

A Comprehensive Survey on Model Compression Techniques for Edge Intelligence

Eman Hamdy
*College of Computing and
Information Technology
Arab Academy for Science,
Technology, and Maritime
Transport
Alex, Egypt*

Mohamed Kholif
*College of Computing and
Information Technology
Arab Academy for Science,
Technology, and Maritime
Transport
Alex, Egypt*

Mohamed Abdelrahman
*College of Computer Science
and Artificial Intelligence
Pharos University
Alex, Egypt*

Abstract— This paper reviews how small neural networks move AI from cloud servers to the device. Deep learning now appears in cameras, wearables, and sensors, but those targets supply only a few kilobytes of RAM, a coin cell's worth of energy, and a millisecond deadline. A ninety megabyte model that scores well on a desktop fails when the target is an ARM M-class microcontroller. Investigators, therefore, compress models while keeping high accuracy. Three compression methods receive most attention - pruning deletes weights that add little to the output. Quantization shortens each weight from thirty-two bits to eight or even one. Knowledge distillation trains a miniature student network to copy the soft answers of a large teacher network. Those steps are applied alone or chained into a single pipeline. Popular tiny architectures are also surveyed - MobileNet, ShuffleNet, and EfficientNet-Lite. Some reviewed studies report that hybrid compression schemes, such as pruning followed by quantization or knowledge distillation followed by pruning, can reduce model size and, in certain cases, improve inference latency compared with individual techniques. However, these benefits are not universal and depend on the compression strategy, model architecture, target hardware, and inference framework. **Furthermore, the survey compares existing studies in terms of application domains, datasets, baseline models, compression strategies, model size, computational complexity, accuracy, hardware platforms, inference latency, memory utilization, and energy consumption, where reported. Particular attention is given to the distinction between theoretical compression gains and actual hardware-level performance, considering the influence of accelerator support and inference frameworks. The comparative analysis highlights the trade-offs among accuracy, computational efficiency, memory requirements, energy consumption, and deployment complexity. The reviewed literature also reveals challenges in comparing experimental results across heterogeneous datasets, evaluation metrics, and hardware platforms. Finally, this survey identifies research gaps in standardized benchmarking, hardware-aware optimization, comprehensive deployment evaluation, and the selection of suitable compression strategies for specific edge applications.**

Keywords— Lightweight networks · Model Optimization · Computer vision · Edge devices · Computing power. Edge Intelligence, Model Compression, Quantization, Pruning, Knowledge Distillation,

1. INTRODUCTION

Artificial intelligence is integrated with edge computing, which leads to direct work on the edge of the network and causes an evolution in data processing and analysis. It also helps solve problems related to traditional artificial intelligence systems that rely on cloud computing, response time, energy consumption, and data privacy [22,6]. Edge AI is used on smartphones in many applications, including Internet of Things(IOT) devices, allowing data to be located close to the source rather than relying on centralized cloud infrastructure [23].

Cloud computing uses powerful computing resources by sending data from the client to central data centers for processing, causing significant delays due to data transfer and network congestion, especially in applications that require immediate responses, such as document processing and autonomous systems [24,25]. As a result, sensitive applications suffer from delays, reduced performance, and reliability. Therefore, edge computing processes data at or

near the point of generation, which speeds up communication, reduces bandwidth consumption, and improves system reliability by reducing dependency on cloud computing [7, 26], and enhances privacy and security by decreasing exposure to third-party network providers [4, 28].

The development of object detection systems and computer vision based on deep learning requires large amounts of computing resources, energy, and memory. This reliance mainly on high-performance central processing units (CPUs) or graphics processing units (GPUs) has led to significant advances in Internet of Things technologies and applications, achieving high accuracy and rapid reasoning capabilities. However, despite the accuracy achieved by these models, they are not suitable for application on edge devices [1, 29, 30]. Hence, the discrepancy between model complexity and terminal device capabilities has become a decisive factor in terminal artificial intelligence research.

In this context, the development of lightweight AI models has become a key enabler for effective Edge AI deployment. These models are very important for various applications that require fast processing, such as autonomous driving and healthcare, as they are designed to use fewer computing resources, memory, and energy consumption while maintaining accuracy and reliability. By reducing resource requirements and complexity. Lightweight AI solutions provide faster and more reliable performance, reduce network congestion, and lower operating costs, making them suitable for all applications.

The research examines the ways to build small artificial intelligence models that run on edge devices. It reviews the compression tricks, the network shapes, the design steps that fit the device limits, and the hardware aware plans that let the chip help the software. This paper aims to provide a complete and clear explanation of how to place an accurate model on a small machine without loss of speed.

Real-time choices are now required everywhere - IoT networks and mobile hardware expand fast - tiny Edge AI models become essential. Those models let engineers build smart services that scale, save energy, and keep the user data on the device, even when the hardware holds little memory.

1.1 The main contributions of this survey are summarized as follows:

Unified taxonomy of model compression techniques:

A structured classification of pruning, quantization, knowledge distillation, and hybrid compression methods, highlighting their principal mechanisms, variants, and training requirements.

Comparative analysis of compression methods and lightweight architectures: An examination of the advantages, limitations, and trade-offs of different optimization strategies in terms of model size, computational complexity, predictive performance, memory requirements, and deployment considerations.

Hardware-aware analysis of edge deployment: An investigation of the relationship between theoretical compression gains and practical inference performance, considering hardware capabilities, accelerator support, inference frameworks, latency, memory utilization, and energy consumption.

Critical synthesis of existing studies: A comparative examination of published research according to application domains, datasets, baseline models, compression methods, evaluation metrics, and deployment platforms, while acknowledging limitations in cross-study comparability.

Identification of research gaps and future directions: An analysis of challenges related to hardware-aware compression selection, standardized benchmarking, practical deployment evaluation, and the effective integration of multiple compression techniques.

1.2 Scope and Taxonomy of the Survey

This survey focuses primarily on model compression as the central mechanism for enabling efficient deep learning deployment on resource-constrained edge devices. To position model compression within the broader Edge AI ecosystem, the surveyed approaches are organized into four hierarchical levels, illustrated in Figure 1.

At the model level, the core focus is model compression, including pruning, quantization, knowledge distillation, and hybrid compression. These techniques directly modify the model representation, parameterization, or training process to reduce resource requirements while maintaining predictive performance.

At the architecture level, lightweight model design provides an alternative or complementary approach by constructing efficient neural network architectures from the beginning. Architectures such as MobileNet, ShuffleNet, SqueezeNet, and EfficientNet-Lite are therefore considered complementary to post-training or training-based compression techniques rather than being treated as compression methods themselves.

At the hardware-software level, hardware-aware optimization considers the characteristics of the target processor, accelerator, memory hierarchy, supported numerical formats, inference framework, and execution kernels. This level is important because reductions in model parameters or theoretical computational complexity do not necessarily result in equivalent improvements in measured hardware performance.

At the system level, distributed learning, federated learning, and edge-cloud collaboration determine where training, inference, communication, and data processing are performed. These approaches provide deployment and system-level mechanisms that complement model compression but do not themselves constitute model compression techniques.

Accordingly, the relationship among these levels can be summarized as follows: model compression and lightweight architecture design reduce model-side resource requirements; hardware-aware optimization determines how efficiently the resulting model can be executed on a target platform; and distributed, federated, and edge-cloud architectures determine how computation, training, data, and communication are distributed across the overall system.

This hierarchical organization allows the survey to distinguish the central research topic—model compression—from complementary model-design, hardware, and system-level approaches. It also provides a common framework for analyzing the reviewed studies according to model efficiency, deployment efficiency, and system-level requirements.

As for how the rest of this paper is laid out, Section 2 covers the background behind Edge AI, including what motivates its development and the challenges it still faces. In Section 3, we look at related work, particularly around how AI has evolved at the edge, and we also go over applications involving lightweight object detection models. Section 4 gets into the materials and methods behind deploying these models on edge devices, touching on architectural choices for both training and inference, along with key performance benchmarks. Section 5 focuses on the metrics used to evaluate Edge AI systems. Finally, we wrap things up with conclusions and a discussion of where future research could — and probably should — be headed.

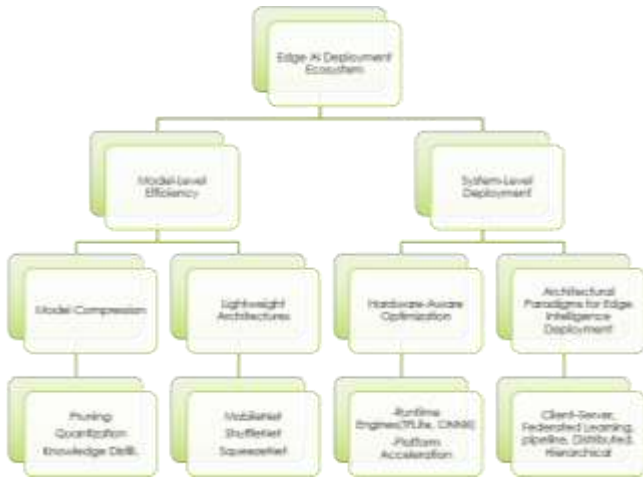


Figure 1. Hierarchical taxonomy of the survey.

2. BACKGROUND

Models grow larger and more intricate each year - the pressure to trim them rises. Model compression offers a concrete fix; it narrows the gap between the increasing computing demands of modern AI and the slower progress of memory and hardware. The technique reduces a model's size while keeping almost the same accuracy. Because the footprint shrinks, the same network now fits inside a smartphone or an embedded board that owns only a modest budget for RAM besides CPU cycles. **Model compression can reduce the storage requirements of neural networks and may decrease their computational workload. However, a reduction in the number of parameters does not necessarily result in lower inference latency or energy consumption. The actual performance gains depend on the compression method, memory access patterns, hardware capabilities, and the optimization support provided by the inference framework.** In a mobile app, a compressed network returns answers faster and lets the battery last longer - both effects on the user's satisfaction [63].

Billions of new IoT nodes create demand for local intelligence that reacts instantly, without visible lag [33]. A self-driving car, an assembly robot, as well as a wrist-worn health sensor must decide within milliseconds, and they cannot pause until a distant server replies. Sending raw sensor data streams to the cloud also overloads links and exposes private data to extra risk [34].

Instead of sending everything to a faraway server, analysis happens near the source of data creation. By shifting computation locally, reliance on central hubs drops sharply [8,35]. Responses speed up while privacy improves naturally. Performance stays stable even as demand grows, since devices avoid round-trip delays to the cloud [35].

2.1 AI on Edge

A typical artificial intelligence setup forwards each unprocessed data feed straight to a faraway cloud facility. Stored there, information undergoes examination before answers travel back. Instead of relying on far-off centers, Edge AI runs those computations directly within devices like smartphones, vehicles, sensors, or street-level units, capturing inputs initially [36]. Speed matters; this shift happens mainly because waiting costs time. When users ask

something, they expect answers immediately with no delays. Sensitive information should remain close, not sent far off. Even when the connection to remote servers drops, things must keep working. Since the model operates locally, responses happen instantly.

2.2 Historical Evolution: From Cloud to the Intelligent Edge

The planet now hosts billions of networked sensors, and each new program demands more bytes processed at the network's edges. Cloud data centers cannot hide their limits any longer. Latency stays high, traffic bills climb each quarter, and regulators question the practice of sending sensitive records to remote data centers. Driverless vehicles, live video analytics, and patient monitors amplify every drawback.

The move toward on device computation did not come from one sudden breakthrough, it advanced through many small steps that removed separate bottlenecks. Figure 2 shows the order of those steps. Each advance prepared the base for the hardware and software stack that now carries the name edge AI [57].

The primary advantage of distributing data in this way is that it shortens the path the data must travel as well as this single change cuts response time [58].

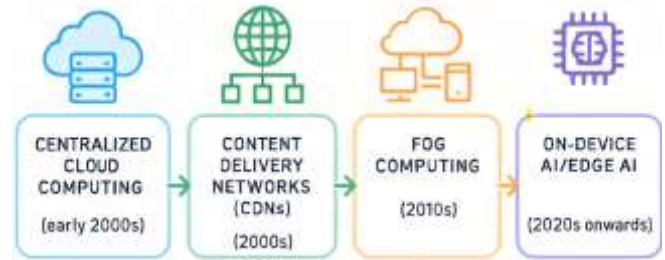


Figure 2. Progression established the foundational layers that constitute the modern Edge AI landscape[57].

2.3 Edge AI and System-Level Deployment

Edge AI can be implemented within different system-level deployment configurations depending on where computation, training, communication, and model coordination are performed. Distributed artificial intelligence (DAI) distributes computational tasks across multiple nodes, while federated learning enables multiple devices to participate in collaborative model training without requiring centralized collection of the original training data [61].

These approaches complement model compression rather than replacing it. A compressed or lightweight model can be deployed within a distributed or federated architecture, while edge-cloud collaboration can determine whether inference or other computational tasks remain on the edge or are transferred to a more capable computing layer. Thus, system-level deployment approaches define the distribution of computation and data, whereas model compression primarily addresses the efficiency of the neural network itself.

2.4 Challenges for AI on the Edge

Deep neural network models or Transformer models cannot be run directly on these devices without compressing the model or using a lightweight model, as edge devices operate within limited constraints of memory, storage, and processor speed, whereas cloud computing servers contain a powerful graphics processing unit (GPU) or tensor processing unit (TPU) [33].

Systems rely on battery life, so researchers are developing hardware-aware optimization methods for edge devices and low-power TinyML frameworks, as there are many applications that depend on battery life and consume limited power, such as wearable health monitors, autonomous drones, and field sensors, which often rely on small batteries or harvested energy. Practical solutions that balance energy and accuracy remain important [3].

