





Review

Embedded Artificial Intelligence: A Comprehensive Literature Review

Xiaoyuan Huang ^{1,2,*} , Hongcheng Wang ² , Shiyin Qin ^{2,3}  and Su-Kit Tang ^{1,*} ¹ Faculty of Applied Sciences, Macao Polytechnic University, Macao SAR 999078, China² School of Electrical Engineering and Intelligentization, Dongguan University of Technology, Dongguan 523808, China; wanghc@dgut.edu.cn (H.W.); qinsy@dgut.edu.cn (S.Q.)³ School of Automation Science and Electrical Engineering, Beijing University of Aeronautics and Astronautics, Beijing 100191, China

* Correspondence: xiaoyuan.huang@mpu.edu.mo (X.H.); sktang@mpu.edu.mo (S.-K.T.)

Abstract

Embedded Artificial Intelligence (EAI) integrates AI technologies with resource-constrained embedded systems, overcoming the limitations of cloud AI in aspects such as latency and energy consumption, thereby empowering edge devices with autonomous decision-making and real-time intelligence. This review provides a comprehensive overview of this rapidly evolving field, systematically covering its definition, hardware platforms, software frameworks and tools, core algorithms (including lightweight models), and detailed deployment processes. It also discusses its widespread applications in key areas like autonomous driving and smart Internet of Things (IoT), as well as emerging directions. By analyzing its core challenges and innovative opportunities in algorithms, hardware, and frameworks, this review aims to provide relevant researchers and developers with a practical guidance framework, promoting technological innovation and adoption.

Keywords: EAI; deep learning; resource constraints; hardware platforms; software frameworks; model optimization; deployment



Academic Editor: Cecilio Angulo

Received: 23 July 2025

Revised: 21 August 2025

Accepted: 27 August 2025

Published: 29 August 2025

Citation: Huang, X.; Wang, H.; Qin, S.; Tang, S.-K. Embedded Artificial Intelligence: A Comprehensive Literature Review. *Electronics* **2025**, *14*, 3468. <https://doi.org/10.3390/electronics14173468>

Copyright: © 2025 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

The transformative potential of Artificial Intelligence (AI) is undeniable, with its influence permeating diverse sectors ranging from healthcare and finance to transportation and entertainment [1–3]. Traditionally, AI applications have predominantly resided on cloud-based infrastructures or High-Performance Computing (HPC) environments [4,5], leveraging vast computational resources and ample memory to execute complex algorithms and process massive datasets. This type of cloud-based AI excels in scenarios requiring massive data aggregation and complex, large-scale model training. However, this centralized approach is often limited by latency, bandwidth constraints, data privacy, and energy consumption, hindering its application in time-critical and resource-sensitive scenarios [6–8]. For an autonomous vehicle detecting pedestrians, the time it takes to send critical decision-making data back and forth to a cloud server is simply unacceptable. In contrast, embedded AI (EAI) places intelligence directly at the point of data collection. This paradigm shift from centralized to distributed processing is not just a choice, but an inevitable trend driven by the growing demand for real-time processing, low latency, enhanced privacy, and optimized energy efficiency, giving rise to the emergence of EAI [9].

The development of EAI can be divided into three key eras, as illustrated in Figure 1. The core paradigm of the embryonic stage (pre-2016) was the exploration of miniaturization

from cloud AI, marked by early compression algorithms such as SqueezeNet. The mobile AI explosion period (2017–2019) saw the maturity of the MobileNet series and TensorFlow Lite (TFLite), which enabled a large-scale shift of AI's focus from the cloud to the edge, making real-time inference on-device mainstream. The TinyML era (2020–present) has seen the ultimate downscaling of AI capabilities to microcontrollers, driven by technologies such as AI-accelerated microcontrollers and lightweight Transformers, giving rise to always-on, microwatt-level intelligence [10,11]. In the future, EAI is heading towards the next paradigm shift, from static inference to dynamic, adaptive learning [12–14].

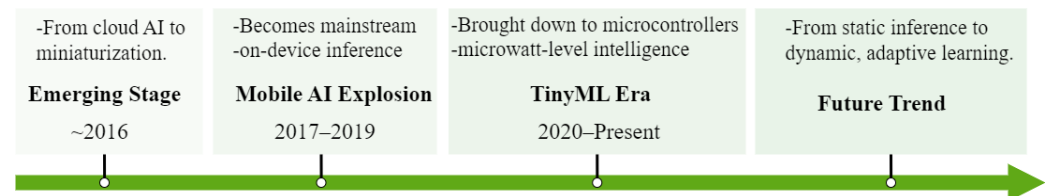


Figure 1. The development timeline of EAI.

EAI transfers the locus of AI computation from the cloud to the edge, embedding intelligent capabilities directly into resource-constrained devices. EAI empowers edge devices to make autonomous decisions, adapt to dynamic environments, and interact seamlessly with the physical world, all without a constant reliance on network connectivity [15–17]. This transition has unlocked possibilities for a plethora of novel applications, including autonomous vehicles, smart sensors, wearable devices, industrial automation systems, and personalized healthcare solutions [18–20]. The challenges entailed in designing and deploying efficient and robust EAI systems necessitate a holistic approach that considers hardware limitations, algorithmic complexity, and application-specific requirements.

Comprehensive surveys on the swiftly progressing EAI technology have also been conducted by researchers in recent years. Kodgire et al. [21] explored the key trade-offs in EAI, especially the balance between computing efficiency and energy consumption. The article focuses on the application of EAI in different fields and discusses the core challenges it faces. It is a study that focuses on the core contradictions and application breadth of EAI. Elkhaliq [22] is an application-driven review that focuses on the specific implementation of AI in the field of smart homes. It deeply explores the opportunities and challenges in this specific scenario, such as resource limitations and data security. The core of Peccia et al.'s [23] systematic review is embedded distributed reasoning of deep neural networks. It focuses on how to decompose and deploy complex reasoning tasks to multiple embedded devices, so its discussion focuses on related algorithms, deployment strategies, and challenges. Zhang et al. [24] conducted a relatively comprehensive review of artificial intelligence technology in embedded systems. The article focuses on the hardware platform and core algorithms of EAI, and deeply analyzes key challenges such as resource limitations and data security, but pays less attention to the software ecosystem and deployment process. Kotyal et al. [25] reviewed the progress and challenges of AI applications from a broader perspective. They discussed the implementation of AI in various industries in a broad sense, but their core focus was not entirely on the specific constraints and technology stacks of embedded environments. Serpanos et al. [26] outline a future roadmap for embedded AI from a macro perspective, with an emphasis on defining its scope, elucidating its vast potential in key application domains, and identifying the core challenges and opportunities that must be addressed to realize this vision, thereby providing guidance for future research and policymaking in Europe.

This survey significantly surpasses contemporaneous related works [21–26] in terms of comprehensiveness and systematic coverage, with a comparison of the core thematic scope presented in Table 1. First, at the theoretical level, it introduces an innovative

mathematical formalization, thereby establishing a rigorous theoretical foundation for EAI—an aspect absent in prior works that primarily relied on descriptive definitions. Second, at the technology stack level, it is the only survey that systematically covers the entire ecosystem from hardware platforms to software frameworks, complemented with detailed comparative tables that provide developers with a valuable “hardware-to-software” selection guide, whereas other reviews either focus narrowly on hardware or neglect the software ecosystem. Furthermore, in terms of core algorithmic analysis, this work not only covers conventional and classical models but also systematically presents the latest lightweight neural network architectures tailored for embedded environments, thereby demonstrating its cutting-edge orientation. Most notably, through clear deployment flowcharts and step-by-step breakdowns, it offers a thorough and practical exposition of “how to implement EAI,” providing an actionable end-to-end guideline that distinguishes it from prior works, which tend to remain at an abstract level of discussion. In summary, by virtue of its comprehensive knowledge chain, in-depth technical analysis, and strong practical guidance, this survey endeavors to serve as a highly valuable “one-stop” reference for both academic research and engineering practice in the field.

Table 1. Comparison of the core thematic scope of EAI comprehensive survey.

Core Thematic	Ours	Kodgire et al. [21]	Elkhalik [22]	Peccia et al. [23]	Zhang and Li [24]	Kotyal et al. [25]	Serpanos et al. [26]
Definition	✓			✓		✓	
Hardware Platforms	✓				✓		
Software Frameworks and Tools	✓						
Algorithms (including lightweight models)	✓			✓	✓	✓	
Deployment	✓			✓			
Application	✓	✓	✓		✓	✓	✓
Challenges and Opportunities	✓	✓	✓	✓	✓	✓	✓

The paper also integrates insights from multiple domains such as autonomous driving and smart homes, which are highlighted as key application areas for EAI [22,23]. Furthermore, the review proposes practical solutions for core challenges like resource constraints and data security—critical issues identified in the literature on EAI and its applications [22,24]. By synthesizing these diverse perspectives, this review not only enhances the analytical depth of the field but also provides a more detailed account of the opportunities within the EAI landscape, thereby offering readers a comprehensive guide to navigate this rapidly evolving domain [24–28].

The remainder of this paper is organized as follows: Section 2 systematically introduces the formal definition of EAI from both theoretical and mathematical perspectives, clarifying its core optimization objectives. Section 3 provides an in-depth exploration of the diverse hardware platforms for EAI, ranging from microcontrollers to specialized AI chips, and presents a practical strategy for their selection. Section 4 comprehensively reviews the software frameworks and toolchains that support EAI development, offering a valuable reference for developers in choosing the appropriate technology stack. Section 5 details the core algorithms applicable to EAI, not only reviewing traditional machine learning and classic deep learning models but also placing a special emphasis on state-of-the-art lightweight neural network architectures. Section 6, illustrated with a flowchart, breaks down the end-to-end deployment process—from model optimization to hardware

integration—revealing the critical stages of transforming an algorithm into a product. Section 7 vividly showcases the widespread applications of EAI in fields such as autonomous driving and the Internet of Things through a rich set of case studies. Section 8 provides a profound analysis of the core challenges currently facing EAI and looks ahead to its future opportunities in algorithms, hardware, and applications. Finally, Section 9 concludes the paper.

2. Definition of EAI

EAI can be defined as the deployment and execution of AI algorithms on embedded systems [16–18]. Embedded systems are specialized computer systems meticulously designed for specific tasks. They are typically characterized by resource constraints, including limited processing power, finite memory capacity, and stringent energy-consumption budgets [29,30]. This definition necessitates achieving high AI performance within the operational constraints of the embedded environment, which requires a delicate balance.

Mathematically, we can formulate the objective of EAI as optimizing the performance of an AI model under resource constraints. Let:

$$\begin{aligned} & \underset{Model}{\text{maximize}} \quad Performance(Model) \\ & \text{subject to} \quad Resources(Model) \leq Hardware Limits \end{aligned} \quad (1)$$

where the objective function $Performance(Model)$ represents the model performance metric to be maximized (e.g., accuracy, F1-score, etc.); and the constraint $Resources(Model) \leq Hardware Limits$ requires that all resource consumption of the model (e.g., computational complexity, memory, energy consumption) does not exceed the limitations of the hardware platform. This formulation transforms an abstract design philosophy into a measurable and executable constrained optimization problem. This mathematical framework not only reveals the inherent trade-off between performance and resources that is unavoidable in the EAI domain but also provides a unified theoretical guide and evaluation standard for all optimization techniques, such as model compression and hardware-aware Neural Architecture Search (NAS).

The core distinction from traditional AI optimization lies in a fundamental shift in mindset: (1) Traditional AI optimization pursues unconstrained performance maximization. Its central question is, “How powerful a model can we build?” where resources are typically considered scalable; (2) EAI optimization, in contrast, seeks a Pareto-optimal solution within a fixed, insurmountable hardware “box.” Its central question is, “What is the best performance we can achieve under the given strict constraints?”

Specifically, the resources consumed by a model can be represented as a vector, such as:

$$Resources(Model) = [F_{Model}, M_{Model}, E_{Model}], \quad (2)$$

where F_{Model} denotes the model’s computational complexity (e.g., measured in floating point number operations, FLOPs), M_{Model} represents its memory footprint (e.g., in megabytes), and E_{Model} is its energy consumption per inference.

Correspondingly, the hardware platform imposes maximum allowable values for these resources, which can be denoted as

$$F_{Model} \leq F_{Limits}, \quad M_{Model} \leq M_{Limits}, \quad E_{Model} \leq E_{Limits} \quad (3)$$

where F_{Limits} , M_{Limits} , E_{Limits} are the maximum allowable values imposed by the hardware on computational complexity (FLOPs), memory, and energy consumption, respectively. If

it is necessary to simultaneously optimize performance and resources (e.g., Pareto optimization), the problem can be extended to a multi-objective form:

$$\underset{Model}{optimize} \quad [Performance(Model), -Resources(Model)] \quad (4)$$

In this case, it is necessary to strike a balance between performance improvement and resource reduction in order to seek Pareto-optimal solutions. For example, a common small-scale case study involves the joint optimization of pruning and quantization for a specific neural network on a target embedded device. By exploring different pruning rates (e.g., removing 30%, 50%, or 70% of redundant weights) and quantization strategies (e.g., 8-bit or 4-bit integer quantization), a series of models with varying levels of accuracy and resource costs (such as model size or inference latency) can be generated. When these models are plotted on a two-dimensional graph with accuracy on the Y-axis and model size on the X-axis, the Pareto frontier emerges as a curve consisting of optimal models. Any model on this curve is considered “Pareto-optimal,” meaning that no other configuration can achieve higher accuracy with the same or smaller model size. Developers can then select a model point along this frontier that maximizes performance under the strict resource constraints of their application. This principle underpins many automated model compression and hardware-aware NAS frameworks [31].

This framework provides a mathematical foundation for model compression (e.g., pruning, quantization), NAS, and deployment in EAI. This optimization problem highlights the core challenge of EAI: maximizing AI performance while adhering to the inherent resource limitations of embedded systems. Unlike traditional AI systems residing in cloud servers or powerful workstations [4,5], EAI operates directly on the target device, enabling real-time inference and decision-making. Its goal is to achieve a synergy between AI capabilities and hardware constraints, ensuring that the deployed algorithms are efficient, accurate, and robust within the operational environment [16,17].

3. Hardware Platforms of EAI

To solve the core optimization problem articulated in the preceding section—namely, maximizing AI performance subject to stringent hardware limitations—it is imperative to first gain a deep understanding of the physical basis for these constraints: the embedded hardware platforms. Spanning a wide spectrum from low-power microcontrollers to high-performance AI Systems-on-Chip (SoC), the choice of platform necessitates a meticulous trade-off among computational power, power consumption, cost, and the complexity of the development ecosystem. This section systematically reviews the major EAI hardware platforms and puts forward a practical strategy for their selection [32].

3.1. Major Hardware Platforms of EAI

Current mainstream EAI hardware platforms are categorized into five types based on their chip architecture: Microcontroller Unit (MCU), Micro Processor Unit (MPU), Graphics Processing Unit (GPU), Field-Programmable Gate Array (FPGA), Application-Specific Integrated Circuit (ASIC), AI System-on-Chip (SoC).

Tables 2 and 3 lists mainstream EAI hardware platforms and compares them across multiple dimensions to facilitate assessment. These key comparison dimensions include: specific chip models, manufacturers, architectural, the core processor type, AI accelerators, the achievable computational power, as well as runtime power consumption and other key features. These factors collectively determine a specific hardware platform’s suitability and competitiveness in specific EAI applications.

3.1.1. MCU

MCUs are distinguished by their extremely low power consumption, low cost, small form factor, and robust real-time hardware characteristics [33]. They typically integrate Central Processing Unit (CPU) (such as the ARM Cortex-M series [34]), limited Random Access Memory (RAM), and Flash memory. In terms of AI capabilities, the constrained hardware resources of MCUs render them suitable for executing lightweight AI models, such as simple keyword spotting, sensor data anomaly detection, and basic gesture recognition [35,36]. Their hardware architecture is often optimized to support these AI applications, and many contemporary MCUs integrate hardware features conducive to AI computation [33,37]. Typical hardware representatives in the MCU domain include: STM32F series (STMicroelectronics, Geneva, Switzerland), which, with its extensive product line and integrated hardware resources, can execute optimized neural network models [38]; i.MX RT series (NXP Semiconductors N.V., Eindhoven, Netherlands), which are crossover MCUs combining the real-time control characteristics of traditional MCUs with the application-processing capabilities approaching those of MPUs, thereby providing a hardware foundation for more demanding EAI tasks [39]; ESP32 series (Espressif Systems, Shanghai, China), a low-cost MCU family with integrated Wi-Fi and Bluetooth functionality, whose hardware design also incorporates considerations for accelerated support for digital signal processing and basic neural network operations, often leveraged by libraries such as ESP-DL [40].

3.1.2. MPU

MPUs are central to general-purpose computing in embedded systems, typically incorporating CPU cores based on architectures such as ARM [41]. These cores excel at executing complex logical control and sequential tasks and are increasingly undertaking computational responsibilities for edge AI [33,42]. ARM-based MPUs are well-suited for scenarios where AI tasks coexist with substantial non-AI-related computations, or where stringent real-time AI requirements are not paramount, such as in smart home control centers and low-power edge computing nodes [42]. Modern high-performance MPUs featuring ARM CPU cores (e.g., STM32-MP13x series [43]) often employ multi-core architectures and widely support the ARM NEON™ Single Instruction Multiple Datastream (SIMD) instruction set architecture extension [38]. This NEON technology can significantly accelerate common vector and matrix operations prevalent in AI algorithms, thereby enhancing the computational efficiency of certain AI models [33,44]. In many resource-constrained embedded systems, developers often directly leverage the existing computational resources of the MPU's CPU cores, augmented by NEON capabilities, to efficiently execute lightweight AI models [35]. For instance, in many ARM-based MPUs or SoCs, including the MT7986A with Cortex-A53 processor (MediaTek Inc., Hsinchu City, Taiwan) [45] and Raspberry Pi 5 with Cortex-A76 processor (Raspberry Pi Foundation, Cambridge, United Kingdom) [46], the CPU cores handle the primary computational tasks. Even without dedicated AI accelerators, integrated technologies like NEON often provide sufficient performance for a variety of lightweight EAI applications [44].

Table 2. Mainstream MCP/MPU hardware platforms of EAI.

