Model Compression and Optimization for Efficient Inference

Romina Soledad Molina, Ph.D.

School on Applied AI for Sustainable Development | smr 4210 | Trieste, Italy
March 2026

KaleidoForge
Lightning Bridges for Fast AI

Outline

- Model compression techniques in machine learning
- Pruning
- Quantization
- Knowledge distillation
- How to Combine Compression Techniques
- Choosing the Right Compression Strategy

Model compression techniques in machine learning

Model compression techniques in machine learning

Compression is the process of reducing the size of data, often with the aim of preserving important information or, in some cases, enabling perfect reconstruction.

The goal is to make data storage or transmission more efficient by using fewer resources (e.g., memory, storage, bandwidth).

Lossy compression

Model compression techniques in machine learning

- Reduces model size and computational cost
- Enables deployment under hardware constraints
- Targets efficient inference
- Balances accuracy and efficiency

Model compression aims to reduce the size and complexity of machine learning models in order to enable efficient inference under real-world constraints.

Model compression techniques in machine learning

Model and Data Compression

Compression in Machine Learning aims to reduce resource requirements while maintaining performance.

Model compression: Smaller and more efficient models

Data compression: Reduced dataset size with minimal information loss

Model compression techniques in machine learning

Applicability

Not all compression techniques are suitable for every type of model or dataset.

The decision of which compression strategy to apply depends on factors such as the desired trade-off between model size and performance, the computational resources available, and the nature of the data being processed.

Model compression techniques in machine learning

Improving efficiency often introduces a performance-efficiency tradeoff.

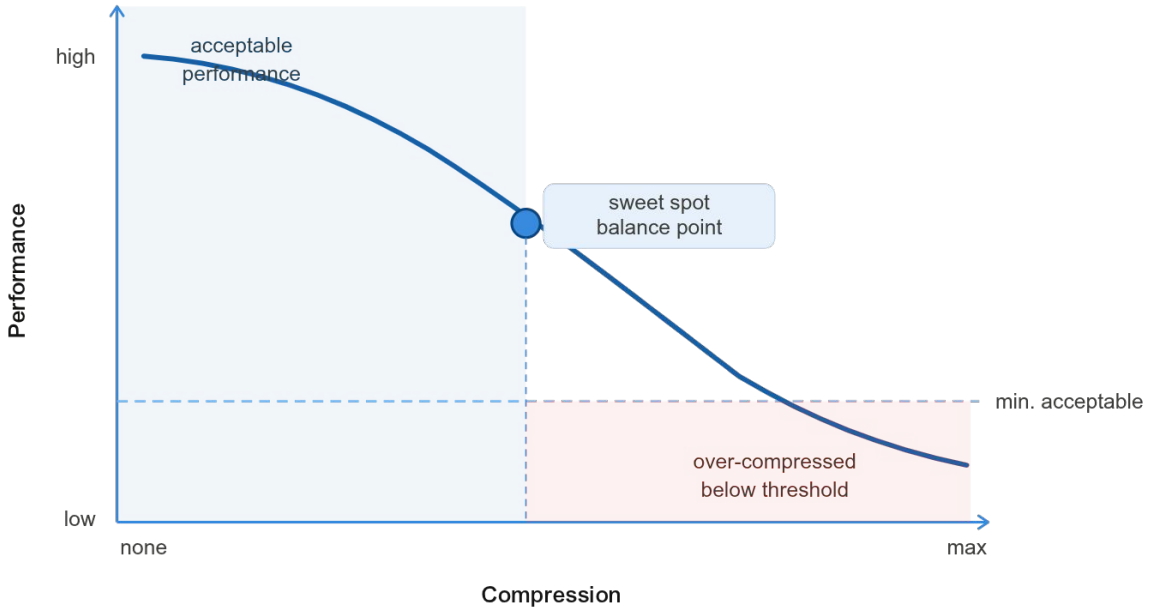
Model compression aims to find the best balance between the two.

What does efficiency mean?

Lower latency	Less memory	Lower power	Smaller model size
faster responses	*reduced footprint*	*energy consumption*	*Runs at the edge*

Model compression techniques in machine learning

Compression vs Performance



Model compression techniques in machine learning

Why Compress?

Memory

4–8×

smaller model footprint

Latency

2–5×

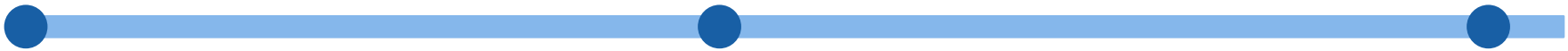
faster inference

Cost

↓↓

on-device inference

Train (cloud) → Compress → Deploy (edge)



Pruning

Pruning

- This technique reduces the size and complexity of deep learning models by removing redundant or unnecessary weights and neurons.

- Its primary goal is to **improve efficiency** by lowering memory requirements and accelerating inference, while preserving model performance.

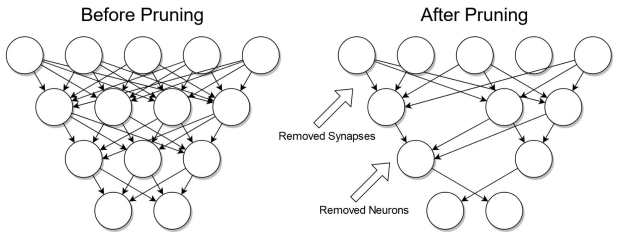
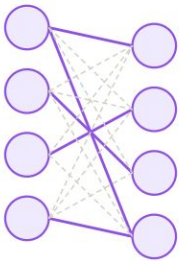


Image from https://en.wikipedia.org/wiki/Decision_tree_pruning#:~:text=Pruning%20is%20a%20data%20compression,and%20redundant%20to%20classify%20instances.

