# 24EC504 – Unit IV – Part 1: TinyML, Edge AI and Machine Learning on Microcontrollers

## 1. Introduction to TinyML

### 1.1 From Machine Learning to TinyML

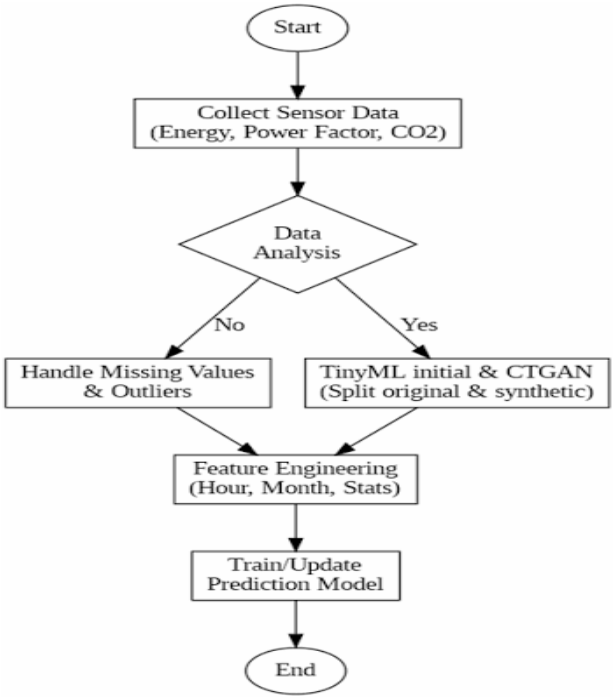
Machine Learning lets computers learn from data and decide without being explicitly programmed for every rule. It finds patterns in data and uses them to predict on new data (image recognition, speech processing, translation, recommenders). Three types:

- **Supervised learning:** trained on labelled data (inputs mapped to known outputs); used for classification and regression: image/speech recognition, predictive analytics, fraud detection, medical diagnosis.
- **Unsupervised learning:** finds hidden patterns in unlabelled data; clustering, dimensionality reduction, association rules (for example customer segmentation).
- **Reinforcement learning:** an agent learns the best sequence of actions by trial and error with rewards or penalties (game agents, robotic control).

Traditionally ML runs on servers, desktop GPUs or the cloud. But many always-on devices (sensors, wearables, appliances) have only a few hundred KB of RAM and run on milliwatts. TinyML brings AI to exactly these devices.

### 1.2 What is TinyML?

TinyML is a fast-growing field that runs ML models, typically deep neural networks, directly on extremely low-power, low-memory embedded devices such as microcontrollers rather than on the cloud. A typical model uses tens to hundreds of KB of RAM and Flash and consumes milliwatts to microwatts, allowing always-on, battery-powered operation for months or years. TinyML sits at the intersection of **machine learning** (algorithms and models), **embedded systems** (constrained hardware, real-time firmware) and **applications** (the real use case).




### 1.3 The TinyML Workflow

1. **Model training:** trained on a high-performance system (cloud server or GPU workstation) with a large representative dataset.
2. **Model optimisation and deployment:** simplified to fit the MCU's memory, storage and compute using **quantisation** (32-bit float → 8-bit integers) and **pruning** (removing redundant weights); converted to a compact file (for example `.tflite` FlatBuffer) and cross-compiled into the firmware.
3. **On-device inference:** the optimised model runs on real-time sensor data (audio, vibration, image) with no constant cloud communication.

### 1.4 Key Characteristics

- **Ultra-low power:** micro-watt to milli-watt; multi-year operation on a coin cell.
- **Small memory footprint:** tens or a few hundred KB.
- **Runs entirely on-device:** no network dependency for inference.
- **Real-time response:** latency typically in milliseconds.
- **Enhanced privacy:** raw audio, biosignals and images never leave the device.

### 1.5 Applications

- **Keyword spotting / voice assistants:** always-on wake-word detection on a low-power co-processor.
- **Wearables and healthcare:** heart-rate anomaly detection, fall detection, activity recognition.
- **Predictive maintenance:** abnormal vibration or sound in industrial motors.

