Embedded Machine Learning

02226 Networked Embedded Systems

Riccardo Miccini

Technical University of Denmark

November 13, 2025

Before we start

Structure

- **ML primer**: what is it and what does it do
- **Wetware I**: human needs
- **Hardware**: from cloud to edge
- **Software**: from data to deployment
- **Wetware II**: human responsibilities

This slide deck is **dense**; we won't cover everything but you can use it to dig deeper if you want

Also: some of the references are missing, sorry! I ~~will definitely~~ might update them later on

Learning Objectives

- **Explain** the motivation for performing ML on embedded devices
- **Compare** HW platforms for embedded ML in terms of their capabilities and constraints
- **Describe** model compression techniques and deployment steps
- **Evaluate** embedded ML systems with respect to efficiency, scalability, and societal implications
- **Critically assess** emerging trends and **propose** directions for responsible embedded AI development

Outline

1. Quick introduction to ML

2. Wetware I
   - Motivation and applications

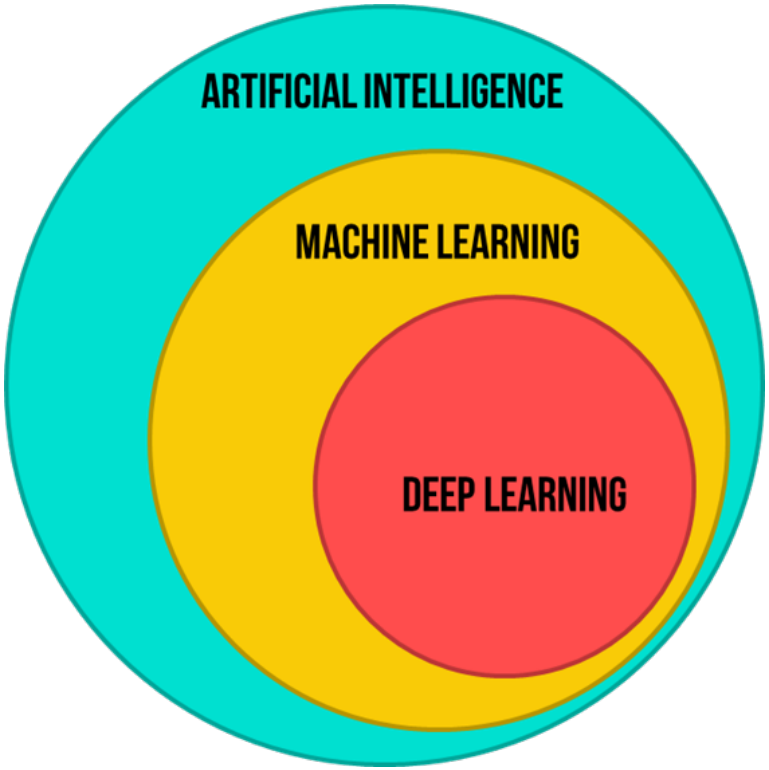
3. Hardware
   - The computing continuum
   - Overview of hardware platforms
   - Neural network acceleration
   - Latest and niche topics

4. Software
   - The embedded ML pipeline
   - Model compression
   - Deployment tools
   - Latest and niche topics

5. Wetware II
   - Responsible embedded ML

Quick introduction to ML

What is even ML/AI/DL?



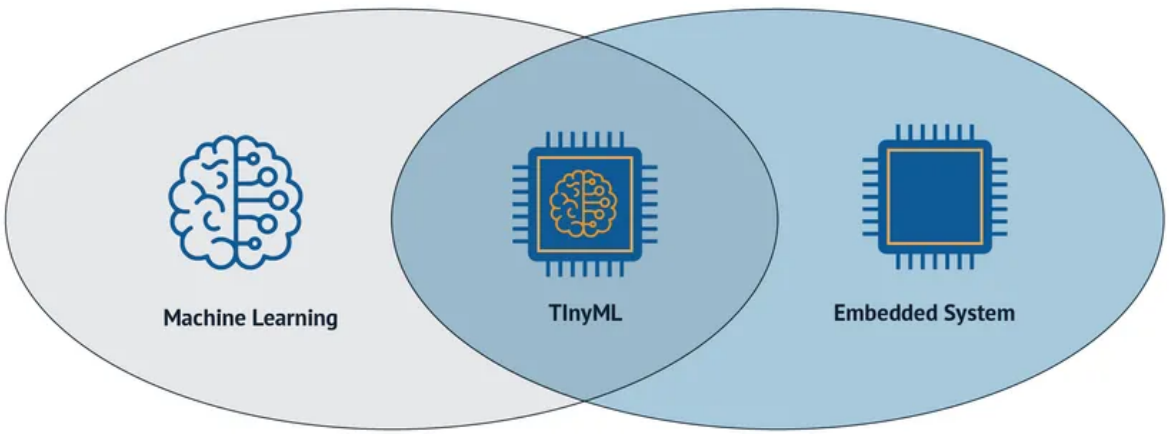
Some terminology

- **Artificial Intelligence**: attempting to perform complex tasks by mimicking human intelligence
- **Machine Learning**: subfield of AI focusing on learning from data rather than explicit rules
- **Deep Learning**: subset of ML techniques based on *deep neural networks* → contemporary AI is mostly deep learning

We will use the terms interchangeably but focus mostly on DL

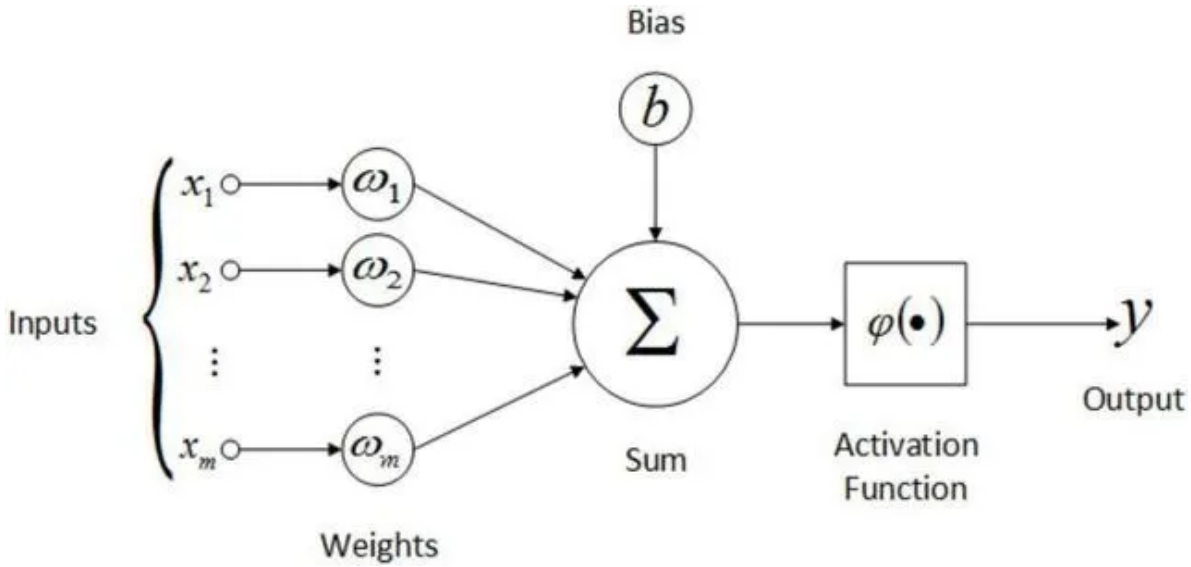
Other buzzwords I should know?

- **Edge AI**: running ML models directly on edge devices (PCs, smartphones, consumer electronics, IoT) rather than in the cloud
- **TinyML**: subfield of ML/DL dealing with the challenge of running model on *low-power, resource-constrained devices* ← our focus
- Less common aliases: AIoT, edge computing, embedded ML, resource-constrained ML, resource-efficient ML...



The artificial neuron

- Compute linear combination (weighted sum) of inputs
- Activation: some non-linear operation (e.g., s-shaped function)
- The weights represent how important each input is

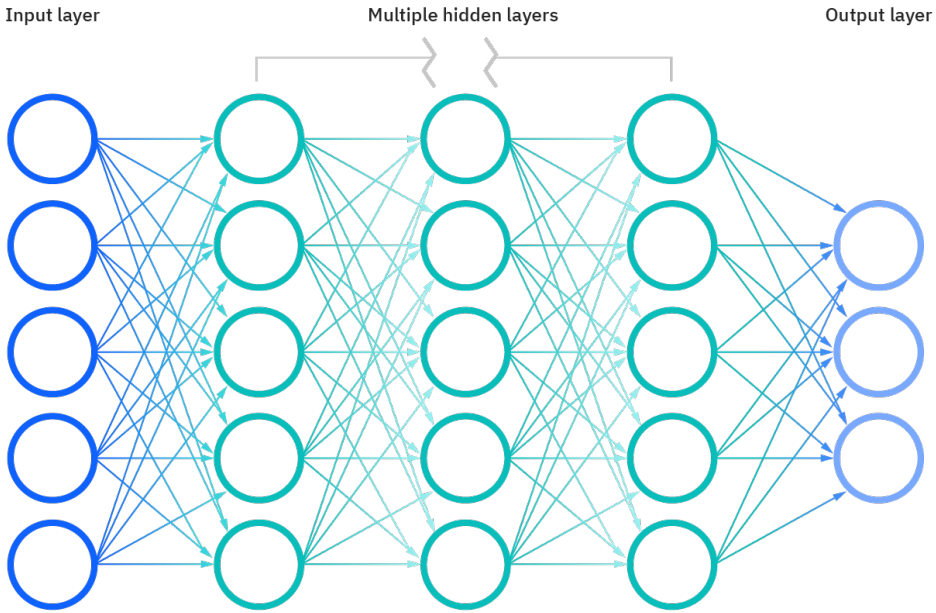


Pretty useless by itself, but if we stack many in series...