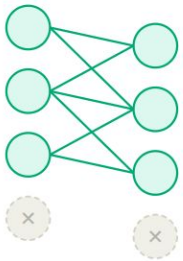
Pruning

Unstructured
weight pruning



— — = w = 0
—— = active weight

Structured
neuron pruning

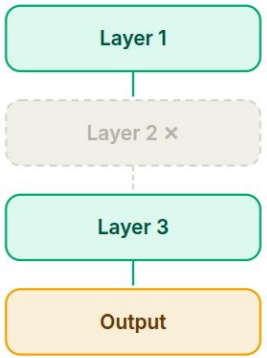


Entire neurons removed
Params ↓ · Dense weights

Structured
filter pruning (CNN)

Entire filters removed
Fewer output channels

Structured
layer pruning



Entire layer removed
Most aggressive · depth ↓

Summary

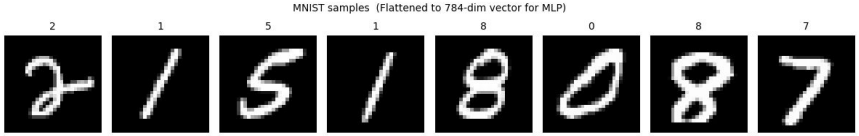
Unstructured	Neuron	Filter (CNN)	Layer
Sparsity ↑ · Params =	Params ↓ · Dense	Channels ↓ · Dense	Depth ↓ · Most aggressive
Needs sparse HW	Works on any HW	Works on any HW	Higher accuracy loss

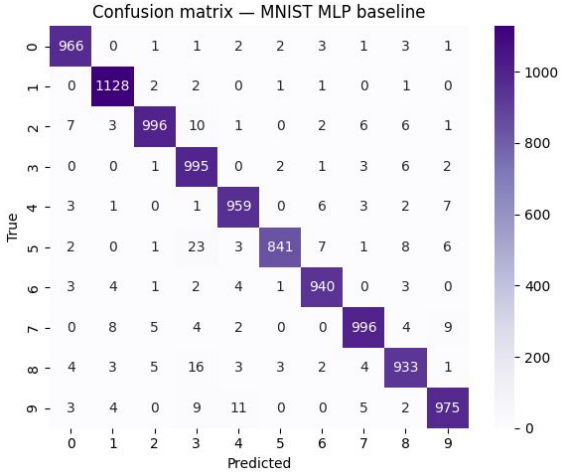
Pruning

Unstructured pruning creates sparse matrices that reduce parameter count but require additional indexing, while **structured pruning** changes the model architecture, resulting in smaller dense models that are much easier to accelerate on hardware.