- **Smart agriculture:** on-node soil moisture/temperature analysis and pest-sound classification where connectivity is poor.
- **Gesture and activity recognition:** from accelerometer data.
- **Smart-home sensors:** occupancy detection, appliance fault detection, environmental monitoring.

### 1.6 Benefits and Challenges

| Benefits | Challenges |
|---|---|
| Very low power, long battery life | Extremely limited RAM/Flash restricts model size and complexity |
| Low latency, decision made on-device | Accuracy trade-off after aggressive quantisation/pruning |
| Works fully offline | Limited MCU-specific tooling and debugging support |
| Enhanced data privacy and security | Cross-compilation and hardware differences across MCU vendors |
| Reduced cloud/bandwidth cost | On-device retraining/updating is difficult |

Common frameworks: TensorFlow Lite for Microcontrollers, CMSIS-NN (ARM), Edge Impulse and STM32Cube.AI (ARM Cortex-M, ESP32 and other MCUs).

## 2. Edge AI for Embedded Devices

### 2.1 Definition

Edge AI runs ML inference directly on local hardware (microcontrollers, embedded processors, AI accelerator chips) close to where data is generated, instead of sending raw data to remote cloud servers.

| Cloud AI | Edge AI |
|---|---|
| Data captured, sent to remote servers, processed, result sent back | Data processed instantly on the device and acted on locally |
| Performance depends on network connectivity and bandwidth | No network round trip; can work fully offline |

### 2.2 Why Edge AI?

- **Low latency:** decisions in milliseconds.
- **Enhanced privacy:** sensitive raw data stays on the device.
- **Cost and bandwidth savings:** local pre-processing filters noise and irrelevant data.
- **Offline operation:** critical for remote or safety-related applications.

### 2.3 Working Principle

A closed loop entirely inside the device: **Sensor input** (sound, image, ECG, vibration) → **Pre-processing** (filter, normalise, feature-extract) → **On-device inference** (a compressed/quantised model classifies or predicts) → **Decision/action** (alert, control signal, display update). The whole cycle finishes in milliseconds and no data leaves the chip unless required.

### 2.4 End-to-End Data Flow

1. **Data acquisition:** sensors sample raw physical data.
2. **Signal conditioning:** noise filtering, amplification, feature extraction.
3. **Quantized model inference:** a pruned/quantised network runs on the MCU.
4. **Post-processing:** thresholding and confidence checks.
5. **Local output / alert:** drives an actuator, display, buzzer or wireless notification.

The pipeline repeats each sampling cycle, often several times per second. Only summarised results, not raw data, are optionally synced to the cloud.

### 2.5 Hardware Enablers and Model Optimisation

- **Hardware:** low-power MCUs and processors (for example Texas Instruments, STMicroelectronics) with neural-network acceleration; toolchains such as ST Edge AI Suite for converting and flashing models; ARM Cortex-M and ESP32-class SoCs combining compute, ADC, DSP instructions and low-power modes.
- **Quantization:** reduce precision (32-bit float → 8-bit integer) to shrink the model and speed up inference.
- **Pruning:** remove redundant weights and connections to meet memory and power limits.
- **Specialised training:** paradigms and toolchains built for resource-constrained targets (for example Literal Labs).

### 2.6 Real-Time Project Example: Wearable ECG Anomaly Detector

A wrist-worn device detects irregular heartbeats on-device:

- **ECG sensor:** an AD8232-type analog front end captures the signal.
- **Signal conditioning:** the MCU filters noise and extracts R-peak features.
- **Quantized CNN (INT8):** classifies each beat as Normal or Arrhythmia.
- **Local alert:** buzzer or LED triggers instantly.
- **BLE sync (optional):** only the alert/summary, not the raw ECG stream, goes to a paired phone.

All classification happens on the wearable's own MCU: instant, works without internet, and raw biosignals never leave the device.

