Computers, Materials & Continua

Tech Science Press

Doi:10.32604/cmc.2025.062819

REVIEW

A Literature Review on Model Conversion, Inference, and Learning Strategies in EdgeML with TinyML Deployment

Muhammad Arif[1,*] **and Muhammad Rashid**[2]

[1]Department of Computer Science and Artificial Intelligence, Umm Al-Qura University, Makkah Al-Mukarama, 21955, Saudi Arabia
[2]Department of Computer and Network Engineering, Umm Al-Qura University, Makkah Al-Mukarama, 21955, Saudi Arabia

*Corresponding Author: Muhammad Arif. Email: mahamid@uqu.edu.sa

Received: 28 December 2024; Accepted: 13 February 2025; Published: 26 March 2025

ABSTRACT: Edge Machine Learning (EdgeML) and Tiny Machine Learning (TinyML) are fast-growing fields that bring machine learning to resource-constrained devices, allowing real-time data processing and decision-making at the network's edge. However, the complexity of model conversion techniques, diverse inference mechanisms, and varied learning strategies make designing and deploying these models challenging. Additionally, deploying TinyML models on resource-constrained hardware with specific software frameworks has broadened EdgeML's applications across various sectors. These factors underscore the necessity for a comprehensive literature review, as current reviews do not systematically encompass the most recent findings on these topics. Consequently, it provides a comprehensive overview of state-of-the-art techniques in model conversion, inference mechanisms, learning strategies within EdgeML, and deploying these models on resource-constrained edge devices using TinyML. It identifies 90 research articles published between 2018 and 2025, categorizing them into two main areas: (1) model conversion, inference, and learning strategies in EdgeML and (2) deploying TinyML models on resource-constrained hardware using specific software frameworks. In the first category, the synthesis of selected research articles compares and critically reviews various model conversion techniques, inference mechanisms, and learning strategies. In the second category, the synthesis identifies and elaborates on major development boards, software frameworks, sensors, and algorithms used in various applications across six major sectors. As a result, this article provides valuable insights for researchers, practitioners, and developers. It assists them in choosing suitable model conversion techniques, inference mechanisms, learning strategies, hardware development boards, software frameworks, sensors, and algorithms tailored to their specific needs and applications across various sectors.

KEYWORDS: Edge machine learning; tiny machine learning; model compression; inference; learning algorithms

1 Introduction

Deep Learning (DL) has emerged as a new Machine Learning (ML) paradigm, capable of automatically learning complex data representations at multiple levels of abstraction [1]. At the same time, the processing and communication capabilities of embedded devices have tremendously increased [2]. As a result, the term Internet of Things (IoT) has emerged that refers to a network of interconnected embedded devices [3]. However, IoT devices generate vast amounts of data and have limited resources. Therefore, cloud computing is integrated into IoT frameworks to provide the required computational and storage capacity. While the cloud offers essential processing power, cloud-stored data may not always be secure [4]. Consequently, IoT frameworks utilize edge computing, which relies on devices that can perceive their environment and process data locally. In this context, Edge Machine Learning (EdgeML) extends edge computing by directly

integrating ML and DL capabilities into edge devices. This recent ML paradigm shifts all or part of the machine learning computation from the cloud to edge devices [5].

Fig. 1 illustrates a typical EdgeML architecture with three layers. The edge layer (front end) is equipped with sensors and signal-processing algorithms. This layer is responsible for data collection and processing. As a result, inference and training of local models take place. It connects to the edge computing layer (near end) via wireless communication. The edge computing layer acts as an intermediary, linking edge devices to the cloud. It supports inference and learning of new data in collaboration with edge devices. It may involve splitting the model across multiple devices for collaborative (distributed) inference or running the entire model on a single device. If an edge device lacks sufficient resources, it can collaborate with other edge devices or the edge server [5,6]. Its key advantages include enhanced data privacy and security, as data is not transmitted to a centralized server. Additionally, the failure of a single device has minimal impact on the learning process, and scalability is easily managed by engaging multiple devices as needed. Strategies for distributed learning on edge devices include federated learning, model splitting/partitioning, and hierarchical clustering learning [7].

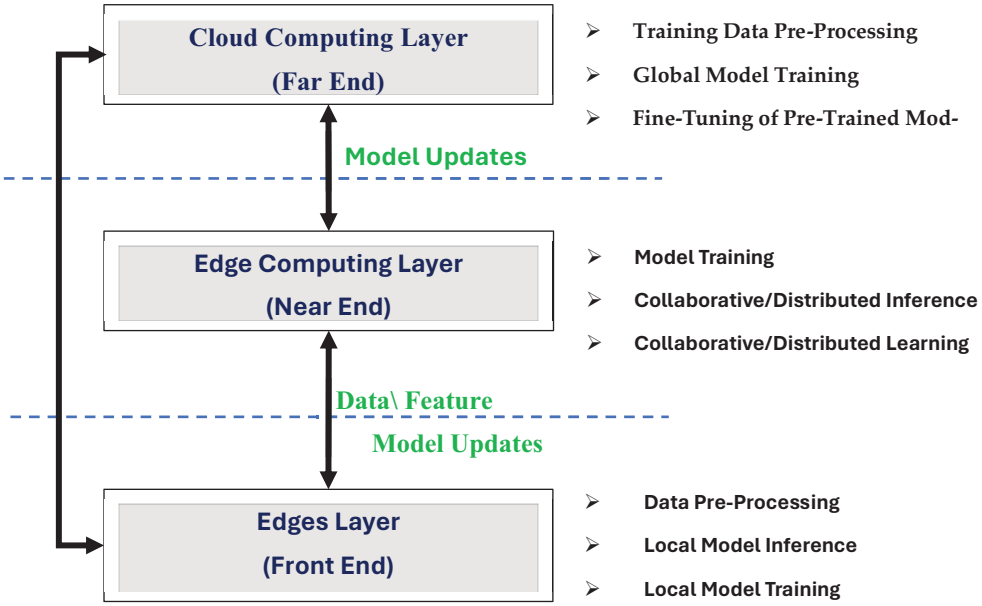

Figure 1: A typical three-layer edge machine learning architecture

While the edges layer and edge computing layer perform local and distributive model training, a cloud computing layer (far end) contains extensive computational resources for global model training and running high-demand algorithms using pre-processed data. Moreover, pre-trained models can be fine-tuned for specific tasks but often require model conversion to reduce complexity for edge device deployment [8]. Consequently, there are various distributed model design and deployment strategies. These distributed or collaborative strategies must be compared in terms of latency, privacy, security, reliability, energy consumption, and computational capabilities.

From the above discussion, it can be concluded that the EdgeML framework involves three main steps. The first step is model conversion or compression, which creates smaller models or converts complex models into simpler ones through pruning, quantization, and knowledge distillation. The model's complexity is tailored to the edge device's resources. The second step is optimized inference that minimizes resource

usage on edge devices. Finally, protocols such as collaborative learning across multiple devices should be implemented to update the model if further learning is needed.

In addition to three major steps in EdgeML, another trend is performing complex processing tasks entirely on edge devices [2]. TinyML (Tiny Machine Learning) enables ML and DL algorithms to run on tiny devices like microcontrollers [9]. TinyML architecture focuses on designing memory-efficient models for edge devices, ensuring high performance. It has led to the concept of the Internet of Intelligent Things (IoIT), which combines embedded hardware, wireless networking, and artificial intelligence. TinyML frameworks rely on specialized sensors, software frameworks, and tools for developing, training, and deploying models on resource-constrained devices [10].

1.1 Motivation for the Review and Limitations of Existing Reviews

Given the variety of model conversion techniques in EdgeML, and the importance of scalable and efficient inference mechanisms and learning strategies for large-scale deployments, conducting a literature review on these topics can enhance understanding of performance improvements in speed, latency, and energy efficiency. Additionally, it can offer a comprehensive overview of TinyML applications across different domains, exploring the latest advancements in hardware, software, and sensor technologies. The limitations of state-of-the-art review articles have been highlighted in Table 1. These limitations reveal that there is no comprehensive literature review that covers model conversion, inference, and learning in EdgeML and TinyML deployment at the same time. Therefore, by synthesizing existing research, the literature review can identify knowledge gaps and suggest future research directions in the most promising areas of EdgeML and TinyML.

Table 1: Summary of state-of-the-art review articles on EdgeML and TinyML

Ref.	Year	Focus	Limitations
		Edge ML	
[5]	2023	• Examines various edge computing paradigms from various perspectives • Provides an overview of EdgeML for limited computing resources	• Does not review state-of-the-art techniques on model conversion, inference mechanisms, and learning strategies • Does not discuss hardware and software frameworks for the deployment of ML and DL on edge devices
[11]	2024	• Emphasizes the evolution of EdgeML • Pinpoints research opportunities in EdgeML to unify ML and edge computing	
[12]	2023	• Addresses key issues in EdgeML and summarizes optimization techniques	

(Continued)

Table 1 (continued)

Ref.	Year	Focus	Limitations
[13]	2023	• Highlights the potential of edge computing • Describes lightweight ML frameworks • Explores privacy issues	• Does not review techniques to create lightweight ML models and learning mechanisms
[14]	2024	• Compression techniques of DNN • Explore the applicability of compressed models in visual applications	• Does not cover the learning and inference aspects of EdgeML on resource-limited architectures
		TinyML	
[10]	2024	• Classifies model optimization techniques • Hardware and software frameworks Discuss TinyML educational resources	• Does not classify hardware and software frameworks as well as model conversion, inference, and learning strategies
[2]	2024	• Ahistorical evolution of IoIT and reviews TinyML models on embedded devices	
[15]	2024	• Edge computing platforms and their role in the deployment of EdgeML workflows • Reviews frameworks and libraries for ML and models on constrained edge devices	• Only limited to the deployment of EdgeML workflows and does not discuss the deployment of TinyML workflow
[16]	2024	• Systematically reviews TinyML models on embedded devices with hardware and software frameworks	• Only limited to TinyML deployment and does not review model conversion, inference, and learning strategies

1.2 Contributions

The limitations of state-of-the-art review articles, highlighted in Table 1, have been rectified by performing this literature review. Particularly, it has explored the answers to the following five research questions:

Research Question 1: What are the state-of-the-art model conversion techniques used in EdgeML, and how do they impact the performance, efficiency, and deployment of machine learning models on resource-constrained devices?

Research Question 2: What are the current state-of-the-art inference mechanisms used in EdgeML, and how do they compare in performance and efficiency?

Research Question 3: How do different learning strategies impact the performance and efficiency of ML models deployed in edge computing environments?

Research Question 4: What are the key challenges and problems and the associated ML/DL models in deploying TinyML on resource-constrained devices in various sectors?

Research Question 5: What are the latest advancements in hardware, software frameworks, and sensors designed explicitly for TinyML frameworks?

It can be observed from the above research questions that the contribution of this literature review is twofold: (1) a critical review of model conversion techniques, inference mechanisms, and learning strategies in EdgeML, (2) identification of hardware development boards, software frameworks, sensors, ML/DL models and applications across various sectors for the deployment of TinyML models on resource-constrained devices.

Table 2 provides an overview of the literature review on model conversion, inference, and learning in EdgeML with TinyML deployment in different sectors. The process starts by developing a review protocol, selecting 90 research articles from renowned databases, categorized into two main areas: (1) model conversion, inference, and learning in EdgeML and (2) model deployment in six different sectors on resource-constrained devices with TinyML. The first category includes 60 articles, divided into model conversion (40 articles), inference (10 articles), and learning (10 articles). The second category comprises 30 articles, divided into six sectors with five articles each.

Table 2: Overview of the literature review on model conversion, inference, and learning in EdgeML with TinyML deployment in different sectors

Selection of 90 research articles according to a review protocol									
Model conversion, learning, and inference in EdgeML (60 articles)				Model deployment with TinyML in (30 articles)					
IEEE (18)	Elsevier (34)	ACM (4)	Springer (4)	IEEE (15)	Elsevier (6)	ACM (6)	Springer (3)		
Classification of selected research works into subcategories (total = 90)									
Model Conversion (40)	Inference Mechanisms (10)	Learning Strategies (10)	Smart Agri. (5)	Health & En. (5)	Vehic. & Aut. (5)	Indus. & Rob. (5)	Ener. (5)	Secu. (5)	
Synthesis of qualitative and quantitative analysis									
• Pruning • Quantization • Factorization • Knowledge Distillation	• On-device • Distributed	• On-device • Continual • Federated	• Identification and comparison of Hardware Development Boards • Identification of Software Frameworks • Identification of 30 applications (examples) in 6 sectors with models and sensors						

It can be seen in Table 2 that most research in EdgeML centers on model conversion, adapting resource-intensive models for deployment on resource-limited devices. This focus stems from the widespread use of powerful deep-learning models in various real-world applications. However, research into learning and inference mechanisms on edge devices remains challenging and needs further exploration.

The first category's comprehensive analysis covers four model compression techniques (pruning, quantization, low-rank factorization, and knowledge distillation), inference mechanisms (on-device inference and split-model inference), and learning strategies (on-device learning, continual learning, and federated learning). The second category's synthesis provides an in-depth discussion on hardware development boards and software frameworks for deploying TinyML models, identifying 30 applications with corresponding ML/DL models and sensors across six major sectors.

1.3 Organization

This article is organized as follows: Section 2 defines categories and reviews protocol development. The summary of major findings on model compression techniques, inference mechanisms, and learning strategies is presented in Section 3, while the results of TinyML deployment on resource-constrained devices are presented in Section 4. Discussion of results from Sections 3 and 4, addressing research questions, is presented in Section 5. A discussion of important aspects and limitations of the research is presented in Section 6. Exploration of challenges and future research directions are presented in Section 7. Finally, Section 8 concludes the article.

2 Methodology of the Literature Review

In order to obtain the answers to research questions formulated in the introductory part of this article, literature review process guidelines provided in [17] have been used. A typical literature review procesds defines categories and develops a review protocol for selecting research articles. Therefore, Section 2.1 describes the essential background of different categories. Subsequently, various steps of the review protocol are described in Section 2.2.

2.1 Background on Categories

The research articles, selected according to the review protocol, are categorized into two types: (1) model conversion, inference, and learning (2) model deployment on resource-constrained devices.

2.1.1 Model Conversion, Inference, and Learning

As the Introduction section mentions, edge devices have computational power, memory, and energy resource constraints. Therefore, it is essential to design ML and DL models considering these limitations. There are two main approaches to achieve this: (a) develop a model tailored to the edge device's limitations and train it on a large dataset to ensure satisfactory performance across various environments, and (b) adopt a large pre-trained model to make it suitable for edge devices. Moreover, due to limited resources, model inference requires device-specific or problem-specific strategies. Inference can be adapted based on available resources and task requirements. Furthermore, retraining models with new data is essential in some scenarios, optimizing performance over time. Therefore, the following subsections provide essential background on model conversion, inference, and continuous learning.

The main goal of model conversion is to reduce model size without compromising performance. Fig. 2 shows various stages of model conversion, which can be applied individually or in combination. After training the original model on sensor data, parameters can be quantized and pruned. Other techniques to reduce model size include knowledge distillation and low-rank factorization. As shown in Fig. 2, the training data is collected from various sensors based on task requirements. A conventional model is developed using knowledge of the problem, data, and performance needs. DL models, popular for tasks like image processing and object detection, have numerous parameters and require significant computational and memory resources. These models are typically trained on cloud platforms or high-performance machines and then converted into edge device-friendly models using pruning. Pruning removes insignificant or redundant parameters, creating sparser models without significantly reducing performance.

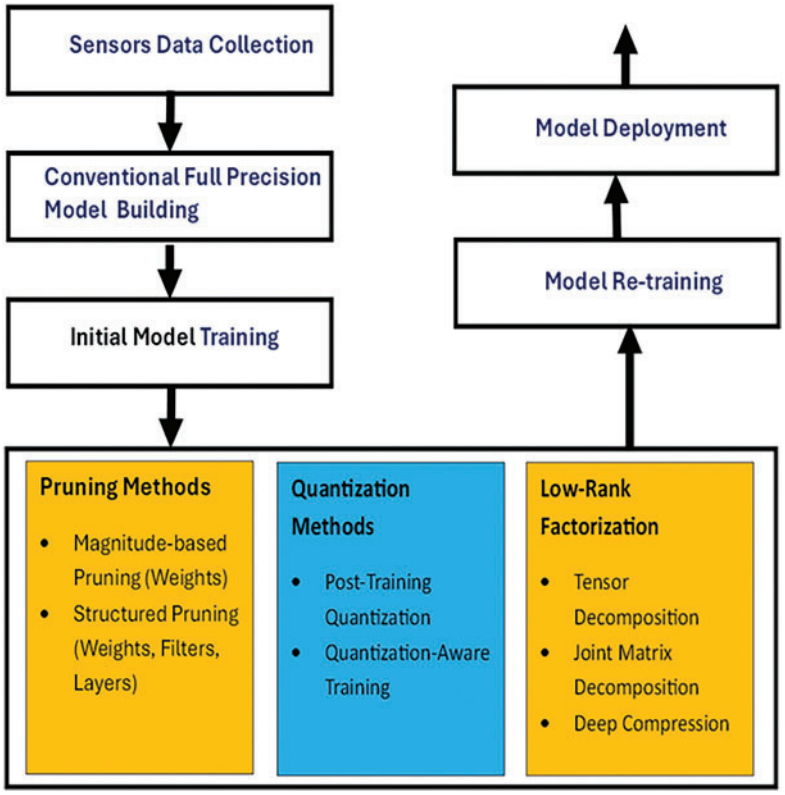

Figure 2: Block diagram of model conversion and compression

While pruning removes less important connections in a model without compromising performance, quantization maps higher bit-width values to lower bit-width values, with accuracy depending on optimal quantization levels. Parameters like weights, biases, and activation functions in deep neural networks can be quantized. On edge devices, gradient and error values may also be quantized. Quantization can be applied to pre-trained models to assess accuracy or performance compromise. Fixed-precision quantization maps 32-bit floating points to k-bit integers, where k is less than 32 bits. For k-bit quantization, the procedure is defined as follows:

$$Q_k(r) = round\left(\frac{r}{\Delta}\right) - Z \tag{1}$$

where Z is an integer zero point value, Δ is the step size, and r is the 32-bit floating point value ranging from α to β. Two simple methods of k-bit quantization are MaxRange method and MinPQE [6]. The MaxRange method selects the step size based on the real value range as follows:

$$\Delta = \frac{\beta - \alpha}{2^k - 1} \tag{2}$$

In contrast, the MinPQE method optimizes step sizes for each neural network layer based on quantization error, improving performance and reducing error through quantization granularity. For instance, 8-bit quantization on the NasNet–A model increases classification error from 4% to 5%. Extreme quantization uses less than 4-bit quantization. The step size, or scaling factor, is crucial for minimizing quantization loss, with much research focused on its optimization. Post-quantization training can enhance accuracy, and pre-training quantization benefits edge devices with limited resources. Low-rank decomposition can reduce the

number of parameters in deep neural networks by approximating the weight matrix to a low-rank structure. However, it may cause significant performance degradation, making it less suitable for model compression.

Fig. 3 explains two types of quantization strategies in model compression. In the quantization-aware training of the model, a quantization policy is defined as one that quantizes the weight, activation function, filters, etc. The model's learning depends on its performance based on the model parameters and the current quantization policy. Hence, the model parameters and the quantization mechanism are updated simultaneously or in batch mode. Meanwhile, in post-training quantization, a model is trained on the dataset first, and then the model parameters are quantized.

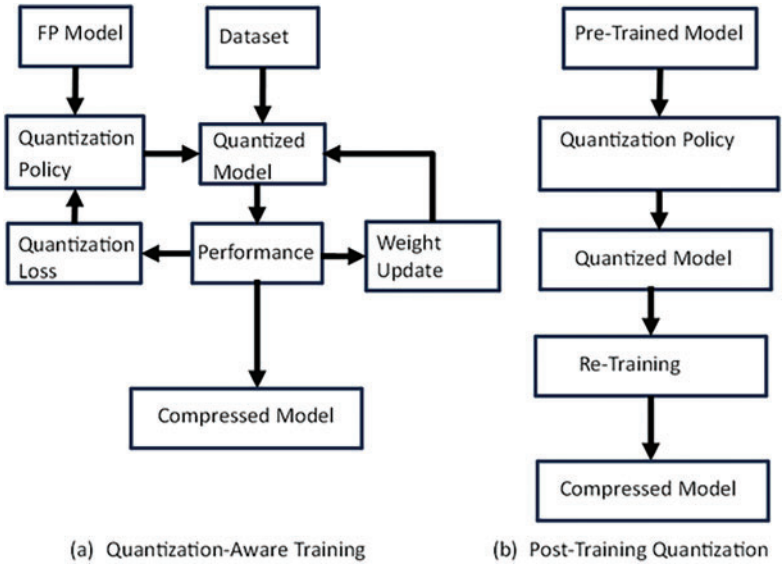

Figure 3: Block diagram of quantization process

A complex teacher model with many parameters is trained on a dataset in knowledge distillation. Once trained and tested, its knowledge is transferred to a smaller, simpler student model. Student models are lightweight, can be deployed on resource-constrained devices, and can be customized according to hardware requirements. This process creates a simpler model with a smaller memory footprint and lower computational requirements. There are three main categories for transferring knowledge: response-based, features-based, and relation-based knowledge distillation [18].