Platforms Models ¹	Manu-Facturer	Arch-Itecture	Processor	AI Accelerator	Computational Power	Power tion ²	Consump-	Other Key Features
STM32F [38]	STMicroelectronics, Geneva, Switzerland	MCU	Cortex-M0/M3/M4/M7/M33 etc.	ART Accelerator in some models	Tens to hundreds of DMIPS ³ , several GOPS ⁴ in some models (with accelerator)	Low power (mA-level op., µA-level standby)		Rich peripherals, CubeMX ecosystem, Security features (integrated in some models)

Table 2. Cont.

Platforms Models ¹	Manu-Facturer	Arch-Itecture	Processor	AI Accelerator	Computational Power	Power Consump-tion ²	Other Key Features
i.MX RT [39]	NXP Semiconductors N.V., Eindhoven, Netherlands	MCU	Cortex-M7, Cortex-M33	2D GPU (PXP), NPU (e.g., RT106F, RT117F) in some models	Hundreds to thousands of DMIPS, NPU can reach 1–2 TOPS	Medium power, performance-oriented	High-performance real-time processing, interface control, EdgeLock security subsystem
ESP32-S3 [40]	Espressif Systems (Shanghai) Co., Ltd., Shanghai, China.	MCU	Xtensa LX6/LX7 (dual-core), RISC-V	Supports AI Vector Instructions	CPU: Up to 960 DMIPS; AI: Approx. 0.3–0.4 TOPS (8-bit integer, INT8)	Low power (higher when Wi-Fi/BT active)	Integrated Wi-Fi and Bluetooth, Open-source SDK (ESP-IDF), Active community, High cost-performance
STM32-MP13x [41–44]	STMicroelectronics	MPU	ARM Cortex-A7, Cortex-M4	No dedicated GPU; AI runs via CPU/CMSIS-NN	A7: Max ~2000 DMIPS; M4: Max ~200 DMIPS	<1 W	Entry-level Linux or RTOS system design
MT7986A (Filogic 830) [45]	MediaTek Inc., Hsinchu City, Taiwan.	MPU	ARM Cortex-A53	Hardware Network Acceleration Engine, AI Engine (approx. 0.5 TOPS)	CPU: Up to ~16,000 DMIPS	~7.5 W	High-speed network interface
Raspberry Pi 5 [46]	Raspberry Pi Ltd, Cambridge, United Kingdom.	MPU	Broadcom BCM2712 (quad-core ARM Cortex-A76)	VideoCore VII GPU	~60–80 GFLOPS ⁵	Passive Cooling; 2.55 W (Idle); 6.66 W (Stress) ⁶	Graphics performance and general-purpose computing capability

¹ Name or series of the chip/module. ² Typical or rated power consumption. Actual power can vary significantly based on workload and configuration. ³ DMIPS: Dhrystone Million Instructions executed Per Second. ⁴ GOPS: Giga Operations Per Second. A unit of measure for the computing power of a processor. TOPS, as used in the following text, stands for Tera Operations Per Second. ⁵ GFLOPS: Giga FLOPS. FLOPS stands for Floating-point Operations Per Second and is a common performance parameter for GPUs. TFOPS, as used in the following text, stands for Tera Floating-point Operations Per Second. ⁶ Raspberry Pi 5 Power: The “Passive Cooling” indicates it can run without a fan for some tasks. The power figures 2.55 W (Idle) and 6.66 W (Stress) are specific measurements. Active cooling is often recommended for sustained heavy loads.

3.1.3. GPU

GPUs possess massively parallel processing capabilities, excelling at handling matrix operations, and are suitable for accelerating deep learning algorithms [33,47]. They are primarily used in AI applications requiring high-performance computing and parallel processing, such as autonomous driving, video analytics, and advanced robotics [48,49]. Mainstream GPU vendors include NVIDIA, AMD, and ARM (Mali). NVIDIA’s GPUs are widely used in embedded systems, such as the Jetson series (NVIDIA, Santa Clara, CA, USA) [50,51]; their CUDA and TensorRT toolchains facilitate the development and deployment of AI applications, widely used in robotics, drones, and other fields [52]. Through programming frameworks like CUDA or OpenCL, developers can conveniently leverage GPUs to accelerate AI model training and inference [33,53]. Mobile SoCs like Qualcomm’s Snapdragon series (featuring Adreno GPUs) (Qualcomm, San Diego, CA, USA) [54] and MediaTek’s Dimensity series (often featuring ARM Mali or custom GPUs) [55] also integrate powerful GPUs to accelerate AI applications on smartphones.

3.1.4. FPGA

The FPGA is a programmable hardware platform where hardware accelerators can be customized as needed, making it suitable for scenarios requiring customized hardware acceleration and strict latency requirements, such as high-speed data acquisition, real-time image processing, and communication systems [56,57]. It offers a high degree of flexibility and reconfigurability, enabling optimized data paths and parallel computation. Products such as the Zynq 7000 SoC family (AMD, Santa Clara, CA, USA) [58] and the Arria 10 SoC FPGAs (Intel, Santa Clara, CA, USA) [59] integrate FPGA’s programmable logic with hardware processors, such as Cortex-A9 (Arm, Cambridge, United Kingdom), on the same chip. This makes them suitable for applications demanding both high-performance processing

from the CPU and flexible, high-throughput hardware-acceleration capabilities from the FPGA fabric.

3.1.5. ASIC

ASICs are specialized chips meticulously designed for particular applications, offering the highest performance and lowest power consumption for their designated tasks [33,60]. The hardware logic of an ASIC is fixed during the manufacturing process, and its functionality cannot be modified once produced. Consequently, the design and manufacturing (Non-Recurring Engineering, NRE) costs for ASICs are typically high, and they offer limited flexibility compared to programmable solutions [60]. Google's Tensor Processing Unit (TPU), such as Tensor G3 (Google LLC, Mountain View, CA, USA) is a prime example of an ASIC, specifically engineered to accelerate the training and inference of machine learning (ML) models, particularly within the TensorFlow framework [61]. Atlas series AI processors (Huawei, Shenzhen, China), which employ the Da Vinci architecture, also fall into the ASIC category, designed for a range of AI workloads [62].

3.1.6. AI SoC

The AI SoC refers to a system-on-chip that integrates hardware-acceleration units specifically designed for AI computation, such as Neural Processing Unit (NPU) or, as sometimes termed by vendors, Knowledge Processing Unit (KPU) [33,63]. These integrated AI-acceleration units often employ efficient parallel computing architectures, such as Systolic Arrays [64], and are specifically optimized to perform core operations in deep learning models, including matrix multiplication and convolution [33]. This architectural specialization enables their widespread application in various EAI scenarios, including image recognition, speech processing, and natural language understanding, playing a crucial role particularly in edge computing devices where power consumption and cost are stringent requirements [63,65]. For example, RV series SoCs (e.g., RV1126/RV1109) (Microchip, Chandler, AZ, USA) [66] and K230 (Kendryte, Singapore) [67] are SoCs that integrate NPUs or KPUs to accelerate diverse edge AI computing tasks. By highly integrating modules such as CPUs, GPUs, Image Signal Processors (ISPs), and dedicated NPUs/KPUs onto a single chip, AI SoCs not only enhance AI computational efficiency but also reduce system power consumption, cost, and physical size, making them a crucial hardware foundation for promoting the popularization and implementation of AI technology across various industries [33,63].

Table 3. Mainstream GPU/FPGA/ASIC hardware platforms of EAI.

Platforms Models	Manu-Facturer	Arch-Itecture	Processor	AI Accelerator	Computational Power	Power Consumption	Other Key Features
Jetson Orin NX [50] ¹	NVIDIA Corporation, Santa Clara, CA, USA.	GPU	ARM Cortex-A78AE	NVIDIA Ampere GPU, Tensor Cores	100.0 TOPS (16 GB ver.), 70.0 TOPS (8 GB ver.)	10–25 W	CUDA, TensorRT
Jetson Nano [51]	NVIDIA Corp.	GPU	ARM Cortex-A57, NVIDIA Maxwell GPU	NVIDIA Maxwell GPU	0.472 TFLOPs (16-bit floating-point, FP16)/5 TOPS (INT8, sparse)	5–10 W (typically)	CUDA, TensorRT
Zynq 7000 [58]	AMD (Xilinx), Santa Clara, CA, USA	FPGA	Dual-core Arm Cortex-A9 MPCore	Programmable Logic (PL) for custom accelerators	PS (A9): Max ~5000 DMIPS	Low power (depends on design and load)	Highly customizable hardware, Real-time processing capability

Table 3. Cont.

Platforms Models	Manu-Facturer	Arch-Itecture	Processor	AI Accelerator	Computational Power	Power Consumption	Other Key Features
Arria 10 [59]	Intel Corp., Santa Clara, CA, USA	FPGA	Dual-core Arm Cortex-A9 (SoC versions)	FPGA fabric for custom accelerators	SoC (A9): Max ~7500 DMIPS	Low power (depends on design and load)	Enhanced FPGA and Digital Signal Processing (DSP) capabilities
iCE40HX1K [68]	Lattice Semiconductor, Hillsboro, OR, USA.	FPGA	iCE40 LM (FPGA family)	FPGA fabric for custom accelerators	1280 Logic Cells	Very low power	Small form factor, Low power consumption
Smart Fusion2 [69]	Microchip Technology Inc., Chandler, AZ, USA.	FPGA	ARM Cortex-M3	FPGA fabric for custom accelerators	MCU (M3): ~200 DMIPS	Low power	Secure, high-performance for comms, industrial interface, automotive markets
Google Tensor G3 [61]	Google Inc., Mountain View, CA, USA.	ASIC	1× Cortex-X3, 4× Cortex-A715, 4× Cortex-A510	Mali-G715 GPU	N/A (High-end mobile SoC performance)	N/A (Mobile SoC)	Emphasizes AI functions and image processing
Coral USB Accelerator [70]	Google Inc.	ASIC	ARM 32-bit Cortex-M0+ Microprocessor (controller)	Edge TPU	4.0 TOPS (INT8)	2.0 W	USB interface, Plug-and-play
MLU220-SOM [71]	Cambricon Technologies, Beijing, China.	ASIC	4× ARM Cortex-A55	Cambricon MLU (Memory Logic Unit) NPU	16.0 TOPS (INT8)	15 W	Edge intelligent SoC module
Atlas 200I DK A2 [62,72]	Huawei Technologies Co., Ltd., Shenzhen, China.	ASIC	4 core @ 1.0 GHz	Ascend AI Processor	8.0 TOPS (INT8), 4.0 TOPS (FP16)	24 W	Software and hardware development kit
RK3588 [66]	Fuzhou Rockchip Electronics Co., Ltd. (Rockchip), Fuzhou, China.	AI SoC	4× Cortex-A76 + 4× Cortex-A55	ARM Mali-G610 MC4, NPU	NPU: 6.0 TOPS	~8.04 W	Supports multi-camera input
RV1126 [66]	Rockchip	AI SoC	ARM Cortex-A7, RISC-V MCU	NPU	2.0 TOPS (NPU)	1.5 W	Low power, Optimized for visual processing
K230 [67]	Kendryte (Canaan Inc.), Singapore.	AI SoC	2× C908 (RISC-V), RISC-V MCU	NPU	1.0 TOPS (NPU)	2.0 W	Low power, Supports TinyML
AX650N [73]	AXERA (Axera Tech), Ningbo, China.	AI SoC	ARM Cortex-A55	NPU	72.0 TOPS (INT4), 18.0 TOPS (INT8)	8.0 W	Video encoding/decoding, Image processing
RDK X3 Module [74]	Horizon Robotics, Beijing, China.	AI SoC	ARM Cortex-A53 (Application Processors)	Dual-core Bernoulli BPU	5.0 TOPS	10 W	Optimized for visual processing

¹ While Jetson platforms are Systems-on-Module (SoMs) containing CPUs, their primary AI acceleration and categorization for high-performance AI tasks stem from their powerful integrated NVIDIA GPUs.

In the autonomous driving sector, NPUs can process multi-channel camera streams in real-time with extremely low latency to perform complex tasks like object detection and semantic segmentation—a feat that is challenging for general-purpose CPUs or GPUs to match. In smart security, NPUs empower edge cameras with high-accuracy facial recognition and behavior-analysis capabilities, significantly reducing bandwidth and computational demands on back-end servers. Similarly, in consumer electronics and smart home devices, the proliferation of low-power NPUs has made functions like offline voice recognition and intelligent image enhancement possible, which not only improves responsiveness but also fundamentally protects user data privacy. Therefore, NPUs/KPUs

are not merely hardware accelerators; they are strategic components that define the core competitiveness of the next generation of intelligent embedded products.

3.2. Hardware Platform Selection Strategy

When selecting an EAI hardware platform, it is necessary to comprehensively consider application requirements, algorithm complexity, and development cost and cycle, and to ultimately determine the most suitable solution through prototype verification and performance testing [33,75].

The selection strategy for EAI hardware platforms is shown in Figure 2. Firstly, application requirements need to be clarified. This includes specific requirements for computational power, power consumption, latency, cost, security, and reliability [76]. For example, autonomous driving has extreme demands for high computational power and low latency, while smart homes focus more on low power consumption and cost [77]. Next, algorithm complexity needs to be evaluated. This involves model type such as Convolutional Neural Network (CNN), Recurrent Neural Networks (RNN) and Transformer, model size (MB), and computational complexity (FLOPs) [78]. Lightweight models can run on CPUs, while complex models require GPU, FPGA, or ASIC acceleration [79]. Concurrently, development cost and cycle must be considered. GPUs and AI chips have more mature development tools and community support, which can shorten the development cycle, whereas FPGAs and ASICs require more hardware design and verification work and involve considerations such as development tools, community support, software ecosystem, and Internet Protocol (IP) licensing fees [80]. Finally, through prototype verification and performance testing, evaluate the hardware platform's performance in specific application scenarios using benchmarks, real-world tests, power and latency tests, stress tests, and performance-analysis tools, and continuously optimize [81,82]. For instance, using the Huawei's Atlas 200 DK for prototype verification, or reducing hardware performance demands through model quantization and pruning techniques [78,83].

To illustrate these selection principles in a real-world context, the design of wearable devices such as smartwatches or fitness trackers clearly demonstrates why a MCU is a far more suitable choice than a MPU. Firstly, from the perspective of application requirements, wearables demand extremely low power consumption for multi-day battery life and have strict constraints on form factor; an MCU's power draw in active and sleep modes (often in the μA to mA range) is orders of magnitude lower than an MPU's (typically in the hundreds of mA range). Secondly, concerning algorithmic complexity, the AI tasks on these devices are typically lightweight, such as keyword spotting or activity recognition. The computational power of modern MCUs, especially those with specialized instructions (e.g., ARM Cortex-M with Helium), is sufficient for these TinyML models, whereas an MPU's high-performance cores would lead to significant underutilization and inefficiency. Furthermore, from the viewpoint of development cost and cycle, the highly integrated nature of MCUs (with CPU, memory, and peripherals on a single chip) greatly simplifies design, reduces costs, and shrinks product size, in stark contrast to MPU-based systems that require multiple external components. Finally, regarding performance (real-time), wearables need instant responsiveness, and an MCU, running on an RTOS or bare-metal, can boot instantly and provide deterministic feedback, perfectly suiting tasks like health monitoring. Cumulatively, an MPU is clearly "overkill" in this scenario. The selection strategy, therefore, definitively guides the choice of an MCU as the optimal solution, as it meets all stringent constraints while providing sufficient performance for the target AI tasks.

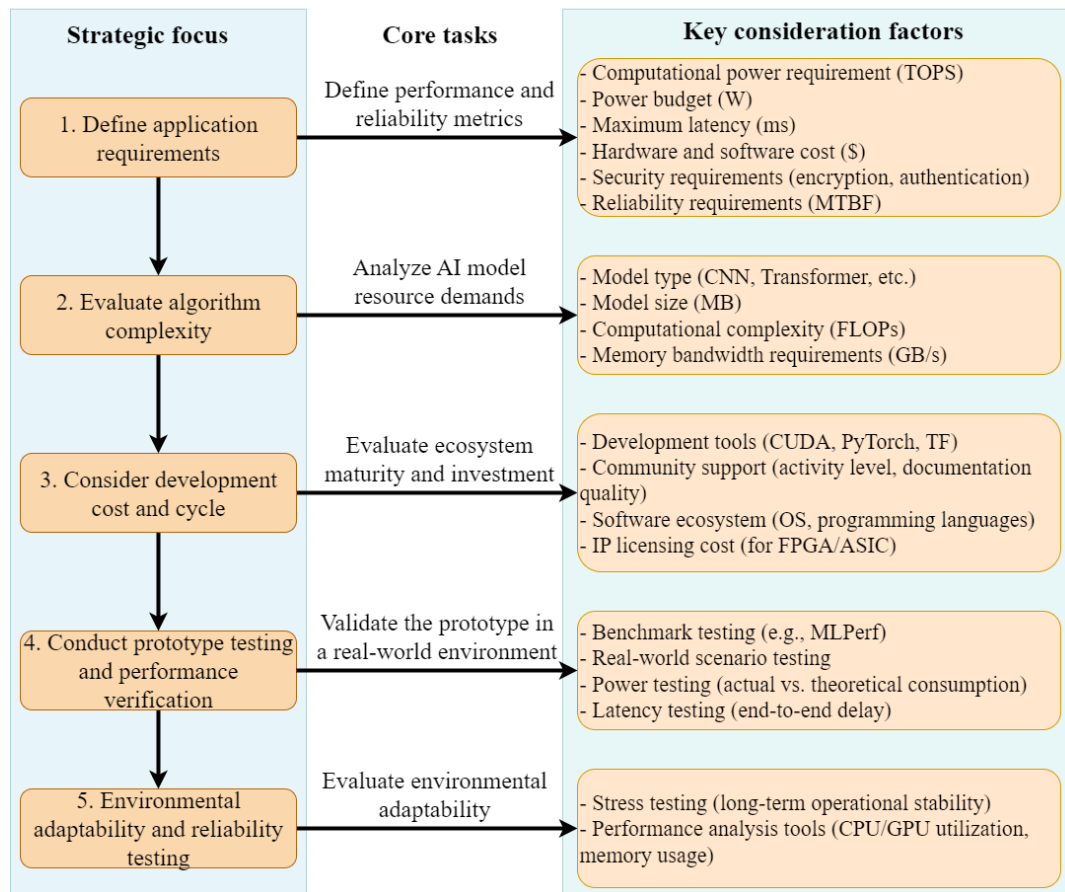


Figure 2. The selection strategy for EAI hardware platforms.

