DEPLOYING AI ON RESOURCE-CONSTRAINED DEVICES, COMPRESSION, AND FEDERATED LEARNING AT THE EDGE.

Mrs. S. Rani[1]
[1]Assistant Professor, Annamalai University
Thirunelvelli

Abstract

The rapid expansion of artificial intelligence (AI) applications to the edge of networks presents new opportunities for real-time, privacy-aware processing. However, deploying AI on resource-constrained devices such as microcontrollers and IoT nodes introduces critical challenges due to limited memory, computational capacity, and energy. This paper investigates state-of-the-art compression techniques and federated learning frameworks that enable efficient, scalable, and secure AI deployment at the edge. We analyze model pruning, quantization, knowledge distillation, and novel federated optimization strategies that reduce the dependency on cloud computation while preserving data privacy and reducing communication overhead. Through comparative evaluation and real-world case studies, we highlight how synergizing these approaches empowers edge AI for healthcare, smart homes, and industrial IoT systems.

Keywords

Edge AI, Resource-Constrained Devices, Model Compression, Federated Learning, TinyML, Edge Computing, On-Device Intelligence, IoT.

1. Introduction

Deploying Artificial Intelligence (AI) on edge devices has become increasingly important in recent years, particularly with the rise of the Internet of Things (IoT) and the need for real-time processing and analysis of sensor data. An Edge AI Deployment Model refers to the architecture and process by which AI models are trained, optimized, and deployed to run directly on edge devices (e.g., mobile phones, Raspberry Pi, IoT sensors) instead of relying entirely on cloud infrastructure. This enables low-latency, offline inference, and improved privacy and bandwidth efficiency. Edge devices, such as smartphones, smart home devices, and industrial sensors, are capable of running AI models that can make decisions and take actions locally, without relying on cloud connectivity. This reduces latency, improves security, and enables more efficient use of network resources.

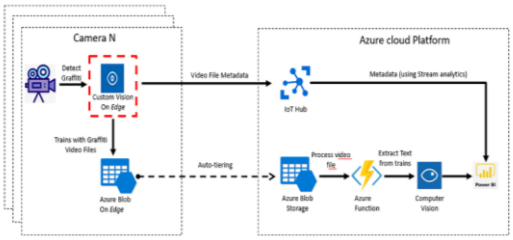

Fig 1. Architecture

As the demand for intelligent real-time applications grows, deploying AI models directly on *resource-constrained devices* (e.g., microcontrollers, smartphones, wearable's, IoT sensors) has emerged as a critical trend. Deploying compact models on affordable hardware enables access to intelligent systems where cloud infrastructure is limited or non-existent. AI processing at the edge eliminates the need to send data to the cloud, enabling instantaneous inference and decision-making — essential for Autonomous vehicles Medical alert systems (e.g., fall detection, arrhythmia monitoring) , Industrial automation. By processing sensitive data locally, edge AI avoids transmitting it to centralized servers, significantly reducing the risk of data breaches. IoT deployments often operate in environments with limited or intermittent internet access. Local inference enables models to adapt to the specific context or user behavior.

There are several deployment model types for edge AI, each suited to different performance and

resource requirements. On-device inference is the most autonomous approach, where the AI model is fully executed on the edge device without relying on external communication, ideal for applications requiring low latency and privacy. In contrast, cloud-assisted edge AI involves partial processing at the edge, with heavier tasks offloaded to the cloud—balancing responsiveness with computational efficiency. Federated learning enables multiple edge devices to collaboratively train a shared global model while keeping data local, preserving privacy and reducing the need for raw data transmission. Another emerging model is split computing, where AI model layers are distributed between the device and the cloud to optimize performance and energy use. Each of these deployment models addresses specific trade-offs between latency, accuracy, power consumption, data privacy, and infrastructure complexity.

