

Engineering Systems and Intelligent Technologies ESIT

ISSN: 3071-253X/© 2026, The Authors. (CC BY 4.0) License.

Journal Homepage

<https://pub.scientificirg.com/index.php/ESIT>



Cross-Layer Hardware and Software Co-Design and Evaluation of Embedded Intelligence

Ola Farid^{a,1}

^a Computer Science Department, Faculty of Science, Beni-Suef University, Beni-Suef City, 62511, Egypt; E-mail: ola3131313@gmail.com

ABSTRACT

Embedded intelligence places sensing, inference, adaptation, and decision making inside physical systems limited by energy, memory, latency, cost, and reliability. This narrative review examines the field along four connected axes: hardware and software co-design, computational substrates, optimization at the model, mapping, and system levels, and applications in industrial automation, automotive electronics, healthcare monitoring, and the Internet of Things. Timing, safety, and security are treated as requirements that cut across all four axes. The review separates embedded intelligence from constrained on-device machine learning and from edge artificial intelligence, and it frames deployment as a cross-layer problem in which model quality, hardware mapping, timing, energy, security, and adaptation constrain one another. Two literature-based case studies, one on a field-programmable gate array and one on distributed microcontrollers, show how a gain at one layer produces a cost at another. A recomputation of the efficiency ratios reported in one of the studies shows why measurement boundaries and metric definitions must be declared. Six research priorities are derived from limitations that recur in the reviewed sources: lifecycle energy accounting, assurance of learned components, security of distributed deployments, reproducible benchmarking, portable deployment toolchains, and concept drift after deployment. Each priority is paired with a research direction and evaluation criteria for later studies. The evidence consists mainly of surveys, analytical studies, and laboratory implementations, so quantitative values are reported as stated in the sources, recomputed values are labeled as such, and approaches are not ranked. The proposed formulations and criteria are conceptual.

PAPER INFORMATION

HISTORY

Received: 16 August 2026

Revised: 3 October 2026

Accepted: 18 October 2026

Online: 28 October 2026

MSC

68M15; 68T07; 68R10;
94A60

KEYWORDS

Cross-Layer;
Hardware and Software
Co-design;
Embedded Intelligence;
Real-time Embedded Systems;
Internet of Things.

1. INTRODUCTION

Embedded systems that sense the environment, interpret data, and act within strict limits on time, energy, and cost underpin modern industrial, automotive, healthcare, and consumer technology. In many constrained deployments, perception and decision functions are placed close to the sensor or actuator, which reduces communication latency, communication energy, and the exposure of raw data. This review uses the term embedded intelligence for this practice. The practice rests on the deliberate integration of hardware and software, a discipline documented in the co-design surveys of the 1990s [1, 2] and reshaped over the last decade by learned models small enough to run on microcontrollers [3, 4].

Local processing is motivated by three principal factors. First, sending raw sensor data to a remote server adds latency that some control loops in robotics, automotive safety, and industrial automation cannot tolerate [5]. Second, transmitting raw

¹Corresponding author at Computer Science Department, Faculty of Science, Beni-Suef University, Beni-Suef City, 62511, Egypt;
E-mail: ola3131313@gmail.com

data can impose substantial communication energy and latency costs, particularly for battery-powered or intermittently connected nodes [5, 6]. Third, privacy requirements, especially in healthcare and consumer electronics, favor processing near the data source over centralized aggregation of sensitive records [6].

Because related terms are used inconsistently in the literature, the following definitions are adopted throughout this review. Embedded intelligence is the umbrella concept for sensing, inference, adaptation, and decision making integrated into a physical embedded system. Constrained on-device machine learning is the subset in which learned models run under severe limits on memory, computation, and energy, commonly on microcontroller-class devices [7]. The label TinyML, which appears in the titles of several works discussed in this review, names this same subset and does not denote a separate field. Edge artificial intelligence (AI) is broader than constrained on-device learning, covering gateways, mobile processors, and edge servers. Edge intelligence adds a distributed architectural view: AI applied to edge computing problems, with models trained and executed at the edge [8]. Not every edge AI system runs on constrained devices, and not every embedded intelligence system uses machine learning. Control algorithms without learned components fall within scope only where they shape the partitioning of hardware and software. Cloud baselines and mobile graphics processing units (GPUs) serve only as reference points for comparison, not as deployment targets within scope.

Meeting these demands requires progress at three levels. At the hardware level, substrates ranging from low-power microcontrollers to field-programmable gate arrays (FPGAs), application-specific integrated circuits (ASICs), and neuromorphic processors must deliver adequate throughput within tight power budgets, and reported power values depend on the measurement boundary, duty cycle, and workload (Section 9.4). At the software level, models and control algorithms must be compressed, quantized, and scheduled to fit within kilobytes of memory and short compute windows [9, 10]. At the system level, the partition of function between hardware and software, which the co-design surveys treat as a central design problem [1, 2], the choice of communication protocols, and the verification of timing and safety properties remain open engineering problems that extend the classical co-design questions.

Earlier surveys address individual layers of this stack. Section 2 compares representative related works in **Table 1**, which gives the scope of each work and the integration emphasized by this review. The co-design surveys of De Micheli and Gupta [1] and of Wolf [2] predate learned components and serve as foundations. Reviews of constrained machine learning concentrate on methods, platforms, workflows, and models [11, 3, 7, 9, 4, 10]. Deng et al. [8] treat edge intelligence at network scale and divide it into AI for edge and AI on edge. Other works address neuromorphic computing [12], adaptation to concept drift [13], and the lifecycle impact of constrained learning [14]. Cross-layer and co-design workflows already appear in this literature [7, 4].

The central premise of this review is that embedded intelligence should not be evaluated as a model quality problem alone. A model that improves accuracy may exceed a memory ceiling, invalidate a timing bound, increase update risk, or require communication that undermines determinism. Conversely, a hardware or deployment gain may reduce energy or memory use while degrading accuracy, robustness, or assurance. The review therefore treats embedded intelligence as a cross-layer design and evaluation problem.

The review is organized along four axes: co-design, computational substrates, optimization, and applications. The axes follow the path from design principles to deployed systems, and this decomposition is not claimed to be the only possible one. Timing, safety, and security requirements cut across all four axes. Five research questions operationalize the axes and the cross-cutting requirements.

1. How do classical co-design principles extend to systems with learned components? (Section 3)
2. Which computational substrates are used for embedded intelligence, and how do the trade-offs compare? (Section 4)
3. Which optimization techniques adapt models to those substrates, and at which stage does each apply? (Section 5)
4. How do timing, safety, and security requirements constrain these choices? (Section 6, with literature-based case studies in Section 7)
5. How do these choices appear in four application domains, and which challenges remain open? (Sections 8 and 9)

The stated scopes of the earlier works do not frame the coupling of end-to-end timing, assurance, and update security with memory, energy, and adaptation, and this review addresses that coupling by tracing a chain from model choice through hardware mapping, timing, energy, assurance, and adaptation. Three contributions follow. First, it fixes the relation among embedded intelligence, constrained on-device machine learning, and edge AI, and it organizes the reviewed literature by co-design, substrates, optimization, and applications. Second, it examines how model and deployment choices interact with memory, timing, energy, safety, and security, using scoped analytical formulations, comparative tables, and two literature-based case studies that separate reported findings from interpretation. Third, it proposes six research priorities with evaluation criteria and distinguishes limitations reported in the literature from priorities derived in this review. The review seeks neither exhaustive coverage nor a quantitative ranking across heterogeneous studies.

The remainder of the manuscript follows the order of the research questions. Section 2 describes the review methodology. Section 3 reviews hardware and software co-design, and Section 4 surveys computational substrates. Section 5 examines optimization and deployment. Section 6 discusses timing, safety, and security in cyber-physical systems, and Section 7 presents two literature-based case studies. Section 8 reviews application domains, Section 9 presents six research priorities, and Section 10 concludes the review.

2. REVIEW METHODOLOGY

This article is a narrative review based on a targeted literature search and does not follow a systematic review protocol. It therefore does not claim exhaustive coverage or protocol-based reproducibility, and screening counts are not reported. **Figure 1** summarizes the workflow. The search covered IEEE Xplore, the ACM Digital Library, ScienceDirect, SpringerLink, and MDPI, which host most of the venues represented in the article, along with the proceedings sites of NeurIPS, ICML, and USENIX Security for machine learning and security literature. The publication period spans from 1994, the year of the earliest survey included, to 2026. The final search and verification of the sources were completed in October 2026.

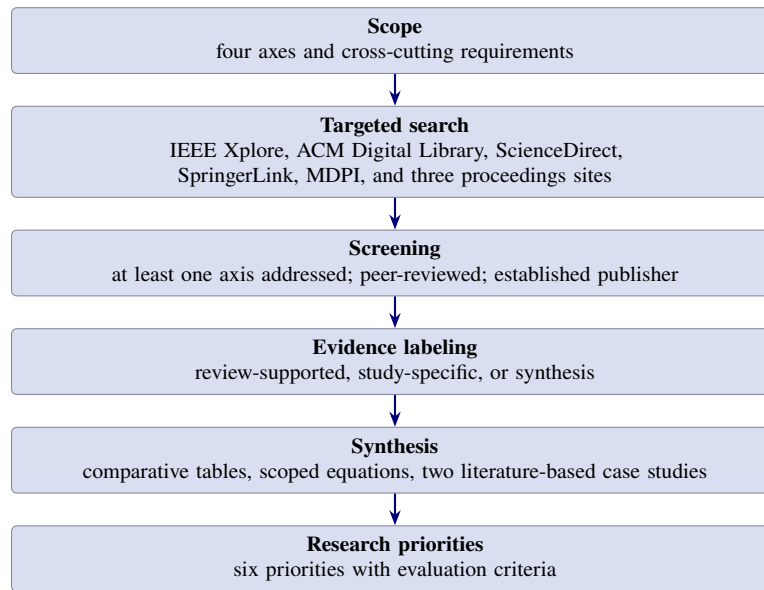


Figure 1: Workflow of the targeted narrative review

A source was included if it addressed at least one of the four axes and appeared in a peer-reviewed journal, conference proceedings, or professional magazine of an established publisher. The co-design surveys of the 1990s are included because the review extends the partitioning and timing concepts established by those surveys. The two case studies of Section 7 were selected because each reports measured results for a learned function on a distinct substrate and each exposes at least one coupling between layers.