Deep neural networks

...we get a **Deep Neural Network**

a *Multilayer Perceptron*, specifically



- Interesting property: **can approximate any (continuous) function**
- But how do we approximate a very complex function
- Example: *"is this bunch of pixels showing a cat?"*

Deep learning

We need to **learn** from data

i.e., known instances of cats and non-cats

Training a DNN

- We start with random weights → model is pretty useless
- We feed some data into the model and get a *prediction* (output)
- We compute the *loss function* (error → how much is the model wrong?)
- We adjust the model weights based on how they contribute to the error
- How are weights actually adjusted? Backpropagation + gradient descent

With enough training examples, the model will learn the task

Is it really that simple?

Yes

- DL is proving effective at many tasks
- How? Larger models + lots of data and compute → *bitter lessons* [1] and *neural scaling laws* [2]

and

No

- Data labeling is mostly manual → expensive and time-consuming
- Training and running big models is (computationally) expensive

Beyond multilayer perceptrons

- Wait! Do we just treat a picture as a vector of pixels?
- Different types of data: text, images, audio, sensors...
- ...require different model architectures (convolutional neural networks, recurrent neural networks, transformers, etc.)

Wetware I

Motivation and applications

Human needs

~~What can we use embedded ML for?~~

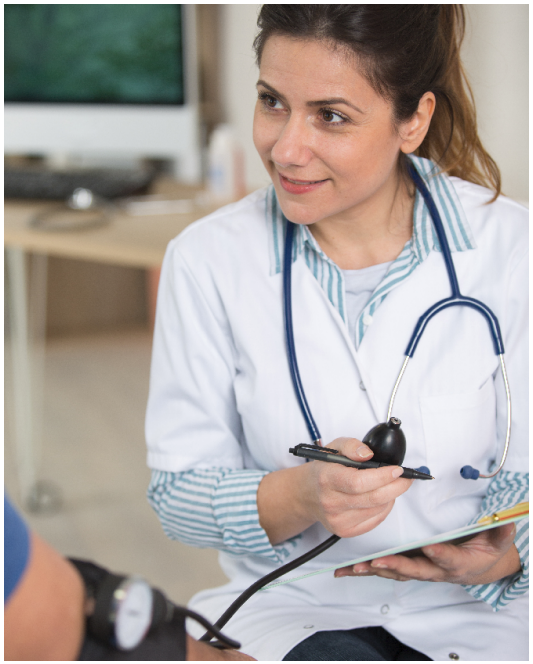
What do we *need*?

Current challenges

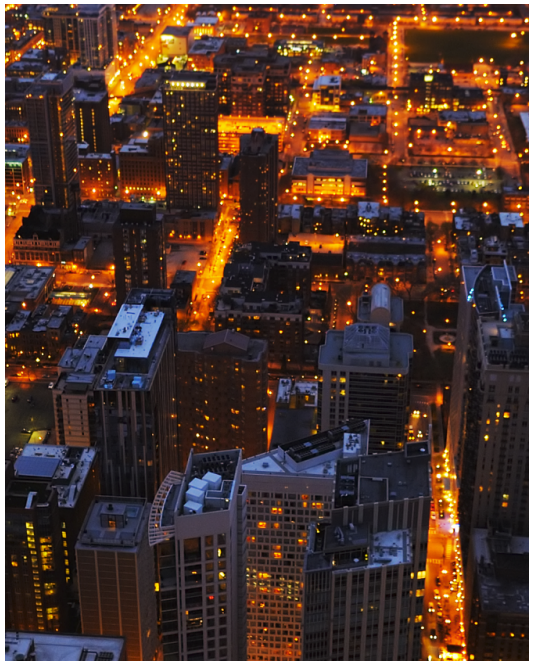
By 2050...

- the global population is expected to reach 9.7 billion
- 22 % of people will be older than 60
- 68 % of people will live in urban areas

Food security



Preventative healthcare



Efficient resource management

Where can embedded ML help?

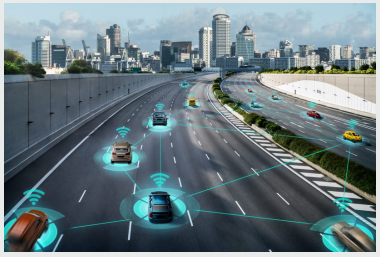
Manufacturing and industrial

Fault detection, predictive maintenance

Healthcare

Wearable monitoring devices, AI-assisted diagnostics

Automotive and transportation



Autonomous and assisted driving, traffic management

Smart cities and infrastructure

Water, waste, and power grid management

Agriculture and food production

Precision farming, pest detection, yield prediction

Environmental monitoring

Real-time alerts (fires, floods, poaching...)

BLERP! [3]

Why does it need to be *embedded*?

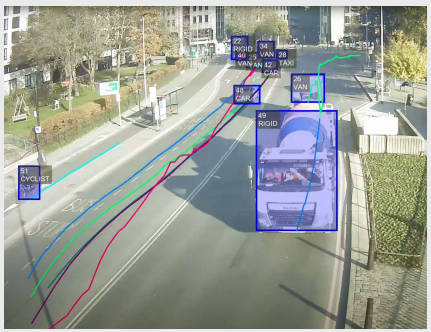
- **Bandwidth**: transmitting raw data from thousands of devices would clog infrastructure
- **Latency**: local processing lowers response time, enabling real-time applications
- **Economics**: maintaining a cloud infrastructure and transmitting data to it is expensive
- **Reliability**: more fault-tolerant systems due to fewer external dependencies
- **Privacy**: fewer opportunities for breaches and abuse when data stays on the device

Some real-world examples

Early wildfire detection [4]

- Gas sensors (Bosch BME688) mounted on trees
- On-device ML model recognizes fire-related gases
- When fire is detected, node sends alert to gateway
- Ultra-low-power, always-on, LoRaWAN mesh

Traffic monitoring and management [5]



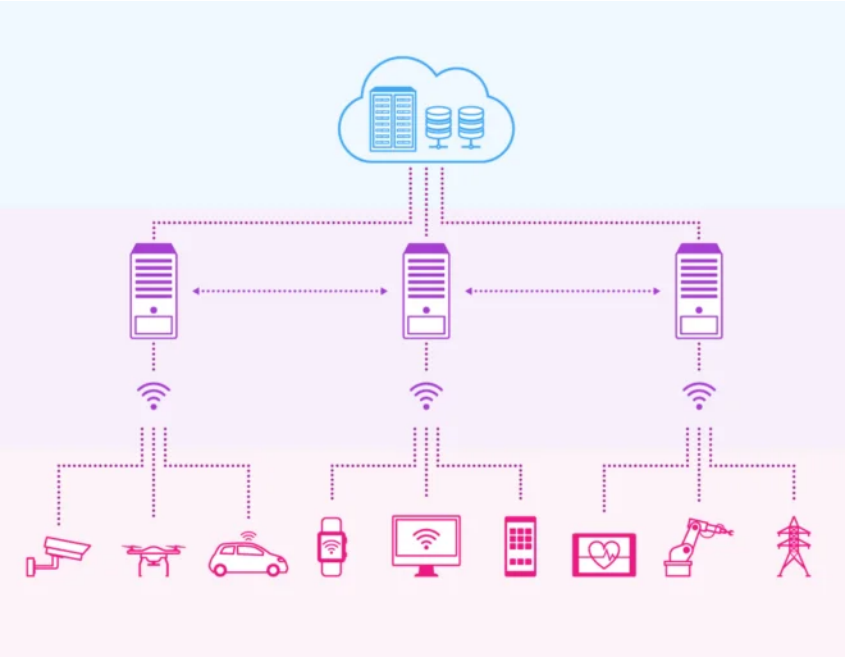
- Camera-based devices installed along roads
- Control traffic lights based on estimated demand
- Collect traffic data for data-driven policy making
- Multiple computer vision models running on-device

Hardware

The computing continuum

The computing continuum

- **Cloud**: centralized, large-scale compute and storage → high throughput, high latency
- **Fog**: intermediate gateways or local servers → pre-processing, caching, aggregation, filtering...
- **Edge**: end nodes (IoT, industrial or user devices) → real-time operation, limited resources



Real-world solutions often operate at multiple levels

Example: smart speaker

- Edge processing for wake word detection and noise suppression
- Cloud processing for speech recognition and voice assistant

What makes embedded ML different?