Model Type	Description
On-device Inference	AI runs fully on the device, no internet needed
Cloud-assisted Edge AI	Preprocessing on edge, heavy lifting in cloud
Federated Learning	Collaborative training using multiple devices
Split Computing	Neural network layers split across device and cloud

Table 1. 1: Deployment Model Types

2. Challenges

Despite the growing interest in edge AI, deploying machine learning models on resource-constrained devices presents significant challenges across multiple technical dimensions:

2.1. Latency Constraints

While edge deployment aims to reduce latency by enabling local inference, limited computational power may lead to delays in real-time processing, especially for deep learning models with complex architectures. Achieving fast response times under tight performance margins remains a central concern.

2.2. Limited Memory and Storage

Most edge devices, such as microcontrollers and low-power embedded systems, have severely restricted memory (e.g., kilobytes to a few megabytes of RAM) and storage space. Deploying standard AI models requires significant optimization through techniques like model pruning, quantization, and weight sharing to fit within hardware limitations without compromising accuracy.

2.3. Energy and Power Efficiency

Edge devices often rely on batteries or energy-harvesting sources, making power consumption a critical challenge. Running AI models locally, especially for continuous sensing or real-time inference, must be highly energy-efficient to ensure long operational lifespans.

2. 4. Connectivity and Network Dependency

While one of the goals of edge AI is to reduce cloud dependency, some architectures still rely on periodic synchronization or updates from central servers. In low-connectivity or remote environments, maintaining model updates, security patches, or federated learning participation becomes difficult.

2. 5. Data Privacy and Security

Handling sensitive data (e.g., personal health records or audio from home assistants) at the edge introduces new risks if the device is not secured properly. In addition, implementing privacy-preserving techniques such as differential privacy, secure aggregation, and encrypted communication is computationally expensive and difficult to integrate into lightweight devices.

Challenges	***Description***
Latency	*Inference must be fast despite limited compute resources*
Memory	*Models must fit within limited RAM and flash storage*
Power	*Must operate under strict energy budgets, often battery-powered*
Connectivity	*Needs to handle offline mode or limited bandwidth scenarios*
Privacy	*Sensitive data requires strong local protection without heavy overhead*

Table 2.1: Summary Table of Key Challenges

3. Resource-Constrained Edge Environments

Resource-constrained edge environments refer to computational settings where devices operate with limited hardware capabilities, including restricted processing power, memory, storage, and energy supply. Typical examples include microcontrollers, IoT sensors, embedded systems, smartphones, and wearable devices deployed in fields such as smart homes, healthcare, agriculture, and industrial automation. These devices are designed to be low-cost, portable, and energy-efficient, often powered by batteries or energy-harvesting technologies. However, these benefits come at the cost of limited support for running computationally intensive tasks like deep learning inference. Consequently, deploying AI models in such environments demands careful consideration of model size, runtime efficiency, and hardware compatibility. The dynamic and often remote nature of edge settings further introduces challenges related to network connectivity, real-time performance, and data security. Designing AI systems for these environments requires lightweight architectures, optimized algorithms, and robust mechanisms for local inference, model updates, and minimal energy consumption.

3.1 Edge Devices

- Raspberry Pi

A versatile single-board computer (SBC) with Linux-based OS support, widely used in prototyping, education, and light edge computing tasks. Equipped with quad-core ARM processors, moderate RAM (up to 8GB), and USB/HDMI interfaces, Raspberry Pi is suitable for applications like image classification, home automation, and low-power AI inference using accelerators like Google Coral Edge TPU.

Fig 3.1 Raspberry Pi

- STM32 Microcontrollers

Developed by STMicroelectronics, STM32 chips are ultra-low-power microcontrollers (MCUs) based on ARM Cortex-M cores. They are commonly used in battery-operated embedded systems, offering limited RAM and flash storage but exceptional energy efficiency. STM32 devices are ideal for running tiny machine learning models via Tiny ML frameworks for applications such as vibration analysis, gesture recognition, and predictive maintenance.