Search topics were drawn from the four axes and the cross-cutting requirements and are listed in **Table 2**.

Evidence is classified consistently throughout. A review-supported statement draws on one or more surveys or reviews. A study-specific statement reflects the findings of a single system or study and applies to its workload. A synthesis is an interpretation not explicitly stated in any individual source. Quantitative values are reported as given in the original sources and are not pooled across studies; values recomputed for this review are clearly identified. Four categories of limitation are distinguished: limitations of individual studies, gaps in the assembled literature, engineering challenges reported across multiple sources, and research priorities proposed in this review.

The six research priorities in Section 9 were derived from limitations that recur across the available sources and were selected for their relevance across system layers. For each priority, Section 9 distinguishes between evidence reported in the sources and interpretations developed in this review.

Table 1: Representative related works and the integration emphasized by this review

Work	Type and year	Scope	Integration
De Micheli and Gupta [1]; Wolf [2]	Surveys, 1997 and 1994	Joint design of hardware and software to meet system-level goals; Wolf stresses performance, cost, and design time and covers performance analysis, partitioning, and co-simulation	Extension of partitioning and timing concepts to learned components, security, and post-deployment adaptation
Ray [11]	Review, 2022	State of the art and prospects of TinyML	Joint treatment of memory, timing, and assurance
Abadade et al. [3]	Survey, 2023	Taxonomy of TinyML techniques and applications in healthcare, smart farming, environment, and anomaly detection	Substrate comparison with evidence labels
Saha et al. [7]	Review, 2022	Machine learning on microcontroller-class hardware, a closed-loop development workflow, application classes, use cases, and open challenges	Explicit integration of timing, assurance, and update security with the constrained machine learning workflow
Lin et al. [9]	Magazine feature, 2023	TinyML definition, challenges, and applications, MCUNet system-algorithm co-design, and on-device training	Deadline and assurance conditions for adaptation
Capogrosso et al. [4]	Survey, 2024	Three implementation workflows, a taxonomy of model optimization and learning techniques, hardware devices, and software tools	Cross-layer analysis linking deployment choices to timing, security, and update assurance
Lê et al. [10]	Review, 2026	Efficient network architectures and compression for microcontrollers, with hardware acceleration and algorithm-architecture co-design	Explicit separation of flash and peak RAM budgeting in deployment analysis
Sipola et al. [15]	Scoping study, 2022	Edge AI hardware in five categories and three software trends	Portability and reporting criteria for toolchains
Deng et al. [8]	Survey, 2020	Edge intelligence as the combination of edge computing and AI, divided into AI for edge and AI on edge	Constrained-device view with deadlines and memory limits
Kudithipudi et al. [12]	Journal article, 2025	Neuromorphic computing at scale	Neuromorphic computation placed as a paradigm that overlaps with substrates
Gama et al. [13]	Survey, 2014	Concept drift types, detection, and adaptation methods	Adaptation limited by device cost, rollback, and safety approval
Prakash et al. [14]	Magazine article, 2023	Lifecycle assessment of the environmental impact of TinyML	Accounting boundary and per-decision energy metric

Table 2: Search topics by review axis

Axis	Search topics
Co-design	hardware and software co-design; hardware and software partitioning; co-simulation; embedded system design methodology; reconfigurable systems
Computational substrates	machine learning on microcontrollers; FPGA neural network accelerators; ASIC deep learning accelerators; neuromorphic and spiking neural network hardware; edge AI hardware
Optimization	TinyML; model compression, quantization, and pruning; neural architecture search for microcontrollers; on-device training; federated and gossip learning; concept drift adaptation
Applications	Industry 4.0 and predictive maintenance; over-the-air vehicle updates; pervasive health monitoring; digital twins for healthcare; edge intelligence for the Internet of Things
Cross-cutting requirements	real-time and worst-case execution time analysis; safety of cyber-physical systems; adversarial machine learning; model extraction, poisoning, and backdoors; secure software updates; energy benchmarking of TinyML; lifecycle carbon footprint

Note: Topics were drawn from the four review axes and the cross-cutting requirements. FPGA: field-programmable gate array; ASIC: application-specific integrated circuit; TinyML: machine learning on microcontroller-class devices.

3. HARDWARE AND SOFTWARE CO-DESIGN FOUNDATIONS

3.1 Origins and Core Concepts

Embedded computer systems have traditionally been treated as a co-design problem in which the design of hardware and software components influence one another, and which must meet performance, cost, and design time goals [2]. De Micheli and Gupta define co-design as meeting system-level objectives by exploiting the synergism of hardware and software through concurrent design [1]. Wolf places fault tolerance and verification outside the scope of that survey [2], so these topics enter this review through later sources. The instruction set architecture marks the boundary between the two domains, and it shapes which functions are efficient in software and which are better realized as dedicated logic [1].

This body of work established three concepts that remain central. The first is partitioning, the decision of which functions run on programmable processors and which are realized as fixed or reconfigurable hardware, which the early surveys describe as critical to the quality of a system [1, 2]. The second is co-simulation, the joint modeling of hardware and software behavior, often at different levels of abstraction, to validate performance and function before physical implementation [1, 2]. The third is a system-level view in which the value of a product depends on both its hardware and its software and on the application it serves [1].

3.2 Co-Design with Learned Components

Classical co-design problems are typically posed for deterministic functions such as control and signal processing. Integrating statistical learning into embedded platforms extends the partitioning question to inference and, increasingly, to on-device training [9]. The decision now covers control logic and data paths and also which layers of a neural network run on a microcontroller core, which are offloaded to an accelerator, and which are pruned or quantized to fit the available memory [4, 7]. A machine-learning-oriented survey distinguishes machine-learning-oriented, hardware-oriented, and co-design workflows for constrained devices [4].

Learned components change the design problem in several ways. Execution time can depend on the input or on the execution policy, although this is not a property of every learned model. Models may be retrained or updated after deployment, which can invalidate an earlier hardware and software partition. Input statistics are often nonstationary. Accuracy, latency, energy, and assurance pull in different directions, and a gain in one rarely comes without a cost in another. Real-time control systems must execute functions within predefined time windows, and the variability that caches and data-dependent delays introduce into software timing has led designers of tightly constrained systems to adopt deliberately simple architectures, for example by turning caches off [1]. Worst-case execution time (WCET) must therefore be established and verified [16], which limits the software abstractions and algorithms available for on-device intelligence. The cross-layer consequence is that a model cannot be judged apart from the hardware, timing, and update policy in which it is embedded, and this motivates the optimization work of Section 5. **Table 3** relates classical co-design concerns to learned-system extensions and to the verification issues that follow.

Table 3: Classical co-design concerns and learned-system extensions

Classical concern	Learned-system extension	New verification issue	Evidence basis
Hardware and software partitioning [1, 2]	Partitioning of models, operators, and accelerators [9]	Interaction of accuracy and timing	Synthesis
Interface definition at the instruction set [1]	Runtime and model interfaces	Version compatibility	Synthesis
Performance validation by co-simulation [1, 2]	Data-dependent inference timing	Uncertain execution-time bounds	Synthesis
Reconfiguration [1]	Model and parameter updates	Update assurance	Synthesis

4. COMPUTATIONAL SUBSTRATES FOR EMBEDDED INTELLIGENCE

Two concepts are frequently conflated in discussions of embedded intelligence: the implementation substrate and the computational paradigm. The implementation substrate is the physical platform that executes a workload, such as a microcontroller or processor, a field-programmable gate array (FPGA), or an application-specific integrated circuit (ASIC). The computational paradigm describes how the workload is expressed, for example through dense computation, sparse computation, or event-driven spiking computation.

A computational paradigm may be realized on more than one substrate; consequently, substrates and paradigms form an overlapping design space rather than a set of mutually exclusive alternatives. For example, a neuromorphic processor realizes the spiking paradigm in custom silicon. It is therefore discussed as a computational paradigm in Section 4.5 rather than as a fourth implementation substrate.

4.1 *Microcontrollers*

Microcontrollers are a common target of constrained on-device learning. Typical targets have no dynamic random access memory (DRAM), often no operating system, and static random access memory (SRAM) of a few hundred kilobytes or less, with flash holding code and constants [9]. A review of machine learning for microcontroller-class hardware describes a closed-loop development workflow in which model compression and architecture search adapt models to such limits [7].

4.2 *Field-Programmable Gate Arrays*

FPGAs offer an intermediate option between the flexibility of software and the efficiency of fixed hardware. Field-programmable technology blurred the distinction between hardware and software already in the 1990s, because the interconnection of logic can be configured after manufacturing [1]. Run-time reconfigurability suits cyber-physical systems that need both adaptability and hardware-level throughput. The ARTICo³ framework generates reconfigurable, FPGA-based accelerator systems for edge computing in cyber-physical systems and lets the user trade computing performance, energy, and fault tolerance at run time [17]. The ONNX-to-hardware flow of Manca et al. starts from models in the QONNX format, an extension of the Open Neural Network Exchange (ONNX) format for arbitrary-precision quantization, generates streaming accelerators with a high-level synthesis tool, and merges execution profiles so that the inference engine can switch profiles at run time [18]. In the reported evaluation on a small MNIST classifier, five mixed-precision profiles span accuracy from 95.3 to 98.9 percent at an unchanged latency of 329 microseconds, while lower precision reduced lookup table use from 12 to 6 percent [18]. CFU Playground is a full-stack open-source framework for accelerating TinyML on FPGAs, and its authors report speedups between 55 and 75 times through iterative customization [19].

4.3 *Application-Specific Integrated Circuits*

For a stable and well-characterized workload, custom silicon can reach high energy efficiency. Early co-design work already described this trade: hard-wired circuits often deliver the best performance and the lowest power, but nonrecurring engineering cost is high and flexibility is low [1]. Eyeriss illustrates the category. It is an accelerator for convolutional networks that reconfigures its architecture to the network shape and optimizes the energy of the whole system, including off-chip memory. The reported 65 nm test chip has 168 processing elements and processes the five convolutional layers of AlexNet at about 35 frames per second with a measured power of 278 mW [20].