Inference Strategies

Once deployed on an edge device, a machine learning model infers decisions based on its task. Due to limited resources, different inference techniques are used for efficiency [19]. Lightweight models like TinyML or SqueezeNet can perform direct inference on the edge device, offering low latency, no communication requirement, high privacy, and data security. For heavier computation, the model can be split into parts to run on multiple edge devices or servers or in collaboration with the cloud. It preserves data privacy by preprocessing on the edge device and sharing features with the server. Partitioning-based inference methods include data-based partitioning, where input data is divided among devices, and model partitioning, where the model is split among devices or servers.

Learning Strategies

Deep neural network-based solutions on edge or IoT devices are crucial for automation, offering high latency, privacy, and reduced communication bandwidth. High latency enables faster decision-making,

which is essential for many applications. Edge machine learning often uses computational offloading to manage limited resources, transferring tasks to robust edge servers or cloud platforms. Task offloading decisions depend on the network connection, latency requirements, and DL model structure. Various strategies exist for offloading tasks. Federated learning (FL) enhances data privacy on edge devices by training models collaboratively with other devices or servers [20]. Data remains on edge devices, and only model parameters are shared, reducing network bandwidth utilization and latency while improving model generalization on heterogeneous data. Each device trains on a subset of data using stochastic gradient descent method variants. The main server initializes tasks, assigns them to edge devices, and aggregates optimized model parameters to create a global model. This process iterates to minimize global loss, with the server updating and distributing the global model parameters.

2.1.2 Model Deployment on Resource-Constrained Embedded Devices

TinyML enables the deployment of ML and DL models on embedded devices like microcontrollers and single-board computers. This technology makes low-power edge devices "smart," allowing them to perform complex tasks independently. Implementing ML and DL models on these devices is more challenging than the conventional EdgeML approach, where models are hosted in the cloud and run on powerful computers.

Hardware Requirements

It is challenging for a hardware platform to simultaneously meet energy, cost, and processing efficiency. Some platforms may have enough processing power but lack energy efficiency or cost-effectiveness, and *vice versa*. Choosing the right device for a specific application is crucial in the TinyML paradigm. Devices with higher computational power that still consider energy efficiency and cost are called High-end TinyML architectures. Similarly, devices with lower computational power, suitable for small tasks only, are called Low-end TinyML architectures [2]. Examples of high-end TinyML architectures are single-board computers (SBCs), which run entire operating systems and TinyML models in real-time. They are used as sensor nodes and gateways in IoT. On the other hand, low-end TinyML architectures target energy efficiency and cost-effectiveness, making them suitable for more straightforward tasks. This class includes microcontroller units (MCUs) with low-power processors, limited RAM, and simple interfaces.

Software Frameworks

Software frameworks for model deployment offer pre-built tools and libraries, allowing developers to focus on model optimization rather than low-level hardware details. These frameworks optimize ML and DL models for resource-constrained devices, reducing model size and improving inference speed. They enable models to run on different hardware with minimal changes, making it easier to scale TinyML applications across various devices. Commonly used frameworks include Edge Impulse, TensorFlow, and TensorFlow Lite.

Model Types

Different machine learning models are tailored for specific tasks based on the application problem. For example, CNNs (Convolutional Neural Networks) identify objects, faces, and scenes in images. Similarly, time-series models like RNNs (Recurrent Neural Networks) and LSTMs (Long Short-Term Memory Networks) handle sequential data and capture temporal dependencies. Clustering models like K-Means and DBSCAN group similar data points without predefined labels used in anomaly detection.

Applications

TinyML models are ideal for battery-operated devices and real-time decision-making applications. They require small, inexpensive hardware, reducing deployment costs. Their adaptability across different devices

and platforms makes them suitable for scalable applications. Local processing enhances privacy and security by keeping data on the device. Consequently, TinyML models can be deployed in various fields, such as healthcare, agriculture, smart homes, and industrial automation.

Sensors

Different sensors are used in TinyML applications to capture various data types for specific tasks—for example, cameras for image classification and accelerometers for activity recognition. The right sensor ensures accurate and relevant data, improving model performance and reliability. Different environments require different sensors, like soil moisture sensors for agriculture and temperature sensors for smart homes. Power-efficient sensors are suitable for battery-operated devices, maintaining overall power efficiency. Ensuring sensor compatibility with hardware and software platforms ensures seamless integration and better performance of the TinyML system.

2.2 Review Protocol Development

The development of a review protocol is essential in a typical literature review process. The developed protocol in this section contains all the required steps. These steps are: criteria for selecting and rejecting studies (Section 2.2.1), search process (Section 2.2.2), and data extraction & synthesis (Section 2.2.3).

2.2.1 Selection and Rejection Criteria

i. **Subject-Relevant:** Select research only if it is relevant to our research context and supports the answers to our research questions.
ii. **Publication Date (2018–2025):** Select research published between 2020 and 2025 to include the latest studies. Reject any research published before 2018.
iii. **Publisher:** Select research published in one of the four renowned scientific databases: IEEE, Springer, Elsevier, and ACM.
iv. **Crucial Effects**: Select research that significantly affects IIoT development through the EdgeML and TinyML approach.
v. **Results-Oriented**: Select results-oriented research with proposals and outcomes supported by solid facts and experimentation.
vi. **Repetition:** Avoid including identical research within the same context.

2.2.2 Search Process

The selection and rejection criteria outlined in Section 2.2.1 indicate that we have chosen four scientific databases (IEEE, ELSEVIER, SPRINGER, and ACM). These databases include high-impact journals and conference proceedings. Using search terms like Edge AI, TinyML, and EdgeML, we applied a "2018–2025" filter. The search terms and results for each database are summarized in Table 3. If a search term has produced thousands of results, we used advanced search options (e.g., "where abstract contained", "where title contained") provided by these databases to get more precise results. Finally, Fig. 4 shows various steps during the selection of research articles. We specified search terms in four scientific databases and analyzed approximately 43,651 results based on our selection and rejection criteria. We discarded 26,991 studies by title, 9173 by abstract, and 6495 after a general review. Finally, we performed a detailed review of the remaining 992 articles and selected 90 research articles.

Table 3: Details of search terms and search results in selected databases

S. no.	Search term	Number of search results			
		IEEE	Elsevier	Springer	ACM
1.	Edge AI	4077	1012	51	586
2.	EdgeML	1741	857	2	373
3.	TinyML	497	59	31	254
4.	TinyML hardware	161	13	27	216
5.	TinyML applications	273	36	31	250
6.	TinyML software	79	8	24	197
7.	Edge machine learning	5899	2155	717	1189
8.	Internet of intelligent things	7847	1794	42	565
9.	TinyML model	416	44	32	4
10.	TinyML microcontroller	212	23	100	18
11.	Edge quantization	820	227	549	154
12.	Edge pruning	750	176	858	320
13.	Edge low-rank factorization	17	4	223	74
14.	Edge knowledge distillation	348	91	421	66
15.	Edge federated learning	2650	412	504	295
16.	Edge continual learning	94	21	215	35
17.	Edge model splitting	521	212	1537	145

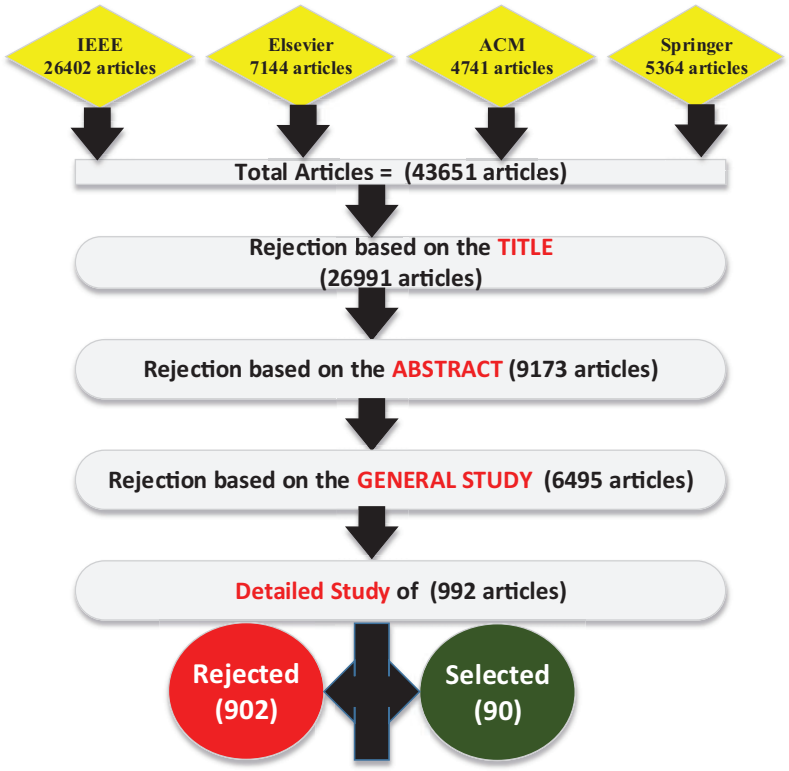

Figure 4: Search process (various steps during the selection of research articles)

2.2.3 Data Extraction and Synthesis

As shown in Table 4, data extraction and synthesis are conducted for selected studies to address our research questions. For data extraction (serial numbers 2 to 6), we gather essential details from each study to ensure they meet the selection and rejection criteria. We perform a detailed analysis for data synthesis (serial numbers 7 to 10), thoroughly examining and categorizing each study. Each study is meticulously reviewed to fit the corresponding category. Statistics of research articles according to the publication year are provided in Fig. 5.

Table 4: Details of data extraction and synthesis

S. no.	Description	Details
1.	Bibliographic information	Title, author, publication year, publisher details, and type of research (i.e., journal or conference)
		Data extraction
2.	Overview	The basic proposal and objective
3.	Results	Results acquired from the selected research
4.	Data collection	Quantitative or qualitative
5.	Assumption	Assumptions (if any) to validate the results
6.	Validation	Validation method used to validate its proposal
		Data synthesis
7.	Model conversion techniques	Table 5: Results of structured pruning methods Table 6: Results of quantization methods Table 7: Results on quantization-aware training Table 8: Results of low-rank factorization Table 9: Results of knowledge distillation
8.	Inference mechanisms	Table 10: Results of inference strategies
9.	Learning strategies	Table 11: Results of continual learning methods Table 12: Results of federated learning methods
10.	TinyML sectors, applications, and models	Table 13: Applications and machine learning models in six identified sectors
11.	Hardware boards	Table 14: Summary of development boards
12.	Software frameworks	Table 15: Distribution of selected research works according to three software frameworks
13.	Sensors	Table 16: Sensors used for the deployment

3 Results on Model Conversion, Inference and Learning

Based on the methodology of Section 2, the selected research studies have been classified into two major categories: (1) model conversion, inference, and learning in EdgeML (2) model deployment on resource-constrained devices with TinyML. Consequently, the results on model conversion (Section 3.1), efficient inference (Section 3.2), and continuous training strategies (Section 3.3) are presented in this section, while the results of model deployment will be presented in Section 4.

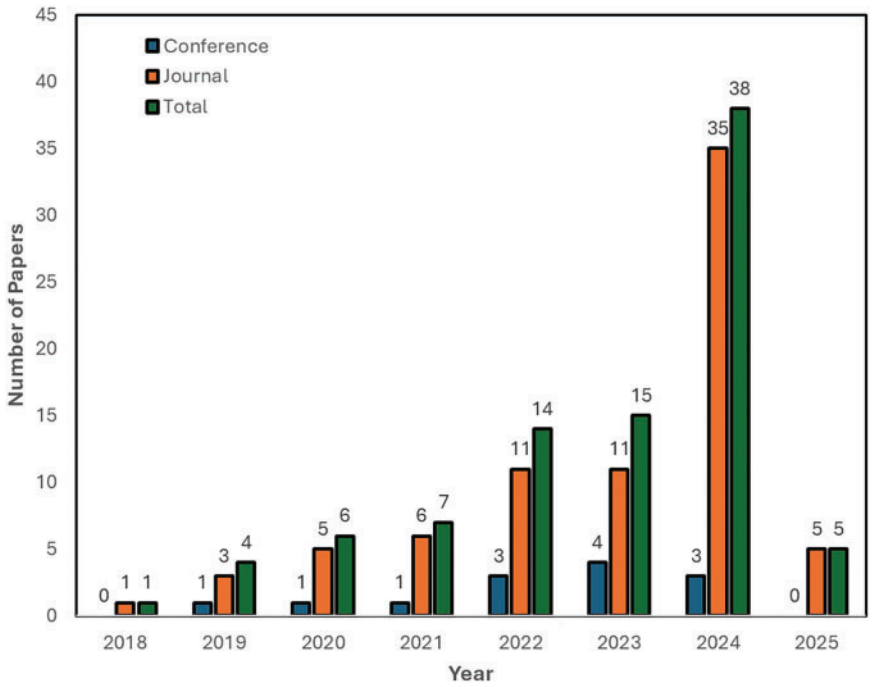

Figure 5: Statistics of research articles according to the publication year

3.1 Model Conversion Methods

Deploying large models on resource-constrained edge devices is impractical. Two options are available: (1) Customize a small DL model with fewer layers and full precision parameters to fit within the edge device's memory and require less computational power, (2) Convert and compress an existing model by removing insignificant parameters and using quantized parameters with lower precision or bit-width to achieve similar performance. The learning process can incorporate quantized gradients if training occurs on edge devices. The following subsections will detail various techniques for model conversion and compression.

3.1.1 Pruning Methods

Pruning methods are divided into structured and unstructured methods. Unstructured pruning creates irregular, sparse models by pruning any weight without constraints, followed by retraining to mitigate performance loss. However, this method requires specific hardware and software redesigns for efficiency. Therefore, Li et al. [21] proposed a flexible rate filter pruning method (structured) using a loss-aware process. Structured pruning removes redundant filters and channels to make the models lighter in size. Filter norms can help identify insignificant filters. For example, low norm filters exhibit low activation in feature maps and can be pruned. The sensitivity of each filter to model performance can be measured to prune unimportant filters [22]. Structured pruning is straightforward for DL models with simple convolutional layers. However, the pruning strategy must consider the consistency of feature maps and residual flows for residual block-based networks. Due to the complexity of DL models in pattern recognition, most research focuses on structured pruning methodologies, as shown in Table 5.

In CNNs, filters in convolutional layers are pruned to improve efficiency. Various methods have been proposed, such as Capped L1-norm balances regularization and filter selection [23]. Yu et al. [24] treat pruning as a nonconvex-constrained optimization problem, using information from a Taylor-based

approximation to quantify filter contributions and prune the network. Another method, proposed by He et al. [22], prunes filters layer by layer using a standard loss function like cross-entropy, followed by fine-tuning to improve performance. This process is repeated for each layer until the entire network is pruned. The goal of these methods [22–24] is to optimize the model's performance while reducing complexity.

Table 5: Results of structured pruning methods on CNN architectures

Ref.	Dataset/ Model	Pruning method	Baseline accuracy	Accuracy	Pruned parameters	Model size reduction	Flops saved
[21]	CIFAR-10 ResNet-110	Flexible pruning	94.2%	94.2%	—	—	64%
[22]	CIFAR-10 VGG-16	Filters	93.6%	93.3%	—	—	52%
[23]	CIFAR-110 VGG-16	Filters	73.4%	73.6%	6.4 M	56%	44%
[23]	CIFAR-110 ResNet-164	Filters	76.8%	76.8%	0.8 M	30%	51%
[24]	CIFAR-10 VGG-16	Hybrid	93.7%	93.5%	0.76 M	94%	73%
[24]	CIFAR-10 ResNet-56	Hybrid	93.9%	90%	0.08 M	90%	87%
[25]	CIFAR-100 VGG-16	Filters	73.9%	73.6%	—	50%	—
[25]	CIFAR-100 ResNet50	Filters	75.9%	74%	—	—	51%
[26]	CIFAR-10 VGG-16	Filters	93.9%	93.4%	—	93%	81%
[26]	CIFAR-10 ResNet-110	Filters	93.2%	93.2%		62%	62%
[27]	CIFAR-100 VGG-16	Filters	73.6%	73.4%	0.85 M	94%	84%
[27]	CIFAR-100 ResNet-50	Filters	71%	70.9%	18 M	76%	76%
[28]	CIFAR-10 VGG-16	Filters	93.9%	92.8%	1.15 M	92%	84%
[28]	CIFAR-10 ResNet-110	Filters	93.5%	93.4%	0.52 M	69.9%	73%
[29]	CIFAR-10 VGG-16	Weights	93.6%	93.2%	—	77%	64%
[29]	CIFAR-10 ResNet-110	Weights	92.5%	92.3%	—	46%	49%

In addition to the works in [22–24], some filter pruning methods are based on the statistics of the feature map generated by the entire training dataset. In this context, Mondal et al.'s method [25] evaluates filter importance for each class separately using the normalized L1-norm of the feature map. Filters generating

dominant patterns for a class are not pruned. Similarly, Sarvani et al.'s knowledge transfer method [26] uses a customized regularizer function to transfer knowledge from pruned filters to important ones, minimizing information loss by adjusting L1-norm values. The mutual information theory method, proposed by Lu et al. [27], prunes filters in two steps. The first step calculates the class relevance of each filter using conditional mutual information on mini-batches. The second step averages class synthesis evaluation criteria (relevance and redundancy) across mini-batches and prunes filters with lesser contributions.

While methods in [25–27] aim to optimize model performance by selectively pruning filters based on their importance and relevance, Wavelet Transform-Based pruning uses cosine similarity and energy-weighted components of high and low frequencies to determine the importance score of each feature map [28]. A multi-objective evolutionary framework by Chung et al. [29] balances performance and efficiency for edge devices using a fitness function based on the pruned model's flops ratio and error rate. It employs three pruning criteria: L1-norm-based, percentage of zero activations, and gradient-based. The converged pruned architectures provide a Pareto front for solutions balancing accuracy and inference time. These methods [28] and [29] aim to optimize model performance while considering hardware constraints and efficiency.

To summarize, Table 5 highlights the performance of different pruning methods and their performance on various architectures and problems. The tables show that pruning the weights or filters can reduce the computational cost of the model without sacrificing accuracy. For a higher reduction of Flops (78%), a drop of around 3% accuracy is observed. Structured pruning (Table 5) is based on the selection or pruning of the filters present in deep learning architectures. Such a pruning method deals with the weight matrix sparsity in a better and more optimized way when considering hardware implementation.

3.1.2 Quantization Methods

State-of-the-art model conversion methods using quantization are shown in Table 6. Various quantization methods used in the research works of Table 6 are mixed precision quantization [30], vector quantization [31], quantization-loss aware algorithm [32], smart-DNN+ [33], multi-branch topology [34], ultra-low bit quantization [35], Quantized MobileNetV2 [36], tiny CNN for fall prediction [37], EtinyNet [38], Weight quantization based on clustering [39], layer-wise quantization [40] and extreme quantization [41].