Fig 3.2 STM32 *Microcontrollers*

- Mobile SoCs

Found in smartphones and tablets, mobile SoCs like Qualcomm Snapdragon, Apple A-series, and MediaTek chips offer advanced processing power with integrated AI engines (e.g., NPU, DSP, GPU). These platforms can run complex neural networks in real time, making them ideal for mobile computer vision, speech processing, and augmented reality applications.

3.2 Typical Limitations

Edge devices deployed in real-world scenarios often operate under significant hardware constraints:

- Compute Constraints

Most edge devices lack the processing power of general-purpose CPUs or GPUs found in data centers. Microcontrollers and lightweight SoCs generally operate at lower clock frequencies (tens to hundreds of MHz) and may not support floating-point operations or parallel processing. This severely limits the complexity and depth of neural network models that can be executed in real time.

- Memory Limitations

Many edge devices are equipped with only a few kilobytes to a few megabytes of RAM and flash storage. This restricts the size of AI models and

the volume of intermediate data that can be processed and stored, necessitating techniques such as quantization, pruning, and on-the-fly inference without data buffering.

- Power Constraints

A significant proportion of edge deployments rely on battery-powered or energy-harvesting sources. As a result, devices must minimize power usage to ensure long-term operation. Frequent wireless communication or heavy computations can quickly drain available energy, making ultra-low-power design a key requirement.

Limitation	Description	Mitigation Strategy
Compute	Low processing speed, limited cores	Model quantization, efficient algorithms
Memory	Small RAM/flash footprints	TinyML, pruning, streaming inference
Power	Battery/energy-harvested systems	Duty cycling, low-power sleep modes
Application Domain	**Edge Use Case**	**Resource Challenge**
Smart Agriculture	Crop monitoring, environmental sensing	Remote access, low power, minimal compute
Wearables	Health analytics, activity tracking	Size, power, and real-time constraints
Surveillance	Real-time object/person detection	High data input, low latency requirements
Smart Homes	Voice, energy, motion control	Always-on, minimal latency, privacy
Industrial IoT	Predictive diagnostics	Harsh environments, reliable uptime

Table 3.2 Summary Table

4. Model Compression Techniques

To enable the deployment of AI models on devices with limited compute, memory, and power, various model compression techniques are employed. These methods aim to reduce the model size and computational load while preserving accuracy as much as possible.

4.1 Pruning

Pruning eliminates less important weights or neurons from a model, creating a sparse architecture that requires fewer operations and less memory.

Unstructured Pruning: Removes individual weights based on magnitude. Structured Pruning: Removes entire neurons, channels, or layers—more hardware-efficient.

4.2 Quantization

Quantization reduces the precision of weights and activations from 32-bit floating-point to lower bit-width formats (e.g., 8-bit integers). This significantly reduces memory footprint and allows faster inference on processors with integer arithmetic support.

Post-Training Quantization: Applied after training; fast but may affect accuracy. Quantization-Aware Training (QAT): Models are trained with quantization in mind, maintaining higher accuracy.

4.3 Knowledge Distillation

This technique involves training a smaller model (student) to replicate the behavior of a larger, more accurate model (teacher). The student model learns not just the final predictions but also the softened output distributions of the teacher, preserving performance in a compact form.

These compression techniques are foundational in TinyML and Edge AI applications, where balancing model performance with resource constraints is essential. Combining multiple methods often yields the best results.

5. Federated Learning at the Edge

Federated Learning (FL) is a decentralized machine learning paradigm that enables multiple edge devices to collaboratively train a shared global model without exchanging raw data. Instead of transmitting sensitive user or sensor data to a central server, each edge device—such as smartphones, IoT sensors, or embedded systems—trains a local copy of the model using its own private data and only shares model updates (e.g., gradients or weights) with a central aggregator. The server then aggregates these updates (typically using algorithms like FedAvg) and distributes the updated global model back to the devices.