A trade-off exists between response time as well as accuracy - the primary goal of edge AI is fast reaction but when engineers shrink or simplify a model to accelerate it, the model loses precision. It is hard to decide how much speed to retain and how much accuracy to give up or the choice becomes critical in domains where a wrong answer endangers life, like autonomous driving or medical screening [36].

Data privacy and security at large scale, If data remain on the device and not move to the cloud, privacy improves. When thousands or millions of identical edge units are deployed across an area, each unit must block intrusion, malware or theft of its trained parameters. Federated learning and signed update protocols lower risk but they create new problems like extra network traffic, uncertain links [8, 69].

Figure 3 summarizes these challenges along with potential applications where edge computing technologies make, or could make, a tangible difference.

2.5 Applications of Edge AI

Autonomous Vehicles - Edge AI lets an unmanned vehicle decide instantly. On-board hardware reads camera, LiDAR and radar streams then selects an action within milliseconds without waiting for the cloud [60].

Smart Cities and IoT: Edge AI is integral to smart cities, where traffic management, energy consumption, and infrastructure maintenance are monitored and optimized through locally deployed AI models[60].

Healthcare: Wearables and medical devices that monitor patient vitals can use Edge AI for on-the-spot analysis of health data, such as heart rate, glucose levels, or Electrocardiogram (ECG) readings, reducing the need for real-time cloud communication[60].

Manufacturing and Industrial Automation: In Industry 4.0, Edge AI enables predictive maintenance, quality control, and real-time monitoring of industrial processes with local computation to ensure that critical operations continue uninterrupted[60].

Retail and Customer Experience: AI-powered cameras and sensors deployed in retail environments can analyze customer behavior and optimize the store layout or inventory management based on local analysis [2,31].



Figure 3. Challenges and potential applications of edge intelligence [8].

3. Literature Review

Recent research on Edge AI has increasingly focused on reducing the computational and memory requirements of deep learning models while maintaining acceptable predictive performance on resource-constrained devices. The reviewed studies employ different strategies, including model compression, lightweight architecture design, and hardware- or system-level optimization. However, their reported results vary substantially because the studies address different tasks, datasets, model architectures, and deployment platforms.

Therefore, this literature review does not consider the reported performance values as directly comparable across all studies. Instead, the reviewed works are analyzed according to common dimensions, including the compression or optimization strategy, model size and parameter reduction, computational complexity, predictive performance, inference latency, memory consumption, energy requirements, training or retraining requirements, and hardware compatibility. This approach allows the discussion to move beyond describing individual studies and to identify the conditions under which different optimization strategies are more appropriate for Edge AI deployment.

The reviewed studies are first discussed according to the main model-level optimization strategies: knowledge distillation, pruning, quantization, and hybrid compression. To address the reviewers' request for a more systematic and critical literature review, the studies are additionally compared using a common set of dimensions: task and dataset, baseline architecture, compression or optimization mechanism, parameter/model-size reduction, computational complexity, predictive performance, latency, memory, energy, training requirements, and hardware/deployment compatibility. The numerical results reported by different studies are not treated as directly interchangeable because the experimental conditions differ across datasets, architectures, input sizes, frameworks, hardware platforms, and measurement protocols.

Table 1 presents an overview of Studies on Lightweight Model Optimization for Edge AI.

3.1 Model Compression Techniques:

3.1.1 Quantization:

IN[17] the authors present how to deploy deep-learning-based object recognition on edge computing devices, covering the key introduction of how latency, bandwidth, and privacy concerns push AI to the network edge rather than cloud-only processing. They survey relevant datasets

implicitly via applications (for instance, a case study of real-time power-transmission-line inspection) and discuss the use of lightweight models—specifically compressed versions like a quantized Tiny YOLOv7 (with INT8 precision) for edge deployment. They discuss how the model compression/quantization and hardware-aware optimization (e.g., using frameworks such as TensorFlow Lite, ONNX, TensorRT) is integrated into the model pipeline, and they evaluate performance on hardware platforms such as the Raspberry Pi 4B, NVIDIA Jetson Nano, and Jetson Orin Nano edge boards. Their results show that the quantized Tiny YOLOv7 on the Jetson Orin Nano achieved the best edge-AI deployment score (EADS) when balancing accuracy, inference time, RAM utilization and power consumption, while full YOLOv7 (FP32) performed very poorly in inference time on weaker hardware.

This study illustrates that the practical benefit of quantization is strongly influenced by the target platform. The use of INT8 precision can reduce the computational and memory requirements of a model, but the resulting improvement in latency or energy consumption depends on whether the hardware and inference framework efficiently support the selected numerical format. Therefore, a reduction in precision should not by itself be interpreted as evidence of a proportional reduction in real-world inference time.

The reviewed evidence also highlights a trade-off between deployment simplicity and accuracy preservation. The extent to which quantization affects predictive performance depends on the model, selected precision, and quantization configuration. Consequently, quantization should be selected according to the required accuracy, available training resources, and capabilities of the target deployment platform rather than assuming that a lower precision is always preferable.

Compared with pruning, quantization can reduce the numerical representation of an existing model without necessarily changing its network structure. This makes it attractive when memory footprint and low-precision computation are important constraints. However, the two approaches address different aspects of model efficiency and can also be combined when supported by the deployment framework.

Overall, the reviewed study supports the use of quantization as an effective approach for edge deployment when the selected precision is well matched to the target hardware. Nevertheless, model-level indicators such as reduced precision or model size should be complemented by measured latency, memory, and energy results to determine the actual deployment benefit.

3.1.2 Pruning and Lightweight Architectures:

IN[16], the study used the BRATS(Brain Tumor Segmentation) MRI dataset for training and validation. The authors built their model on the RetinaNet architecture, replacing its heavy ResNet backbone with MobileNet to reduce computation, and further applied structured pruning to remove redundant filters and layers, achieving a compact and efficient network without significant accuracy loss. Achieving efficiency on compact devices like the NVIDIA Jetson Nano or Raspberry Pi 4 became possible through MobileNet paired with pruning. Shown in Figure 4 is the flow of the suggested approach. Real-world tests revealed an

overall average precision of 32.1%, reaching 0.143 for smaller lesions yet rising to 0.497 when detecting larger ones. Better than RetinaNet and similar models, it used notably lower memory and energy. Accuracy meets affordability here - proof that slimmed networks and flexible architectures enable smart diagnosis tools on portable medical devices.

The results indicate that combining a lightweight backbone with pruning can provide a practical reduction in resource requirements, particularly for applications targeting constrained edge platforms. However, the reported improvements are specific to the selected architecture, medical-imaging task, and deployment environment. Therefore, they should not be interpreted as evidence that pruning alone will consistently produce the same accuracy, latency, or energy benefits across different models and hardware platforms.

The combination of lightweight architecture design and pruning may therefore be beneficial when both model capacity and deployment resources are constrained. Nevertheless, the appropriate strategy depends on the target application, model characteristics, required accuracy, and hardware capabilities.

Overall, the reviewed evidence suggests that pruning and lightweight architectures should be evaluated not only according to model-level reductions but also according to measured deployment performance. Reported parameter, FLOPs, or sparsity reductions should therefore be considered indicators of potential efficiency, while latency, memory, and energy measurements provide stronger evidence of practical edge-device benefits.

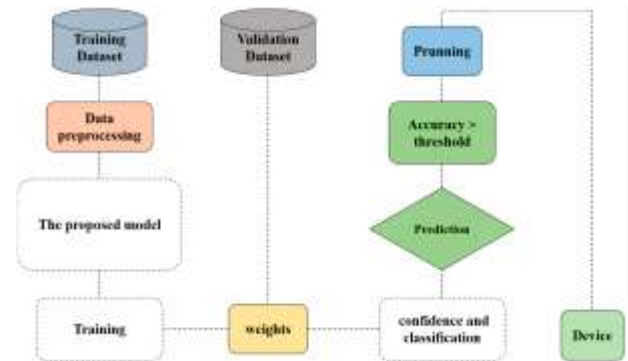


Figure 4. Workflow of model [16].

3.1.3 Knowledge Distillation:

IN[14] introduce several deep learning setups that were put to work spotting fires automatically, with attention given to flexible designs that run well on edge devices. three main models: YOLOv8, Detection Transformer (DETR), and Detectron2, were trained separately, then compared. Although they all performed well, the YOLOv8 stood out thanks to its faster processing and better flexibility. As a result, attention turned toward improving it further using techniques that pass learning between models. Instead of starting from scratch, the plan involved using YOLOv8m - the medium-sized model - to shape how the smaller YOLOv8n learns. In effect, smarter decisions made by the larger network were reflected within the smaller network. For this exchange, a technique called Autodistill was applied, funneling key patterns from the more capable YOLOv8m

In addition, the reported accuracy and inference performance in [14] should be interpreted within the specific experimental setting. The results were obtained for a particular fire-detection dataset, model architecture, Raspberry Pi platform, and deployment configuration. Therefore, they demonstrate the feasibility of KD for this application rather than establishing that KD will consistently provide the same accuracy or latency advantages across different Edge AI tasks and hardware platforms.

Compared with pruning and quantization, KD primarily transfers knowledge rather than directly removing parameters or reducing numerical precision. Its main benefit is therefore the improvement of the accuracy–compactness trade-off of a predefined student model. Nevertheless, KD introduces additional training complexity because a teacher model and a distillation procedure are required. Its deployment advantage is realized only through the resulting student model, while the teacher does not need to be retained for inference. KD can therefore be particularly attractive when a suitable lightweight architecture already exists but its predictive performance is insufficient for the target application.

The diagram illustrates the proposed framework. It begins with an input image, which is processed by a 'Base model' (YOLOv8m) to produce a segmented image with labels. This segmented image is then processed by two 'Target models' (YOLOv8n and DETR) to produce a final segmented image with labels. The final output is then processed by 'Edge Devices' to produce a final segmented image with labels.

3.2.1 Simulation-Based Architectures:

[illegible]

3.2.2 Federated Learning and Resource Allocation:

IN[18] This paper presents a lightweight neural network model called Linge for disease risk prediction, designed to be deployed on edge computing devices using federated learning. The main goal is to ease the load on central servers and to keep storage and bandwidth demands from growing, all while the model keeps its high prediction accuracy. Training plus tests rely on the "Acute Ischemic Stroke MRI" dataset from Kaggle - it holds MRI scans of patients who

experienced acute ischemic infarction. The chosen model, named Linge, starts from EfficientNetV2-S and then receives multiple key changes. It carries a lightweight attention module and switches between two convolutional block types according to depth - EFused-MBConv blocks operate in deep layers, whereas EMBCConv blocks operate in shallow layers. This split lowers computation cost but still lets the network extract critical features. Measurements show that a single transaction needs only 7.63 MB but also that the full model occupies about 155.28 MB of memory, a size small enough for resource limited edge devices. All experiments ran on a distributed edge computing setup. The central server held an Intel i7-13700F processor together with an RTX 4060Ti graphics card; each edge server held an Intel i5-13400F processor together with an RTX 3060Ti graphics card. No exotic parts were chosen - the hardware matches gear that could appear in an actual deployment. The resulting accuracy surpassed the initial target. The Linge network surpasses other compact competitors, including the standard EfficientNetV2. It attains 96% accuracy, 95% precision, 93% recall, 95% specificity, 98% F1 score and 97% area under the curve (AUC). Beyond those values, both inference time and memory demand drop noticeably. Such reductions are essential when the aim is to assess disease risk instantly on edge hardware.

3.2.3 Compact CNN Design for Efficient Classification

IN[13] addresses a question that gains urgency each day - how do we tell a photograph that came from a camera shutter from one that a computer produced? The authors rely on a small convolutional neural network whose design is almost austere. Eight convolutional layers feed into two fully connected layers and the training routine applies nothing more exotic than batch normalization, ReLU activations, max pooling windows and dropout. The network is ordinary by design. The team evaluated it on three separate image collections - the first, called CIFAKE, holds 120 000 pictures - half are real photographs borrowed from CIFAR-10, the other half are fakes produced by Stable Diffusion 1.4. The second set contains 50000 pictures split between genuine photos plus outputs from Midjourney version 6. The third set is a custom bundle of 100 000 satellite frames, half recorded - orbiting sensors and half fabricated by StyleGANv3. On every set, the classifier scored high - it labeled 96.6 percent of CIFAKE pictures correctly, reached 99.9 percent accuracy on the Midjoyage collection and finished at 98.7 percent on the satellite imagery. The model reaches the same level as the latest and most intricate techniques and it even exceeds them in some tests. This result is notable because the architecture remains simple. A section that held attention covers interpretability through Grad-CAM++. On authentic photographs, the highest neural responses gather around the main objects. On synthetic photographs, the same responses move to borders and to zones that carry intricate texture. The difference is small but it shows which cues the network values during classification. To summarise, the authors deliver a lightweight convolutional neural network that runs quickly, returns correct predictions and allows inspection. Inspection matters when the task is to expose deepfakes or to protect the trustworthiness of visual records. This study contributes evidence for lightweight architecture design rather than a directly isolated quantization, pruning, or

KD intervention. It should therefore be used as a complementary architectural baseline rather than mixed numerically with compression-only studies.

3.3 Real-World Application

The reviewed studies demonstrate that model optimization is applied across a wide range of real-world Edge AI scenarios, including surveillance, healthcare, infrastructure inspection, and agricultural applications. These applications impose different requirements on accuracy, latency, memory, energy consumption, and reliability. Therefore, the suitability of a compression or lightweight-model strategy depends not only on the achieved model reduction but also on the requirements of the target application.