In the cloud...

- Virtually **unlimited compute**, memory, and storage
- Less latency-sensitive; data can be **processed in batches**
- Models can be retrained often and **deployed instantly**
- Relatively **general-purpose** HW; can run any model

On the edge...

- **Constrained resources** (e.g., due to cost or battery operation)
- **Always-on**, streaming, and **real-time** data processing/operation
- After deployment, **updating models is difficult** or impossible
- Application-dependent HW with **limited flexibility** and support
- Models must be **tailored** to the HW (or vice versa)

Hardware

Overview of hardware platforms

Overview of hardware platforms

Name	Characteristics	Applications	Examples
Microprocessor (CPU/MPU)	High freq, multi-core, sophisticated instructions and cache hierarchy	Robotics, automotive, home automation, medical, smartphones	x86, ARM Cortex-A
Microcontroller (MCU)	Low freq, low power, built-in memory and peripherals	Wide range; wearables, personal devices, IoT, industrial automation	ARM Cortex-M, RISC-V, ESP32
Graphics processing unit (GPU)	Single instruction, multiple threads; high parallelism, bandwidth, and throughput	Multiple heavy workloads, computer vision, smartphones	NVIDIA Jetson, Adreno
Neural processing unit (NPU)	Heterogeneous designs; low-precision math, multi-level local memory, static graph execution	Anything requiring accelerated neural network inference	Qualcomm HTA, ARM Ethos-U/N, Google EdgeTPU, Myriad X, NVIDIA DLA

Real-world solutions often combine multiple platforms

Other platforms and terms

DSP (digital signal processor)

- Optimized instructions (SIMD, fixed-point math, HW loops)
- Used for audio, radar, image, telecommunications...

SoC (system-on-chip)

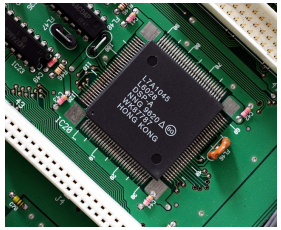
- Processing, memory, storage, and connectivity on one die
- Flexible/adaptable: from mobile phones to IoT

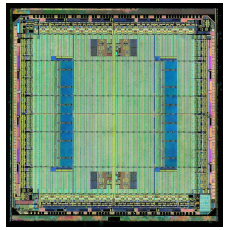
FPGA (field-programmable gate array)

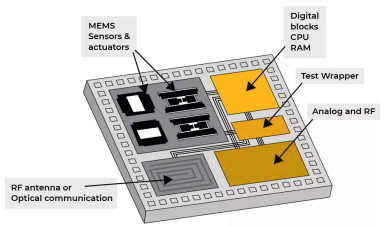
- Reconfigurable digital logic
- Prototyping or low-volume domain-specific tasks

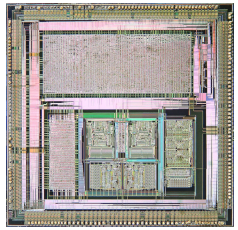
ASIC (application-specific integrated circuit)

- High-efficiency, single-purpose chip (e.g., video encoding)
- Like FPGA but for large volumes





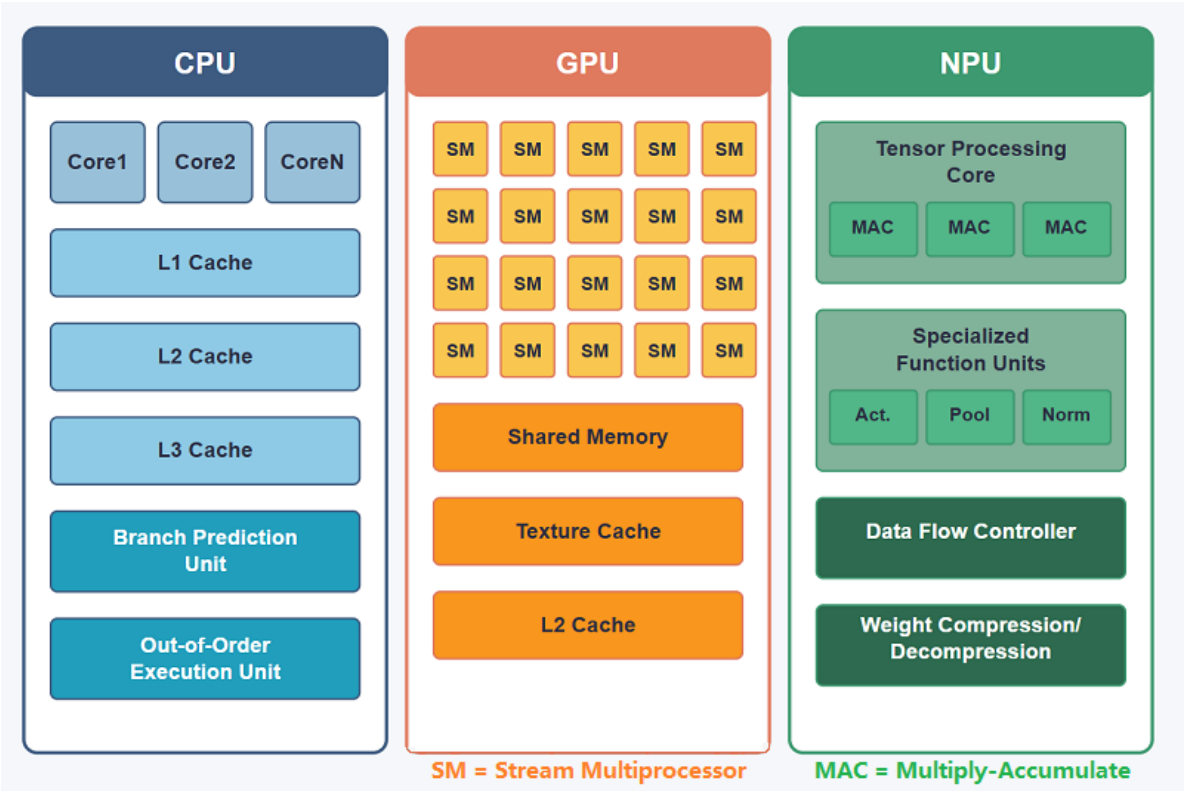




What changes in practice?

Different HW for different workloads

- CPU: few cores with many specialized instructions → **general-purpose**
- GPU: many simpler cores running same instruction → **parallel processing**
- NPU: fixed-function HW (MAC arrays, activations) → **neural networks**



Real-world examples

Reolink RLC-810A surveillance camera

- SoC: Novatek NT9852x (2x Cortex-A9 CPU + NPU)
- On-device: motion detection, object recognition (person, vehicle, animal)
- Fog (hub device): local storage, video captioning
- Cloud: remote storage, natural language querying

Butterfly Network iQ3 handheld ultrasound probe

- SoC: Microchip MPFS250T (4x RISC-V CPU + FPGA)
- On-device: image acquisition and preprocessing
- Smartphone: various predictive and diagnostic models (bladder volume, B-line count)
- Cloud: data storage, AI model marketplace

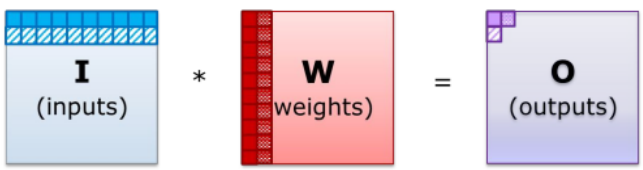
Hardware

Neural network acceleration

Why do we need custom accelerators?

Are neural networks any special?

- Pre-determined, **static computational graphs** (no conditional branching, fixed-length loops) → no need for control flow
- Inputs and model weights are **used multiple times** → keep data close to computation, avoid unnecessary memory transfers
- Mostly boil down to **multiply-accumulate (MAC)** operations
- Can often run with very **low-precision** arithmetics (INT8 or less)

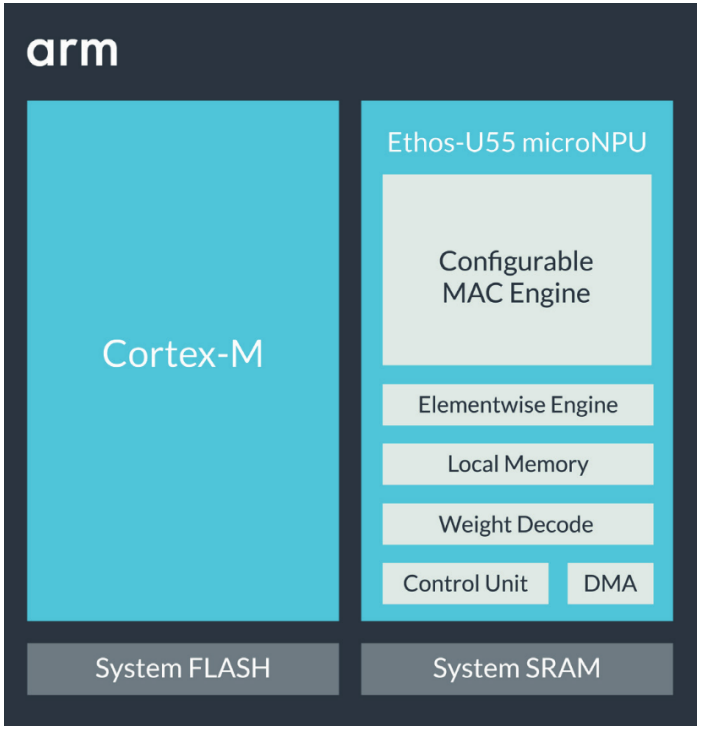


Design priorities for NPUs

- Energy efficiency over flexibility
- Minimal data movement
- High compute utilization
- **Maximize throughput per Watt**

Architecture overview

Case study:
ARM Ethos-U65 microNPU
(IoT, wearables, consumer...)