Table 6: Results of post-training quantization methods (*DSC = dice similarity coefficient)

Ref.	Dataset/Model	Bits W/A	Baseline accuracy	Accuracy	FP32 model	Compressed model
[30]	UFPR ResNet-50	2	99.6%	99.92%	328.5 M	20.5 M
[31]	CIFAR-10 VGG-Like	3	93.5%	93%	20.44 M	1.94 M
[32]	ILSVRC-12 ResNet-18	3/3	69.2%	68.6%	—	—
[33]	CIFAR-100 VGG-16	2	66%	66%	138 M	23 M
[34]	CIFAR-100 ResNet-20	4	68.7%	69.4%	0.27 M	0.19 MB (Memory)

(Continued)

Table 6 (continued)

Ref.	Dataset/Model	Bits W/A	Baseline accuracy	Accuracy	FP32 model	Compressed model
[35]	BRATS Residual UNet	2	0.8418 DSC*	0.8363 DSC*	5.9 M	0.25 M
[35]	LiTS 3D UNet	2	0.7843 DSC*	0.7509	94.8 M	3.1 M
[36]	MobileNetV2 Tongue Dataset	8	99%	98%	26 M (Flash memory)	5.4 (Flash memory)
[37]	SisFall TinyCNN	8	98.4%	98.3%	Parameter size = 546	Parameter size = 546
[38]	ETinyNet	—	—	66.5%	—	477 K
[39]	1D MINIST LeNet5	Clustering	83.7%	83.5%	114.06 M	70.27 M
[40]	LiTs, BraTS2020 3D Unmet	4/4	78.36%, 84.8%	78.26%, 84.4%	—	—
[41]	YOLOv5l MPQ-YOLOl	1/1 + 4/4	87%	74%	178.4 M	12.6 M

Kolf et al.'s mixed precision quantization [30] reduces model memory footprint by first quantizing a 32-bit floating point model to an 8-bit integer model, then iteratively training and reducing weights to as low as two bits. As a result, the method achieves a 16-fold reduction in memory with minimal accuracy loss on models like ResNet18, ResNet50, and MobileFaceNet. Vector Quantization [31] balances quantization loss and model accuracy, achieving 5 to 16 times size reduction with minimal accuracy loss on different models. The quantization-loss-aware algorithm [32], uses Taylor's expansion to quantize deep neural network weights to low bit-widths for stable convergence. Smart-DNN+ [33] provides a layer-wise quantization protocol to compress models from full-precision to binary quantization. The multi-branch topology [34] avoids quantization error due to bit-width switching by using fixed 2-bit weights in each branch and combining branches to achieve the desired bit-width.

Ultra-low bit quantization [35] employs an adaptive quantizer with two tunable parameters to minimize quantization error. Both weights and activation functions are quantized, and full-precision weights are discarded after training. It achieves a comparable segmentation accuracy with one- and two-bit quantization. In [36], a 8-bit quantized MobileNetV2 model has been deployed on a Cam H7 Plus embedded device to classify oral cavity cancer. Tiny CNN for fall prediction [37] is trained on wearable sensor data, achieving over 98% accuracy on two datasets after being quantized to 8 bits. Xu et al.'s EtinyNet [38] is a seven-layer CNN with an extremely tiny backbone for visual processing, operating at 160 mW and processing 30 frames per second (FPS).

Automated quantization and retraining of the deep neural network models using multi-objective optimization (NSGA-II) is proposed in [39]. The authors have used two objectives, accuracy and model size, to find the Pareto front of the solutions. Quantization is done through vector quantization by representing the weight space into sub-regions, and a centroid of every sub-region is selected as a quantized weight representation. Results of various models and datasets showed a decrease in model size by at least 30% without significant loss of accuracy. Zhang et al. [40] have proposed a layer-wise quantization framework with an

alternating direction method of multipliers to achieve fast convergence during deep neural network training. They have applied the proposed method for volumetric medical image segmentation. A weight regularization term is added to the quantization objective to retain the knowledge of the full precision weights.

The MPQ-YOLO [41] is an ultra-low mixed quantization of the YOLO model for edge devices, combining 1-bit backbone and 4-bit head quantization with a dedicated training policy. The backbone, containing convolutional layers, is highly compressed with 1-bit quantization, while the head, sensitive to quantization error, uses 4-bit quantization. When compressing the model with extreme quantization, a compromise is desired between efficiency and the model's accuracy or performance [35,41].

To summarize, quantization methods map full-precision values to the nearest quantized value, but clustering-based quantization assigns similar weight values to a single cluster center. As evident from Table 6, It is challenging to decide which quantization method is better considering the accuracy and model compression ratio tradeoff. Regularization, which prevents overfitting in neural networks, must be modified for quantized networks. Post-training quantization can significantly drop model performance, so retraining or fine-tuning is necessary. Therefore, quantization-aware training simulates the effect of low precision during training, allowing the model to learn parameters that improve performance and reduce quantization errors simultaneously. Quantization to a lower number of bits may reduce the model size considerably. However, activation functions associated with each layer of the deep neural networks are difficult to quantize due to their nonlinear nature. A combination of the quantization of weights and the activation function achieves better results in terms of performance. Moreover, the quantization levels depend on the model architecture and the problem it is solving. A combination of weight and activation function quantization performs better in accuracy in many models. Therefore, the results of quantization-aware training are shown in Table 7.

Table 7: Results of quantization-aware training methods

Ref.	Dataset/Model	Bits W/A	Baseline accuracy	Accuracy	FP32 model	Compressed model
[42]	CIFAR-10 GXNOR-Nets	3	92.88%	92.5%	—	—
[43]	CIFAR-10 DenseNet	2	94.3%	94%	7 M	0.49 M
[44]	CIFAR-10 ResNet-20	2	91.8%	90.9%	—	—
[45]	CIFAR-10 ResNet50	2.68/4	76%	75.4%	97.28 M	9.45 M
[46]	CIFAR-10 ResNet-18	8	88%	79%	—	—
[47]	ImageNet ResNet-18	3/3	70.2%	69.2%	11.6 M	~45% decrease
[47]	ImageNet ResNet-18	5/5	70.2%	70.4%	11.6 M	~35% decrease

Note: W: Weights, A: Activation.

The work in [42] has introduced a multi-step activation function quantization method and a derivative approximation technique for backpropagation in discrete deep neural networks. This algorithm quantizes

both activation functions and weights to ternary values, creating a binary sparse network. A symmetric mixture of Gaussian modes [43] is a soft quantization-aware method that uses low-bit fixed quantization. It involves training with real-valued weights and generating posterior distributions for post-quantization. In [44], a training mechanism in a finite weight search space is presented. A Hessian-based mixed precision quantization-aware training method is used to optimize the search for the best bit configuration [45], employing a Pareto frontier method based on the average Hessian trace for different configurations. The quantized process does not consider energy consumption in quantization-aware training. Hence, Hamming Weight-based Energy Aware Quantization (HAMQ) is proposed in [46]. Considering the Compute-in-memory (CIM) architecture for edge devices with limited resources, HAMQ provides better quantization for energy efficiency. Jung et al. [47] proposed a trainable quantizer based on a quantization-interval-learning (QIL) framework. They obtained the optimal values of quantization intervals by minimizing the task loss of the network. Using the QIL framework, better quantization levels of weights and activation functions are achieved on ResNet variants without degrading the ImageNet dataset's accuracy, as shown in Table 7.

3.1.3 Model Compression Using Low-Rank Factorization

Table 8 provides the results of low-rank factorization methods. The tensor decomposition method leverages shared tensor structures and parameters across DNN layers [48]. The optimization achieves 93% accuracy with an eight-fold compression on ResNet-18 using CIFAR-10. It is important to note that matrix and tensor decomposition identify and remove redundant parameters, decomposing larger weight matrices into smaller, storage-friendly ones. On the other hand, convolutional layers are factorized into depth-wise or pointwise convolutions, reducing the computational time during inference.

Table 8: Results of low-rank factorization methods to compress the model

Ref.	Factorization	Model	Dataset	Compression
[48]	Tensors decomposition	ResNet-18	CIFAR-10	8 times
[49]	Sparse low-rank factorization	VGG-16, VGG-19	CIFAR-10	C* = 3.6
[50]	Tensor decomposition	VGG16, VGG19, ResNet-50	CIFAR-100	Compression ratio 98%, 97%, 95%
[51]	Joint matrix decomposition	ResNet-50 ResNet-34	CIFAR-10	6 times 22 times
[52]	Deep compression	VGG-16	ImageNet	15 times

Note: *C = Sparsified Compression ratio (ratio between truncated SVD and SLR).

The low-rank factorization is effective for compressing fully connected layers in deep neural networks and can act as a regularization method to improve performance. Higher-order singular value decomposition (SVD) can compress convolutional layers. Therefore, sparse low-rank factorization, using SVD and truncating SVD matrices, achieves good compression ratios on VGG16, VGG19, and Lenet5 [49]. However, it is challenging to deploy bigger models due to mobile devices' limited storage, computational power, and energy. Hence, the Tensor decomposition method can compress the network parameters [50]. A rank decomposition algorithm decomposes the tensor into a limited number of principal vectors. Hence, all the convolutional layers of the DNN can be decomposed and compressed. The original parameters can be reproduced and applied from these principal vectors in DNN architecture. When applied to VGG16,

VGG19, and ResNet50, a compression ratio of 98%, 97%, and 95% is achieved, respectively, with insignificant loss of accuracy on the CIFAR100 dataset. Due to the similarity among weight tensors of the different layers within a neural network model, simultaneous tensor decomposition can compress the model without considerably decreasing performance. The authors developed two tensor decompositions for fully or partially structure-sharing cases.

Chen et al. [51] proposed joint matrix decomposition to improve the compression ratio. They have suggested that compressing the convolutional layers separately produces less efficient CNN. Hence, three different joint matrix decomposition methods are tested on three CNN architectures. After compression, finetuning recovers some of the accuracy loss. On ResNet-34, a good compression ratio is achieved with an accuracy loss of less than 1%. However, the compression ratio on ResNet-50 is low (x6.2), with an accuracy loss of 3%. A deep compression technique is proposed in [52] based on global average pooling, iterative filter pruning, applying truncated SVD on the fully connected layers, and quantization. A pre-trained VGG16 model is used for the experiments on the ImageNet dataset. They have achieved a compression rate of 60 times (VGG16: 138 M, VGG16 compressed: 8.66 M parameters) with degradation in the classification accuracy of 0.85%.

3.1.4 Model Compression Using Knowledge Distillation

Table 9 shows results on knowledge distillation methods. The background on knowledge distillation techniques has been provided in Section 2.1.1. The work in [53] has used Efficient-Net-B0, with some modifications, as a teacher model to train a lightweight student model using a knowledge distillation technique. The student model is a simplified Efficient-Net-B0 model with a frozen convolution head and MBConvBlock-25. A response-based knowledge distillation method is used to calculate distillation loss. A distillation loss in the response-based knowledge distillation is calculated based on the difference between the logit of the teacher and student models. The student model is trained along with the teacher model using distillation loss and ground truth labels. If the gap between the teacher and student models is too big, then knowledge distillation may fail, and the student model may not follow the teacher model [54]. Hence, the teacher model can be simplified by using Tucker decomposition of the tensors to ensure that the student model follows the teacher model during knowledge distillation. Using VGG16 as a teacher model and replacing the convolutional layers with the tucket decomposition layers, a simpler model can distill the knowledge to the student model (LeNet in this paper). The decomposition rank is 16 to maximize the accuracy improvement of the student model.

Table 9: Results of knowledge distillation methods used to compress the models

Ref.	Dataset	Teacher model				Student model		
		Model	Accuracy	Size	Flops	Size	Accuracy	Flops
[53]	BOSSbase	Efficient-Net-B0	97.8%	5.3 M	0.56 B	3.6 M	98%	0.34 B
[54]	UTKinect-Action3D	VGG16	99%	138 M	15 B	4.3 M	98.7%	4.4 M
[55]	DFU dataset	InceptionV3	98%	28.5 M	5.71 M	0.49 M	96%	0.42 M
[56]	FaceForensics++	XceptionNet	96.3%	20.8 M	6 B	2.74 M	96%	1.2 B
[57]	CIFAR-100	ResNet56	72.3%	0.85 M	—	0.27 M	72%	—

(Continued)

Table 9 (continued)

Ref.	Dataset	Teacher model				Student model		
		Model	Accuracy	Size	Flops	Size	Accuracy	Flops
[58]	URBAN100	RCAN	29.09 PSNR	15.44 M	35.3 G	4.28 M	32.85 PSNR	9.79 G
[59]	Histopathologic cancer dataset	ResNet50	95.75%	26 M	—	11 M	95.8% 96.6% (Ensemble)	—
[60]	Vaihingen	ResNet50	88%	24 M	4.1 B	14 M	84%	116 B

Amjad et al. [55] proposed a lightweight model called DFU-LWNet and trained it by InceptionV3 (teacher model) through knowledge distillation. The proposed model contains three convolutional layers with the max-pooling layers, and convolutional modules are borrowed from the Efficient-Net model. Finally, a customized classifier is added. Response-based knowledge distillation is used to transfer knowledge. Comparable classification accuracy is achieved with a DFU-LWNet with only 0.48 M parameters compared to the InceptionV3 model with 28.5 M parameters. Xu et al. [56] used a feature-based knowledge distillation technique to train the student model using cross-entropy loss, knowledge distillation loss, and gradient-guided feature distillation loss. XceptionNet is used as a teacher model and trained on the deepfake video dataset FaceForensics++ dataset. In the gradient-guided feature loss (feature-based distillation), the intermediate layers of the teacher and student models are mapped to define the distillation target. Gradient-guided weights define the importance of different channels in the feature maps. A decayed teaching strategy I used to modify the gradient-guided weights. A comparable accuracy is achieved by the student model with only 2.74 M parameters compared to the teacher model (20.8 M parameters). Usually, the SoftMax scaling factor (fixed temperature value) does not change for the data samples and considers all the samples of equal difficulty level. Hence, Ham et al. [57] explored the effect of data difficulty level on knowledge distillation. The model distills the knowledge based on three difficulty levels. The difficulty level is estimated through the Euclidean distance between the teacher's and pruned teacher's predictions. They have tested their method on various combinations of teacher-student models for CIFAR-100 and FGVR datasets.

A hybrid knowledge distillation from intermediate layers of the teacher and student model is used to create a single image super-resolution [58]. For this purpose, auxiliary up-samplers are added to the teacher and student models to create intermediate super-resolution images. Once the up-samplers of the teacher model are trained, the frequency similarity matrix (using discrete wavelet transform) and adaptive channel fusion are used to distill the knowledge and update the student model's up-samplers parameters. The models are trained using the DIV2K dataset and tested on various datasets, including BSD100 and Urban100. The comparable peak signal-to-noise ratio (PSNR) achieves a compression ratio of four times in 2X and 4X super-resolution. Niyaz et al. [59] used an interesting concept of knowledge distillation from one teacher model to multiple student models with collaborative learning among the student models.

Furthermore, they have compared the offline KD (training the teacher model first) and online KD (both teacher and students trained simultaneously) effect on the classification performance. An ensemble makes the final prediction of the prediction of the student models. Moreover, different learning styles, final prediction with one student, and intermediate layer features with other students are also investigated. Results of ResNet50 (teacher) and ResNet18 (students) are given in the table below for the histopathologic cancer detection dataset.

A multidimensional KD approach is adopted to improve the capability of transferring knowledge from the teacher to the student model [60]. ResNet-50 is the backbone of the teacher model, and MobileNet V2 is the backbone of the student model. Outputs of five different channels from the teacher and student models are fused, multiscale information is extracted, and feature-based distillation loss is calculated. Apart from this distillation loss, five other distillation losses, namely, inter-layer relation-based distillation loss, intra-layer feature-based distillation loss, wavelet transform response-based distillation loss, and logits distillation loss. They tested their methodology on two datasets, Potsdam and Vaihingen, and compared them with other published methods to prove the efficacy of the proposed method.

3.2 Inference Strategies

Once the model is deployed on the edge device, the inference mechanism has two options. Either the inference is performed entirely on the device, with the results utilized and communicated by the edge device (on-device inference), or the inference is carried out in collaboration with other edge devices or edge servers (distributed inference). Sections 3.2.1 and 3.2.2 describe on-device inference and distributed inference, respectively. A summary of different inference mechanisms is presented in Table 10.

Table 10: Comparison of inference strategies used in the edge ML

Ref.	Inference strategy	Dataset	Model	Performance
[61]	On-device inference	KTH, UCI	CNN (7.7 K parameters)	Inference < 3 s Energy = 250 mJ
[62]	Model split (Multiple points)	—	GoogleNet	Latency = 2.058 s Energy = 7616 mJ
[63]	Model split (Layers based)	Customized	AlexNet	Save inference time by 12% to 66% Best latency = 1.45 s
[64]	Task-aware Splitting	Metal casting dataset	CNN	Better latency under different bandwidth
[65]	Optimal Splitting	—	ResNet, AlexNet, VGG	Latency and energy consumption under different settings
[66]	Model Partitioning	ImageNet	VGG-16, ResNet-34, MobileNetV1	84% reduction in Latency, 14x speedup
[67]	Model split (Layers based)	ModelNet40	VGG-16	Inference latency compared with different Inference schemes
[68]	Horizontal Partitioning	—	AlexNet, VGG16-BN, ConvNext	More robust at the expense of maximum memory and energy
[69]	Energy-aware Model split	—	VGG-16, MobileNetV2	Load reduction in ESS = 15%–20%, LSS = 60%, Energy saving 18%, 52%
[70]	Privacy-aware Model split	CIFAR-10, MNIST	Customized CNN	Strong privacy, better runtime on datasets

3.2.1 On-Device Inference

The on-device inference is feasible when the small model size and the edge device have sufficient computational and energy resources. TinyML models, typically with parameter sizes under 1 M, often make on-device inference a viable option. For instance, Nooruddin et al. [61] introduced a two-stream multi-resolution fusion method for human activity recognition from video data. They utilized a customized CNN model with quantization-based conversion. This compressed model was deployed on three tiny edge devices and tested on the KTH and UCF11 datasets. The proposed model achieved 98% accuracy, with 7.7 K parameters requiring 575 KB of memory. The inference time on the Arduino Nano 33 BLE Sense device was under 3 s, with a power consumption of 250 mJ.

3.2.2 Distributed Inference

A comparison of state-of-the-art inference methods is provided in Table 10.

The DeepWear framework in [62] offloads deep learning tasks from wearable sensors to handheld devices via Bluetooth, eliminating the need for an internet connection. This approach enhances model performance and reduces the energy footprint of edge devices. The authors have explored various model-splitting strategies, examining their impact on latency and energy consumption. The model has achieved an inference speedup of two to three times and energy savings of 18% to 32%.

In [63], a DNNOff strategy is proposed, which comprises three components. The extraction component extracts the structure and parameters of the DNN mode, an offloading mechanism, and an estimation model that defines the offloading strategy on edge devices. In the adaptive offloading scheme, a decision must be made for each layer on whether the computation will be on the edge of the server. A random forest regression is used to predict the execution time of each layer. The offloading schemes are tested on the AlexNet model, splitting the model into edge server, cloud server, and edge devices. Gautam et al. [64] proposed a task-aware DNN splitting scheme for EdgeML smart manufacturing. The model contains sensing and edge layers. The edge layer consists of an edge computing node and an edge server. Each layer of DNN is a potential splitting point, and the splitting policy is based on the task execution time, bandwidth between the device and the edge server, and profiling parameters. An optimal splitting policy is adopted based on minimizing average execution time. Energy optimization of the edge devices is not considered [65].

Self-aware model partitioning [66] considers the model partitioning by the edge device according to its computational resources status or time constraint on the inference. It engages a subset of available devices with sufficient resources to collaborate in the inference task by offloading partial inference tasks. Collaborative inference with two to four devices shows an improvement in inference performance. Another work on collaborative inference on edge devices focuses on a selective scheme that reduces data redundancy and bandwidth resource availability [67]. Multi-view images have a lot of spatial correlation taken by different edge devices. So, multi-view classification from edge devices can collaborate with centralized servers, splitting the inference tasks. In the selective ensemble inference, each edge device decides whether it provides its inference to the server for ensemble decision or not. The above methods focused on vertical partitioning of the model in which an entire layer is assigned to a node. Hence, different layers are assigned to different nodes. This type of partitioning produces high throughput but has a higher failure risk. In case of failure of one node, the whole inference procedure is affected as the entire layer is comprised.

Guo et al. [68] proposed the RobustDiCE method for robust distribution of the tasks among the inference nodes. This method divides and distributes every model layer among the inference nodes (horizontal partitioning). In case of a node failure, it is easy to recover the inference flow of the model. Hence, in this scheme, neurons of every layer are evenly distributed and assigned to the edge devices taking part in the

inference. The method is evaluated on AlexNet, VGG16, and ConvNext models against failures of the devices. Experimental results showed that accuracy does not drop significantly for one device failure. However, more than one device failure affects the accuracy by more than 20%. In [69], authors proposed two strategies of model splitting for battery-operated IoT devices and regular-powered IoT devices. In the early split strategy, the maximum part of the model is offloaded to the edge server, saving the energy consumption of the battery-operated edge devices.