Pruning

Results - MNIST MLP

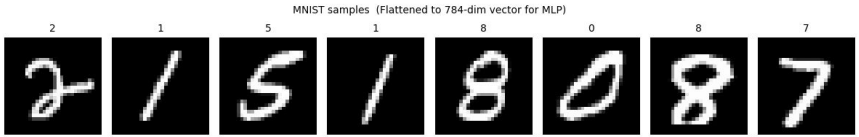


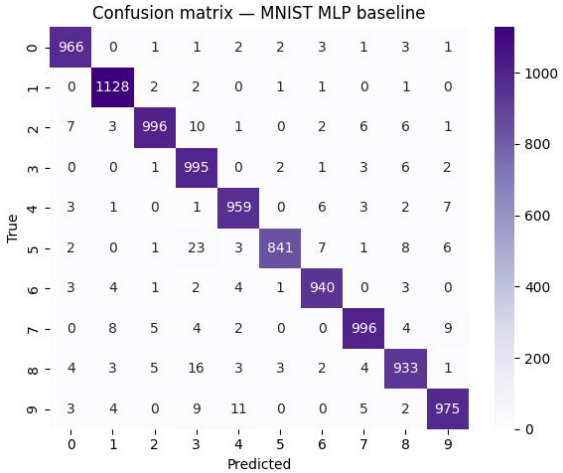


Accuracy: 0.9709

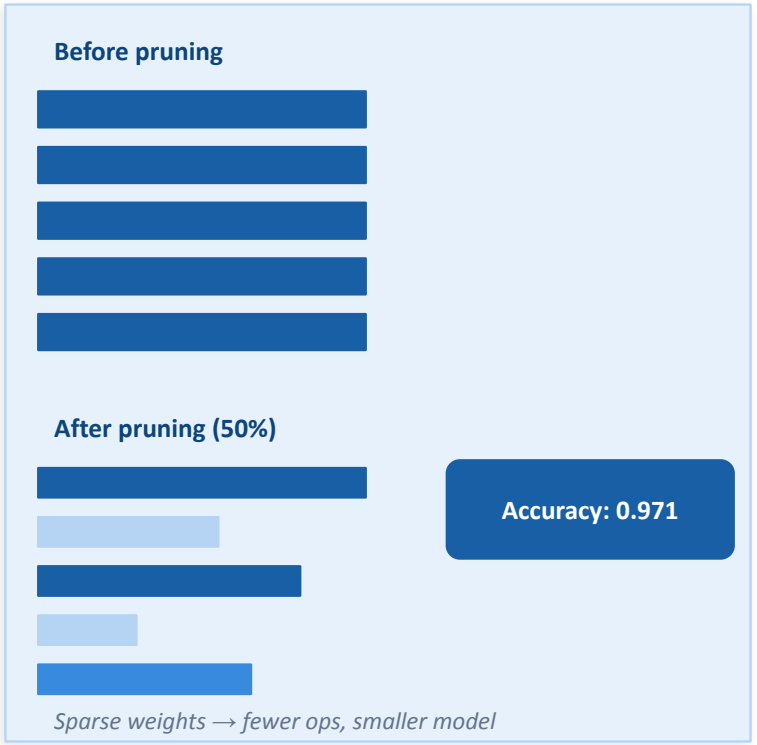
Pruning

Results - MNIST MLP





Accuracy: 0.9709

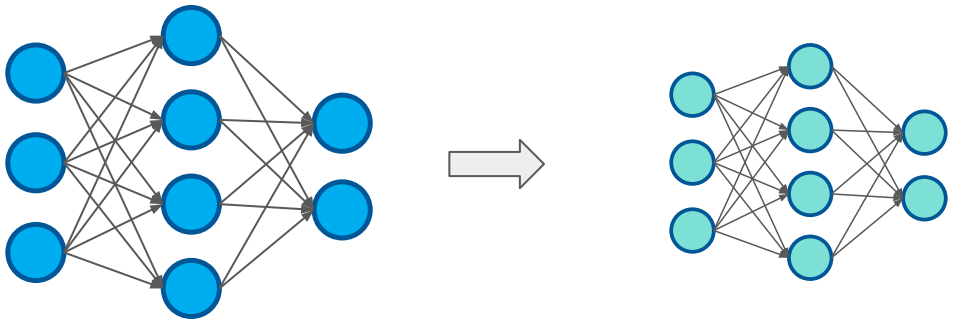


Quantization

Quantization

Quantization is a technique that reduces the numerical precision of a neural network's parameters by converting floating-point values (e.g., 32-bit) into lower-precision representations, such as 16-bit or 8-bit.

The main goal is to reduce model size and accelerate inference, especially on resource-constrained devices.



Quantization Variants

PTQ	**Post-Training Quantization**	Quantize after training is complete. Fast and simple, no retraining needed. Small accuracy drop.
QAT	**Quantization-Aware Training**	Simulate quantization during training so the model adapts. Best accuracy retention.
DQ	**Dynamic Quantization**	Weights quantized statically; activations quantized at runtime. Good for NLP / RNNs.
FIQ	**Full Integer Quantization**	Both weights and activations in integer. Required for microcontrollers and edge TPUs.
QAP	**Quantization-Aware Pruning**	Combines pruning and quantization in one training pass. Maximum compression.

Quantization Variants

Post-Training Quantization

Post-Training Quantization (PTQ) is a quantization technique applied to a trained model.
It converts weights and activations from floating-point precision to lower-precision formats.

PTQ is typically used when the goal is to reduce model size and improve inference efficiency **without retraining the model**.

PTQ is simple and fast to apply, but may introduce some accuracy degradation compared to training-aware approaches.

Quantization Variants

Dynamic Quantization

Dynamic Quantization (DQ) is a quantization method that applies **quantization only to the model's weights**, while activations remain in floating-point format and are quantized dynamically at runtime.

This approach is particularly useful when fast inference is prioritized over maximum model size reduction, as it requires no retraining and introduces minimal accuracy impact.

Dynamic quantization is a lightweight alternative to PTQ and QAT, trading maximum compression for simplicity and speed.

Quantization Variants

Full Integer Quantization

Full Integer Quantization (FIQ) is a quantization technique in which both **weights and activations are fully converted to integer representations** (e.g., INT8).

This approach is widely used on specialized hardware platforms, such as TPUs, NPUs, and microcontrollers, to maximize inference efficiency, reduce power consumption, and improve performance.

Full integer quantization delivers the highest efficiency, but requires careful calibration or quantization-aware training.

Quantization Variants

Quantization-Aware Training

Quantization-aware training (QAT) is a training technique in which the model learns to adapt to quantization effects before deployment on hardware.

Instead of training the model entirely in full precision (e.g., 32-bit floating point) and quantizing it afterward, **quantization is incorporated directly during training**.

QAT typically achieves better accuracy than PTQ, but at the cost of increased training complexity and computational effort.

QAT shifts part of the quantization cost from deployment to training in order to preserve accuracy.