4. Software Frameworks of EAI

Although hardware provides the underlying computational power for EAI, it cannot directly execute abstract AI models on its own. A critical bridge is needed between the hardware and the algorithms: the software framework [84]. These frameworks provide the necessary tools, libraries, and runtime environments to efficiently convert, optimize, and deploy AI models onto target hardware, thereby truly unleashing its potential. This section will provide a comprehensive review of the mainstream software frameworks that support EAI development and their ecosystems.

4.1. Key Frameworks for EAI

Table 4 summarizes several key software frameworks for the EAI domain, each with its unique features and advantages.

TFLite [85]: TFLite is Google’s official lightweight deployment solution, designed for mobile and embedded devices, providing a complete toolchain from model conversion and optimization to deployment. Through its flexible “Delegates” mechanism, TFLite can seamlessly offload computational tasks to hardware accelerators like GPUs and Digital Signal Processing (DSPs). Its key advantage is the TFLite for Microcontrollers version, which extends AI capabilities to extremely low-power devices with only kilobyte-level memory.

PyTorch Mobile [86]: PyTorch Mobile is Meta’s official solution for seamlessly deploying PyTorch models to mobile devices while preserving its flexible development experience. It enables an end-to-end workflow from training to deployment *TorchScript* model using lightweight ExecuTorch. Thanks to its superior support for dynamic computational graphs,

PyTorch Mobile is an ideal choice for mobile AI applications that require rapid iteration and complex models.

Open Neural Network Exchange (ONNX) Runtime [87]: ONNX Runtime, developed by Microsoft, is a high-performance, cross-platform inference engine that fulfills the “train once, deploy anywhere” philosophy by supporting the open ONNX format. It can load and execute models from nearly all major frameworks and leverages a powerful “Execution Providers” architecture to call underlying hardware-acceleration libraries for optimal performance, with broad support for devices from the cloud to the edge.

Arm NN [88] and CMSIS-NN [89]: Arm NN and CMSIS-NN [89] are official low-level acceleration libraries from Arm, deeply optimized for its processor IP. Arm NN targets Cortex-A series CPUs and Mali GPUs, intelligently allocating computational tasks to balance performance and power consumption. Meanwhile, CMSIS-NN provides highly optimized kernel functions specifically for Cortex-M MCUs. They often serve as acceleration backends for higher-level frameworks, making them crucial for unlocking the full AI potential of Arm hardware.

NCNN [90]: NCNN is a high-performance, lightweight inference framework from Tencent, built specifically for mobile devices. It has no third-party dependencies, a very small library size, and is deeply optimized for the Arm CPU’s NEON instruction set, delivering excellent CPU inference speed. NCNN also supports Vulkan GPU acceleration, making it ideal for mobile vision applications with strict requirements on speed and resource consumption.

Apache Tensor Virtual Machine (TVM) [91]: Apache TVM is an open-source deep learning compiler stack that compiles models from various frameworks into highly-optimized machine code for heterogeneous hardware, including CPUs, GPUs, and FPGAs. Through automated graph optimization and operator tuning, TVM can rapidly generate high-performance code for emerging hardware, making it a key tool for bridging the gap between AI models and diverse hardware platforms. MediaPipe [92]: MediaPipe is Google’s open-source, cross-platform framework for building real-time, multimodal media-processing pipelines. It offers a suite of pre-built, high-performance AI solutions (like face and hand detection), centered around a graph-based development model that allows developers to flexibly construct complex applications like building blocks. Optimized for real-time performance, MediaPipe helps developers quickly integrate high-quality vision AI features across multiple platforms.

TinyML Frameworks [93–95]: In the field of microcontroller ML (TinyML), extremely lightweight frameworks like uTensor [96] and MicroMLP [97] have emerged. Their common goal is to convert neural network models into a form with a minimal memory footprint. For instance, uTensor achieves this by compiling models directly into C++ code, eliminating runtime interpreter overhead. These frameworks aim to enable AI on the lowest-cost, lowest-power sensor nodes.

End-to-End Platforms: Edge Impulse [98] and SensiML [99] are end-to-end embedded ML (MLOps) cloud platforms designed to significantly lower the barrier to AI development. They provide a complete visual workflow from data collection and model training to firmware generation, allowing developers without deep AI expertise to build applications quickly. Edge Impulse is known for its ease of use, while SensiML excels at handling complex sensor data.

Table 4. Mainstream EAI software frameworks.

Framework Name	Vendor	Supported Hardware	Development Languages	Languages	Use Cases
TFLite [85]	Google	iOS, Linux, Microcontrollers, Edge TPU, GPU, DSP, CPU	C++, Python, Swift	Java,	Image recognition, Speech recognition, Natural language processing, Sensor data analysis

Table 4. Cont.

Framework Name	Vendor	Supported Hardware	Development Languages	Lan-	Use Cases
PyTorch Mobile [86]	Meta (Facebook)	Android, iOS, Linux, CPU	Java, Objective-C, C++		Image recognition, Speech recognition, Natural language processing
ONNX Runtime [87]	Microsoft (and Community/Partners)	CPU, GPU, Dedicated Accelerators (Intel OpenVINO, NVIDIA TensorRT)	C++, C#, Python, JavaScript, Java		Various ML tasks, Model deployment
Arm NN [88]	Arm	Arm Cortex-A, Arm Cortex-M, Arm Ethos-NPU	C++		Image recognition, Speech recognition, Natural language processing, Efficient inference on Arm devices
CMSIS-NN [89]	Arm	ARM Cortex-M	C		Neural network programming for resource-constrained embedded systems
NCNN [90]	Tencent	Android, iOS	C++		Image recognition, Object detection, Face recognition, Mobile AI applications
TVM [91]	Apache (from UW)	CPU, GPU, FPGA, Dedicated Accelerators (e.g., VTA)	C++, Rust, Java		Various ML tasks, Model deployment, Especially for optimizing for heterogeneous hardware
MediaPipe [92]	Google	Android, iOS, Linux, Web	C++, JavaScript	Python,	Real-time media processing, Face detection, Pose estimation, Object tracking
TinyML [93–95]	N/A (Domain/Concept/Community)	Microcontrollers (ARM Cortex-M, RISC-V), Sensors	C, C++, Java		Sensor data analysis, Anomaly detection, Voice activation
uTensor [96]	Open Source (formerly part of MemryX)	Microcontrollers (ARM Cortex-M)	C++		Sensor data analysis, Simple control tasks
MicroMLP [97]	N/A (Concept/Specific library)	Ultra-low-power microcontrollers	C		Simple classification tasks, Gesture recognition
Edge Impulse [98]	Edge Impulse (Company)	Microcontrollers, Linux devices, Sensors	C++, JavaScript	Python,	Various embedded ML applications, especially for scenarios requiring rapid prototyping and deployment
SensiML [99]	SensiML (a QuickLogic company)	Microcontrollers, Sensors	C++, Python		Various embedded ML applications, especially for sensor data analysis requiring low power and high efficiency

4.2. The Typical Performance Characteristics of Key Frameworks

To translate the qualitative descriptions of these frameworks into a more actionable comparison, Table 5 summarizes the typical performance characteristics of several key frameworks across standardized evaluation dimensions, based on their design objectives and publicly available benchmark results. These metrics—*inference latency*, *peak RAM usage*, and *model size*, target Processor tier—represent the most critical considerations for EAI deployment.

Table 5 clearly illustrates the design trade-offs among different frameworks:

- **Ecosystem vs. Performance:** TFLite boasts the most comprehensive ecosystem and toolchain, but its generality also means that performance on specific hardware may not be optimal. Specialized tools like TVM and CMSIS-NN, on the other hand, strive for extreme performance but have relatively higher barriers to entry and integration complexity.
- **Versatility vs. Specificity:** ONNX Runtime, with its strong interoperability, serves as a bridge connecting diverse ecosystems. NCNN and CMSIS-NN, on the other hand, focus on specific areas (mobile CPUs/ ARM Cortex-M), achieving exceptional performance on their respective target platforms.
- **Resource-Consumption Hierarchy:** Frameworks exhibit a clear hierarchy in terms of resource consumption. TFLite for Microcontrollers and CMSIS-NN are at the bottom of the pyramid and are designed for KB-level memory. TFLite and ONNX Runtime are suitable for mobile devices or embedded Linux systems with more memory.

Table 5. Comparison of typical performance characteristics of mainstream EAI software frameworks ¹.

Framework	Inference Latency	Peak RAM Usage	Model Size	Target Processor Tier
TFLite [85]	Medium	Medium	Medium	Mobile/High-End Embedded) (ARM Cortex-A CPU, Mobile GPU, DSP)
ONNX Runtime [87]	Low	Medium	Medium	Mobile/High-End Embedded and Server (ARM Cortex-A, ×86 CPU, NVIDIA GPU)
Apache TVM [91]	Extremely Low	Low	Small	Heterogeneous Platforms (MCU, CPU, GPU, FPGA, ASIC)
Arm CMSIS-NN [88,89]	Extremely Low (on Arm Cortex-M)	Extremely Low	Extremely Small	MCU (ARM Cortex-M)
NCNN [90]	Low (on mobile CPUs)	Low	Small	Mobile (ARM Cortex-A CPU)

¹ The table compares the typical performance characteristics of mainstream EAI software frameworks according to metric tiers, providing guidance for selecting specific software frameworks.

5. Algorithms of EAI

With the hardware foundation and software toolchain established, our attention naturally turns to the heart of any EAI system: the AI algorithm itself. The algorithmic choice is a primary determinant of a system's intelligence, its functional capabilities, and its intrinsic resource demands. For embedded systems, selecting a suitable algorithm necessitates a delicate balance not only of high accuracy but also of computational efficiency, memory footprint, and robustness. This section delves into the various algorithms applicable to EAI, paying special attention to lightweight models custom-designed for resource-constrained settings, and presents a comparative analysis of the experimental outcomes of classic algorithms.

5.1. Traditional ML Algorithms

These algorithms have relatively low computational complexity and are easy to deploy on resource-constrained embedded systems.

5.1.1. Decision Trees

Decision trees [100] perform classification or regression through a series of if-then-else rules. They are easy to understand and implement, have low computational complexity, and are suitable for handling classification and regression problems, especially when the relationships between features are unclear. Decision trees can be used for sensor data analysis, determining device status based on sensor readings; they can also be used for fault diagnosis, determining the cause of failure based on fault symptoms.

5.1.2. Support Vector Machines

Support Vector Machines (SVM) [101] performs classification by finding an optimal hyperplane in a high-dimensional space that maximizes the margin between samples of different classes. SVM performs well on small sample datasets and has good generalization ability, making it suitable for image recognition, text classification, etc. In embedded systems, SVM can be used to recognize simple image patterns, such as handwritten digit recognition or simple object recognition.

5.1.3. K-Nearest Neighbors

K-Nearest Neighbors (KNN) [102] is a distance-based classification algorithm. For a given sample, the KNN algorithm finds the K training samples most similar to it and determines the class of the sample by voting based on the classes of these samples. The KNN algorithm is simple and easy to understand, but its computational complexity is

relatively high, especially with high-dimensional data. Therefore, the KNN algorithm is typically suitable for processing low-dimensional data, such as sensor data classification.

5.1.4. Naive Bayes

Naive Bayes [103] is a classification algorithm based on Bayes' theorem, which assumes that features are mutually independent. The Naive Bayes algorithm is computationally fast, requires less training data, and is suitable for text classification, spam filtering, etc. In embedded systems, Naive Bayes can be used to determine environmental status based on sensor data, for example, to identify air quality.

5.1.5. Random Forests

Random Forests [104] is an ensemble learning algorithm that improves prediction accuracy by integrating multiple decision trees. Random Forests have strong anti-overfitting capabilities and can handle high-dimensional data. In embedded systems, Random Forests can be used for complex sensor data analysis, such as predicting the remaining useful life of equipment.

5.2. Typical Deep Learning Algorithms

Deep learning algorithms, leveraging their powerful feature-extraction and pattern-recognition capabilities, have achieved breakthroughs in multiple fields. This paper/section will review several landmark deep learning algorithms.

5.2.1. CNN

CNNs are the cornerstone of computer vision tasks, excelling particularly in areas such as image recognition, object detection, and video analysis. Their core idea is to automatically learn spatial hierarchical features of images through convolutional layers and reduce feature dimensionality through pooling layers, enhancing the model's translation invariance [2]. Classic CNN architectures like LeNet-5 [105] laid the foundation for subsequent deeper and more complex networks (such as AlexNet [5], VGG [106], ResNet [107], and Inception [108]). Although CNNs are highly effective, they typically contain a large number of parameters and computations, posing severe challenges for direct deployment on resource-constrained embedded devices.

5.2.2. RNN

RNNs and their variants are designed to process sequential data and have widespread applications in natural language processing (e.g., machine translation, text generation), speech recognition, and time series prediction. RNNs capture temporal dependencies in sequences through their internal recurrent structure. However, original RNNs are prone to vanishing or exploding gradient problems, limiting their ability to process long sequences. Long Short-Term Memory (LSTM) networks [109] and Gated Recurrent Units (GRU) [110] effectively alleviate these issues by introducing gating mechanisms, becoming standard models for processing sequential data. Similar to CNNs, complex RNN models, especially those containing multiple stacked LSTM/GRU layers, can also have computational and memory requirements that exceed the capacity of embedded devices.

5.2.3. Continual Learning

Beyond static pre-trained models, an important algorithmic paradigm of EAI is On-Device Continual Learning. The objective of this framework is to enable embedded devices to incrementally learn from continuous data streams, thereby adapting to dynamic environments or providing personalized services without requiring large-scale retraining from scratch. The core algorithmic challenge lies in addressing the issue of catastrophic forget-

ting, where a model forgets previously acquired knowledge while learning new tasks. To tackle this challenge under the resource constraints of embedded devices, related methods often employ parameter-efficient updates (modifying only a small subset of parameters) or data-efficient rehearsal (storing only a tiny set of representative past samples) [111–113], thereby striking a balance between acquiring new capabilities (plasticity) and retaining old knowledge (stability).

5.2.4. Federated Edge AI (FEAI)

Federated Edge AI (FEAI) represents a privacy-preserving distributed learning framework in which models are collaboratively trained across a large number of edge devices without the need to centralize users' raw data. Within this framework, a central server only distributes the global model and aggregates updates uploaded from individual devices, while the actual training computation takes place locally on the devices. The primary algorithmic challenges include handling the non-independent and identically distributed (Non-IID) nature of data across devices and designing memory- and computation-efficient training algorithms for device-side execution [114]. Recent research efforts have focused on addressing these challenges to ensure that even the most resource-constrained micro-controllers can effectively participate in federated learning, thereby enabling collaborative construction of more powerful models while preserving user privacy [115].

5.2.5. Transformer Networks

Transformer Networks were initially designed for natural language-processing tasks (particularly machine translation). Their core is the Self-Attention Mechanism, which can process all elements in a sequence in parallel and capture long-range dependencies, significantly outperforming traditional RNNs [116]. The success of Transformers quickly extended to other domains; for example, Vision Transformer (ViT) [117] applied it to image recognition, achieving performance comparable to or even better than CNNs. However, Transformer models typically have an extremely large number of parameters and high computational complexity (especially the quadratic complexity of the self-attention mechanism), making their direct deployment on embedded devices a significant challenge.

To address this challenge, researchers have proposed several key innovations for developing tiny Transformer models:

- **Hybrid Architectures:** An effective approach is to design hybrid models that combine the respective strengths of CNNs and Transformers. For instance, Mobile Vision Transformer (MobileViT) [118] leverages standard convolutional layers to efficiently capture local features, which are then fed into a lightweight Transformer module to model long-range global dependencies.
- **Complexity Reduction:** This strategy aims to reduce the computational and memory demands of the model. One method is to apply compression techniques such as knowledge distillation, where knowledge is transferred from a large pre-trained Transformer (the “teacher model”) to a much smaller student Transformer, enabling the student model to achieve high performance with significantly fewer parameters [119]. Another method directly modifies the self-attention mechanism itself, employing techniques like linear attention, sparse attention, or kernel-based approximations to reduce the computational complexity from quadratic to linear, thereby making it more suitable for on-device deployment [120].

While these aforementioned typical deep learning algorithms have achieved great success in their respective fields, they are generally designed for servers or workstations with powerful computational capabilities and ample memory. Their common characteristic is a large model scale and intensive computation. Therefore, to endow edge devices

with these powerful AI capabilities, deployment on embedded devices requires model-compression and -optimization techniques such as Quantization [78,83], Pruning [78,121], and Knowledge Distillation [122].

5.3. Lightweight Neural Network Architectures

To address the resource limitations of embedded devices, researchers have designed a series of lightweight neural network architectures. These architectures, through innovative network structure designs and efficient computational units, significantly reduce model parameter count and computational complexity while maintaining high performance.

5.3.1. MobileNet Series

- MobileNetV1 [123]: Proposed by Google, its core innovation is Depthwise Separable Convolutions, which decompose standard convolutions into Depthwise Convolution and Pointwise Convolution, drastically reducing computation and parameters.
- MobileNetV2 [124]: Built upon V1 by introducing Inverted Residuals and Linear Bottlenecks, further improving model efficiency and accuracy.
- MobileNetV3 [125]: Combined NAS technology to automatically optimize network structure and introduced the h-swish activation function and updated Squeeze-and-Excitation (SE) modules, achieving better performance under different computational resource constraints.
- MobileNetV4 [126]: builds upon the success of the previous MobileNet series by introducing novel architectural designs, such as the Universal Inverted Bottleneck (UIB) block and Mobile Multi-Head Query Attention (MQA), to further enhance the model's performance under various latency constraints. MobileNetV4 achieves a leading balance of accuracy and efficiency across a wide range of mobile ecosystem hardware, covering application scenarios from low latency to high accuracy.

5.3.2. ShuffleNet Series

- ShuffleNetV1 [127]: Proposed by Megvii Technology, designed for devices with extremely low computational resources. Its core elements are Pointwise Group Convolution and Channel Shuffle operations, the latter promoting information exchange between different groups of features and enhancing model performance.
- ShuffleNetV2 [128]: Further analyzed factors affecting actual model speed (such as memory access cost) and proposed better design criteria, resulting in network structures that perform better in both speed and accuracy.

5.3.3. SqueezeNet

SqueezeNet [129] is proposed by DeepScale (later acquired by Tesla) and Stanford University, among others, aiming to significantly reduce model parameters while maintaining AlexNet-level accuracy. Its core is the Fire module, which includes a squeeze convolution layer (using 1×1 kernels to reduce channels) and an expand convolution layer (using 1×1 and 3×3 kernels to increase channels).