In contrast, in the late-split strategy, layers of the model requiring heavy computation are offloaded to the edge server. Wang et al. [70] proposed a privacy-preserving protocol for the edge device and edge server collaborative inference in the industrial Internet of Things. Two edge servers participate in the inference and calculate the model output without knowing the data or the model. The results showed that the proposed method can achieve a good tradeoff between latency and throughput.

To summarize the results, the most effective inference strategy is the model splitting among the edge devices. Much research has been done on optimizing. Hence, the inference strategy may vary according to the available resources, inference demand, and energy consumption.

3.3 Learning Strategies

We have classified our results into (a) online learning strategies, (b) continual learning or life-long learning, and (c) federated Learning. Sections 3.3.1–3.3.3 provide results on online learning strategies, continual learning or life-long learning, and federated Learning. Tables 11 and 12 summarize continual learning and federated Learning, respectively.

Table 11: Comparison of continual learning methods in the edge ML

Ref.	Model dataset	TinyModel	Baseline accuracy	TinyModel accuracy	Memory requirement
[71]	Customized MLP Three diseases	Full precision	—	92.8%	Buffer size: 2.8 MB Model size: 0.35 M
[72]	MobileNetV2 OpenLORIS	—	—	97.6%	5.90 M
[73]	MobileNetV1 Core50	Quantized UNIT-8/UINT-7	77%	68% 75%	<4 MB <40 MB
[74]	MCUNet multiple	UINT-8	73.3%	73.7%	0.48 M
[75]	ResNet-20 CIFAR-10	3-bit gradient quantization	90%	89.8%	—

3.3.1 On-Device Learning

In offline strategies, large datasets are collected and used to train deep learning or machine learning models on high-performance computing resources. Once trained, models are reduced in size using techniques like quantization and pruning to fit edge devices, often called TinyML. Model inference can be performed on the edge device alone through model splitting, task offloading, or distributed inference across multiple edge devices. In decentralized learning, edge devices collaborate to train the model. On-device learning is beneficial when new data is continuously available, requiring continuous model updates. It necessitates reliable and continuous wireless communication between edge devices, typically within fixed topologies.

Table 12: Comparison of federated learning methods for Edge ML

Ref.	FL method	Dataset	No. of nodes	Sample per node	Performance	Time complexity
[76]	DeFL	Credit card	8000	~31	89.5%	—
[77]	DeFL	CIFAR-10	~80	—	~80%	Model: 10.7 s W/update: 0.015 s
[78]	HetFL	CIFAR-100	10	10	76%	FLOPs 11.5 M
[79]	HierFL	MNIST	100/3**	—	97.9%	—
[80]	HetFL	CIFAR-10	50	1000	64.5%	60 s (GPU: RTX3080)

Note: DeFL: Decentralized FL, HetFL: Heterogenous FL, HierFL: Hierarchical FL. ** Nodes/servers.

3.3.2 Continual Learning or Life-Long Learning

Continual learning, or life-long learning, involves continuously recording data and learning in a non-stationary environment without losing previously acquired knowledge. It can be done on edge devices, enhancing security and privacy, though memory constraints make it challenging. Various learning methods have been compared in Table 11.

A continual learning framework for disease detection using wearable medical sensors has been proposed [71]. Authors have used a data preservation method to retain the most informative previously learned data. Subsequently, a multilayer perceptron (MLP) was trained and achieved an average accuracy of 92.8% on the edge device. Similarly, the latent replay concept was proposed to address catastrophic forgetting in continual learning by storing activations of initial model layers instead of raw data [72]. The work in [73] has adapted this for 8-bit quantized models, calling it quantized latent replay-based continual learning. They found a tradeoff between latency layers and accuracy. The work in [74] has explored on-device training on Cortex-M MCUs using MCUNet, employing dynamic sparse gradient updates for fully quantized training. Finally, the work in [75] has proposed a framework for continual learning in noisy, dynamic environments, using selective experience replay and low bit-width quantization, achieving minimal accuracy loss (0.02%) with ResNet-20 on CIFAR-10 under heavy image degradation.

3.3.3 Federated Learning

Architectures in this category vary based on aggregation methods and task assignments to edge devices. In centralized FL, a server connects to all edge devices or nodes, selects nodes, and aggregates the model, suitable for a limited number of nodes. In hierarchical FL, instead of a centralized server, various nodes act as aggregation nodes, with edge devices connected to these aggregation nodes. In decentralized FL, all nodes participate in training and aggregating the global model. Similarly, heterogeneous FL addresses variations in data distribution, communication environments, device hardware, and model architectures.

Table 12 compares state-of-the-art federated learning methods in terms of various performance attributes. A deep autoencoder for FL uses non-iterative training to reduce training time [76]. In a multi-node environment, nodes send local model information via the Queuing Telemetry Transport (MQTT) broker protocol, which updates and aggregates this information. This method has proven effective in accuracy, latency, and energy consumption on several datasets. Similarly, Zhang et al. [77] have designed an efficient FL mechanism for edge devices successfully applied to MNIST and CIFAR-10 datasets. Yang et al. [78] have proposed a resource-efficient heterogeneous federated continual learning algorithm for edge devices,

reducing resource consumption by dividing the model into adapter and retainer sub-models. Qiang et al. [79] have introduced a multi-layer federated edge learning framework using edge servers between devices and the cloud to reduce latency and energy consumption. Similarly, Cao et al. [80] have proposed feature-space and output-space alignments for aggregating local models to minimize performance loss due to data heterogeneity. They found that for the CIFAR100 dataset, achieving a target accuracy of 35 requires 30 communication rounds with significant data distribution deviation and 20 rounds with more uniform data distribution.

It is important to note that aggregating local model updates is crucial in federated learning. The issues in aggregating local model updates include imbalanced, varied quality, non-uniformly distributed data subsets among FL clients, and the heterogeneity of edge devices affecting global model training efficiency. Moreover, the convergence is also challenging due to device heterogeneity and asynchronous updates. However, standard aggregation averages model updates, outliers, or malicious updates can corrupt the global model. Therefore, clipping model updates within a defined range can counter outliers [81]. Furthermore, momentum-based aggregation, where edge devices send the momentum term and local model update, speeds up the convergence process [82].

Another critical factor in FL methods is the contribution of local model updates to the global model for defining edge device performance. Therefore, weights are assigned to each contributing edge device based on its reliability and representation in global model training. Similarly, incorporating predictive uncertainty during aggregation can enhance the global model's generalization capability [83]. Furthermore, quantizing local updates improves communication efficiency and energy management for edge devices. While homogeneous quantization simplifies aggregation, real-world scenarios often involve heterogeneous quantization, where devices have varying precision levels. Federated learning with heterogeneous quantization assigns different weights to edge devices to account for quantization errors [84]. For more details on aggregation and learning strategies in federated learning, refer to a comprehensive survey [85].

4 Results on Model Deployment on Resource-Constrained Devices Using TinyML

Section 3 critically reviews model conversion, inference mechanisms, and learning strategies. This section reviews state-of-the-art model deployment techniques on resource-constrained devices using TinyML. Moreover, the results have been classified according to different sectors, as TinyML has numerous applications in various sectors. Therefore, we have identified six major sectors where TinyML-based model deployment has shown promising results. This classification aims to highlight the target problems (practical examples) in each sector, along with the corresponding ML/DL models.

i. **Smart agriculture, or smart farming** sector, involves monitoring/collecting real-time data about various parameters (corps, livestock, soil quality, etc.) in farming and optimizing it to increase yields.
ii. **Medical or healthcare with environmental safety** sector targets monitoring vital signs, using wearable devices to diagnose early health anomalies. Moreover, it also includes monitoring environmental conditions (such as air quality, water quality, and weather patterns) to enable pre-emptive actions.
iii. **Vehicles or the automotive** sector collect and process real-time data for multiple driver assistance systems, enhancing vehicle safety and efficiency.
iv. **The industrial and robotics sectors** focus on monitoring equipment and facilities for signs of wear and tear to enable predictive maintenance, reducing downtime and maintenance costs.
v. **The energy sector** mainly contains applications for managing renewable energy from different sources.
vi. **Secure smart cities and the consumer electronics** sectors include data collection from smart cameras and sensors to detect unusual activities or security breaches in real time, enhancing public safety. In addition to the security of smart cities, this sector also covers the security of smart homes using

various consumer electronic devices. Other applications of TinyML in smart cities are discussed in the environmental sector (such as monitoring air quality, noise levels, and other environmental factors) and the vehicle sector (such as optimizing traffic flow).

The achieved results have been synthesized in different ways to perform a critical analysis and comparative evaluation of the methodologies. The synthesis results are shown in Tables 13–16. Table 13 shows applications (practical problems) and associated ML/DL models used for each practical problem in different sectors. Similarly, Table 14 summarizes fifteen hardware development boards and compares their features for TinyML deployment. The development boards in Table 14 have been extracted from the selected research works. It implies that Table 14 compares selected research works regarding hardware development. In addition to the comparison in terms of hardware development, Table 15 classifies and compares the selected research studies in terms of three major software frameworks. Finally, Table 16 details the sensors used in all the chosen research work.

In the following, we briefly discussed all the selected research studies, organized into six categories. Consequently, Sections 4.1–4.6 provide a critical summary of selected research studies in smart agriculture, healthcare, vehicles, industry, energy, and security, respectively.

4.1 Smart Agriculture or Farming

It is a high-priority sector since it creates economic opportunities and generates most of the world's food. To satisfy the food demand, the major portion of agricultural tasks are required to be automated. While IoT communication technologies in smart agriculture have been reviewed previously [86], the use of intelligent IoT systems in the agriculture sector is a relatively new idea. This subsection provides practical examples (applications) of smart farming, where decision-making operations are performed using edge resources and TinyML techniques and tools.

Table 13: Applications and machine learning models in six identified sectors

Sector name	Applications (practical problems)	Model type	Year	Ref.
Smart agriculture or farming	Watering process automation	Not specified	2022	[87]
	Maize leaf disease detection and classification	CNN	2024	[88]
	Soil quality monitoring and management	CNN, RNN, Q-learning	2024	[89]
	Maize leaf disease detection and classification	CNN	2024	[90]
	Disease detection and classification in plants	CNN	2023	[91]
Smart healthcare	Face mask detection on faces	CNN	2023	[92]
	Recognition of human activities using wearable sensors	CNN	2023	[93]
	Energy-efficient healthcare decision support system	RF, SVM, DT	2024	[94]
	Human activity recognition	CNN, LSTM	2024	[95]
	Real-time blood pressure (bp) estimation	CNN	2024	[96]

(Continued)

Table 13 (continued)

Sector name	Applications (practical problems)	Model type	Year	Ref.
Smart automotive or smart vehicles	To process vehicular data on edge devices for fuel consumption prediction	AutoCloud + TEDA	2024	[97]
	Outlier detection and correction	TEDA-RLS	2024	[98]
	Intrusion detection system	CNN	2024	[99]
	To enhance the cybersecurity in electric vehicle	MLP, RF	2024	[100]
	Real-time driver behavior analysis	AutoCloud + TEDA	2024	[101]
Industrial automation	Identifying anomalies using autoencoders	Neural Networks	2021	[102]
	Operational efficiency and safety	CNN	2022	[103]
	Reactive and dynamic online control for robots	ADMM	2024	[104]
	Vibration-based fault diagnosis of machines	CNN	2023	[105]
	Identification of various bolt defects in steel structures	FOMO	2024	[106]
Energy sector	Real-time fault diagnosis and classification of defects in PV modules	CNN	2024	[107]
	Fault detection in PV modules	DCNN	2022	[108]
	Forecasting solar energy yield	LSTM, BiGRU, BiLSTM, BiRNN	2023	[109]
	Energy consumption prediction on mobile devices	LSTM	2024	[110]
	Upgradation of energy distribution panel	LSTM	2024	[111]
Consumer electronics and security	Detection of jamming attacks in wireless networks	CNN	2022	[112]
	Monitoring the condition of handheld power tools	CNN	2022	[113]
	To detect and combat various Wi-Fi attacks	DNN, LSTM	2024	[114]
	Real-time health monitoring and intrusion detection	Naive Bayes and SVM	2024	[115]
	Energy harvesting resource allocation for Unmanned Aerial Vehicles (UAV)-assisted TinyML consumer electronics	Naive Bayes, SVM, RF, KNN	2024	[116]

Table 14: Summary of development boards in selected research works

S. no.	Development boards	MCU	Core (CPU)	Speed (MHz)	Flash (MB)	SRAM (KB)	Ref.
1.	Wio terminal	ATSAMD51	ARM Cortex-M4F	120	1	256	[87,89]
2.	Arduino Nano 33 BLE Sense	nRF52840	ARM Cortex-M4	64	1	256	[88,90–92,95–98, 102,103, 107,114]
3.	ESP32 development board	ESP32	Xtensa LX6, dual core, 32-bit	240	4	520	[96–98,100,101, 105,109]
4.	STM32F746 discovery kit	STM32F746NGH6	ARM Cortex-M7	216	1	320	[91]
5.	Espress ESP-EYE	ESP32	Tensilica LX6	216	1	320	[91]
6.	Arduino Uno R3	ATmega328P	8-bit AVR	16	32 KB	2	[94]
7.	Raspberry Pi 3 Model B	Single board computer	ARM Cortex A-53	1.2 GHz	16 GB	1 GB	[96,97,105,112]
8.	Raspberry Pi Pico	RP2040 chip	ARM Cortex M0+	133	2	264 KB	[96,97]
9.	Siemens board	Not mentioned	ARM Cortex-M4	90	1	320 KB	[103]
10.	Teensy 4.1	Not mentioned	ARM Cortex-M7	600	7.75	512 kB	[104]
11.	Crazyflie 2.1 Quadrotor	Not mentioned	ARM Cortex-M4	168	1	192 KB	[104]
12.	Arduino Nicla vision	STM32H747AII6	Dual Arm Cortex M7	480	2	1	[106]
13.	Arduino Nano 33 IoT	–	ARM Cortex-M0+ 32-bit SAMD21	48	256 KB	32 KB	[107]
14.	Raspberry Pi 4	Broadcom BCM2711	ARM Cortex-A72	1.5 GHz	None	Up to 8 GB	[108]
15.	nRF52832	–	ARM Cortex-M4	64 MHz	512 KB	64 KB	[113]

Table 15: Summary of software platforms in selected research works

S. no.	Software platform	Supported development boards	Language for deployment	Open source	Manuf.	Ref.
1.	Edge impulse	Wio terminal	C++	No	Edgeimpulse	[87,88], [92]
2.		Arduino Nicla vision				[94] [106]
3.		Arduino Nano 33 BLE sense				[90], [107]
4.	Tensor Flow	Arduino Nano 33 BLE sense	Python, C++ Java, javascript, swift, Go, TensorFlowLite	Yes	Google	[88,91,92,107,114]
		Wio terminal				[89]
		STM32F746 discovery kit				[91]
		Espress ESP-EYE				[91]
		Not mentioned				[93]
		Raspberry Pi 4				[108]

(Continued)

Table 15 (continued)

S. no.	Software platform	Supported development boards	Language for deployment	Open source	Manuf.	Ref.
5.	TensorFlow lite	ESP32 development board	Python, C++ Java, Swift, Objective-C, JavaScript	Yes	Google	[96,97,100,101, 105,109]
		Arduino Uno R3				[94]
		Arduino Nano 33 BLE sense				[95–98,102, 103,107],
		Raspberry Pi 3 Model B				[96,97,105,109]
		Raspberry Pi Pico				[96,97]
		Raspberry Pi 4				[108]
		nRF52832				[113]
		Not mentioned				[103]

Table 16: Summary of sensors used in selected research works

Board	Sensors	Ref.
Wio terminal	Monitoring of temperature, humidity, moisture, light levels, and water levels	[87]
	Monitoring of temperature, humidity, moisture, water level, additional sensors such as motion sensors, microphone, and infrared emitter	[89]
Arduino Nano 33 BLE sense	Motion sensors, environmental sensors, pressure sensors, microphones, and light sensors	[88]
	Camera	[90]
	Motion sensors, camera module, LEDs, and buzzer	[92]
	Motion sensors (accelerometer, gyroscope, magnetometer)	[95]
	Photoplethysmogram (ppg) sensors, ECG sensors	[96]
	Fuel sensors, speed sensors, engine sensors, and environmental sensors	[97]
	On-board diagnostics ii (obd-ii) port	[98]
	3-axis accelerometer sensor	[102]
	3-axis accelerometer for vibration measurement	
	Infrared (ir) camera, temperature sensors, imu	[107]
ESP32	GPS module, water level detection, and temperature sensors	[91]
	Photoplethysmogram (ppg) sensors, ECG sensors	[96]
	Fuel sensors, speed sensors, engine sensors, and environmental sensors	[97]
	On-board diagnostics reader, embedded sensors in vehicles (speed sensors, fuel sensors, engine sensors, environmental sensors)	[98]
	Network traffic sensors	[100]
	Monitoring speed, engine load, throttle position, GPS	[101]
	solar irradiance sensors, weather sensors	[109]

(Continued)

Table 16 (continued)

Board	Sensors	Ref.
Raspberry Pi 3 Model B	Photoplethysmogram (ppg) sensors, ECG sensors	[96]
	Fuel sensors, speed sensors, engine sensors, and environmental sensors	[97]
	Software defined radio (SDR)	[112]
Raspberry Pi Pico	Photoplethysmogram (ppg) sensors, ECG sensors	[96]
	Fuel sensors, speed sensors, engine sensors, and environmental sensors	[97]
Arduino Nicla	High precision and quartz accelerometers, low precision accelerometers	[105]
	acceleration sensors for collecting time-series vibration signals	[106]
Arduino Uno R3	Monitoring of pulse rate and oxygen levels, sensors for room humidity and temperature, gas sensors for the detection of toxic gases, sound sensors	[94]
Siemens	3-axis accelerometer for vibration measurement.	[103]
Teensy 4.1	3-axis accelerometer, optitrack motion-capture system	[104]
Crazyflie	3-axis accelerometer, optitrack motion-capture system	[104]
nRF52832	Accelerometer for monitoring tool usage, temperature, and humidity sensors	[113]
–	IMU sensors, heart rate, blood pressure, pulse rate, and temperature	[93]

One of the preliminary TinyML-based work in the agriculture sector is presented in [87]. The objective is to automate the watering of plants by monitoring environmental conditions and soil moisture, as shown in Table 13. The system is developed around a Wio terminal board, as shown in Table 14. The Edge Impulse (EI) is used for dataset generation, training, inference, and testing purposes, as shown in Table 15. The employed sensors are for sensing the temperature, humidity, soil moisture, level of light, and level of water, as shown in Table 16. While the work in [87] only elaborates on the initial idea, the complete IIoT systems for smart agriculture have been presented in [88,89,111,112].

The IIoT system in [88] employs a customized CNN model to identify maize leaf disease. An important feature of this work is to evaluate the performance of TensorFlow and Edge Impulse frameworks. It has been observed that Edge Impulse is more user-friendly for data collection, labeling, and model deployment. Furthermore, lower memory footprint and power consumption make it suitable for deployment on low-powered edge devices. On the other hand, TensorFlow offers greater customization and control over the model architecture and training process. It has higher accuracy but requires more memory and computational resources. Another IIoT system in smart farming is presented in [89]. The system presents a power-aware and delay-aware TinyML model for monitoring and managing soil quality in agriculture. The model integrates dynamic voltage and frequency scaling (DVFS) as well as sleep/wake strategies based on genetic algorithms, energy harvesting, and task partitioning to optimize energy consumption and reduce delay. It employs CNN, RNNs, and Q-learning models. Like the work in [87], the Wio Terminal is used as the development board, while Tensor Flow is employed to deploy the model on the target development board.

A CNN-based approach for maize leaf disease detection and classification is presented in [90]. The customized CNN model, deployed on Arduino Nano 33 BLE Sense using the Edge Impulse platform,

extracts important visual patterns from maize leaves, enhancing disease identification capabilities. The work presented in [91] aims to identify plant diseases using a customized CNN model. It has employed the TensorFlow framework to deploy the model on different development boards. In addition to the aforementioned applications in smart agriculture [87–91], the other applications include but are not limited to real-time embedded prediction weather system and an end-to-end strategy for enhancing the security of the food supply chain.