Quantization Variants

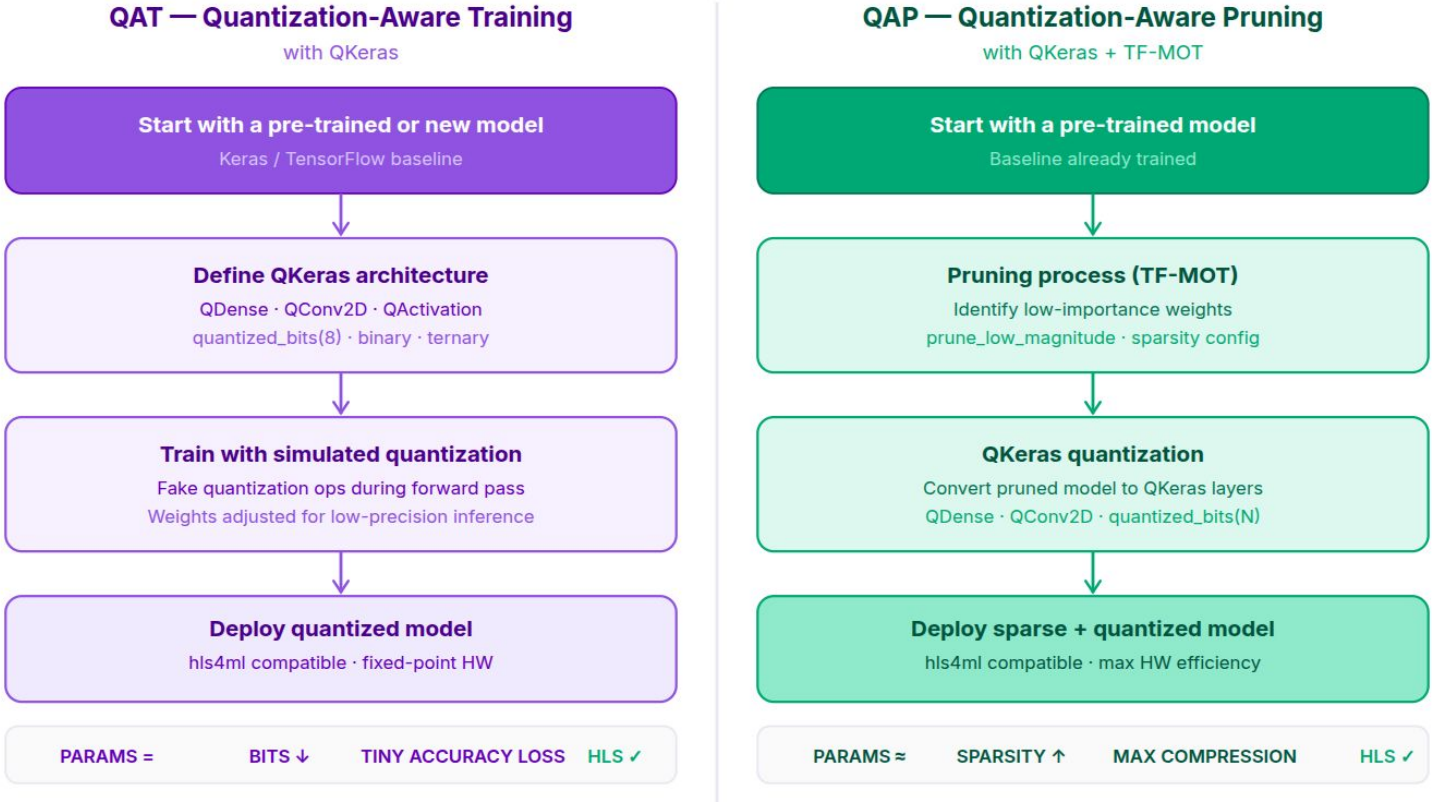
Quantization-Aware Pruning

Quantization-Aware Pruning (QAP) combines pruning with quantization-aware training.
The goal is to jointly optimize sparsity and reduced numerical precision, producing a smaller and more efficient model after quantization, while preserving accuracy.

QAP aligns pruning decisions with quantization effects, ensuring that sparsity remains effective after quantization.

Quantization Variants

QAT vs QAP



Quantization Variants

QAP vs Sequantial Pruning + Quantization

Sequential approach

- Pruning → Quantization
- Pruning is applied in full precision
- Quantization is applied afterward
- Sparsity patterns may be distorted by quantization
- Accuracy degradation can increase after quantization
- Simple, but not robust for low-bit inference.

Quantization-Aware Pruning (QAP)

- Pruning + Quantization-Aware Training (jointly)
- Pruning decisions are made while accounting for quantization effects
- The model learns: which weights to remove and which remaining weights can tolerate low precision.
- Sparsity is preserved after quantization
- More complex training, but better final accuracy.

Quantization Variants

QKeras

Extension of Keras designed to quantize neural network models

Useful when training models with lower precision.

Auto QKeras.

Repository: https://github.com/google/qkeras

Quantization Variants

QKeras

Automatic heterogeneous quantization of deep neural networks for low-latency inference on the edge for particle detectors

Claudionor N. Coelho Jr.
Palo Alto Networks (California, USA)

Aki Kuusela, Shan Li, and Hao Zhuang
Google LLC (California, USA)

Thea Aarrestad,* Vladimir Loncar,† Maurizio Pierini, Adrian Alan Pol, and Sioni Summers
European Organization for Nuclear Research (CERN) (Geneva, Switzerland)

Jennifer Ngadiuba
California Institute of Technology (Caltech) (California,USA)
(Dated: June 22, 2021)

Although the quest for more accurate solutions is pushing deep learning research towards larger and more complex algorithms, edge devices demand efficient inference and therefore reduction in model size, latency and energy consumption. One technique to limit model size is quantization, which implies using fewer bits to represent weights and biases. Such an approach usually results in a decline in performance. Here, we introduce a method for designing optimally heterogeneously quantized versions of deep neural network models for minimum-energy, high-accuracy, nanosecond inference and fully automated deployment on chip. With a per-layer, per-parameter type automatic quantization procedure, sampling from a wide range of quantizers, model energy consumption and size are minimized while high accuracy is maintained. This is crucial for the event selection procedure in proton–proton collisions at the CERN Large Hadron Collider, where resources are strictly limited and a latency of $\mathcal{O}(1)$ μs is required. Nanosecond inference and a resource consumption reduced by a factor of 50 when implemented on field-programmable gate array hardware are achieved.