IN[19] This paper focus shifts toward spotting irregular heartbeats using regular 12-lead ECGs, relying on tiny neural models capable of running directly on microcontrollers. Two streamlined architectures emerge from the study - one called CLTC, short for ConvLSTM2D-liquid time constant. Although smaller in size, each design maintains functional depth through structural efficiency. The alternative model goes by CCfC, which stands for ConvLSTM2D-closed-form continuous time. While built differently, both aim at accurate rhythm analysis under tight hardware limits. Their construction reflects trade-offs between speed, memory, and predictive precision. One follows dynamic state propagation via learned time constants. One approach relies on numerical shortcuts to ease time-based computations. Though operating differently, both aim at running efficiently on energy-limited hardware. Training drew from the Minas Gerais Telemedicine Network (TNMG), which holds more than two million brief ECG segments labeled by experts across six rhythm types. To check if patterns were truly learned rather than copied, evaluation shifted afterward to the 2018 Chinese Physiological Signals Challenge (CPSC) archive - offering a test beyond the original examples. At first, spatial and temporal patterns in cardiac signals get processed through a ConvLSTM2D layer in each setup. Then, data flows into a Neural Circuit Policy block, which uses either liquid time constant neurons or their continuous-time closed-form versions. Right from the start, design choices were driven by tight microcontroller constraints - low computational demand was baked in, not tacked on later. Performance turned out nearly the same across both versions: F1 scores stayed around 0.83, with AUROC approaching 0.96. Despite introduced distortions or absent ECG channels, stability remained strong, essential for unpredictable medical environments. Though CCfC learned quicker and edged ahead in precision, CLTC handled partial signal loss with greater ease during head-to-head tests. Running on an STM32F746G chip, memory usage stayed at 70.6% of RAM, flash storage dipped to 9.4%, drawing around 137 milliwatts. The results demonstrate the feasibility of designing models specifically for highly constrained medical edge devices. However, the study also identifies the need for broader validation across additional datasets and edge platforms before the results can be generalized. A different healthcare application is investigated in [20] paper presents Listening to breath can reveal signs of lung illness - machines now help spot these clues using artificial intelligence. What makes this approach stand out is how

tricky it remains for expert physicians to correctly read such sounds, despite their clinical value. Rather than pull recordings from just one place, the team pulled together two open-access sources: ICBHI 2017 and a collection by Frywan et al., found on Mendeley Data [66]. Merging those sets led to a more stable mix, ending up with 1,219 cases spread among ten types of conditions - asthma, COPD, pneumonia included. Size-wise, the pool isn't massive; yet fairness across labels stops the system from favoring frequent diagnoses. Normalization shaped the raw sound first, after which subtle layers were amplified - key where limited examples exist. Feature gathering followed. Three views emerged: Mel Spectrograms, MFCCs, Chromagrams - not chosen separately but fused into volumetric form. All three enter together; their blend reveals richer detail than any solo version could show, amounting to 510,825 values, near 6.5MB per case. Training unfolded via Adam and sparse categorical cross-entropy - standard tools, yet reliable under these conditions. To see inside decisions, Grad-CAM joined with saliency visuals appeared, exposing attention zones during judgment - a necessity when health outcomes hang in balance. Together, those features lifted performance to 91.04%, beating bulkier standards like MobileNetV2, ResNet50, DenseNet despite a simpler structure. This outcome stands out, given how compact the model is. Despite having minimal parameters, it achieves strong precision while remaining interpretable - making it suitable for real-world deployment where computing power is limited. Its efficiency fits well with the needs in edge environments, especially when analyzing lung sounds under tight constraints **This study is especially useful for the memory- and energy-oriented dimensions of the comparison. It demonstrates why model accuracy alone is insufficient for edge deployment evaluation: RAM, flash, and power measurements provide deployment evidence that parameter count or FLOPs alone cannot provide. At the same time, the different tasks and evaluation protocols prevent direct comparison of the reported accuracy values. Thus, the main observation is not that one architecture is superior, but that the required balance between predictive performance and resource consumption changes with the medical application and target hardware.**

In remote-sensing applications [21], researchers address the problem of detecting tiny details in satellite images, as traditional tools tend to overlook small details in the image. However, the need to identify image details can be challenging due to image blurriness, noise in the frame, or parts of the image being obscured by obstacles. Therefore, researchers added an SSFF block based on the YOLOv8n model, which helps detect objects. This unit facilitates better data integration across different zoom levels. To operate faster without requiring excessive power, the hierarchical path aggregation network (HPANet) works because it operates without needing additional power, uses fewer parameters, and maintains low computational requirements. Instead of losing fine-grained features during analysis, it preserves them through intelligent layer fusion—without the need for complex computations. When evaluated using VisDrone and AI-TOD, the system performed better than before. Accuracy at 0.5 increased by 14.3 percent with gains at 0.5:0.95 jumped by 17.1%, while reducing the model size

by about 31.7%. Overall detection accuracy improved, especially when dealing with small targets. However, there was a slight downside in that the improved model took a little longer to process each image, so the impact on speed must be considered. Overall, the results show that this approach is effective for detecting small objects in complex, realistic remote sensing scenarios. **This result highlights an important trade-off in Edge AI: improving detection capability and reducing model size does not necessarily lead to lower latency. Therefore, the suitability of the architecture depends on whether the application prioritizes detection accuracy and small-object sensitivity or requires the lowest possible inference time.**

Overall, real-world applications show that there is no single optimization strategy that satisfies all Edge AI requirements. Safety-critical and real-time applications may prioritize latency and reliability, healthcare applications may place greater emphasis on predictive performance, while battery-powered or highly constrained devices may prioritize memory and energy consumption. Consequently, compression strategy selection should be driven by the dominant application constraints and validated under realistic deployment conditions.

3.4 Hybrid lightweight model

IN[9], this paper presents how hard it is to deploy large models directly onto small gadgets. Though cloud reliance offers power, shifting processing closer to where data forms brings perks like speed and tighter control over private details. Heavy networks simply do not fit inside low-memory systems, making immediate decisions tricky when internet links fail or lag. This work explores methods to shrink those bulky models so they can actually function within tight hardware limits. Rather than sticking to classic test scenarios, researchers picked practical problems - spotting road damage, judging crop ripeness, telling crops apart from invasive plants. Each visual collection got treated with common adjustments before any modeling began. First came building a standard-sized vision-focused network to measure later results against. From there, compression stepped in: trimming unneeded parts, reducing numerical precision, even transferring learned patterns into smaller versions. Shrinking smartly matters more now that cameras and sensors spread everywhere, yet few solutions balance slimness with performance. Models must stay useful after dieting down through digital thinning processes. One way helped shrink the model, though results shifted based on what it had to do; sometimes precision dropped too far. Because of that challenge, researchers mixed distillation, trimming unused parts, and lighter data formats only after learning finished. That mix worked better than earlier tries. With fewer numbers inside, less space needed, yet performance stayed close to the first version - sometimes stronger. Learning did not take long, tests showed solid outcomes across each group of examples used. Clearly, relying on just one compression approach falls short; however, pairing methods wisely leads to compact models that keep solid accuracy for edge deployment. Instead of pushing extremes, blending

techniques balances performance and size more effectively than standalone solutions ever could.

[67] This study examines model compression techniques applied to ConvNeXt-like architectures deployed on edge hardware limited by computation and memory. Using the CIFAR-10 collection - sixty thousand colored images divided evenly among ten classes - it evaluates several strategies: structured pruning via OTOV3, unstructured weight elimination, dynamic precision scaling, and combined variants, tested over multiple sizes of ConvNeXt models. Surprisingly, discarding parameters randomly shows almost no meaningful impact under practical conditions. Even when data grows thinner, the storage needs and computational load stay nearly the same. Unlike earlier versions, OTOV3 reworks its core structure - removing entire layers or neurons without compromise. Surprisingly, accuracy does not suffer at all. In fact, after refinement, the output gets a small lift. Beyond that, incorporating adaptive quantization pushes outcomes even higher. Turning weights into 8-bit numbers cuts memory needs and boosts speed without much loss in accuracy. When combined with OTOV3, results improved sharply. That pairing shrank the model close to 89.7%, slashing parameters and MACs by about 95%. Even with drastic shrinking, precision went higher, rising 3.8%. On edge hardware, it stayed steady: 92.5% of guesses right, each done in 20 ms. The bigger takeaway here is that a modern architecture like ConvNeXt, which isn't exactly small to begin with, can be trimmed down into something lightweight and efficient enough to run on edge hardware without giving up performance.

IN[64] This study addresses a fundamental problem faced by many working in the field of 5G and artificial intelligence: when all data is sent to cloud computing, response times increase significantly, leading to high bandwidth consumption. When attempting to do everything at the edge computing level, one quickly encounter hardware and battery limitations. Therefore, the researchers propose a three-tier system—edge computing, fog computing, and cloud computing—and test it using MATLAB/Simulink with simulated inputs that mimic real-world scenarios. These inputs include electrocardiogram signals (ECG) (normal and abnormal), noisy motion data, Internet of Things sensor data, and small grayscale surveillance images with a resolution of 28×28 pixels. The system operates on several levels. At the edge computing level, there is a lightweight convolutional neural network that can stop early if it is confident enough in the accuracy of its predictions—the threshold has been set at around 0.85. Furthermore, the reinforcement learning agent monitors factors such as available bandwidth, battery level, CPU load, and edge model reliability, and uses all of this to make an immediate decision on whether a given task should remain at the edge or be transferred to the fog or cloud layer. There is also a privacy-preserving federated learning with FedAvg-style aggregation enhanced by a trust-score module that down-weights or discards unreliable/malicious client updates. Results show clear trade-offs across tiers. Edge processing is fast—about 8.5 milliseconds on average—and energy efficient at about 0.12 joules per task, but accuracy is only about 84.7%, and bandwidth consumption is minimal,

at about 5 kilobytes. Fog computing comes in the middle with a response time of about 22.1 milliseconds, energy expenditure of 0.34 joules, accuracy of 89.6%, and bandwidth of 15 kilobytes. Cloud computing offers the best accuracy at 93.4%, but at a high cost of a response time of nearly 100 milliseconds, energy consumption of 0.57 joules, and bandwidth of 50 kilobytes per task. A notable finding is that the early exit mechanism is activated for approximately 62.5% of edge computing tasks, saving a significant amount of computational power. Trust-based aggregation has also proven useful in maintaining the strength of the unified model when some nodes are not functioning well. Figure 7 provides an overview of how edge computing integrates with cloud computing. Similarly, [65] This paper focuses on mission-critical hybrid AI-edge systems, emphasizing the splitting of functionality between cloud and edge to meet strict latency, reliability, and security requirements. Time-critical inference is pushed to edge devices, while the cloud supports large-scale training, coordination, and updates. Typical approaches combine adaptive orchestration and dynamic task/model partitioning with edge-friendly model optimization (e.g., pruning, quantization, and knowledge distillation), and privacy/security mechanisms such as federated learning, lightweight cryptography (sometimes homomorphic/SMPC), and model-update integrity via authentication/ledger techniques; some designs also integrate lightweight explainable AI to improve operator trust. Most assessments rely on simulated environments - like self-driving tasks, medical tracking, or combat drills - not one fixed benchmark. These show response delays dropping sharply: around 7 milliseconds instead of 120. Accuracy dips slightly, about two percent. Systems also bounce back better during crashes or intrusions. Performance costs stay manageable. Scaling across many devices works without major hiccups. **These studies indicate that, when an application cannot satisfy all requirements through model optimization alone, computation placement and system architecture can become additional mechanisms for balancing performance and resource constraints. They should therefore be considered complementary to model compression rather than as alternative compression techniques.**

3.5 Cross-Method Comparative Analysis

The reviewed studies show that quantization, pruning, knowledge distillation, and hybrid compression address different sources of model inefficiency and therefore involve different trade-offs. Rather than considering one technique universally preferable, the appropriate strategy depends on the deployment objective, model characteristics, available training resources, and target hardware.

3.5.1 Accuracy-Compression Trade-offs

The reviewed studies indicate that compression effectiveness should be evaluated by considering the balance between model compactness and predictive performance. Quantization can provide substantial reductions in model representation and memory requirements, but aggressive precision reduction may affect accuracy depending on the model and quantization configuration. Pruning can achieve considerable parameter and computational reductions, particularly when redundant structural components are

removed, although excessive pruning may reduce model capacity and predictive performance.

Knowledge distillation provides a different approach by improving the predictive capability of a compact student model through knowledge transferred from a larger teacher. This can help maintain accuracy while using a smaller deployment model, but its effectiveness depends on the teacher–student configuration and introduces additional training requirements.

Hybrid compression can combine the complementary effects of multiple techniques and may therefore provide favorable accuracy–efficiency trade-offs in specific settings. However, the reviewed evidence does not support a universal conclusion that hybrid approaches outperform individual techniques. Their effectiveness depends on the selected combination, compression configuration, model architecture, and deployment requirements.

Overall, the accuracy–compression trade-off should be evaluated according to the requirements of the target application rather than according to compression ratio alone. A method that produces greater parameter or model-size reduction may not necessarily provide the best predictive performance or practical deployment efficiency.

3.5.2 Computational, Memory, and Energy Efficiency

The reviewed studies also demonstrate that different compression methods influence computational and resource requirements in different ways. Quantization reduces the numerical representation of model parameters and activations and can provide practical benefits in memory usage and computation when the selected precision is efficiently supported by the target hardware. Pruning reduces parameters or computational structures and can lower the theoretical computational workload, particularly when structured reductions are introduced.

Knowledge distillation achieves efficiency primarily through the resulting compact student model rather than through direct numerical or structural modification of an existing model. Consequently, its deployment cost is determined by the student architecture, while the additional teacher model and distillation procedure mainly affect the training stage.

