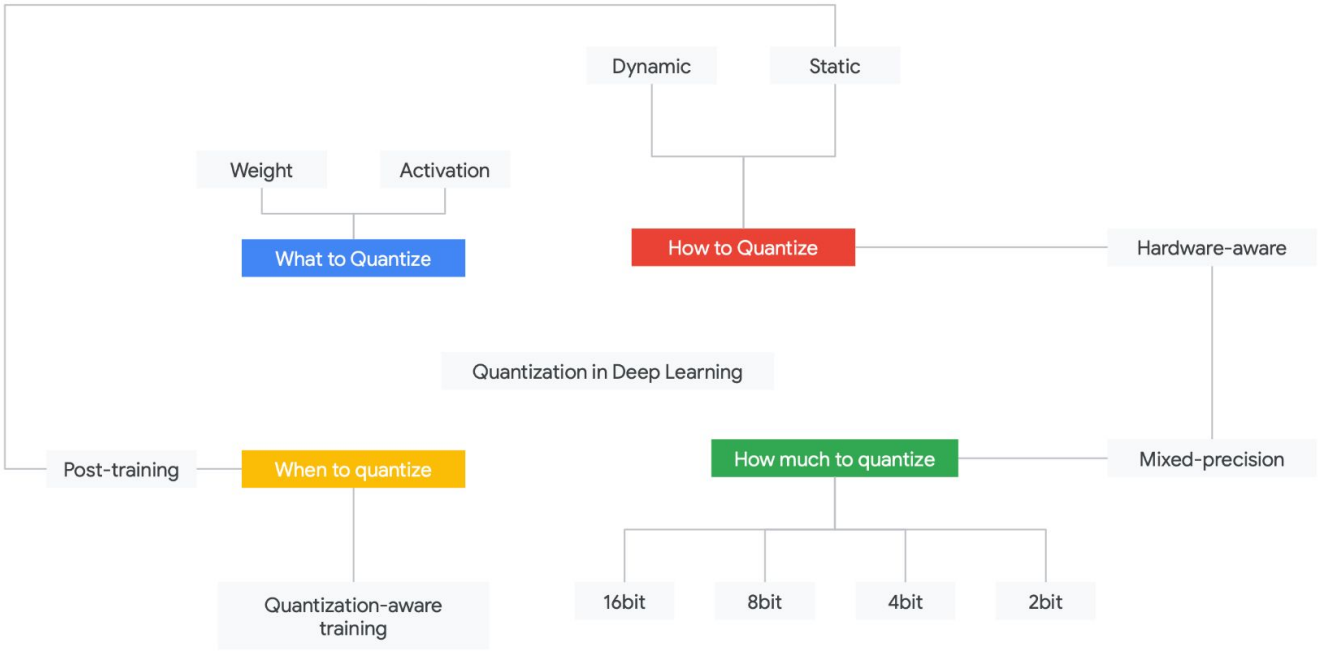
Technique	Ease of use	Accuracy	Latency	Compatibility
Reduced float (post-training)	No data required	Negligible loss	Same or faster than float32	Float16 support or fallback to float32
"Hybrid" quantization (post-training)	No data required	Small loss (≤ float16)	Faster than float	Needs float and integer support
Integer quantization (post-training0	Unlabeled data	Accuracy ≤ hybrid	Fastest	Integer only
Integer quantization (during training)	**Labeled training data**	Accuracy ≥ integer	Fastest	Integer only

