Review

Not peer-reviewed version

Embedded Artificial Intelligence: A Comprehensive Literature Review

Xiaoyuan Huang * , Hongcheng Wang , Shiyin Qin , Su-Kit Tang *

Posted Date: 24 July 2025

doi: 10.20944/preprints202507.1972.v1

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

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, 999078 Macao SAR, China
[2] School of Electrical Engineering and Intelligentization, Dongguan University of Technology, 52300 Dongguan, China
[3] School of Automation Science and Electrical Engineering, Beijing University of Aeronautics and Astronautics, 100000 Beijing, China
[*] Correspondence: huangxy@dgut.edu.cn (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

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. However, this centralized approach often encounters limitations related to latency, bandwidth constraints, data privacy, and energy consumption, impeding their deployment in time-critical and resource-sensitive scenarios [6–8].

The burgeoning demand for real-time processing, low latency, enhanced privacy, and optimized energy efficiency has spurred the emergence of Embedded AI (EAI) [9]. This paradigm shifts 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 [10–12]. 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 [13–15]. 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.

Different from the existing review literature [16–21], this paper explores the field of embedded artificial intelligence in a more comprehensive and in-depth manner, focusing on hardware platforms, software frameworks, lightweight algorithms, and deployment methods. This review provides a meticulous analysis of deployment strategies, comparing on-device and cloud-based inference methods, and offers guidance on their applicability in different scenarios. This aligns with discussions in the literature regarding the balance between computational efficiency and energy consumption in embedded systems [16].

The paper also integrates insights from multiple domains such as autonomous driving and smart homes, which are highlighted as key application areas for EAI [17,18]. 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 [17,19]. 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 [19–23].

The remainder of this paper is organized as follows: section 2 introduces the definition of EAI; section 3 describes the hardware platforms for embedded AI and their selection strategies; section 4 lists the software frameworks; section 5 details traditional and deep learning algorithms for embedded AI, with a particular focus on lightweight algorithms suitable for resource-constrained environments; section 6 covers the deployment process, which is the core stage for specific applications; section 7 showcases the applications of embedded AI; and section 8 discusses the current challenges and future opportunities in the field, before finally concluding the study.

2. Definition of EAI

EAI can be defined as the deployment and execution of AI algorithms on embedded systems [11–13]. 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 [24,25]. 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:

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

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.

If $Resources(Model) = [F_{Model}, M_{Model}, E_{Model}]$, then the constraint can be decomposed into:

$$F_{Model} \leq F_{Limits}, \quad M_{Model} \leq M_{Limits}, \quad E_{Model} \leq E_{Limits} \tag{2}$$

where $F_{Limits}, M_{Limits}, E_{Limits}$ are the maximum allowable values imposed by the hardware on computational complexity (Floating point number operations, 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:

$$\underset{Model}{optimize} \quad [Performance(Model), -Resources(Model)] \tag{3}$$

In this case, it is necessary to trade off performance improvement against the reduction of resource consumption, seeking Pareto frontier solutions. This framework provides a mathematical foundation for model compression (e.g., pruning, quantization), Neural Architecture Search (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 [11,12].

3. Hardware Platforms of EAI

3.1. Major Hardware Platforms of EAI

The rapid development of EAI is placing increasingly stringent requirements on hardware platforms. Selecting the appropriate hardware platform is crucial for successfully deploying EAI solutions, which typically involves a careful trade-off between computing power, power consumption, cost, and the development ecosystem and complexity [26]. 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).

Table 1 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 [27]. They typically integrate CPU cores (such as the ARM Cortex-M series [28]), 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 [29,30]. Their hardware architecture is often optimized to support these AI applications, and many contemporary MCUs integrate hardware features conducive to AI computation [27,31]. Typical hardware representatives in the MCU domain include: STMicroelectronics' STM32 series, which, with its extensive product line and integrated hardware resources, can execute optimized neural network models [32]; NXP's i.MX RT series, 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 [33]. Espressif's ESP32 series, 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 [34].

3.1.2. MPU

MPUs are central to general-purpose computing in embedded systems, typically incorporating CPU cores based on architectures such as ARM [35]. These cores excel at executing complex logical control and sequential tasks and are increasingly undertaking computational responsibilities for edge AI [27,36]. 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 [36]. Modern high-performance MPUs featuring ARM CPU cores (e.g., Cortex-A series [37]) 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 [27,38]. 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 [29]. For instance, in many ARM-based MPUs or SoCs (e.g., the MediaTek MT7986A with Cortex-A53 processor [39], Raspberry Pi 5 with Cortex-A76 processor [40]), 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 [38].

3.1.3. GPU

GPUs possess massively parallel processing capabilities, excelling at handling matrix operations, and are suitable for accelerating deep learning algorithms [27,41]. They are primarily used in AI applications requiring high-performance computing and parallel processing, such as autonomous driving, video analytics, and advanced robotics [42,43]. Mainstream GPU vendors include NVIDIA, AMD, and ARM (Mali). NVIDIA's GPUs are widely used in embedded systems, such as the Jetson series [44,45]; their CUDA and TensorRT toolchains facilitate the development and deployment of AI applications, widely used in robotics, drones, and other fields [46]. Through programming frameworks like CUDA or OpenCL, developers can conveniently leverage GPUs to accelerate AI model training and inference [27,47]. Mobile SoCs like Qualcomm's Snapdragon series (featuring Adreno GPUs) [48] and MediaTek's Dimensity series (often featuring ARM Mali or custom GPUs) [49] 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 [50,51]. It offers a high degree of flexibility and reconfigurability, enabling optimized data paths and parallel computation. Products such as the AMD (formerly Xilinx) Zynq 7000 SoC family [52] and Intel (formerly Altera) Arria 10 SoC FPGAs [53] integrate FPGA's programmable logic with hard-core processors (such as Arm Cortex-A9) 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 [27,56]. 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 [56]. Google's Tensor Processing Unit (TPU) 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 [57]. Huawei's Ascend series of AI processors, which employ the Da Vinci architecture, also fall into the ASIC category, designed for a range of AI workloads [61].

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 Units (NPUs) or, as sometimes termed by vendors, Knowledge Processing Units (KPUs) [27,62]. These integrated AI acceleration units often employ efficient parallel computing architectures, such as Systolic Arrays [63], and are specifically optimized to perform core operations in deep learning models, including matrix multiplication and convolution [27]. 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 [62,64]. For example, Rockchip's RV series SoCs (e.g., RV1126/RV1109) [66] and Kendryte's K230 [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 [27,62].

Table 1. Mainstream EAI hardware platforms.

