

Edge AI on Resource-Constrained Embedded Platforms

Prabhat Sharma¹, Devansh Kulkarni², Meenal Joseph³, Arvind Jaiswal⁴

Assistant Professor, Students

Department of Information Technology

Maharani's Science College for Women, India

Prabhatsharma54@yahoo.com, devansh81k@gmail.com, meenaljosephgp@rediffmail.com

Abstract

Edge Artificial Intelligence (Edge AI) is becoming an important paradigm where AI models are deployed directly on embedded devices instead of relying fully on cloud computing. This approach is very useful for real-time decision making, reduced latency, better privacy and lower bandwidth usage. However, embedded platforms such as microcontrollers, single board computers, and IoT devices usually have very limited computational power, memory, and energy resources. Running complex AI models on such platforms becomes a major technical challenge. This paper presents a review of techniques, architectures, optimization methods, and tools that enable deployment of Edge AI on resource-constrained embedded systems. It discusses model compression, quantization, pruning, hardware accelerators, and lightweight neural network architectures. The paper also highlights practical use cases, current challenges, and future research directions in this emerging field.

Keywords

Edge AI, Embedded Systems, TinyML, Model Compression, Low-Power Devices, Quantization, IoT, Neural Networks

1. Introduction

Artificial Intelligence applications are traditionally executed on powerful cloud servers. But with the rapid growth of Internet of Things (IoT) and embedded devices, there is increasing demand to process data locally on the device itself. This concept is known as Edge AI. It allows real-time processing, reduces dependency on internet connectivity and improves privacy of sensitive data.

Embedded devices like Arduino, Raspberry Pi, ESP32, ARM Cortex-M series have limited RAM, storage and power. Deploying AI models on these devices is not simple because most AI algorithms require high computation and memory. Therefore, new approaches are developed to make AI models lightweight and suitable for embedded platforms.

This paper reviews how Edge AI is implemented on resource-constrained embedded systems and what technologies are making it possible.

2. Overview of Edge AI Architecture

Edge AI architecture refers to the structural arrangement of hardware and software components that allow artificial intelligence models to run directly on embedded or edge devices. Instead of sending raw data continuously to cloud servers for processing, the computation is done locally on the device itself. This architecture is specially designed to support real-time analytics, faster response, reduced network dependency, and better privacy.

A typical Edge AI system includes four primary layers: **sensing layer, embedded processing layer, local inference layer, and optional cloud layer.**

2.1 Sensing Layer (Data Acquisition Layer)

This is the first layer where physical data is collected from the environment using sensors.

These sensors may include:

- Temperature and humidity sensors
- Cameras and microphones
- Motion and vibration sensors
- Biomedical sensors (ECG, heart rate)
- Gas, pressure, and proximity sensors

The data captured by these sensors is often raw, noisy and unstructured. Before feeding into AI models, this data requires filtering and preprocessing.

Key role: Continuous collection of real-world signals for analysis.

2.2 Embedded Processing Layer

This layer consists of a microcontroller, microprocessor, or single-board computer which acts as the brain of the system. Common platforms include:

- ARM Cortex-M microcontrollers
- ESP32, STM32 boards
- Raspberry Pi, BeagleBone
- NVIDIA Jetson Nano (for heavier tasks)

At this stage, basic preprocessing of sensor data takes place such as:

- Signal filtering
- Feature extraction
- Normalization
- Data formatting

Because of limited memory and CPU capability, preprocessing must be lightweight and efficient.

Key role: Prepare sensor data into usable format for AI inference.

2.3 Local AI Inference Layer

This is the core of Edge AI architecture where the trained machine learning or deep learning model runs on the device. The model is already trained offline (usually on cloud or powerful computer) and then deployed to the embedded system for inference only.

This layer performs:

- Pattern recognition
- Classification or prediction
- Anomaly detection
- Decision making

Frameworks like TensorFlow Lite Micro, Edge Impulse, and PyTorch Mobile are used here to run optimized models.

Key role: Real-time decision making without internet.

2.4 Decision and Actuation Layer

Based on the output of the inference engine, the device can perform certain actions automatically:

- Trigger alarms
- Control motors or actuators
- Display results on screen
- Send notifications

This makes the system autonomous and intelligent.

Key role: Convert AI output into physical or digital action.

2.5 Optional Cloud Connectivity Layer

Although Edge AI focuses on local processing, cloud connectivity is still useful for:

- Model updates and retraining
- Data backup and analytics
- Remote monitoring
- Firmware updates

Only processed or summarized data is sent to cloud, reducing bandwidth consumption.

Key role: Supportive layer for management and updates.