5.2 Federated Averaging (FedAvg) Baseline algorithm

This algorithm used in federated learning to aggregate local model updates from multiple edge clients into a single global model. Introduced by Google in 2017, FedAvg is designed to be communication-efficient and scalable across a large number of devices with heterogeneous data and hardware constraints. At a high level, the algorithm proceeds in communication rounds as follows:

Initialization:

A central server initializes a global model $\boldsymbol{\omega}_0$ and sends it to a subset of edge devices (clients).

Local Training:

Each selected client k updates the model using its local data D_k by performing multiple epochs of stochastic gradient descent (SGD):

$$w_{t+1} = \sum_{k=1}^{K} \frac{n_k}{n} w_k^{t+1}$$

where η is the local learning rate and $F_k(w)$ is the loss function on client k's data.

Model Upload:

Clients send their updated models $w_k{}^{t+1}$ back to the server.

Aggregation:

The server performs a weighted average of the local models to form the new global model:

$$w_{t+1} = \sum_{k=1}^{K} \frac{n_k}{n} w_k^{t+1}$$

where n_k is the size of client k's dataset , $n=\sum n_k$ is the total data size.

Repeat:

The process is repeated for multiple rounds until convergence.

Advantages

Reduces the number of communication rounds by using local SGD with multiple steps. Works well with non-IID (non-identically distributed) and unbalanced data across devices. No raw data leaves the client devices.

Limitations

Performance may degrade when client data distributions are highly skewed. Devices with poor connectivity or limited compute may delay rounds. Requires robust handling of client availability and participation.

5.3 Communication Optimization

Communication between edge devices and the central server is often the primary bottleneck, particularly in resource-constrained settings like IoT or mobile edge networks. Since frequent transmission of model updates can consume substantial bandwidth and energy, communication optimization is essential for practical and scalable deployment.

Strategy	**Communication Reduction**	**Accuracy Trade-off**	**Ideal Use Case**
Quantization	4× to 16×	Low to moderate	Mobile FL, low-bandwidth edge
Sparsification	10× to 100×	May reduce accuracy	Sensor networks
Local Epochs (FedAvg)	Fewer rounds needed	Balanced	Most edge FL applications
Hierarchical Aggregation	Reduced latency, energy	Minimal	Smart cities, industrial IoT
Asynchronous FL	Better device utilization	Requires robust logic	Highly heterogeneous systems

Table 5.3 Impact of Optimization

5.4 Security and Privacy

Federated Learning (FL) enhances privacy by ensuring that raw data remains local to edge devices such as smartphones, wearable's, and IoT sensors. However, despite this decentralized approach, FL systems are still vulnerable to security breaches and privacy attacks at multiple stages of the training process.

Threat Type	Defense Mechanism	Notes
Inference Attacks	Differential Privacy, LDP	Reduces information leakage
Poisoning Attacks	Robust Aggregation	Filters outliers and attackers
Man-in-the-Middle	Secure Aggregation + TLS	Encrypts and anonymizes updates
Client Impersonation	Authentication, Sybil Defense	Trust validation, ID verification
Model Inversion	TEEs, DP, Compression	Obfuscates gradient

Table 5.4 Summary Table of Security and Privacy Mechanisms

6. Integrated Frameworks and Toolkits

Deploying AI models on resource-constrained devices and implementing federated learning at the edge requires specialized frameworks and toolkits that support model optimization, compression, training, and secure communication. Several mature and emerging tools facilitate development, testing, and deployment in such environments.