4.4 *Comparative Assessment of Substrates*

Table 4 provides an overlapping overview of the design space across the three implementation substrates. The entries reflect conditional tendencies synthesized from the available sources rather than findings from a controlled benchmark. Adaptability is assessed through two properties: model updatability, the ability to modify the executed model after deployment, and fit of a new model, the practical feasibility of executing a different model within the platform's memory, timing, and energy constraints.

Other dimensions are relevant in practice, including time to deployment and suitability for safety certification. Because these depend on the qualification of toolchains and deployment processes, they are not scored in the table. A scoping study of edge AI hardware distinguishes devices into AI accelerators and processors, FPGAs, systems on chip, system-on-modules, and full computers [15], showing that the boundaries among the three substrate categories are not always sharp. Substrate selection therefore depends on production volume, power budget, workload sparsity, and toolchain maturity.

4.5 *Computational Paradigms*

Neuromorphic computation is treated as a computational paradigm, whereas neuromorphic hardware refers to its physical implementation. Four related concepts should be distinguished. Spiking neural network algorithms constitute a model class; neuromorphic hardware implements such models directly in silicon. Event-driven sensing and event-driven communication are properties of the data path. Conventional hardware can also implement spiking models.

Table 4: Design-space overview of implementation substrates

Dimension	Microcontroller-class processor	FPGA	ASIC
Programmability	Software programmable	Reconfigurable logic	Fixed function, or programmable within a designed operator set
Throughput	Bounded by core count and clock	Parallel when the workload exposes parallelism	High for the target workload
Energy efficiency	Depends on workload and duty cycle	Depends on workload and design effort	High reported for stable workloads
Memory	Small on-chip SRAM and flash	Configurable on-chip memory	Sized per workload
Development effort	Mature toolchains	Hardware skill and tool flows needed	High nonrecurring cost
Model updatability	Through software or parameter updates, within memory limits	Through parameter updates, software, or logic reconfiguration	Through parameter or software updates where supported, bounded by the fabricated architecture
Fit of a new model	Limited by RAM and flash ceilings	Limited by logic and memory resources	Limited by supported operators, memory, and datapath
Sources	[7, 9]	[1, 17, 18]	[1, 20]
Evidence basis	Review-supported	Study-specific	Study-specific

Note: Entries are conditional tendencies and not measured comparisons. SRAM: static random access memory. Evidence basis: Review-supported follows from surveys or reviews, and Study-specific is reported by a single implementation.

Minitaur illustrates the latter case: it is an event-driven spiking-network accelerator implemented on an FPGA [21]. Its authors report 18.73 million postsynaptic currents per second at 1.5 W and 92% accuracy on MNIST, and observe that processing speed depends on network activity rather than network size [21]. The same study reports lower throughput for a text-classification task, in which limited fanout and weight convergence left parallel cores underutilized and weight caching suboptimal [21]. By contrast, a 28 nm convolutional neuromorphic processor with online learning and spike-based retinas realizes the spiking paradigm in specialized silicon [22].

Event-driven architectures can exploit temporal sparsity, but their system-level benefit depends on event rates, memory movement, peripheral circuitry, communication overhead, and the workload used for evaluation. Kudithipudi et al. [12] discuss neuromorphic computing at scale. Sparsity may also be introduced at the model level, for example through architecture search that produces sparse networks for microcontrollers [23]. **Table 5** relates three computational paradigms to representative implementations.

Table 5: Computational paradigms and example implementations

Paradigm	Example implementation	Dependency of the benefit	Sources
Dense computation	Convolutional accelerator on an ASIC; networks on microcontrollers	Data movement and memory hierarchy	[20, 7]
Sparse computation	Sparse architecture search for microcontrollers	Support for sparse formats in the runtime and hardware	[23]
Event-driven spiking	Spiking accelerator on an FPGA; spiking processor in 28 nm silicon	Event rate, memory movement, and communication overhead	[21, 22, 12]

Note: Each paradigm can run on more than one substrate. ASIC: application-specific integrated circuit; FPGA: field-programmable gate array.

5. OPTIMIZATION AND DEPLOYMENT OF EMBEDDED MODELS

Optimization for embedded intelligence operates at three layers and at different stages, as summarized in **Table 6**. Model-level optimization reduces model storage or computational cost through quantization, pruning, knowledge distillation, and low-rank factorization [10]. Mapping-level optimization includes operator fusion, scheduling, and memory planning for a given target, while hardware-aware architecture search spans both model design and target mapping [23]. Operator fusion is classified as a mapping-level technique because it changes how a fixed network executes on a target rather than

altering the model itself; however, its benefit depends on earlier model-level decisions, such as the selected operator set. System-level adaptation includes on-device training, continual learning, and distributed model updates [9, 24, 25, 26, 5]. These layers interact. A compression decision affects memory use and execution time at the mapping layer, while an adaptation policy determines the properties that must be verified during run time.

Table 6: Optimization layers, stages, and techniques for constrained platforms

Layer	Stage	Technique	Purpose	Sources
Model	Training or deployment time	Quantization	Lower numeric precision to cut memory and arithmetic cost	[10]
Model	Training or post-training	Pruning	Remove redundant weights or structures	[10]
Model	Training time	Knowledge distillation	Train a compact student from a larger teacher	[10]
Model	Training or deployment time	Low-rank factorization	Approximate weight matrices with smaller factors	[10]
Model and mapping	Design time	Hardware-aware architecture search	Find networks that meet memory ceilings	[23]
Mapping	Deployment time	Operator fusion and memory planning	Reduce intermediate activation storage and memory traffic	[9]
Mapping	Deployment time	Automated accelerator generation	Produce hardware from model descriptions	[18]
System	Run time	On-device training and continual learning	Adapt models after deployment under resource limits	[9, 24, 25]
System	Run time	Federated and gossip learning	Update models without centralizing raw data	[26, 5]

Note: Layers follow Section 5: model, mapping, and system. Stage indicates when the technique is applied.

5.1 Model Compression

Broad surveys classify TinyML techniques, tools, and application areas [3, 11], whereas a machine-learning-oriented survey focuses on the underlying learning techniques [4]. Efficient model design combines low-bit quantization, structured and unstructured pruning, knowledge distillation, and low-rank factorization, followed by deployment steps such as hardware acceleration and algorithm–architecture co-design [10]. Knowledge distillation and low-rank factorization are discussed only briefly, as the available sources primarily treat them as components of compression pipelines; a more detailed treatment is provided by [10].

A widely used affine quantization scheme [27] maps a real value x to an integer code q , as shown in **Equation 1**,

$$q = \text{clip}\left(\text{round}\left(\frac{x}{s}\right) + z, q_{\min}, q_{\max}\right), \quad \hat{x} = s(q - z), \quad (1)$$

where $s > 0$ is a scale factor, z is an integer zero point, $[q_{\min}, q_{\max}]$ is the integer range set by the bit width, and $\hat{x}$ is the dequantized value. Frameworks may use one scale per tensor or per channel, symmetric or asymmetric ranges, and different rounding conventions, so **Equation 1** is a standard template and not a universal implementation. Quantization and pruning can also change robustness and calibration, and accuracy alone does not establish that a compressed model remains suitable for a safety-relevant task.

5.2 Hardware Mapping and Memory Budgeting

The boundary between model-level and mapping-level optimization is not sharp. Hardware-aware architecture search changes the model while optimizing for a particular target, and SpArSe, which combines architecture search with pruning, reports convolutional networks that meet the working memory limit of microcontrollers with higher accuracy and up to 7.4 times smaller size than previous approaches [23]. Automated flows map a trained model onto reconfigurable hardware, as in the ONNX-to-hardware flow of Section 4 [18]. Flash and RAM are budgeted separately, since weights and code are usually held in flash and the size of activations is limited by SRAM [9]. For directly stored dense or sparse weight values, a

simplified flash budget is shown in **Equation 2**,

$$M_{\text{flash}} = \frac{1}{8} \sum_{l=1}^L n_l b_l + M_{\text{pack}} + M_{\text{idx}} + M_{\text{code}} + M_{\text{const}} \quad (\text{bytes}), \quad (2)$$

where n_l is the number of stored weight values in layer l (all weights for dense storage and retained values for sparse storage), b_l is the bit width, M_{pack} is the padding and alignment overhead of byte rounding and bit packing, M_{idx} is the index storage of sparse formats and is zero for dense storage, M_{code} is executable code and runtime library, and M_{const} holds quantization parameters, tensor shapes, and other metadata. The peak RAM requirement, reported as peak memory by Lin et al. [9], depends on tensor lifetimes and buffer reuse and not on the sum of all activations, as shown in **Equation 3**,

$$M_{\text{RAM}}^{\text{peak}} = \max_t \left[\sum_{i \in \mathcal{A}(t)} m_i + m_{\text{scr}}(t) \right] + M_{\text{rt}}, \quad (3)$$

where $\mathcal{A}(t)$ is the set of distinct live tensor-storage allocations at execution step t , including input and output storage, and m_i is the allocated size in bytes. Aliased tensors or views that share a buffer are counted once. The term $m_{\text{scr}}(t)$ is time-varying scratch and runtime allocation at step t , and M_{rt} is the time-invariant runtime overhead. The boundary must be declared. Stack, heap, operating-system, driver, and communication buffer allocations belong to M_{rt} when fixed over the inference and to $m_{\text{scr}}(t)$ when varying with step t . Operator fusion and memory planning act on $\mathcal{A}(t)$ and therefore on **Equation 3**. Reducing b_l is effective against flash limits, but it shrinks activation tensors only if activations are also quantized, so it does not by itself relax the RAM terms. During on-device training, gradients and optimizer state add further memory, which can be a multiple of the inference requirement. **Equations 2** and **3** are proposed analytical templates and are not calibrated against data.