To summarize the key findings of TinyML deployment in smart agriculture, it can be stated that TinyML-based agriculture systems offer significant results in analyzing specific conditions of individual farms, providing customized recommendations and actions based on real-time data. It has allowed practitioners to make decisions and perform tasks without constant human intervention, which is especially useful for large-scale farming. Moreover, by processing data locally, IIoT-based systems in smart farming are reducing the need for expensive cloud services. It not only lowers operational costs but also minimizes reliance on high-speed internet. Since data is processed on the device, the amount of data that needs to be transmitted is significantly reduced, conserving bandwidth and making the system more efficient.

4.2 Healthcare and Environmental Sector

An intelligent IoT-based healthcare decision support system is becoming paramount in today's healthcare domain. With the increasing prevalence of chronic diseases and the aging population, continuously and remotely monitoring patients' health becomes crucial. IoT devices collect real-time data on vital signs, medication adherence, and lifestyle habits, enabling healthcare providers to make informed decisions swiftly. Consequently, it enhances patient outcomes by allowing early detection of potential health issues and timely interventions. Moreover, it reduces the burden on healthcare facilities by minimizing hospital visits and enabling efficient resource management.

One of the earlier target problems in healthcare is to process images and identify all those cases where a person is not wearing a face mask, as shown in [92]. Using the Edge Impulse platform, a low-cost solution using compressed CNN and transfer learning based on the MobileNetV1 architecture is deployed on the Arduino Nano 33 BLE Sense board. Another practical example of TinyML deployment in healthcare is classifying human activities from wearable sensors using an on-device deep learning inference mechanism, as shown in [93]. A lightweight CNN is trained offline on a stand-alone computer using TensorFlow. The specific microcontroller is not explicitly mentioned, but it is described as having minimal RAM (320 KB of SRAM) and operating at an 80 MHz clock frequency.

The classification of human activities in [93] is further extended to the analysis of human health parameters, as shown in [94]. The framework is centered on Raspberry Pi (fog layer) using TensorFlowLite; it reduces latency and ensures real-time data processing and analysis, which is crucial for time-sensitive healthcare applications. Another human activity recognition system is presented in [95]. The motion data (including acceleration, angular velocity, and magnetic field) is captured at a sampling frequency of 110 Hz. The compressed deep learning models (CNN and LSTM) with pruning and quantization techniques are deployed using TensorFlow Lite. Finally, a real-time blood pressure estimation problem is targetted in [96] using CNN types (AlexNet, LeNet, SqueezeNet, ResNet, and MobileNet) on various edge devices, including Raspberry Pi, ESP32, Raspberry Pi Pico, and Arduino Nano. The objective is to discuss the trade-offs between model size, accuracy, and inference time.

In addition to the aforementioned practical examples extracted from selected research works, there are other IoT healthcare applications, such as the prediction of chronic obstructive pulmonary disease, real-time activity tracking for elderly people and their nurses, a hand gesture recognition approach, etc. These

advantages make TinyML a valuable technology in the healthcare sector, improving patient outcomes and operational efficiency.

4.3 Vehicles or Automotive Sector

Smart vehicles, also known as intelligent or connected vehicles, are transforming the automotive industry by integrating embedded technology, communication technology, and artificial intelligence technology to enhance safety, efficiency, and user experience [117]. The use of intelligent IoT systems has enabled the monitoring of vehicle components in real-time. For example, predicting maintenance proactively can reduce unexpected breakdowns and improve vehicle reliability. It, in turn, reduces the cost and warranty claims. Similarly, IIoT systems in the smart vehicles sector assist in collision avoidance and adaptive cruise control. It enables critical functions to operate without a constant internet connection, ensuring that safety features remain active in all conditions. Another problem in this sector is analyzing driver behavior and detecting signs of drowsiness or distraction, alerting the driver, or taking corrective actions to prevent accidents. It can also monitor the cabin environment (temperature, air quality) and adjust settings automatically to enhance passenger comfort. On-device processing ensures minimal delay (low latency) in executing critical functions, enhancing safety and performance. The low-cost hardware makes advanced features accessible in a broader range of vehicles and can be easily scaled across different vehicle models and types.

Scalable real-time processing of vehicular data streams on edge devices is presented in [97] by combining AutoCloud and TEDA (Typicality and Eccentricity Data Analytics). Real-time processing provides immediate insights and predictions, enhancing decision-making and operational efficiency. Four development boards (Raspberry Pi 3 Model B, ESP32 Wrover IE, Raspberry Pi Pico, and Arduino Nano 33 BLE) have been used to demonstrate the proposed algorithm's feasibility. Another application of smart vehicles is outlier detection and correction in vehicular data streams, as shown in [98]. The system employs the TEDA-RLS algorithm to increase data quality and reliability. The algorithm detects outliers using TEDA and corrects them using RLS filters. C++ is used to implement the algorithm on the ESP-32 microcontroller. In addition to real-time processing of vehicular data streams, intrusion detection systems are becoming increasingly important in the smart vehicles sector. An example of such a system can be found in [99], where a CNN-based approach is deployed on an nRF52840 microcontroller using TensorFlow Lite. A TinyML-based approach, using Multi-Layer Perceptron (MLP) and RF algorithms, is presented in [100] to enhance cybersecurity within the context of Electric Vehicle Charging Infrastructures.

The issues of real-time traffic management and driver behavior analysis in intelligent transportation Systems are addressed in [101] by presenting a multi-layered, stream-oriented data processing methodology for edge computing environments to detect and classify driver behavior patterns. The approach integrates the TEDA framework and an incremental clustering algorithm. The process starts by collecting data from vehicular physical sensors (e.g., speed, engine load, throttle position, RPM) and constructs a radar chart to represent multidimensional sensor readings as a polygon, with the area within the polygon reflecting the vehicle's resource utilization. Subsequently, it identifies and mitigates outliers in the data stream. Moreover, it implements a dynamic window to adapt to changes in data distribution and informs the incremental clustering algorithm about potential shifts. Finally, the AutoCloud algorithm is used for incremental clustering and does not require retaining datasets in memory. The TensorFlow Lite is used to deploy the machine learning models on the ESP32 microcontroller board. The customized AutoCloud algorithm was integrated into embedded systems using C++ and deployed on the hardware using the TensorFlow Lite library through Arduino IDE.

To summarize, the TinyML has been primarily used for predictive maintenance (to monitor vehicle components in real-time), driver assistance systems (to provide real-time alerts for lane departure,

collision warnings, and driver drowsiness detection), and in-vehicle monitoring (monitoring the health and performance of various vehicle systems, such as the engine, brakes, and battery, to ensure optimal performance and safety). Moreover, the TEDA algorithm enhances the capabilities of ML and DL models in the automotive sector by providing robust, real-time data analytics, which is essential for maintaining vehicle safety, performance, and reliability.

4.4 Industrial and Robotics Sector

It is becoming increasingly important to analyze data from machinery in real time to predict failures before they occur. Similarly, monitoring production lines and energy usage is critical to ensure higher quality with reduced cost. In this context, TinyML enables robots to process sensory data in real-time for recognition and classification.

One of the TinyML-based framework pioneers, TinyOL, is presented in [102] where anomaly detection using Autoencoders on Arduino Nano 33 BLE Sense board using the TensorFlowLite framework. It processes streaming data, updates running mean and variance, scales input, makes predictions, and updates weights using online gradient descent algorithms. Consequently, an unsupervised autoencoder is transformed into a supervised anomaly classification model. The encoder's output and reconstruction error are used as features for classifying different anomaly patterns. The system's performance is evaluated in terms of fine-tuning (an ability to adapt existing neural network on MCUs to new data) and multi-anomaly classification (an ability to replace the final layer of an autoencoder with TinyOL, enabling post-training in an online mode).

The work in [103] leverages the W3C Web of Things (WoT) to semantically express IoT device capabilities through thing descriptions (TD). The TD describes IoT devices' metadata and interactions in a standardized format. It introduces semantic models for on-device applications, specifically for neural networks (NN) and complex event processing (CEP) rules. Subsequently, the enriched semantic knowledge is hosted to discover and interoperate edge devices and applications across decentralized networks using a knowledge graph (KG). The case study of the framework is the implementation of a conveyor belt to monitor operational processes and detect irregularities. An Arduino board connected to a camera uses a CNN to detect the presence of workers near a workstation. The CNN processes the image data to determine whether a worker is present.

A powerful tool for controlling dynamic robotic systems with complex constraints is presented in [104], where a high-speed model-predictive control solver (named TinyMPC) is implanted using the Alternating Direction Method of Multipliers (ADMM) algorithm. The ADMM is used to solve convex optimization problems. The demonstrated applications are high-speed trajectory tracking, dynamic obstacle avoidance, and recovery from extreme attitudes. Moreover, the Teensy 4.1 Development Board is used to benchmark TinyMPC against randomly generated trajectory tracking problems. On the other hand, Crazyflie 2.1 Quadrotor demonstrates TinyMPC's performance in real-time dynamic control tasks, including figure-eight trajectory tracking, recovery from extreme initial attitudes, and dynamic obstacle avoidance.

The work in [105] presents a transfer learning framework combined with TinyML-powered CNN architecture for vibration-based fault diagnosis of different machines. Various time-domain features are extracted from vibration signals for fault diagnosis. Features include mean, median, variance, standard deviation, skew, kurtosis, crest, impulse, and shape factors. While the conventional transfer learning strategy is to retain dense layers while freezing convolutional layers, the work in [19] retains convolutional layers while freezing dense layers. To achieve memory efficiency, it retains only the biases of hidden layers. For edge implementation, the online training is conducted on a Raspberry Pi single-board computer, while the edge inference is performed on an ESP32 microcontroller board using TensorFlow Lite.

Another application of intelligent IoT systems in industry and robotics is ensuring the structural integrity of steel constructions through climbing inspection robots, as presented in [106]. The work introduces a real-time bolt-defect detection system using TinyML and a magnetic climbing inspection robot. The magnetic climbing robot has 3D-printed wheels embedded with permanent magnets for secure adhesion to metallic surfaces. The system employs the Faster Objects, More Objects (FOMO) algorithm optimized for edge computing on microcontrollers. It captures images, processes them using the FOMO model, and streams annotated images to the user via the real-time streaming protocol (RTSP). The FOMO model is simplified for multi-object classification and optimized for microcontrollers. Images of bolts in various conditions (normal, loose, missing) were collected and used to train the model.

TinyML is revolutionizing the industrial and robotics sectors through various innovative applications. The TinyOL framework enhances anomaly detection, and W3C WoT standardizes IoT device capabilities, enabling efficient monitoring and worker detection. Similarly, TinyMPC, a high-speed control solver, improves dynamic robotic tasks like trajectory tracking and obstacle avoidance. Lastly, climbing inspection robots utilize TinyML and the FOMO algorithm for real-time bolt-defect detection, demonstrating effective multi-object classification on microcontrollers. These advancements highlight TinyML's potential to improve industrial and robotic systems significantly.

4.5 Energy Sector

Integrating digital technologies is important for managing energy resources efficiently and sustainably, especially when updating old systems. It involves using sensors and devices to monitor and control energy accurately. The growing reliance on renewable energy requires efficient maintenance of large-scale photovoltaic (PV) solar plants. The faults in PV panels, such as hot spots, can significantly reduce efficiency, and therefore, early detection of faults is critical through some monitoring mechanisms.

The work in [107] presents a framework to classify defects in PV modules. The framework captures infrared (IR) images using a low-cost IR camera and processes these images in real-time. The images are pre-processed to reduce noise and then fed into the Tiny CNN model for classification. The model is optimized and integrated into a low-cost, low-power microcontroller for real-time fault diagnosis. The trained TensorFlow TinyCNN model is converted to a TensorFlow Lite version to optimize it for deployment on a microcontroller. The Arduino-integrated development environment is used for writing and uploading the C++ code to microcontrollers, while the OpenCV-Python library preprocesses IR images. Another microcontroller (Arduino Nano 33 IoT) is used to communicate with the Tiny CNN microcontroller and post the results online. This setup enables remote monitoring and real-time visualization of the PV module status on a dedicated webpage.

The work in [108] employs thermographic images of a PV array for fault diagnosis using two deep convolutional neural network (DCNN) models. A binary classifier model architecture detects whether a PV module is faulty. Moreover, a multiclass classifier is also used to diagnose the specific type of fault in a PV module. The models are trained using TensorFlow and Keras libraries. The trained models are converted to TensorFlow Lite format to reduce their size and make them suitable for edge-devices deployment. The optimized models are embedded into a Raspberry Pi 4 microprocessor. Python scripts run the models, send notifications via SMS and email, and display results on an LCD. The work in [109] started by making a solar farm dataset, including power generation and weather-related information. Then, this data is preprocessed using min-max scaling. Various machine learning models are trained and evaluated for their performance in predicting solar energy yield, focusing on tuning hyperparameters to optimize accuracy. Additionally, the study explores the deployment of these models on resource-constrained edge devices using TinyML to enable real-time, low-cost forecasting.

The objective of the work in [110] is to develop a privacy-preserving architectural framework for hybrid energy management systems (HEMS) that leverages TinyML models for short-term energy consumption prediction on mobile devices. This approach aims to enhance user privacy, ensure efficient energy management, and provide real-time, on-device processing capabilities. It employs LSTM neural networks to predict short-term energy consumption in hybrid energy management systems. These models are converted to CoreML and TensorFlow Lite formats to enable deployment on mobile devices. TensorFlow Lite shows minimal performance degradation and thus is more suitable for real-time, on-device processing. The hardware used in the study includes mobile devices, specifically the iPhone 12 Pro, which serves as an edge device for processing and storing data locally. Finally, the work in [111] presents a framework to upgrade the energy distribution panel of an old manufacturing plant by installing sensor devices. These sensors enable remote monitoring and decentralize predictive analysis. The analysis uses 15-min energy demand forecast models based on two-layer LSTM networks.

The key findings from the aforementioned works highlight the innovative use of TinyML in various applications. A framework for classifying PV module defects uses IR images processed by Tiny CNN models for real-time fault diagnosis. Thermographic images of PV arrays are used with DCNN models for fault detection and classification. A privacy-preserving framework for hybrid energy management systems uses LSTM models on mobile devices to predict short-term energy consumption, ensuring efficient energy management. Lastly, sensor devices installed in an old manufacturing plant enable remote monitoring and predictive analysis using LSTM networks, improving energy management. These findings demonstrate TinyML's potential in enhancing real-time monitoring, fault diagnosis, and energy management across various sectors.

4.6 Consumer Electronics and Safety in Smart Cities

All the above categories (smart agriculture, smart healthcare, smart vehicles, smart industrial processes, and smart energy management) have significantly enhanced our living standards. All these smart processes have given birth to "smart cities". The concept of smart cities also includes consumer electronics in smart homes and the security of all the automated processes. This section will review some state-of-the-art consumer electronics in smart homes and the security of some automated processes in smart homes. Consumer electronics in smart homes include devices that can be controlled remotely via smartphones or voice assistants. Security in smart cities includes surveillance systems, emergency response systems, advanced access control systems, cybersecurity, etc. Existing review articles, such as [118], review the technical advancements in consumer electronics and smart cities. However, they purely discuss smart cities in terms of IoT, machine learning, cloud computing, and edge computing. This section reviews some state-of-the-art IIoT in consumer electronics and security.

The work in [112] enhances the detection of jamming attacks in IoT wireless networks using a CNN-based approach deployed on edge devices. The model is trained to classify two types of jamming attacks (constant and periodic). Received Signal Strength (RSS) data is the primary metric for detecting and classifying jamming attacks. The data is collected for different scenarios, including normal channel conditions and two types of jamming attacks. Finally, the trained model is deployed using TensorFlow Lite on edge devices like the Raspberry Pi. The deployed model performs real-time inference on RSS data, detecting and classifying jamming attacks. Another CNN-based approach is presented in [113], where the objective is to classify the usage of handheld power tools. The model was trained and validated using a dataset of over 280 min of three-axis accelerations during different activities (tool transportation, no-load, metal

drilling, and wood drilling). The trained CNN model is converted to TensorFlow Lite format using post-training quantization (to convert the 32-bit floating-point weights to 8-bit integers), reducing the model size significantly. The converted model is deployed on the NRF52832 board.

The work in [114] presents a framework (TinyAP) for detecting and mitigating attacks at the access point level in a Wi-Fi network. The objective is to ensure the safety and privacy of smart home devices. It employs DNN and LSTM, trained on a general-purpose computer, and then converted into a format suitable for deployment on microcontrollers. While the work in [114] focuses on the security of smart homes, the security of medical IoT systems is discussed in [115], enabling real-time health monitoring and intrusion detection. It addresses privacy and security concerns using edge processing for data encryption and TinyML for real-time vital sign analysis and emergency alerts. Moreover, Naive Bayes and SVM are used for real-time analysis of vital signs. However, the article does not mention the type of hardware used for monitoring and processing real-time vital signs (e.g., blood pressure and heart rate). The framework's encryption and intrusion detection capabilities help maintain data integrity and privacy.

Finally, the work in [116] presents an energy-harvesting resource allocation algorithm for UAV-assisted TinyML consumer electronics in low-power IoT networks. Optimizing energy harvesting and resource allocation in IoT networks involves handling interference, resource conflicts, and real-time decision-making. Therefore, Naive Bayes, SVM, RF, and KNN are efficient for classification and regression tasks in resource-constrained environments. However, the article did not provide the corresponding hardware and software details for the model deployment.

To summarize, the key applications in smart home security and consumer electronics are the detection of jamming attacks in IoT wireless networks, the usage of handheld power tools, detecting and mitigating attacks at the access point level in Wi-Fi networks, the security of medical IoT systems and energy-harvesting resource allocation algorithm for UAV-assisted TinyML consumer electronics in low-power IoT networks, optimizing energy use and resource allocation. These findings demonstrate the potential of TinyML and edge computing to enhance the security and efficiency of smart cities.

5 Responses to Formulated Research Question

This section formulates responses to the target research questions (identified in the Introduction Section). The responses to research questions are based on the results of Sections 3 and 4.

Research Question 1: What are the state-of-the-art model conversion techniques used in EdgeML, and how do they impact the performance, efficiency, and deployment of machine learning models on resource-constrained devices?

Answer: This work has classified state-of-the-art model conversion techniques in EdgeML into four categories: Pruning, quantization, low-rank factorization, and knowledge distillation. Their impact on performance, efficiency, and model deployment has been discussed in Section 3. The pruning methods have been compared in Table 5. Results on quantization methods and quantization-aware training have been summarized in Tables 6 and 7, respectively. Similarly, model conversion methods based on low-rank factorization and knowledge distillation methods have been analyzed in Tables 8 and 9, respectively.

Research Question 2: What are the current state-of-the-art inference mechanisms used in EdgeML, and how do they compare in performance and efficiency?

Answer: Edge devices can perform inference in two ways once a model is deployed: either entirely on the device (on-device inference) or in collaboration with other edge devices or servers (distributed inference). Sections 3.2.1 and 3.2.2 detail these methods, and Table 10 summarizes the different inference mechanisms.

Research Question 3: How do different learning strategies impact the performance and efficiency of ML models deployed in edge computing environments?

Answer: The results of our study on different learning strategies are categorized into three main areas: (a) online learning strategies, (b) continual or life-long learning, and (c) federated learning. Sections 3.3.1–3.3.3 present the findings for each area, respectively. Additionally, Table 11 summarizes the results for continual learning, while Table 12 summarizes federated learning. A comprehensive analysis of selected research areas in these sections and tables highlights how various learning strategies impact the performance and efficiency of ML models deployed in edge computing environments.

Research Question 4: What are the key challenges and problems and the associated ML/DL models in deploying TinyML on resource-constrained devices in various sectors?

Answer: Section 4 presents the results of deploying TinyML models on resource-constrained devices organized in different sectors to reflect TinyML's diverse applications. Consequently, we have identified six key sectors where TinyML-based model deployment has shown promising results such that Sections 4.1–4.6 provide critical analysis of the selected research studies in smart agriculture, healthcare, vehicles, industry, energy, and security sectors, respectively. Moreover, Table 13 lists various applications and the associated ML/DL models used in each sector.

Research Question 5: What are the latest advancements in hardware, software frameworks, and sensors designed explicitly for TinyML frameworks?

Answer: The synthesized results for six different TinyML sectors are detailed in Tables 14–16. Table 14 summarizes 15 hardware development boards from the selected research works, highlighting their features and organizing the articles by hardware development. Table 15 categorizes the selected research works by three major software frameworks. Finally, Table 16 details the sensors used in all the research papers selected.