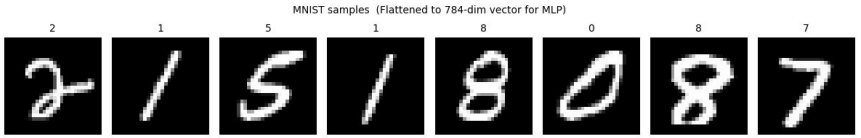
Quantization Variants

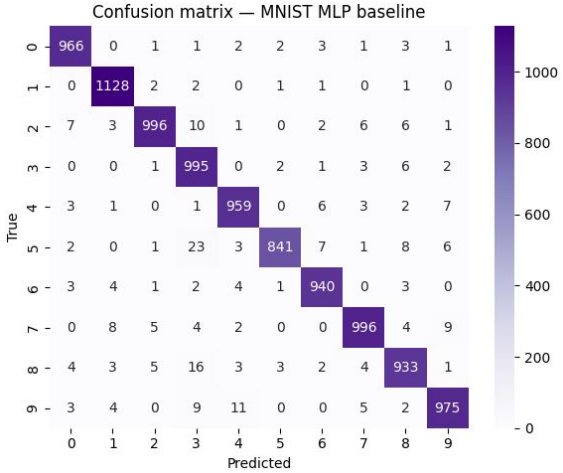
QKeras

```
# MLP architecture
# Create the student QKERAS
studentQ_MLP = keras.Sequential(
    [
        Input(shape=(30,)),
        QDense(20, name='fc1',
                kernel_quantizer=quantized_bits(9,1,alpha=1), bias_quantizer=quantized_bits(23,15,alpha=1)),
        QActivation(activation=quantized_relu(16,15), name='relu1'),
        QDense(10, name='fc2',
                kernel_quantizer=quantized_bits(9,1,alpha=1), bias_quantizer=quantized_bits(23,15,alpha=1)),
        QActivation(activation=quantized_relu(16,15), name='relu2'),
        QDense(10, name='fc6',
                kernel_quantizer=quantized_bits(9,1,alpha=1), bias_quantizer=quantized_bits(23,15,alpha=1)),
        QActivation(activation=quantized_relu(16,15), name='relu3'),

        QDense(4, name='output',
                kernel_quantizer=quantized_bits(32,15,alpha=1), bias_quantizer=quantized_bits(32,15,alpha=1)),
        Activation(activation='softmax', name='softmax')


    ],
    name="student",
)

print_qstats(studentQ_MLP)
```

Quantization

Results - MNIST MLP





Accuracy: 0.9709

Memory footprint by precision

float32

32 bits

int8

8 bits · 4×

int4

4 bits · 8×

8-bit acc: 0.969

Accuracy stays near baseline down to int8 / int4 (QKeras results)

Knowledge Distillation

Knowledge Distillation

Distilling the Knowledge in a Neural Network

Geoffrey Hinton[*†]
Google Inc.
Mountain View
geoffhinton@google.com

Oriol Vinyals[†]
Google Inc.
Mountain View
vinyals@google.com

Jeff Dean
Google Inc.
Mountain View
jeff@google.com

Abstract

A very simple way to improve the performance of almost any machine learning algorithm is to train many different models on the same data and then to average their predictions [3]. Unfortunately, making predictions using a whole ensemble of models is cumbersome and may be too computationally expensive to allow deployment to a large number of users, especially if the individual models are large neural nets. Caruana and his collaborators [1] have shown that it is possible to compress the knowledge in an ensemble into a single model which is much easier to deploy and we develop this approach further using a different compression technique. We achieve some surprising results on MNIST and we show that we can significantly improve the acoustic model of a heavily used commercial system by distilling the knowledge in an ensemble of models into a single model. We also introduce a new type of ensemble composed of one or more full models and many specialist models which learn to distinguish fine-grained classes that the full models confuse. Unlike a mixture of experts, these specialist models can be trained rapidly and in parallel.

Knowledge Distillation

Knowledge Distillation (KD) is a technique that transfers knowledge from a large, high-capacity teacher network to a smaller and faster student (or distilled) network.

The student network is trained to reproduce the behavior of the teacher while being computationally less expensive.

Knowledge distillation allows smaller models to inherit the behavior of larger ones, making it especially attractive for efficient deployment.

Knowledge Distillation

Offline Knowledge Distillation

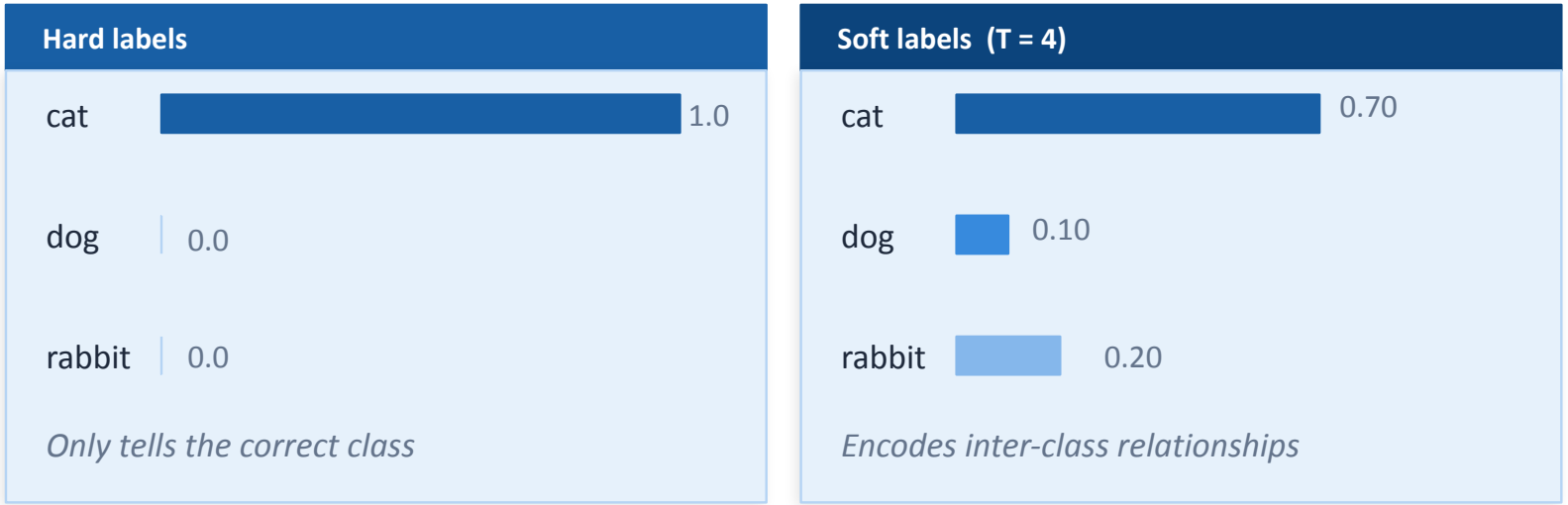
Offline distillation uses a pre-trained teacher model to guide the training of a smaller student model. It is the most common form of KD and does not require joint training of teacher and student.