5.3.4. EfficientNet Series

- EfficientNetV1 [130]: Proposed by Google, it uniformly scales network depth, width, and input image resolution using a Compound Scaling method. It also used neural architecture search to obtain an efficient baseline model B0, which was then scaled to create the B1-B7 series, achieving state-of-the-art accuracy and efficiency at the time.
- EfficientNetV2 [131]: Building on V1, it further optimized training speed and parameter efficiency by introducing Training-Aware NAS and Progressive Learning, and incorporated more efficient modules like Fused-MBConv.

5.3.5. GhostNet Series

- GhostNetV1 [132]: proposed by Huawei Noah's Ark Lab, introduces the Ghost Module. This module generates intrinsic features using standard convolutions, then applies cheap linear operations to create more "ghost" features, significantly reducing computation and parameters while maintaining accuracy.
- GhostNetV2 [133], also from Huawei Noah's Ark Lab, enhances V1 by incorporating Decoupled Fully Connected Attention (DFCA). This allows Ghost Modules to efficiently capture long-range spatial information, boosting performance beyond V1's local feature focus with minimal additional computational cost.
- GhostNetV3 [134], from relevant researchers, extends GhostNet's efficiency to ViTs. It applies Ghost Module principles—like using cheap operations within ViT components—to create lightweight ViT variants with reduced complexity and parameters, making them suitable for edge deployment [134].

5.3.6. MobileViT

MobileViT (Mobile Vision Transformer) [118]: Proposed by Apple Inc. (Cupertino, CA, USA), it is one of the representative works successfully applying Transformer architecture to mobile vision tasks. It ingeniously combines convolutional modules from MobileNetV2 and self-attention modules from Transformers to design a lightweight ViT, achieving competitive performance on mobile devices.

The design of these lightweight networks provides feasible solutions for deploying advanced deep learning models on resource-constrained embedded devices, driving the rapid development of EAI.

5.4. Performance Comparison of Mainstream Lightweight Neural Networks

To gain a deeper understanding of the design philosophies and performance trade-offs of different lightweight architectures, Table 6 summarizes the key performance metrics of mainstream lightweight neural networks, including the MobileNet series, ShuffleNet series, SqueezeNet, GhostNet, and MobileViT, on the standard image classification benchmark (ImageNet). This comparison highlights differences in Top-1 accuracy, parameters, computational cost (FLOPs), inference latency, and test hardware, offering a clear view of how each architecture balances efficiency and performance.

A comprehensive analysis of the table allows us to distill four key insights into the evolution of lightweight neural network architectures. First, the field has undergone a co-evolution of efficiency and accuracy: innovations such as depthwise separable convolutions and neural architecture search have enabled higher model performance under reduced resource consumption. Second, design principles have moved beyond merely minimizing theoretical computations (FLOPs) and entered a hardware-aware stage; as exemplified by ShuffleNetV2, accounting for practical hardware bottlenecks, such as memory access costs, has become critical for achieving real-world speed improvements. Third, the outstanding performance of MobileViT on dedicated NPUs highlights the power of algorithm–hardware co-design, demonstrating that tightly coupling model efficiency with target hardware is central to future performance breakthroughs. Finally, the overall evolution does not aim for a single optimal solution but rather constructs a Pareto frontier of performance, allowing developers to make informed trade-offs among accuracy, latency, and model size according to specific application requirements.

Table 6. Performance comparison of mainstream lightweight neural networks.

Model Architecture	Top-1 Accuracy (ImageNet) ¹	Params	FLOPs (G) ²	Latency ³	Test Hardware
SqueezeNet ⁴ [129]	0.58	1.25 M	0.82		
MobileNetV1(1.0×) [123]	0.71	4.2 M	0.57	113 ms	Google Pixel 1 CPU
MobileNetV2(1.0×) [124]	0.72	3.5 M	0.3	75 ms	Google Pixel 1 CPU
MobileNetV3-Large (1.0×) [125]	0.75	5.4 M	0.22	51 ms	Google Pixel 1 CPU
MobileNetV3-Small (1.0×) [125]	0.67	2.5 M	0.06	15 ms	Google Pixel 1 CPU
ShuffleNetV1 (1.0×, g = 3) [127]	0.69	3.4 M	0.14	50 ms	Qualcomm Snapdragon 820
ShuffleNetV2 (1.0×) [128]	0.69	2.3 M	0.15	39 ms	Qualcomm Snapdragon 820
GhostNet (1.0×) [132]	0.74	5.2 M	0.14	74.5 ms	Qualcomm Snapdragon 625
MobileViT-XS [118]	0.75	2.3 M	0.7	7.28 ms	Apple A14 NPU -ANE ⁵

¹ ImageNet Top-1: The aforementioned models are primarily designed for image classification, and thus the main evaluation metric is Top-1 accuracy. They can serve as backbone networks for object detectors (e.g., You Only Look Once, YOLO), whose mAP performance depends on the overall detector design and is therefore not listed here. ² FLOPs (G): Giga FLOPs, i.e., billions of floating-point operations. Here, FLOPs typically refer to MACs (Multiply-Accumulate operations). Original model architecture papers rarely report precise energy consumption, as it is influenced by hardware, clock frequency, workload, and other factors. In general, lower inference latency and fewer FLOPs usually imply lower energy consumption per inference. ³ Inference latency is highly dependent on the testing hardware and software stack; therefore, special attention should be paid to the “Test Hardware” column when comparing latency across models. ⁴ The SqueezeNet paper was published relatively early, and its reported latency is not directly comparable to that on modern mobile devices; however, its core contribution lies in achieving an extremely small model size. ⁵ ANE: Apple Neural Engine, a dedicated AI accelerator (also known as NPU) on Apple devices.

6. Deployment of EAI

Following our discussion of the core components of EAI systems—hardware, software, and algorithms—we now address their integration through a systematic deployment process. This crucial phase, often unaddressed in existing surveys, is a complex engineering practice that extends beyond simple model conversion to include optimization, compilation, hardware integration, and iterative validation. This section details the end-to-end workflow for converting a model from a development environment into a stable application on a target embedded device.

As is shown in Figure 3, a typical deployment workflow can be broken down into key stages: model training, model optimization and compression, model conversion and compilation, hardware integration and inference execution, and performance evaluation and validation.

6.1. Model Training

The starting point of the deployment process is obtaining an AI model that meets functional requirements. This is typically accomplished in environments with abundant computational resources (e.g., GPU clusters) and mature deep learning frameworks (e.g., TensorFlow [135] or PyTorch [86]). Developers utilize large-scale datasets to train the model, aiming to achieve high predictive accuracy or target metrics [2]. The model produced at this stage is typically based on 32-bit floating-point (FP32) operations, possessing a large number of parameters and high computational complexity. While it performs well on servers, it often far exceeds the carrying capacity of typical embedded devices.

The core challenge in this stage is the “Train-Deploy Gap”. A model trained on large-scale, high-quality datasets in a resource-rich server environment may not generalize well to the low-quality, noisy data collected by embedded devices in real-world, complex settings.

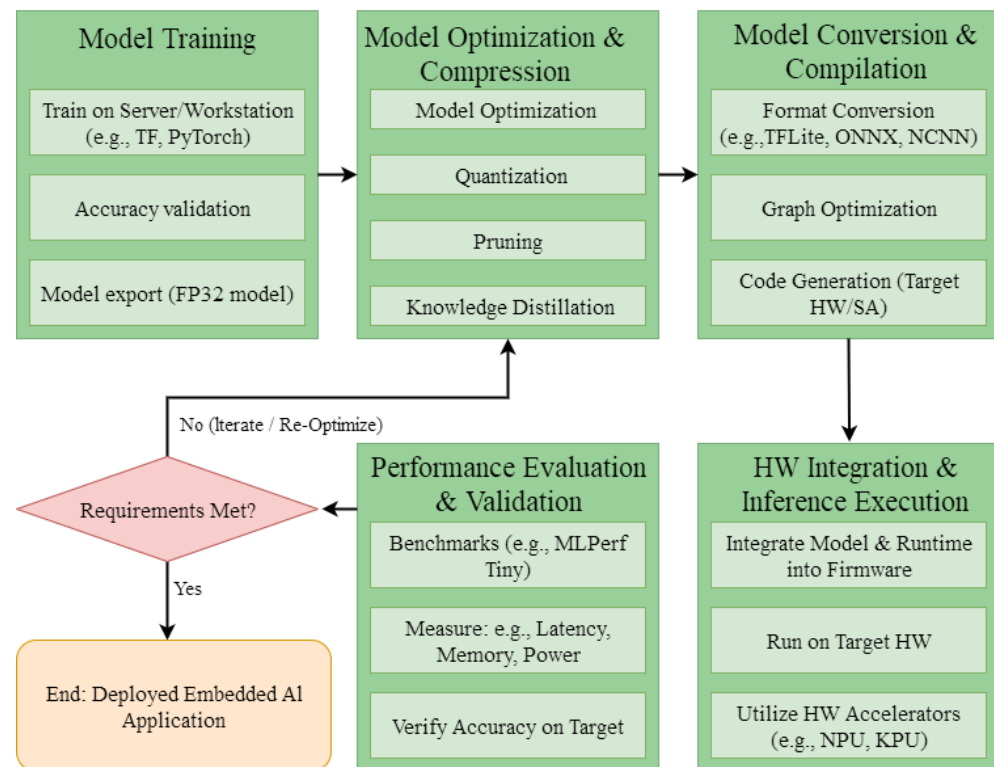


Figure 3. The typical deployment workflow.

6.2. Model Optimization and Compression

This constitutes a critical step in adapting models to embedded environments. Due to the stringent constraints of embedded devices in terms of computational power, memory capacity, and power consumption, it is essential to optimize the originally trained models. The goal of this stage is to significantly reduce model size and computational complexity while preserving accuracy as much as possible. Common optimization techniques include Quantization, Pruning, and Knowledge Distillation.

To assess the independent impact of each technique, ablation studies are indispensable. It should be noted, however, that we do not need to re-conduct such experiments, as the effectiveness of these methods has already been thoroughly validated in numerous seminal works. The data presented in Table 7 are extracted from these classic studies, with the aim of clearly illustrating the core contributions of each technique.

- **Quantization:** Converts FP32 weights and/or activations into low-bitwidth representations (e.g., INT8, FP16), thereby exploiting more efficient integer or half-precision computational units [83,136,137]. This method yields the most direct and significant improvement in inference latency. As demonstrated in ablation studies (e.g., Configuration 1), quantization can provide a 2–3× speedup on compatible hardware, at the cost of typically slight and controllable accuracy degradation (e.g., approximately 1.5% loss on ResNet-50).
- **Pruning:** Removes redundant weights or structures (e.g., channels, filters) to produce sparse models with fewer parameters and lower computational requirements [138]. Pruning techniques excel in maintaining accuracy, often achieving nearly lossless performance (e.g., Configuration 2 and Configuration 4, <0.5% accuracy drop) after careful fine-tuning. However, their practical impact on inference latency depends heavily on the type of pruning (structured vs. unstructured) and the degree of hardware support for sparse computation. In many cases, pruning primarily reduces the

theoretical computational load (FLOPs), offering potential rather than guaranteed acceleration.

- **Knowledge Distillation:** Trains a compact “student” model under the guidance of a larger “teacher” model, transferring knowledge to enhance the performance of the smaller network [122,139,140]. Distillation does not directly alter inference latency; its core value lies in significantly boosting the upper bound of model accuracy. As shown in experiments (e.g., Configuration 3), distillation can even yield more than a 1.2% absolute accuracy gain for a strong baseline. This enables the deployment of a smaller and faster student model that, through distillation, can match or surpass the accuracy of its larger counterpart, thereby indirectly achieving low latency.

These optimization techniques can be applied independently or in combination, with the choice depending on the target hardware capabilities and the specific accuracy–latency requirements of the application. Ablation studies indicate that combined approaches often deliver the best results. For example, knowledge distillation can effectively compensate for the accuracy degradation caused by quantization and pruning (e.g., Configuration 5 [140]), sometimes even leading to a “compressed yet stronger” model. A typical advanced optimization pipeline proceeds as follows: pruning is first applied to obtain an efficient compact architecture, knowledge distillation is then employed to restore and further enhance accuracy, and finally quantization is applied to achieve inference acceleration on the target hardware—thereby striking an optimal balance among accuracy, latency, and model size (e.g., Configuration 6 [141]).

Table 7. Ablation studies of major model-compression techniques.

Config	Technique Combination	Model and Dataset	Baseline Acc.	Optimized Acc.	Acc. Loss/Gain	Speed Up
1	Quantization Only [83]	ResNet-50/ ImageNet	Top-1: 76.4%	Top-1: 74.9%	−0.015	4× (size) 2–3× (latency)
2	Pruning Only [138]	VGG-16/ImageNet	Top-5: 92.5%	Top-5: 92.1%	−0.004	FLOPs reduced by 50.2%
3	Knowledge Distillation Only [139]	ResNet-34 (Student)/ ImageNet	Top-1: 73.31%	Top-1: 74.58%	0.0127	1× (vs. student model)
4	Pruning (Automated) + Distillation (Implicit) [31]	MobileNetV1/ ImageNet	Top-1: 70.9%	Top-1: 70.7%	−0.002	FLOPs reduced by 50% (2× speedup)
5	Quantization + Knowledge Distillation [140]	ResNet-18 (Student)/ CIFAR-100	Top-1: 77.2%	Top-1: 78.4%	+1.2% (vs. stu- dent) (Acc. sur- passes FP32 teacher)	4-bit Quantization
6	Pruning + Quan- tization + (Self-)Distillation [141]	OFA (MobileNetV3)/ ImageNet		Top-1: 79.9%	(No traditional base- line)	Specialized sub-network (pruned) + INT8 quan- tization Latency: 7 ms (Samsung Note10)

The primary challenge at this stage is the difficult “Accuracy-Efficiency Trade-off.” Nearly all optimization techniques, particularly quantization and pruning, introduce some degree of accuracy loss while reducing resource consumption. The greatest difficulty lies in selecting the right combination of techniques to minimize this accuracy degradation to an acceptable level while satisfying strict hardware constraints.

6.3. Model Conversion and Compilation

The goal of conversion and compilation is to generate a compact, efficient model representation that can run on the target hardware. The optimized model needs to be converted into a format that the target embedded platform’s inference engine can understand and execute. This process is not just about file format conversion; it often involves further graph optimization and code generation.

- **Format Conversion:** This crucial step is responsible for converting the optimized model from its original training framework (e.g., TensorFlow, PyTorch) or a common interchange format like ONNX [87] into a specific format executable by the target embedded inference engine. For example, TFLite Converter [85] generates .tflite files; NCNN [90] uses its conversion toolchain (e.g., onnx2ncnn) to produce its proprietary .param and .bin files; Intel's OpenVINO [142] creates its optimized Intermediate Representation (IR) format (.xml and .bin files) via the Model Optimizer. Additionally, many hardware vendors' Software Development Kits (SDKs) also provide conversion tools to adapt models to their proprietary, hardware-acceleration-friendly formats.
- **Graph Optimization:** Inference frameworks or compilers (e.g., Apache TVM [91]) perform platform-agnostic and platform-specific graph optimizations, such as operator fusion (merging multiple computational layers into a single execution kernel to reduce data movement and function call overhead), constant folding, and layer replacement (substituting standard operators with hardware-supported efficient ones), among others.
- **Code Generation:** For some frameworks (e.g., TVM) or SDKs targeting specific hardware accelerators (NPUs), this stage generates executable code or libraries highly optimized for the target instruction set (e.g., ARM Neon SIMD, RISC-V Vector Extension) or hardware-acceleration units.

The most common and challenging issue in this phase is "Operator Incompatibility." The set of operators supported by training frameworks (like PyTorch/TensorFlow) is far larger than that supported by embedded inference engines (like TFLite for microcontrollers). If a model contains an operator not supported by the target framework, the conversion process will fail, forcing developers to either redesign the model or manually implement custom operators, which significantly increases development complexity.

6.4. Hardware Integration and Inference Execution

This stage involves deploying the converted model to run on the actual embedded hardware.

- **Integration:** Integrating the converted model files (e.g., .tflite files or weights in C/C++ array form) and inference engine libraries (e.g., the core runtime of TFLite for MCUs [85], ONNX Runtime Mobile, or specific hardware vendor inference libraries) into the firmware of the embedded project. This typically involves including the corresponding libraries and model data within the embedded development environment (e.g., based on Makefile, CMake, or IDEs like Keil MDK, IAR Embedded Workbench).
- **Runtime Environment:** The inference engine runs in an embedded operating system (e.g., Zephyr, FreeRTOS, Mbed OS) or a bare-metal environment. It requires allocating necessary memory for it (typically statically allocated to avoid the overhead and uncertainty of dynamic memory management, especially on MCUs [85]).
- **Application Programming Interfaces (APIs) Invocation:** Application code, by calling APIs provided by the inference engine, completes steps such as model loading, input data preprocessing, inference execution (run() or similar functions), and obtaining output results and post-processing.
- **Hardware Accelerator Utilization:** If the target platform includes hardware accelerators (e.g., NPUs, GPUs), it is necessary to ensure the inference engine is configured with the correct "delegate" or "execution provider" [89] to offload computationally intensive operations (e.g., convolutions, matrix multiplications) to the hardware accelerator for execution, fully leveraging its performance and power efficiency advantages [117]. This often requires integrating drivers and specialized libraries provided

by the hardware vendor (e.g., ARM CMSIS-NN [143] for optimizing operations on ARM Cortex-M).

The central challenge here is “System-Level Resource Management and Integration.” This includes not only the precise and safe memory allocation for model weights and intermediate activations (the “Tensor Arena”) within extremely limited RAM but also the correct configuration and invocation of drivers and libraries for NPUs. Furthermore, it involves ensuring that the AI inference task can be completed deterministically within its deadline without interfering with other real-time tasks in the system (such as control loops).

6.5. Performance Evaluation and Validation

After deployment, the actual performance of the model must be comprehensively evaluated and validated on the target embedded hardware.