Control logic

- **Host MCU** requests an inference job
- **DMA** handles different streams (commands, input, output, weights...)
- **Control unit** parses command stream and orchestrates data transfer and computation

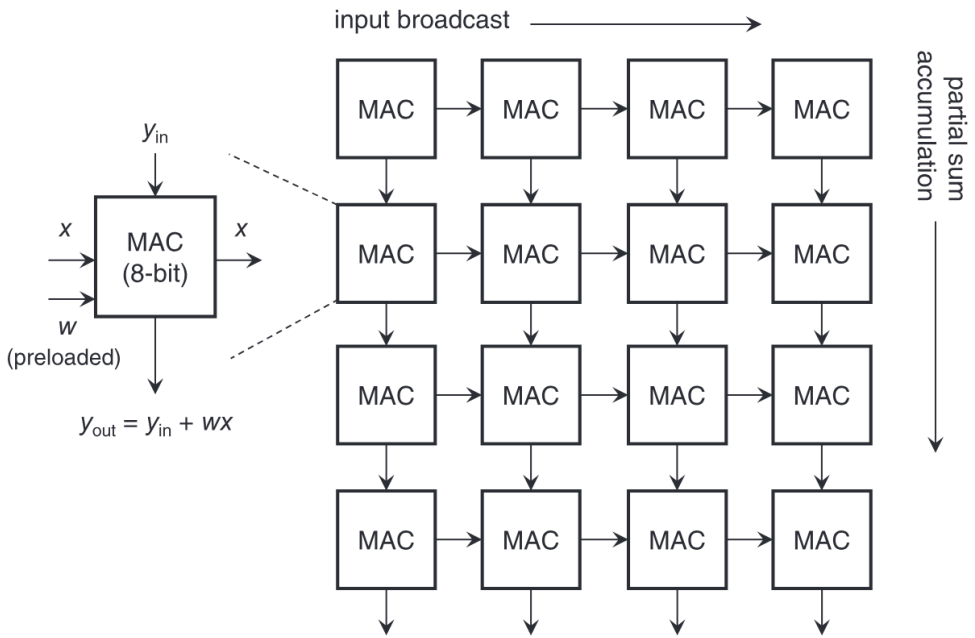
Compute units

- **MAC engine** accelerates dot products
- **Elementwise engine** parallelizes activation functions and other ops
- Weights for current layer and partial results are kept in **local memory**

Systolic arrays

Goal: perform many MACs with minimal overhead

- Weights are pre-loaded into each MAC unit
- Input data flows left to right →
- Partial results flow top to bottom ↓



Advantages

- Input/weights reuse
- Local data movement
- Minimal overhead
- High throughput

Hardware

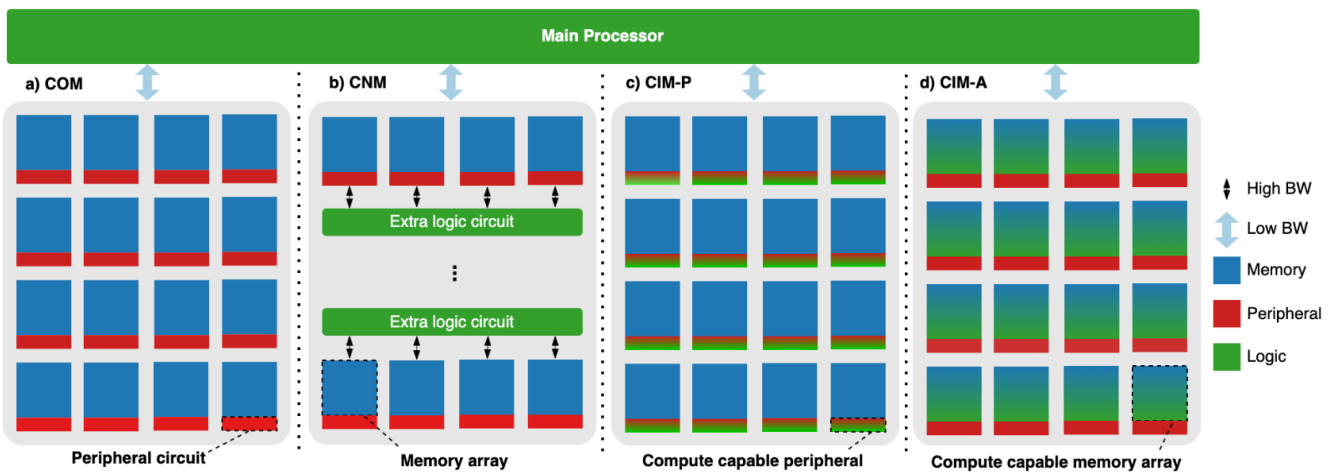
Latest and niche topics

Compute-in-memory

Problem: modern computational workloads are increasingly more data-intensive ⟶ computing systems are **bottlenecked by memory**

Compute-in-memory: a paradigm shift to...

- Perform computation where it makes sense (i.e., close to the data)
- Make computing architectures more data-centric

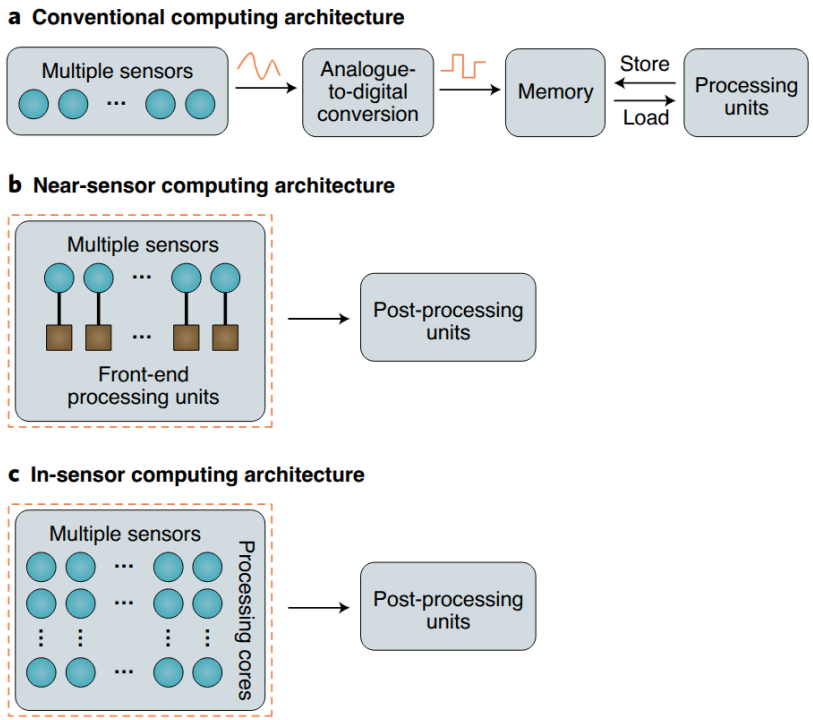


In/near-sensor computing

- **Problem**: data transfer from external peripherals is even more expensive and slow
- **Solution**: offload some computation (preprocessing, feature extraction) to the sensor itself → transfer less data

Examples

- **IMU** (accelerometer, gyroscope): step count, gesture recognition
- **Microphone**: voice activity detection, keyword spotting
- **Camera**: region-of-interest extraction



Neuromorphic computing

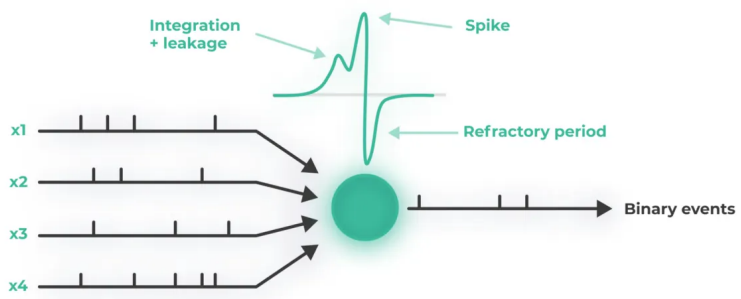
- **Conventional neural networks**: clock-driven, synchronous, and dense matrix operations ⟶ power-hungry
- **Neuromorphic systems**: event-driven, asynchronous, and sparse spiking neurons ⟶ efficient and biologicaly motivated