Advantage: Simple to apply and effective at improving student accuracy.
Challenge: Requires training and storing a large teacher model.

Knowledge Distillation

Dark knowledge

Hinton et al., 2015



Dark knowledge

The information encoded in the relative probabilities of incorrect classes.

They capture inter-class similarity: richer signal than a one-hot vector.

Knowledge Distillation

Temperature (T)

Controls how soft the teacher's output probabilities are.

T = 1 (sharp)

This is 100% a cat. Nothing else.

Student only learns the label, no finer-grained distinctions.

T = 5 (soft)

70% cat, 20% rabbit, 10% dog.

Student learns inter-class similarity, the structure of knowledge.

Higher T → softer probabilities → more dark knowledge transferred.

Alpha (α)

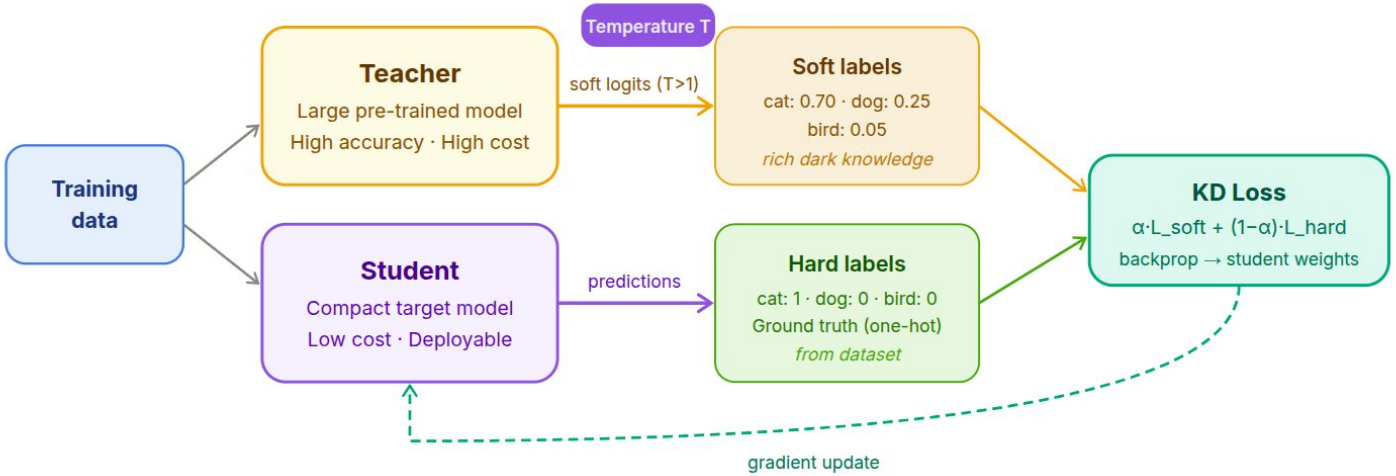
Balances the two loss terms during training.

$$\text{Loss} = \alpha \cdot L_{distill} + (1-\alpha) \cdot L_{CE}$$

α = 0.9	Listens mostly to the teacher. Relies on soft labels.
α = 0.5	Equal weight to teacher and ground truth.
α = 0.1	Relies mostly on true labels. Teacher is a weak guide.

Typical values: α = 0.7–0.9, T = 3–5.

Knowledge Distillation



Teacher model
Large, accurate, expensive. Already trained: weights are frozen during distillation.

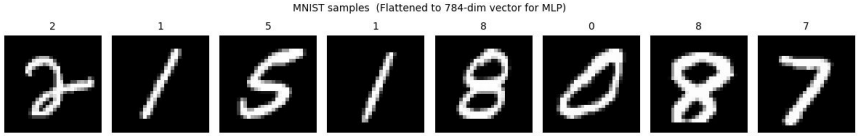
Student model
Compact and deployable. Learns from both hard labels and the teacher's soft outputs.

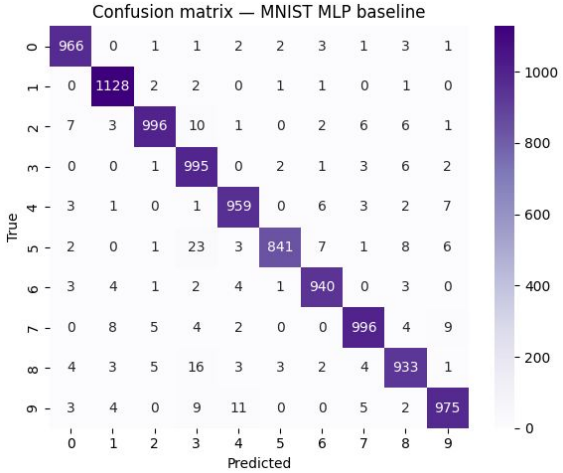
Soft labels
Teacher probabilities at T>1. Carry inter-class similarity (the "dark knowledge").

KD Loss
α·KL(soft) + (1−α)·CE(hard). α balances teacher guidance vs. ground truth.