- **Benchmarking:** To ensure objective and reproducible performance evaluation and enable fair comparisons across different hardware and software solutions, adopting industry-standard benchmarks is crucial. In the resource-constrained domain, MLPerf Tiny [36] is the most prominent benchmark suite. It provides an “apples-to-apples” comparison platform for the inference latency and energy consumption of various solutions through its standardized models, datasets, and measurement rules. Benchmarking with MLPerf Tiny helps developers validate their solutions and make more informed technical decisions.
- **Key Metrics Measurement:** It is necessary to accurately measure the model’s inference latency (time taken for a single inference), throughput (number of inferences processed per unit time), memory footprint (peak RAM and Flash/Read-Only Memory (ROM) occupation), and power consumption (Energy per Inference or Average Power).
- **Accuracy Validation:** Evaluate the accuracy or other task-relevant metrics of the deployed model (after optimization and conversion) on real-world data or representative datasets to ensure it meets application requirements, and compare it with the pre-optimization accuracy to assess whether the accuracy loss due to optimization is within an acceptable range.

It is important to emphasize that the linear workflow depicted in Figure 3 is an idealized model simplified for clarity. In real-world applications, this process is highly iterative and nonlinear. This is especially true during the model-optimization and -compression phase, where the order of internal steps (such as pruning and quantization) is not fixed but heavily depends on the selected toolchain and optimization strategy, such as choosing between “pruning before quantization” and the more advanced “quantization aware pruning”. Therefore, optimization itself is a complex mini-workflow. More importantly, the feedback loop from “performance evaluation” back to “optimization” is the essence of the entire deployment process: if the evaluation results fail to meet application requirements (e.g., excessive latency, memory usage, or significant accuracy degradation), it is necessary to return to previous steps and make adjustments. This may involve trying different optimization strategies (such as adjusting quantization parameters or using different pruning rates), selecting different lightweight models, adjusting model conversion/compilation options, or even redesigning the model architecture or considering a different hardware platform. This design-optimization-deployment-evaluation cycle may require multiple iterations to achieve the ultimate goal.

The core challenge in this final stage is achieving “Representative and Accurate Validation.” Performance metrics (such as latency and power consumption) measured on a development board can differ significantly from the final product’s performance in its actual operating environment (e.g., under varying temperatures or battery voltages). Designing test cases that accurately reflect the final deployment scenario and using professional tools

to precisely measure metrics like energy consumption are critical to preventing product failures in real-world applications.

In summary, the deployment of EAI models is a complex process involving multidisciplinary knowledge, requiring meticulous engineering practices and repeated iterative optimization. Its ultimate goal is to achieve efficient and reliable on-device intelligence while satisfying the stringent constraints of embedded systems.

7. Applications of EAI

EAI is no longer a distant vision but a reality driving transformation across numerous industries. By directly deploying AI capabilities on Edge Devices, EAI significantly enhances system autonomy, operational efficiency, and real-time responsiveness. This chapter aims to delve into the diverse applications of EAI, elaborating on its specific implementations, impacts, and future potential in various fields.

7.1. Autonomous Vehicles

Autonomous driving is one of the most prominent and technologically challenging application areas for EAI. Systems need to process heterogeneous data from multiple sources such as cameras, Light Detection and Ranging (LiDAR), Radar, and ultrasonic sensors in real-time to accomplish complex tasks including environmental perception, object detection (e.g., vehicles, pedestrians, cyclists, traffic signals), path planning, and vehicle control. Among these, CNNs, particularly variants like You Only Look Once (YOLO) [144] or Single Shot MultiBox Detector (SSD) [145], are typically deployed on high-performance AI SoCs, such as the NVIDIA Jetson series or Huawei's Ascend series, to meet the stringent real-time requirements of functional safety standards such as ISO 26262 [146] (see Section 3.1.6). The deployment process relies on TensorRT or the vendor's own efficient inference engine, which maximizes the parallel computing power of the dedicated NPU through techniques such as operator fusion and precision calibration to achieve millisecond-level end-to-end latency.

Meanwhile, RNNs and their variants like LSTM, along with Deep Reinforcement Learning (DRL) algorithms, play a key role in processing sequential data, path planning, and complex decision-making [147]. Advanced Driver-Assistance Systems (ADAS) functions such as Lane Keeping Assist (LKA), Adaptive Cruise Control (ACC), Autonomous Emergency Braking (AEB), and Blind Spot Monitoring (BSM) heavily rely on EAI for real-time environmental perception and precise control, significantly enhancing driving safety [148]. Achieving high-level (L4) or even full (L5) autonomous driving places extremely high demands on the robustness, reliability, and real-time capabilities of EAI systems, which must be able to handle extreme weather conditions, complex traffic flows, and unforeseen emergencies. EAI also plays a crucial role in monitoring the status of core components in autonomous vehicles, particularly in Battery-management Systems (BMS) [149]. For example, a study proposed a novel LSTM-RNN model that achieves high-precision, real-time prediction of lithium-ion battery State-of-charge by expanding network inputs and constraining outputs.

Small robots used for last-mile logistics delivery operate on sidewalks or in limited areas, typically combining visual and LiDAR data, and utilizing EAI for localization, navigation, and obstacle avoidance [150].

7.2. Smart Sensors and IoT Devices

EAI is profoundly transforming the IoT ecosystem. Traditional IoT devices primarily collect data and upload it to the cloud for processing, whereas smart sensors integrated with AI possess the capability to perform data analysis and inference on-device. This edge

computing paradigm effectively reduces data transmission bandwidth requirements, shortens system response latency, and enhances data privacy and security [151]. By analyzing sensor data from industrial equipment (e.g., motors, pumps, bearings) such as vibration, temperature, and acoustics, EAI algorithms (e.g., LSTM-based anomaly-detection models) can predict potential failures in advance, enabling proactive maintenance and avoiding significant losses due to unexpected downtime [152]. AI-integrated sensors can monitor environmental parameters like air quality (e.g., PM2.5, O₃) and water quality (e.g., turbidity, pH) in real-time. Edge AI algorithms can instantly identify pollution events, analyze pollution sources, or trigger alerts [153]. Smart agriculture benefits from EAI-driven sensors that monitor soil moisture, nutrient content, meteorological conditions, and crop growth status (e.g., identifying pests and diseases through image analysis), empowering precision agriculture, optimizing irrigation and fertilization strategies, and improving resource utilization and crop yields [154]. By analyzing data from thermostats, lighting controllers, occupancy sensors, and security cameras, EAI algorithms can optimize building energy consumption (Heating, Ventilation and Air Conditioning control, HVAC), enhance security levels (anomaly detection), and improve occupant comfort and experience [155].

These LSTM or CNN models deployed to the IoT edge must be extremely lightweight to achieve long battery life on battery-powered devices. They are typically deployed on low-power MCUs (such as the ARM Cortex-M series), which occupy only tens of KB of memory (see Section 3.1.1). Key to achieving this deployment is the use of frameworks designed for bare-metal environments, such as TFLite for microcontrollers or CMSIS-NN (see Section 4). These frameworks, through 8-bit integer quantization and optimized kernel libraries, make it possible to run neural networks on extremely resource-constrained hardware.

7.3. Wearable Devices and Healthcare

The application of EAI in the healthcare sector is driving advancements in personalized health management, early disease screening, and improved patient care outcomes. Smart wristbands, watches, and other wearable devices utilize built-in Inertial Measurement Units (IMUs) and other sensor data, employing EAI algorithms (e.g., CNN or RNN-based models) to accurately recognize user physical activity types (walking, running, sleeping, etc.), providing exercise tracking and health recommendations [156]. These devices continuously monitor vital signs such as heart rate, heart rate variability (HRV), blood oxygen saturation (SpO_2), and even non-invasive blood glucose. EAI can analyze this data in real-time to detect abnormalities like arrhythmia and sleep apnea, and issue warnings to users or medical personnel [157]. For example, a lightweight CNN model for real-time heart rate abnormality detection (e.g., arrhythmia) must be designed to strictly adhere to MCU resource constraints. Developers might choose an efficient architecture like MobileNetV3-Small (see Section 5.3) and leverage TFLite for Microcontrollers for 8-bit quantization, ultimately deploying it on a Cortex-M4/M33 core MCU with only a few hundred KB of Flash and tens of KB of RAM. The entire application must consume power in the microamp or milliamp range to ensure weeks of battery life.

In medical image analysis, EAI algorithms (especially CNNs) are used to assist doctors in identifying abnormal features in X-rays, Computed Tomography (CT), or Magnetic Resonance Imaging (MRI), such as tumors, fractures, or lesion areas, improving diagnostic efficiency and accuracy, particularly in resource-limited scenarios [158]. Smart devices are being developed for individuals with disabilities. For example, AI-based smart prosthetics can understand user intent through electromyography (EMG) or neural signals to provide more natural control; visual assistance devices utilize EAI to describe the surrounding environment for visually impaired individuals [159].

7.4. Industrial Automation and Robotics

EAI is a core driving force behind Industry 4.0 and next-generation industrial automation, enabling robots and automated systems to perform complex tasks with greater autonomy, flexibility, and intelligence. Collaborative robots (Cobots) can work safely and efficiently alongside humans in shared workspaces. EAI (especially AI combined with vision and force sensors) endows Cobots with environmental perception, collision avoidance, and task adaptation capabilities, which are key to achieving human-robot collaboration [160]. Utilizing deep learning-based computer vision technology, EAI systems can automate the detection of minute defects, dimensional deviations, or assembly errors on industrial product surfaces, often surpassing human eye detection in accuracy and speed [161]. By analyzing sensor data from production lines in real-time (e.g., temperature, pressure, flow rate, energy consumption), EAI algorithms can identify production bottlenecks, predict equipment performance degradation, or recommend optimal process parameters, continuously improving production efficiency and resource utilization [162]. In environments such as warehouses, factories, or hospitals, Autonomous Mobile Robots (AMRs) utilize EAI, (combining Simultaneous Localization and Mapping (SLAM) path planning, and perception algorithms) to navigate autonomously, transport materials, and perform inventory tasks, thereby improving internal logistics efficiency [163].

The application of EAI is not limited to physical operations on production lines but also extends to predictive analysis and economic decision support for complex industrial assets. For example, in a study on container ship construction cost forecasting, a complex model integrating CNN, bidirectional LSTM, and attention mechanisms was successfully used to capture the nonlinear and time-varying characteristics of price fluctuations [164]. This powerful architecture capable of processing complex time series data, once lightweight, is expected to be deployed on edge controllers of large industrial equipment (such as ships and drilling platforms) in the future.

7.5. Consumer Electronics

EAI has been widely integrated into various consumer electronics products, significantly enhancing user experience and product functionality. In smartphones, AI-driven computational photography techniques (e.g., night mode, portrait blur), more accurate voice assistants (offline voice recognition), personalized recommendation systems, and device performance optimization (e.g., smart power management) are becoming increasingly common [165]. The improvement of on-device Natural Language Processing (NLP) capabilities in smart speakers allows some voice interactions to be completed locally, improving response speed and privacy protection [166]. In smart TVs, AI is used for image quality optimization (e.g., super-resolution, scene recognition to adjust picture quality), smart content recommendation, and voice or gesture-based interactive control. EAI also enables drones to achieve autonomous flight control, intelligent target tracking, real-time obstacle avoidance, and high-precision image data acquisition and analysis [167].

7.6. Security and Surveillance

EAI is revolutionizing traditional security and surveillance systems, shifting them from passive recording to active analysis and early warning, thereby improving monitoring efficiency and accuracy. By running AI algorithms on cameras or edge servers, intelligent video analytics can be realized, including facial recognition, vehicle recognition, crowd density estimation, behavior analysis (e.g., loitering, fall detection), and intrusion detection [168]. Systems can also perform anomaly detection, automatically identifying events or behaviors in surveillance videos that deviate from normal patterns, such as abnormal intrusions, unattended object detection, or fights, and trigger alarms in a timely man-

ner [169]. For perimeter security, AI analyzes data from infrared, radar, or video sensors to more accurately identify and distinguish real intrusion threats (e.g., people, vehicles) from environmental interference (e.g., animals, weather), reducing false alarms.

7.7. Sustainability and Energy Management

Beyond enhancing the intelligence of individual devices, EAI is demonstrating significant potential in advancing sustainability and energy management. By deploying EAI on edge devices within the power grid, such as smart meters and inverters, fine-grained management of a distributed smart grid can be achieved. EAI algorithms can analyze local power-consumption patterns and renewable energy generation (e.g., from solar panels) in real-time to autonomously perform demand response, load balancing, and fault diagnosis. This improves the overall energy efficiency, stability, and renewable energy integration capacity of the grid. Furthermore, in smart buildings, AI-embedded sensors can optimize HVAC and lighting systems by dynamically adjusting energy usage based on real-time occupancy and environmental conditions, thereby minimizing energy waste. These applications illustrate that EAI is not merely a tool for improving individual efficiency but also a key enabling technology for building a greener and more sustainable future.

7.8. Emerging and Future Applications

The application boundaries of EAI are continuously expanding, and its fusion with other cutting-edge technologies is constantly creating new possibilities. Combined with the high-speed, low-latency characteristics of 5G/6G, EAI and edge computing will support more complex distributed intelligent applications, such as Vehicle-to-Everything (V2X), large-scale real-time IoT analytics, and immersive Augmented Reality (AR)/Virtual Reality (VR) [170]. AI-driven smart prosthetics and exoskeletons are being developed to respond more naturally and intuitively to user intent (even via Brain-Computer Interfaces, BCI), enhancing the quality of life for people with disabilities [171]. Personalized adaptive learning systems, where EAI is deployed on educational devices or platforms to analyze student learning behavior and progress in real-time, dynamically adjusting teaching content, difficulty, and pace, can achieve truly personalized educational experiences [172]. Furthermore, On-Device Learning enables edge devices not only to perform inference but also to use local data for model fine-tuning or continuous learning, further enhancing personalization and adaptability while protecting user privacy [173]. Photonic Computing, using photons for computation, promises unprecedented speed and energy efficiency for real-time edge applications with massive data streams, such as high-resolution optical sensing. Similarly, brain-inspired Neuromorphic Hardware, via its event-driven, asynchronous processing, will empower next-generation EAI applications requiring ultra-low-power, real-time responses to sparse, asynchronous sensor data, such as in biomimetic robotics or always-on health monitors.

8. Challenges and Opportunities

Despite significant progress in EAI, several key challenges remain to be overcome. Concurrently, the burgeoning field of intelligent embedded systems also presents numerous opportunities.

8.1. Challenges

8.1.1. Resource Constraints

Limited processing power, memory, and energy availability pose significant constraints on the complexity and performance of AI algorithms. Embedded devices often need to operate on battery power or in low-power modes, making the deployment of

computationally intensive AI models challenging. For example, deep learning models typically require substantial computational resources and memory to achieve high accuracy, which is infeasible on resource-constrained embedded devices. Algorithms need to trade off between accuracy, computational complexity, and model size to fit the constraints of embedded systems. Furthermore, energy efficiency is a critical consideration, as running AI algorithms can consume significant energy, thereby shortening the device's battery life [33]. Resource constraints are not only present at the hardware level but also at the software level. For instance, the resource consumption of the operating system, compiler, and AI framework needs to be considered, as this affects the final algorithm efficiency. Additionally, different resource-management strategies need to be designed for various application scenarios; for example, applications with high real-time requirements might need prioritized CPU resource allocation, while data-intensive applications might need prioritized memory resource allocation.

8.1.2. Model Optimization

Developing efficient model-compression and -optimization techniques is crucial for deploying large AI models on embedded systems. Deep learning models often have a very large number of parameters, and deploying them directly on embedded devices can lead to issues like excessive memory consumption and slow inference speeds. Therefore, model-compression techniques such as pruning, quantization, and knowledge distillation are needed to reduce model size, lower computational complexity, and improve inference speed, while maintaining model accuracy as much as possible [78,122]. Model optimization includes not only compression but also optimization for specific hardware platforms. For example, optimization for ARM platforms can fully leverage NEON instruction sets to improve computational efficiency. Furthermore, hardware-software co-optimization methods can be employed, such as designing dedicated hardware accelerators to speed up specific AI algorithms.

8.1.3. Hardware Acceleration

Utilizing hardware accelerators such as GPUs and NPUs is crucial for achieving real-time performance. General-purpose processors (CPUs) are less efficient when executing AI algorithms, whereas dedicated hardware accelerators like GPUs and NPUs can significantly improve the execution speed of AI algorithms. GPUs have a highly parallel architecture, suitable for performing computationally intensive tasks such as matrix operations. NPUs are processors specifically designed for AI algorithms, offering higher energy efficiency. For example, in autonomous driving systems, a large amount of image data needs to be processed in real-time; using GPUs or NPUs can accelerate the execution of algorithms like object detection and image segmentation, thereby ensuring system real-time capability [61,174]. The choice of hardware accelerator needs to be determined based on the specific application scenario. For instance, GPUs might be more suitable for visual applications, while NPUs might be better for voice applications. Additionally, factors such as the cost, power consumption, and development difficulty of hardware accelerators also need to be considered.

8.1.4. Security and Privacy

Protecting sensitive data and ensuring the security of EAI systems is a critical issue. Embedded devices are often deployed in various environments and may collect and process large amounts of sensitive data, such as personal identification information and health data. If EAI systems have security vulnerabilities, attackers might exploit these vulnerabilities to steal data, tamper with models, or control devices. Therefore, various security measures,

such as data encryption, access control, and vulnerability scanning, need to be implemented to protect the security of EAI systems [175].

However, in safety-critical industries, the challenges extend far beyond traditional data security to encompass functional safety and reliability, which are governed by stringent industry standards. The integration of EAI introduces unique and formidable challenges in these domains. In autonomous driving, the core difficulties lie in ensuring perception models are robust against adversarial attacks and in certifying that their non-deterministic decision-making processes comply with functional safety standards like ISO 26262. For medical devices, the challenge shifts to the verifiability and certifiability of AI-driven decisions to mitigate the risks of “black-box” models and secure regulatory approval. Meanwhile, in industrial automation, the key is to guarantee real-time and deterministic performance in scenarios like human-robot collaboration, where any unpredictable latency could lead to catastrophic outcomes. Therefore, for safety-critical EAI systems, the focus must shift from preventing external attacks to providing verifiable guarantees of safe, reliable, and predictable behavior.

The embedded domain offers several established security architectures, each with its own trade-offs. Differential privacy technology is a protection method that has been widely studied recently [176]. Its core idea is to perturb (perturb) the data and add noise before collecting or publishing it, thereby hiding the real data and preventing attackers with background knowledge from obtaining private information through guesswork. TinySec [177], while highly mature due to its simplicity and low overhead—demonstrating the feasibility of achieving strong security on resource-constrained devices—suffers from severe limitations in scalability and security owing to its reliance on a single static key. ContiSec [178] builds upon this by providing greater flexibility, yet it commonly encounters similar challenges in managing pre-shared keys. In sharp contrast, Zigbee Security [179], as an industry standard, delivers a comprehensive security suite that incorporates robust centralized key management, making it highly scalable and resilient. However, such advanced security comes at the cost of substantial resource overhead and complexity, rendering it unsuitable for the most resource-limited embedded devices.