Main technologies

- **Spiking NNs**: mimic activation patterns of biological neurons
- **Specialized HW**: mixed-signal or digital neuromorphic accelerators, spike encoders/decoders
- **Event-based sensors**: asynchronous cameras/microphones emitting spikes only on intensity/amplitude changes

Advantages

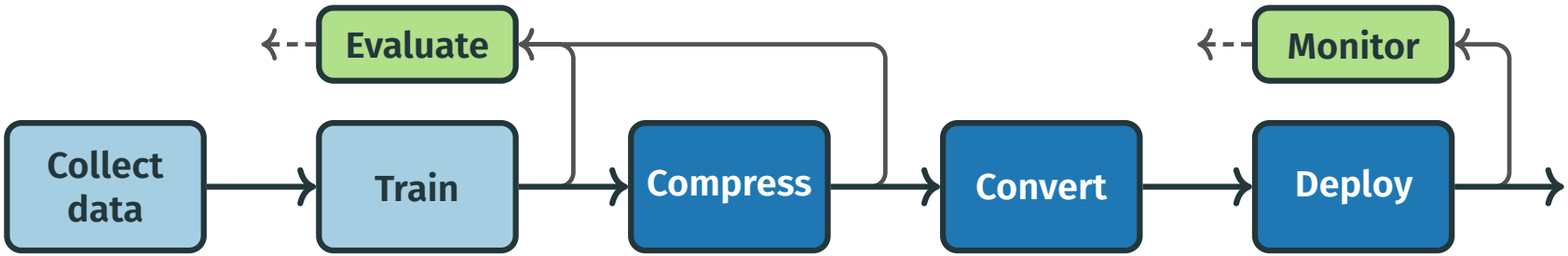
- Ultra-low power
- Continual learning



Software

The embedded ML pipeline

The embedded ML pipeline



- **Collect data**: acquire, label, and pre-process data
- **Train**: design model and train it, often on GPUs/cloud
- **Compress**: reduce size and computational complexity
- **Convert**: export to intermediate format
- **Deploy**: compile into executable code and integrate into firmware

Evaluate/monitor what, exactly?

- After compression: check for performance degradation using task-specific metrics (e.g., accuracy, mean squared error)
- In production: make sure model performs as expected in real-world conditions (and potentially acquire new data)

What embedded ML also cares about

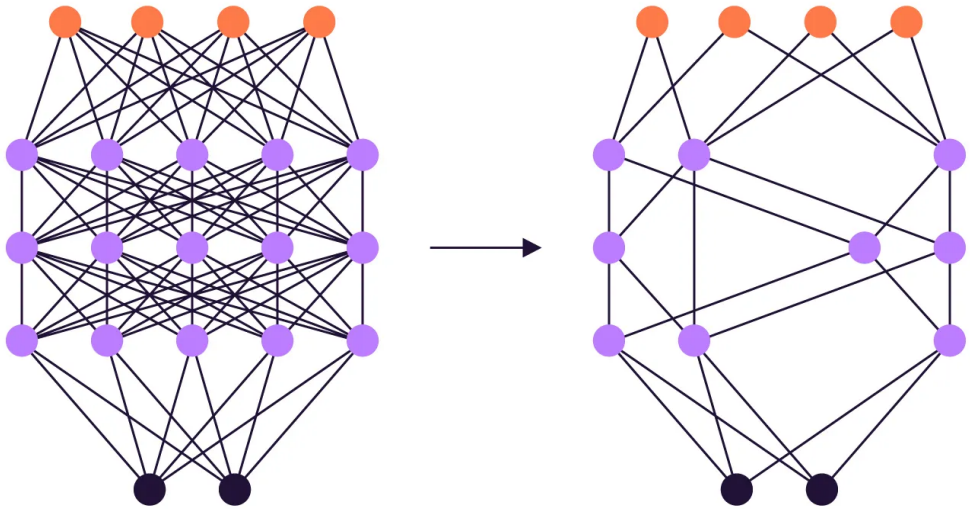
- **Model size** (kB, MB): determined by parameter count and precision
- **Operations** (FLOPs, MACs): related to model size and architecture
- **Inference time** (µs, ms): duration of inference on single sample
- **Throughput** (FPS): amount of data processed per unit of time
- **Memory footprint** (kB, MB): working memory required to perform inference, including parameters and the intermediate activations
- **Energy consumption** (µJ): energy required to perform inference

Software

Model compression

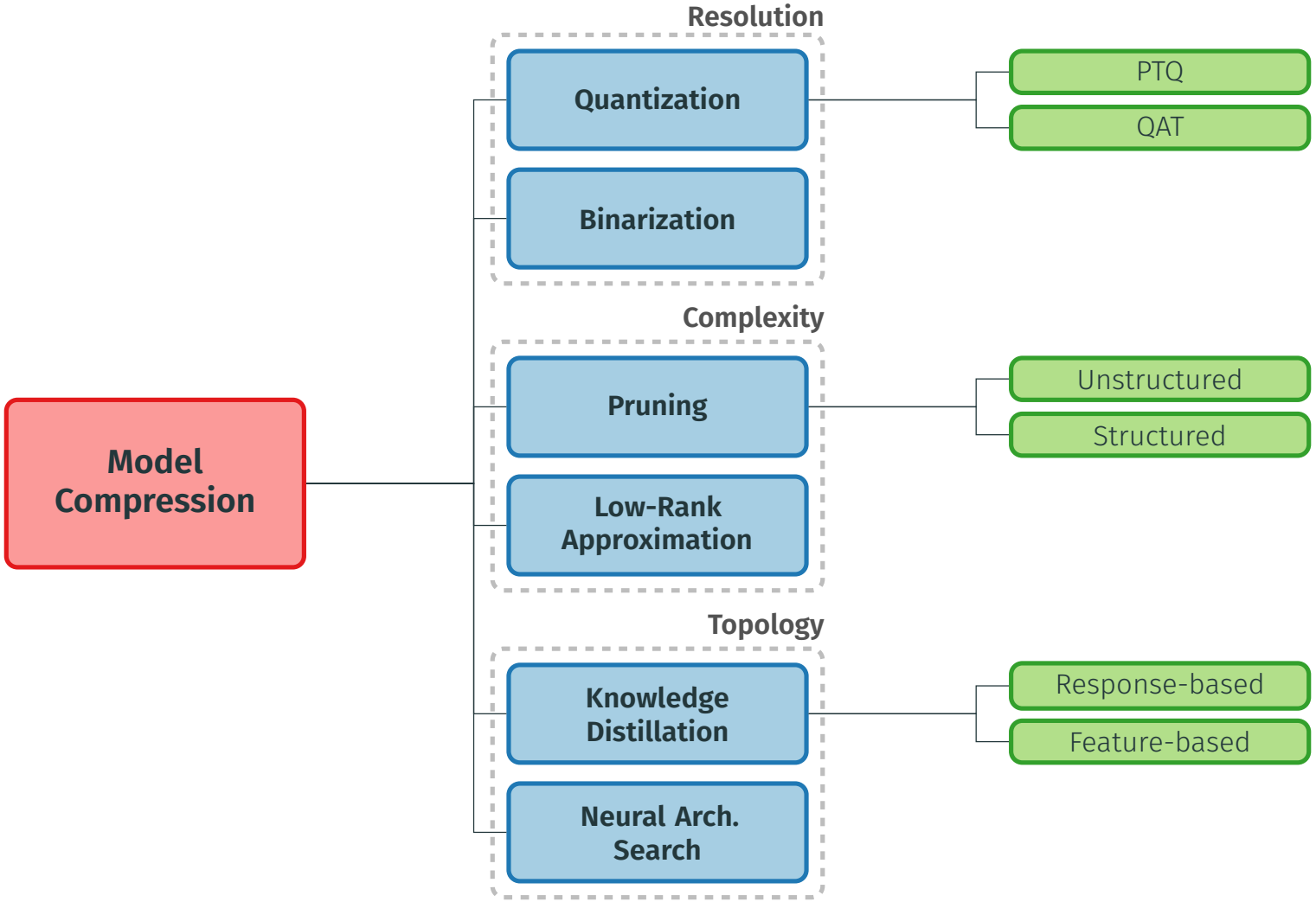
Model compression: what and why

Model compression reduces the resource requirements of a model while maintaining its performance



- DL models are often **overparameterized** and **redundant**
- Model compression helps us figure out what's worth keeping