Platforms models[1]	Manu-facturer	Arch-itecture	Processor	AI Accelerator	Computational Power	Power Consumption[2]	Other Key Features
STM32F [32]	ST Microelectronics	MCU	Cortex-M0/M3/M4/M7/M33 etc.	ART Accelerator/NPU (e.g., STM32U5, H7 with NPU) in some models	Tens to hundreds of DMIPS[3], several GOPS[4] in some models (with NPU)	Low power (mA-level op., µA-level standby)	Rich peripherals, CubeMX ecosystem, Security features (integrated in some models)
i.MX RT [33]	NXP	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, MIPI interface, LCD interface control, EdgeLock security subsystem
ESP32-S3 [34]	Espressif	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 [35–38]	ST Microelectronics	MPU	ARM Cortex-A7, Cortex-M4	No dedicated GPU; AI runs via CPU/CMSIS-NN	A7: Max ∼2000 DMIPS; M4: Max ∼200 DMIPS	<1W	Entry-level Linux or RTOS system design
MT7986A (Filogic 830) [39]	MediaTek	MPU	ARM Cortex-A53	Hardware Network Acceleration Engine, AI Engine (NPU, approx. 0.5 TOPS)	CPU: Up to ∼16000 DMIPS	∼7.5W	High-speed network interface
Raspberry Pi 5 [40]	Raspberry Pi	MPU	Broadcom BCM2712 (quad-core ARM Cortex-A76)	VideoCore VII GPU	∼60-80 GFLOPS[5]	Passive Cooling; 2.55W (Idle); 6.66W (Stress)[6]	Graphics performance and general-purpose computing capability
Jetson Orin NX [44][7]	NVIDIA	GPU	ARM Cortex-A78AE	NVIDIA Ampere GPU, Tensor Cores	100.0 TOPS (16GB ver.), 70.0 TOPS (8GB ver.)	10-25W	CUDA, TensorRT
Jetson Nano [45]	NVIDIA	GPU	ARM Cortex-A57, NVIDIA Maxwell GPU	NVIDIA Maxwell GPU	0.472 TFLOPs (16-bit floating-point, FP16) / 5 TOPS (INT8, sparse)	5-10W (typically)	CUDA, TensorRT
Zynq 7000 [52]	AMD (Xilinx)	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
Arria 10 [53]	Intel (Altera)	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 [54]	Lattice	FPGA	iCE40 LM (FPGA family)	FPGA fabric for custom accelerators	1280 Logic Cells	Very low power	Small form factor, Low power consumption
Smart Fusion2 [55]	Microchip	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 [57]	Google	ASIC	1x Cortex-X3, 4x Cortex-A715 , 4x 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 [58]	Google	ASIC	ARM 32-bit Cortex-M0+ Microprocessor (controller)	Edge TPU	4.0 TOPS (INT8)	2.0W	USB interface, Plug-and-play
MLU220-SOM [59]	Cambricon	ASIC	4x ARM Cortex-A55	Cambricon MLU (Memory Logic Unit) NPU	16.0 TOPS (INT8)	15W	Edge intelligent SoC module
Atlas 200I DK A2 [60,61]	Huawei	ASIC	4 core @ 1.0 GHz	Ascend AI Processor	8.0 TOPS (INT8), 4.0 TOPS (FP16)	24W	Software and hardware development kit
RDK X3 Module [65]	Horizon Robotics	AI SoC	ARM Cortex-A53 (Application Processors)	Dual-core Bernoulli BPU	5.0 TOPS	10W	Optimized for visual processing
RK3588 [66]	Rockchip	AI SoC	4x Cortex-A76 + 4x Cortex-A55	ARM Mali-G610 MC4, NPU	NPU: 6.0 TOPS	∼8.04W	Supports multi-camera input
RV1126 [66]	Rockchip	AI SoC	ARM Cortex-A7, RISC-V MCU	NPU	2.0 TOPS (NPU)	1.5W	Low power, Optimized for visual processing
K230 [67]	Kendryte (Canmv)	AI SoC	2x C908 (RISC-V), RISC-V MCU	NPU	1.0 TOPS (NPU)	2.0W	Low power, Supports TinyML
AX650N [68]	Axera Semiconductor	AI SoC	ARM Cortex-A55	NPU	72.0 TOPS (INT4), 18.0 TOPS (INT8)	8.0W	Video encoding/decoding, Image processing

[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 2.55W (Idle) and 6.66W (Stress) are specific measurements. Active cooling is often recommended for sustained heavy loads. [7] 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.

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 [27,69].

The selection strategy for EAI hardware platforms is shown in Figure 1. Firstly, application requirements need to be clarified. This includes specific requirements for computational power , power consumption, latency, cost, security, and reliability [70]. 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 [71]. Next, algorithm complexity needs to be evaluated. This involves model type (Convolutional Neural Network, Recurrent Nearal Networks, Transformer), model size, and computational complexity [72]. Lightweight models can run on CPUs, while complex models require GPU, FPGA, or ASIC acceleration [73]. 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 IP licensing fees [74]. 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 [75,76]. For instance, using the Huawei Atlas 200 DK for prototype verification, or reducing hardware performance demands through model quantization and pruning techniques [72,77].

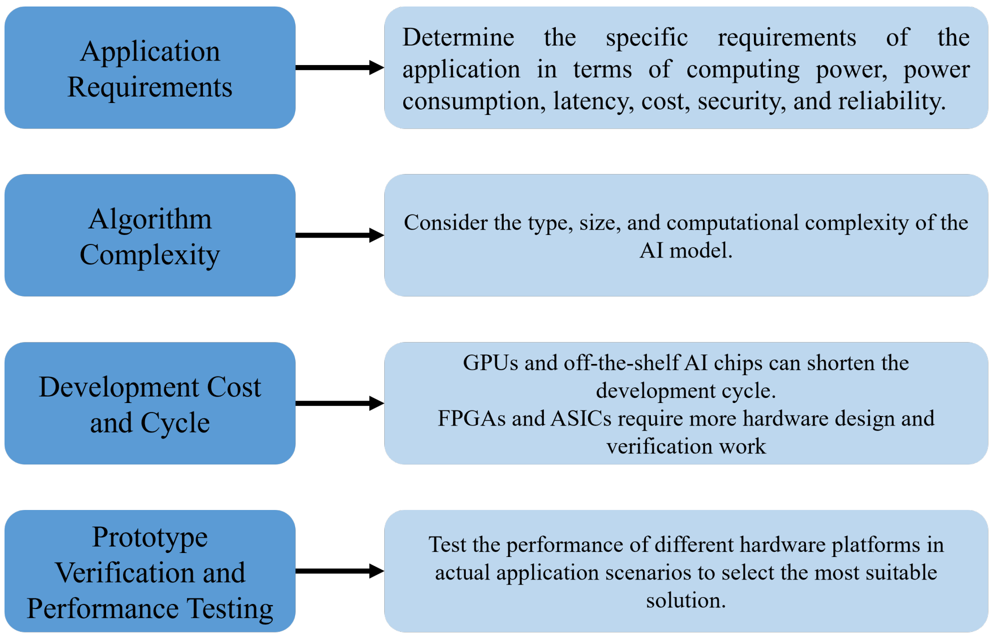
Application Requirements
Determine the specific requirements of the application in terms of computing power, power consumption, latency, cost, security, and reliability.
Algorithm Complexity
Consider the type, size, and computational complexity of the AI model.
Development Cost and Cycle
GPUs and off-the-shelf AI chips can shorten the development cycle.
FPGAs and ASICs require more hardware design and verification work
Prototype Verification and Performance Testing
Test the performance of different hardware platforms in actual application scenarios to select the most suitable solution.