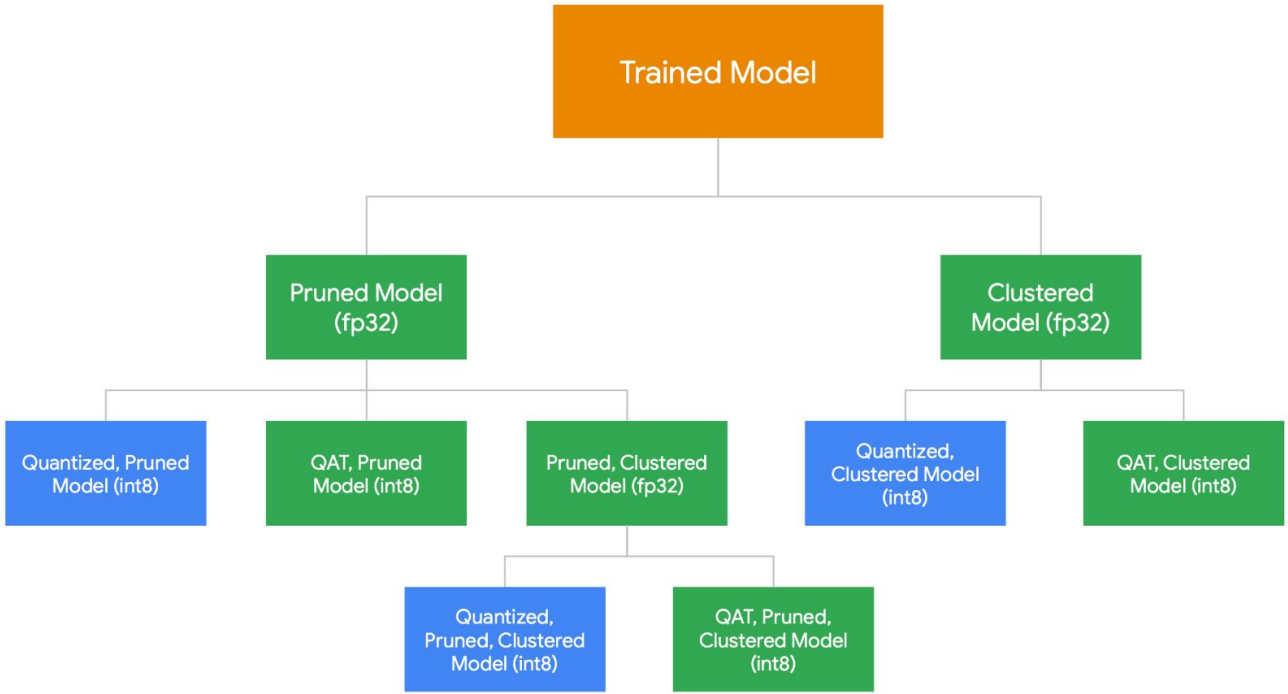

Quantization in Deep Learning
How much to quantize
16bit
8bit
4bit
2bit


Quantization in Deep Learning
When to quantize
Post-training
Quantization-aware training
How much to quantize
16bit
8bit
4bit
2bit


Weight
Activation
What to Quantize
Quantization in Deep Learning
Post-training
When to quantize
Quantization-aware training
How much to quantize
16bit
8bit
4bit
2bit


Dynamic
Static
Weight
Activation
What to Quantize
How to Quantize
Hardware-aware
Quantization in Deep Learning
Post-training
When to quantize
How much to quantize
Mixed-precision
Quantization-aware training
16bit
8bit
4bit
2bit


Dynamic
Static
Weight
Activation
What to Quantize
How to Quantize
Hardware-aware
Quantization in Deep Learning
Post-training
When to quantize
How much to quantize
Mixed-precision
Quantization-aware training
16bit
8bit
4bit
2bit


Trained Model
Pruned Model (fp32)
Clustered Model (fp32)
Quantized, Pruned Model (int8)
QAT, Pruned Model (int8)
Pruned, Clustered Model (fp32)
Quantized, Clustered Model (int8)
QAT, Clustered Model (int8)
Quantized, Pruned, Clustered Model (int8)
QAT, Pruned, Clustered Model (int8)

Paper Discussions

Paper Discussion #1 - A Survey of Quantization Methods for Efficient Neural Network Inference

Amir Gholami*, Sehoon Kim*, Zhen Dong*, Zhewei Yao*, Michael W. Mahoney, Kurt Keutzer
University of California, Berkeley
{amirgh, sehoonkim, zhendong, zheweiy, mahoneymw, keutzer}@berkeley.edu