The reported results further indicate that reductions in parameters, FLOPs/MACs, model size, or numerical precision should not be interpreted as direct measurements of latency or energy consumption. These model-level indicators describe potential computational efficiency, whereas latency, throughput, memory consumption, and energy consumption depend on the target hardware, inference framework, implementation, and execution conditions. Therefore, measured deployment metrics are required to determine whether a reported compression gain translates into a practical edge-device benefit.

3.5.3 Hardware Compatibility and Deployment Requirements

Hardware compatibility is an important consideration when selecting a compression strategy. Quantization can provide significant deployment benefits when the target processor or accelerator supports the selected low-precision format efficiently. However, the same precision reduction may provide limited practical acceleration on hardware or software stacks that do not

efficiently exploit the corresponding numerical representation.

Pruning also demonstrates different hardware implications depending on the resulting model structure. Structured pruning can provide more direct computational benefits because the resulting model remains compatible with conventional dense execution, whereas the practical acceleration obtained from unstructured sparsity depends more strongly on the availability of efficient sparse computation support.

Knowledge distillation generally has limited hardware requirements during deployment because only the student model is executed. Its main computational overhead occurs during training, when the teacher and student models are jointly involved in the distillation process. This makes KD attractive when a compact student model can satisfy the computational constraints of the target device but requires improved predictive performance.

Hybrid compression introduces additional deployment considerations because the target platform must efficiently support the combined effects of the selected compression methods. Consequently, hardware compatibility should be evaluated for the final compressed model rather than for each compression technique in isolation.

3.5.4 Training and Implementation Complexity

The reviewed approaches also differ in their training and implementation requirements. Post-training quantization can be applied after model training and may therefore require less modification to the original training procedure, whereas training-aware quantization requires additional training or fine-tuning. Pruning may similarly require fine-tuning after parameters or structural components have been removed, particularly when relatively high compression levels are targeted.

Knowledge distillation requires an additional teacher–student training procedure. Although this increases development complexity, the teacher does not need to be retained during deployment. Hybrid compression can introduce greater implementation complexity because multiple compression mechanisms must be configured, evaluated, and potentially fine-tuned together.

Accordingly, the selection of a compression strategy should be application- and hardware-dependent. Quantization may be appropriate when memory efficiency and low-precision execution are priorities; structured pruning may be suitable when reducing computational workload through regular model structures is required; knowledge distillation may be beneficial when a compact student requires improved predictive performance; and hybrid compression may be considered when multiple resource constraints must be addressed simultaneously. In all cases, the final compressed model should be evaluated on the intended hardware and inference framework to determine its actual deployment performance.

3.6 Critical Comparison of Compression Strategies

Quantization is most appropriate when memory footprint and low-precision arithmetic are important and the target processor or accelerator provides efficient support for the selected precision. Post-training

quantization (PTQ) generally offers lower development complexity because the already-trained model can be converted using calibration data, whereas quantization-aware training (QAT) introduces additional training cost but can be preferable when aggressive precision reduction causes unacceptable accuracy degradation.

Structured pruning is preferable when the goal is to physically reduce filters, channels, or other regular network structures that conventional dense hardware can execute efficiently. Unstructured pruning can produce high sparsity but does not necessarily produce proportional latency gains unless the inference stack provides efficient sparse kernels or hardware support.

Knowledge distillation is preferable when a compact student model needs additional predictive capability without retaining the teacher during inference. Its principal trade-off is additional training complexity rather than inference-time model overhead.

Hybrid compression can be preferable when a single technique cannot satisfy all deployment constraints. The submitted evidence supports complementarity—for example, structural reduction combined with precision reduction or distillation—but does not justify the stronger claim that hybrid methods are universally superior. Hybrid pipelines also increase configuration, tuning, and deployment complexity.

3.7 Limitations of Cross-Study Comparison

Direct numerical comparison across the reviewed papers is limited because the studies use different datasets, tasks, model architectures, input resolutions, batch sizes, software frameworks, hardware accelerators, and measurement protocols. For example, an inference result reported in frames per second on a Raspberry Pi cannot be treated as numerically equivalent to a latency measurement in milliseconds on an Intel CPU, even when both describe deployment on an edge device. Similarly, parameter reduction, FLOPs/MACs reduction, serialized model-size reduction, runtime memory, power, and energy are distinct metrics and should not be substituted for one another.

Where possible, the revised comparison therefore reports the original metric exactly as presented in the submitted manuscript and marks unavailable values as NR rather than estimating them. This prevents the systematic table from creating false precision or implying cross-study equivalence that the experimental protocols do not support.

3.8 Synthesis

The reviewed studies indicate that there is no single compression technique that is optimal for all Edge-AI deployments. Quantization is attractive for reducing numerical representation and enabling low-precision execution; structured pruning is attractive when regular network structures can be removed while retaining hardware-efficient dense execution; knowledge distillation is useful for transferring predictive behavior from larger models to compact students; and hybrid

methods can address multiple resource constraints simultaneously. The appropriate choice should therefore be determined by the dominant deployment constraint, the required accuracy, the available training budget, and—critically—the capabilities of the target hardware and inference framework.

Hybrid methods can produce strong trade-offs in specific architectures and deployment settings, but their superiority cannot be established without controlled experiments under matched datasets, models, hardware, and evaluation protocols.

4. METHODS

4.1 AI on Edge: Framework

TensorFlow Lite, commonly known as TFLite, originated when Google examined TensorFlow. It simplifies the framework so that AI models operate on devices with limited processing capacity. The included toolkit provides reliable utilities that enable models to function on hardware that would normally fail to support them. TFLite has become widespread when developers must deploy a model on small, inexpensive hardware, they typically select TFLite first. [37,60].

PyTorch ecosystem to address a related challenge, but they apply distinct approaches. When a model has already been trained in standard PyTorch, and the goal is to execute it on a smartphone or on an Internet-of-Things unit without repeated requests to a remote server, PyTorch Mobile provides the necessary framework. The library supports execution on a GPU or on a neural processing unit (NPU) when such hardware is present; latency stays low. Developers have produced augmented reality programs, robot control modules, and image classification utilities that operate locally on the device and that do not rely on cloud computation. [59].

ONNX removes a problem that frustrated developers for years in the past: a model trained with PyTorch required a complex conversion before it ran with TensorFlow tools. ONNX supplies a single file format that works across frameworks. Train in one environment, export to ONNX, then execute the same file on a different stack. Because the specification is open source, adoption spread quickly [38,59].

4.2 AI on Edge Light model

Lightweight models began with a simple requirement. Deep learning models expanded and became more intricate, and engineers understood that extra hardware alone would not solve every issue. Investigators then searched for methods to lower model size while accuracy remained intact. The approach decreases the number of parameters and lowers the required amount of calculation, but the model stays adequate for tasks. Several compression techniques appeared, and they also demonstrated strong results in real settings [44,45]. Table 2 offers a direct comparison of well-known lightweight models and shows how each one measures against the rest.

4.2.1 Quantization

Neural networks hold their weights as well as activations in 32-bit floating-point format by default. For years, no one challenged that choice. Researchers asked whether such precision was necessary. The answer was negative. The weights can shrink to 16-bit or 8-bit form, sometimes even to integers, and the network still delivers solid accuracy. Quantization performs this change. It replaces the large, high-precision numbers with smaller ones that occupy less memory. Quantization reduces the numerical precision of weights and/or activations, which can reduce model memory requirements and, when supported by the target hardware and inference framework, may also reduce inference latency and energy consumption [9,46]. The quantization process needs to map real values $x \in [x_{\min}, x_{\max}]$ to integer values $q \in [q_{\min}, q_{\max}]$ can be delineated through the following Equation 1 :

$$q = \text{round}(x/s) + z \quad [1]$$

x represents input real numbers. s represents scale factor (float $\rightarrow$ int). z represents zero-point (offset, to represent negative values). q : represents output quantized integers
De-quantization Equation shown in Equation 2.

$$x^{\wedge} = s \cdot (q - z) \quad [2]$$

$x^{\wedge}$ represents dequantized weight (approximation of x).

Scale: $S = (x_{\max} - x_{\min}) / (q_{\max} - q_{\min})$

Zero-point: $Z = q_{\min} - (x_{\min}/S)$

When we need to decide the range of values $[q_{\min}, q_{\max}]$ that will be mapped into low-bit integers. But this introduces two sources of error: Clipping error, when values fall outside the chosen range, they are cut off. Rounding error when values inside the range are rounded to discrete quantization levels. There are different methods to try to balance these two errors[62].

A. Min-Max Quantization

Range is set to the minimum and maximum of all values. An advantage that covers the entire data, with no clipping error. Disadvantage: It is very sensitive to outliers (a single extreme value makes the range too wide, which increases rounding error).

B. Percentile-based Quantization

Instead of using the absolute min/max, use percentiles of the data distribution. Advantage that ignores extreme outliers and focuses on the “main body” of data. Disadvantage is still needs dataset statistics.

4.2.1.1 Taxonomy of Model Quantization

Quantization techniques can be classified according to several independent dimensions, including the training stage, activation handling, numerical representation, quantization granularity, and precision allocation.

A. Symmetric vs. Asymmetric:

Symmetric and asymmetric quantization differ in how the real-valued range is mapped to the integer representation.

Symmetric quantization

Zero-point is always 0. The range is symmetric around zero. Simpler math is often more efficient for hardware (like GPUs/TPUs) because implementation efficiency depends on the specific accelerator, operator, and software stack. Works well if data (weights/activations)

are centered around zero. But not flexible if the data is skewed [47].

Asymmetric quantization

Useful when activations or weights are not symmetric around zero. More flexible, but slightly more complex computation.

B. Uniform vs. Non-uniform:

Quantization can also be distinguished according to the spacing between consecutive quantization levels.

Uniform quantization

Values within the same quantization step size are mapped to the same quantization level. Easy to implement in hardware.

Non-uniform quantization

Non-uniform quantization is proposed, whose quantization steps are non-uniformly spaced. More accurate if the data has a skewed distribution. Harder to implement efficiently in hardware [47].

C. Static vs. Dynamic quantization:

Static and dynamic quantization differ primarily in how activation quantization parameters are determined during inference.

Static quantization

Quantization parameters (scale & zero-point) are pre-computed using a calibration dataset. The time to decide the clipping range ($q_{\min}, q_{\max}$) is crucial and can be divided into static and dynamic quantization. Requires a calibration dataset and an extra pre-processing step.

Dynamic quantization

Weights are quantized once before inference (to int8, for example). Activations are quantized dynamically at runtime — meaning scale and zero-point are computed on the fly for each batch. Common for activations, especially in NLP models. No calibration dataset needed. Very simple to apply but slightly slower (extra cost to compute activation ranges at runtime).

D. Post-training quantization (PTQ) vs. Quantization-aware training(QAT):

The distinction between post-training quantization (PTQ) and quantization-aware training (QAT) is based on when quantization is incorporated into the model development process.

Post-training quantization (PTQ)

Quantize a trained model after training. Fast, no retraining required. May reduce accuracy significantly, especially for small models or sensitive layers.

Quantization-aware training(QAT)

The idea is to integrate quantization directly into the training process. So while the model is learning, it's also getting exposed to the kind of noise and rounding errors that come with lower precision. It basically gets a chance to adjust and compensate for all that during training, which is why the accuracy you end up with tends to be noticeably better than what you would get if you just quantified the model afterwards. Quantization-aware training shows a clear lead once the target format shrinks to eight-bit

integers. The drawback is that it needs a substantial training set, and must adjust both the model weights and the quantization together. This forces you to restart training from the beginning or from a point very close to it, which costs extra hours, extra processor cycles, and extra human work.

The choice between PTQ and QAT depends on the target bit-width, model sensitivity, availability of training data, computational budget, and supported deployment hardware. PTQ is often suitable when deployment simplicity and limited training resources are priorities, whereas QAT may be beneficial when preserving accuracy under aggressive quantization is essential [47].

E. Per-Tensor vs. Per-Channel Quantization.

The level of detail applied during the quantization process dictates the balance between computational simplicity and the preservation of model accuracy. This granularity is generally divided into two approaches:

Per-Tensor Quantization

This technique maps an entire tensor using one shared scale factor and a single zero-point. It minimizes memory overhead and simplifies the mathematical operations required during inference, making it highly compatible with basic hardware. The downside emerges when the tensor contains extreme outliers or wide numerical variations. A single broad range forces smaller, typical weights into very few discrete levels, which magnifies the quantization error and often degrades the final predictive performance.

Per-Channel Quantization

Instead of treating the whole tensor uniformly, this alternative assigns distinct quantization parameters to each slice, typically along the output channels of a convolutional filter. Adapting the scale to fit the exact distribution of individual channels isolates the impact of outliers and protects the model's accuracy. The trade-off is architectural dependency. While storing extra scaling factors adds only minor memory overhead, the real challenge is computational support. The chosen edge accelerator and its paired software libraries must possess specific low-level instructions to process varying scales efficiently. Without native hardware support, the actual inference latency will worsen despite the improved accuracy.

F. Fixed-Precision versus Mixed-Precision Quantization

Another critical design choice in model compression is determining whether the bit-width remains constant or varies across the network architecture.

Fixed-Precision Quantization

This strategy enforces a uniform bit-width (such as 8-bit integers) across all selected layers and operations. The primary advantage of this approach is deployment simplicity. Since the data format remains consistent throughout the network, inference engines can execute operations smoothly without requiring complex scheduling logic. This makes it highly compatible with almost all standard

edge hardware. However, applying a rigid, aggressive low-bit format to the entire model often degrades accuracy, as highly sensitive layers are forced into the same constrained numerical representation as less important ones.