### 2.7 Applications of Edge AI

Healthcare wearables (ECG/EEG, fall detection, fitness trackers); industrial IoT (predictive maintenance); smart cameras (on-device detection); autonomous vehicles (obstacle detection,

driver assistance); smart home (voice assistants without a cloud round trip); smart agriculture (soil monitoring in low-connectivity fields).

## 3. Machine Learning on Microcontrollers

### 3.1 Why Microcontrollers Are Different

An MCU is a small computer on one chip (CPU, RAM, Flash, I/O) with resources orders of magnitude smaller than a phone or PC. A common embedded-ML MCU has about 256 KB RAM, 1–2 MB Flash, 50–200 MHz clock, and no OS or a minimal RTOS. ML on it means rethinking model architecture, numeric representation, memory management and toolchain while meeting real-time deadlines and a strict power budget (often under 1 mW average in always-on use).

### 3.2 Constraints

| Constraint | Typical budget | Implication for ML |
|---|---|---|
| RAM | 20 KB – 512 KB | Tiny activation/feature buffers; no large intermediate tensors |
| Flash / program memory | 128 KB – 2 MB | Weights, inference code and firmware must fit together |
| Clock speed | 16–200 MHz, no GPU | Integer arithmetic and hand-optimised kernels |
| Power | µW to a few mW | Short bursts or duty-cycled operation |
| No OS / file system | Bare-metal or RTOS | Model compiled into firmware; no loading from disk |

### 3.3 The ML-on-MCU Pipeline

1. **Data collection and labelling:** representative sensor data, usually collected off-device.
2. **Model design:** compact architectures (small CNNs, depthwise-separable convolutions, tiny RNN/GRUs, decision trees, simple dense networks) with few parameters and low FLOPs.
3. **Training:** on a workstation or cloud GPU with TensorFlow/Keras or PyTorch.
4. **Quantization:** post-training or quantization-aware training converts FP32 to INT8 or lower, cutting memory by up to 4× and speeding inference on MCUs without an FPU.
5. **Conversion:** to a lightweight format such as the `.tflite` FlatBuffer.
6. **Code generation / compilation:** embed the model as a byte array linked at compile time, or generate portable C/C++ inference code.
7. **On-device inference and testing:** flash and validate accuracy, latency, RAM/Flash use and power draw under real conditions.

### 3.4 Common Model Types

- **Decision trees / random forests:** extremely lightweight; simple tabular sensor classification.
- **Small dense networks:** basic gesture/vibration classification from a few features.
- **1-D CNNs:** efficient for time series such as audio, ECG and accelerometer data.
- **Depthwise-separable CNNs (MobileNet-style):** lightweight image classification, for example person/no-person detection.
- **Tiny recurrent networks (quantised GRU/LSTM):** used sparingly for sequence tasks such as keyword spotting when 1-D CNNs are not enough.

### 3.5 Key Optimisation Techniques

- **Quantization:** FP32 → INT8 or INT4 for weights and activations.
- **Pruning:** remove low-contribution weights/neurons for a sparse, smaller network.
- **Knowledge distillation:** train a small student model to mimic a larger teacher.
- **Weight clustering:** group similar weights so they store and compress better.
- **Operator fusion and kernel optimisation:** combine operations into optimised routines (for example ARM CMSIS-NN) to cut memory traffic and CPU cycles.

### 3.6 Popular Tooling

| Tool | Role |
|---|---|
| TensorFlow Lite for Microcontrollers | Google's C++ inference engine for MCUs with kilobytes of memory |
| CMSIS-NN | ARM's optimised neural-network kernels for Cortex-M |
| Edge Impulse | End-to-end platform: data collection, training, optimisation, one-click deployment |
| STM32Cube.AI | Converts trained networks into optimised C code for STM32 |
| uTensor, microTVM | Open-source compilers/runtimes generating efficient embedded inference code |

### 3.7 Example Applications

Voice/keyword spotting (tiny CNN on a low-power co-processor); gesture recognition (accelerometer/gyroscope); predictive maintenance (1-D CNN on motor vibration); environmental sensing (air-quality or gas-sensor classification on a battery node).