***Abstract*—As soon as abstract mathematical computations were adapted to computation on digital computers, the problem of efficient representation, manipulation, and communication of the numerical values in those computations arose. Strongly related to the problem of numerical representation is the problem of quantization: in what manner should a set of continuous real-valued numbers be distributed over a fixed discrete set of numbers to minimize the number of bits required and also to maximize the accuracy of the attendant computations? This perennial problem of quantization is particularly relevant whenever memory and/or computational resources are severely restricted, and it has come to the forefront in recent years due to the remarkable performance of Neural Network models in computer vision, natural language processing, and related areas. Moving from floating-point representations to low-precision fixed integer values represented in four bits or less holds the potential to reduce the memory footprint and latency by a factor of 16x; and, in fact, reductions of 4x to 8x are often realized in practice in these applications. Thus, it is not surprising that quantization has emerged recently as an important and very active sub-area of research in the efficient implementation of computations associated with Neural Networks. In this article, we survey approaches to the problem of quantizing the numerical values in deep Neural Network computations, covering the advantages/disadvantages of current methods. With this survey and its organization, we hope to have presented a useful snapshot of the current research in quantization for Neural Networks and to have given an intelligent organization to ease the evaluation of future research in this area.**

means that it is not possible to deploy them for many resource-constrained applications. This creates a problem for realizing pervasive deep learning, which requires real-time inference, with low energy consumption and high accuracy, in resource-constrained environments. This pervasive deep learning is expected to have a significant impact on a wide range of applications such as real-time intelligent healthcare monitoring, autonomous driving, audio analytics, and speech recognition.

Achieving efficient, real-time NNs with optimal accuracy requires rethinking the design, training, and deployment of NN models [71]. There is a large body of literature that has focused on addressing these issues by making NN models more efficient (in terms of latency, memory footprint, and energy consumption, etc.), while still providing optimal accuracy/generalization trade-offs. These efforts can be broadly categorized as follows.

*a) **Designing efficient NN model architectures**:* One line of work has focused on optimizing the NN model architecture in terms of its micro-architecture [101, 111, 127, 167, 168, 212, 253, 280] (e.g., kernel types such as depth-wise convolution or low-rank factorization) as well as its macro-architecture [100, 101, 104, 110, 214, 233] (e.g., module types such as residual, or inception). The classical techniques here mostly found new architecture modules using manual search, which is not scalable. As such, a new line of work is to design Automated machine learning (AutoML) and Neural Architecture Search (NAS) methods. These aim to find in an automated way the right NN architecture, under given constraints of model size

Paper Discussion #2 - The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks

THE LOTTERY TICKET HYPOTHESIS: FINDING SPARSE, TRAINABLE NEURAL NETWORKS

Jonathan Frankle
MIT CSAIL
jfrankle@csail.mit.edu

Michael Carbin
MIT CSAIL
mcarbin@csail.mit.edu

ABSTRACT

Neural network pruning techniques can reduce the parameter counts of trained networks by over 90%, decreasing storage requirements and improving computational performance of inference without compromising accuracy. However, contemporary experience is that the sparse architectures produced by pruning are difficult to train from the start, which would similarly improve training performance.

We find that a standard pruning technique naturally uncovers subnetworks whose initializations made them capable of training effectively. Based on these results, we articulate the *lottery ticket hypothesis*: dense, randomly-initialized, feed-forward networks contain subnetworks (*winning tickets*) that—when trained in isolation—reach test accuracy comparable to the original network in a similar number of iterations. The winning tickets we find have won the initialization lottery: their connections have initial weights that make training particularly effective.

We present an algorithm to identify winning tickets and a series of experiments that support the lottery ticket hypothesis and the importance of these fortuitous initializations. We consistently find winning tickets that are less than 10-20% of the size of several fully-connected and convolutional feed-forward architectures for MNIST and CIFAR10. Above this size, the winning tickets that we find learn faster than the original network and reach higher test accuracy.

Guest Speaker

Song Han

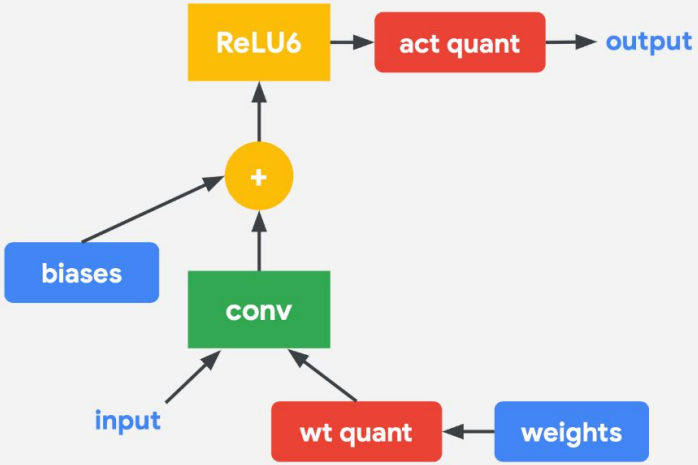
Song Han is an associate professor at MIT EECS. He is one of the pioneers of TinyML research that brings deep learning to IoT devices, enabling “learning on the edge”. Song’s cutting-edge research in efficient AI computing has profoundly influenced the industry. He was the cofounder of DeePhi (now part of AMD), and cofounder of OmniML (now part of NVIDIA).