Figure 1. The selection strategy for EAI hardware platforms.

4. Software Frameworks of EAI

Software frameworks play a crucial role in supporting the efficient development and deployment of AI algorithms on embedded systems [78]. These frameworks provide tools and libraries for model optimization, hardware acceleration, and cross-platform compatibility. Table 2 summarizes several key frameworks for the EAI domain, each with its unique features and advantages.

TensorFlow Lite (TFLite) [79]: TensorFlow Lite (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 DSPs. Its key advantage is the TensorFlow Lite for Microcontrollers version, which extends AI capabilities to extremely low-power devices with only kilobyte-level memory.

PyTorch Mobile [80]: 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 using TorchScript or the more 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 [81]: 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 [82] & CMSIS-NN [83] : Arm NN and CMSIS-NN [83] 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 [84] : 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) [85]: 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 [86]: 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 [87–89]: In the field of microcontroller ML (TinyML), extremely lightweight frameworks like uTensor [90] and MicroMLP [91] 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 [92] and SensiML [93] 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 2. Mainstream EAI hardware platforms.

Framework Name	Vendor	Supported Hardware	Development Languages	Use Cases
TensorFlow Lite [79]	Google	iOS, Linux, Microcontrollers, Edge TPU, GPU, DSP, CPU	C++, Python, Java, Swift	Image recognition, Speech recognition, Natural language processing, Sensor data analysis
PyTorch Mobile [80]	Meta (Facebook)	Android, iOS, Linux, CPU	Java, Objective-C, C++	Image recognition, Speech recognition, Natural language processing
ONNX Runtime [81]	Microsoft (and Community/Partners)	CPU, GPU, Dedicated Accelerators (Intel OpenVINO, NVIDIA TensorRT)	C++, C#, Python, JavaScript, Java	Various ML tasks, Model deployment
Arm NN [82]	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 [83]	Arm	ARM Cortex-M	C	Neural network programming for resource-constrained embedded systems
NCNN [84]	Tencent	Android, iOS	C++	Image recognition, Object detection, Face recognition, Mobile AI applications
TVM [85]	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 [86]	Google	Android, iOS, Linux, Web	C++, Python, JavaScript	Real-time media processing, Face detection, Pose estimation, Object tracking
TinyML [87–89]	N/A (Domain/Concept/Community)	Microcontrollers (ARM Cortex-M, RISC-V), Sensors	C, C++, Java	Sensor data analysis, Anomaly detection, Voice activation
uTensor [90]	Open Source (formerly part of MemryX)	Microcontrollers (ARM Cortex-M)	C++	Sensor data analysis, Simple control tasks
MicroMLP [91]	N/A (Concept/Specific library)	Ultra-low-power microcontrollers	C	Simple classification tasks, Gesture recognition
Edge Impulse [92]	Edge Impulse (Company)	Microcontrollers, Linux devices, Sensors	C++, Python , JavaScript	Various embedded ML applications, especially for scenarios requiring rapid prototyping and deployment
SensiML [93]	SensiML (a QuickLogic company)	Microcontrollers, Sensors	C++, Python	Various embedded ML applications, especially for sensor data analysis requiring low power and high efficiency

5. Algorithms of EAI

Selecting appropriate AI algorithms is crucial for achieving optimal performance on embedded systems. Algorithms must be computationally efficient, memory-friendly, and robust. This section details various AI algorithms suitable for embedded systems, including traditional ML algorithms, deep learning algorithms, and model optimization and compression techniques.

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 [94] 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) [95] 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) [96] 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 [97] 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 [98] 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. Convolutional Neural Network

Convolutional Neural Networks (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 [99] laid the foundation for subsequent deeper and more complex networks (such as AlexNet [5], VGG [100], ResNet [101], and Inception [102]). 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. Recurrent Neural Networks

Recurrent Neural Networks (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 [103] and Gated Recurrent Units (GRU) [104] 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. 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 [105]. The success of Transformers quickly extended to other domains; for example, Vision Transformer (ViT) [106] 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.

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 [72,77], Pruning [72,107], and Knowledge Distillation [108].

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 [109]: 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 [110]: Built upon V1 by introducing Inverted Residuals and Linear Bottlenecks, further improving model efficiency and accuracy .
- MobileNetV3 [111]: 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 [112]: 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 [113]: 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 [114]: 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 [115] 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 1x1 kernels to reduce channels) and an expand convolution layer (using 1x1 and 3x3 kernels to increase channels).

5.3.4. EfficientNet Series

- EfficientNetV1 [116]: 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 [117]: 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 [118]: 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 [119], 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 [120], 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 [111-3].

5.3.6. MobileViT

MobileViT (Mobile Vision Transformer) [121]: Proposed by Apple Inc., 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.

6. Deployment of EAI

Successfully deploying an AI model trained on servers or workstations onto resource-constrained embedded devices typically involves a systematic, multi-stage process. This process extends beyond mere model conversion; it is an iterative endeavor involving the co-optimization of algorithms, software, and hardware. As is shown in Figure 2, 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.