Model compression: taxonomy

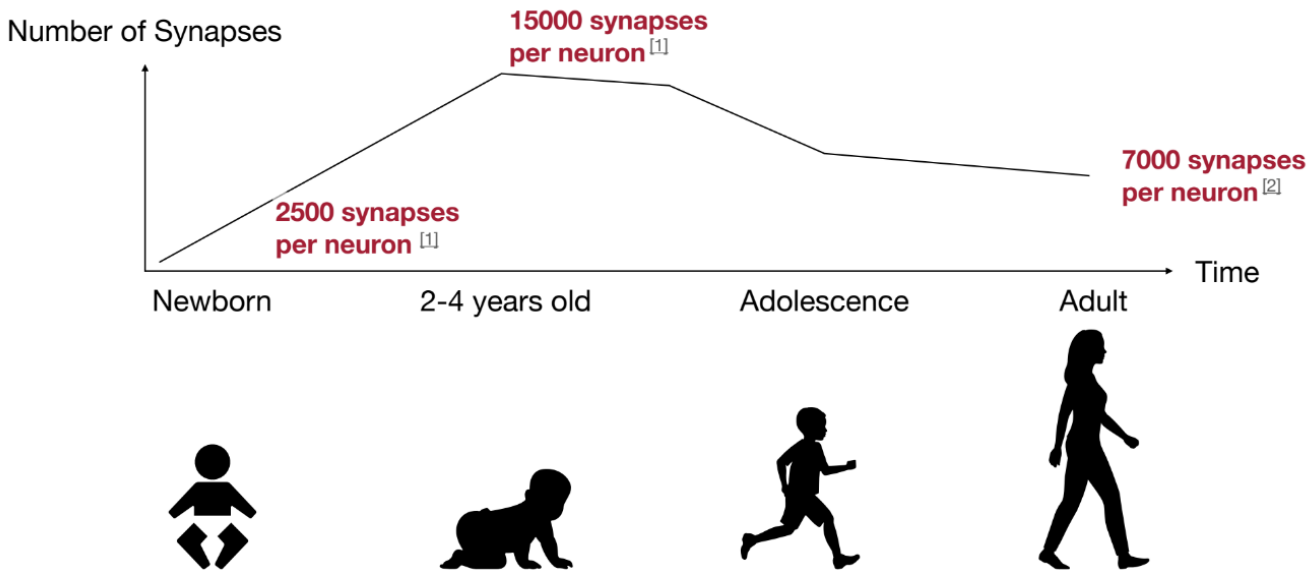


Model compression: classification of techniques

- **Quantization** — including **Binarization** — reduces the numerical precision, i.e., the number of bits per parameter or operation
- **Pruning** and **Low-Rank Approximation** reduce the computational complexity, i.e., the number of arithmetic operations
- **Knowledge Distillation** and **Neural Architecture Search** alter the model topology, resulting in smaller or fewer layers

Model compression: pruning (1)

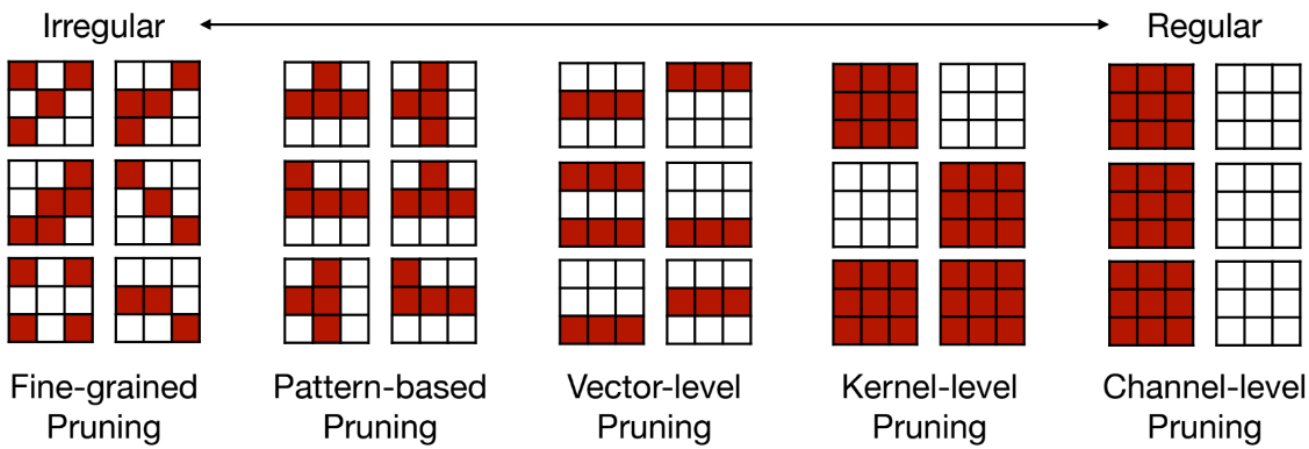
Pruning removes redundant or less important parameters from a neural network, reducing size and computation [6, 7]



Model compression: pruning (2)

Granularity: how should we prune?

- **Unstructured**: removes individual weights without considering position; results in sparse matrices → requires specialized HW
- **Structured**: removes entire groups (filters, channels, neurons, attention heads); lower compression but works with generic HW
- **Semi-structured**: removes blocks/patterns, balances compression ratio and HW support (easier to accelerate)



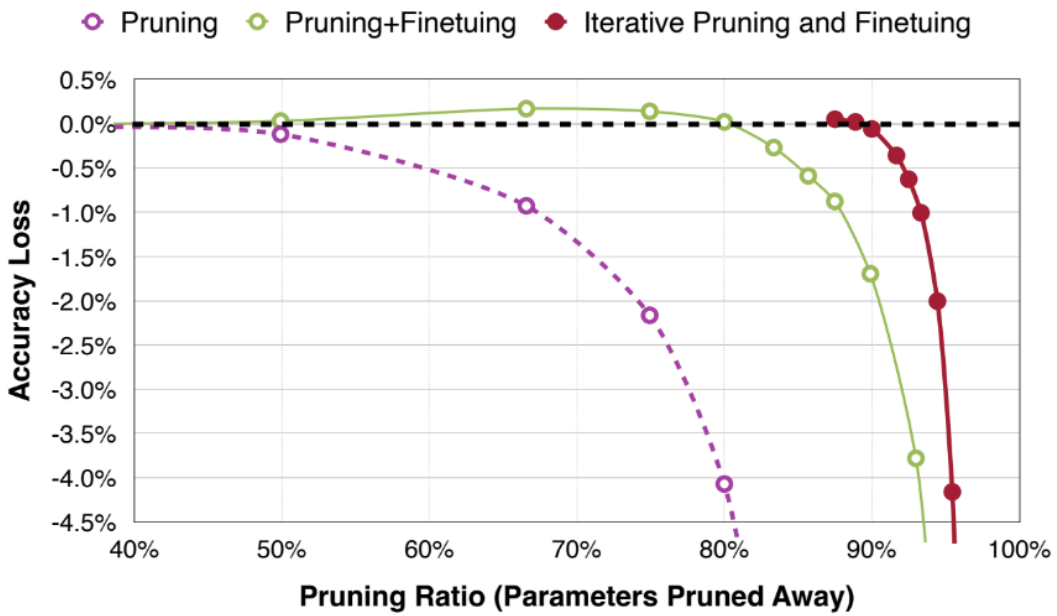
Model compression: pruning (3)

Pruning criteria: what should we remove?

- **Weight magnitude**: absolute value of weights
- ℓ_1/ℓ_2 **norm**: used for groups of weights in structured pruning
- **Sensitivity/Saliency**: remove weights that don't affect train loss
- **BatchNorm**: use learned scaling factors as channel importance

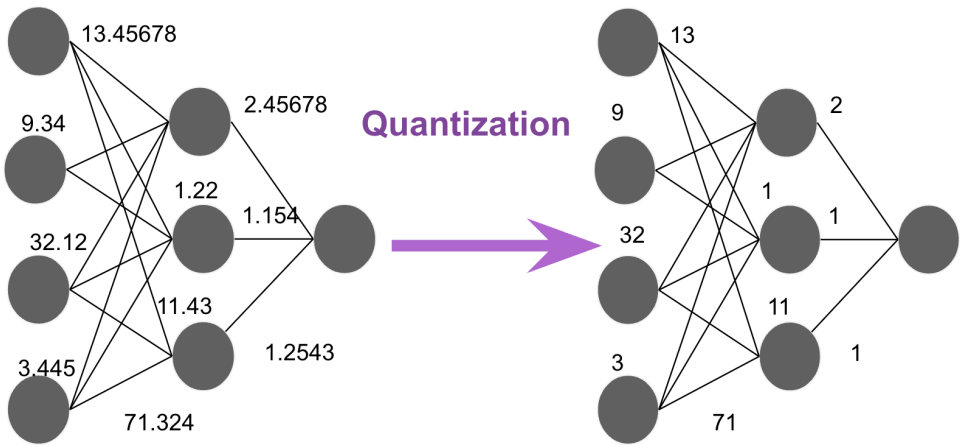
Other aspects

- When to prune? During training, after, or iteratively
- How much? Pruning ratios, global vs. per-layer