Personal Website

Google Scholar

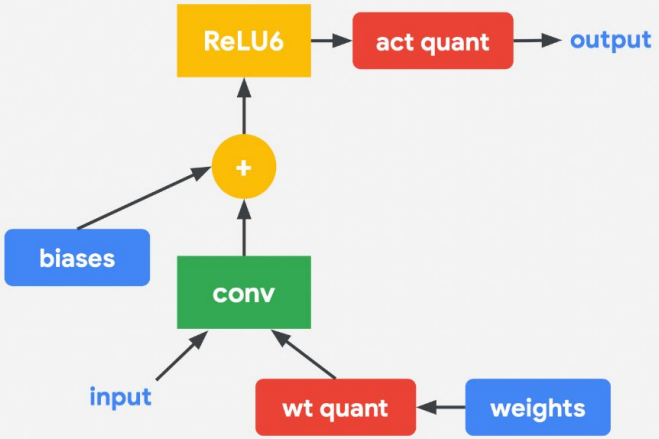
Quantization-aware Training

- **Mimic** the **inference path during** the **training** phase



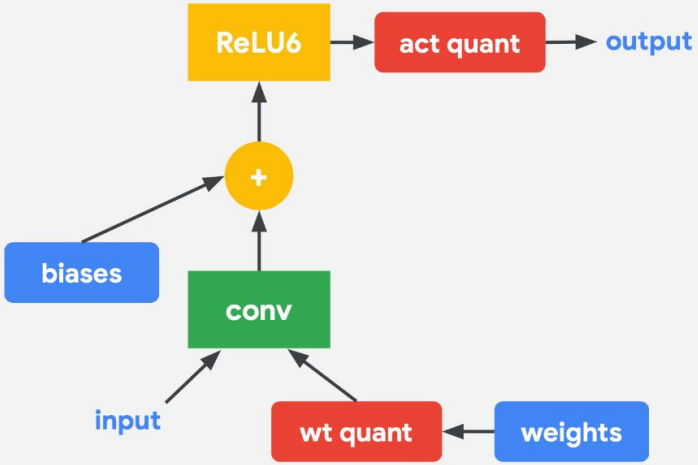
Quantization-aware Training

- Mimic the inference path during the training phase
- Expose the training pipeline to the errors observed



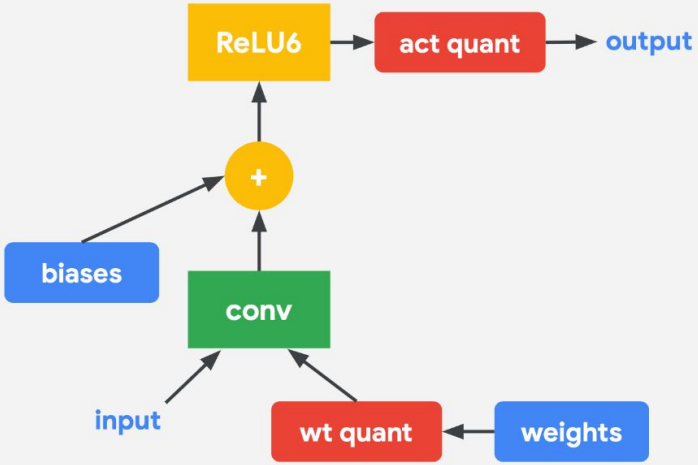
Quantization-aware Training

- **Mimic** the **inference path during** the **training** phase
- **Expose** the **training pipeline to** the **errors** observed
- Allow the **training phase** to **recover** the **error** ***"naturally"***