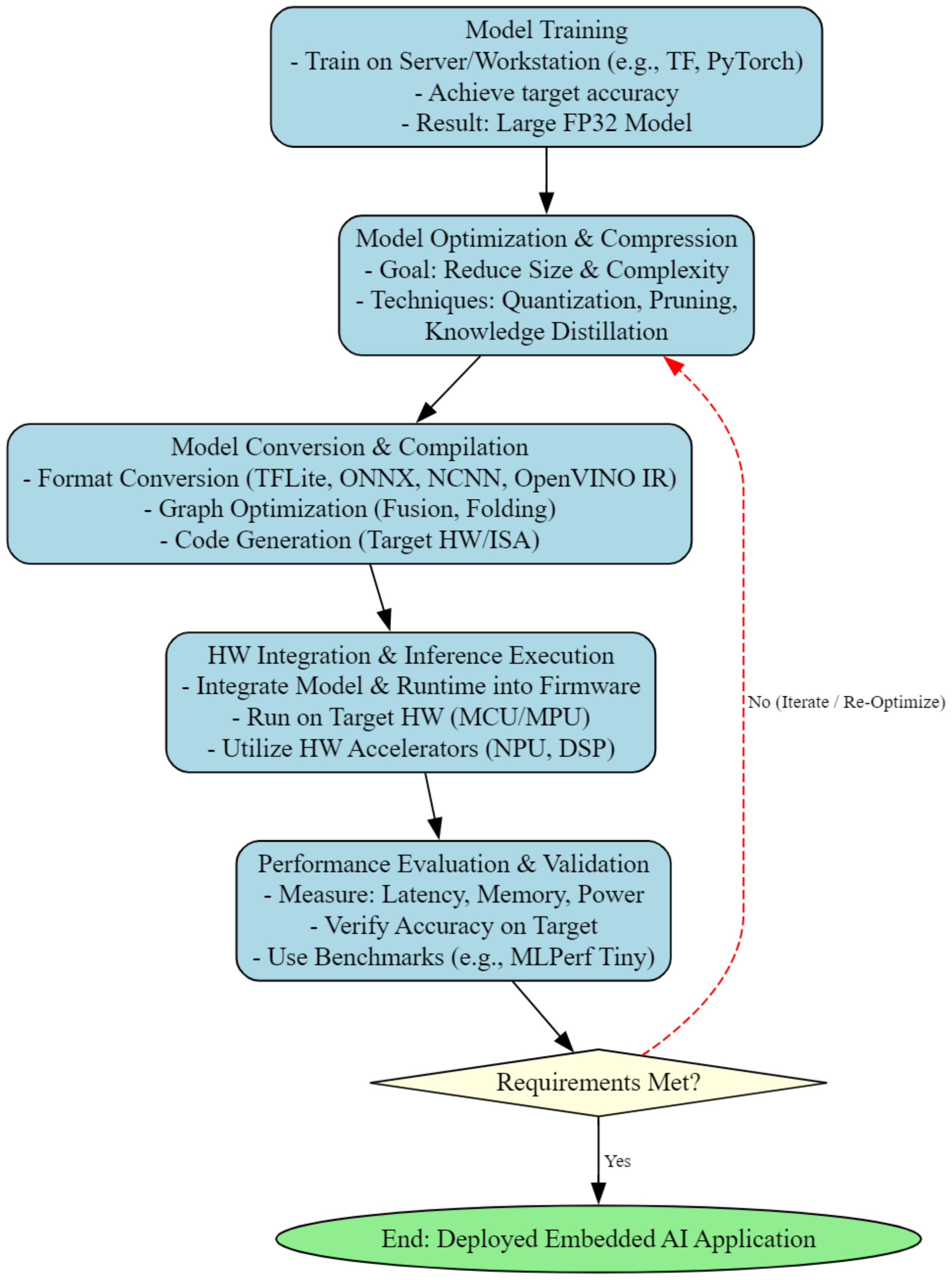
Model Training
- Train on Server/Workstation (e.g., TF, PyTorch)
- Achieve target accuracy
- Result: Large FP32 Model
Model Optimization & Compression
- Goal: Reduce Size & Complexity
- Techniques: Quantization, Pruning, Knowledge Distillation
Model Conversion & Compilation
- Format Conversion (TFLite, ONNX, NCNN, OpenVINO IR)
- Graph Optimization (Fusion, Folding)
- Code Generation (Target HW/ISA)
HW Integration & Inference Execution
- Integrate Model & Runtime into Firmware
- Run on Target HW (MCU/MPU)
- Utilize HW Accelerators (NPU, DSP)
Performance Evaluation & Validation
- Measure: Latency, Memory, Power
- Verify Accuracy on Target
- Use Benchmarks (e.g., MLPerf Tiny)
Requirements Met?
No (Iterate / Re-Optimize)
Yes
End: Deployed Embedded AI Application

Figure 2. The typical deployment workflow.

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 [122] or PyTorch [80]). 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.

6.2. Model Optimization and Compression

This is a crucial step in adapting the model to the embedded environment. Due to the stringent limitations of embedded devices in terms of computational power, memory size, and power consumption, the original trained model must be optimized. This stage aims to significantly reduce the model size, decrease computational complexity, while preserving model accuracy as much as possible. Common optimization techniques include:

- Quantization: Converting FP32 weights and/or activations to lower bit-width representations, e.g., 8-bit integer (INT8) and 16-bit floating-point (FP16), leveraging more efficient integer or half-precision arithmetic units [77,123,124].
- Pruning: Removing redundant weights or structures (e.g., channels, filters) in the model to generate a sparse model, thereby reducing the number of parameters and operations [72].
- Knowledge Distillation: Using a large "teacher" model to guide the training of a smaller "student" model, transferring knowledge to improve the performance of the smaller model [108].

These optimization techniques can be used individually or in combination. The choice of which technique(s) to use depends on the capabilities of the target hardware and the application's requirements for accuracy and latency.

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 [81] into a specific format executable by the target embedded inference engine. For example, TensorFlow Lite Converter [79] generates .tflite files; NCNN [84] uses its conversion toolchain (e.g., onnx2ncnn) to produce its proprietary .param and .bin files; Intel's OpenVINO [125] 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 [126]) 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.

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 TensorFlow Lite for MCUs [79], 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 [79]).
- 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" [83] 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 [127] for optimizing operations on ARM Cortex-M).

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.

- 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/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.
- Benchmarking: Standardized benchmarking tools (e.g., MLPerf Tiny [30]) can be run on the target hardware for fair comparison with other implementations or platforms.
- Iterative Optimization: If the evaluation results fail to meet application requirements (e.g., excessive latency, memory overflow, excessive accuracy degradation), it is necessary to return to previous steps for adjustments. This may involve trying different optimization strategies (e.g., adjusting quantization parameters, using different pruning rates), selecting different lightweight models, adjusting model conversion/compilation options, or even redesigning the model architecture or considering a change of hardware platform. This design-optimize-deploy-evaluate cycle may need to be iterated multiple times to achieve the final goal.

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 YOLO (You Only Look Once) [128] or Single Shot MultiBox Detector (SSD) [129], are widely used for efficient object detection and semantic segmentation. 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 [130]. 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 [131]. 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 [43]. 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 [132].

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 [133]. 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 [134]. 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 [135]. 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 [136]. By analyzing data from thermostats, lighting controllers, occupancy sensors, and security cameras, EAI algorithms can optimize building energy consumption (HVAC control), enhance security levels (anomaly detection), and improve occupant comfort and experience [137].

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 [138]. 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 [139]. 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 [140]. 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 [141].

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 [142]. 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 [143]. 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 [144]. In environments such as warehouses, factories, or hospitals, Autonomous Mobile Robots (AMRs) utilize EAI (combining SLAM, path planning, and perception algorithms) to navigate autonomously, transport materials, and perform inventory tasks, thereby improving internal logistics efficiency [145].

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 [146]. 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 [147]. 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 [148].

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 [149]. 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 manner [150]. 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. 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 AR/VR [151]. 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 [152]. 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 [153]. 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 [154].