Mixed-Precision Quantization

Instead of a blanket approach, this method allocates varying bit-widths to different sections of the model based on their individual sensitivity to information loss. For instance, robust layers might be aggressively compressed to 4-bit formats, while critical feature extractors are kept at 16-bit precision to preserve accuracy. This targeted allocation provides an optimal balance between maintaining the network's predictive power and minimizing the computational footprint. The main obstacle is hardware compatibility; extracting actual speedups from this approach requires an edge accelerator capable of seamlessly handling multiple numerical formats and switching between them at runtime without introducing latency bottlenecks.

G. Relationship Between Quantization Dimensions

The aforementioned quantization categories represent complementary dimensions rather than mutually exclusive approaches. PTQ and QAT classify quantization according to the training stage, whereas static and dynamic quantization distinguish how activation quantization parameters are determined. Symmetric and asymmetric quantization describe the numerical mapping strategy, while per-tensor and per-channel quantization define the granularity of quantization parameters. Furthermore, fixed-precision and mixed-precision quantization determine the bit-width allocation across the network. These dimensions can be combined to construct quantization pipelines tailored to specific edge deployment requirements. For example, a model may employ per-channel symmetric weight quantization with static INT8 activation quantization using a PTQ approach. The selection of an appropriate configuration depends on model sensitivity, accuracy requirements, calibration data availability, and hardware support.

4.2.2 Taxonomy of Pruning

In prune, we remove unneeded parameters from deep networks. After training a network with millions of coefficients, many of those coefficients do not make a meaningful contribution to the final output after training is finished. Thus, pruning is the obvious solution: we remove unneeded weights from the network! With pruning, we evaluate each connection in the network and rank it by the amount of contribution it provides. Depending on the pruning strategy, it can reduce the number of non-zero parameters, the dimensions of the network, or its theoretical computational complexity. However, these reductions do not necessarily translate into smaller model files, lower memory consumption, or faster inference. In particular, removing individual weights while retaining the original

dense tensor dimensions may reduce the number of non-zero parameters without reducing the computations performed by conventional dense inference kernels. Actual deployment benefits depend on the resulting model representation, hardware capabilities, and the support provided by the inference software. However, to ensure good accuracy of the model, we need to remove weights with care [48]. If we remove too many weights during the pruning process, we can lose important connections and as a result, performance will drop significantly. This situation is referred to as over-pruning. An example of how over-pruning occurs with trees would be to remove dead wood; some is acceptable, but if we remove too much, we will kill the tree [63]. An illustration of the difference between a fully connected network and a compressed network after pruning is shown in Figure 8 [63].

Structured Pruning vs. Unstructured Pruning: **Structured Pruning**

Through the process of structural pruning of the network, the overall structure of the network is preserved by removing the whole block in one operation so that the processor uses less memory and eliminates entire calculations, such as filters, channels and neurons, creating a lighter, faster and more efficient model on edge devices than prior to structural pruning. One consequence of structural pruning on the overall network's size is a considerable decrease in physical size and fewer bytes are used than for the original, pre-structural pruned model, while maintaining an accuracy greater than 90%. Structured pruning that physically removes channels or filters may reduce theoretical FLOPs and MACs, but the resulting latency improvement still depends on the target hardware and implementation. Structural pruning can also be combined with other compression methods to be very useful in deploying to edge devices that have very limited time and energy [11, 51]. There are several ways to perform structural pruning, the most common is to rank filters/channels by using either L1 or L2 criteria and to remove the lowest ranked filters/channels.

Unstructured Pruning:

Unstructured pruning reduces the size of the model in a clear and direct manner, as it examines every weight in the network and replaces the least significant weight with zero. Consequently, accuracy decreases and produces a chaotic sparsity pattern because weights are set to zero at random locations without regard for the network's structure, leaving the network irregular. Crucially, this unstructured sparsity does not always translate into actual speedups. Unlike structured pruning, reducing FLOPs through unstructured pruning requires specific target hardware and inference libraries optimized for sparse computation to achieve lower inference latency. For example, unstructured pruning may substantially increase the sparsity ratio without reducing the operations executed by dense inference kernels. To restore accuracy, the network must be retrained after

intensive pruning, which requires significant computational power. [11].

B. Layer-Wise vs. Global Pruning

The scope of how sparsity is allocated across a neural network fundamentally shapes both its predictive performance and structural efficiency.

Layer-Wise Pruning

This method treats each layer in isolation, enforcing a specific sparsity target on its parameters. The primary advantage is tight structural control, ensuring that no single layer is inadvertently decimated. However, applying rigid or uniform pruning thresholds without measuring individual layer sensitivity often degrades the network's accuracy, as critical feature extractors may be forced to drop essential weights.

Global Pruning

Instead of confining the pruning budget to individual layers, this approach evaluates and ranks all weights across the entire architecture simultaneously. Consequently, highly redundant layers undergo aggressive compression, while sensitive layers retain the majority of their parameters. Although this adaptive distribution theoretically optimizes the network's capacity, actual accuracy retention relies heavily on the quality of the chosen importance metric and the subsequent fine-tuning strategy. Ultimately, selecting between global and layer-wise techniques requires balancing target sparsity against hardware constraints.

C. Static vs. Dynamic Pruning

Pruning methods also diverge based on whether the network's sparse topology is permanently frozen or continuously adapted during operation.

Static Pruning:

The network architecture is permanently modified prior to deployment. Once redundant weights are identified and pruned, the resulting structure remains fixed throughout inference. If structural elements (such as entire filters) are physically removed, standard edge hardware can immediately benefit from the reduced computational load. However, the extent of any actual inference speedup depends on hardware capabilities, supported operators, and the optimization provided by the inference framework. Conversely, if the pruned weights are merely masked as zeros within a dense tensor, realizing actual latency improvements is highly unlikely without hardware specifically designed for sparse matrix multiplication.

Dynamic Pruning:

This technique alters the computational graph on the fly, selectively activating or bypassing specific channels and blocks based on the complexity of the incoming data. While skipping unnecessary computations naturally reduces the workload, the conditional logic required to make these runtime decisions introduces its own overhead. Translating this input-adaptive behavior into measurable speedups depends heavily on highly optimized software kernels and inference engines.

D. Relationship Between Pruning Dimensions

The pruning categories represent complementary dimensions that can be combined to develop different pruning strategies. Structured and unstructured pruning determine the granularity of parameter removal, whereas layer-wise and global pruning define how the pruning budget is allocated across the network. Static and dynamic pruning distinguish whether the pruned structure remains fixed or adapts during inference. Consequently, pruning methods can combine these dimensions, such as global

structured pruning or layer-wise unstructured pruning. The effectiveness of each combination depends on the selected importance criterion, target sparsity, accuracy requirements, and hardware capabilities. In edge environments, structured pruning is generally compatible with conventional dense execution when it produces physically smaller network dimensions, whereas unstructured pruning and input-adaptive dynamic pruning may require specialized software or hardware support to achieve practical inference acceleration.

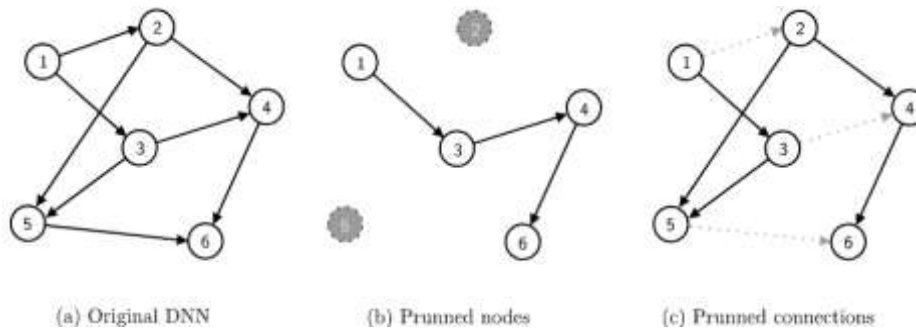


Figure 8. Weight pruning in a deep neural network proceeds through three stages. Part (a) presents the original network - every link is present and every neuron is active. Part (b) marks the neurons and links that contribute little to the output - those elements are selected for deletion. Part (c) displays the network after the operation - the removed links are drawn as dashed lines - the amount of reduction is visible [63].

4.2.3 Taxonomy of Knowledge Distillation

Knowledge distillation is a machine learning method that reduces a large and complex "teacher" model into a smaller "student" model. The large model, named the teacher, passes its knowledge to the small model, named the student. The student mimics the teacher's predictions. It uses the "soft targets", which contains the probability value for every class, not just the one with the highest score. This supplies the student with a richer training signal. The final model executes faster and needs less memory but it achieves accuracy almost equal to that of the teacher [49,8]. Engineers adopt this approach when they place deep learning on devices that have tight limits on memory and power, like mobile phones or drones [52,62].

A. Response-Based Distillation:

This classical scheme targets the very end of the network pipeline. The compact student architecture learns by aligning its final logits with the softened prediction

probabilities produced by the teacher via temperature scaling. Rather than learning solely from discrete, binary ground-truth labels, the student absorbs the teacher's soft distribution, which inherently reflects nuanced similarities and relationships across non-target classes.

B. Feature-Based Distillation:

Recognizing that final predictions discard rich internal transformations, this approach transfers spatial and contextual cues directly from the teacher's intermediate layers. The student is guided by intermediate feature maps or latent activation states extracted from deeper stages. This structural guidance proves particularly effective for dense visual computing tasks like object detection, where fine-grained spatial representations directly dictate output qua

C. Relation-Based Distillation:

Instead of matching static layer outputs or individual embeddings in isolation, this strategy aims to replicate the geometric structure formed across multiple inputs. The student is trained to reconstruct relational dependencies, such as cross-sample similarity metrics, metric distances, or topological feature graphs captured by the teacher. In doing so, it captures the overarching structural manifold rather than memorizing isolated data points.

D. Self-Distillation:

Standard KD workflows require designing, training, and storing an expensive external teacher, which increases training resource overhead. Self-distillation circumvents this bottleneck by deriving the training signal directly from within the target network itself. Typical setups route supervisory guidance from deeper internal blocks down to shallower layers, train auxiliary multi-exit classifiers concurrently, or use historical weights from earlier epochs as a dynamic teacher. This self-contained setup provides regularizing benefits while drastically cutting computational demands.

E. Comparative Perspective on Knowledge Distillation Strategies

The four knowledge distillation categories differ according to the type of supervisory information transferred from the teacher to the student. Response-based distillation transfers final prediction distributions, feature-based distillation utilizes intermediate representations, relation-based distillation preserves relationships among samples or learned features, and self-distillation derives supervisory signals from within the same network or training process. These strategies are not necessarily mutually exclusive and can be combined within hybrid distillation frameworks. For edge deployment, the choice of distillation strategy should consider the student model's architecture, the complexity of the training procedure, the availability of a pretrained teacher, and the computational and memory constraints of the target device. Although knowledge distillation can facilitate the development of compact models, its effectiveness should be evaluated using both predictive performance and deployment metrics, including model size, inference latency, memory consumption, and energy efficiency.

4.3 Lightweight model design

Searching for efficient and effective neural network (NN) architectures is important. Two critical approaches: lightweight model design and model compression. Both methodologies aim to enhance the ease of deployment and performance of deep neural networks (DNNs), especially in resource-constrained environments. Lightweight model design focuses on constructing deep neural networks with an emphasis on minimalism and efficiency. The objective is to create inherently efficient models that can operate effectively on devices with reduced computational capacity and memory, such as smartphones, IoT appliances, and embedded systems, and make these post-training models smaller and easier to use in limited resource settings. These methods are used to make networks smaller-scale. They

balance keeping the network efficient while reducing its size. Table 3 summarizes lightweight model architectures [63].

4.3.1 SqueezeNet architecture:

SqueezeNet rests on a component named the fire module. This module is split into two stages. A squeeze layer first applies only 1x1 convolutions. An expand layer then mixes 1x1 and 3x3 convolutions. Those small filters shrink the total parameter count far below typical levels, but accuracy on standard image benchmarks stays competitive. The compact size suits storage-starved and memory-starved settings like embedded systems or apps running on mobile devices.

4.3.2 ShuffleNetV2 architecture:

ShuffleNetV2 keeps speed high while it preserves accuracy - channel shuffling lets the network share features again without extra parameters. The design also trims computational overhead through careful control of memory use plus of data motion between layers. The resulting speed suits real-time phone jobs like object detection and image classification.

4.3.3 MobileNetV2 architecture:

MobileNetV2 was launched by Google. The model replaces standard convolutions with depthwise separable convolutions, which split the operation into smaller and cheaper steps. This change alone reduces computation. The key novelty is the inverted residual structure. Traditional residual links stay wide - the inverted version starts narrow, expands through pointwise convolutions to gather many features, then narrows again. An efficient "expand then compress" pattern running through the whole network. The network learns complex representations but avoids the high cost of larger models.

4.3.4 MobileNetV3 architecture:

Google advanced the line with MobileNetV3 - engineers used neural architecture search (NAS), an algorithm that discovers layouts tuned for mobile hardware. Squeeze-and-Excitation blocks (SE blocks) were inserted, which let the network emphasize channels that carry useful information for each input. The team also adopted the hard-swish activation function, a choice that raises both speed and accuracy. All updates combined in V3 perform better on phones and on devices that have limited resources.

4.3.5 EfficientNet-Lite architecture:

EfficientNet shows that you do not require a huge model that consumes many resources to obtain strong results. Earlier architectures usually altered only one aspect - they added extra layers or expanded the network. EfficientNet changes depth, width, and resolution together through a method named compound scaling, and it achieves notable performance. EfficientNet-Lite extends this idea for users who want a version that operates on a phone or an edge device without draining the battery or filling the memory. It keeps most of the accuracy of the original model while it runs faster, plus uses fewer parameters - it suits deployment outside a powerful cloud server.