5.3 On-Device Adaptation and Distributed Learning

Much of the reviewed deployment-oriented work concerns inference, while a growing body of work addresses training on the device, which surveys treat as an extension of the field beyond inference [9]. TinyDL, proposed in 2017, trains and executes deep models on an embedded GPU platform by transforming streaming input data into a lower-dimensional form tuned to memory, runtime, and energy budgets, and it reports training energy reductions of up to 99.6 times on one sensing task [24]. The evaluation platform is an embedded GPU board and not a microcontroller, so the result illustrates resource-aware adaptation and not a microcontroller deployment. Catastrophic forgetting remains an obstacle to incremental training, and replay of compressed representations of earlier data is one proposed mitigation [25]. Replay is not free on constrained devices, since it needs extra memory, may retain sensitive data, and requires computation to select samples. Other approaches include regularization of parameter changes, isolation of parameters per task, compact prototype memories, generative replay, and periodic retraining assisted by an edge server. On-device training further raises concerns that the underlying sources treat only in part: training energy, flash wear, numerical stability at low precision, privacy risk from retained local data, the safety implications of unverified updates, and recovery from failed updates.

Five modes of adaptation should be distinguished. On-device fine-tuning updates a model locally. Continual learning updates it sequentially while limiting forgetting. Federated learning aggregates updates from many devices. Gossip learning exchanges models among peers. Server-assisted retraining allows an edge or cloud server to retrain and redistribute the model. Federated and gossip learning both keep raw data on the devices, but they differ in structure. In the coordinator-based form, federated learning aggregates updates through a central node, whereas gossip learning requires no aggregation server and exchanges models among peers [26]. Neither approach guarantees privacy on its own, since exchanged updates can leak information or be poisoned, and both introduce communication and synchronization costs. Surveys of edge intelligence identify communication efficiency as a major concern in federated training at the edge [8], and Park et al. examine collective training and inference over wireless links at the edge [5]. The energy consumed by model exchanges should therefore be counted together with the energy used for inference.

6. TIMING, SAFETY, AND SECURITY IN CYBER-PHYSICAL SYSTEMS

This section addresses the fourth research question. It treats safety and assurance, then timing, and then the security of distributed deployments.

6.1 Safety Context and Related Properties

Embedded intelligence does not operate apart from the physical process it monitors or controls. Cyber-physical systems couple cyber and physical parts to provide mission-critical services, and operation of such systems must ensure safety, security, and sustainability under the dynamics of the physical environment [28]. These concerns tie sensing accuracy and control loop timing to the consequences of delayed or wrong actuation. For control systems, reliability, availability, and safety rank above raw performance as design criteria [1]. Run-time evaluation based on operational data has been proposed to assess the safety of agent-based cyber-physical systems, complementing offline verification with monitoring of deployed behavior [29].

Several related properties are easily conflated. Safety concerns the prevention of unacceptable physical consequences. Security concerns resistance to malicious interference. Privacy concerns limiting the exposure of sensitive information. Reliability concerns correct service over time, robustness concerns tolerance of noise and distribution shift, and availability concerns continued operation despite faults or attacks. The properties interact. Quantization or pruning may change robustness, continual adaptation may improve accuracy under drift while complicating safety assurance, federated learning may reduce raw-data centralization without providing privacy, and a verified signature on an update does not guarantee the safety of an adapted model. A mechanism that serves one property therefore needs to be checked against the others.

6.2 Latency Budgets and Deadline Classes

Real-time constraints impose requirements beyond those of general-purpose machine learning deployment. A sufficient deadline condition for a sequential path from sensing to actuation, built from component bounds as in worst-case timing analysis [16, 1], is shown in **Equation 4**,

$$T_{e2e}^{WC} \leq B_{e2e} := t_{sense}^{WC} + t_{queue}^{WC} + t_{comm}^{WC} + t_{inf}^{WCET} + t_{act}^{WC} \leq D, \quad (4)$$

where T_{e2e}^{WC} is the worst-case end-to-end latency under the specified operating assumptions, B_{e2e} is the sum of component bounds, each summed term is a credible upper bound on the corresponding sensing, queueing, communication, inference, or actuation delay, and D is the deadline of the loop. The bound requires stated scheduling and interference assumptions, such as a fixed execution order and bounded contention for shared resources. It is conservative and does not necessarily equal the exact worst-case latency, because the individual worst cases may not occur together. A hard guarantee needs a bound for every term and not only for inference, and deriving such bounds is difficult in heterogeneous systems. Measured values, or terms treated as random variables, support only empirical or probabilistic statements. Even with locally integrated processing, the queueing and communication terms need not vanish, because internal buses, direct memory access transfers, memory contention, synchronization, and accelerator invocation can contribute materially to latency.

Deadlines also differ in strictness. Hard deadlines tolerate no misses, firm deadlines make a late result useless, and soft deadlines degrade quality gradually. A learned component may run under a firm or soft deadline within an architecture whose control loop has a hard deadline. In that case the architecture must preserve the hard deadline independently of the inference result, for example through a fallback controller, and cannot assume that the two deadline classes coexist safely.

Three latency quantities must be kept apart: average latency, tail latency, and a WCET bound. A measured 99th percentile is not equivalent to a formal worst-case bound. Real-time requirements favor architectures for which a credible upper bound on end-to-end latency can be established, which may favor deterministic or tightly bounded accelerators. Obtaining such bounds remains difficult in heterogeneous systems with caches, shared memory, dynamic scheduling, and complex software stacks.

Equations 1 to 4 differ in kind. **Equation 1** is the standard affine quantization mapping [27]. **Equations 2 and 3** are budgeting identities that follow the flash and peak SRAM distinction of Lin et al. [9], with terms assigned to a declared boundary. **Equation 4** is a conservative composition of component bounds as used in worst-case timing analysis [16], and a bound established by analysis differs from a latency measured on a particular run. These expressions are analytical templates and are not calibrated against data.

6.3 Security of Distributed Deployments

Distributing intelligence across many devices enlarges the attack surface. Two properties must be kept apart. Artifact authenticity concerns whether firmware, a model file, or an update comes from an authorized source and arrives unmodified. Behavioral trustworthiness concerns whether the model behaves acceptably even when its artifact is authentic. A successfully verified signature supports authenticity and integrity under stated key-management assumptions, and it does not establish acceptable model behavior. **Table 7** separates six threats by attacker capability, security objective, mitigation, residual risk,

and cost. Adversarial attacks on learned components are organized in a taxonomy in [30], and secure update channels for deployed devices are surveyed in [31]. Model extraction through query access, shown to reach near-perfect fidelity for popular model classes, is studied in [32], poisoning of training data in [33], backdooring in [34], survivable key compromise in update systems in [35], and energy and latency attacks on neural networks in [36], where crafted inputs raised latency and energy by a factor of about 30 on language models. Further concerns include secure boot, hardware roots of trust, device attestation, side channels, and model confidentiality.

Table 7: Threats, attacker capability, mitigations, and evaluation measures

Threat	Attacker capability	Objective	Mitigation	Residual risk	Overhead and measure
Unauthorized modification of firmware or model	Write access to storage or to the update channel	Integrity and authenticity of artifacts	Secure boot, signed artifacts, protected storage [31]	Stolen keys, physical bypass of boot	Verification time and energy; tamper detection rate
Compromised authorized update	Control of a legitimate update source or signing key	Limited damage and recovery	Role separation, threshold signing, and key revocation [35]; staged rollout	A correctly signed model that behaves wrongly	Metadata and staging memory; time to detect and recover
Data poisoning and backdoors	Influence over training or update data	Trustworthy model behavior	Data provenance and validation on clean held-out data before activation [33, 34, 30]	Backdoors that pass validation	Validation compute and memory; attack success rate and clean accuracy change
Model extraction	Query access, or physical access to flash	Confidentiality of the model	Query limits and monitoring [32]; protected storage against physical readout	Side channels, physical readout	Monitoring and storage overhead; extraction fidelity under a query budget
Rollback or downgrade	Replay of an older valid update	Freshness of artifacts	Signed metadata with version numbers and expiry [35]; version counters	Counter reset by physical tampering	Nonvolatile counter; rejection rate of replayed updates
Resource exhaustion	Crafted inputs or request floods	Availability and timing	Per-inference time or energy cutoff [36]; input bounds, watchdog, rate limits	Unknown worst-case inputs	Monitoring cost; worst-case latency and energy under crafted inputs

Note: Mitigations, residual risks, and overhead measures are syntheses of the findings. The last column lists the resource cost and the evaluation measure of each mitigation.

The mitigation combinations, residual risks, and overhead measures in **Table 7** are syntheses of the findings.

7. CROSS-LAYER SYNTHESIS CASE STUDIES

Two case studies from the literature organize the available information around five headings: system and requirement, reported implementation, reported results, cross-layer interpretation, and limits of the evidence. The case studies cover two substrates, an FPGA and a set of microcontrollers, and both are laboratory implementations. These case studies illustrate the proposed cross-layer perspective and do not constitute a controlled validation. Reported content is stated first, and interpretation is marked as such.

7.1 Case Study 1: Digital Twin Learning on an FPGA

Xu et al. study the learning of patient-specific digital twins for precision healthcare on an FPGA, a mobile GPU, and a cloud GPU baseline [37]. Two healthcare tasks are reported, automated insulin delivery and cardiac monitoring. **Table 8** states the ratios for each task and the basis on which each was obtained.

System and requirement

The study motivates its work by the need for fast, resource-efficient learning of a mathematical representation of patient dynamics in mission-critical healthcare applications, which existing model recovery techniques often cannot deliver because of iterative solvers and high compute and memory demands. The proposed learning framework is amenable to acceleration on reconfigurable hardware, and its use is shown for synthetic data generation in type 1 diabetes and for proactive coronary artery disease detection.

Table 8: Digital twin learning ratios derived from the tabulated values of the study

Quantity and direction	Insulin task	Cardiac task	Measurement basis
Runtime, cloud GPU over FPGA	1.67 (423.28 / 253.84)	4.04 (103.4 / 25.6)	Runtime recorded in software
Runtime, mobile GPU over FPGA	2.2	4.5	Method not stated for the mobile GPU
DRAM footprint, cloud GPU over FPGA	28.56 (6118.36 / 214.23)	25.6 (3101.6 / 121.3)	Process-level memory read through a software library
DRAM footprint, mobile GPU over FPGA	11.0	8.7	Method not stated for the mobile GPU
Tabulated metric, FPGA over cloud GPU	8.8	3.6	Runtime divided by power; FPGA power estimated, GPU power measured
Recomputed energy, cloud GPU over FPGA	24.5	Not available	Product of listed power and runtime; the cardiac table lists no power
Recomputed energy, mobile GPU over FPGA	2.5	Not available	Same basis as the preceding row

Note: Ratios are recomputed from the tabulated values of the study. Runtime is in seconds, power in watts, and the DRAM footprint in the units of the study. Recomputed energy is the product of listed power and runtime. DRAM: dynamic random access memory; GPU: graphics processing unit; FPGA: field-programmable gate array.