8. Challenges and Opportunities

Despite significant progress in EAI (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 [27]. 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 [72,108]. 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 [57,155] . 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 [156]. Furthermore, attention needs to be paid to privacy protection issues, for example, by employing techniques like Differential Privacy to prevent data leakage [157]. Security and privacy issues are not just technical problems but also involve ethical and legal considerations. For example, clear data usage policies need to be established, and users need to be informed about how their data is collected, used, and shared. Additionally, a comprehensive security emergency response mechanism needs to be established to promptly address security incidents.

8.1.5. 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 [158]. 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.

8.2. 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 [159]. 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 [160]. Photonic Computing is also a potential hardware acceleration technology that uses photons for computation, offering advantages of high speed and low power consumption. Furthermore, Neuromorphic Computing architectures, which mimic the structure and function of the human brain and feature high parallelism and low power consumption, can be investigated.
- 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 [122]. 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. 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 [161]. Research into hardware security technologies, such as Trusted Execution Environ-

ments (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 [162]. 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.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.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.T.; project administration, X.H.; funding acquisition, X.H., H.W. and S.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.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

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
ASIC	Application-Specific Integrated Circuit
BSM	Blind Spot Monitoring
CNNs	Convolutional Neural Networks
Cobots	Collaborative robots
CT	Computed Tomography
DFCA	Decoupled Fully Connected
DRL	Deep Reinforcement Learning
EAI	Embedded AI
EMG	electromyography
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
IMUs	Inertial Measurement Units
INT8	8-bit integer
IoT	Internet of Things
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
MRI	Magnetic Resonance Imaging
NAS	Neural Architecture Search
NPUs	Neural Processing Units
NRE	Non-Recurring Engineering
ONNX	Open Neural Network Exchange
RAM	Random Access Memory
RISC-V	Reduced Instruction Set Computing-V
RNNs	Recurrent Neural Networks
SDKs	Software Development Kits
SIMD	Single Instruction Multiple Datastream
SoC	AI System-on-Chip
SSD	Single Shot MultiBox Detector
SVM	Support Vector Machines
TFLite	TensorFlow Lite
TPU	Tensor Processing Unit
TVM	Tensor Virtual Machine
UIB	Universal Inverted Bottleneck

ViT	Vision Transformer
YOLO	You Only Look Once

References

1. Bengio, Y. Learning Deep Architectures for AI. *Foundations and Trends® in Machine Learning* **2009**, *2*, 1–127. https://doi.org/10.1561/2200000006.
2. LeCun, Y.; Bengio, Y.; Hinton, G. Deep learning. *Nature* **2015**, *521*, 436–444. https://doi.org/10.1038/nature14539.
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. https://doi.org/10.1038/nature16961.
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. https://doi.org/10.1109/icassp.2013.6639348.
5. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. ImageNet classification with deep convolutional neural networks. *Communications of the ACM* **2017**, *60*, 84–90. https://doi.org/10.1145/3065386.
6. Satyanarayanan, M. The Emergence of Edge Computing. *Computer* **2017**, *50*, 30–39. https://doi.org/10.1109/mc.2017.9.
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. https://doi.org/10.1109/icccbda51879.2021.9442546.
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, 2022; p. 7–151. https://doi.org/10.1201/9781003338659-2.
9. Moons, B.; Bankman, D.; Verhelst, M., Embedded Deep Neural Networks. In *Embedded Deep Learning*; Springer International Publishing, 2018; p. 1–31. https://doi.org/10.1007/978-3-319-99223-5_1.
10. Warden, P.; Situnayake, D. *Tinyml: Machine learning with tensorflow lite on arduino and ultra-low-power microcontrollers*; O'Reilly Media, 2019.
11. 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. https://doi.org/10.1109/ipsn.2016.7460664.
12. Guo, C.; Ci, S.; Zhou, Y.; Yang, Y. A survey of energy consumption measurement in embedded systems. *IEEE Access* **2021**, *9*, 60516–60530.
13. Bonomi, F.; Milito, R.; Zhu, J.; Addepalli, S. Fog computing and its role in the internet of things. In Proceedings of the Proceedings of the first edition of the MCC workshop on Mobile cloud computing, Helsinki, Finland, 17 August 2012; pp. 13–16. https://doi.org/10.1145/2342509.2342513.
14. 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. https://doi.org/10.1002/inst.12216.
15. Ray, P.P. A survey of IoT cloud platforms. *Future Computing and Informatics Journal* **2016**, *1*, 35–46.
16. Shilpa Kodgire, D.; Roopali Palwe, M.; Ganesh Kharde, M., EMBEDDED AI. In *Futuristic Trends in Artificial Intelligence Volume 3 Book 12*; Iterative International Publishers, Selfypage Developers Pvt Ltd, 2024; p. 137–152. https://doi.org/10.58532/v3biai12p4ch1.
17. Elkhalik, W.A. AI-Driven Smart Homes: Challenges and Opportunities. *Journal of Intelligent Systems and Internet of Things* **2023**, *8*, 54–62. https://doi.org/10.54216/jisiot.080205.
18. Peccia, F.N.; Bringmann, O. Embedded Distributed Inference of Deep Neural Networks: A Systematic Review. *arXiv* **2024**, arXiv:2405.03360.
19. Zhang, Z.; Li, J. A Review of Artificial Intelligence in Embedded Systems. *Micromachines* **2023**, *14*, 897. https://doi.org/10.3390/mi14050897.
20. Kotyal, K.; Nautiyal, P.; Singh, M.; Semwal, A.; Rai, D.; Papnai, G.; Nautiyal, C.T.; G.Malathi.; Krishnaveni, S. Advancements and Challenges in Artificial Intelligence Applications: A Comprehensive Review. *Journal of Scientific Research and Reports* **2024**, *30*, 375–385. https://doi.org/10.9734/jsrr/2024/v30i102465.