6 Discussion and Limitations of the Literature Review

Section 5 has answered the target research questions (listed in the Introduction) by utilizing the results in Sections 3 and 4. This section further discusses some important aspects of model design and deployment. Mainly, it includes automating quantization and pruning policies for efficient model conversion (Section 6.1), an in-depth discussion on fault resilience in distributed edge machine learning models (Section 6.2), the influence of hardware constraints on edge devices (Section 6.3), ethical concerns of deployment of Edge ML (Section 6.4) and limitations of this literature review (Section 6.5).

6.1 Automating Quantization & Pruning Policies for Efficient Model Conversion

Section 3 reveals that converting models for edge devices involves transforming large, complex models into smaller, efficient versions suitable for resource-constrained hardware. Common techniques include quantization, pruning, low-rank factorization, and knowledge distillation. Quantization, the most widely used technique, can be applied post-training or during training (quantization-aware training). Post-training quantization is easier to implement but may reduce accuracy, while quantization-aware training maintains better accuracy. Pruning, the second most common technique, offers better accuracy but requires more memory and computation. Low-rank factorization and knowledge distillation are less common due to their complexity and longer training times.

Hernandez et al. [119] have provided an interesting theoretical analysis of the generalization of the quantization approaches and studied an algorithm-independent generalization error bound. However, many quantization-aware training algorithms show efficacy through practical applications with lesser theoretical proof of the performance bound of asymptotic convergence analysis. Both quantization-aware training

and post-training quantization have advantages and disadvantages. Post-training quantization is easy to implement and does not usually require retraining. So, in situations where we have a trained model and we don't have computational resources and/or time for training the model, post-training quantization is a good option. A quantization policy during the training can maintain higher model performance in quantization-aware training. Moreover, a model can be designed according to the hardware requirements. It will require additional time and computational resources.

In addition to the results in Section 3, automating quantization and pruning policies according to hardware constraints is crucial in model conversion. In [120], the quantization policy is defined as an optimization problem aimed at minimizing cross-entropy loss while adhering to a hardware budget constraint. This constraint limits the number of bits assigned to the model's weight tensors within the hardware budget. On the MNIST dataset, the optimized average bit size of the weights is 1.46, with no accuracy loss, achieving a compression rate of over 2000 on LeNet. Similarly, hardware-aware automated quantization in [121] uses reinforcement learning to optimize policies based on latency and energy feedback. The ResPrune method [122] is another technique that uses stochastic optimization to prune filters, saving over 50% of FLOPS on various datasets. An evolutionary pruning model [123] optimizes pruned models for better classification results. Chen et al. [124] have proposed a two-stage framework for optimizing compression ratios, achieving 3 to 36 times compression with 2 to 3-bit quantization. Guo et al. [125] have introduced a multi-agent reinforcement learning framework for automatic pruning, achieving 30% to 50% compression. Albanese et al. [126] used pruned and quantized MobileNetV2 and SqueezeNet models to detect artifacts in plastic components with high accuracy and inference rates.

6.2 Fault Resilience in Distributed Edge Machine Learning Models

It has been observed from Section 3 that inference can be performed entirely on the device (on-device inference) or in collaboration with other devices or servers (distributed inference). For this purpose, EdgeML incorporates various learning strategies to optimize performance and resource usage, including (a) online learning, (b) continual learning, and (c) federated learning. While distributed edge machine learning solutions using model conversion techniques are highly effective, these frameworks are also susceptible to various sources of errors and faults [127]. Therefore, the purpose of fault-resilient edge machine learning models is to maintain the performance and functionality of the model despite various types of errors and faults.

There are many sources of faults and errors in these frameworks [128], such as data-related issues, which include noisy data, incomplete data, and drift in the data distribution used to train the models. Similarly, ensuring redundancy in the inference of edge machine learning models is crucial for maintaining reliability and performance. It implies that if one edge device fails, others can take over its tasks without significant disruption. Communication channels are another critical point where errors can be introduced in edge machine learning systems. During data transfer between edge devices or between edge devices and servers, errors such as packet loss, corruption, or delays can occur. It can lead to incomplete or inaccurate data being processed.

Identifying and diagnosing faults early in the framework can prevent incorrect inferences or system failures. Making the system fault-tolerant involves error correction codes, inherent model design, and decentralized or distributed management [129]. Decentralized management distributes decision-making across multiple nodes, reducing the impact of individual node failures [127]. Deep learning models are programmed and deployed on the hardware, assuming that input data is genuine and error-free, there is no error or bug in the program, and the hardware is provided as described without any pre-deployment or post-deployment fault. In reality, the input data may be corrupt or noisy due to failure or malfunction of sensors

or adversarial attacks [130]. Programs may have bugs or data-oriented wrong calculations, and hardware may face failures due to environmental or structural effects. The fault resilience of the deep learning models should be analyzed against these types of faults, reflecting the models' reliability under such circumstances. A systematic review of designing a framework for fault injection can be found in [131]. In the following, we discuss some research works focusing on fault injection.

Narayanan et al. [132] have proposed high-level fault injection frameworks for TensorFlow-based applications. Similarly, Laster et al. [133] have studied the effect of transient hardware faults on the misclassification of DNN. Syed et al. [134] investigated the impact of faults on the deep neural network at different quantization levels. They have found that good quantization of the model increases the resiliency of the DNN. Ruospo et al. [135] have investigated the effect of fixed and floating point quantization of CNN on reliability. An open-source fault injection framework, darknet, tests the CNN resilience against the faults. They have tested LeNet and YOLO architectures by injecting faults at different layers. They have concluded that fixed point data quantization provides a better tradeoff between memory footprint reduction and resilience.

Liu et al. [136] have proposed a distribution-based error detector to improve the bit error resilience of DNN. They have used memory errors and register fault injectors. The results on LeNet and AlexNet showed that DED improved the error resilience of the models. The effect of the two pruning methods, namely magnitude-based pruning and filter-based structured pruning, on the reliability of the DNN deployed on FPGA is studied in [137]. A hardware injection tool is used for reliability evaluations.

Furthermore, the effect of weight quantization is also investigated. The classification accuracy of VGG16 on the CIFAR-10 dataset is studied for different quantization levels and pruning methods. They have found that 8-bit quantization is a better option, and reliability does not increase much as we increase the bit-width to be larger than 8. Filter-based structured pruning method is less reliable than magnitude-based pruning. DNN with higher pruning rates is more robust to weight errors but less reliable to errors on the configuration bits.

6.3 Influence of Hardware Constraints on Edge Devices

The specific hardware constraints of edge devices significantly influence the design and deployment of models in EdgeML and TinyML.

Memory Constraints: Edge devices often have limited memory, necessitating model optimization techniques to reduce the size of machine learning models. Techniques such as pruning, quantization, and knowledge distillation are commonly used to compress models without significantly compromising accuracy. For instance, pruning removes unnecessary parameters, while quantization reduces the precision of the model weights, both of which help in fitting the model within the limited memory available on edge devices.

Computational Capabilities: The computational power of edge devices varies widely. Devices with limited computational capabilities require computationally efficient models. It often involves designing lightweight models or using specialized architectures like MobileNets or SqueezeNet, optimized for low-power and low-latency inference. Additionally, techniques such as model partitioning can distribute the computational load across multiple devices or offload some tasks to more powerful servers, balancing the computational requirements.

Energy Consumption: Energy efficiency is crucial for edge devices, especially those that rely on battery power. Models deployed on these devices must be optimized to minimize energy consumption. It can be achieved through techniques like event-driven computing, where the device only activates when necessary, and by using energy-efficient algorithms that reduce the number of computations required. For example, event-driven computing can significantly extend battery life by avoiding constant data processing.

6.4 Ethical Concerns of Deployment of Edge ML

When machine learning models are deployed on resource-constrained devices such as smartphones, smartwatches, and other IoT devices, handling the privacy issue of sensitive personal data is a significant concern that needs to be addressed. Robust security infrastructure is required to optimize the robustness of the IoT devices against data breaches. This issue has been discussed in many ways, including with the newly developed blockchain technology. Still, incorporating robust security measures in resource-constrained devices is difficult due to limited power availability. The learning of the models with limited access to the training data from a localized environment may create a bias in the models. Such a problem can be handled by sharing the models among the edge devices and placing the central model on the cloud or local servers. Many decisions are made by edge machine learning models in sensitive environments, such as maintaining security or providing healthcare in smart cities. In such cases, transparency and accountability of automated decision-making are more significant ethical concerns.

6.5 Limitations of the Literature Review

Despite strictly following the guidelines and adhering to our review protocol, there are certain limitations: (1) While we used appropriate search terms and thoroughly scanned the results, some terms returned thousands of results that could not be exhaustively reviewed. Additionally, some research was rejected based on titles, which may not accurately reflect the content. Therefore, we do not claim our research is exhaustive. (2) We utilized four renowned scientific databases: IEEE, ELSEVIER, ACM, and SPRINGER, which provide many journal and conference publications. However, other databases also contain significant research work. Consequently, there is a chance we missed relevant recent studies from other sources. Nonetheless, we believe the ultimate findings of this literature review are not significantly affected, as the selected databases provide high-quality, up-to-date research literature.

7 Challenges and Future Directions

Sections 3 and 4 provide results into two major categories: (1) model conversion, inference, and learning in EdgeML and (2) model deployment on resource-constrained devices with TinyML. Based on these results, Sections 5 and 6 have formulated responses to identified research questions and associated discussion. Consequently, this section identifies probable issues due to the rapid increase in smart devices, which are projected to reach 29 billion by 2030 [138]. These devices generate vast amounts of data, which can be utilized to build numerous IIoT systems. However, deploying these models on edge devices has several challenges due to their computational and energy constraints. Here are some key challenges and future research directions:

Resource Optimization for Dynamic and Heterogeneous Edge Environments: The edge environment is becoming increasingly dynamic, heterogeneous, and diverse. Edge devices now operate in various environments, from remote, communication-constrained areas to highly demanding and dangerous situations such as firefighting, battlefields, and harsh weather conditions [139]. Adaptive solutions are needed to dynamically allocate resources based on decision-making requirements, current workload, and communication status among edge devices or between edge devices and edge or cloud servers [140]. Therefore, effective resource management is crucial for end-to-end machine learning applications deployed on autonomous devices. For battery-operated edge devices, extending operational life is also a key concern. Consequently, efficient and innovative computational algorithms are becoming a priority. Developing multi-objective optimal policies for sharing and balancing resources among collaborating edge devices is another important future research direction [141].

Additionally, optimizing the placement of edge nodes and servers is essential to enhance the quality of collaborative tasks for mobile edge devices [142]. We need a suitable prediction mechanism to predict the workload patterns and allocate the resources in the dynamic environment. Resource scheduling requires a robust mechanism in the presence of communication instability and variation of available resources. Reinforcement learning can be a good future direction in solving this problem. In the heterogeneous environment, scalability issues are a significant concern for resource management strategies, which can be handled by solving dynamic scheduling issues.

Resource-Aware Learning and Sustainability: Although models can be compressed to fit on-edge devices, inference, and continual learning (as discussed in Section 3 of this article) still demand additional resources. It implies that maintaining learned knowledge and acquiring new information from dynamic environments is challenging. Analyzing data, selecting important information for learning, and discarding irrelevant data require extra computational resources and energy. When edge devices collaborate or communicate with servers, designing an optimal policy for inference and learning is always challenging [143]. Therefore, balancing acquiring new knowledge with making the model available for inference necessitates task-based scheduling procedures. Moreover, ensuring timeliness and accuracy through specialized model design will be crucial for future edge ML applications [144]. Consequently, resource sustainability will be a significant research topic, as resource-aware federated learning uses neural architecture search to generate deployable models on heterogeneous devices based on their resources. Yu et al. [145] proposed on-demand customized model deployment considering the available resources on edge devices. Extensive research is also required in this direction. Selecting the data relevant to the learning already trained is a primary step in efficient learning without wasting computation resources on learning redundant data. Ge et al. [146] proposed an adaptive personalized FL when the data distribution among the users is homogenous. Hence, the scheme identifies users with homogenous patterns and engages them in collaborative FL using one-shot screening based on the learning loss without communicating the original data. Due to privacy issues related to personal data, it is not safe to transmit this data to the edge server. Hence, specific properties of the dataset on which the machine learning model is trained should be kept and used to predict the novelty of the new information.

New Learning Paradigms and Algorithms: While this literature review has discussed various learning paradigms, including on-device, federated, continual, or life-long, and split model learning, adapting to changing data distributions remains a significant challenge and requires stable learning procedures. The review highlights that substantial work has been done on quantizing and pruning pre-trained models. However, there is still a need for extensive research in quantization-aware and prune-aware training of models. Specialized learning algorithms for quantized models based on multi-objective loss functions could be more beneficial for future model designs [147]. Due to its limited computational resources, TinyML faces additional challenges in incorporating these learning strategies effectively. Addressing these challenges will be crucial for advancing the capabilities and applications of TinyML [148].

Hardware-Oriented Model Design: As EdgeML and TinyML technologies evolve, optimizing hardware to meet the specific needs of various applications and algorithms becomes increasingly important [149]. Custom hardware can be designed to accelerate particular machine learning algorithms, improving performance and efficiency. Tailoring hardware to the needs of specific applications can significantly reduce power consumption. By designing hardware optimized for specific tasks, it is possible to reduce latency, enabling real-time data processing and decision-making. However, creating hardware optimized for particular applications can limit its scalability across different use cases, requiring multiple designs for various applications. Moreover, developing custom hardware can be expensive and time-consuming, which may not be feasible for all organizations. In the hardware-oriented model design, hardware constraints can be identified, and

a multi-objective optimization algorithm can be used to achieve a tradeoff between model complexity and hardware design.

Better Interoperability and Scalability: Section 4 of this LITERATURE REVIEW has highlighted that the EdgeML and TinyML paradigms are applied to diverse applications (such as healthcare, automotive, energy, vehicles, industry, and energy) in a dynamic and varied environment. The diversity of edge devices, each with unique constraints, adds complexity to EdgeML and TinyML framework [11]. In collaborative inference and learning, these edge devices must work together to complete tasks despite their hardware and software variability. This heterogeneity necessitates standard protocols for integrating information and learning. Recent research [150–152] has addressed some interoperability issues, but much more work is needed in this area. Scalability is another significant concern in deploying edge computing solutions. Computation offloading to edge servers or other edge devices can improve performance, save energy, and reduce computation time. However, the capacity of edge servers to handle requests from numerous edge devices can constrain performance.

Additionally, the heterogeneity of edge machine learning architectures can lead to scalability challenges. Interoperability can be improved by ensuring the compatibility of the communication protocols among the devices. Hence, the standardization of communication protocols for heterogeneous devices is desirable for future practical applications. Making smart multi-protocol gateways can be a future solution to this problem.

New Computational Algorithms Aiming for Edge Devices: New computational algorithms are required to efficiently deploy and compute machine learning models and are well-suited for limited-resource edge devices. Hyperdimensional computing (HDC) is a new learning paradigm suitable for lesser computation and energy requirements [153]. Hence, efficient machine learning solutions can be developed in the high-dimensional space with lower precision parameters. Moreover, implementing the HDC on diverse hardware platforms is challenging and needs further research and development [154].

New Edge AI Architectures Combining Cloud, Fog, and Edge Layers: New edge AI (artificial intelligence) architectures are needed to accommodate the diverse range of edge devices and servers, optimizing synchronization and communication within limited bandwidth constraints. Typically, EdgeML architectures are multi-tiered, comprising multiple computing layers, from edge devices to fog servers and cloud servers. Efficient resource utilization is challenging due to the varying levels of granularity across these layers [155]. Multi-tenant edge AI allows multiple applications to run concurrently, optimizing resource use. However, this architecture faces key challenges in meeting performance requirements such as latency, accuracy, and energy consumption [156].

Data Privacy and Access Security: Edge ML is decentralized in computing and contains various edge devices and servers. Therefore, the vulnerability of edge devices is a big issue. Due to their distributed nature and operation in an untrusted environment, edge devices are prone to various cyber-attacks. Hence, protecting the sensitive data collected by edge devices and stored locally is a significant challenge. Securing the transmission of the data over multiple communication channels is also a challenging task. Moreover, maintaining a dynamic edge ML environment where edge devices may leave or join a dynamic trust model between devices is an issue to tackle [157]. Conventional cryptographical solutions are no longer safe after the emergence of quantum computing. So, in edge machine learning, creating quantum-safe protocols will be a future challenge [158].

Operational Efficiency: While edge devices collect data for inference and learning, misleading data may corrupt the learning by edge devices. So, avoiding the adversarial data and recognizing the legitimate data for learning and inference is also challenging [159]. Adversarial robustness in a machine learning model can be achieved by defining new adversarial learning strategies that can incorporate adversarial examples in the

learning process. In this way, the trained models can recognize and resist adversarial attacks on the continual learning process. Model hardening is another helpful research direction that includes data sanitization, adversarial training, and differential privacy to mitigate the threats. A good survey in this direction has recently been published [17].

Edge ML on Quantum Edge Devices: Integrating EdgeML and TinyML with quantum edge devices is another promising future research direction that aims to utilize the advantages of quantum computing, such as enhanced processing power and speed, to optimize further machine learning models deployed on edge devices [160]. In addition to the ability of quantum computing to solve complex problems more efficiently than classical computing, post-quantum cryptography algorithms offer advanced security features that could protect data processed on edge devices [161]. However, it may require developing quantum algorithms tailored for EdgeML and TinyML applications. Developing specific quantum algorithms for EdgeML and TinyML can unlock new possibilities for real-time data processing and analytics, making these technologies even more powerful and versatile.

Photonic Neural Networks on the Edge Devices: Photonic neural networks use light for computations and leverage important characteristics of light in terms of low energy consumption, parallel implementation, and faster processing speed. Hence, photonic deep neural networks can be a good choice regarding efficiency and performance [162]. However, coherent optical processing is challenging in photonic neural networks [163]. Moreover, deploying the photonic neural network on resource-constrained devices still requires further research in this area, with a major focus on miniaturization, cost efficiency, and integration with other components on edge devices.

Compliance with Regulations and Standards: As EdgeML and TinyML technologies become more integrated into various applications, they must meet regulatory requirements and industry standards for widespread adoption and safe deployment. Edge devices often handle sensitive data, so compliance with data protection regulations like GDPR or CCPA is essential [164]. Similarly, meeting safety standards is vital for applications in critical sectors such as healthcare or automotive. In addition to security and safety standards, interoperability is also important. Furthermore, developing standards that ensure different devices and systems can work together seamlessly is important for the scalability of EdgeML and TinyML solutions [165]. Finally, compliance with environmental regulations regarding energy consumption and electronic waste is becoming increasingly important as the number of deployed devices grows.

8 Conclusion

This literature review has highlighted significant advancements in model conversion, inference mechanisms, and learning strategies in EdgeML, emphasizing their impact on the performance, efficiency, and deployment of machine learning models on resource-constrained devices with TinyML. It identifies and discusses various techniques, such as pruning, quantization, knowledge distillation, and low-rank factorization, along with their respective advantages and challenges. The document also explores different inference mechanisms, including on-device and distributed inference, and various learning strategies like continual learning and federated learning. Furthermore, it delves into deploying TinyML models across different sectors, including smart agriculture, healthcare, automotive, industry, energy, and security. It provides insights into the hardware development boards, software frameworks, and sensors used in these applications, showcasing the versatility and potential of TinyML in real-world scenarios. Despite the significant progress, the review acknowledges several challenges and future research directions. These include resource optimization, new learning paradigms, interoperability, scalability, security, and the integration of quantum computing with EdgeML and TinyML. Addressing these challenges will be crucial for the continued advancement and widespread adoption of these technologies.

In conclusion, this review is a valuable resource for researchers, practitioners, and developers, offering a detailed overview of the current landscape and future directions in EdgeML and TinyML. Practitioners can leverage techniques like pruning, quantization, and knowledge distillation to reduce model size and improve inference speed, making deploying complex models on resource-constrained devices feasible. Similarly, developers can choose between on-device and distributed inference based on the specific application requirements, balancing latency, computational load, and energy consumption. Moreover, continual and federated learning can help maintain model accuracy over time without frequent retraining, particularly useful in dynamic environments. The insights into hardware development boards, software frameworks, and sensors can guide practitioners in selecting the right tools and components for deploying TinyML models in various sectors such as smart agriculture, healthcare, automotive, industry, energy, and security. By incorporating these practical implications, practitioners and developers can effectively apply the advancements discussed in the review to their projects, driving innovation and improving the deployment of machine learning models on resource-constrained devices.