8.1.5. Hardware-Based Threats

Beyond software-level vulnerabilities, EAI systems face a more insidious and powerful class of hardware-based threats that can bypass all software security measures. Due to the physical accessibility of embedded devices, attackers can employ Side-Channel Attacks (SCA) to reverse-engineer model weights by analyzing physical leakages such as power consumption or electromagnetic radiation [180], or use Fault Injection Attacks (FIA) to manipulate model outputs by inducing computational errors with means like voltage glitches or lasers [181]. Furthermore, Hardware Trojans originating from untrusted supply chains can implant model “backdoors,” while physical probing and reverse engineering can directly extract model parameters stored on-chip [182]. These threats demonstrate that a truly secure EAI system must adopt a full-stack defense strategy spanning software, algorithms, and hardware, including the adoption of hardware-level countermeasures such as power-balancing logic, hardware redundancy, and anti-tamper packaging.

8.1.6. Data Availability

Collecting sufficient and representative training data can be challenging in some embedded applications. Deep learning models require a large amount of training data to achieve good performance. However, in some embedded applications, acquiring large amounts of data can be very difficult. For instance, in the medical field, collecting patient medical data requires ethical approval and necessitates protecting patient privacy. In

the industrial sector, some fault data may be very rare and difficult to collect. Therefore, research into techniques like Few-shot Learning and Transfer Learning is needed to train high-performance AI models with limited data [183]. Data Augmentation is also an effective method that can generate more data by performing operations such as rotation, scaling, and cropping on existing data. Additionally, Semi-supervised Learning methods can be employed, utilizing unlabeled data to improve model performance.

8.1.7. Ethical Implications

In addition to technical challenges, the widespread deployment of EAI also raises profound ethical considerations. A core issue is model bias: models trained in the cloud and hardened on edge devices may replicate and amplify social biases (such as racial or gender bias) in the training data on a large scale, leading to persistent unfair judgments [184]. In addition, although edge processing enhances privacy, the large number of AI-enabled sensors also brings new risks regarding data privacy and surveillance, and it becomes crucial to find an ethical balance between convenience and personal space [185]. Finally, when EAI systems make incorrect decisions, their “black box” nature makes attribution of responsibility and decision transparency a difficult problem. Exploring explainable AI (XAI) methods suitable for resource-constrained devices is essential for building social trust [186]. Addressing these ethical issues requires interdisciplinary efforts to ensure that EAI technologies can benefit society in a fair and responsible manner.

8.2. Opportunities

The increasing demand for intelligent embedded systems offers numerous opportunities for innovation and development. Future research directions include:

- Developing novel AI algorithms specifically designed for resource-constrained environments. This includes developing more lightweight and efficient neural network architectures, as well as exploring new ML approaches, such as knowledge-based reasoning and symbolic AI [187]. Both the efficiency and the interpretability of the algorithms should be considered. In some applications, such as medical diagnosis, the decision-making process of AI models needs to be explained for doctors to make judgments. Therefore, there is a need to develop interpretable AI algorithms and research how to integrate knowledge into AI models.
- Exploring new hardware architectures and acceleration technologies. This includes researching novel memory technologies, In-Memory Computing architectures, and developing more efficient specialized AI chips [188]. In terms of memory, research focuses on maximizing on-chip Static Random-Access Memory (SRAM) capacity and bandwidth through techniques such as 3D stacking, as well as exploring the use of non-volatile memories like Magnetoresistive Random Access Memory (MRAM) and Resistive Random Access Memory (RRAM) in In-Memory Computing to fundamentally overcome the “memory wall” bottleneck. In terms of computation, the development of more efficient dedicated AI accelerators is central. This involves not only designing fine-grained dataflows tailored to specific neural network operators but also implementing reconfigurable, domain-specific AI coprocessors on open instruction set architectures such as RISC-V. Such hardware–software co-design enables future AI chips to deliver optimal energy efficiency for specific EAI tasks.
- Improving the efficiency and robustness of EAI frameworks. This includes developing more efficient compilers, optimizers, and runtime environments, as well as researching new model validation and testing techniques to ensure the reliability and security of EAI systems [135]. There is a need to develop cross-platform EAI frameworks to deploy AI models on different hardware platforms. Additionally, research is needed

on how to increase the automation level of EAI frameworks, for example, through automatic model optimization and automatic code generation.

- Addressing the security and privacy challenges associated with EAI deployment [189]. This includes developing new encryption techniques, differential privacy methods, and researching distributed learning methods such as Federated Learning to train AI models while protecting data privacy [190]. Research into hardware security technologies, such as Trusted Execution Environments (TEE), is needed to protect the security of AI models and data. Furthermore, research into Adversarial Attack defense techniques is required to improve the robustness of AI models.
- Developing new applications for EAI across various industries. This includes areas such as smart homes, autonomous driving, healthcare, and industrial automation. EAI is expected to enable more intelligent, efficient, and secure applications in these fields [8]. It is necessary to deeply understand the specific needs of various industries and customize EAI solutions accordingly. Additionally, attention should be paid to emerging application scenarios, such as the Metaverse and Web3.0, where EAI is expected to play a significant role.

9. Conclusions

EAI, as a rapidly advancing interdisciplinary field, is demonstrating tremendous potential to reshape various industries. This review provides a systematic and comprehensive overview of the key aspects of EAI, encompassing its fundamental definition, heterogeneous hardware platforms, software development frameworks, core algorithms, model deployment strategies, and its increasingly diverse application scenarios.

Through in-depth analysis, this review identifies the central challenge of the field as achieving efficient AI computation under strict resource constraints. Conversely, its primary opportunity lies in seamlessly integrating intelligent capabilities into the physical world's edge devices through technological innovation. Consequently, the success of future research and practice will hinge upon the breakthroughs made in addressing challenges such as resource constraints, model optimization, hardware acceleration, and security and privacy.

We believe that by continuously addressing these challenges and seizing the resultant opportunities, researchers and practitioners can collectively drive the evolution of EAI, enabling its more pervasive, efficient, and reliable integration from the cloud into the physical world. This progression will not only accelerate the adoption of intelligent technology across industries but will also profoundly reshape their landscapes, ultimately integrating seamlessly into our daily lives and ushering in a more intelligent era.

Author Contributions: Conceptualization, X.H., S.Q. and S.-K.T.; methodology, X.H. and S.Q.; formal analysis, X.H. and S.Q.; investigation, X.H. and S.Q.; resources, X.H., S.Q. and S.-K.T.; data curation, X.H.; writing—original draft preparation, X.H.; writing—review and editing, X.H. and H.W.; visualization, X.H.; supervision, S.-K.T.; project administration, X.H.; funding acquisition, X.H., H.W. and S.-K.T. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the General Program for the National Key R&D Program of China, Grant No. 2023YFC2206500; the Social Development in Science and Technology of Dongguan, Grant Nos. 20221800901472 and 20231800940532; the Songshan Lake sci-tech commissioner Program, Grant No. 20234373-01KCJ-G; the Institutional Educational Quality Project of Dongguan University of Technology, Grant Nos. 202102072, 202302060, and 2024020105; and the Undergraduate Innovation and Entrepreneurship Training Program of Dongguan University of Technology, Grant Nos. 202411819161 and 202511819008.

Data Availability Statement: The data involved in this article come from references. Relevant data can be found through the literature and will not be given separately here.

Acknowledgments: This work is supported in part by the research grant (RP/FCA-09/2023) offered by Macao Polytechnic University and in part by Faculty of Applied Sciences, Macao Polytechnic University (fca.532e.13c4.9).

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

ACC	Adaptive Cruise Control
ADAS	Advanced Driver-Assistance Systems
AEB	Autonomous Emergency Braking
AI	Artificial Intelligence
AMRs	Autonomous Mobile Robots
AR	Augmented Reality
ASIC	Application-Specific Integrated Circuit
BCI	Brain-Computer Interfaces
BMS	Sattery-management systems
BSM	Blind Spot Monitoring
CNNs	Convolutional Neural Networks
Cobots	Collaborative robots
CPU	Central Processing Unit
CT	Computed Tomography
DFCA	Decoupled Fully Connected
DRL	Deep Reinforcement Learning
EAI	Embedded AI
EMG	electromyography
FEAI	Federated Edge AI
FIA	Fault Injection Attacks
FLOPs	Floating point number operations
FP16	16-bit floating-point
FP32	32-bit floating-point
FPGA	Field-Programmable Gate Array
GPU	Graphics Processing Unit
GRU	Gated Recurrent Units
HPC	High-Performance Computing
HRV	heart rate variability
HVAC	Heating, Ventilation and Air Conditioning
IMUs	Inertial Measurement Units
INT8	8-bit integer
IoT	Internet of Things
IP	Internet Protocol
IR	Intermediate Representation
ISPs	Image Signal Processors
KNN	K-Nearest Neighbors
KPUs	Knowledge-Processing Units
LKA	Lane Keeping Assist
LSTM	Long Short-Term Memory
MCU	Microcontroller Unit
ML	Machine learning

MPU	Micro Processor Unit
MQA	Multi-Head Query Attention
MRAM	Magnetoresistive Random Access Memory
MRI	Magnetic Resonance Imaging
NAS	Neural Architecture Search
NLP	Natural Language Processing
NPU _s	Neural Processing Units
NRE	Non-Recurring Engineering
ONNX	Open Neural Network Exchange
RAM	Random Access Memory
RISC-V	Reduced Instruction Set Computing-V
RNN _s	Recurrent Neural Networks
ROM	Read-Only Memory
RRAM	Resistive Random Access Memory
SCA	Side-Channel Attacks
SDK _s	Software Development Kits
SIMD	Single Instruction Multiple Datastream
SLAM	Simultaneous Localization and Mapping
SoC	System-on-Chip
SRAM	Static Random-Access Memory
SSD	Single Shot MultiBox Detector
SVM	Support Vector Machines
TEE	Trusted Execution Environments
TFLite	TensorFlow Lite
TPU	Tensor Processing Unit
TVM	Tensor Virtual Machine
UIB	Universal Inverted Bottleneck
V2X	Vehicle-to-Everything
ViT	Vision Transformer
VR	Virtual Reality
YOLO	You Only Look Once

References

1. Bengio, Y. Learning Deep Architectures for AI. *Found. Trends[®] Mach. Learn.* **2009**, *2*, 1–127. [[CrossRef](#)]
2. LeCun, Y.; Bengio, Y.; Hinton, G. Deep learning. *Nature* **2015**, *521*, 436–444. [[CrossRef](#)] [[PubMed](#)]
3. Silver, D.; Huang, A.; Maddison, C.J.; Guez, A.; Sifre, L.; van den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. Mastering the game of Go with deep neural networks and tree search. *Nature* **2016**, *529*, 484–489. [[CrossRef](#)] [[PubMed](#)]
4. Heigold, G.; Vanhoucke, V.; Senior, A.; Nguyen, P.; Ranzato, M.; Devin, M.; Dean, J. Multilingual acoustic models using distributed deep neural networks. In Proceedings of the 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, Vancouver, BC, Canada, 26–31 May 2013; pp. 8619–8623. [[CrossRef](#)]
5. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. ImageNet classification with deep convolutional neural networks. *Commun. ACM* **2017**, *60*, 84–90. [[CrossRef](#)]
6. Satyanarayanan, M. The Emergence of Edge Computing. *Computer* **2017**, *50*, 30–39. [[CrossRef](#)]
7. Li, K.; Chang, C.; Yun, K.; Zhang, J. Research on Container Migration Mechanism of Power Edge Computing on Load Balancing. In Proceedings of the 2021 IEEE 6th International Conference on Cloud Computing and Big Data Analytics (ICCCBDA), Chengdu, China, 24–26 April 2021; pp. 386–390. [[CrossRef](#)]
8. Vermesan, O.; Friess, P.; Guillemin, P.; Sundmaeker, H.; Eisenhauer, M.; Moessner, K.; Le Gall, F.; Cousin, P. Internet of Things Strategic Research and Innovation Agenda. In *Internet of Things*; River Publishers: Gistrup, Denmark, 2022; pp. 7–151. [[CrossRef](#)]
9. Moons, B.; Bankman, D.; Verhelst, M. Embedded Deep Neural Networks. In *Embedded Deep Learning*; Springer International Publishing: Cham, Switzerland, 2018; pp. 1–31. [[CrossRef](#)]
10. Lee, C.Y.; Fite, M.; Rao, T.; Achour, S.; Kapetanovic, Z. HyperCam: Low-Power Onboard Computer Vision for IoT Cameras. *arXiv* **2025**, arXiv:2501.10547.

11. Cai, H.; Gan, C.; Zhu, L.; Han, S. Tinytl: Reduce memory, not parameters for efficient on-device learning. *Adv. Neural Inf. Process. Syst.* **2020**, *33*, 11285–11297.
12. Peng, D.; Fu, Z.; Wang, J. Pockettllm: Enabling on-device fine-tuning for personalized llms. *arXiv* **2024**, arXiv:2407.01031.
13. Vivekanand, C.V.; Purushothaman, R.; Kushbu, S.C.; Selvam, M.A.J. Tiny ML-based Non-Invasive Approach of Cardiac Monitoring. In Proceedings of the 2024 7th International Conference on Contemporary Computing and Informatics (IC3I), Greater Noida, India, 18–20 September 2024; Volume 7, pp. 529–534.
14. Noh, S.H.; Shin, B.; Choi, J.; Lee, S.; Kung, J.; Kim, Y. FlexNeRFer: A Multi-Dataflow, Adaptive Sparsity-Aware Accelerator for On-Device NeRF Rendering. In Proceedings of the 52nd Annual International Symposium on Computer Architecture, Tokyo, Japan, 21–25 June 2025; pp. 1894–1909.
15. Warden, P.; Situnayake, D. *Tinyml: Machine Learning with Tensorflow Lite on Arduino and Ultra-Low-Power Microcontrollers*; O'Reilly Media: Sebastopol, CA, USA, 2019.
16. Lane, N.D.; Bhattacharya, S.; Georgiev, P.; Forlivesi, C.; Jiao, L.; Qendro, L.; Kawsar, F. DeepX: A Software Accelerator for Low-Power Deep Learning Inference on Mobile Devices. In Proceedings of the 2016 15th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN), Vienna, Austria, 11–14 April 2016; pp. 1–12. [\[CrossRef\]](#)
17. Guo, C.; Ci, S.; Zhou, Y.; Yang, Y. A survey of energy consumption measurement in embedded systems. *IEEE Access* **2021**, *9*, 60516–60530. [\[CrossRef\]](#)
18. Bonomi, F.; Milito, R.; Zhu, J.; Addepalli, S. Fog computing and its role in the internet of things. In Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing, Helsinki, Finland, 17 August 2012; pp. 13–16. [\[CrossRef\]](#)
19. Madni, A.M.; Sievers, M.; Madni, C.C. Adaptive Cyber-Physical-Human Systems: Exploiting Cognitive Modeling and Machine Learning in the Control Loop. *Insight* **2018**, *21*, 87–93. [\[CrossRef\]](#)
20. Ray, P.P. A survey of IoT cloud platforms. *Future Comput. Inform. J.* **2016**, *1*, 35–46. [\[CrossRef\]](#)
21. Shilpa Kodgire, D.; Roopali Palwe, M.; Ganesh Kharde, M. EMBEDDED AI. In *Futuristic Trends in Artificial Intelligence Volume 3 Book 12*; IIP Iterative International Publishers: Novi, MI, USA; Selfypage Developers Pvt Ltd.: Chikkamagaluru, India, 2024; pp. 137–152. [\[CrossRef\]](#)
22. Elkhaliq, W.A. AI-Driven Smart Homes: Challenges and Opportunities. *J. Intell. Syst. Internet Things* **2023**, *8*, 54–62. [\[CrossRef\]](#)
23. Peccia, F.N.; Bringmann, O. Embedded Distributed Inference of Deep Neural Networks: A Systematic Review. *arXiv* **2024**, arXiv:2405.03360. [\[CrossRef\]](#)
24. Zhang, Z.; Li, J. A Review of Artificial Intelligence in Embedded Systems. *Micromachines* **2023**, *14*, 897. [\[CrossRef\]](#)
25. Kotyal, K.; Nautiyal, P.; Singh, M.; Semwal, A.; Rai, D.; Papnai, G.; Nautiyal, C.T.; Malathi, G.; Krishnaveni, S. Advancements and Challenges in Artificial Intelligence Applications: A Comprehensive Review. *J. Sci. Res. Rep.* **2024**, *30*, 375–385. [\[CrossRef\]](#)
26. Serpanos, D.; Ferrari, G.; Nikolakopoulos, G.; Perez, J.; Tauber, M.; Van Baelen, S. Embedded Artificial Intelligence: The ARTEMIS Vision. *Computer* **2020**, *53*, 65–69. [\[CrossRef\]](#)
27. Dunne, R.; Morris, T.; Harper, S. A Survey of Ambient Intelligence. *ACM Comput. Surv.* **2021**, *54*, 1–27. [\[CrossRef\]](#)
28. Sannasy Rao, K.; Lean, C.P.; Ng, P.K.; Kong, F.Y.; Basir Khan, M.R.; Ismail, D.; Li, C. AI and ML in IR4.0: A Short Review of Applications and Challenges. *Malays. J. Sci. Adv. Technol.* **2024**, *4*, 141–148. [\[CrossRef\]](#)
29. Lee, E.A. Cyber Physical Systems: Design Challenges. In Proceedings of the 2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC), Orlando, FL, USA, 5–7 May 2008; pp. 363–369. [\[CrossRef\]](#)
30. Marwedel, P. *Embedded System Design*; Springer International Publishing: Cham, Switzerland, 2018. [\[CrossRef\]](#)
31. He, Y.; Lin, J.; Liu, Z.; Wang, H.; Li, L.J.; Han, S. Amc: Automl for model compression and acceleration on mobile devices. In Proceedings of the European conference on computer vision (ECCV), Munich, Germany, 8–14 September 2018; pp. 784–800.
32. Lin, H.Y. Embedded Artificial Intelligence: Intelligence on Devices. *Computer* **2023**, *56*, 90–93. [\[CrossRef\]](#)
33. Sze, V.; Chen, Y.H.; Yang, T.J.; Emer, J.S. Efficient Processing of Deep Neural Networks: A Tutorial and Survey. *Proc. IEEE* **2017**, *105*, 2295–2329. [\[CrossRef\]](#)
34. Yiu, J. Introduction to ARM® Cortex®-M Processors. In *The Definitive Guide to ARM® CORTEX®-M3 and CORTEX®-M4 Processors*; Elsevier: Amsterdam, The Netherlands, 2014; pp. 1–24. [\[CrossRef\]](#)
35. Ray, P.P. A review on TinyML: State-of-the-art and prospects. *J. King Saud Univ.-Comput. Inf. Sci.* **2022**, *34*, 1595–1623. [\[CrossRef\]](#)
36. Mattson, P.; Reddi, V.J.; Cheng, C.; Coleman, C.; Diamos, G.; Kanter, D.; Micikevicius, P.; Patterson, D.; Schmuelling, G.; Tang, H.; et al. MLPerf: An Industry Standard Benchmark Suite for Machine Learning Performance. *IEEE Micro* **2020**, *40*, 8–16. [\[CrossRef\]](#)
37. Shankar, V. Edge AI: A Comprehensive Survey of Technologies, Applications, and Challenges. In Proceedings of the 2024 1st International Conference on Advanced Computing and Emerging Technologies (ACET), Ghaziabad, India, 23–24 August 2024; pp. 1–6. [\[CrossRef\]](#)
38. STMicroelectronics. STM32Cube.AI-STMicroelectronics-STM32 AI. 2023. Available online: <https://stm32ai.st.com/stm32-cube-ai/> (accessed on 30 May 2025).
39. NXP. i.MX RT Crossover MCUs. 2024. Available online: <https://www.nxp.com/products/processors-and-microcontrollers/arm-microcontrollers/i-mx-rt-crossover-mcus:IMX-RT-SERIES> (accessed on 27 May 2025).