Reported implementation

The compared platforms are a PYNQ-Z2 FPGA board, a Jetson Orin Nano module as the mobile GPU, and a workstation with an RTX 6000 GPU that serves as the cloud baseline. The FPGA model is a gated recurrent unit network implemented with high-level synthesis. Runtime is recorded in software, and the DRAM footprint is process-level memory read through a software library. The FPGA power comes from the power analysis of the Vivado design tool, whereas the mobile GPU and cloud GPU powers were read with tegrastats and nvidia-smi, respectively.

Reported results

For the insulin task, the FPGA implementation is 1.67 times faster than the cloud baseline, and the mobile GPU runs about 2.2 times longer than the FPGA. The DRAM footprint is 28.56 times smaller on the FPGA than on the cloud baseline, which the study reports as over 28.5 times, and the mobile GPU footprint is about 11 times that of the FPGA. For the cardiac task, the runtime ratios are 4.04 and 4.5, and the footprint ratios are 25.6 and 8.7, so neither set of ratios is universal across the two tasks. The reported performance-per-watt ratio of 8.8 for the insulin task is analyzed in Section 9.4. The average error of the FPGA exceeds that of the cloud baseline for both tasks, 4.60 against 2.90 and 7.20 against 6.10, in the units of the study, while the mobile GPU reaches 3.07 on the insulin task and 8.07 on the cardiac task.

Cross-layer interpretation

The requirement is a latency and resource requirement attached to a physical process. The footprint is process-level DRAM use and is not the peak RAM of **Equation 3**, so it cannot be used directly for a microcontroller-style memory budget. The runtimes are software measurements, so no deadline guarantee follows from **Equation 4**. The memory and energy advantages come with an accuracy cost, and any update of a twin in a closed-loop medical setting would need validation against that cost.

Limits of the evidence

The power values combine an estimate with measurements, the recording method for mobile GPU runtime and memory is not stated, and drift and secure updating are not evaluated. The evidence is a single laboratory study of two tasks.

7.2 Case Study 2: Distributed Inference on Microcontrollers

Bolat et al. split a pre-trained LeNet network across several microcontrollers that run inference together over a wireless sensor network without a central coordinator, and test the scheme on vibration data from an industrial water pump [38].

System and requirement

The study states that running deep neural networks on low-power microcontrollers is difficult because of limited memory, computation, and energy. The task is vibration-based anomaly detection with induced motor imbalance at three intensities, posed as binary and multi-class classification. The requirement is a monitoring function with low latency and low energy on devices of limited memory.

Reported implementation

The implementation uses three ESP32-S3 Sense modules, each paired with an ADXL345 accelerometer, which exchange data over Wi-Fi with the transmission control protocol. The non-distributed baseline model of 226 KB can run on a single ESP32-S3 module using its external memory. A single STM32H7 device serves as the reference for the power comparison with that baseline, because the baseline fits in its internal SRAM and external memory therefore does not distort the power measurement.

Reported results

The study reports accuracy of 99.01 percent for the binary task and 98.84 percent for the multi-class task, which matches the non-distributed baseline, together with a latency reduction of about 50 percent. An int8 quantized single-device model reaches 98.68 and 97.19 percent on the two tasks.

Cross-layer interpretation

Splitting the model lowers the RAM and flash requirement of each node, so **Equation 2** and **Equation 3** apply per node, although the 226 KB baseline already fits on one module with external memory, so the reported benefit is latency and not feasibility. Splitting also adds intermediate data that must cross the radio link. In contrast to the quantized model, the split model preserves the baseline accuracy. The communication term of **Equation 4** now sits on the inference path, and Wi-Fi with the transmission control protocol offers no deterministic bound, so the reported reduction describes measured behavior and does not establish a deadline guarantee. Wireless exchange of intermediate activations enlarges the attack surface, and a model that is spread over several nodes needs coordinated, versioned updates.

Limits of the evidence

The test data come from a controlled laboratory rig, so the work demonstrates feasibility rather than field performance. The study, as summarized, does not address security or update management, and those obligations are therefore presented as interpretation.

7.3 What the Case Studies Show

In both case studies a gain at one layer arrives with a consequence at another that a single-layer summary would not show. In the first, lower memory use and lower energy per task come with higher error and with measurements that do not support a deadline claim. In the second, splitting a model across nodes meets the memory limit of each node and moves part of the latency into a communication term that lacks a deterministic bound. The pattern supports treating model quality, hardware mapping, timing, energy, assurance, and adaptation as jointly constrained.

8. APPLICATION DOMAINS

Figure 2 shows the arrangement common to the reviewed domains. A compact learning model on an embedded processor, drawn as a chip at the center with a small neural network graphic, is connected to four endpoints by one arrow in each direction. Clockwise from the upper left, the endpoints are an industrial robotic cell with a vision head above a conveyor, a wearable health monitor showing physiological waveforms, street-level smart infrastructure with wireless nodes, and a vehicle with roof-mounted sensing. These endpoints correspond to industrial automation, healthcare monitoring, the Internet of Things, and automotive electronics, in that order. The figure contains no labels and is a conceptual schematic rather than a depiction of a measured system.

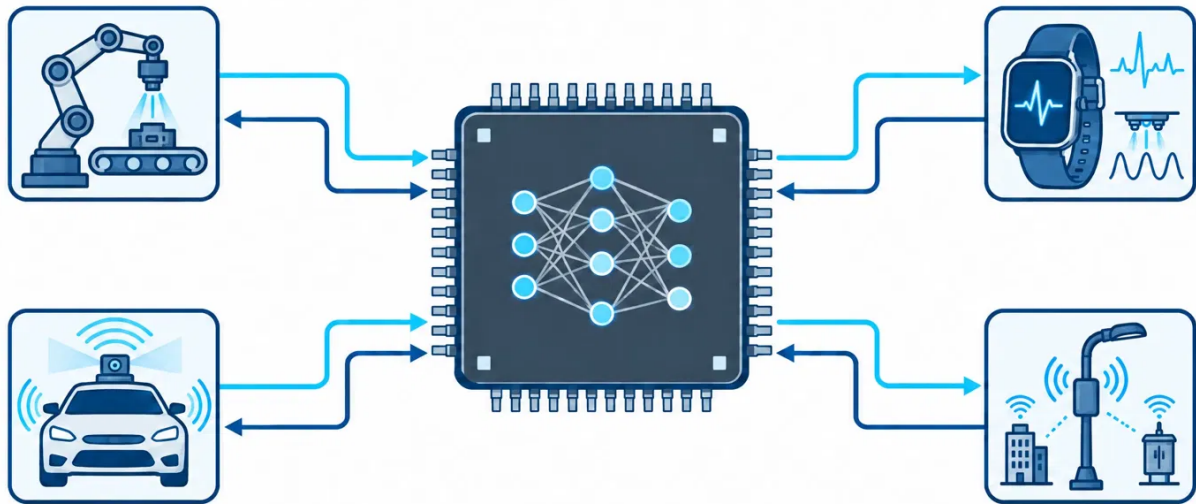


Figure 2: Embedded intelligence across four application domains

The four domains recur across the reviewed literature and represent distinct constraint profiles: determinism and availability in industry, functional safety and updating in vehicles, battery life, privacy, and regulation in healthcare, and energy, connectivity, and scale in distributed sensing. The Internet of Things (IoT) is an architectural category that overlaps with the other three, since industrial sensing, connected vehicles, and wearable monitors are all IoT deployments. It is treated as the architectural lens for edge intelligence and constrained devices, and the smart infrastructure endpoint of **Figure 2** belongs to it.

Table 9 maps each domain to the type of evidence and deployment stage. Four stages are distinguished: analysis or simulation, laboratory implementation, prototype demonstration, and operational deployment. A literature review is not assigned a stage of its own, since the primary studies it covers must be assessed individually. An empirical survey of organizations is also treated separately from a literature review. The constraints and open issues listed in the table are syntheses of the underlying sources. Overall, the evidence includes reviews, analytical work, laboratory implementations, and one prototype, but it does not establish the operational performance of learned functions in any of the four domains.

8.1 Industrial Automation

The industrial evidence spans three levels. Frank et al. report adoption patterns of connected sensing and analytics technologies in 92 manufacturing companies under the Industry 4.0 label [39]. Reconfigurable edge frameworks such as ARTICo³ target adaptive embedded processing in cyber-physical applications [17]. At the level of a learned function, a laboratory case study of vibration-based anomaly detection on an industrial water pump reports accuracy that matches a single-device baseline together with a latency reduction of about 50 percent when a network is split across three microcontrollers [38]. The pump data come from a controlled rig with induced imbalance, so the result demonstrates feasibility and not field performance. Predictive maintenance, process monitoring, and quality inspection are the tasks most often proposed for these capabilities, and among the primary studies only predictive maintenance is evaluated, in the laboratory. The industrial evidence therefore supports statements about adoption, platform flexibility, and laboratory

Table 9: Application domains, evidence, deployment stage, and open issues

Domain	Function	Evidence type and stage	Dominant constraint	Open issue
Industrial automation	Adoption of connected sensing and analytics [39]; adaptive edge processing [17]; vibration-based anomaly detection on microcontrollers [38]	Empirical survey of 92 firms; platform implementation; laboratory case study	Determinism, availability	Validation of learned monitoring on plants that change over time
Automotive electronics	Secure over-the-air updating [31]	Survey, no stage assigned	Real time, functional safety	Validation of learned components and safe updates
Healthcare monitoring	Context sensitivity of monitoring [40, 41]; digital twin learning [37]	Analytical studies; laboratory implementation	Battery, privacy, context-dependent accuracy	Validation within the applicable regulatory category
Internet of Things	Edge intelligence [8]; edge AI hardware and software [15]; machine learning as a service [6]	Survey and scoping study, no stage assigned; architecture with a case study	Energy, connectivity	Update management at scale

Note: Deployment stages are analysis or simulation, laboratory implementation, prototype demonstration, and operational deployment. A literature review is not assigned a stage. Dominant constraints and open issues are syntheses of the underlying sources.

feasibility, and it does not support statements about the performance of learned monitoring in operating plants. Validation on plants whose behavior changes over time remains open.