Knowledge Distillation

Results - MNIST MLP





Accuracy: 0.9709

Teacher (64-32-10)

soft labels

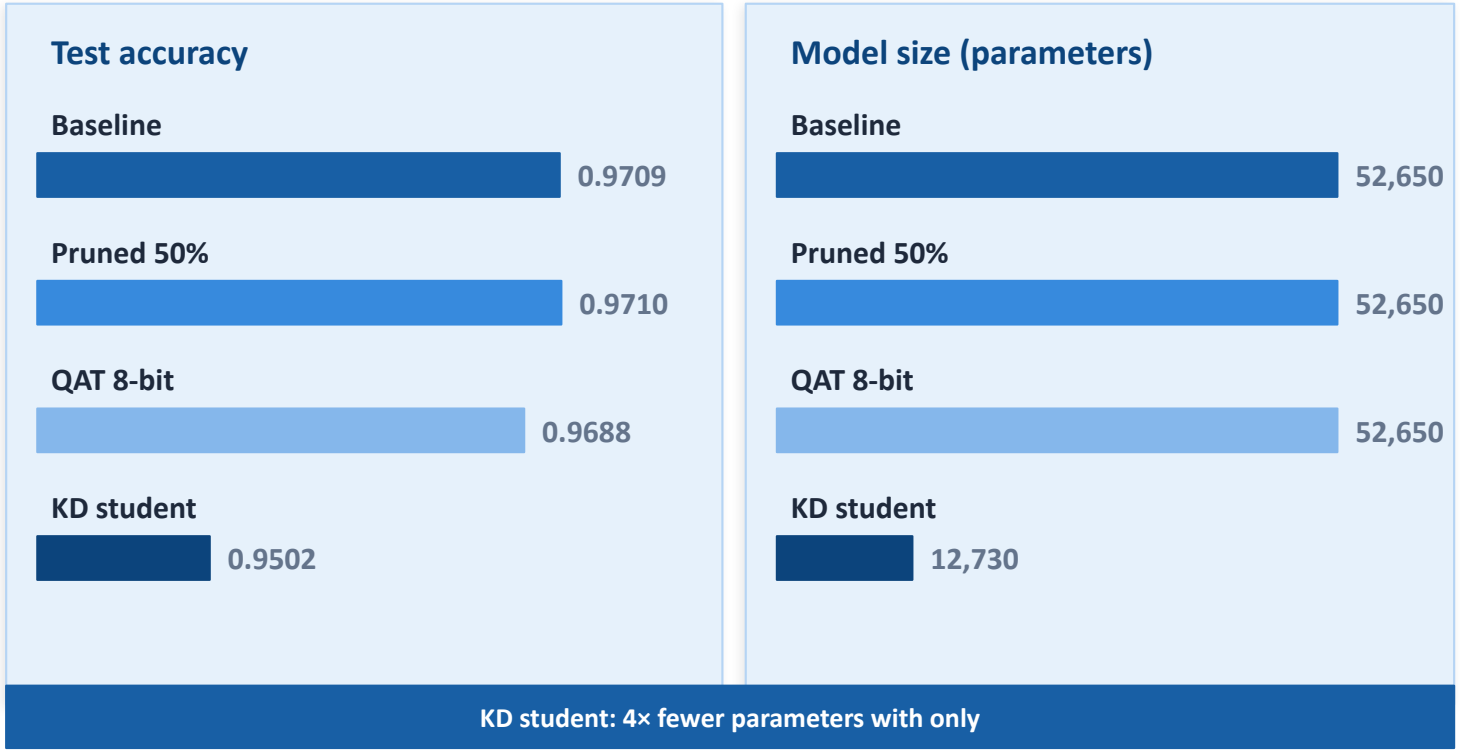
Student (16-10)

52,650 → 12,730 params

Student acc: 0.950

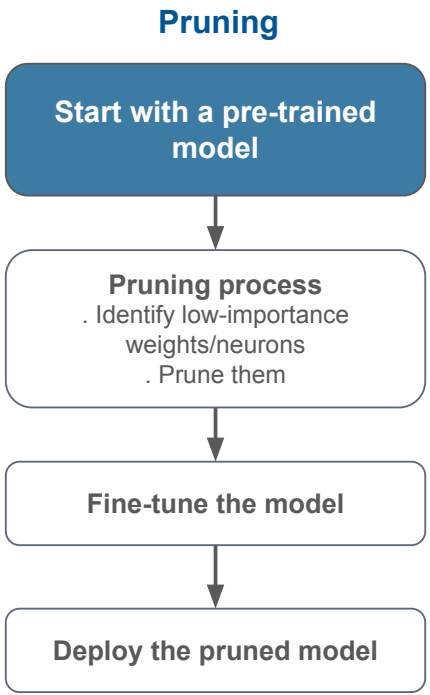
Knowledge Distillation

Results - MNIST MLP



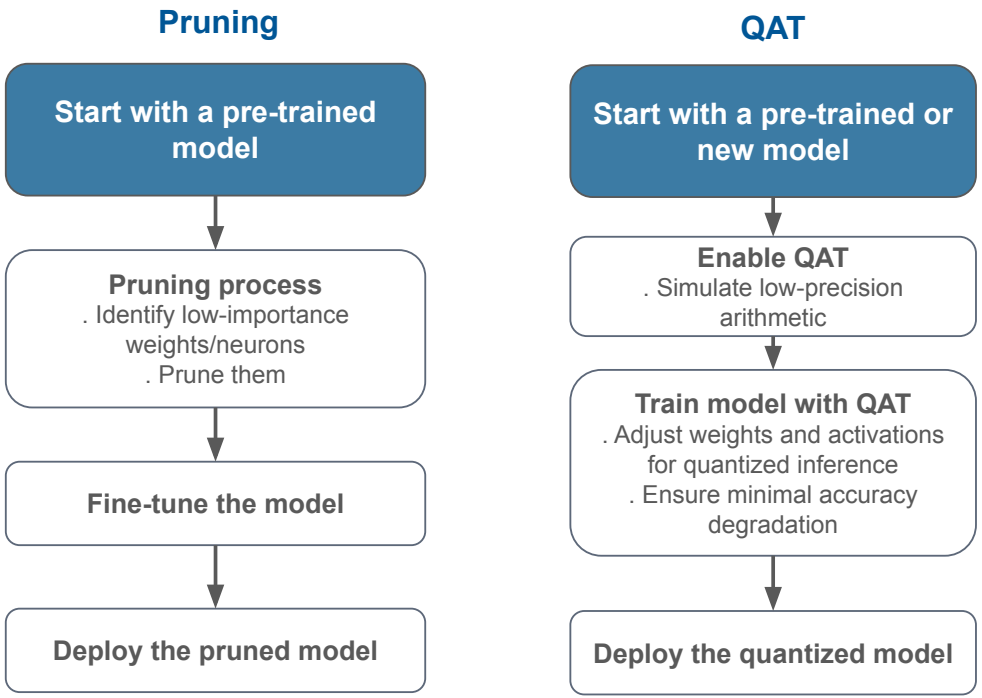
How to Combine Compression Techniques

Some Examples..