40. ESP-DL. ESP-DL Introduction. 2023. Available online: https://docs.espressif.com/projects/esp-dl/zh_CN/latest/introduction/readme.html (accessed on 4 June 2025).
41. Ibrahim, D. Architecture of ARM microcontrollers. In *Arm-Based Microcontroller Multitasking Projects*; Elsevier: Amsterdam, The Netherlands, 2021; pp. 13–32. [CrossRef]
42. Murshed, M.G.S.; Murphy, C.; Hou, D.; Khan, N.; Ananthanarayanan, G.; Hussain, F. Machine Learning at the Network Edge: A Survey. *ACM Comput. Surv.* **2021**, *54*, 1–37. [CrossRef]
43. ARM. Key Architectural Points of ARM Cortex-A Series Processors. 2023. Available online: <https://developer.arm.com/documentation/den0013/d/ARM-Architecture-and-Processors/Key-architectural-points-of-ARM-Cortex-A-series-processors> (accessed on 4 June 2025).
44. Zamojski, P.; Elfouly, R. Developing Standardized SIMD API Between Intel and ARM NEON. In Proceedings of the 2018 International Conference on Computational Science and Computational Intelligence (CSCI), Las Vegas, NV, USA, 12–14 December 2018; pp. 1410–1415. [CrossRef]
45. Mediatek. MediaTek | Filogic 830 | Premium Wi-Fi 6/6E SoC. 2023. Available online: <https://www.mediatek.com/products/broadband-wifi/mediatek-filogic-830> (accessed on 5 June 2025).
46. Raspberry Pi Ltd. raspberry-pi-5-product-brief. Available online: <https://datasheets.raspberrypi.com/rpi5/raspberry-pi-5-product-brief.pdf> (accessed on 3 May 2025).
47. Intel. What Is a GPU? 2023. Available online: <https://www.intel.cn/content/www/cn/zh/products/docs/processors/what-is-a-gpu.html> (accessed on 30 June 2025).
48. Janai, J.; Güney, F.; Behl, A.; Geiger, A. Computer Vision for Autonomous Vehicles: Problems, Datasets and State of the Art. *Found. Trends® Comput. Graph. Vis.* **2020**, *12*, 1–308. [CrossRef]
49. Badue, C.; Guidolini, R.; Carneiro, R.V.; Azevedo, P.; Cardoso, V.B.; Forechi, A.; Jesus, L.; Berriel, R.; Paixão, T.M.; Mutz, F.; et al. Self-driving cars: A survey. *Expert Syst. Appl.* **2021**, *165*, 113816. [CrossRef]
50. NVIDIA. NVIDIA Jetson Orin. 2025. Available online: <https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-orin/> (accessed on 15 May 2025).
51. Kurniawan, A. Administering NVIDIA Jetson Nano. In *IoT Projects with NVIDIA Jetson Nano*; Apress: New York, NY, USA, 2020; pp. 21–47. [CrossRef]
52. NVIDIA. Jetson Developer Kits. 2025. Available online: <https://developer.nvidia.com/embedded/jetson-developer-kits> (accessed on 15 May 2025).
53. Ashbaugh, B.; Lake, A.; Rovatsou, M. Khronos™ group. In Proceedings of the 3rd International Workshop on OpenCL (IWOCCL '15), New York, NY, USA, 12–13 May 2015; pp. 1–23. [CrossRef]
54. Pipes, M.A. Qualcomm snapdragon “bullet train”. In Proceedings of the ACM SIGGRAPH 2015 Computer Animation Festival, SIGGRAPH '15, Los Angeles, CA, USA, 9–13 August 2015; ACM: New York, NY, USA, pp. 124–125. [CrossRef]
55. MediaTek. MediaTek | Dimensity | 5G Smartphone Chips. n.d. Available online: <https://www.mediatek.com/products/smartphones/dimensity-5g> (accessed on 16 June 2025).
56. Wang, T.; Wang, C.; Zhou, X.; Chen, H. A survey of FPGA based deep learning accelerators: Challenges and opportunities. *arXiv* **2018**, arXiv:1901.04988.
57. Cheremisinov, D.I. Protecting intellectual property in FPGA Xilinx design. *Prikl. Diskretn. Mat.* **2014**, *2*, 110–118. [CrossRef]
58. Limited, A.E. AC7Z020 SoM with AMD Zynq™ 7000 SoC XC7Z020. 2023. Available online: <https://www.en.alinx.com/Product/SoC-System-on-Modules/Zynq-7000-SoC/AC7Z020.html> (accessed on 6 June 2025).
59. Intel. Arria® 10 FPGA and SoC FPGA. 2023. Available online: <https://www.altera.com/products/fpga/arria/10> (accessed on 6 June 2025).
60. Reuther, A.; Michaleas, P.; Jones, M.; Gadepally, V.; Samsi, S.; Kepner, J. Survey of Machine Learning Accelerators. In Proceedings of the 2020 IEEE High Performance Extreme Computing Conference (HPEC), Waltham, MA, USA, 22–24 September 2020; pp. 1–12. [CrossRef]
61. Jouppi, N.; Young, C.; Patil, N.; Patterson, D. Motivation for and Evaluation of the First Tensor Processing Unit. *IEEE Micro* **2018**, *38*, 10–19. [CrossRef]
62. Xiao, Y.; Wang, Z. Albench: A tool for benchmarking Huawei ascend AI processors. *CCF Trans. High Perform. Comput.* **2024**, *6*, 115–129. [CrossRef]
63. Chen, Y.H.; Krishna, T.; Emer, J.S.; Sze, V. Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks. *IEEE J. Solid-State Circuits* **2017**, *52*, 127–138. [CrossRef]
64. Kung, H.T.; Leiserson, C.E. Systolic arrays (for VLSI). In *Proceedings of the Sparse Matrix Proceedings 1978*; Society for Industrial and Applied Mathematics: Philadelphia, PA, USA, 1979; pp. 256–282.
65. Shafiee, A.; Nag, A.; Muralimanohar, N.; Balasubramanian, R.; Strachan, J.P.; Hu, M.; Williams, R.S.; Srikumar, V. ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars. *ACM SIGARCH Comput. Archit. News* **2016**, *44*, 14–26. [CrossRef]

66. Rockchip Electronics Co., Ltd. Rockchip RV11 Series. 2023. Available online: https://www.rock-chips.com/a/en/products/RV11_Series/index.html (accessed on 6 June 2025).
67. Kendryte. Kendryte Developer Community-Product Center. n.d. Available online: <https://www.kendryte.com/en/products> (accessed on 6 June 2025).
68. Lattice Semiconductor. iCE40 LP/HX. 2025. Available online: <https://www.latticesemi.com/ice40> (accessed on 30 June 2025).
69. Microchip. SmartFusion® 2 FPGAs. 2025. Available online: <https://www.microchip.com/en-us/products/fpgas-and-plds/system-on-chip-fpgas/smartfusion-2-fpgas> (accessed on 30 June 2025).
70. Google LLC. USB Accelerator Datasheet. 2025. Available online: <https://coral.ai/docs/accelerator/datasheet/> (accessed on 16 July 2025).
71. cambricon. MLU220-SOM, 2025. Available online: <https://www.cambricon.com/index.php?m=content&c=index&a=lists&catid=56> (accessed on 16 July 2025).
72. Huawei. Atlas 200I DK A2 Developer Kit. 2025. Available online: <https://www.hiascend.com/hardware/developer-kit-a2/resource> (accessed on 16 July 2025).
73. Axera Semiconductor. AX650N. 2025. Available online: <https://www.axera-tech.com/Product/125.html> (accessed on 6 June 2025).
74. Horizon Robotics. RDK X3 Robot Development Kit. 2023. Available online: <https://note.com/npaka/n/nc5d3c1a3b223> (accessed on 6 June 2025).
75. Bavikadi, S.; Dhavle, A.; Ganguly, A.; Haridass, A.; Hendy, H.; Merkel, C.; Reddi, V.J.; Sutradhar, P.R.; Joseph, A.; Pudukotai Dinakarrao, S.M. A Survey on Machine Learning Accelerators and Evolutionary Hardware Platforms. *IEEE Des. Test* **2022**, *39*, 91–116. [CrossRef]
76. Benmeziane, H.; Maghraoui, K.E.; Ouarnoughi, H.; Niar, S.; Wistuba, M.; Wang, N. A comprehensive survey on hardware-aware neural architecture search. *arXiv* **2021**, arXiv:2101.09336. [CrossRef]
77. Gerla, M.; Lee, E.K.; Pau, G.; Lee, U. Internet of vehicles: From intelligent grid to autonomous cars and vehicular clouds. In Proceedings of the 2014 IEEE World Forum on Internet of Things (WF-IoT), Seoul, Republic of Korea, 6–8 March 2014; pp. 241–246. [CrossRef]
78. Han, S.; Liu, X.; Mao, H.; Pu, J.; Pedram, A.; Horowitz, M.; Dally, B. Deep compression and EIE: Efficient inference engine on compressed deep neural network. In Proceedings of the 2016 IEEE Hot Chips 28 Symposium (HCS), Cupertino, CA, USA, 21–23 August 2016; pp. 1–6 [CrossRef]
79. Blaiech, A.G.; Ben Khalifa, K.; Valderrama, C.; Fernandes, M.A.; Bedoui, M.H. A Survey and Taxonomy of FPGA-based Deep Learning Accelerators. *J. Syst. Archit.* **2019**, *98*, 331–345. [CrossRef]
80. Presutto, M. Current AI Trends: Hardware and Software Accelerators. *R. Inst. Technol.* **2018**, 1–21. Available online: https://www.academia.edu/37781087/Current_Artificial_Intelligence_Trends_Hardware_and_Software_Accelerators_2018_ (accessed on 6 June 2025).
81. Ott, P.; Speck, C.; Matenaer, G. Deep end-to-end Learning im Automotivebereich /Deep end-to-end learning in automotive. In *ELIV 2017*; VDI Verlag: Düsseldorf, Germany, 2017; pp. 49–50. [CrossRef]
82. Colucci, A.; Marchisio, A.; Bussolino, B.; Mrazek, V.; Martina, M.; Masera, G.; Shafique, M. A fast design space exploration framework for the deep learning accelerators: Work-in-progress. In Proceedings of the 2020 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS), Shanghai, China, 9 November 2020; pp. 34–36.
83. Jacob, B.; Kligys, S.; Chen, B.; Zhu, M.; Tang, M.; Howard, A.; Adam, H.; Kalenichenko, D. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. In Proceedings of the 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–23 June 2018; pp. 2704–2713. [CrossRef]
84. Banbury, C.R.; Reddi, V.J.; Lam, M.; Fu, W.; Fazel, A.; Holleman, J.; Huang, X.; Hurtado, R.; Kanter, D.; Lokhtov, A.; et al. Benchmarking tinyml systems: Challenges and direction. *arXiv* **2020**, arXiv:2003.04821.
85. Zaman, F. *TensorFlow Lite for Mobile Development: Deploy Machine Learning Models on Embedded and Mobile Devices*; Apress: New York, NY, USA, 2020. [CrossRef]
86. Alla, S.; Adari, S.K. Introduction to Deep Learning. In *Beginning Anomaly Detection Using Python-Based Deep Learning*; Apress: New York, NY, USA, 2019; pp. 73–122. [CrossRef]
87. Microsoft Corporation. ONNX. n.d. Available online: <https://onnx.ai/> (accessed on 7 June 2025).
88. Arm Ltd. Arm NN SDK-Efficient ML for Arm CPUs, GPUs, NPU. 2023. Available online: <https://www.arm.com/products/silicon-ip-cpu/ethos/arm-nn> (accessed on 7 June 2025).
89. ARM-software. GitHub-ARM-software/CMSIS-NN: CMSIS-NN Library. 2023. Available online: <https://github.com/ARM-software/CMSIS-NN> (accessed on 7 June 2025).
90. Ni, H. The Ncnn Contributors. ncnn. 2017. Available online: <https://github.com/Tencent/ncnn> (accessed on 7 June 2025).
91. Chen, T.; Moreau, T.; Jiang, Z.; Shen, H.; Yan, E.Q.; Wang, L.; Hu, Y.; Ceze, L.; Guestrin, C.; Krishnamurthy, A. TVM: End-to-end optimization stack for deep learning. *arXiv* **2018**, arXiv:1802.04799.

92. Lugaresi, C.; Tang, J.; Nash, H.; McClanahan, C.; Uboweja, E.; Hays, M.; Zhang, F.; Chang, C.L.; Yong, M.G.; Lee, J.; et al. Mediapipe: A framework for building perception pipelines. *arXiv* **2019**, arXiv:1906.08172. [CrossRef]
93. mit-han-lab. Tiny Machine Learning[website]. 2024. Available online: <https://github.com/mit-han-lab/tinyml> (accessed on 7 June 2025).
94. Sanchez-Iborra, R.; Skarmeta, A.F. TinyML-Enabled Frugal Smart Objects: Challenges and Opportunities. *IEEE Circuits Syst. Mag.* **2020**, *20*, 4–18. [CrossRef]
95. Shafique, M.; Theodoridis, T.; Reddy, V.J.; Murmann, B. TinyML: Current Progress, Research Challenges, and Future Roadmap. In Proceedings of the 2021 58th ACM/IEEE Design Automation Conference (DAC), San Francisco, CA, USA, 5–9 December 2021; pp. 1303–1306 [CrossRef]
96. uTensor. uTensor-TinyML AI Inference Library. 2023. Available online: <https://github.com/uTensor/uTensor> (accessed on 1 June 2025).
97. jczic. GitHub-jczic/MicroMLP: A Micro Neural Network Multilayer Perceptron for MicroPython (Used on ESP32 and Pycom Modules). n.d. Available online: <https://github.com/jczic/MicroMLP> (accessed on 1 June 2025).
98. Edgeimpulse, Inc. The Edge AI Platform. 2025. Available online: <https://edgeimpulse.com/> (accessed on 7 June 2025).
99. SensiML. SensiML-Making Sensor Data Sensible. 2020. Available online: <https://sensiml.com/> (accessed on 7 June 2025).
100. Quinlan, J.R. Induction of decision trees. *Mach. Learn.* **1986**, *1*, 81–106. [CrossRef]
101. Cortes, C.; Vapnik, V. Support-vector networks. *Mach. Learn.* **1995**, *20*, 273–297. [CrossRef]
102. Cover, T.; Hart, P. Nearest neighbor pattern classification. *IEEE Trans. Inf. Theory* **1967**, *13*, 21–27. [CrossRef]
103. Srivastava, T.; De, D.; Sharma, P.; Sengupta, D. Empirical Copula based Naive Bayes Classifier. In Proceedings of the 2023 International Conference on Artificial Intelligence and Knowledge Discovery in Concurrent Engineering (ICECONF), Chennai, India, 5–7 January 2023; pp. 1–7. [CrossRef]
104. Breiman, L. Random Forests. *Mach. Learn.* **2001**, *45*, 5–32. [CrossRef]
105. Lecun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. Gradient-based learning applied to document recognition. *Proc. IEEE* **1998**, *86*, 2278–2324. [CrossRef]
106. Simonyan, K.; Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv* **2014**, arXiv:1409.1556.
107. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep Residual Learning for Image Recognition. In Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 27–30 June 2016; pp. 770–778. [CrossRef]
108. Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; Rabinovich, A. Going deeper with convolutions. In Proceedings of the 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Boston, MA, USA, 7–12 June 2015; pp. 1–9. [CrossRef]
109. Hochreiter, S.; Schmidhuber, J. Long Short-Term Memory. *Neural Comput.* **1997**, *9*, 1735–1780. [CrossRef]
110. Cho, K.; Van Merriënboer, B.; Gulcehre, C.; Bahdanau, D.; Bougares, F.; Schwenk, H.; Bengio, Y. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv* **2014**, arXiv:1406.1078. [CrossRef]
111. Ravaglia, L.; Rusci, M.; Nadalini, D.; Capotondi, A.; Conti, F.; Benini, L. A tinyml platform for on-device continual learning with quantized latent replays. *IEEE J. Emerg. Sel. Top. Circuits Syst.* **2021**, *11*, 789–802. [CrossRef]
112. Guo, H.; Zeng, F.; Zhu, F.; Wang, J.; Wang, X.; Zhou, J.; Zhao, H.; Liu, W.; Ma, S.; Wang, D.H.; et al. A comprehensive survey on continual learning in generative models. *arXiv* **2025**, arXiv:2506.13045. [CrossRef]
113. McMahan, B.; Moore, E.; Ramage, D.; Hampson, S.; y Arcas, B.A. Communication-efficient learning of deep networks from decentralized data. In Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS 2017), Ft. Lauderdale, FL, USA, 20–22 April 2017; pp. 1273–1282.
114. Zhao, Y.; Li, M.; Lai, L.; Suda, N.; Civin, D.; Chandra, V. Federated learning with non-iid data. *arXiv* **2018**, arXiv:1806.00582. [CrossRef]
115. Wen, D.; Jeon, K.J.; Huang, K. Federated dropout—A simple approach for enabling federated learning on resource constrained devices. *IEEE Wirel. Commun. Lett.* **2022**, *11*, 923–927. [CrossRef]
116. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, Ł.; Polosukhin, I. Attention is all you need. *Adv. Neural Inf. Process. Syst.* **2017**, *30*, 6000–6010. [CrossRef]
117. Dosovitskiy, A.; Beyer, L.; Kolesnikov, A.; Weissenborn, D.; Zhai, X.; Unterthiner, T.; Dehghani, M.; Minderer, M.; Heigold, G.; Gelly, S.; et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv* **2020**, arXiv:2010.11929.
118. Mehta, S.; Rastegari, M. Mobilevit: Light-weight, general-purpose, and mobile-friendly vision transformer. *arXiv* **2021**, arXiv:2110.02178.
119. Jiao, X.; Yin, Y.; Shang, L.; Jiang, X.; Chen, X.; Li, L.; Wang, F.; Liu, Q. Tinybert: Distilling bert for natural language understanding. *arXiv* **2019**, arXiv:1909.10351.
120. Wang, S.; Li, B.Z.; Khabsa, M.; Fang, H.; Ma, H. Linformer: Self-attention with linear complexity (2020). *arXiv* **2006**, arXiv:2006.04768.
121. LeCun, Y.; Denker, J.; Solla, S. Optimal brain damage. *Adv. Neural Inf. Process. Syst.* **1989**, *2*, 598–605.