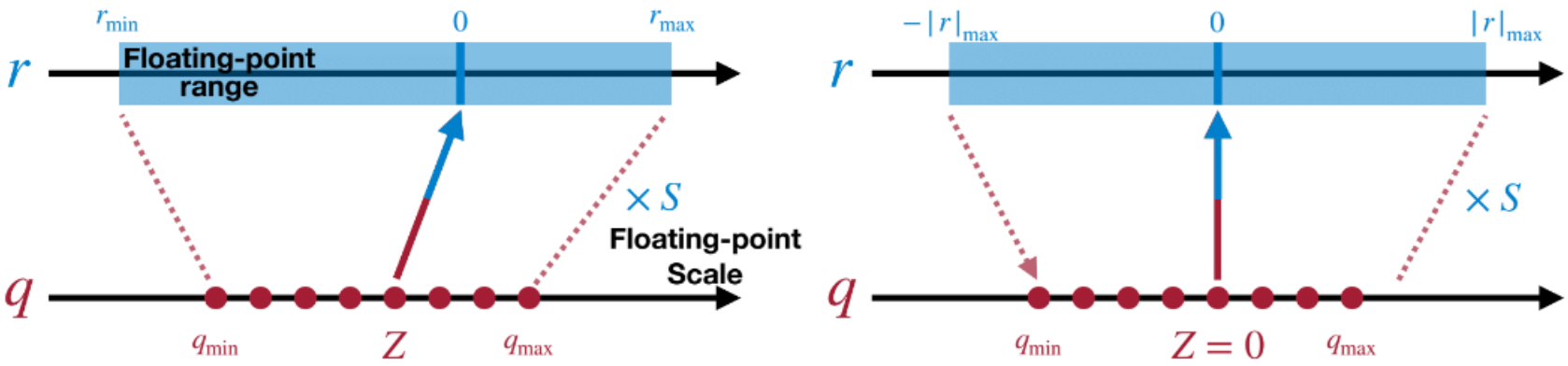
Model compression: quantization (1)

Quantization maps model weights and activations into lower-precision approximations (e.g., `int8` or `int4`) [7–9]



- Quantized models take up less memory and can run on hardware that doesn't support floating-point arithmetics (e.g., MCUs)
- **Uniform quantization** (most common method) is a linear mapping of integer values to floating-point values

Model compression: quantization (2)



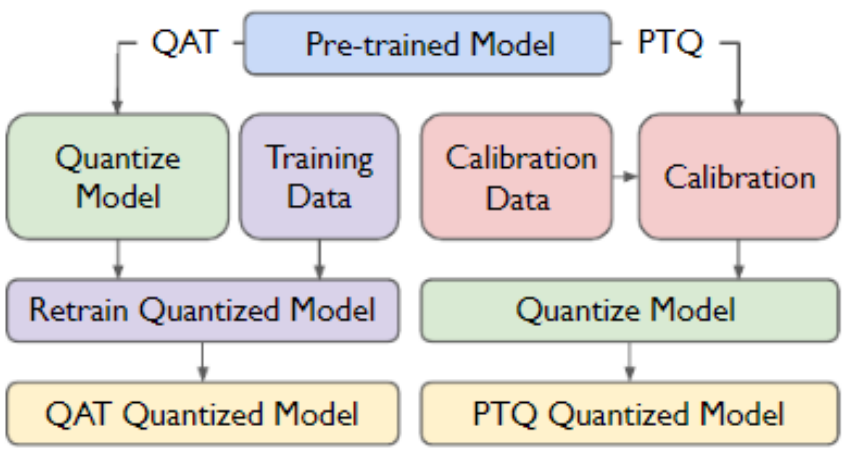
Main parameters

- **Scaling factor** S: distance between quantization levels
- **Zero-point** Z: offset to center the quantization range
- $[r_{\min}, r_{\max}]$: range of real-valued values
- $[q_{\min}, q_{\max}]$: range of quantized values

Model compression: quantization (3)

When to quantize?

- **Post-training quantization (PTQ)**: simple to implement, doesn't require labeled data or training steps, few hyperparameters
- **Quantization-aware training (QAT)**: simulates effect of quantization during training and adjusts weights and quantization parameters; more effective at low bit-widths

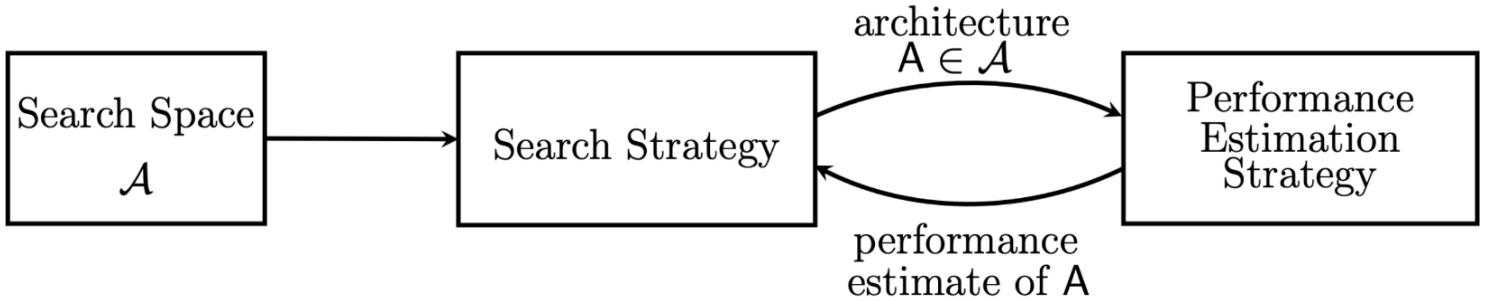


Additional considerations

- **Symmetric** ($Z = 0$, used for weights) vs. **asymmetric** (used for activations)
- **Per-tensor** (same S and Z for entire layer) vs. **per-channel**

Model compression: neural architecture search (1)

Neural Architecture Search (NAS) automates neural network design by searching for architectures that maximize an objective [10, 11]

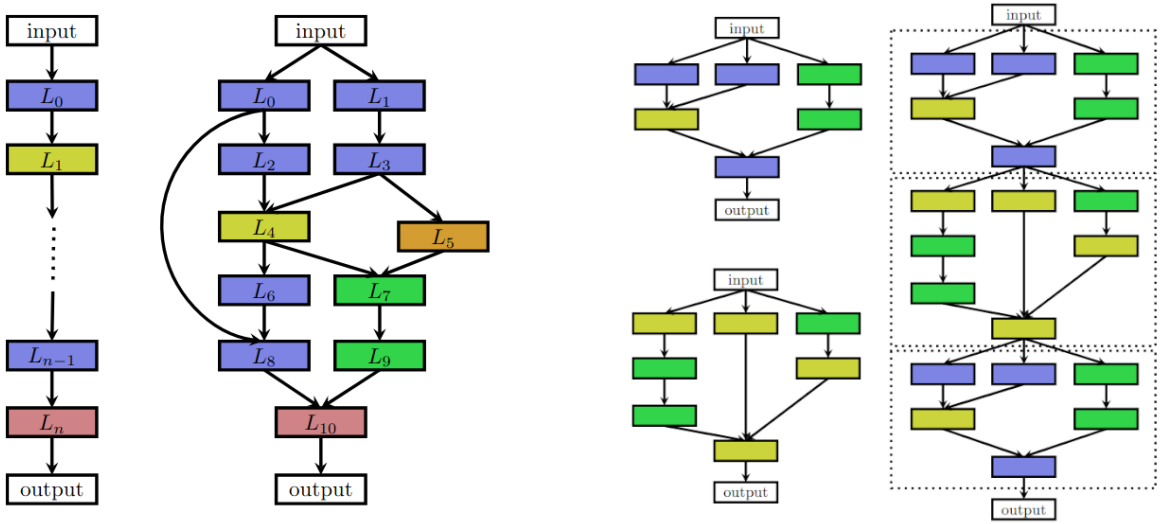


- Can address **multiple objectives** simultaneously (e.g., accuracy, latency, size, memory usage, etc.)
- Can include **hardware-aware** constraints or even **integrate HW platform design** as part of the search

Model compression: neural architecture search (2)

Search space

- Set of all possible architectures that can be discovered
- Defined by a set of **architectural choices** (number of layers, number of units per layer, type of operations, node connections)
- Trade-off between human insight, granularity, size of space, and computational budget



Model compression: neural architecture search (3)

Search strategies

- **Random search**: sample architectures randomly
- **Reinforcement learning**: *controller* network generates architecture based on *reward* signal
- **Genetic algorithms**: evolve populations of architectures through selection, crossover, and mutation
- **Bayesian optimization**: build *surrogate model* of performance; use *acquisition function* to pick next architecture

Performance estimation

- Problem: training and evaluating each model is very expensive
- Efficient alternatives: learning curve extrapolation, zero-cost proxies, dataset distillation, weight inheritance

Software

Deployment tools

Deployment tools

How do we go from compressed model to executable binary?