Acknowledgement: The authors acknowledge the Computers, Materials & Continua for their support for the paper.

Funding Statement: The authors received no specific funding for this study.

Author Contributions: The authors confirm their contribution to the paper as follows: study conception and design: Muhammad Arif and Muhammad Rashid; data collection: Muhammad Arif; analysis and interpretation of results: Muhammad Arif and Muhammad Rashid; draft manuscript preparation: Muhammad Arif and Muhammad Rashid. All authors reviewed the results and approved the final version of the manuscript.

Availability of Data and Materials: No dataset is used in the paper.

Ethics Approval: Not applicable.

Conflicts of Interest: The authors declare no conflicts of interest to report regarding the present study.

References

1. Ahmed SF, Alam MSB, Hassan M, Rozbu MR, Ishtiak T, Rafa N, et al. Deep learning modelling techniques: current progress, applications, advantages, and challenges. Artif Intell Rev. 2023;56(11):13521–617. doi:10.1007/s10462-023-10466-8.
2. Oliveira F, Costa DG, Assis F, Silva I. Internet of intelligent things: a convergence of embedded systems, edge computing and machine learning. Internet Things. 2024;26(9):101153. doi:10.1016/j.iot.2024.101153.
3. Lesch V, Züfle M, Bauer A, Iffländer L, Krupitzer C, Kounev S. A literature review of IoT and CPS—what they are, and what they are not. J Syst Softw. 2023;200(3):111631. doi:10.1016/j.jss.2023.111631.
4. Parast FK, Sindhav C, Nikam S, Yekta HI, Kent KB, Hakak S. Cloud computing security: a survey of service-based models. Comput Secur. 2022;114(1):102580. doi:10.1016/j.cose.2021.102580.
5. Singh R, Gill SS. Edge AI: a survey. Internet Things Cyber-Phys Syst. 2023;3(5):71–92. doi:10.1016/j.iotcps.2023.02.004.
6. Duan Q, Huang J, Hu S, Deng R, Lu Z, Yu S. Combining federated learning and edge computing toward ubiquitous intelligence in 6G network: challenges, recent advances, and future directions. IEEE Commun Surv Tutor. 2023;25(9):2892–950. doi:10.1109/COMST.2023.3316615.
7. Cao X, Başar T, Diggavi S, Eldar YC, Letaief KB, Poor HV, et al. Communication-efficient distributed learning: an overview. IEEE J Sel Areas Commun. 2023;41(4):851–73. doi:10.1109/JSAC.2023.3242710.
8. Lu S, Lu J, An K, Wang X, He Q. Edge computing on IoT for machine signal processing and fault diagnosis: a review. IEEE Internet Things J. 2023;10(13):11093–116. doi:10.1109/JIOT.2023.3239944.
9. Immonen R, Hämäläinen T. Tiny machine learning for resource-constrained microcontrollers. J Sens. 2022;2022(1):7437023. doi:10.1155/2022/7437023.

10. Tsoukas V, Gkogkidis A, Boumpa E, Kakarountas A. A review on the emerging technology of TinyML. ACM Comput Surv. 2024;56(10):1–37. doi:10.1145/3661820.
11. Meuser T, Lovén L, Bhuyan M, Patil SG, Dustdar S, Aral A, et al. Revisiting edge AI: opportunities and challenges. IEEE Internet Comput. 2024;28(4):49–59. doi:10.1109/MIC.2024.3383758.
12. Hua H, Li Y, Wang T, Dong N, Li W, Cao J. Edge computing with artificial intelligence: a machine learning perspective. ACM Comput Surv. 2023;55(9):1–35. doi:10.1145/3555802.
13. Hoffpauir K, Simmons J, Schmidt N, Pittala R, Briggs I, Makani S, et al. A survey on edge intelligence and lightweight machine learning support for future applications and services. ACM J Data Inform Qual. 2023;15(2):1–30. doi:10.1145/3581759.
14. Bhalgaonkar S, Munot M. Model compression of deep neural network architectures for visual pattern recognition: current status and future directions. Comput Elect Eng. 2024;116(3):109180. doi:10.1016/j.compeleceng.2024.109180.
15. Grzesik P, Mrozek D. Combining machine learning and edge computing: opportunities, challenges, platforms, frameworks, and use cases. Electronics. 2024;13(3):640. doi:10.3390/electronics13030640.
16. Capogrosso L, Cunico F, Cheng DS, Fummi F, Cristani M. A machine learning-oriented survey on tiny machine learning. IEEE Access. 2024;12(1):23406–26. doi:10.1109/ACCESS.2024.3365349.
17. Paracha A, Arshad J, Farah MB, Ismail K. Machine learning security and privacy: a review of threats and countermeasures. EURASIP J Inf Secur. 2024;2024(1):1–23. doi:10.1186/s13635-024-00158-3.
18. Gou J, Yu B, Maybank SJ, Tao D. Knowledge distillation: a survey. Int J Comput Vis. 2021;129(6):1789–819. doi:10.1007/s11263-021-01453-z.
19. Zhao Z, Barijough KM, Gerstlauer A. Deepthings: distributed adaptive deep learning inference on resource-constrained iot edge clusters. IEEE Trans Comput Aided Des Integr Circuits Syst. 2018;37(11):2348–59. doi:10.1109/TCAD.2018.2858384.
20. Konečný J, McMahan HB, Ramage D, Richtárik P. Federated optimization: distributed machine learning for on-device intelligence. arXiv:161002527. 2016.
21. Li G, Ma X, Wang X, Yue H, Li J, Liu L, et al. Optimizing deep neural networks on intelligent edge accelerators via flexible-rate filter pruning. J Syst Archit. 2022;124(10):102431. doi:10.1016/j.sysarc.2022.102431.
22. He Y, Dong X, Kang G, Fu Y, Yan C, Yang Y. Asymptotic soft filter pruning for deep convolutional neural networks. IEEE Trans Cybern. 2019;50(8):3594–604. doi:10.1109/TCYB.2019.2933477.
23. Kumar A, Shaikh AM, Li Y, Bilal H, Yin B. Pruning filters with L1-norm and capped L1-norm for CNN compression. Appl Intell. 2021;51(2):1152–60. doi:10.1007/s10489-020-01894-y.
24. Yu F, Cui L, Wang P, Han C, Huang R, Huang X. EasiEdge: a novel global deep neural networks pruning method for efficient edge computing. IEEE Internet Things J. 2020;8(3):1259–71. doi:10.1109/JIOT.2020.3034925.
25. Mondal M, Das B, Roy SD, Singh P, Lall B, Joshi SD. Adaptive CNN filter pruning using global importance metric. Comput Vis Image Underst. 2022;222(12):103511. doi:10.1016/j.cviu.2022.103511.
26. Sarvani C, Dubey SR, Ghorai M. UFKT: unimportant filters knowledge transfer for CNN pruning. Neurocomputing. 2022;514(3):101–12. doi:10.1016/j.neucom.2022.09.150.
27. Lu J, Wang R, Zuo G, Zhang W, Jin X, Rao Y. Enhancing CNN efficiency through mutual information-based filter pruning. Digit Signal Process. 2024;151(1):104547. doi:10.1016/j.dsp.2024.104547.
28. Liu Y, Fan K, Zhou W. FPWT: filter pruning via wavelet transform for CNNs. Neural Netw. 2024;179(2):106577. doi:10.1016/j.neunet.2024.106577.
29. Chung K, Lee C, Tsang Y, Wu C, Asadipour A. Multi-objective evolutionary architectural pruning of deep convolutional neural networks with weights inheritance. Inf Sci. 2024;685(8):121265. doi:10.1016/j.ins.2024.121265.
30. Kolf JN, Elliesen J, Damer N, Boutros F. Towards extreme face and periocular recognition model compression with mixed-precision quantization. Eng Appl Artif Intell. 2024;137(5):109114. doi:10.1016/j.engappai.2024.109114.
31. Gong C, Chen Y, Lu Y, Li T, Hao C, Chen D. Minimal loss DNN model compression with vectorized weight quantization. IEEE Trans Comput. 2020;70(5):696–710. doi:10.1109/TC.2020.2995593.
32. Peng H, Wu J, Zhang Z, Chen S, Zhang H-T. Deep network quantization via error compensation. IEEE Trans Neural Netw Learn Syst. 2021;33(9):4960–70. doi:10.1109/TNNLS.2021.3064293.

33. Wu D, Yang W, Zou X, Xia W, Li S, Hu Z, et al. Smart-DNN+: a memory-efficient neural networks compression framework for the model inference. ACM Trans Archit Code Optim. 2023;20(4):1–24. doi:10.1145/3617688.
34. Zhong Y, Zhou Y, Chao F, Ji R. MBQuant: a novel multi-branch topology method for arbitrary bit-width network quantization. Pattern Recognit. 2025;158(7):111061. doi:10.1016/j.patcog.2024.111061.
35. Zhang R, Chung AC. MedQ: lossless ultra-low-bit neural network quantization for medical image segmentation. Med Image Anal. 2021;73(1):102200. doi:10.1016/j.media.2021.102200.
36. Shamim MZM. Hardware deployable edge-AI solution for prescreening of oral tongue lesions using TinyML on embedded devices. IEEE Embedd Syst Lett. 2022;14(4):183–6. doi:10.1109/LES.2022.3160281.
37. Yu X, Park S, Kim D, Kim E, Kim J, Kim W, et al. A practical wearable fall detection system based on tiny convolutional neural networks. Biomed Signal Process Control. 2023;86(4):105325. doi:10.1016/j.bspc.2023.105325.
38. Xu K, Zhang H, Li Y, Zhang Y, Lai R, Liu Y. An ultra-low power tinyml system for real-time visual processing at edge. IEEE Trans Circuits Syst II: Express Briefs. 2023;70(7):2640–4. doi:10.1109/TCSII.2023.3239044.
39. Thonglek K, Takahashi K, Ichikawa K, Nakasan C, Nakada H, Takano R, et al. Automated quantization and retraining for neural network models without labeled data. IEEE Access. 2022;10:73818–34. doi:10.1109/ACCESS.2022.3190627.
40. Zhang R, Chung AC. EfficientQ: an efficient and accurate post-training neural network quantization method for medical image segmentation. Med Image Anal. 2024;97(1):103277. doi:10.1016/j.media.2024.103277.
41. Liu X, Wang T, Yang J, Tang C, Lv J. MPQ-YOLO: ultra low mixed-precision quantization of YOLO for edge devices deployment. Neurocomputing. 2024;574(7):127210. doi:10.1016/j.neucom.2023.127210.
42. Deng L, Jiao P, Pei J, Wu Z, Li G. GXNOR-Net: training deep neural networks with ternary weights and activations without full-precision memory under a unified discretization framework. Neural Netw. 2018;100:49–58. doi:10.1016/j.neunet.2018.01.010.
43. Enderich L, Timm F, Burgard W. SYMOG: learning symmetric mixture of Gaussian modes for improved fixed-point quantization. Neurocomputing. 2020;416:310–5. doi:10.1016/j.neucom.2019.11.114.
44. Lu Q, Jiang W, Xu X, Hu J, Shi Y. Quantization through search: a novel scheme to quantize convolutional neural networks in finite weight space. In: 28th Asia and South Pacific Design Automation Conference (ASP-DAC); 2023; Tokyo, Japan. p. 378–83.
45. Huang Z, Han X, Yu Z, Zhao Y, Hou M, Hu S. Hessian-based mixed-precision quantization with transition aware training for neural networks. Neural Netw. 2024;182(1–4):106910. doi:10.1016/j.neunet.2024.106910.
46. Sharma S, Kang B, Kidambi NV, Mukhopadhyay S. HamQ: hamming weight-based energy aware quantization for analog compute-in-memory accelerator in intelligent sensors. IEEE Sens J. 2024. doi:10.1109/JSEN.2024.3382479.
47. Jung S, Son C, Lee S, Son J, Han J-J, Kwak Y, et al. Learning to quantize deep networks by optimizing quantization intervals with task loss. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition; 2019; Long Beach, CA, USA. p. 4345–54.
48. Sun W, Chen S, Huang L, So HC, Xie M. Deep convolutional neural network compression via coupled tensor decomposition. IEEE J Sel Top Signal Process. 2020;15(3):603–16. doi:10.1109/JSTSP.2020.3038227.
49. Swaminathan S, Garg D, Kannan R, Andres F. Sparse low rank factorization for deep neural network compression. Neurocomputing. 2020;398(11):185–96. doi:10.1016/j.neucom.2020.02.035.
50. Nekooei A, Safari S. Compression of deep neural networks based on quantized tensor decomposition to implement on reconfigurable hardware platforms. Neural Netw. 2022;150:350–63. doi:10.1016/j.neunet.2022.02.024.
51. Chen S, Zhou J, Sun W, Huang L. Joint matrix decomposition for deep convolutional neural networks compression. Neurocomputing. 2023;516(3):11–26. doi:10.1016/j.neucom.2022.10.021.
52. Hsiao T-Y, Chang Y-C, Chou H-H, Chiu C-T. Filter-based deep-compression with global average pooling for convolutional networks. J Syst Archit. 2019;95(3):9–18. doi:10.1016/j.sysarc.2019.02.008.
53. Hussain I, Tan S, Huang J. A knowledge distillation based deep learning framework for cropped images detection in spatial domain. Signal Process Image Commun. 2024;124(24):117117. doi:10.1016/j.image.2024.117117.
54. Dai C, Lu S, Liu C, Guo B. A light-weight skeleton human action recognition model with knowledge distillation for edge intelligent surveillance applications. Appl Soft Comput. 2024;151(10):111166. doi:10.1016/j.asoc.2023.111166.

55. Amjad K, Asif S, Waheed Z, Guo Y. A novel lightweight deep learning framework with knowledge distillation for efficient diabetic foot ulcer detection. Appl Soft Comput. 2024;167(1):112296. doi:10.1016/j.asoc.2024.112296.
56. Xu X, Tang S, Zhu M, He P, Li S, Cao Y. A novel model compression method based on joint distillation for deepfake video detection. J King Saud Univ Comput Inf Sci. 2023;35(9):101792. doi:10.1016/j.jksuci.2023.101792.
57. Ham G, Cho Y, Lee J-H, Kang M, Choi G, Kim D. Difficulty level-based knowledge distillation. Neurocomputing. 2024;606:128375. doi:10.1016/j.neucom.2024.128375.
58. Xie J, Gong L, Shao S, Lin S, Luo L. Hybrid knowledge distillation from intermediate layers for efficient single image super-resolution. Neurocomputing. 2023;554(7):126592. doi:10.1016/j.neucom.2023.126592.
59. Niyaz U, Sambyal AS, Bathula DR. Leveraging different learning styles for improved knowledge distillation in biomedical imaging. Comput Biol Med. 2024;168(12):107764. doi:10.1016/j.compbiomed.2023.107764.
60. Fan X, Zhou W. Multidimensional knowledge distillation for multimodal scene classification of remote sensing images. Digit Signal Process. 2024;157:104876. doi:10.1016/j.dsp.2024.104876.
61. Nooruddin S, Islam MM, Karray F, Muhammad G. A multi-resolution fusion approach for human activity recognition from video data in tiny edge devices. Inf Fusion. 2023;100(3):101953. doi:10.1016/j.inffus.2023.101953.
62. Xu M, Qian F, Zhu M, Huang F, Pushp S, Liu X. DeepWear: adaptive local offloading for on-wearable deep learning. IEEE Trans Mob Comput. 2019;19(2):314–30. doi:10.1109/TMC.2019.2893250.
63. Chen X, Li M, Zhong H, Ma Y, Hsu C-H. DNNOff: offloading DNN-based intelligent IoT applications in mobile edge computing. IEEE Trans Ind Inform. 2021;18(4):2820–9. doi:10.1109/TII.2021.3075464.
64. Gauttam H, Pattanaik KK, Bhadauria S, Nain G, Prakash PB. An efficient DNN splitting scheme for edge-AI enabled smart manufacturing. J Ind Inf Integr. 2023;34(14):100481. doi:10.1016/j.jii.2023.100481.
65. Xue M, Wu H, Peng G, Wolter K. DDPQN: an efficient DNN offloading strategy in local-edge-cloud collaborative environments. IEEE Trans Serv Comput. 2021;15(2):640–55. doi:10.1109/TSC.2021.3116597.
66. Chen Y, Yu Z, Jin Y, Mwase C, Hu X, Da Xu L, et al. Self-aware collaborative edge inference with embedded devices for IIoT. Future Gener Comput Syst. 2025;163:107535. doi:10.1016/j.future.2024.107535.
67. Palena M, Cerquitelli T, Chiasserini CF. Edge-device collaborative computing for multi-view classification. Comput Netw. 2024;254(8):110823. doi:10.1016/j.comnet.2024.110823.
68. Guo X, Jiang Q, Pimentel AD, Stefanov T. Model and system robustness in distributed CNN inference at the edge. Integration. 2025;100(1):102299. doi:10.1016/j.vlsi.2024.102299.
69. Khan MA, Hamila R, Erbad A, Gabbouj M. Distributed inference in resource-constrained iot for real-time video surveillance. IEEE Syst J. 2022;17(1):1512–23. doi:10.1109/JSYST.2022.3198711.
70. Wang J, He D, Castiglione A, Gupta BB, Karuppiah M, Wu L. PCNN CEC: efficient and privacy-preserving convolutional neural network inference based on cloud-edge-client collaboration. IEEE Trans Netw Sci Eng. 2022;10(5):2906–23. doi:10.1109/TNSE.2022.3177755.
71. Li C-H, Jha NK. DOCTOR: a multi-disease detection continual learning framework based on wearable medical sensors. ACM Trans Embed Comput Syst. 2024;23(5):1–33. doi:10.1145/3679050.
72. Pellegrini L, Graffieti G, Lomonaco V, Maltoni D. Latent replay for real-time continual learning. In: 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS); 2020; Las Vegas, NV, USA. p. 10203–9.
73. Ravaglia L, Rusci M, Nadalini D, Capotondi A, Conti F, Benini L. A tinyml platform for on-device continual learning with quantized latent replays. IEEE J Emerg Sel Top Circuits Syst. 2021;11(4):789–802. doi:10.1109/JETCAS.2021.3121554.
74. Deutel M, Hannig F, Mutschler C, Teich J. On-device training of fully quantized deep neural networks on Cortex-M microcontrollers. IEEE Trans Comput Aided Des Integr Circuits Syst. 2024. doi:10.1109/TCAD.2024.3484354.
75. Xia Z, Kim J, Kang M. LEAF: an adaptation framework against noisy data on edge through ultra low-cost training. In: Proceedings of the 61st ACM/IEEE Design Automation Conference; 2024; San Francisco, CA, USA. p. 1–6.
76. Novoa-Paradela D, Fontenla-Romero O, Guijarro-Berdiñas B. Fast deep autoencoder for federated learning. Pattern Recognit. 2023;143(2):109805. doi:10.1016/j.patcog.2023.109805.
77. Zhang H, Bosch J, Olsson HH. Enabling efficient and low-effort decentralized federated learning with the EdgeFL framework. Inf Softw Tech. 2025;178(3):107600. doi:10.1016/j.infsof.2024.107600.