Table 2: Systematic Comparison of Model Compression Techniques for Edge AI

Technique	Core idea / How it works	Advantages (When it's useful)	Drawbacks / Trade-offs / When to be careful
Quantization	Reduce numerical precision of weights and activations (e.g., from 32-bit floating point to 8-bit integers or lower) [56].	- Speeds up inference (integer operations are typically faster / more efficient than high-precision floating point) and good for deployment on resource/constrained hardware [56]. - Improves energy efficiency, important for battery-powered or low-power devices.	- Reduced precision may degrade model accuracy, especially for sensitive tasks or aggressive quantization (e.g., 4-bit, binary). - Some quantization methods (like Quantization-Aware Training, QAT) require retraining with simulated quantization noise to preserve accuracy, which increases complexity.
Pruning	Remove redundant or “unimportant” weights/connections (could be individual weights unstructured, or structured) to shrink the neural network.	- Reduces model size (fewer parameters) and computation, and less memory / faster inference - Can be applied after or during training (so it doesn’t require completely redesigning the network). - Good for edge devices or embedded systems with limited storage/compute.	- If over-pruned, accuracy can drop (especially without proper fine-tuning). - Unstructured sparsity may lead to irregular memory/computation patterns, and some hardware doesn’t efficiently leverage sparsity, so speedup may be limited [55].
Knowledge Distillation (KD)	Use a large, well-trained “teacher” model, then train a smaller “student” model to mimic the teacher’s output (soft labels or internal representations). The student, therefore, learns to approximate the teacher’s behavior with far fewer parameters.	- The student model can retain a large portion of the teacher’s accuracy while being significantly lighter. - Good for edge / resource-constrained deployments (storage, memory, compute). - Often results in better generalization/robustness than small models trained from scratch.	- Requires a teacher model and additional training for student, so training complexity and cost increase.

Table 3: Summary of lightweight model architectures

Architecture	year of launch	Key Features	Impact and Applications
MobileNetV2	2018	Depthwise separable convolutions, inverted residuals, linear bottlenecks	Improved mobile device efficiency, suitable for real-time image classification and mobile applications
EfficientNet-Lite	2019	Compound scaling, optimized for mobile devices	High efficiency with better resource utilization, adaptable for mobile devices with fewer resources
ShuffleNetV2	2018	Optimized channel operations, enhanced channel split	High accuracy for mobile and edge devices, real-time image classification, and object detection
SqueezeNet	2016	Efficient for low resource devices	Extremely small model size, ideal for resource-constrained environments and mobile applications
EfficientNetV2	2021	Progressive learning, fused convolution blocks	Improved training efficiency and accuracy for various deep learning tasks, setting new benchmarks in efficient neural network design

4.4 Architectural Paradigms for Edge Intelligence Deployment

Edge devices run artificial intelligence while they cope with restricted compute capacity, privacy requirements, and the demand for low latency. Those conditions show why distinct edge AI architectures exist - each one tackles the listed constraints. Edge AI systems are built in multiple ways, each chosen to match a specific goal like low delay, data privacy, or low resource use [6].

The Client-Server pattern illustrates one option. The edge device gathers data and performs only simple operations. It then sends the data to a central server, but that server also executes heavy tasks like inference with large models or distribution of updates [39]. This arrangement suits workloads that need high compute power.

Federated learning changes how data is shared. Devices train models on their own storage and send only the learned

parameters to a central server [50, 53, 62]. The raw data stays on the device personal information does not travel across the network. This protects user privacy.

A pipeline architecture imitates a factory line - one node collects raw samples, the next node filters those samples and

a further node executes inference. The sequence repeats for every stage. Each step starts immediately after the previous step ends, latency remains minimal. Video analytics systems but also factory control loops gain advantage from this rapid succession [41].

Distributed architecture assigns tasks to many nodes right away. If a single node stops, the rest of the nodes keep working, so the service stays online. When traffic rises, new nodes join the group and the system grows without a central point of failure. Autonomous vehicles as well as city sensor grids require this constant availability [54]. When one node breaks, the others carry on. The system endures faults and spreads the workload across every active node. The design expands easily as demand increases [42].

Hierarchical architecture arranges nodes into separate layers. The lowest layer contains small sensors that capture measurements. Mid layer gateways compress the data or forward the streams to upper levels. The top layer houses powerful servers that execute heavy analytics. Hospitals, but also industrial parks, adopt this pattern to share load and to limit hardware expense [43].

No single “best” architecture appears as the final answer - every design elevates one priority - power, privacy, speed, or scalability. The option that fits your project hinges on the precise requirements of the application. Tables 4.5 provide a concise summary of the AI-on-the-edge architecture.

Table 4: AI-on-the-edge architecture

Architecture/Technique	Description	Advantages	Disadvantages/Trade-offs	Use Cases
Client-Server	Edge devices process lightweight tasks, while complex computations are handled by centralized servers	Efficient for tasks needing centralized processing	Can incur high communication costs and latency	Applications requiring high computational power
Federated Learning	Local devices collaboratively train models, keeping data decentralized for privacy	Enhances privacy, reduces bandwidth use	Increases complexity and may require frequent updates	Healthcare, personalized services, security applications
Pipeline Architecture	Tasks are distributed across devices, edge servers, and the cloud in a sequential flow	Optimizes latency and resource use for real-time applications	May introduce complexity in task allocation	Video Analytics, Industrial automation, smart surveillance
Distributed Architecture	Tasks are distributed across multiple devices without a central server	Improves performance, scalability, and fault tolerance	Complexity in synchronization across devices	IoT, autonomous driving, distributed systems
Hierarchical Architecture	Tasks are distributed based on computational capabilities, with intermediate aggregation	Efficient for large-scale systems, reduces communication overhead	Complexity in managing multiple tiers	Healthcare, smart cities, large-scale industrial systems

Table 5: AI-on-the-edge architecture.

Dimension	On-device	Edge Gateway (Local Server)	Edge cloud (MEC)	Hybrid split	Federated learning
Where inference runs	On the endpoint device	On a local on-prem gateway	At telco/ISP edge	Part local + part remote	Inference: anywhere (often on-device). Training: on devices + central aggregation
Latency	Lowest	Low	Low-medium	Low (if designed well)	Inference depends; training is periodic/slower
Privacy	Best (data stays local)	Very good (stays on site)	Medium (data leaves site)	Variable (depends on what you send)	Very good (shares model updates, not raw data)
Scalability (device count)	High, but fleet updates can be harder	Medium-high (per-site gateways)	High (provider-managed)	Medium	Very high (designed for many clients)
Offline capability	High	Medium-high (LAN)	Low	Medium	Inference can be offline; training needs a connection sometimes
When use	need <50ms response, privacy is strict, or connectivity is unreliable.	Many sensors/cameras feed one site, and you want centralized local compute.	need GPU-class inference but want lower latency than central cloud.	Keep raw data local, but need heavier processing remotely.	The model must learn from user data without collecting it centrally.
examples	smart camera, phone	factory gateway, store server	AR/VR, city-scale analytics	speech/vision with offload	personalization across users

5. Discussion

5.1 Performance Metrics

These systems run on devices with limited resources, so concrete methods for measuring their performance are necessary. There are a few key metrics [3].

Response time is simply the time it takes for the system to process data, from the moment the request arrives to the moment the result is received. **When evaluating compressed models, it is essential to distinguish theoretical computational complexity from measured hardware performance. Parameter reduction, FLOPs reduction, memory reduction, and real-world latency are related but distinct quantities; a model with fewer FLOPs will not necessarily execute faster unless the hardware efficiently maps the remaining computations.**

Data transfer rate is the response time, which is how fast a single task is completed, while throughput is how many tasks you can handle at once. A system may respond quickly to a single request, but crash completely when it receives a hundred requests at the same time. That's why throughput is important. It shows whether a system can scale up when pressure increases, or whether it will collapse under pressure.

Resource utilization has a limited set of CPUs, GPUs, memory, and storage space on these end devices, and the goal is to utilize them to the maximum extent possible without wasting them. It's a delicate balancing. Overstressing the hardware causes it to overheat and fail. Underutilizing it means wasting optimal performance. Getting this right is harder than it sounds, especially when trying to run complex models on a device.

Robustness measures how a system continues functioning when the operating environment deviates from ideal conditions. If the inputs include errors, lighting changes, sensors malfunctioning, or parts that begin to degrade, the performance level may be adversely impacted. A well-designed system will provide accurate results, even under these external pressures. Systems that only work well within a laboratory with perfect conditions will provide little assistance to real-world applications.

When the accuracy of a model is measured as it drifts from reality, what becomes extremely more important than how efficiently or quickly something runs is when it provides accurate results. Accuracy must be maintained whenever an image classifier makes a decision, an anomaly detection system issues an alert, or an automated agent takes action. Slight decreases in accuracy will gradually diminish the user's confidence, and once the user's confidence is diminished, it takes longer to regain the user's confidence than to destroy it.

Security on edge devices can encompass sensitive information such as medical records, surveillance videos and private corporate records. If one device is compromised or if it has been breached, both the contents of that device and the manner in which it functions will be used against the user as the damage resulting from that activity will not be limited to the user of the compromised device; instead, it will have a much wider-reaching negative effect. This is why there must be strong encryption safeguards for protected devices and why staircase authentication for access must be established.

5.2 Distinguishing Theoretical Model Efficiency from Measured Hardware Performance

The evaluation of model compression techniques requires a clear distinction between model-level efficiency indicators and actual deployment performance. Model-level indicators include the total number of parameters, the number of non-zero parameters, sparsity ratio, FLOPs, MACs, and serialized model size. These metrics describe different aspects of the model: parameter count reflects the number of model parameters, sparsity indicates the proportion of zero-valued parameters, FLOPs and MACs estimate computational workload, and serialized model size measures the storage occupied by the exported model representation.

In contrast, hardware-level performance indicators include measured inference latency, throughput, peak memory consumption, and energy consumption. These measurements are affected not only by model architecture and computational complexity but also by hardware capabilities, memory access patterns, accelerator support, inference libraries, compiler optimizations, and runtime overhead. For example, unstructured pruning may substantially increase the sparsity ratio without reducing the operations executed by dense inference kernels. Conversely, structured pruning that physically removes channels or filters may reduce theoretical FLOPs and MACs, but the resulting latency improvement still depends on the target hardware and implementation.

Memory consumption should also be distinguished from model file size, since runtime memory may include model weights, intermediate activations, temporary buffers, and framework overhead. Similarly, power consumption and energy per inference represent different measurements and should not be used interchangeably. To ensure meaningful comparisons, experimental studies should report the target hardware, inference framework, input dimensions, batch size, and measurement conditions, together with the relevant model-level and deployment-level metrics.

The reviewed studies further demonstrate that theoretical compression gains do not always translate proportionally into hardware-level improvements. For example, a reduction in model size or computational complexity may be accompanied by limited improvement in processing time when the resulting architecture or operations are not efficiently supported by the target platform. Similarly, the practical benefit of unstructured sparsity can remain limited when dense execution kernels are used. These observations indicate that compression metrics such as parameter count,

model size, FLOPs, and sparsity should be interpreted as indicators of potential efficiency rather than direct measurements of deployment performance. Hardware-level metrics, including measured latency, throughput, memory consumption, and energy, therefore provide complementary evidence of the actual effectiveness of a compression method on a specific deployment platform.

6. Conclusion

This survey examined model compression techniques for Edge Intelligence and positioned them within the broader Edge AI deployment ecosystem. Pruning, quantization, knowledge distillation, and hybrid compression were identified as the central model-level approaches for reducing resource requirements. Lightweight architectures provide a complementary approach by incorporating efficiency into the network design itself.

The review also highlights that model-level efficiency cannot be considered independently from the deployment platform. Hardware-aware optimization influences how effectively compressed or lightweight models can be executed, while distributed learning, federated learning, and edge-cloud collaboration determine how computation, training, communication, and model coordination are distributed across the system.

A key observation from the reviewed literature is that reductions in parameters, FLOPs, MACs, or model size should not automatically be interpreted as equivalent reductions in inference latency, memory consumption, or energy usage. Practical performance depends on the target hardware, inference framework, supported operators, memory behavior, and deployment configuration.

Future research should therefore focus on hardware-aware and application-aware compression selection, standardized benchmarking across edge platforms, and unified frameworks that jointly consider model efficiency and system-level deployment requirements. Such approaches can provide a more reliable basis for selecting compression strategies according to the requirements of specific Edge AI applications and devices.