122. Hinton, G.; Vinyals, O.; Dean, J. Distilling the knowledge in a neural network. *arXiv* **2015**, arXiv:1503.02531. [\[CrossRef\]](#)
123. Howard, A.G.; Zhu, M.; Chen, B.; Kalenichenko, D.; Wang, W.; Weyand, T.; Andreetto, M.; Adam, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv* **2017**, arXiv:1704.04861. [\[CrossRef\]](#)
124. Sandler, M.; Howard, A.; Zhu, M.; Zhmoginov, A.; Chen, L.C. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In Proceedings of the 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–23 June 2018; pp. 4510–4520. [\[CrossRef\]](#)
125. Howard, A.; Sandler, M.; Chen, B.; Wang, W.; Chen, L.C.; Tan, M.; Chu, G.; Vasudevan, V.; Zhu, Y.; Pang, R.; et al. Searching for MobileNetV3. In Proceedings of the 2019 IEEE/CVF International Conference on Computer Vision (ICCV), Seoul, Republic of Korea, 27 October–2 November 2019; pp. 1314–1324. [\[CrossRef\]](#)
126. Qin, D.; Lechner, C.; Delakis, M.; Fornoni, M.; Luo, S.; Yang, F.; Wang, W.; Banbury, C.; Ye, C.; Akin, B.; et al. MobileNetV4: Universal Models for the Mobile Ecosystem. In *Computer Vision—ECCV 2024*; Springer Nature Switzerland: Cham, Switzerland, 2024; pp. 78–96. [\[CrossRef\]](#)
127. Zhang, X.; Zhou, X.; Lin, M.; Sun, J. ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices. In Proceedings of the 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–23 June 2018; pp. 6848–6856. [\[CrossRef\]](#)
128. Ma, N.; Zhang, X.; Zheng, H.T.; Sun, J. ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design. In *Computer Vision—ECCV 2018*; Springer International Publishing: Cham, Switzerland, 2018; pp. 122–138. [\[CrossRef\]](#)
129. Iandola, F.N.; Han, S.; Moskewicz, A.W.; Ashraf, K.; Dally, W.J.; Keutzer, K. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and < 0.5 MB model size. *arXiv* **2016**, arXiv:1602.07360.
130. Tan, M.; Le, Q. Efficientnet: Rethinking model scaling for convolutional neural networks. In Proceedings of the International Conference on Machine Learning, Long Beach, CA, USA, 9–15 June 2019; pp. 6105–6114.
131. Tan, M.; Le, Q. Efficientnetv2: Smaller models and faster training. In Proceedings of the International Conference on Machine Learning, Virtual, 18–24 July 2021; pp. 10096–10106.
132. Han, K.; Wang, Y.; Tian, Q.; Guo, J.; Xu, C.; Xu, C. GhostNet: More Features From Cheap Operations. In Proceedings of the 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Seattle, WA, USA, 13–19 June 2020; pp. 1577–1586. [\[CrossRef\]](#)
133. Tang, Y.; Han, K.; Guo, J.; Xu, C.; Xu, C.; Wang, Y. GhostNetv2: Enhance cheap operation with long-range attention. *Adv. Neural Inf. Process. Syst.* **2022**, *35*, 9969–9982.
134. Liu, Z.; Hao, Z.; Han, K.; Tang, Y.; Wang, Y. Ghostnetv3: Exploring the training strategies for compact models. *arXiv* **2024**, arXiv:2404.11202.
135. Babu, R.G.; Nedumaran, A.; Manikandan, G.; Selvameena, R. TensorFlow: Machine Learning Using Heterogeneous Edge on Distributed Systems. In *Deep Learning in Visual Computing and Signal Processing*; Apple Academic Press: Palm Bay, FL, USA, 2022; pp. 71–90. [\[CrossRef\]](#)
136. Liang, T.; Glossner, J.; Wang, L.; Shi, S.; Zhang, X. Pruning and quantization for deep neural network acceleration: A survey. *Neurocomputing* **2021**, *461*, 370–403. [\[CrossRef\]](#)
137. Gholami, A.; Kim, S.; Dong, Z.; Yao, Z.; Mahoney, M.W.; Keutzer, K. A Survey of Quantization Methods for Efficient Neural Network Inference. In *Low-Power Computer Vision*; Chapman and Hall/CRC: Boca Raton, FL, USA, 2022; pp. 291–326. [\[CrossRef\]](#)
138. Molchanov, P.; Tyree, S.; Karras, T.; Aila, T.; Kautz, J. Pruning convolutional neural networks for resource efficient inference. *arXiv* **2016**, arXiv:1611.06440.
139. Zagoruyko, S.; Komodakis, N. Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer. *arXiv* **2016**, arXiv:1612.03928.
140. Kim, J.; Bhalgat, Y.; Lee, J.; Patel, C.; Kwak, N. Qkd: Quantization-aware knowledge distillation. *arXiv* **2019**, arXiv:1911.12491. [\[CrossRef\]](#)
141. Cai, H.; Gan, C.; Wang, T.; Zhang, Z.; Han, S. Once-for-all: Train one network and specialize it for efficient deployment. *arXiv* **2019**, arXiv:1908.09791.
142. Vasudevan, S.K.; Pulari, S.R.; Vasudevan, S. Intel OpenVino: A Must-Know Deep Learning Toolkit. In *Deep Learning*; Chapman and Hall/CRC: Boca Raton, FL, USA, 2021; pp. 245–260. [\[CrossRef\]](#)
143. Lai, L.; Suda, N.; Chandra, V. Cmsis-nn: Efficient neural network kernels for arm cortex-m cpus. *arXiv* **2018**, arXiv:1801.06601.
144. Redmon, J.; Divvala, S.; Girshick, R.; Farhadi, A. You Only Look Once: Unified, Real-Time Object Detection. In Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 27–30 June 2016; pp. 779–788. [\[CrossRef\]](#)
145. Liu, W.; Anguelov, D.; Erhan, D.; Szegedy, C.; Reed, S.; Fu, C.Y.; Berg, A.C. SSD: Single Shot MultiBox Detector. In *Computer Vision—ECCV 2016*; Springer International Publishing: Cham, Switzerland, 2016; pp. 21–37. [\[CrossRef\]](#)

146. International Organization for Standardization, *ISO 26262-1:2018 — Road vehicles — Functional safety — Part 1: Vocabulary*, International Organization for Standardization (ISO): Geneva, Switzerland, 2018. Available online: <https://www.iso.org/obp/ui/en/#iso:std:iso:26262:-1:ed-2:v1:en> (accessed on 30 June 2025).
147. Grigorescu, S.; Trasnea, B.; Cocias, T.; Macesanu, G. A survey of deep learning techniques for autonomous driving. *J. Field Robot.* **2019**, *37*, 362–386. [\[CrossRef\]](#)
148. Ma, Y.; Wang, Z.; Yang, H.; Yang, L. Artificial intelligence applications in the development of autonomous vehicles: A survey. *IEEE/CAA J. Autom. Sin.* **2020**, *7*, 315–329. [\[CrossRef\]](#)
149. Chen, J.; Zhang, Y.; Wu, J.; Cheng, W.; Zhu, Q. SOC estimation for lithium-ion battery using the LSTM-RNN with extended input and constrained output. *Energy* **2023**, *262*, 125375. [\[CrossRef\]](#)
150. Kim, M.; Lee, S.; Ha, J.; Lee, H. Make Your Autonomous Mobile Robot on the Sidewalk Using the Open-Source LiDAR SLAM and Autoware. *IEEE Trans. Intell. Veh.* **2024**, 1–12. [\[CrossRef\]](#)
151. Shi, W.; Cao, J.; Zhang, Q.; Li, Y.; Xu, L. Edge Computing: Vision and Challenges. *IEEE Internet Things J.* **2016**, *3*, 637–646. [\[CrossRef\]](#)
152. Zhao, R.; Yan, R.; Chen, Z.; Mao, K.; Wang, P.; Gao, R.X. Deep learning and its applications to machine health monitoring. *Mech. Syst. Signal Process.* **2019**, *115*, 213–237. [\[CrossRef\]](#)
153. Sharma, H.; Haque, A.; Blaabjerg, F. Machine learning in wireless sensor networks for smart cities: A survey. *Electronics* **2021**, *10*, 1012. [\[CrossRef\]](#)
154. Kamilaris, A.; Prenafeta-Boldú, F.X. Deep learning in agriculture: A survey. *Comput. Electron. Agric.* **2018**, *147*, 70–90. [\[CrossRef\]](#)
155. Wang, Z.; Hong, T.; Piette, M.A. Building thermal load prediction through shallow machine learning and deep learning. *Appl. Energy* **2020**, *263*, 114683. [\[CrossRef\]](#)
156. Wang, J.; Chen, Y.; Hao, S.; Peng, X.; Hu, L. Deep learning for sensor-based activity recognition: A survey. *Pattern Recognit. Lett.* **2019**, *119*, 3–11. [\[CrossRef\]](#)
157. Majumder, S.; Deen, M.J. Smartphone Sensors for Health Monitoring and Diagnosis. *Sensors* **2019**, *19*, 2164. [\[CrossRef\]](#)
158. Esteva, A.; Kuprel, B.; Novoa, R.A.; Ko, J.; Swetter, S.M.; Blau, H.M.; Thrun, S. Dermatologist-level classification of skin cancer with deep neural networks. *Nature* **2017**, *542*, 115–118. [\[CrossRef\]](#)
159. Su, H.; Qi, W.; Li, Z.; Chen, Z.; Ferrigno, G.; De Momi, E. Deep neural network approach in EMG-based force estimation for human–robot interaction. *IEEE Trans. Artif. Intell.* **2021**, *2*, 404–412. [\[CrossRef\]](#)
160. Villani, V.; Pini, F.; Leali, F.; Secchi, C. Survey on human–robot collaboration in industrial settings: Safety, intuitive interfaces and applications. *Mechatronics* **2018**, *55*, 248–266. [\[CrossRef\]](#)
161. Weimer, D.; Scholz-Reiter, B.; Shpitalni, M. Design of deep convolutional neural network architectures for automated feature extraction in industrial inspection. *CIRP Ann.* **2016**, *65*, 417–420. [\[CrossRef\]](#)
162. Lee, J.; Bagheri, B.; Kao, H.A. A Cyber-Physical Systems architecture for Industry 4.0-based manufacturing systems. *Manuf. Lett.* **2015**, *3*, 18–23. [\[CrossRef\]](#)
163. Bogue, R. Robots in the laboratory: A review of applications. *Ind. Robot. Int. J.* **2012**, *39*, 113–119. [\[CrossRef\]](#)
164. Su, M.; Su, Z.; Bae, S.H.; Li, J.; Park, K.S. An exploration of shipbuilding price prediction for container ships: An integrated model application of deep learning. *Res. Transp. Bus. Manag.* **2025**, *59*, 101248. [\[CrossRef\]](#)
165. Chen, L.C.; Papandreou, G.; Kokkinos, I.; Murphy, K.; Yuille, A.L. DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs. *IEEE Trans. Pattern Anal. Mach. Intell.* **2018**, *40*, 834–848. [\[CrossRef\]](#)
166. Shan, C.; Zhang, J.; Wang, Y.; Xie, L. Attention-Based End-to-End Speech Recognition on Voice Search. In Proceedings of the 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Calgary, AB, Canada, 15–20 April 2018; pp. 4764–4768. [\[CrossRef\]](#)
167. Carrio, A.; Sampedro, C.; Rodriguez-Ramos, A.; Campoy, P. A Review of Deep Learning Methods and Applications for Unmanned Aerial Vehicles. *J. Sens.* **2017**, *2017*, 3296874. [\[CrossRef\]](#)
168. Ulku, I.; Akagündüz, E. A Survey on Deep Learning-based Architectures for Semantic Segmentation on 2D Images. *Appl. Artif. Intell.* **2022**, *36*, 2032924. [\[CrossRef\]](#)
169. Ramachandra, B.; Jones, M.J.; Vatsavai, R.R. A survey of single-scene video anomaly detection. *IEEE Trans. Pattern Anal. Mach. Intell.* **2020**, *44*, 2293–2312. [\[CrossRef\]](#)
170. Park, J.; Samarakoon, S.; Bennis, M.; Debbah, M. Wireless Network Intelligence at the Edge. *Proc. IEEE* **2019**, *107*, 2204–2239. [\[CrossRef\]](#)
171. Vilela, M.; Hochberg, L.R. Applications of brain-computer interfaces to the control of robotic and prosthetic arms. *Handb. Clin. Neurol.* **2020**, *168*, 87–99. [\[PubMed\]](#)
172. Ouyang, F.; Jiao, P. Artificial intelligence in education: The three paradigms. *Comput. Educ. Artif. Intell.* **2021**, *2*, 100020. [\[CrossRef\]](#)
173. Wang, X.; Han, Y.; Leung, V.C.; Niyato, D.; Yan, X.; Chen, X. Convergence of edge computing and deep learning: A comprehensive survey. *IEEE Commun. Surv. Tutor.* **2020**, *22*, 869–904. [\[CrossRef\]](#)

174. Owens, J.; Houston, M.; Luebke, D.; Green, S.; Stone, J.; Phillips, J. GPU Computing. *Proc. IEEE* **2008**, *96*, 879–899. [[CrossRef](#)]
175. Papernot, N.; McDaniel, P.; Goodfellow, I.; Jha, S.; Celik, Z.B.; Swami, A. Practical Black-Box Attacks against Machine Learning. In Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, Abu Dhabi, United Arab Emirates, 2–6 April 2017; pp. 506–519. [[CrossRef](#)]
176. Dwork, C. Differential privacy: A survey of results. In Proceedings of the International Conference on Theory and Applications of Models of Computation, Xi'an, China, 25–29 April 2008; pp. 1–19.
177. Karlof, C.; Sastry, N.; Wagner, D. TinySec: A link layer security architecture for wireless sensor networks. In Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems, Baltimore, MD, USA, 3–5 November 2004; pp. 162–175.
178. Casado, L.; Tsigas, P. Contikisec: A secure network layer for wireless sensor networks under the contiki operating system. In *Proceedings of the Nordic Conference on Secure IT Systems*; Springer: Berlin/Heidelberg, Germany, 2009; pp. 133–147.
179. Infrastructure, L.E.C.; Layer, M.L.P. IEEE standard for low-rate wireless networks. *IEEE Stand* **2015**, *2015*, 1–708.
180. Maji, S.; Banerjee, U.; Chandrakasan, A.P. Leaky nets: Recovering embedded neural network models and inputs through simple power and timing side-channels—Attacks and defenses. *IEEE Internet Things J.* **2021**, *8*, 12079–12092. [[CrossRef](#)]
181. Liu, Y.; Wei, L.; Luo, B.; Xu, Q. Fault injection attack on deep neural network. In Proceedings of the 2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), Irvine, CA, USA, 13–16 November 2017; pp. 131–138.
182. Barabino, B.; Lai, C.; Olivo, A. Fare evasion in public transport systems: A review of the literature. *Public Transp.* **2020**, *12*, 27–88. [[CrossRef](#)]
183. Wang, Y.; Yao, Q.; Kwok, J.T.; Ni, L.M. Generalizing from a Few Examples: A Survey on Few-shot Learning. *ACM Comput. Surv.* **2020**, *53*, 1–34. [[CrossRef](#)]
184. Mehrabi, N.; Morstatter, F.; Saxena, N.; Lerman, K.; Galstyan, A. A survey on bias and fairness in machine learning. *ACM Comput. Surv. (CSUR)* **2021**, *54*, 1–35. [[CrossRef](#)]
185. Zuboff, S. *The Age of Surveillance Capitalism: The Fight for a Human Future at the New Frontier of Power*, Edn; PublicAffairs: New York, NY, USA, 2019.
186. Rudin, C. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nat. Mach. Intell.* **2019**, *1*, 206–215. [[CrossRef](#)]
187. Marcus, G. The next decade in AI: Four steps towards robust artificial intelligence. *arXiv* **2020**, arXiv:2002.06177. [[CrossRef](#)]
188. Sebastian, A.; Le Gallo, M.; Khaddam-Aljameh, R.; Eleftheriou, E. Memory devices and applications for in-memory computing. *Nat. Nanotechnol.* **2020**, *15*, 529–544. [[CrossRef](#)]
189. Sheikh, A.M.; Islam, M.R.; Habaebi, M.H.; Zabidi, S.A.; Bin Najeeb, A.R.; Kabbani, A. A survey on edge computing (EC) security challenges: Classification, threats, and mitigation strategies. *Future Internet* **2025**, *17*, 175. [[CrossRef](#)]
190. Li, T.; Sahu, A.K.; Talwalkar, A.; Smith, V. Federated Learning: Challenges, Methods, and Future Directions. *IEEE Signal Process. Mag.* **2020**, *37*, 50–60. [[CrossRef](#)]

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.

Reproduced with permission of copyright owner. Further reproduction
prohibited without permission.