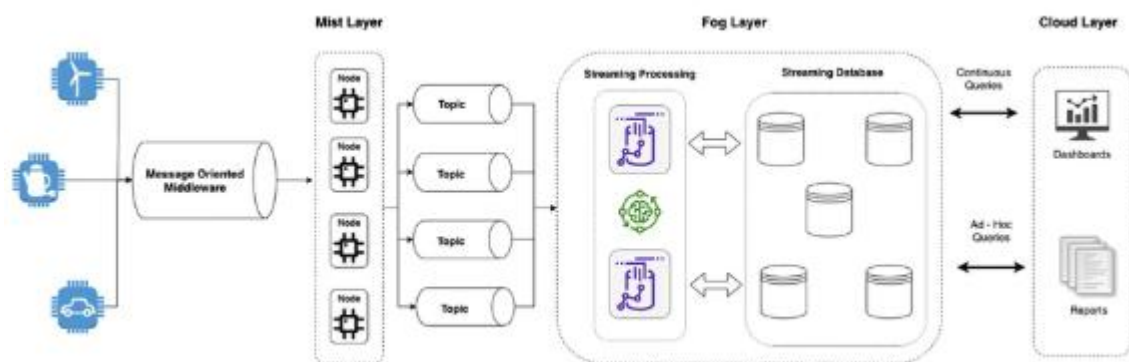


Figure 1: Basic Edge AI Architecture

3. Characteristics of Resource-Constrained Embedded Platforms

Resource-constrained embedded platforms are small computing devices designed to perform specific tasks with very limited hardware capability. Examples include microcontrollers (ARM Cortex-M series, AVR, PIC), IoT boards (ESP32, STM32), and small single board computers. Unlike desktop computers or cloud servers, these devices operate with strict limits on **memory, processing power, storage, and energy**. Understanding these characteristics is very important before deploying Edge AI models on them.

3.1 Limited Random Access Memory (RAM)

Most microcontrollers used in embedded systems have RAM in the range of **64 KB to 512 KB**, which is extremely small compared to gigabytes available in computers. AI models, especially

deep neural networks, usually require large memory for weights, buffers, and intermediate computations.

Because of this:

- Only very small models can be loaded into memory
- Intermediate tensors must be carefully managed
- Memory overflow can crash the system

This limitation forces developers to use model compression and memory-efficient programming.

3.2 Low Processing Power (CPU Constraints)

Embedded processors often run at clock speeds in **tens or hundreds of MHz** and lack floating-point units or GPUs. They are not designed for heavy mathematical computations required by deep learning algorithms.

Effects of this constraint:

- Slower inference time if model is large
- Need for integer arithmetic instead of floating point
- Use of optimized libraries and hardware instructions

Hence, AI models must be simplified to run within acceptable time limits.

3.3 Limited Storage (Flash/ROM)

Program memory (Flash) in embedded devices is also limited, usually a few **hundred KB to few MB**. The AI model, firmware, and application code must all fit into this storage.

This leads to:

- Requirement of small model file size
- Use of compressed model formats (e.g., .tflite)
- Removal of unnecessary libraries and code

3.4 Energy and Power Constraints

Many embedded devices operate on batteries or energy harvesting methods. Continuous high computation drains power quickly.

Important considerations:

- AI inference must be energy-efficient

- Device may operate in sleep mode and wake periodically
- Lightweight processing increases battery life

Energy efficiency is one of the main reasons for preferring Edge AI over cloud communication.

3.5 Real-Time Processing Requirements

Embedded systems are often used in applications where immediate response is required, such as:

- Industrial fault detection
- Health monitoring devices
- Autonomous vehicles and drones

The system cannot afford long delays. AI models must provide results within milliseconds.

Table1: Embedded platforms face several limitations:

Constraint	Description
Limited RAM	Often less than 512 KB in microcontrollers
Low Processing Power	MHz range processors, no GPU
Limited Storage	Few MB flash memory
Energy Constraints	Battery operated devices
Real-Time Requirements	Fast response needed

4. Lightweight Neural Network Models for Edge

Running traditional deep neural networks such as VGG, ResNet, or large CNNs on embedded devices is almost impossible due to memory and computation limits. Therefore, researchers have designed **lightweight neural network architectures** that require fewer parameters, less memory, and lower computation while still providing acceptable accuracy. These models are specially created for mobile, IoT, and embedded Edge AI applications.

Lightweight models reduce the number of layers, parameters, and operations, and often use clever design techniques like **depthwise separable convolutions, bottleneck layers, and parameter sharing**.

4.1 Design Principles of Lightweight Models

Lightweight neural networks are built using some common principles:

- Reduce number of parameters
- Replace standard convolutions with efficient operations
- Use smaller input image sizes
- Use fewer filters and layers
- Optimize for integer arithmetic instead of floating point

These principles help the model to fit within small RAM and storage of embedded devices.

4.2 MobileNet Family

MobileNet is one of the most popular lightweight CNN architectures designed for edge and mobile devices.

Key idea: Use **depthwise separable convolution** instead of standard convolution.

This reduces computation by almost **8 to 9 times** compared to traditional CNN.

Versions:

- MobileNet V1 – Basic depthwise separable convolutions
- MobileNet V2 – Introduced inverted residuals and linear bottlenecks
- MobileNet V3 – Further optimized for speed and power efficiency

Use cases: Image classification, object detection on embedded cameras.