8.2 Automotive Electronics

Perception, sensor fusion, and control are typical functions of vehicle electronics, whereas the available evidence concerns the updating of deployed vehicles. Because vehicles must be updated after deployment, secure over-the-air software update mechanisms are an important topic, and these mechanisms require access to the in-vehicle communication network, which makes security critical [31]. The survey does not establish how learned perception or control components are validated, and that question would need a broader evidence base. Pairing accelerators for perception with real-time control software illustrates the co-design problem of Section 3. Validation of learned components and the safety of such updates remain open.

8.3 Healthcare Monitoring

Healthcare applications span consumer wellness monitoring, clinical monitoring, and regulated medical devices, which differ in validation, privacy, and regulatory requirements, and results from one category should not be assumed to transfer to another. Studies of pervasive health monitoring show that mobility-induced context changes alter processing requirements, energy sources, and the interaction between a medical device and the body, and can cause unsafe conditions such as drug overdose [40]. A later study extends the analysis to smart mobile applications for healthcare [41]. Digital twin learning for precision healthcare is a further task, examined in Section 7 [37]. The evidence consists of analytical studies and a single laboratory implementation. Context-aware processing pipelines and validation within the applicable regulatory category remain open.

8.4 Internet of Things

Edge intelligence, the combination of edge computing and artificial intelligence, is the architectural framework most closely tied to embedded intelligence in IoT deployments [8]. A scoping study of edge AI hardware and software covers five hardware categories and three software trends, namely neural network optimization, mobile device software, and microcontroller software, and describes an ecosystem that is still developing [15]. A machine learning as a service architecture for IoT devices serves pre-trained models at the edge, and its feasibility is tested with a case study [6]. The sources address generic constrained-device architecture and not a specific sector, which is consistent with the overlap noted above. Update management at scale remains open.

9. RESEARCH PRIORITIES AND EVALUATION CRITERIA

Six research priorities are examined: lifecycle energy accounting, assurance of learned components, security of distributed deployments, reproducible benchmarking, portable deployment toolchains, and concept drift. Each subsection that follows ends with the research direction and the evaluation basis for its priority. **Table 10** traces each priority to the limitations reported in the underlying sources and separates the part that is derived in this review. **Table 11** gives the layer, the cross-layer dependencies, and the proposed evaluation criteria. The priorities differ in kind. Lifecycle energy accounting is a measurement problem, assurance is a verification problem, security is a threat and mitigation problem, benchmarking is a methodological problem, toolchain portability is an engineering problem, and concept drift is an adaptation problem. The criteria are proposals for later studies.

Table 10: Source-to-priority matrix

Priority	Limitation reported in the underlying sources	Element derived in this review
Lifecycle energy	A lifecycle analysis finds that the footprint of microcontroller systems is significant at global scale [14]	A single accounting boundary and allocation rule across all terms
Assurance	Certified controller synthesis that treats human actions as disturbances can yield designs that fail under unmodeled actions [42]; attacks that cause misclassification can lead to erroneous decisions and unreliable operation [30]; run-time evaluation complements offline checks [29]	Combination of formal and run-time assurance, and a requirement coverage metric
Security	Over-the-air updates need access to the in-vehicle network, so security is critical [31]; attacks and defenses on learned components are organized in a system-driven taxonomy [30]	Threat table with embedded resource overheads
Benchmarking	The lack of a widely accepted, reproducible benchmark is the stated motivation of a TinyML benchmark suite [43]	Completeness of reporting and declared boundaries, motivated by the mixed estimated and measured power and the nonstandard efficiency metric in [37]
Toolchains	Hand-crafting a network for a device takes time and experience, which motivates automated flows [18]; FPGAs often lack accessibility and ease of development [17]; the edge AI ecosystem is still developing [15]	Cross-target portability metric
Concept drift	Incremental updating of networks leads to catastrophic forgetting [25]; drift detection and adaptation methods are surveyed [13]	Coupling of adaptation to rollback and safety approval

Note: The right column lists the element derived in this review, as distinct from the limitations reported in the underlying sources.

9.1 Lifecycle Energy Accounting

Embedded intelligence is often motivated by energy savings relative to cloud offloading, yet inference-time energy does not capture the full cost of the machine learning lifecycle. A lifecycle analysis of machine learning on microcontrollers concludes that the carbon footprint of such systems is significant at global scale, which calls for accounting that covers hardware manufacturing and end of life in addition to operation [14]. A per-device accounting framework proposed in this review extends the stages of that analysis with data collection, training, updates, and communication, as shown in **Equation 5**,

$$E_{\text{life}} = \frac{E_{\text{data}} + E_{\text{train}}}{N_{\text{dev}}} + E_{\text{update}} + N E_{\text{dec}} + E_{\text{comm}} + E_{\text{idle}} + E_{\text{hw}} + E_{\text{end}}, \quad (5)$$

where $N_{\text{dev}} > 0$ is the number of devices that share the allocated costs, E_{data} and E_{train} are the shared energy of data collection and training, and the remaining terms are per device: model updates, N decisions with mean energy E_{dec} over the accounting period, communication outside updates, idle and sleep states, hardware production, and end-of-life processing. All terms are expressed in joules over the same accounting period. Shared energy is allocated equally across devices, although a different allocation rule is acceptable if it is stated. The service lifetime and accounting period are declared for every term. Terms overlap unless a boundary is declared, and double counting is avoided by assigning each joule to exactly one term. Radio energy spent during a model update belongs to E_{update} rather than to E_{comm} . The per-decision term includes the sensing, preprocessing, inference, and postprocessing on the decision path within the declared boundary, whereas

Table 11: Evaluation criteria for the six research priorities

Priority	Layer and dependencies	Proposed evaluation criteria
Lifecycle energy	System; hardware, communication, updates	Total energy within the declared boundary divided by the number of correct, timely decisions over the same interval. The correctness criterion and deadline are fixed in advance, and the ratio is reported as undefined when the denominator is zero. Task-appropriate error measures, missed or late decisions, and any amortization of shared lifecycle costs are reported alongside.
Assurance	System and model; model, runtime, control	Requirement evidence coverage, the fraction of specified safety requirements with traceable evidence assessed against predefined acceptance criteria, and scenario coverage against an enumerated scenario set, reported separately. Monitor false-negative rate and fallback response time are added. Neither coverage fraction alone establishes safety.
Security	System; firmware, model, communication	Attack success rate and detection latency under a stated attacker capability, recovery time to a verified state, and resource overhead.
Benchmarking	Evaluation; all layers	Repeatability, the spread across repeated runs under one protocol. Reproduction agreement, assessed with an equivalence tolerance declared in advance and uncertainty estimates for the original and repeated results. Completeness of reported metadata.
Toolchains	Mapping; model and substrate	Cross-target build success with verified resource compliance, meaning that the build meets declared memory, timing, and accuracy limits on each target, and performance relative to a hand-tuned baseline.
Concept drift	Adaptation; model, data, safety	Performance before and after adaptation on the same shifted test set against a matched frozen-model baseline, and retention on a fixed pre-drift reference set. Drift-detection delay and false alarms. Rollback success, the share of injected failed updates after which the device returns to the last verified version, passes a self-test, and meets its timing limit.

Note: Layer names the design layer addressed by the priority, and dependencies list the layers that constrain it. The criteria are proposals for later studies.

sensing performed only to collect training data belongs to E_{data} . A common approximation is $E_{\text{dec}} \approx P_{\text{act}}, t_{\text{dec}}$, where P_{act} is the average power over the decision path as measured at the declared boundary. Whether memory and peripheral energy are included depends on what that measurement covers, and sleep transitions and startup are omitted unless the path includes those steps. **Equation 5** is an accounting framework developed in this review from the lifecycle stages of [14].

Research direction: Lifecycle-aware energy accounting with declared boundaries, together with efficient event-driven hardware [12]. Energy per correct decision should be reported alongside task performance and failure measures, so that an energy ratio does not obscure application-relevant errors, and emissions should be reported separately from energy.

9.2 Assurance of Learned Components

Learned components complicate formal verification because the behavior of a trained model is an emergent property of data and optimization and not of an explicit rule set. Assurance involves several distinct activities: verification of the model, verification of the surrounding software, validation against operational data, run-time monitoring, run-time enforcement or fallback control, and certification or regulatory approval. A system-driven taxonomy of attacks and defenses specifies machine learning applications and adversarial system models so that experiments can be replicated, and it notes that attacks causing misclassification can lead to erroneous decisions and unreliable operation [30]. Synthesis of operationally safe controllers for systems that combine a human operator, an embedded intelligence component, and a physical plant remains an active direction, and it reflects the difficulty of formal guarantees for human-in-the-loop systems [42]. Run-time monitoring of deployed behavior [29] complements such offline guarantees, although monitoring can detect anomalous behavior without establishing its cause or a formal guarantee.

Research direction: Formal methods combined with run-time assurance for systems with learned components. Requirement evidence coverage records traceability to predefined acceptance criteria, while scenario coverage records which declared scenarios have been assessed. Neither alone establishes safety, and both are interpreted together with the quality of the evidence, the false-negative rate of monitors, and the fallback response time.

9.3 Security Methods for Embedded Deployments

The threat analysis of Section 6.3 shows that distributing intelligence enlarges the attack surface and that a verified signature establishes artifact authenticity and not acceptable model behavior. Privacy is treated separately from security.

Decentralized learning protocols keep raw data local but add communication and coordination between devices [26], and exchanged updates can be exposed to inference and poisoning, so data locality alone does not establish privacy.

Research direction: Security methods designed for the limits of embedded and cyber-physical systems. Evaluation uses attack success rate, detection latency, recovery time, and resource overhead under a stated attacker capability. Each protection consumes energy and memory that constrained devices can ill afford, and the underlying sources do not quantify that cost for learned embedded systems.

9.4 Reproducible Benchmarking

Comparisons assembled from several studies combine figures obtained with different datasets, hardware, and measurement procedures, so accuracy, latency, and energy are comparable only when measurement boundaries and metric definitions match. The digital twin study of Section 7 shows two separate problems [37].