21. Serpanos, D.; Ferrari, G.; Nikolakopoulos, G.; Perez, J.; Tauber, M.; Van Baelen, S. Embedded Artificial Intelligence: The ARTEMIS Vision. *Computer* **2020**, *53*, 65–69. https://doi.org/10.1109/mc.2020.3016104.
22. Dunne, R.; Morris, T.; Harper, S. A Survey of Ambient Intelligence. *ACM Computing Surveys* **2021**, *54*, 1–27. https://doi.org/10.1145/3447242.
23. 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. *Malaysian Journal of Science and Advanced Technology* **2024**, p. 141–148. https://doi.org/10.56532/mjsat.v4i2.291.
24. 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, Florida, USA, 5–7 May 2008; pp. 363–369. https://doi.org/10.1109/isorc.2008.25.
25. Marwedel, P. *Embedded System Design*; Springer International Publishing, 2018. https://doi.org/10.1007/978-3-319-56045-8.
26. Lin, H.Y. Embedded Artificial Intelligence: Intelligence on Devices. *Computer* **2023**, *56*, 90–93. https://doi.org/10.1109/mc.2023.3280397.
27. Sze, V.; Chen, Y.H.; Yang, T.J.; Emer, J.S. Efficient Processing of Deep Neural Networks: A Tutorial and Survey. *Proceedings of the IEEE* **2017**, *105*, 2295–2329. https://doi.org/10.1109/jproc.2017.2761740.
28. Yiu, J., Introduction to ARM® Cortex®-M Processors. In *The Definitive Guide to ARM® CORTEX®-M3 and CORTEX®-M4 Processors*; Elsevier, 2014; p. 1–24. https://doi.org/10.1016/b978-0-12-408082-9.00001-4.
29. Ray, P.P. A review on TinyML: State-of-the-art and prospects. *Journal of King Saud University-Computer and Information Sciences* **2022**, *34*, 1595–1623.
30. 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. https://doi.org/10.1109/mm.2020.2974843.
31. 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. https://doi.org/10.1109/acet61898.2024.10730112.
32. STMicroelectronics. STM32Cube.AI-STMicroelectronics-STM32 AI, 2023. (accessed on 30 May 2025).
33. NXP. i.MX RT Crossover MCUs, 2024. (accessed on 27 May 2025).
34. ESP-DL. ESP-DL Introduction, 2023. (accessed on 4 June 2025).
35. Ibrahim, D., Architecture of ARM microcontrollers. In *Arm-Based Microcontroller Multitasking Projects*; Elsevier, 2021; p. 13–32. https://doi.org/10.1016/b978-0-12-821227-1.00002-5.
36. Murshed, M.G.S.; Murphy, C.; Hou, D.; Khan, N.; Ananthanarayanan, G.; Hussain, F. Machine Learning at the Network Edge: A Survey. *ACM Computing Surveys* **2021**, *54*, 1–37. https://doi.org/10.1145/3469029.
37. ARM. Key architectural points of ARM Cortex-A series processors, 2023. (accessed on 4 June 2025).
38. 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. https://doi.org/10.1109/csci46756.2018.00272.
39. Mediatek. MediaTek | Filogic 830 | Premium Wi-Fi 6/6E SoC, 2023. (accessed on 5 June 2025).
40. Raspberry Pi Ltd. raspberry-pi-5-product-brief. (accessed on 3 May 2025).
41. Intel. What Is a GPU?, 2023. (accessed on 30 June 2025).
42. Janai, J.; Güney, F.; Behl, A.; Geiger, A. Computer Vision for Autonomous Vehicles: Problems, Datasets and State of the Art. *Foundations and Trends® in Computer Graphics and Vision* **2020**, *12*, 1–308. https://doi.org/10.1561/0600000079.
43. 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 Systems with Applications* **2021**, *165*, 113816. https://doi.org/10.1016/j.eswa.2020.113816.
44. NVIDIA. NVIDIA Jetson Orin, 2025. (accessed on 15 May 2025).
45. Kurniawan, A., Administering NVIDIA Jetson Nano. In *IoT Projects with NVIDIA Jetson Nano*; Apress, 2020; p. 21–47. https://doi.org/10.1007/978-1-4842-6452-2_3.
46. NVIDIA. Jetson Developer Kits, 2025. (accessed on 15 May 2025).
47. Ashbaugh, B.; Lake, A.; Rovatsou, M. Khronos™ group. In Proceedings of the Proceedings of the 3rd International Workshop on OpenCL (IWOCL '15), New York , United States, 12–13 May 2015; pp. 1–23. https://doi.org/10.1145/2791321.2791337.
48. Pipes, M.A. Qualcomm snapdragon "bullet train". In Proceedings of the ACM SIGGRAPH 2015 Computer Animation Festival. ACM, 2015, SIGGRAPH '15, p. 124–125. https://doi.org/10.1145/2745234.2746856.

49. MediaTek. MediaTek | Dimensity | 5G Smartphone Chips, n.d. (accessed on 16 June 2025).
50. Wang, T.; Wang, C.; Zhou, X.; Chen, H. A survey of FPGA based deep learning accelerators: Challenges and opportunities. *arXiv* **2018**, arXiv:1901.04988.
51. Cheremisinov, D.I. Protecting intellectual property in FPGA Xilinx design. *Prikladnaya diskretnaya matematika* **2014**, p. 110–118. https://doi.org/10.17223/20710410/24/10.
52. Limited, A.E. AC7Z020 SoM with AMD Zynq™ 7000 SoC XC7Z020, 2023. (accessed on 6 June 2025).
53. Intel. Arria® 10 FPGA and SoC FPGA, 2023. (accessed on 6 June 2025).
54. Lattice Semiconductor. iCE40 LP/HX, 2025. (accessed on 30 June 2025).
55. Microchip. SmartFusion® 2 FPGAs, 2025. (accessed on 30 June 2025).
56. 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, Massachusetts, USA, 22–24 September 2020; pp. 1–12. https://doi.org/10.1109/hpec43674.2020.9286149.
57. Jouppi, N.; Young, C.; Patil, N.; Patterson, D. Motivation for and Evaluation of the First Tensor Processing Unit. *IEEE Micro* **2018**, *38*, 10–19. https://doi.org/10.1109/mm.2018.032271057.
58. Google LLC. USB Accelerator datasheet, 2025. (accessed on 16 July 2025).
59. cambricon. MLU220-SOM, 2025. (accessed on 16 July 2025).
60. Huawei. Atlas 200I DK A2 Developer Kit, 2025. (accessed on 16 July 2025).
61. Xiao, Y.; Wang, Z. AIbench: a tool for benchmarking Huawei ascend AI processors. *CCF Transactions on High Performance Computing* **2024**, *6*, 115–129. https://doi.org/10.1007/s42514-024-00187-x.
62. Chen, Y.H.; Krishna, T.; Emer, J.S.; Sze, V. Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks. *IEEE Journal of Solid-State Circuits* **2017**, *52*, 127–138. https://doi.org/10.1109/jssc.2016.2616357.
63. 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.
64. Shafiee, A.; Nag, A.; Muralimanohar, N.; Balasubramonian, 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 Computer Architecture News* **2016**, *44*, 14–26. https://doi.org/10.1145/3007787.3001139.
65. Horizon Robotics. RDK X3 Robot Development Kit, 2023. (accessed on 6 June 2025).
66. Rockchip Electronics Co., L. Rockchip RV11 Series, 2023. (accessed on 6 June 2025).
67. Kendryte. Kendryte Developer Community-Product Center, n.d. (accessed on 6 June 2025).
68. Axera Semiconductor. AX650N, 2025. (accessed on 6 June 2025).
69. Bavikadi, S.; Dhavlle, 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 Design & Test* **2022**, *39*, 91–116. https://doi.org/10.1109/MDAT.2022.3161126.
70. 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.
71. 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, Korea, 6–8 March 2014; pp. 241-246. https://doi.org/10.1109/wf-iot.2014.6803166.
72. 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 https://doi.org/10.1109/hotchips.2016.7936226.
73. Blaiech, A.G.; Ben Khalifa, K.; Valderrama, C.; Fernandes, M.A.; Bedoui, M.H. A Survey and Taxonomy of FPGA-based Deep Learning Accelerators. *Journal of Systems Architecture* **2019**, *98*, 331–345. https://doi.org/10.1016/j.sysarc.2019.01.007.
74. Presutto, M. Current AI Trends: Hardware and Software Accelerators. *Royal Institute of Technology* **2018**, pp. 1–21.
75. 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, 2017; p. 49–50. https://doi.org/10.51202/9783181022993-49.
76. 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.