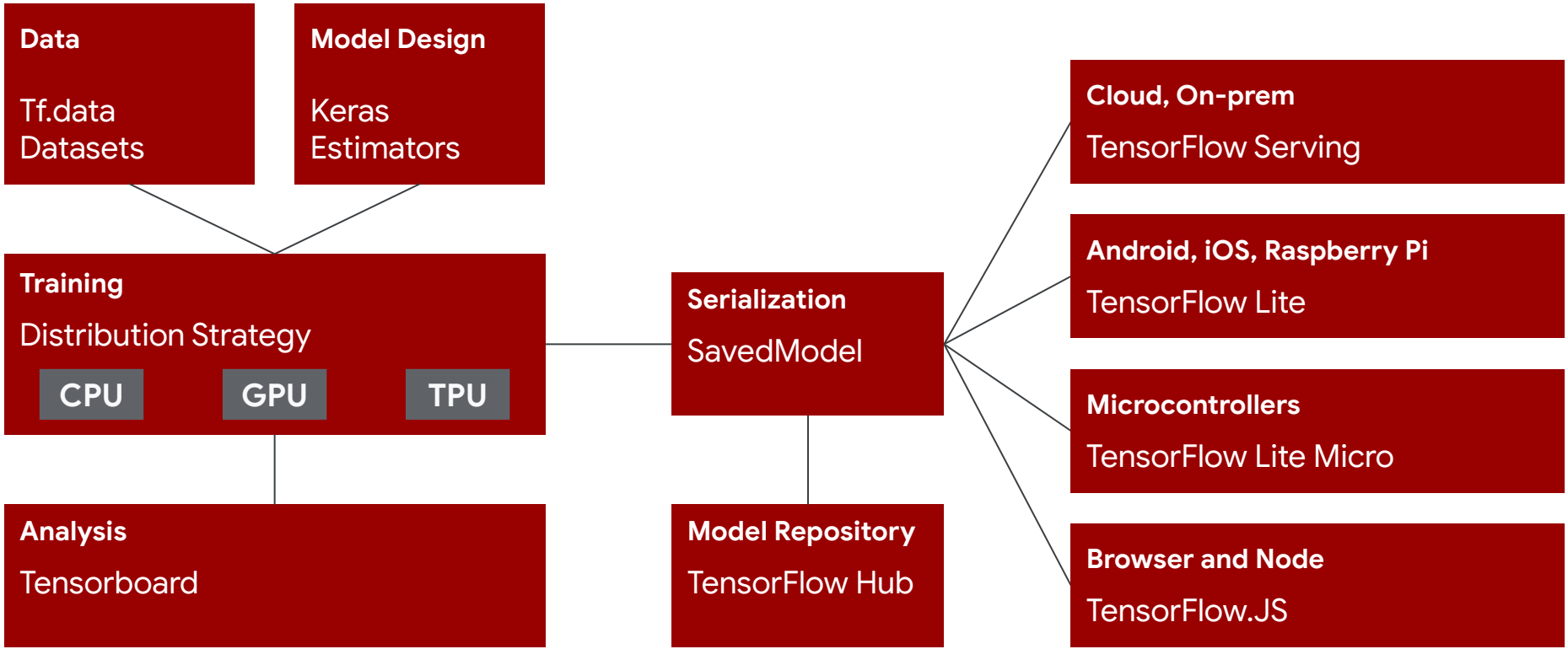


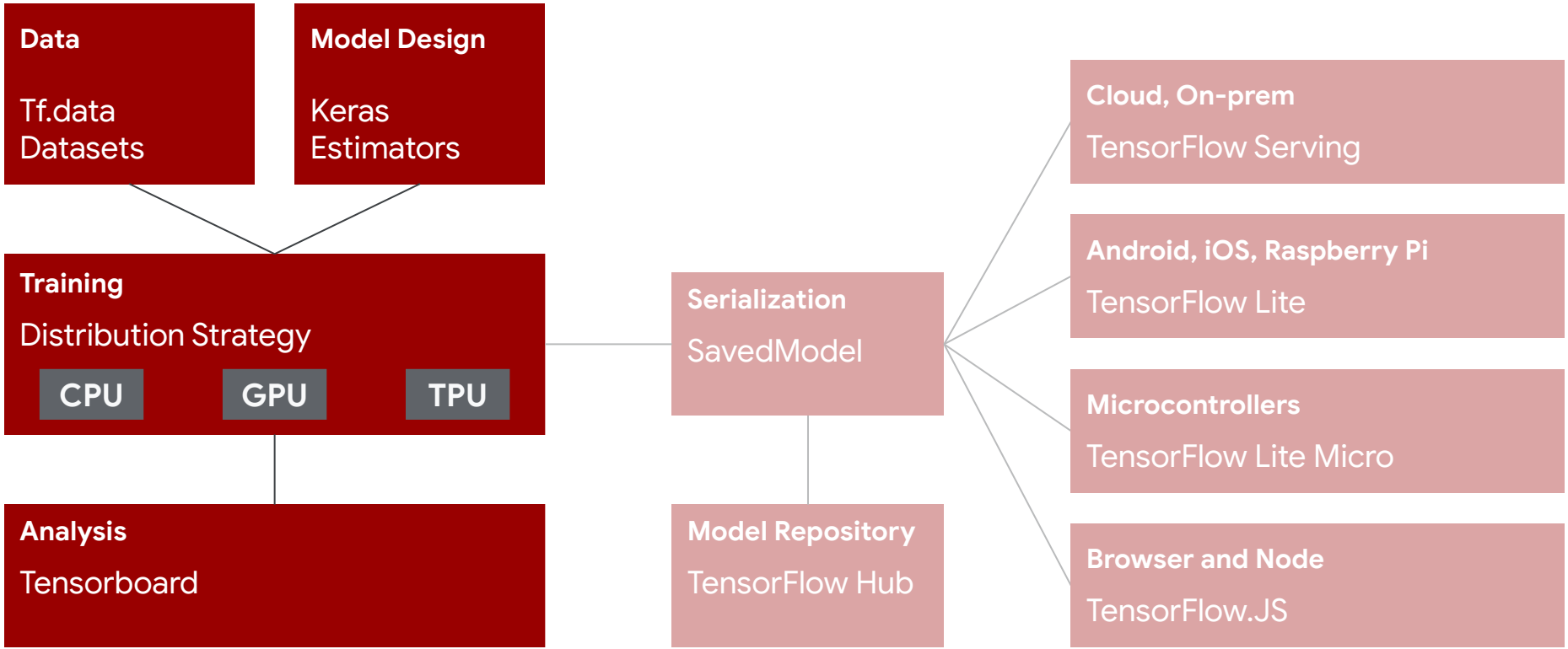
Quantization-aware Training

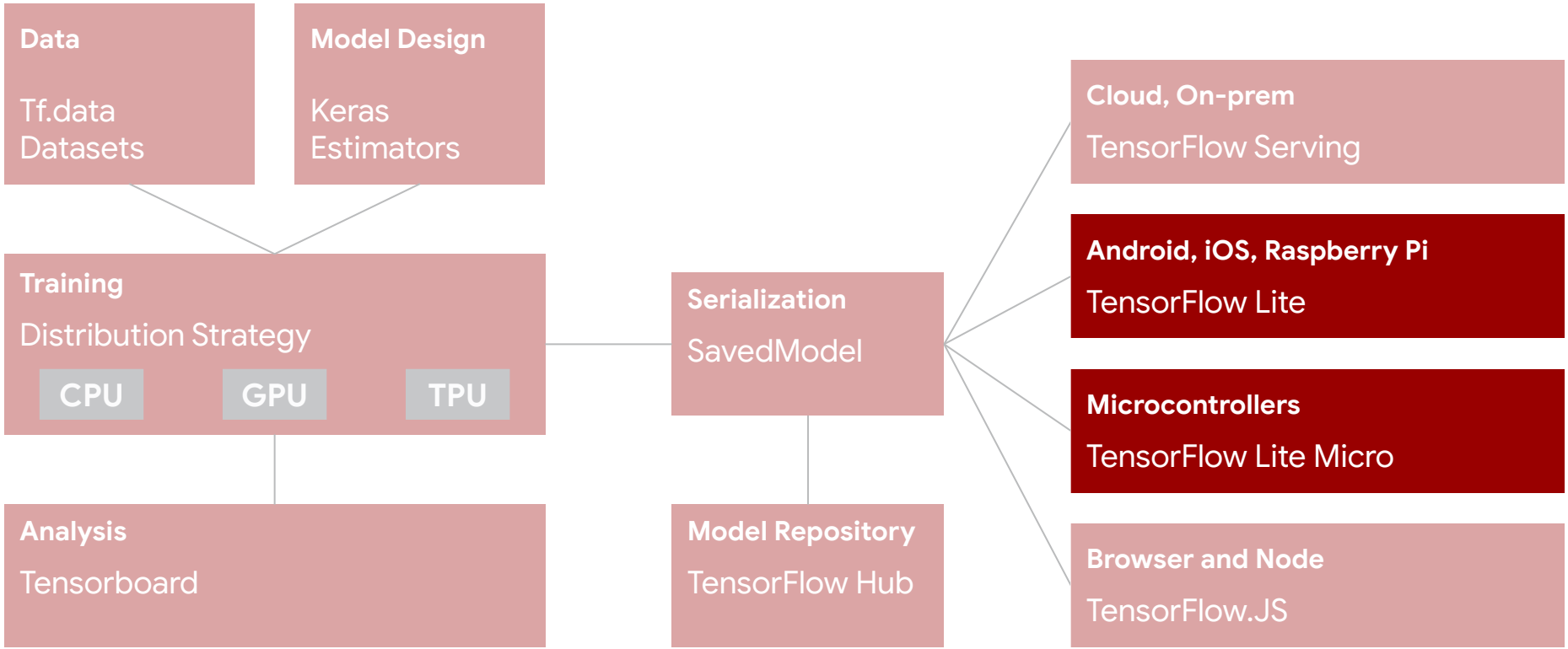
- **Mimic** the **inference path during** the **training** phase
- **Expose** the **training pipeline to** the **errors** observed
- Allow the **training phase** to **recover** the **error** ***"naturally"***
- **Weights** *and* **inputs** use the same int8—mimic int8 MAC



	Floating-point Baseline	Post-training Quantization (PTQ)	Quantization-Aware Training (QAT)
MobileNet v1 1.0 224	71.03%	69.57%	71.06%
MobileNet v2 1.0 224	70.77%	70.20%	70.01%
Resnet v1 50	76.30%	75.95%	76.10%