78. Yang Z, Zhang S, Li C, Wang M, Wang H, Zhang M. Efficient knowledge management for heterogeneous federated continual learning on resource-constrained edge devices. Future Gener Comput Syst. 2024;156:16–29. doi:10.1016/j.future.2024.02.018.
79. Qiang X, Hu Y, Chang Z, Hamalainen T. Importance-aware data selection and resource allocation for hierarchical federated edge learning. Future Gener Comput Syst. 2024;154(6):35–44. doi:10.1016/j.future.2023.12.014.
80. Cao S, Wu H, Wu X, Ma R, Wang D, Han Z, et al. FedDA: resource-adaptive federated learning with dual-alignment aggregation optimization for heterogeneous edge devices. Future Gener Comput Syst. 2025;163(3):107551. doi:10.1016/j.future.2024.107551.
81. Wang T, Zheng Z, Lin F. Federated learning framework based on trimmed mean aggregation rules. Expert Syst Appl. 2025;270(11):126354. doi:10.1016/j.eswa.2024.126354.
82. Xu J, Wang S, Wang L, Yao AC-C. FedCM: federated learning with client-level momentum. arXiv:210610874. 2021.
83. Thorgeirsson AT, Gauterin F. Probabilistic predictions with federated learning. Entropy. 2020;23(1):41. doi:10.3390/e23010041.
84. Chen S, Shen C, Zhang L, Tang Y. Dynamic aggregation for heterogeneous quantization in federated learning. IEEE Trans Wirel Commun. 2021;20(10):6804–19. doi:10.1109/TWC.2021.3076613.
85. Akhtarshenas A, Vahedifar MA, Ayoobi N, Maham B, Alizadeh T, Ebrahimi S, et al. Federated learning: a cutting-edge survey of the latest advancements and applications. Comput Commun. 2024;228:107964. doi:10.1016/j.comcom.2024.107964.
86. Karunathilake E, Le AT, Heo S, Chung YS, Mansoor S. The path to smart farming: innovations and opportunities in precision agriculture. Agriculture. 2023;13(8):1593. doi:10.3390/agriculture13081593.
87. Tsoukas V, Gkogkidis A, Kakarountas A. A tinyml-based system for smart agriculture. In: Proceedings of the 26th Pan-Hellenic Conference on Informatics; 2022; Athens, Greece. p. 207–12.
88. Gookyi DAN, Wulnye FA, Arthur EAE, Ahiadormey RK, Agyemang JO, Agyekum KO-BO, et al. TinyML for smart agriculture: comparative analysis of TinyML platforms and practical deployment for maize leaf disease identification. Smart Agri Technol. 2024;8(2):100490. doi:10.1016/j.atech.2024.100490.
89. Bhattacharya S, Pandey M. Deploying an energy efficient, secure & high-speed sidechain-based TinyML model for soil quality monitoring and management in agriculture. Expert Syst Appl. 2024;242(5):122735. doi:10.1016/j.eswa.2023.122735.
90. Wulnye FA, Arthur EAE, Gookyi DAN, Asiedu DKP, Wilson M, Agyemang JO. TinyML implementation on microcontrollers: the case of maize leaf disease identification. In: 2024 Conference on Information Communications Technology and Society (ICTAS); 2024; Durban, South Africa. p. 180–5.
91. Dockendorf C, Mitra A, Mohanty SP, Kougianos E. Lite-Agro: exploring light-duty computing platforms for IoAT-Edge AI in plant disease identification.In. In: IFIP International Internet of Things Conference; 2023; Denton, TX, USA. p. 371–80.
92. Azevedo MB, de Medeiros TA, Medeiros MA, Silva I, Costa DG. Detecting face masks through embedded machine learning algorithms: a transfer learning approach for affordable microcontrollers. Mach Learn Appl. 2023;14(10):100498. doi:10.1016/j.mlwa.2023.100498.
93. Saha B, Samanta R, Ghosh S, Roy RB. BandX: an intelligent IoT-band for human activity recognition based on TinyML. In: Proceedings of the 24th International Conference on Distributed Computing and Networking; 2023; Kharagpur, India. p. 284–5.
94. Arthi R, Krishnaveni S. Optimized tiny machine learning and explainable AI for trustable and energy-efficient fog-enabled healthcare decision support system. Int J Comput Intell Syst. 2024;17(1):229. doi:10.1007/s44196-024-00631-4.
95. Gaud N, Rathore M, Suman U. MHCNLS-HAR: multi-headed CNN-LSTM based human activity recognition leveraging a novel wearable edge device for elderly health care. IEEE Sens J. 2024;24(21):35394–405. doi:10.1109/JSEN.2024.3450499.
96. Sun B, Bayes S, Abotaleb AM, Hassan M. The case for TinyML in healthcare: CNNs for real-time on-edge blood pressure estimation. In: Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing; 2023; Tallinn, Estonia. p. 629–38.

97. Andrade P, Silva I, Diniz M, Flores T, Costa DG, Soares E. Online processing of vehicular data on the edge through an unsupervised TinyML regression technique. ACM Trans Embed Comput Syst. 2024;23(3):1–28. doi:10.1145/3591356.
98. Andrade P, Silva M, Medeiros M, Costa DG, Silva I. TEDA-RLS: a TinyML incremental learning approach for outlier detection and correction. IEEE Sens J. 2024;24(22):38165–73. doi:10.1109/JSEN.2024.3458917.
99. Im H, Lee S. TinyML-based intrusion detection system for in-vehicle network using convolutional neural network on embedded devices. IEEE Embedd Syst Lett. 2024. doi:10.1109/LES.2024.3475470.
100. Saini M, Adebayo SO, Arora V. IoT-Fog-based framework to prevent vehicle-road accidents caused by self-visual distracted drivers. Multimed Tools Appl. 2024;83(42):90133–51. doi:10.1007/s11042-024-19050-w.
101. Medeiros M, Flores T, Silva M, Silva I. A multi-layered methodology for driver behavior analysis using TinyML and edge computing. In: 2024 IEEE International Conference on Evolving and Adaptive Intelligent Systems (EAIS); 2024; Madrid, Spain. p. 1–8.
102. Ren H, Anicic D, Runkler TA. Tinyol: TinyML with online-learning on microcontrollers. In: 2021 International Joint Conference on Neural Networks (IJCNN); 2021; Shenzhen, China. p. 1–8.
103. Ren H, Anicic D, Runkler TA. Towards semantic management of on-device applications in industrial IoT. ACM Trans Internet Technol. 2022;22(4):1–30. doi:10.1145/3510820.
104. Nguyen K, Schoedel S, Alavilli A, Plancher B, Manchester Z. TinyMPC: model-predictive control on resource-constrained microcontrollers. In: 2024 IEEE International Conference on Robotics and Automation (ICRA); 2024; Yokohama, Japan. p. 1–7.
105. Asutkar S, Chalke C, Shivgan K, Tallur S. TinyML-enabled edge implementation of transfer learning framework for domain generalization in machine fault diagnosis. Expert Syst Appl. 2023;213(1):119016. doi:10.1016/j.eswa.2022.119016.
106. Lin T-H, Chang C-T, Putranto A. Tiny machine learning empowers climbing inspection robots for real-time multiobject bolt-defect detection. Eng Appl Artif Intell. 2024;133(12):108618. doi:10.1016/j.engappai.2024.108618.
107. Ksira Z, Mellit A, Blasuttigh N, Pavan AM. A novel embedded system for real-time fault diagnosis of photovoltaic modules. IEEE J Photovolt. 2024;14(12):354–62. doi:10.1109/JPHOTOV.2024.3359462.
108. Mellit A. An embedded solution for fault detection and diagnosis of photovoltaic modules using thermographic images and deep convolutional neural networks. Eng Appl Artif Intell. 2022;116(5668):105459. doi:10.1016/j.engappai.2022.105459.
109. Hayajneh AM, Alasali F, Salama A, Holderbaum W. Intelligent solar forecasts: modern machine learning models & TinyML role for improved solar energy yield predictions. IEEE Access. 2024;12:10846–64. doi:10.1109/ACCESS.2024.3354703.
110. Boiko O, Komin A, Shendryk V, Malekian R, Davidsson P. TinyML on mobile devices for hybrid energy management systems. In: 2024 IEEE International Conferences on Internet of Things (iThings) and IEEE Green Computing & Communications (GreenCom) and IEEE Cyber, Physical & Social Computing (CPSCom) and IEEE Smart Data (SmartData) and IEEE Congress on Cybermatics; 2024; Copenhagen, Denmark. p. 200–7.
111. Fernandes R, Costa C, Gomes R, Vilaça N. SmartLVEnergy: an AIoT framework for energy management through distributed processing and sensor-actuator integration in legacy low-voltage systems. IEEE Sens J. 2024;24(13):20726–41. doi:10.1109/JSEN.2024.3403484.
112. Hussain A, Abughanam N, Qadir J, Mohamed A. Jamming detection in IoT wireless networks: an edge-AI based approach. In: Proceedings of the 12th International Conference on the Internet of Things; 2023; Delft, Netherlands. p. 57–64.
113. Giordano M, Baumann N, Crabolu M, Fischer R, Bellusci G, Magno M. Design and performance evaluation of an ultralow-power smart IoT device with embedded TinyML for asset activity monitoring. IEEE Trans Instrum Meas. 2022;71:2510711. doi:10.1109/TIM.2022.3165816.
114. Agrawal A, Maiti RR. TinyAP: an intelligent access point to combat Wi-Fi Attacks using TinyML. IEEE Internet Things J. 2024;12(2):2135–45. doi:10.1109/JIOT.2024.3467328.

115. Saranya T, Jeyamala D, Sellamuthu S. A secure framework for MIoT: TinyML-powered emergency alerts and intrusion detection for secure real-time monitoring. In: 2024 8th International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC); 2024; Kirtipur, Nepal. p. 13–21.
116. Huang J, Yu T, Chakraborty C, Yang F, Lai X, Alharbi A, et al. An energy harvesting algorithm for UAV-assisted TinyML consumer electronic in low-power IoT networks. IEEE Trans Consum Electron. 2024;70(4):7346–56. doi:10.1109/TCE.2024.3419784.
117. Ahmad U, Han M, Jolfaei A, Jabbar S, Ibrar M, Erbad A, et al. A comprehensive survey and tutorial on smart vehicles: emerging technologies, security issues, and solutions using machine learning. IEEE Trans Intell Transp Syst. 2024;25(11):15314–41. doi:10.1109/TITS.2024.3419988.
118. Huda NU, Ahmed I, Adnan M, Ali M, Naeem F. Experts and intelligent systems for smart homes' transformation to sustainable smart cities: a comprehensive review. Expert Syst Appl. 2024;238(9):122380. doi:10.1016/j.eswa.2023.122380.
119. Hernandez C, Taslimi B, Lee HY, Liu H, Pardalos PM. Training generalizable quantized deep neural nets. Expert Syst Appl. 2023;213(2):118736. doi:10.1016/j.eswa.2022.118736.
120. Yang H, Gui S, Zhu Y, Liu J. Automatic neural network compression by sparsity-quantization joint learning: a constrained optimization-based approach. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition; 2020; Seattle, WA, USA. p. 2175–85.
121. Wang K, Liu Z, Lin Y, Lin J, Han SHAQ. Hardware-aware automated quantization with mixed precision. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition; 2019; Long Beach, CA, USA. p. 8612–20.
122. Jayasimhan A, Pabitha P. ResPrune: an energy-efficient restorative filter pruning method using stochastic optimization for accelerating CNN. Pattern Recognit. 2024;155:110671. doi:10.1016/j.patcog.2024.110671.
123. Poyatos J, Molina D, Martinez AD, Del Ser J, Herrera F. EvoPruneDeepTL: an evolutionary pruning model for transfer learning based deep neural networks. Neural Netw. 2023;158(3):59–82. doi:10.1016/j.neunet.2022.10.011.
124. Chen W, Wang P, Cheng J. Towards automatic model compression via a unified two-stage framework. Pattern Recognit. 2023;140(1):109527. doi:10.1016/j.patcog.2023.109527.
125. Guo B, Chang X, Chao F, Zheng X, Lin C-M, Chen Y, et al. ARLP: automatic multi-agent transformer reinforcement learning pruner for one-shot neural network pruning. Knowl Based Syst. 2024;300:112122. doi:10.1016/j.knosys.2024.112122.
126. Albanese A, Nardello M, Fiacco G, Brunelli D. Tiny machine learning for high accuracy product quality inspection. IEEE Sens J. 2022;23(2):1575–83. doi:10.1109/JSEN.2022.3225227.
127. Chen X, Xu G, Xu X, Jiang H, Tian Z, Ma T. Multicenter hierarchical federated learning with fault-tolerance mechanisms for resilient edge computing networks. IEEE Trans Neural Netw Learn Syst. 2024;36(1):47–61. doi:10.1109/TNNLS.2024.3362974.
128. Brandic I. Sustainable and trustworthy edge machine learning. IEEE Internet Comput. 2021;25(5):5–9. doi:10.1109/MIC.2021.3104383.
129. Tiwari RG, Haroon M, Tripathi MM, Kumar P, Agarwal AK, Jain V. A system model of fault tolerance technique in distributed system and scalable system using machine learning. In: Software-defined network frameworks. New York, NY, USA: CRC Press; 2024. p. 1–16.
130. Zhang J, Chen B, Cheng X, Binh HTT, PoisonGAN YS. Generative poisoning attacks against federated learning in edge computing systems. IEEE Internet Things J. 2020;8(5):3310–22. doi:10.1109/JIOT.2020.3023126.
131. Bolchini C, Cassano L, Miele A. Resilience of deep learning applications: a systematic literature review of analysis and hardening techniques. Comput Sci Rev. 2024;54(521):100682. doi:10.1016/j.cosrev.2024.100682.
132. Narayanan N, Chen Z, Fang B, Li G, Pattabiraman K, Debardeleben N. Fault injection for TensorFlow applications. IEEE Trans Dependable Secure Comput. 2022;20(4):2677–95. doi:10.1109/TDSC.2022.3175930.
133. Laskar S, Rahman MH, Zhang B, Li G. Characterizing deep learning neural network failures between algorithmic inaccuracy and transient hardware faults. In: 2022 IEEE 27th Pacific Rim International Symposium on Dependable Computing (PRDC); 2022; Beijing, China. p. 54–67.

134. Syed RT, Ulbricht M, Piotrowski K, Krstic M. Fault resilience analysis of quantized deep neural networks. In: 2021 IEEE 32nd International Conference on Microelectronics (MIEL); 2021; Nis, Serbia. p. 275–9.
135. Ruospo A, Sanchez E, Traiola M, O'connor I, Bosio A. Investigating data representation for efficient and reliable convolutional neural networks. Microprocess Microsyst. 2021;86(7):104318. doi:10.1016/j.micpro.2021.104318.
136. Liu Z, Yang X. An efficient structure to improve the reliability of deep neural networks on ARMs. Microelectron Reliab. 2022;136(2):114729. doi:10.1016/j.microrel.2022.114729.
137. Gao Z, Yao Y, Wei X, Yan T, Zeng S, Ge G, et al. Reliability evaluation of FPGA based pruned neural networks. Microelectron Reliab. 2022;130(8):114498. doi:10.1016/j.microrel.2022.114498.
138. Ficco M, Guerriero A, Milite E, Palmieri F, Pietrantuono R, Russo S. Federated learning for IoT devices: enhancing TinyML with on-board training. Inf Fusion. 2024;104(3):102189. doi:10.1016/j.inffus.2023.102189.
139. Gupta BB, Quamara M. An overview of Internet of Things (IoT): architectural aspects, challenges, and protocols. Concurr Comput. 2020;32(21):e4946. doi:10.1002/cpe.4946.
140. Nguyen MT, Truong HL. On optimizing resources for real-time end-to-end machine learning in heterogeneous edges. Softw Pract Exp. 2025;55(3):541–58. doi:10.1002/spe.3383.
141. Liu Q, Mo R, Xu X, Ma X. Multi-objective resource allocation in mobile edge computing using PAES for Internet of Things. Wirel Netw. 2024;30(5):3533–45. doi:10.1007/s11276-020-02409-w.
142. Asghari A, Azgomi H, Darvishmofarahi Z. Multi-objective edge server placement using the whale optimization algorithm and game theory. Soft Comput. 2023;27(21):16143–57. doi:10.1007/s00500-023-07995-3.
143. Moustakas T, Tziouvaras A, Kolomvatsos K. Data and resource aware incremental ML training in support of pervasive applications. Computing. 2024;106(11):3727–53. doi:10.1007/s00607-024-01338-2.
144. Yoosefi A, Kargahi M. Resource-aware in-edge distributed real-time deep learning. Internet of Things. 2024;27(8):101263. doi:10.1016/j.iot.2024.101263.
145. Yu S, Muñoz JP, Jannesari A. Resource-aware heterogeneous federated learning with specialized local models. In: European Conference on Parallel Processing; 2024; Madrid, Spain. p. 389–403.
146. Ge Y, Zhou Y, Jia L. Adaptive personalized federated learning with one-shot screening. IEEE Internet Things J. 2024;11(9):15375–85. doi:10.1109/JIOT.2023.3346900.
147. Nimmagadda Y. Model optimization techniques for edge devices. In: Model optimization methods for efficient and edge AI: federated learning architectures, frameworks and applications. Piscataway, NJ, USA: Wiley-IEEE Press; 2025. p. 57–85.
148. Rajapakse V, Karunanayake I, Ahmed N. Intelligence at the extreme edge: a survey on reformable tinyml. ACM Comput Surv. 2023;55(13s):1–30. doi:10.1145/3583683.
149. Liu F, Li H, Hu W, He Y. Review of neural network model acceleration techniques based on FPGA platforms. Neurocomputing. 2024;610(7):128511. doi:10.1016/j.neucom.2024.128511.
150. Subramaniam EVD, Srinivasan K, Qaisar SM, Pławiak P. Interoperable IoMT approach for remote diagnosis with privacy-preservation perspective in edge systems. Sensors. 2023;23(17):7474. doi:10.3390/s23177474.
151. Pliatsios A, Kotis K, Goumopoulos C. A systematic review on semantic interoperability in the IoE-enabled smart cities. Internet Things. 2023;22(1):100754. doi:10.1016/j.iot.2023.100754.
152. Nilsson J, Javed S, Albertsson K, Delsing J, Liwicki M, Sandin F. AI concepts for system of systems dynamic interoperability. Sensors. 2024;24(9):2921. doi:10.3390/s24092921.
153. Chang C-Y, Chuang Y-C, Huang C-T, Wu A-Y. Recent progress and development of hyperdimensional computing (HDC) for edge intelligence. IEEE J Emerg Sel Top Circuits Syst. 2023;13(1):119–36. doi:10.1109/JETCAS.2023.3242767.
154. Hassan E, Bettayeb M, Mohammad B. Advancing hardware implementation of hyperdimensional computing for edge intelligence. In: 2024 IEEE 6th International Conference on AI Circuits and Systems (AICAS); 2024; Abu Dhabi, United Arab Emirates. p. 169–73.
155. Wang Y, Shang F, Lei J. Multi-granularity fusion resource allocation algorithm based on dual-attention deep reinforcement learning and lifelong learning architecture in heterogeneous IIoT. Inf Fusion. 2023;99:101871. doi:10.1016/j.inffus.2023.101871.

156. Mir NF. AI-assisted edge computing for multi-tenant management of edge devices in 6G networks. In: 2023 2nd International Conference on 6G Networking (6GNet); 2023; Paris, France. p. 1–3.
157. Jedidi A. Dynamic trust security approach for edge computing-based mobile IoT devices using artificial intelligence. Eng Res Express. 2024;6(2):25211. doi:10.1088/2631-8695/ad43b5.
158. Khan MA, Puri D. Challenges and opportunities in implementing quantum-safe key distribution in IoT devices. In: 2024 3rd International Conference for Innovation in Technology (INOCON); 2024; Bangalore, India. p. 1–7.
159. Dharani D, Anitha Kumari K. A smart surveillance system utilizing modified federated machine learning: gossip-verifiable and quantum-safe approach. Concurr Comput. 2024;36(24):e8238. doi:10.1002/cpe.8238.
160. Ansere JA, Gyamfi E, Sharma V, Shin H, Dobre OA, Duong TQ. Quantum deep reinforcement learning for dynamic resource allocation in mobile edge computing-based IoT systems. IEEE Trans Wirel Commun. 2023;23(6):6221–33. doi:10.1109/TWC.2023.3330868.
161. Karakaya A, Ulu A. A survey on post-quantum based approaches for edge computing security. Wiley Interdiscip Rev Comput Stat. 2024;16(1):e1644. doi:10.1002/wics.1644.
162. Khonina SN, Kazanskiy NL, Skidanov RV, Butt MA. Exploring types of photonic neural networks for imaging and computing—a review. Nanomaterials. 2024;14(8):697. doi:10.3390/nano14080697.
163. Bandyopadhyay S, Sludds A, Krastanov S, Hamerly R, Harris N, Bunandar D, et al. Single-chip photonic deep neural network with forward-only training. Nat Photonics. 2024;18(12):1335–43. doi:10.1038/s41566-024-01567-z.
164. Rani F, Chollet N, Vogt L, Urbas L. Industrial edge MLOps: overview and challenges. Comput Aided Chem Eng. 2024;53(11):3019–24. doi:10.1016/B978-0-443-28824-1.50504-4.
165. Shabir MY, Torta G, Basso A, Damiani F. Toward secure TinyML on a standardized AI architecture. In: Device-edge-cloud continuum: paradigms, architectures and applications. Cham, Switzerland: Springer; 2023. p. 121–39.