Table 1: Summary of Studies on Lightweight Model Optimization for Edge AI

Related Work	Task	Baseline Model	Compression Method	Training Stage	Dataset	Parameters / Model Size	Hardware	Accuracy Metric	FLOPs/ MACs	Latency	Memory	Energy
IN[13]	Image detection/classification	Compact CNN with 8 convolutional and 2 fully connected layers	Lightweight architecture design	NR	CIFAKE, Midjourney, StyleGAN3 dataset.	NR	NR	96.6% accuracy CIFAKE. 99.9% on Midjourney v6. 98.7% on StyleGANv3,	NR	NR	NR	NR
IN[14]	Real-time fire detection using drones	YOLOv8m → YOLOv8n	Knowledge Distillation (YOLOv8m to YOLOv8n)	Student-model training through KD. Exact training duration NR	The dataset comprises 7187 images of various fire incidents, with 3738 and 3449 images for the fire and no fire categories, respectively.	Student has approximately 3.01M parameters .	Raspberry Pi 5B (8GB) integrated with a Pi Camera Module 3 and mounted on a DJI F450 drone.	95.21% accuracy, 98.9% precision, 98% recall.	8.2 GFLOPs	~8 FPS	NR	NR
IN[15]	Edge resource management and task placement	EdgeAISim simulation framework	Integrated advanced AI models such as Multi-Armed Bandit with Upper Confidence Bound(MAB-UCB), Deep Q-Networks (DQN) Deep Q-Networks with Graphical Neural Network(DQN-GNN), and Actor-Critic Network to optimize power usage while efficiently managing task migration within the edge computing environments.	Algorithm training/configuration depends on the selected simulation method	The study relies on simulation-generated data rather than a benchmark dataset.	NR	Simulation-based edge environment (EdgeAISim built on EdgeSimPy); no physical hardware deployment.	Power consumption, resource utilization, and task completion efficiency.	NR	NR	NR	NR
IN[16]	Brain-tumor detection	The authors built their model on the RetinaNet architecture , replacing its heavy ResNet backbone with MobileNet to reduce computation	Lightweight backbone + structured pruning (MobileNet + Pruning)	Training/fine-tuning based	BRATS(brain tumor Segmentation) MRI dataset.	NR	NVIDIA Jetson Nano, Raspberry Pi4	32.1% AP; 0.143 small lesions; 49.7% for large tumors	NR	NR	Lower memory reported	Lower energy reported
IN[17]	Real-time object recognition / power-line inspect	Tiny YOLOv7	Quantization	Quantized deployment	Real-time power-transmission-line inspection (case study)	NR	Raspberry Pi, NVIDIA Jetson Nano, Jetson Orin Nano	Latency, RAM utilization, power consumption	NR	NR	NR	NR

Related Work	Task	Baseline Model	Compression Method	Training Stage	Dataset	Parameters / Model Size	Hardware	Accuracy Metric	FLOPs/MACs	Latency	Memory	Energy
IN[18]	Acute ischemic stroke prediction.	EfficientNet V2-S-based lightweight CNN	Lightweight CNN with attention mechanism; EFused-MBConv and EMBCConv; federated learning	Distributed/federated model training.	Acute Ischemic Stroke MRI (Kaggle)	~155.28 MB	Distributed edge setup: Intel i7-13700F + RTX 4060Ti (central), Intel i5-13400F + RTX 3060Ti (edge)	ccuracy of 96%, a precision of 95%, a recall of 93%, a specificity of 95%, an F1 score of 98%, and an AUC of 97%.	NR	NR	low memory (155.28 MB),	NR
IN[19]	ECG rhythm classification	- ConvLSTM 2D-liquid time-constant (CLTC). - ConvLSTM 2D-closed-form continuous-time (CCfC).	Architecture combines a ConvLSTM2D layer for spatiotemporal feature extraction with a compact Neural Circuit Policy (NCP) layer based on liquid time-constant or closed-form continuous-time neurons	Model training; CLTC reported as more robust to missing channels, while CCfC trains faster	-Telehealth Network of Minas Gerais (TNMG) dataset. -China Physiological Signal Challenge 2018 (CPSC) dataset for generalization.	NR	STM32F746G microcontroller	F1 $\approx$ 0.83, AUROC $\approx$ 0.96;	NR	CCfC faster and slightly more accurate, CLTC better with missing channels.	70.6% RAM and 9.4% flash utilization;	low power (~137 mW);
IN[20]	Respiratory/lung-sound classification	Compared with MobileNetV2, ResNet50, and DenseNet	Custom lightweight CNN	NR	ICBHI 2017 + Fraiwan et al. respiratory sound dataset (1219 samples, 10 classes)	model size ~6.5 MB;	Not specified (assumed edge-capable device)	Accuracy 91.04%; efficient, interpretable, suitable for real-time/edge deployment,	NR	NR	NR	NR
IN[21]	Small-object detection in remote sensing	YOLOv8n enhanced	Small Object Detection Layer. -- SSFF module. -Hierarchical Path Aggregation Network (HPANet).	Modified architecture trained/evaluated;	- VisDrone Dataset - AI-TOD Dataset (Aerial Images-Tiny Object Detection)	-Model size reduced 31.7%; -Reduces the parameter count by 33%	NVIDIA RTX 4090 GPU, PyTorch 2.0, Ubuntu 20.04	mAP@0.5 $\uparrow$ 14.3%, mAP@0.5:0.95 $\uparrow$ 17.1%. improved detection for small objects. slight processing speed trade-off.	NR	NR	NR	NR
IN [9]	Pothole detection, paddy-rice maturity detection, and soybean-weed detection.	large CNN model with 52 million parameters and large size (~603 MB)	Hybrid lightweight model was proposed by combining: Knowledge Distillation, Pruning (50% ratio), and Post-Training Quantization (INT8).	Training time was reduced significantly (3–6 minutes). Student-model training with KD; pruning and PTQ applied.	Three real-world image classification data: Pothole Detection Dataset, Paddy Rice Maturity Detection Dataset, Soybean Weed Detection Dataset.	Model size reduced from 603 MB $\rightarrow$ 3.3 MB.	Framework: TensorFlow. Platform: Google Colab with NVIDIA V100 GPU.	Accuracy reached up to 97%, comparable to the original CNN.	NR	NR	NR	NR

Related Work	Task	Baseline Model	Compression Method	Training Stage	Dataset	Parameters / Model Size	Hardware	Accuracy Metric	FLOPs/ MACs	Latency	Memory	Energy
IN [64]	Hybrid AI for 5G edge/fog/cloud applications	Lightweight CNN + system architecture	Early-exit inference, RL-based task placement, federated learning, trust-score aggregation	Training/inference distributed across system tiers	Evaluated in MATLAB using simulated but realistic inputs, including ECG healthcare signals (normal/arrhythmic), motion/IoT sensor streams with noise, and low-resolution grayscale surveillance images (28×28)	NR	MATLAB R2023a simulation	-Edge accuracy (~84.7%) -Fog accuracy (~89.6%) -Cloud yields the highest accuracy (~93.4%)	NR	Edge achieves the lowest average latency (~8.5 ms) -Fog provides intermediate performance (~22.1 ms) -Cloud the highest latency (~98.3 ms)	NR	-Edge energy (~0.12 J) -Fog energy (~0.34 J) -Cloud energy (~0.57 J),
IN[65]	Mission-critical hybrid AI–edge systems	Hybrid AI–Edge systems emphasizes splitting functionality between cloud and edge	Adaptive orchestration/offloading, federated learning, split computing, secure communication (lightweight cryptography, mentions of homomorphic encryption/SMP C), blockchainbased model update authentication, and edge explainable AI (XAI) for transparency	Training, updates, and inference are distributed between cloud and edge	Simulated scenarios and testbed-style case studies	NR	Hybrid cloud–edge architecture; mixed simulated/testbed evidence	Edge accuracy was slightly lower (example 95.3% cloud vs 93.1% edge)	NR	Latency improved from about 120 ms (cloud-only) to about 7 ms (hybrid)	NR	NR
IN[67]	Image classification	ConvNeXt	Hybrid Approach (Structured Pruning OTOV3 + Dynamic Quantization) produced the most efficient lightweight ConvNeXt model suitable for edge deployment	Training + compression	CIFAR-10 dataset	The best performance is obtained by combining OTOV3 with dynamic quantization , achieving approximately 89.7% reduction in model siz. 95.43% parameter reduction.	-Cloud-based Experiment Setup: Google Colab Pro+ was used to utilize GPU resources (NVIDIA A100 and V100 Tensor Core GPUs). -Edge-based Experiment Setup: A compressed model was deployed on edge with Intel Core i7-1165G7 CPU, 16 GB RAM (Edge deployment using laptop camera)	3.8% accuracy improvement, with successful edge deployment reaching 92.5% accuracy	95% reduction in parameters and MACs,	20 ms	16 GB RAM	NR

References

- [1] P. Mittal, "A comprehensive survey of deep learning-based lightweight object detection models for edge devices," *Artificial Intelligence Review*, vol. 57, art. 242, 2024, doi: <https://doi.org/10.1007/s10462-024-10877-1>.
- [2] H. Joshi, "Edge-AI for Agriculture: Lightweight Vision Models for Disease Detection in Resource-Limited Settings," *arXiv preprint arXiv:2412.18635*, Dec. 2024, doi: <https://doi.org/10.48550/arXiv.2412.18635>.
- [3] V. Shankar, "Edge AI: A Comprehensive Survey of Technologies, Applications, and Challenges," in *Proc. 2024 1st Int. Conf. Advanced Computing and Emerging Technologies (ACET)*, Aug. 2024, doi: <https://doi.org/10.1109/ACET61898.2024.10730112>.
- [4] J. Suo, Z. Xingzhou, S. Zhang, W. Zhou, and W. Shi, "Feasibility analysis of machine learning optimization on GPU-based low-cost edges," in *Proc. 2021 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Internet of People and Smart City Innovation (SmartWorld/UIC/ATC/ScalCom/IoP/SCI)*, 2021, pp. 89–96.
- [5] S. Duan, D. Wang, J. Ren, F. Lyu, Y. Zhang, H. Wu, and X. Shen, "Distributed artificial intelligence empowered by end-edge-cloud computing: A survey," *IEEE Communications Surveys & Tutorials*, vol. 25, pp. 591–624, 2022.
- [6] X. Wang, Z. Tang, J. Guo, T. Meng, C. Wang, T. Wang, and W. Jia, "Empowering Edge Intelligence: A Comprehensive Survey on On-Device AI Models," *ACM Computing Surveys*, vol. 57, no. 9, art. 228, Sep. 2025, doi: <https://doi.org/10.1145/3724420>.
- [7] S. P. Veluru, "AI at the Edge: Transforming Real-Time Data Processing," *International Journal of Emerging Trends in Computer Science and Information Technology*, 2025, doi: <https://doi.org/10.56472/ICCSAIML25-106>.
- [8] T. Wang, J. Guo, B. Zhang, G. Yang, D. Li, "Deploying AI on Edge: Advancement and Challenges in Edge Intelligence," *Mathematics*, vol. 13, no. 11, art. 1878, 2025, doi: <https://doi.org/10.3390/math13111878>.
- [9] A. Musa, H. A. Kakudi, M. Hassan, M. Hamada, U. Umar, and M. L. Salisu, "Lightweight Deep Learning Models for Edge Devices—A Survey," *International Journal of Computer Information Systems and Industrial Management Applications*, Jan. 2025, doi: <https://doi.org/10.70917/2025014>.
- [10] K. Liu, Q. Zheng, K. Tao, and Z. Li, "Low-bit Model Quantization for Deep Neural Networks: A Survey," *arXiv preprint arXiv:2505.05530*, May 2025, doi: <https://doi.org/10.48550/arXiv.2505.05530>.
- [11] S. Park, H. Kim, H. Kim, and J. Choi, "Pruning With Scaled Policy Constraints for Light-Weight Reinforcement Learning," *IEEE Access*, vol. 12, pp. 36055–36065, 2024, doi: <https://doi.org/10.1109/ACCESS.2024.3367002>.
- [12] S. Park, H. Kim, H. Kim, and J. Choi, "Pruning With Scaled Policy Constraints for Light-Weight Reinforcement Learning," *IEEE Access*, vol. 12, pp. 36055–36065, 2024, doi: <https://doi.org/10.1109/ACCESS.2024.3367002>.
- [13] A. L. Ladević, T. Kramberger, R. Kramberger, and D. Vlahek, "Detection of AI-Generated Synthetic Images with a Lightweight CNN," *AI*, vol. 5, pp. 1575–1593, Sep. 2024, doi: <https://doi.org/10.3390/ai5030076>.
- [14] M. F. S. Titu, M. A. Pavel, G. K. O. Michael, H. Babar, U. Aman, and R. Khan, "Real-Time Fire Detection: Integrating Lightweight Deep Learning Models on Drones with Edge Computing," *Drones*, vol. 8, no. 9, art. 483, 2024, doi: <https://doi.org/10.3390/drones8090483>.
- [15] A. R. Nandhakumar, A. Branwal, P. Choudhary, M. Golec, and S. S. Gill, "EdgeAISim: A toolkit for simulation and modelling of AI models in edge computing environments," *Measurement: Sensors*, art. 100939, 2024, doi: <https://doi.org/10.1016/j.measen.2023.100939>.
- [16] A. Abdusalomov, S. Mirzakhililov, S. Umirzakova, A. S. Buriboev, A. Meliboev, B. Muminov, and H. S. Jeon, "Accessible AI Diagnostics and Lightweight Brain Tumor Detection on Medical Edge Devices," *Electronics*, vol. 13, no. 3, art. 529, 2024, doi: <https://doi.org/10.3390/bioengineering12010062>.