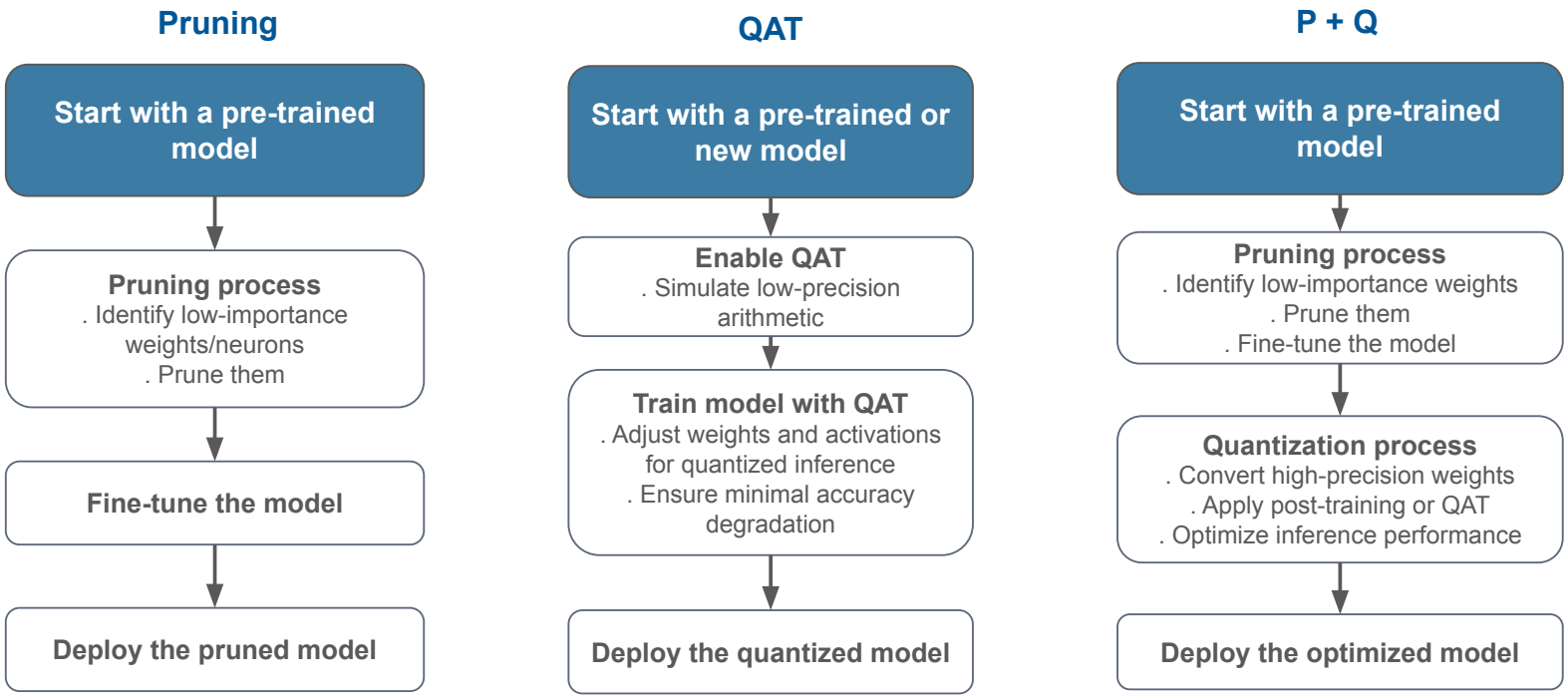
How to Combine Compression Techniques

Some Examples..



How to Combine Compression Techniques

Some Examples..



Choosing the Right Compression Strategy

Model compression is not a single technique,
but a design space where accuracy, efficiency, and hardware constraints must be balanced.

Every optimization we've discussed (pruning, quantization, or distillation) improves efficiency at the cost of something else. The goal is not to avoid trade-offs, but to choose the right ones.

Demo:
MNIST-based binary classification
Traing large → Compress → Deploy Small

Model Compression and Optimization for Efficient Inference

Romina Soledad Molina, Ph.D.

School on Applied AI for Sustainable Development | smr 4210 | Trieste, Italy
March 2026

KaleidoForge
Lightning Bridges for Fast AI

Choosing the Right Compression Strategy

Key factors to consider

- Accuracy requirements
  How much performance loss is acceptable?
- Inference constraints
  Latency, throughput, and real-time requirements
- Hardware target
  CPU, GPU, FPGA, edge device, accelerator
- Resource limits
  Memory footprint and power consumption
- Training budget
  Availability of data, time, and compute for retraining

Choosing the Right Compression Strategy

Key factors to consider

- Accuracy requirements
  How much performance loss is acceptable?
- Inference constraints
  Latency, throughput, and real-time requirements
- Hardware target
  CPU, GPU, FPGA, edge device, accelerator
- Resource limits
  Memory footprint and power consumption
- Training budget
  Availability of data, time, and compute for retraining

Practical guidelines

- Pruning
  Reduce model complexity and computation
- Quantization
  Improve inference efficiency and hardware utilization
- Knowledge Distillation
  Transfer performance to smaller models
- Combined strategies
  Achieve the best trade-off for deployment