Work / Toolkit	Focus Area	Key Features	Supported Devices
TensorFlow Lite	Model compression & deployment	Quantization, pruning, microcontroller support	Smartphones, STM32, ESP32
TensorFlow Federated	Federated learning simulation	Custom aggregation, privacy features	Research & prototyping on edge
PyTorch Mobile & PySyft	Edge deployment & FL with privacy	Secure computation, FL, DP	Mobile, embedded devices
STM32Cube.AI	Embedded AI deployment	Code generation, quantization	STM32 MCUs
Flower & FedML	Federated learning frameworks	Flexible client-server FL systems	Heterogeneous edge networks

Table 6.1 Integrated Framework

7. Challenges and Future Directions

Deploying AI on resource-constrained devices, combined with model compression and federated learning at the edge, presents several significant challenges and promising future directions. Resource limitations such as limited processing power, memory, and battery life restrict the complexity of models that can be deployed on devices like microcontrollers and mobile SoCs. Additionally, the heterogeneity of edge environments—in terms of hardware capabilities, network connectivity, and non-IID data distributions—complicates efficient and effective model training and deployment. Communication overhead remains a major bottleneck in federated learning due to constrained bandwidth and intermittent connectivity, necessitating advanced compression and communication optimization techniques. Privacy and security concerns also persist, as model updates can inadvertently leak sensitive information, and edge devices are vulnerable to attacks like data poisoning and tampering. Scalability is challenged by inconsistent client participation and the need for robust aggregation methods. Looking ahead, future research is focused on developing ultra-efficient model compression techniques, hierarchical federated learning architectures, and lightweight yet strong privacy-preserving mechanisms tailored for edge hardware. Adaptive client selection, incentive mechanisms, and cross-platform frameworks will improve participation and interoperability, while integration with emerging AI accelerators promises enhanced performance. Overall, addressing these challenges through a combination of hardware-software co-design, novel algorithms, and security protocols will be key to unlocking the full potential of AI at the edge.

8. Conclusion

Deploying AI on resource-limited edge devices is viable with compression and federated learning, offering a pathway to real-time, decentralized, and privacy-preserving intelligent systems. Future advancements in adaptive models, energy optimization, and edge-cloud collaboration will further unlock potential in diverse sectors.

9. References

[1]. Han, S., Mao, H., & Dally, W. J. (2016). Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. ICLR.

[2]. McMahan, B., Moore, E., Ramage, D., et al. (2017). Communication-efficient learning of deep networks from decentralized data. AISTATS.

[3]. Lane, N. D., & Warden, P. (2018). The deep (learning) transformation of mobile and embedded computing. IEEE Computer.

[4]. Liang, T., Glossner, J., Wang, L., Shi, S., & Zhang, X. (2021). Pruning and Quantization for Deep Neural Network Acceleration: A Survey. Neurocomputing, 461, 1–18.

[5]. Lee, J., Mukhanov, L., Molahosseini, A. S., Minhas, U., Hua, Y., Martinez del Rincon, J., Dichev, K., Hong, C.-H., & Vandierendonck, H. (2021). Resource-Efficient Deep Learning: A Survey on Model-, Arithmetic-, and Implementation-Level Techniques.

[6]. Geng, X., Wang, Z., Chen, C., Xu, Q., Xu, K., Jin, C., Gupta, M., Yang, X., Chen, Z., Aly, M. M. S., Lin, J., Wu, X., Li, X. (2024). From Algorithm to Hardware: A Survey on Efficient and Safe Deployment of Deep Neural Networks.

[7]. Mishra, R., Gupta, H. P., & Dutta, T. (2020). A Survey on Deep Neural Network Compression: Challenges, Overview, and Solutions.

[8]. Dantas, P.V., Sabino da Silva, W., Cordeiro, L.C., Applied Intelligence,(2024)."A Comprehensive Review of Model Compression Techniques in Machine Learning".

[9]. Gu, L., Du, Y., Zhang, Y., et al., arXiv preprint, 2024, "Lossless Compression of Deep Neural Networks: A High-dimensional Neural Tangent Kernel Approach".

[10]. Liu, W., Zhang, M., Shi, C., et al., Neural Processing Letters, 2024. "Deep Convolutional Neural Network Compression Method: Tensor Ring Decomposition with Variational Bayesian Approach"