77. 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. https://doi.org/10.1109/cvpr.2018.00286.
78. Banbury, C.R.; Reddi, V.J.; Lam, M.; Fu, W.; Fazel, A.; Holleman, J.; Huang, X.; Hurtado, R.; Kanter, D.; Lokhmotov, A.; et al. Benchmarking tinyml systems: Challenges and direction. *arXiv* **2020**, arXiv:2003.04821.
79. Zaman, F. *TensorFlow Lite for Mobile Development: Deploy Machine Learning Models on Embedded and Mobile Devices*; Apress, 2020. https://doi.org/10.1007/978-1-4842-6666-3.
80. Alla, S.; Adari, S.K., Introduction to Deep Learning. In *Beginning Anomaly Detection Using Python-Based Deep Learning*; Apress, 2019; p. 73–122. https://doi.org/10.1007/978-1-4842-5177-5_3.
81. Microsoft Corporation. ONNX, n.d. (accessed on 7 June 2025).
82. Ltd., A. Arm NN SDK - Efficient ML for Arm CPUs, GPUs, NPUs, 2023. (accessed on 7 June 2025).
83. ARM-software. GitHub - ARM-software/CMSIS-NN: CMSIS-NN Library, 2023. (accessed on 7 June 2025).
84. Ni, H.; The ncnn contributors. ncnn, 2017. (accessed on 7 June 2025).
85. 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, *11*, 20.
86. 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.
87. mit-han-lab. Tiny Machine Learning[website], 2024. (accessed on 7 June 2025).
88. Sanchez-Iborra, R.; Skarmeta, A.F. TinyML-Enabled Frugal Smart Objects: Challenges and Opportunities. *IEEE Circuits and Systems Magazine* **2020**, *20*, 4–18. https://doi.org/10.1109/mcas.2020.3005467.
89. Shafique, M.; Theocharides, 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 https://doi.org/10.1109/dac18074.2021.9586232.
90. uTensor. uTensor - TinyML AI Inference Library, 2023.
91. jczic. GitHub - jczic/MicroMLP: A micro neural network multilayer perceptron for MicroPython (used on ESP32 and Pycom modules), n.d.
92. Edgeimpulse, Inc. The Edge AI Platform, 2025. (accessed on 7 June 2025).
93. SensiML. SensiML - Making Sensor Data Sensible, 2020. (accessed on 7 June 2025).
94. Quinlan, J.R. Induction of decision trees. *Machine Learning* **1986**, *1*, 81–106. https://doi.org/10.1007/bf00116251.
95. Cortes, C.; Vapnik, V. Support-vector networks. *Machine Learning* **1995**, *20*, 273–297. https://doi.org/10.1007/bf00994018.
96. Cover, T.; Hart, P. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory* **1967**, *13*, 21–27. https://doi.org/10.1109/tit.1967.1053964.
97. 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. https://doi.org/10.1109/iceconf57129.2023.10083573.
98. Breiman, L. Random forests. *Machine learning* **2001**, *45*, 5–32.
99. Lecun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE* **1998**, *86*, 2278–2324. https://doi.org/10.1109/5.726791.
100. Simonyan, K.; Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv* **2014**, arXiv:1409.1556.
101. 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. https://doi.org/10.1109/cvpr.2016.90.
102. 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. https://doi.org/10.1109/cvpr.2015.7298594.
103. Hochreiter, S.; Schmidhuber, J. Long Short-Term Memory. *Neural Computation* **1997**, *9*, 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735.

104. 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.
105. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, Ł.; Polosukhin, I. Attention is all you need. *Advances in neural information processing systems* **2017**, *30*.
106. 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.
107. LeCun, Y.; Denker, J.; Solla, S. Optimal brain damage. *Advances in neural information processing systems* **1989**, *2*.
108. Hinton, G.; Vinyals, O.; Dean, J. Distilling the knowledge in a neural network. *arXiv* **2015**, arXiv:1503.02531.
109. 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.
110. 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. https://doi.org/10.1109/cvpr.2018.00474.
111. 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, Korea, 27 October 2019 to 2 November 2019; pp. 1314–1324. https://doi.org/10.1109/iccv.2019.00140.
112. Qin, D.; Leichner, 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, 2024; p. 78–96. https://doi.org/10.1007/978-3-031-73661-2_5.
113. 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. https://doi.org/10.1109/cvpr.2018.00716.
114. 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, 2018; p. 122–138. https://doi.org/10.1007/978-3-030-01264-9_8.
115. 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.
116. Tan, M.; Le, Q. Efficientnet: Rethinking model scaling for convolutional neural networks. In Proceedings of the International conference on machine learning, Long Beach, California, USA, 9-15 June 2019; pp. 6105–6114.
117. 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.
118. 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. https://doi.org/10.1109/cvpr42600.2020.00165.
119. Tang, Y.; Han, K.; Guo, J.; Xu, C.; Xu, C.; Wang, Y. GhostNetv2: Enhance cheap operation with long-range attention. *Advances in Neural Information Processing Systems* **2022**, *35*, 9969–9982.
120. Liu, Z.; Hao, Z.; Han, K.; Tang, Y.; Wang, Y. Ghostnetv3: Exploring the training strategies for compact models. *arXiv* **2024**, arXiv:2404.11202.
121. Mehta, S.; Rastegari, M. Mobilevit: light-weight, general-purpose, and mobile-friendly vision transformer. *arXiv* **2021**, arXiv:2110.02178.
122. 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, 2022; p. 71–90. https://doi.org/10.1201/9781003277224-4.
123. 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.
124. 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, 2022; p. 291–326. https://doi.org/10.1201/9781003162810-13.