4.3 SqueezeNet

SqueezeNet is designed to achieve AlexNet-level accuracy with **50x fewer parameters**.

Key idea: Use “fire modules” that squeeze and expand channels to reduce parameters.

Advantages:

- Very small model size (<1 MB)
- Suitable for microcontrollers with limited flash memory

4.4 ShuffleNet

ShuffleNet uses **pointwise group convolution** and **channel shuffle** techniques to reduce computation cost.

Advantages:

- Very low computation requirement
- Efficient for low-power CPUs

Often used in mobile vision applications.

4.5 EfficientNet-Lite

EfficientNet models scale depth, width, and resolution in a balanced way. EfficientNet-Lite is a modified version for edge devices.

Advantages:

- Better accuracy-to-size ratio
- Optimized for TensorFlow Lite deployment

4.6 TinyML Models

TinyML refers to machine learning models that can run on microcontrollers with KB-level memory.

Examples:

- Keyword spotting (voice recognition)
- Gesture recognition using accelerometer data
- Simple anomaly detection models

These models are usually shallow neural networks or small CNNs designed to run in less than 100 KB RAM.

Table 1: Some neural network models are specially designed for embedded usage:

Model	Purpose	Advantage
MobileNet	Image classification	Low computation cost
SqueezeNet	Vision tasks	Very small model size
TinyML models	Sensor data processing	Fits in microcontrollers
EfficientNet-Lite	Edge vision	Balanced accuracy and size

5. Model Optimization Techniques

5.1 Quantization

Quantization reduces precision of weights from 32-bit floating point to 8-bit or even 4-bit integers. This reduces memory usage and speeds up computation.

5.2 Pruning

Pruning removes unnecessary neurons or weights from the model without affecting accuracy much.

5.3 Knowledge Distillation

A small model (student) is trained to mimic a large model (teacher).

5.4 Weight Sharing

Multiple connections share same weight values to reduce storage.

6. Hardware Support for Edge AI

Some hardware platforms now provide AI acceleration:

Hardware	Feature
Google Coral TPU	Edge AI accelerator
NVIDIA Jetson Nano	GPU based inference
ARM Ethos-U	MicroNPU for Cortex-M
ESP32-S3	AI instruction support

These hardware components make AI inference faster on embedded devices.

7. Software Frameworks for Edge AI

Several frameworks help developers deploy AI on small devices:

Framework	Use
TensorFlow Lite Micro	Microcontroller AI
Edge Impulse	TinyML development
PyTorch Mobile	Edge deployment
OpenMV	Vision on microcontrollers

8. Applications of Edge AI on Embedded Platforms

8.1 Smart Agriculture

Soil sensors with AI predict irrigation needs.

8.2 Healthcare Wearables

Heart rate and ECG monitoring with on-device processing.

8.3 Smart Surveillance

Local face detection without sending video to cloud.

8.4 Industrial Monitoring

Vibration sensors detect machine faults in real time.

9. Challenges in Edge AI Deployment

- Accuracy loss after model compression
- Thermal and power limitations
- Debugging difficulty in embedded environment
- Security of AI models on device
- Lack of standardized tools

10. Comparison: Cloud AI vs Edge AI

Feature	Cloud AI	Edge AI
Latency	High	Very Low
Privacy	Low	High
Bandwidth	High usage	Minimal
Power	Not device dependent	Critical
Reliability	Needs internet	Works offline

11. Future Directions

- Development of ultra-light AI models
- Better TinyML toolchains
- Integration of AI accelerators in microcontrollers
- Edge federated learning
- Improved energy-efficient designs

12. Conclusion

Edge AI on resource-constrained embedded platforms is becoming very important with growth of IoT devices. By using model optimization techniques like quantization and pruning, and with help of lightweight neural networks and hardware accelerators, AI inference is now possible even on small devices. Although there are challenges related to memory, power and

accuracy, continuous research is making this field more practical. Edge AI will play a major role in future smart systems where real-time and privacy aware processing is needed.

References

1. Lane, N., et al., "DeepX: A Software Accelerator for Low-Power Deep Learning Inference on Mobile Devices," IPSN, 2016.
2. Han, S., et al., "Deep Compression: Compressing Deep Neural Networks," ICLR, 2016.
3. Howard, A., et al., "MobileNets: Efficient CNNs for Mobile Vision Applications," arXiv, 2017.
4. Banbury, C., et al., "Benchmarking TinyML Systems," arXiv, 2020.
5. Warden, P., Situnayake, D., *TinyML*, O'Reilly Media, 2019.
6. TensorFlow Lite Micro Documentation, Google, 2022.
7. Reddi, V., et al., "MLPerf Tiny Benchmark," IEEE Micro, 2021.
8. Sze, V., et al., "Efficient Processing of Deep Neural Networks," Morgan & Claypool, 2017.
9. ARM, "Ethos-U MicroNPU Technical Overview," ARM Whitepaper, 2021.
10. Edge Impulse, "Deploying ML on Embedded Devices," Technical Guide, 2023.