The first problem is the mixing of estimated and measured power. The FPGA power comes from the power analysis of the Vivado design tool, which yields an estimate, whereas the mobile GPU power was read with tegrastats and the cloud GPU power with nvidia-smi, which yield measurements. The study does not state whether the estimate and the two measurements cover the same part of the system, such as the processing host and the board. A ratio built from one estimated and two measured values does not share a common measurement boundary and should not be read as a like-for-like efficiency comparison.

The second problem is the construction of the metric. For a task of fixed size W completed in time T at average power $\bar{P}$, the energy is $E = \bar{P}T$, and performance per watt is the work done per unit of energy, in line with the microjoules per inference reported by MLPerf Tiny [43], as shown in **Equation 6**,

$$\eta = \frac{W/T}{\bar{P}} = \frac{W}{E}, \quad E = \bar{P}T, \quad (6)$$

so a lower energy for the same work means a higher efficiency. The tables of the study instead list a quantity labeled performance per watt in seconds per watt, and the listed values equal runtime divided by average power. For the FPGA, mobile GPU, and cloud GPU of the insulin task, 253.84 s at 4.905 W, 562.75 s at 5.532 W, and 423.28 s at 72.00 W give 51.75, 101.73, and 5.88 s/W, matching the tabulated 51.76, 101.74, and 5.88 within rounding. In terms of energy, the tabulated quantity is shown in **Equation 7**,

$$\rho = \frac{T}{\bar{P}} = \frac{T^2}{E}, \quad (7)$$

which grows with runtime at fixed power and therefore differs from the conventional definition of efficiency. As a worked example, a task that runs for 10 s at 5 W uses 50 J and gives $\rho = 2$ s/W, whereas the same task run for 20 s at 5 W uses 100 J and gives $\rho = 4$ s/W, so the tabulated quantity doubles while the energy per task also doubles. The ratio of the FPGA to the cloud GPU is $\rho_F/\rho_C = (T_F/T_C)^2 E_C/E_F = (1/1.67)^2 \times 24.5 \approx 8.8$, which is the reported figure. It is the recomputed energy ratio reduced by the squared runtime ratio, and it is not a ratio of work per joule. The same construction gives the mobile GPU about twice the tabulated value of the FPGA, although its recomputed energy per task is about 2.5 times that of the FPGA. The tabulated values for the cardiac task agree with the same construction and with the same three power values, since 25.6 s at 4.905 W gives 5.22 s/W and 103.4 s at 72.00 W gives 1.44 s/W.

The recomputation uses only values printed in the study. The FPGA energy inherits the estimated power, and the runtime interval and the power interval are not stated to coincide, so the energies indicate direction and order of magnitude and are not measured values. The 1.67 runtime ratio and the 28.56 memory ratio describe the insulin task only, since the cardiac task gives 4.04 and 25.6, and neither pair is a universal ratio across the two healthcare tasks.

No widely accepted, reproducible benchmark existed for ML inference on ultra-low-power systems, and the MLPerf Tiny suite was introduced to fill that gap, measuring accuracy, latency, and energy across four tasks in its initial release [43]. The absence of a standard benchmark is therefore not the real obstacle: what matters is the completeness of reporting, an explicit statement of the system boundary, and a precise definition of the metric. **Table 12** lists the minimum reporting requirements that would make results comparable. The same requirements apply to neuromorphic designs, where event rates and workloads vary widely.

Research direction: Standardized cross-platform benchmarking with explicit hardware-dependent reporting and not hardware-agnostic benchmarking, because a benchmark that measures device-level energy, timing, or memory cannot be fully independent of the device. Evaluation uses repeatability across runs, the agreement of independent reproductions under a matched protocol, and the completeness of reported metadata. Hardware-specific effects cannot be removed entirely, and the protocol is meant to expose such effects and not to conceal them.

Table 12: Minimum reporting requirements for embedded intelligence benchmarks

Item	Minimum requirement
Measurement boundary	Chip, board, accelerator, or system level, and whether sensing, preprocessing, postprocessing, and communication are included
Latency	Mean, a high percentile, and the observed maximum, without equating a measured percentile with a WCET bound
Energy	Energy per task in joules from average power over the same interval as the runtime, active and idle power, whether each power value is measured or estimated, the instrument or estimation tool, sampling rate, and supply voltage
Efficiency metric	Performance per watt defined as work divided by energy with the unit of work stated, and ratios formed only from values of one basis, all measured or all estimated
Intermittent workloads	Duty cycle and wake-up cost
Predictive performance	Dataset, preprocessing, data split, task-appropriate quality and error measures, and calibration where applicable
Resource use	Parameter count, numerical precision, batch size, flash use, peak RAM and its measurement boundary, and FPGA or ASIC resource use where applicable
Operating conditions	Clock frequency, voltage, temperature, and control of throttling and frequency scaling
Software stack	Compiler, flags, runtime, and accelerator versions
Repetition	Number of runs and variance or confidence interval
Updates	Retraining and update cost when adaptation is evaluated

Note: WCET: worst-case execution time; RAM: random access memory; FPGA: field-programmable gate array; ASIC: application-specific integrated circuit. The items define the minimum information needed to compare reported results.

9.5 Portable Deployment Toolchains

Toolchains for deploying models on constrained hardware vary by vendor and hardware family. A scoping study of edge AI describes the software ecosystem at the time of its review as still developing [15]. Open-source flows that map networks onto FPGAs, including the ONNX-to-hardware flow [18] and the CFU Playground framework [19], are partial responses. The sources describe flows tied to particular substrates, and none presents a deployment path shared by all substrates.

Research direction: More portable model-to-device flows, extending the automation of network-to-hardware mapping demonstrated in [18, 19]. Successful compilation is not sufficient evidence of portability: a build can succeed and still exceed memory or timing limits. Evaluation therefore requires verified resource compliance on each target, together with a comparison against a hand-tuned baseline. Preserving performance across very different substrates carries a cost that these sources do not quantify.

9.6 Concept Drift and Continual Adaptation

Concept drift refers to post-deployment changes in the data-generating distribution or in predictive relationships that can affect model performance. A survey of drift adaptation distinguishes sudden, incremental, gradual, and recurring drift and categorizes methods for detecting and adapting to it [13]. On constrained devices, drift detection and model adaptation are distinct tasks with different costs. Adaptation also raises questions of update frequency, validation of each update, rollback after a failed update, catastrophic forgetting of earlier behavior [25], and, in safety-critical settings, formal approval. In such settings, fully automatic adaptation is therefore rarely appropriate, and human confirmation may be required. Learning a digital twin of the monitored process [37] may offer a complementary system-level route, but that study does not establish continual drift adaptation.

Research direction: On-device continual adaptation [24, 25], with digital twins as a complementary system-level approach. Evaluation separates the adaptation benefit on a shifted test set from retention on a fixed pre-drift reference set. It also reports drift-detection delay and false alarms, update energy, and rollback success under injected failed updates. Reconciling adaptation with safety approval remains the central difficulty.

10. CONCLUSION

This narrative review organized embedded intelligence around co-design, computational substrates, optimization, and application context, with timing, safety, and security treated as cross-cutting requirements. The reviewed literature supports the following concise answers to the five research questions. Classical concepts of partitioning, co-simulation, and system

context remain relevant, while learned components add data-dependent timing, model error, and post-deployment updates that must be handled in the same design process. No substrate dominates across all design dimensions, and the choice depends on workload, power, flexibility, and deployment constraints. Optimization has to be considered across the model, mapping, and system levels, because a choice at one level changes memory, timing, and verification at the others. Timing, safety, security, and update requirements constrain deployment and are best stated as measurable quantities.

The two literature-based case studies illustrate these dependencies without providing a controlled comparison across substrates. Reported memory, runtime, and efficiency gains must be read within the measurement boundary and metric definition of each study, for the task on which each ratio was obtained, and alongside accuracy. A latency gain that depends on a wireless link does not amount to a deadline guarantee. The application evidence is uneven. It consists mainly of surveys, analytical studies, and laboratory implementations, and it does not establish operational performance of learned functions in the four domains discussed.

Six research priorities connect these observations to evaluation practice: lifecycle energy accounting, assurance of learned components, security of distributed deployments, reproducible benchmarking, portable deployment toolchains, and concept drift. The proposed criteria are conceptual and await validation on shared benchmarks, and the equations are analytical templates that have not been calibrated against data. Because the evidence is heterogeneous and the search was targeted, statements about gaps describe the literature reviewed, and no quantitative ranking across studies is attempted.

The practical contribution of this review is a cross-layer reporting and evaluation framework for embedded intelligence. Future studies should report model, hardware, timing, energy, security, and adaptation outcomes within explicitly declared measurement boundaries. Such reporting would make cross-platform results more comparable, expose trade-offs that single-metric evaluations conceal, and support safer deployment of embedded systems with learned components. Future work should validate the proposed criteria on shared benchmarks, generate field-scale evidence for learned functions, and extend cross-substrate comparisons to additional platforms and application domains.

AUTHOR CONTRIBUTION STATEMENT

Conceptualization, literature analysis, and revision were carried out by the author.

ETHICS APPROVAL AND CONSENT TO PARTICIPATE

Ethics declaration: not applicable. This study did not involve human participants or animals. Therefore, ethical approval and consent to participate are not applicable.

CONSENT FOR PUBLICATION

Not Applicable.

DATA AVAILABILITY

This study is a narrative literature review. No data were generated or analyzed.

ACKNOWLEDGMENTS

The author gratefully acknowledges the intellectual contribution of the broader interdisciplinary literature reviewed in this article. The author also acknowledges the use of ChatGPT-5 for assistance with manuscript refinement.

FUNDING

No Funding.

DISCLOSURE STATEMENT

The author declares no conflict of interest.