- **Problem**: different HW targets (MCUs, SoCs, FPGAs) require different representations and compilation flows
- **"Solution"**: a few **model exchange formats** (ONNX, TFLite, NNEF) and many **application-specific tools** with different focuses

Overview of tools

Inference engines

- Convert model into internal representation (offline) and execute it using a runtime that's integrated into the firmware
- Most popular approach, with many runtimes available
- Examples: TFLite/TFLM, ExecuTorch, ONNX Runtime, Tract

Compilers and code generators

- Generate optimized, target-specific machine code or native code (e.g., C/C++, Rust) for direct execution
- Examples: TVM, XLA, IREE/MLIR, uTensor, onnx2c

Most tools are a mix of both approaches

Vendor-specific tools

Tool	Vendor	Target	Description
STM32Cube.AI	STMicro	MCU	Converter + code generator + runtime
eIQ Toolkit	NXP	MCU / MPU	Converter + runtime + quant.
MPLAB ML Suite	Microchip	MCU	End-to-end solution
Reality AI Tools	Renesas	MCU	End-to-end solution
Vela	Arm	NPU	Compiler + partitioner + quant.
ESP-NN	Espressif	MCU	Optimized kernels for TFLM
TensorRT	NVIDIA	MPU / GPU	Compiler + runtime
OpenVINO	Intel	CPU / NPU	Converter + optimizer + runtime + quant.
Vitis AI	AMD / Xilinx	FPGA / SoC	Quantizer + compiler + runtime
TIDL / Edge AI SDK	Texas Instruments	DSP / SoC	Converter + compiler + runtime
RKNN Toolkit	Rockchip	NPU / SoC	Converter + quantizer + runtime
DRP-AI TVM	Renesas	NPU / SoC	Compiler + runtime
SNPE / QNN SDK	Qualcomm	SoC	Converter + runtime + quant.
NeuroPilot	MediaTek	SoC	Converter + compiler + runtime
Edge TPU Compiler	Google	TPU	Compiler + runtime
HailoRT	Hailo	NPU	Compiler + runtime
Akida Toolsuite	BrainChip	SoC / NPU	Converter + quant. + runtime
GAP Flow	GreenWaves	DSP / SoC	Compiler + runtime + auto-tuning
Edge Impulse	Multi-vendor	Multi-target	End-to-end solution (web UI)

…plus many more smaller players

Software

Latest and niche topics

Continual learning

Continual learning enables models to adapt to new data after deployment [12]

Motivation

- In real-world, data changes over time (*concept drift*)
- Re-training (in the cloud) and re-deploying might be infeasible for many embedded or privacy-sensitive applications

Approaches

- **Progressive networks**: allocate new parameters for new tasks
- **Regularization**: penalize large weight changes
- **Memory-based**: store or generate samples from previous tasks

Federated learning

Federated learning enables distributed training without sharing raw data [13]

Advantages

- Improved **privacy** and **data ownership**
- **Reduced bandwidth usage** compared to raw data transmission
- Enables **personalization** and cross-device **collaboration**

Principle

- Clients (devices) train locally on their own data
- Only model updates or gradients are sent to the central server
- Server aggregates updates and broadcasts new global model

Dynamic inference

Dynamic neural networks can adapt their structures or parameters to the input during inference [14]

Downsides of static neural networks

- Fixed amount of computation regardless of difficulty
- Generalizing across all input conditions requires large models
- Fixed trade-off between performance and efficiency

Some examples of DynNNs

- **Early-exiting**: control how many layers are used
- **Mixture-of-Experts**: switch between different sub-networks
- **Slimming**: decide how many neurons or channels are used

Wetware II

Responsible embedded ML

Responsible embedded ML

How can we do it right?

No idea, but here's an example
of how it's been done **wrong**

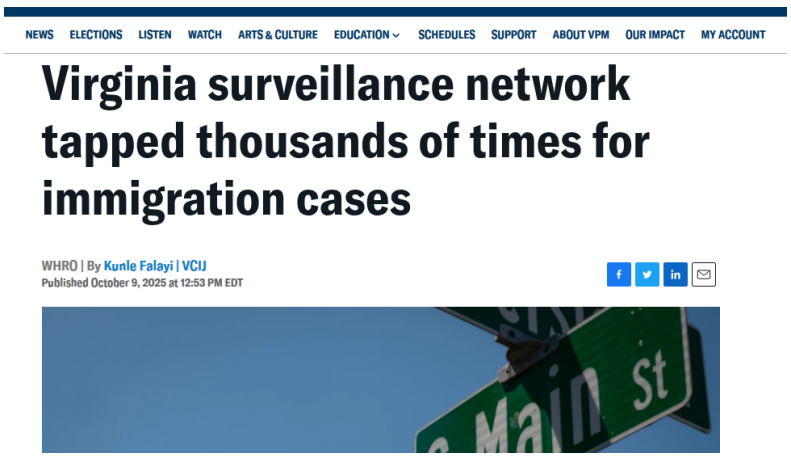
Flock Safety

- Camera-based automated license plate readers (ALPR)
- On-device **licence plate recognition and vehicle fingerprinting**
- 90 000 devices across 6000 clients (including 5000 police depts)
- Claim: **10 % of US crimes solved** using Flock (700 000 cases/year)
- $300 million sales in 2024, $7.5 billion valuation [15]

Embedded ML success story?

Some concerning aspects [16]

- **Privacy**: no possibility to opt-out, data often shared with third parties
- **Data protection**: reports of hacked devices and leaked credentials
- **Fairness**: disproportionate deployment in marginalised communities
- **Transparency**: no publicly available data on accuracy or error rates
- **Accountability**: lack of oversight on how data is used
- **Public engagement**: deployed without consultating local community
- **Reporting**: official impact studies and figures widely disputed
- **Compliance**: devices installed without permit; long-term surveillance without warrant may constitute violation of Fourth Amendment



Additional resources

Online courses

- EdX course on TinyML from Harvard (self-paced, paid): 🔗
- Course on TinyML from MIT (lecture recordings, free): 🔗
- Course on Embedded ML from Edge Impulse (self-paced, free enrollment): 🔗
- MEAD course on Embedded AI (Feb 2026, paid): 🔗
- TinyML book (free preview, video tutorials): 🔗
- Course on Deep Learning from Stanford (lecture recordings, free, iconic): 🔗

Conferences and other events

- Edge AI/TinyML Summit: 🔗
- Edge AI workshops at CVPR: 🔗 🔗
- ACACES Summer School: 🔗

Curated lists

- Edge AI: 🔗
- Awesome TinyML: 🔗

References (1)

[1] R. Sutton, "The bitter lesson," Incomplete Ideas, [Online]. Available: http://www.incompleteideas.net/IncIdeas/BitterLesson.html

[2] Y. Bahri, E. Dyer, J. Kaplan, J. Lee, and U. Sharma, "Explaining neural scaling laws," *Proceedings of the National Academy of Sciences*, Jul. 2024, DOI: 10.1073/pnas.2311878121

[3] J. Bier, "AI and vision at the edge," EE Times, [Online]. Available: https://www.eetimes.com/ai-and-vision-at-the-edge/

[4] D. Networks, "Silvanet - AI wildfire detection in minutes," Dryad, [Online]. Available: https://www.dryad.net/silvanet

[5] VivaCity, "Smart signal control solution," VivaCity, [Online]. Available: https://vivacitylabs.com/products/smart-signal-control/

[6] H. Cheng, M. Zhang, and J. Q. Shi, "A survey on deep neural network pruning: Taxonomy, comparison, analysis, and recommendations," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Dec. 2024, DOI: 10.1109/TPAMI.2024.3447085

[7] T. Liang, J. Glossner, L. Wang, S. Shi, and X. Zhang, *Pruning and quantization for deep neural network acceleration: A survey*, Jun. 2021, arXiv: 2101.09671[cs].

[8] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, *A survey of quantization methods for efficient neural network inference*, Jun. 2021, arXiv: 2103.13630[cs].

[9] M. Nagel, M. Fournarakis, R. A. Amjad, Y. Bondarenko, M. van Baalen, and T. Blankevoort, *A white paper on neural network quantization*, Jun. 2021, arXiv: 2106.08295[cs].

References (2)

[10] C. White et al., *Neural architecture search: Insights from 1000 papers*, Jan. 2023, DOI: 10.48550/arXiv.2301.08727

[11] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *Journal of Machine Learning Research*, 2019. [Online]. Available: http://jmlr.org/papers/v20/18-598.html

[12] M. D. Lange et al., "A continual learning survey: Defying forgetting in classification tasks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021, DOI: 10.1109/TPAMI.2021.3057446

[13] E. Gabrielli, G. Pica, and G. Tolomei, *A survey on decentralized federated learning*, Aug. 2023, DOI: 10.48550/arXiv.2308.04604

[14] Y. Han, G. Huang, S. Song, L. Yang, H. Wang, and Y. Wang, "Dynamic neural networks: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Nov. 2022, DOI: 10.1109/tpami.2021.3117837

[15] T. Brewster, "AI startup flock thinks it can eliminate all crime in america," Forbes, [Online]. Available: https://www.forbes.com/sites/thomasbrewster/2025/09/03/ai-startup-flock-thinks-it-can-eliminate-all-crime-in-america/

[16] L. Daniel, "Privacy violated, warrantless surveillance alleges flock safety camera lawsuit," Forbes, [Online]. Available: https://www.forbes.com/sites/larsdaniel/2024/10/22/warrantless-surveillance-federal-lawsuit-challenges-flock-safety-cameras/