- [17] N. Surantha and N. Sutisna, "Key Considerations for Real-Time Object Recognition on Edge Computing Devices," *Applied Sciences*, early access, Jul. 4, 2025. doi: <https://doi.org/10.3390/app15137533>.
- [18] Zhou, F.; Hu, S.; Du, X.; Wan, X.; Wu, J. A, "Lightweight Neural Network Model for Disease Risk Prediction in Edge Intelligent Computing Architecture", *Future Internet* 2024, 16, 75. <https://doi.org/10.3390/fi16030075>.
- [19] Huang Z, Herbozo Contreras LF, Leung WH, Yu L, Truong ND, Nikpour A, Kavehei O. "Efficient Edge-AI Models for Robust ECG Abnormality Detection on Resource-Constrained Hardware". *J Cardiovasc Transl Res.* 2024 Aug;17(4):879-892. doi: <https://doi.org/10.1007/s12265-024-10504-y>.
- [20] T. Wanasinghe, S. Bandara, S. Madusanka, D. Meedeniya, M. Bandara and I. D. L. T. Díez, "Lung Sound Classification With Multi-Feature Integration Utilizing Lightweight CNN Model," in *IEEE Access*, vol. 12, pp. 21262-21276, 2024, doi: <https://doi.org/10.1109/ACCESS.2024.3361943>
- [21] Nie, H.; Pang, H.; Ma, M.; Zheng, R. A Lightweight Remote Sensing Small Target Image Detection Algorithm Based on Improved YOLOv8. *Sensors* 2024, 24, 2952. <https://doi.org/10.3390/s24092952>.
- [22] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A Survey on Mobile Edge Computing: The Communication Perspective," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017, doi: <https://doi.org/10.1109/COMST.2017.2745201>.
- [23] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge Intelligence: Paving the Last Mile of Artificial Intelligence with Edge Computing," *Proc. IEEE*, vol. 107, no. 8, pp. 1738–1762, Aug. 2019, doi: <https://doi.org/10.1109/JPROC.2019.2918951>.
- [24] N. Alqahtani, M. Cheema, and A. Toosi, "Benchmarking Deep Learning Models for Object Detection on Edge Devices," *arXiv preprint arXiv:2409.16808*, 2024. doi: <https://doi.org/10.48550/arXiv.2409.16808>
- [25] R. Sivagami and P. Priyadharshini, "Edge Computing in Embedded Systems," *International Journal of Innovative Research in Science, Engineering and Technology*, vol. 13, no. 12, pp. 45–52, 2024.
- [26] M. Lezcano et al., "Key Considerations for Real-Time Object Recognition on Edge Computing Devices," *Applied Sciences*, vol. 15, no. 13, pp. 7533–7548, 2025. doi: <https://doi.org/10.3390/app15137533>
- [27] J. Kim, S. Chang, and N. Kwak, "PQK: Model Compression via Pruning, Quantization, and Knowledge Distillation", *arXiv preprint*, 2021. doi: <https://doi.org/10.48550/arXiv.2106.14681>.
- [28] L. Santos and F. Duarte, "Data Privacy in Edge-AI and Federated Learning," *Journal of Information Systems Engineering and Management*, vol. 9, no. 4, pp. 230–241, 2024.
- [29] R. Zeinaty et al., "Designing Object Detection Models for TinyML: Foundations, Comparative Analysis, Challenges, and Emerging Solutions," *arXiv preprint arXiv:2508.08352*, 2025. doi: <https://doi.org/10.48550/arXiv.2508.08352>.
- [30] J. Wu, Z. Lin, P. Liu, and H. Zhao, "ED-TOOLBOX: Efficient Detection Framework Adaptation for Edge Computing," *arXiv preprint arXiv:2412.18230*, 2024. doi: <https://doi.org/10.48550/arXiv.2412.18230>.
- [31] P. Wang et al., "BED: A Real-Time Object Detection System for Edge Devices," *arXiv preprint arXiv:2202.07503*, 2022. doi: <https://doi.org/10.48550/arXiv.2202.07503>.
- [32] Gartner, "Infrastructure Trends Shaping the Future of Cloud, Data Center and Edge," Gartner Press Release, 2023.
- [33] B. A. Motetti, M. Risso, A. Burrello, E. Macii, M. Poncino and D. J. Pagliari, "Joint Pruning and Channel-Wise Mixed-Precision Quantization for Efficient Deep Neural Networks," *IEEE Trans. Computers*, vol. 73, no. 11, pp. 1-14, Nov. 2024.
- [34] E. Fazeldehkordi, M. Bagheri, A. Mansouri and M. Jalili, "A Survey of Security Architectures for Edge Computing-Based IoT," *J. Cybersecurity Inf. Syst.*, vol. 3, no. 3, pp. 19-36, 2022

- [35] Z. Yang, M. Chen, W. Saad, C. S. Hong and M. Shikh-Bahaei, "Energy Efficient Federated Learning Over Wireless Communication Networks," *IEEE Trans. Wireless Commun.*, vol. 20, no. 2, pp. 1387-1400, Feb. 2021.
- [36] K. Yan, Y. C. Chiu, M. Li and S. Guo, "A Survey of Energy-Efficient Strategies for Federated Learning in Mobile Edge Computing," *IEEE Access*, vol. 12, pp. 1-24, 2024.
- [37] R. David, J. Duke, A. Jain, V. Janapa Reddi, N. Jeffries, J. Li, N. Kreeger, I. Nappier, M. Natraj, T. Wang, P. Warden, R. Rhodes, "TensorFlow Lite Micro: Embedded Machine Learning on TinyML Systems", in *Proc. Machine Learning and Systems (MLSys)*, vol. 3, 2021, pp. 800–811.
- [38] W. -F. Lin et al., "ONNC: A Compilation Framework Connecting ONNX to Proprietary Deep Learning Accelerators", 2019 *IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, Hsinchu, Taiwan, 2019, pp. 214-218, doi: <https://doi.org/10.1109/AICAS.2019.8771510>.
- [39] B. Liu, Z. Luo, H. Chen, and C. Li, "A survey of state-of-the-art on edge computing: Theoretical models, technologies, directions, and development paths," *IEEE Access*, vol. 10, pp. 54038–54063, 2022 doi: <https://doi.org/10.1109/ACCESS.2022.3176106>
- [40] H. Li, R. Sahu, M. Zaheer, S. Talwalkar, V. Smith, "Federated Learning: Challenges, Methods, and Future Directions," *IEEE Signal Processing Magazine*, vol. 37, no. 3, pp. 50–60, 2020.
- [41] F. Ilhan, S. F. Tekin, S. Hu, T. Huang, K. H. Liu, L. Liu, "Hierarchical Deep Neural Network Inference for Device-Edge-Cloud Systems," in *Proc. ACM Web Conference (WWW '23)*, 2023, pp. 302–305, doi: <https://doi.org/10.1145/3543873.3587370>.
- [42] M. S. Ansari, Y. Qiao, Y. Ye, and B. Lee, "Security of Distributed Intelligence in Edge Computing: Threats and Countermeasures," in *Cloud-to-Thing Continuum*, Springer, 2020, doi: https://doi.org/10.1007/978-3-030-41110-7_6.
- [43] L. Liu, J. Zhang, S. H. Song, K. Letaief, "Client–Edge–Cloud Hierarchical Federated Learning," *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, Dublin, Ireland, 2020, pp. 1-6, doi: <https://doi.org/10.1109/ICC40277.2020.9148862>.
- [44] A. Musa et al., "Lightweight Deep Learning Models For Edge Devices — A Survey," *Int. J. Comput. Inf. Syst. Ind. Manag. Appl.*, vol. 17, no. 18, 2025. doi: <https://doi.org/10.70917/2025014>
- [45] J. Liu, "A Review of Model Lightweighting Techniques Based on Edge Computing," *Academic Journal of Science and Technology*, Jan. 2025. doi: <https://doi.org/10.54097/b8xdxw07>
- [46] T. Suwannaphong, F. Jovan, I. Craddock, et al., "Optimising TinyML with quantization and distillation of transformer and mamba models for indoor localisation on edge devices", *Scientific reports*, vol. 15, no. 1, p. 7342, 2025. doi: <https://doi.org/10.1038/s41598-025-94205-9>
- [47] R. Zhang, H. Jiang, W. Wang, and J. Liu, "Optimization Methods, Challenges, and Opportunities for Edge Inference: A Comprehensive Survey", *Electronics*, vol. 14, no. 7, pp. 1345-1359, 2025. doi: <https://doi.org/10.3390/electronics14071345>.
- [48] D. R. Rakandhiya, A. N. Labiib Nabhaan, and A. Setyanto, "Deep Learning Model Compression Techniques Performance on Edge Devices", *Matrik Jurnal Manajemen Teknik Informatika dan Rekayasa Komputer*, vol. 23, no. 3, pp. 3961-3972, 2024. doi: <https://doi.org/10.30812/matrik.v23i3.3961>
- [49] X. Zhang, L. Zhang, M. Zhang, Z. Zhang, P. Li, X. Shi, and F. Zhou, "MLD-Net: A Multi-Level Knowledge Distillation Network for Automatic Modulation Recognition," *Sensors*, vol. 25, no. 23, p. 7143, 2025. doi: <https://doi.org/10.3390/s25237143>.
- [50] C. T. Sajan, H. M. Sunny, and A. Pratap, "Federated learning in edge AI: a systematic review of applications, privacy challenges, and preservation techniques", *Indonesian Journal of Electrical Engineering and Computer Science*, Nov. 2025. doi: <https://doi.org/10.11591/ijeecs.v40.i2.pp926-940>.
- [51] Yu, T., Zhang, C., Ma, M. et al. , "Recursive least squares method for training and pruning convolutional neural networks". *Applied Intelligence*, (2023). doi: <https://doi.org/10.1007/s10489-023-04740-z>

- [52] Wang, R., Wan, S., Zhang, W. et al., "Progressive multi-level distillation learning for pruning network". *Complex & Intelligent Systems*. (2023). doi: <https://doi.org/10.1007/s40747-023-01036-0>
- [53] Y. He and L. Xiao, "Structured Pruning for Deep Convolutional Neural Networks: A Survey", *IEEE Transactions on Pattern Analysis & Machine Intelligence*, May 2024, doi: <https://doi.org/10.1109/TPAMI.2023.3334614>.
- [54] V. Chowdary, "Scalable AI Pipelines in Edge-Cloud Environments: Challenges and Solutions for Big Data Processing," *International Journal of Engineering and Computer Science*, Oct. 2023, doi: <https://doi.org/10.18535/ijecs.v12i10.4683>.
- [55] B. Liu, B.-B. Hu, M. Zhao, S.-L. Peng and J.-M. Chang, "Model Compression Algorithm via Reinforcement Learning and Knowledge Distillation", *Mathematics* 2023. doi: <https://doi.org/10.3390/math11224589>.
- [56] D. Ngo, H.-C. Park and B. Kang, "Edge Intelligence: A Review of Deep Neural Network Inference in Resource-Limited Environments", *Electronics* 2025. doi: <https://doi.org/10.3390/electronics14122495>.
- [57] M. Abou Ali and F. Dornaika, "Edge Artificial Intelligence: A Systematic Review of Evolution, Taxonomic Frameworks, and Future Horizons," *arXiv*, 2025. doi: <https://arxiv.org/abs/2510.01439>.
- [58] P. Sun, "Security and privacy protection in cloud computing: Discussions and challenges," *Journal of Network and Computer Applications*, Apr. 2020, doi: <https://doi.org/10.1016/j.jnca.2020.102642>.
- [59] M. Abou Ali and F. Dornaika, "Edge Artificial Intelligence: A Systematic Review of Evolution, Taxonomic Frameworks, and Future Horizons," *arXiv*, 2025. doi: <https://doi.org/10.48550/arXiv.2510.01439>
- [60] V. C. Manduva, "AI-Driven Edge Computing in the Cloud Era: Challenges and Opportunities," *International Journal of Scientific Research and Management (IJSRM)*, 2023, doi: <https://doi.org/10.18535/ijssrm/v11i02.ec5>.
- [61] S. Duan et al., "Distributed artificial intelligence empowered by end-edge-cloud computing: A survey," *EEE Communications Surveys & Tutorials*, pp. 591-624, 2022, doi: <https://doi.org/10.1109/COMST.2022.3141790>.
- [62] H. Gauttam, G. Nain, K. K. Pattanaik, and P. Mendes, "Edge-AI: A systematic review on architectures, applications, and challenges", *Journal of Network and Computer Applications*, vol. 245, p. 104375, Jan. 2026, doi: <https://doi.org/10.1016/j.jnca.2025.104375>.
- [63] P. Vilar Dantas, W. Sabino da Silva Jr, L. C. Cordeiro, and C. B. Carvalho, "A comprehensive review of model compression techniques in machine learning," *Applied Intelligence*, vol. 54, pp. 11804–11844, Aug. 2024, doi: <https://doi.org/10.1007/s10489-024-05747-w>.
- [64] Amit, Amandeep, and Khushi, "Hybrid AI Modal for Edge Computing in 5G," *International Journal for Research in Applied Science & Engineering Technology (IJRASET)*, vol. 13, no. 6, pp. 2655–2659, Jun. 2025, ISSN: 2321-9653.
- [65] S. K. Rachapalli, "Hybrid AI-Edge architectures for mission-critical decision systems," *International Journal on Science and Technology (IJSAT)*, vol. 14, no. 4, pp. 1–12, Oct.–Dec. 2023.
- [66] Fraiwan, Mohammad; Fraiwan, Luay; Khassawneh, Basheer; Ibnian, Ali (2021), "A dataset of lung sounds recorded from the chest wall using an electronic stethoscope", *Mendeley Data*, V3, doi: <https://doi.org/10.17632/jwyy9np4gv.3>
- [67] S. Francy and R. Singh, "Edge AI: Evaluation of Model Compression Techniques for Convolutional Neural Networks," *arXiv*, Sep. 2024.
- [68] Y. Xu, T. M. Khan, Y. Song, et al., "Edge deep learning in computer vision and medical diagnostics: A comprehensive survey," *Artificial Intelligence Review*, vol. 58, art. no. 93, 2025, doi: <https://doi.org/10.1007/s10462-024-11033-5>.
- [69] G. Arjunan, "Optimizing Edge AI for real-time data processing in IoT devices: Challenges and solutions," *International Journal of Scientific Research and Management (IJSRM)*, vol. 11, no. 6, pp. 944–953, 2023, doi: <https://doi.org/10.18535/ijssrm/v11i06.ec2>.