REFERENCES

- [1] G. De Micheli and R. K. Gupta, "Hardware/software co-design," *Proceedings of the IEEE*, vol. 85, no. 3, pp. 349–365, 1997.
- [2] W. Wolf, "Hardware-software co-design of embedded systems," *Proceedings of the IEEE*, vol. 82, no. 7, pp. 967–989, 1994.
- [3] Y. Abadade, A. Temouden, H. Bamoumen, N. Benamar, Y. Chtouki, and A. S. Hafid, "A comprehensive survey on tinyml," *IEEE Access*, vol. 11, pp. 96892–96922, 2023.
- [4] L. Capogrosso, F. Cunico, D. S. Cheng, F. Fummi, and M. Cristani, "A machine learning-oriented survey on tiny machine learning," *IEEE Access*, vol. 12, pp. 23406–23426, 2024.
- [5] J. Park, S. Samarakoon, M. Bennis, and M. Debbah, "Wireless network intelligence at the edge," *Proceedings of the IEEE*, vol. 107, no. 11, pp. 2204–2239, 2019.
- [6] S. A. R. Zaidi, A. M. Hayajneh, M. Hafeez, and Q. Z. Ahmed, "Unlocking edge intelligence through tiny machine learning (tinyml)," *IEEE Access*, vol. 10, pp. 100867–100877, 2022.
- [7] S. S. Saha, S. S. Sandha, and M. Srivastava, "Machine learning for microcontroller-class hardware: A review," *IEEE Sensors Journal*, vol. 22, no. 22, pp. 21362–21390, 2022.
- [8] S. Deng, H. Zhao, W. Fang, J. Yin, S. Dustdar, and A. Y. Zomaya, "Edge intelligence: The confluence of edge computing and artificial intelligence," *IEEE Internet of Things Journal*, vol. 7, no. 8, pp. 7457–7469, 2020.
- [9] J. Lin, L. Zhu, W.-M. Chen, W.-C. Wang, and S. Han, "Tiny machine learning: Progress and futures [feature]," *IEEE Circuits and Systems Magazine*, vol. 23, no. 3, pp. 8–34, 2023.
- [10] M. T. Lê, P. Wolinski, and J. Arbel, "Efficient neural networks for tiny machine learning: A comprehensive review," *ACM Transactions on Embedded Computing Systems*, vol. 17, Apr. 2026.
- [11] P. P. Ray, "A review on tinyml: State-of-the-art and prospects," *Journal of King Saud University Computer and Information Sciences*, vol. 34, no. 4, pp. 1595–1623, 2022.
- [12] D. Kudithipudi, C. Schuman, C. M. Vineyard, T. Pandit, C. Merkel, R. Kubendran, J. B. Aimone, G. Orchard, C. Mayr, R. Benosman, *et al.*, "Neuromorphic computing at scale," *Nature*, vol. 637, no. 8047, pp. 801–812, 2025.
- [13] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," *ACM Computing Surveys*, vol. 46, Mar. 2014.
- [14] S. Prakash, M. Stewart, C. R. Banbury, M. Mazumder, P. Warden, B. Plancher, and V. J. Reddi, "Is TinyML sustainable?," *Communications of the ACM*, vol. 66, pp. 68–77, Nov. 2023.
- [15] T. Sipola, J. Alatalo, T. Kokkonen, and M. Rantonen, "Artificial intelligence in the iot era: A review of edge ai hardware and software," in *2022 31st Conference of Open Innovations Association (FRUCT)*, pp. 320–331, 2022.
- [16] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström, "The worst-case execution-time problem—overview of methods and survey of tools," *ACM Transactions on Embedded Computing Systems*, vol. 7, no. 3, pp. 1–53, 2008.
- [17] A. Rodríguez, J. Valverde, J. Portilla, A. Otero, T. Riesgo, and E. De la Torre, "Fpga-based high-performance embedded systems for adaptive edge computing in cyber-physical systems: The artico3 framework," *Sensors*, vol. 18, no. 6, 2018.
- [18] F. Manca, F. Ratto, and F. Palumbo, "Onnx-to-hardware design flow for adaptive neural-network inference on fpgas," in *Embedded Computer Systems: Architectures, Modeling, and Simulation* (L. Carro, F. Regazzoni, and C. Pilato, eds.), (Cham), pp. 85–96, Springer Nature Switzerland, 2025.

- [19] S. Prakash, T. Callahan, J. Bushagour, C. Banbury, A. V. Green, P. Warden, T. Ansell, and V. J. Reddi, “Cfu playground: Full-stack open-source framework for tiny machine learning (tinymml) acceleration on fpgas,” in *2023 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 157–167, 2023.
- [20] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, “Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks,” *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, 2017.
- [21] D. Neil and S.-C. Liu, “Minitaur, an event-driven fpga-based spiking network accelerator,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 12, pp. 2621–2628, 2014.
- [22] C. Frenkel, J.-D. Legat, and D. Bol, “A 28-nm convolutional neuromorphic processor enabling online learning with spike-based retinas,” in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, 2020.
- [23] I. Fedorov, R. P. Adams, M. Mattina, and P. N. Whatmough, “SpArSe: Sparse architecture search for CNNs on resource-constrained microcontrollers,” in *Advances in Neural Information Processing Systems*, vol. 32, pp. 4977–4989, Curran Associates, Inc., 2019.
- [24] B. Darvish Rouhani, A. Mirhoseini, and F. Koushanfar, “Tinydl: Just-in-time deep learning solution for constrained embedded systems,” in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–4, 2017.
- [25] T. L. Hayes, K. Kalle, R. Shrestha, M. Acharya, and C. Kanan, “Remind your neural network to prevent catastrophic forgetting,” in *Computer Vision – ECCV 2020* (A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, eds.), (Cham), pp. 466–483, Springer International Publishing, 2020.
- [26] I. Hegedűs, G. Danner, and M. Jelasity, “Gossip learning as a decentralized alternative to federated learning,” in *Distributed Applications and Interoperable Systems* (J. Pereira and L. Ricci, eds.), (Cham), pp. 74–90, Springer International Publishing, 2019.
- [27] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, “Quantization and training of neural networks for efficient integer-arithmetic-only inference,” in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2704–2713, 2018.
- [28] A. Banerjee, K. K. Venkatasubramanian, T. Mukherjee, and S. K. S. Gupta, “Ensuring safety, security, and sustainability of mission-critical cyber–physical systems,” *Proceedings of the IEEE*, vol. 100, no. 1, pp. 283–299, 2012.
- [29] I. Lamrani, A. Banerjee, and S. K. S. Gupta, “Operational data-driven feedback for safety evaluation of agent-based cyber–physical systems,” *IEEE Transactions on Industrial Informatics*, vol. 17, no. 5, pp. 3367–3378, 2021.
- [30] K. Sadeghi, A. Banerjee, and S. K. S. Gupta, “A system-driven taxonomy of attacks and defenses in adversarial machine learning,” *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 4, no. 4, pp. 450–467, 2020.
- [31] S. Halder, A. Ghosal, and M. Conti, “Secure over-the-air software updates in connected vehicles: A survey,” *Computer Networks*, vol. 178, p. 107343, 2020.
- [32] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, “Stealing machine learning models via prediction {APIs},” in *25th USENIX security symposium (USENIX Security 16)*, pp. 601–618, 2016.
- [33] B. Biggio, B. Nelson, and P. Laskov, “Poisoning attacks against support vector machines,” in *Proceedings of the 29th International Conference on Machine Learning (ICML)*, pp. 1467–1474, Omnipress, 2012.
- [34] T. Gu, K. Liu, B. Dolan-Gavitt, and S. Garg, “Badnets: Evaluating backdooring attacks on deep neural networks,” *IEEE Access*, vol. 7, pp. 47230–47244, 2019.
- [35] J. Samuel, N. Mathewson, J. Capps, and R. Dingledine, “Survivable key compromise in software update systems,” in *Proceedings of the 17th ACM Conference on Computer and Communications Security, CCS ’10*, (New York, NY, USA), pp. 61–72, Association for Computing Machinery, 2010.
- [36] I. Shumailov, Y. Zhao, D. Bates, N. Papernot, R. Mullins, and R. Anderson, “Sponge examples: Energy-latency attacks on neural networks,” in *2021 IEEE European Symposium on Security and Privacy (EuroS&P)*, pp. 212–231, 2021.
- [37] B. Xu, A. Banerjee, M. Urooj, and S. K. Gupta, “Accelerated digital twin learning for edge ai: A comparison of fpga and mobile gpu,” in *2025 IEEE 68th International Midwest Symposium on Circuits and Systems (MWSCAS)*, pp. 149–153, 2025.
- [38] Y. Bolat, I. Murray, Y. Ren, and N. Ferdosian, “Decentralized distributed sequential neural networks inference on low-power microcontrollers in wireless sensor networks: A predictive maintenance case study,” *Sensors*, vol. 25, no. 15, 2025.

- [39] A. G. Frank, L. S. Dalenogare, and N. F. Ayala, "Industry 4.0 technologies: Implementation patterns in manufacturing companies," *International Journal of Production Economics*, vol. 210, pp. 15–26, 2019.
- [40] A. Banerjee and S. K. S. Gupta, "Your mobility can be injurious to your health: Analyzing pervasive health monitoring systems under dynamic context changes," in *2012 IEEE International Conference on Pervasive Computing and Communications*, pp. 39–47, 2012.
- [41] A. Banerjee and S. K. Gupta, "Analysis of smart mobile applications for healthcare under dynamic context changes," *IEEE Transactions on Mobile Computing*, vol. 14, no. 5, pp. 904–919, 2015.
- [42] A. Banerjee, I. Lamrani, and S. K. S. Gupta, "Synthesizing operationally safe controllers for human-in-the-loop human-in-the-plant hybrid close loop systems," in *Pattern Recognition* (A. Antonacopoulos, S. Chaudhuri, R. Chellappa, C.-L. Liu, S. Bhattacharya, and U. Pal, eds.), vol. 15329 of *Lecture Notes in Computer Science*, (Cham), pp. 17–35, Springer Nature Switzerland, 2025.
- [43] C. R. Banbury, V. J. Reddi, P. Torelli, J. Holleman, N. Jeffries, C. Király, P. Montino, D. Kanter, S. Ahmed, D. Pau, U. Thakker, A. Torrini, P. Warden, J. Cordaro, G. Di Guglielmo, J. Duarte, S. Gibellini, V. Parekh, H. Tran, N. Tran, W. Niu, and X. Xu, "MLPerf Tiny benchmark," in *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks* (J. Vanschoren and S. Yeung, eds.), vol. 1, pp. 1–19, 2021.