125. Vasudevan, S.K.; Pulari, S.R.; Vasudevan, S., Intel OpenVino: A Must-Know Deep Learning Toolkit. In *Deep Learning*; Chapman and Hall/CRC, 2021; p. 245–260. https://doi.org/10.1201/9781003185635-11.
126. 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, *11*, 20.
127. Lai, L.; Suda, N.; Chandra, V. Cmsis-nn: Efficient neural network kernels for arm cortex-m cpus. *arXiv* **2018**, arXiv:1801.06601.
128. 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. https://doi.org/10.1109/cvpr.2016.91.
129. 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, 2016; p. 21–37. https://doi.org/10.1007/978-3-319-46448-0_2.
130. Grigorescu, S.; Trasnea, B.; Cocias, T.; Macesanu, G. A survey of deep learning techniques for autonomous driving. *Journal of Field Robotics* **2019**, *37*, 362–386. https://doi.org/10.1002/rob.21918.
131. Ma, Y.; Wang, Z.; Yang, H.; Yang, L. Artificial intelligence applications in the development of autonomous vehicles: A survey. *IEEE/CAA Journal of Automatica Sinica* **2020**, *7*, 315–329.
132. 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 Transactions on Intelligent Vehicles* **2024**.
133. Shi, W.; Cao, J.; Zhang, Q.; Li, Y.; Xu, L. Edge Computing: Vision and Challenges. *IEEE Internet of Things Journal* **2016**, *3*, 637–646. https://doi.org/10.1109/jiot.2016.2579198.
134. Zhao, R.; Yan, R.; Chen, Z.; Mao, K.; Wang, P.; Gao, R.X. Deep learning and its applications to machine health monitoring. *Mechanical Systems and Signal Processing* **2019**, *115*, 213–237. https://doi.org/10.1016/j.ymssp.2018.05.050.
135. Sharma, H.; Haque, A.; Blaabjerg, F. Machine learning in wireless sensor networks for smart cities: a survey. *Electronics* **2021**, *10*, 1012.
136. Kamilaris, A.; Prenafeta-Boldú, F.X. Deep learning in agriculture: A survey. *Computers and Electronics in Agriculture* **2018**, *147*, 70–90. https://doi.org/10.1016/j.compag.2018.02.016.
137. Wang, Z.; Hong, T.; Piette, M.A. Building thermal load prediction through shallow machine learning and deep learning. *Applied Energy* **2020**, *263*, 114683. https://doi.org/10.1016/j.apenergy.2020.114683.
138. Wang, J.; Chen, Y.; Hao, S.; Peng, X.; Hu, L. Deep learning for sensor-based activity recognition: A survey. *Pattern Recognition Letters* **2019**, *119*, 3–11. https://doi.org/10.1016/j.patrec.2018.02.010.
139. Majumder, S.; Deen, M.J. Smartphone Sensors for Health Monitoring and Diagnosis. *Sensors* **2019**, *19*, 2164. https://doi.org/10.3390/s19092164.
140. 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. https://doi.org/10.1038/nature21056.
141. 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 Transactions on Artificial Intelligence* **2021**, *2*, 404–412.
142. 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. https://doi.org/10.1016/j.mechatronics.2018.02.009.
143. Weimer, D.; Scholz-Reiter, B.; Shpitalni, M. Design of deep convolutional neural network architectures for automated feature extraction in industrial inspection. *CIRP Annals* **2016**, *65*, 417–420. https://doi.org/10.1016/j.cirp.2016.04.072.
144. Lee, J.; Bagheri, B.; Kao, H.A. A Cyber-Physical Systems architecture for Industry 4.0-based manufacturing systems. *Manufacturing Letters* **2015**, *3*, 18–23. https://doi.org/10.1016/j.mfglet.2014.12.001.
145. Bogue, R. Robots in the laboratory: a review of applications. *Industrial Robot: An International Journal* **2012**, *39*, 113–119. https://doi.org/10.1108/01439911211203382.
146. 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 Transactions on Pattern Analysis and Machine Intelligence* **2018**, *40*, 834–848. https://doi.org/10.1109/tpami.2017.2699184.
147. 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, Canad, 15-20 April 2018; pp. 4764–4768. https://doi.org/10.1109/icassp.2018.8462492.

148. Carrio, A.; Sampedro, C.; Rodriguez-Ramos, A.; Campoy, P. A Review of Deep Learning Methods and Applications for Unmanned Aerial Vehicles. *Journal of Sensors* **2017**, *2017*, 1–13. https://doi.org/10.1155/2017/3296874.
149. Ulku, I.; Akagündüz, E. A Survey on Deep Learning-based Architectures for Semantic Segmentation on 2D Images. *Applied Artificial Intelligence* **2022**, *36*. https://doi.org/10.1080/08839514.2022.2032924.
150. Ramachandra, B.; Jones, M.J.; Vatsavai, R.R. A survey of single-scene video anomaly detection. *IEEE transactions on pattern analysis and machine intelligence* **2020**, *44*, 2293–2312.
151. Park, J.; Samarakoon, S.; Bennis, M.; Debbah, M. Wireless Network Intelligence at the Edge. *Proceedings of the IEEE* **2019**, *107*, 2204–2239. https://doi.org/10.1109/jproc.2019.2941458.
152. Vilela, M.; Hochberg, L.R. Applications of brain-computer interfaces to the control of robotic and prosthetic arms. *Handbook of clinical neurology* **2020**, *168*, 87–99.
153. Ouyang, F.; Jiao, P. Artificial intelligence in education: The three paradigms. *Computers and Education: Artificial Intelligence* **2021**, *2*, 100020. https://doi.org/10.1016/j.caeai.2021.100020.
154. Wang, X.; Han, Y.; Leung, V.C.; Niyato, D.; Yan, X.; Chen, X. Convergence of edge computing and deep learning: A comprehensive survey. *IEEE communications surveys & tutorials* **2020**, *22*, 869–904.
155. Owens, J.; Houston, M.; Luebke, D.; Green, S.; Stone, J.; Phillips, J. GPU Computing. *Proceedings of the IEEE* **2008**, *96*, 879–899. https://doi.org/10.1109/jproc.2008.917757.
156. Papernot, N.; McDaniel, P.; Goodfellow, I.; Jha, S.; Celik, Z.B.; Swami, A. Practical Black-Box Attacks against Machine Learning. In Proceedings of the 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. https://doi.org/10.1145/3052973.3053009.
157. 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.
158. Wang, Y.; Yao, Q.; Kwok, J.T.; Ni, L.M. Generalizing from a Few Examples: A Survey on Few-shot Learning. *ACM Computing Surveys* **2020**, *53*, 1–34. https://doi.org/10.1145/3386252.
159. Marcus, G. The next decade in AI: four steps towards robust artificial intelligence. *arXiv* **2020**, arXiv:2002.06177.
160. Sebastian, A.; Le Gallo, M.; Khaddam-Aljameh, R.; Eleftheriou, E. Memory devices and applications for in-memory computing. *Nature Nanotechnology* **2020**, *15*, 529–544. https://doi.org/10.1038/s41565-020-0655-z.
161. Li, T.; Sahu, A.K.; Talwalkar, A.; Smith, V. Federated Learning: Challenges, Methods, and Future Directions. *IEEE Signal Processing Magazine* **2020**, *37*, 50–60. https://doi.org/10.1109/msp.2020.2975749.
162. 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, 2022; p. 7–151. https://doi.org/10.1201/9781003338659-2.