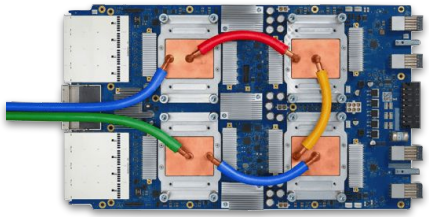

Data
Tf.data
Datasets
Model Design
Keras
Estimators
Training
Distribution Strategy
CPU
GPU
TPU
Analysis
Tensorboard
Serialization
SavedModel
Model Repository
TensorFlow Hub
Cloud, On-prem
TensorFlow Serving
Android, iOS, Raspberry Pi
TensorFlow Lite
Microcontrollers
TensorFlow Lite Micro
Browser and Node
TensorFlow.JS


Data
Tf.data
Datasets
Model Design
Keras
Estimators
Training
Distribution Strategy
CPU
GPU
TPU
Analysis
Tensorboard
Serialization
SavedModel
Model Repository
TensorFlow Hub
Cloud, On-prem
TensorFlow Serving
Android, iOS, Raspberry Pi
TensorFlow Lite
Microcontrollers
TensorFlow Lite Micro
Browser and Node
TensorFlow.JS


Data
Tf.data
Datasets
Model Design
Keras
Estimators
Training
Distribution Strategy
CPU
GPU
TPU
Analysis
Tensorboard
Serialization
SavedModel
Model Repository
TensorFlow Hub
Cloud, On-prem
TensorFlow Serving
Android, iOS, Raspberry Pi
TensorFlow Lite
Microcontrollers
TensorFlow Lite Micro
Browser and Node
TensorFlow.JS

Framework Differences

TensorFlow



TensorFlow Lite

TensorFlow Lite Micro

Model	TensorFlow	TensorFlow Lite	TensorFlow Lite Micro
Training	Yes	No	No
Inference	Yes *(but inefficient on edge)*	Yes *(and efficient)*	Yes *(and even **more** efficient)*
How Many Ops	~1400	~130	~50
Native Quantization Tooling + Support	No	Yes	Yes