Received March 19, 2022, accepted May 4, 2022, date of publication May 16, 2022, date of current version June 9, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3175574

Embedded Intelligence: State-of-the-Art and Research Challenges

KAH PHOOI SENG[1,2], (Senior Member, IEEE), AND LI-MINN ANG[3], (Senior Member, IEEE)
[1]School of AI and Advanced Computing, Xi'an Jiaotong-Liverpool University, Suzhou 215123, China
[2]School of Computer Science, Queensland University of Technology, Brisbane, QLD 4000, Australia
[3]School of Science, Technology and Engineering, University of the Sunshine Coast, Moreton Bay, QLD 4502, Australia

Corresponding author: Kah Phooi Seng (Jasmine.Seng@xjtlu.edu.cn)

ABSTRACT Recent years have seen deployments of increasingly complex artificial intelligent (AI) and machine learning techniques being implemented on cloud server architectures and embedded into edge computing devices for supporting Internet of Things (IoT) and mobile applications. It is important to note that these embedded intelligence (EI) deployments on edge devices and cloud servers have significant differences in terms of objectives, models, platforms and research challenges. This paper presents a comprehensive survey on EI from four aspects: (1) First, the state-of-the-art for EI using a set of evaluation criteria is proposed and reviewed; (2) Second, EI for both cloud server accelerators and low-complexity edge devices are discussed; (3) Third, the various techniques for EI are categorized and discussed from the system, algorithm, architecture and technology levels; and (4) The paper concludes with the lessons learned and the future prospects are discussed in terms of the key role EI is likely to play in emerging technologies and applications such as Industry 4.0. This paper aims to give useful insights and future prospects for the developments in this area of study and highlight the challenges for practical deployments.

INDEX TERMS Embedded systems, SoC, FPGA, GPU, parallel architecture, machine learning, deep learning, IoT, edge AI.

I. INTRODUCTION

Recent years have seen the paradigms for artificial intelligence (AI) technologies and IoT emerging and evolving from studies conducted in universities and research laboratories to become technologies which impact consumers and society. These earlier studies were mainly software-based and executed on general purpose computers. In recent years, Edge AI, AI for Edge IoT and On-Device AI are the emerging terms which are closely associated with AI [1]–[3]. In contrast to software-based AI approaches, these approaches also need to consider the hardware platforms on which the AI and embedded hardware technologies are deployed. Thus, there is an interplay and the emergence of a new discipline between "intelligence" and embedded systems. We refer to these approaches as embedded intelligence (EI) technologies for ease of discussion. The authors in [4] discuss two categories of future AI systems: (1) AI systems which replace the human in the control loop; and (2) AI systems which exhibit humanistic intelligence to collaborate with human beings. An example of Type (1) would be using AI for Advanced Driver Assistance System (ADAS) and of Type (2) would be using AI for facilitating interaction with human users through mobile devices. In this scenario, AI techniques like face, emotion and gesture recognition have a vital role to play.

These EI examples serve to demonstrate the need for high performance AI algorithm performance, real-time decision-making, energy efficiency and low power consumption to realize EI for real-world and practical IoT deployments. There are several challenges which need to be addressed to realize the vision for practical EI deployments which can be on cloud servers or edge devices: (1) Real-time decision making – a good example to illustrate this would be the ADAS which takes in inputs from on-board vehicle sensors and cameras and needs to make an almost instantaneous decision on the course of action to be undertaken. In this scenario, the AI processing (e.g., classification of objects on the road, identification of hazards, etc.) needs to be executed in (near) real-time using on-board or edge AI processors. There is insufficient time (or latency) to transfer the

The associate editor coordinating the review of this manuscript and approving it for publication was Bo Pu.

data to a centralized cloud to perform the AI processing; (2) Energy efficiency and low-power consumption – due to AI technologies being deployed on mobile and sensor devices or nodes, the constraints of battery-life and energy efficiency become significant. Techniques to distribute and balance AI tasks which can be performed on the edge and on the centralized cloud are also important to realize the goal of energy efficiency; (3) AI algorithms which are deployed on the edge or on-board processing platforms still require good algorithm and classification performance; and (4) Security for EI deployments. A final point to note is that the challenges and requirements for edge EI devices may be vastly different from the challenges and requirements for EI accelerators for servers. An EI server accelerator may prioritize the capability to be able to service diverse tasks from multiple streams and give high precision results whereas the EI edge device may prioritize energy efficient processing of tasks and be able to accept lower precision results.

There are various approaches and technologies to realize the EI vision: (1) Traditional central processing unit (CPU) and microcontroller approaches; (2) Graphics processing unit (GPU) approaches [5], [6]; (3) Field programmable gate array (FPGA) approaches [7], [8]; and (4) System-on-chip (SoC) and field programmable system-on-chip (FPSoC) [10]. There are different techniques and approaches which are available to the designer for deployment to meet specific EI requirements. In this paper, we classify the EI techniques into four categories: (1) System level techniques; (2) Algorithm-level techniques; (3) Architecture-level techniques; and (4) Technology-level techniques. Examples of system level EI techniques would be design/task partitioning (e.g., to decide which parts of the EI are implemented in hardware and which in software), scheduling and load balancing. System-level techniques would often aim for throughput, scalability and flexibility measures. Binary neural networks [11], [12] which tradeoff classification performance for lower complexity are examples of algorithm level techniques. Other examples of algorithm-level techniques would be tiling approaches to be able to accommodate large deep neural networks (DNNs) into a fixed sized memory. Examples of EI techniques at the architecture level include custom computational primitives to perform efficient convolution operations for DNNs, pipelining and approaches to hide the memory latency for computational processing. Examples of EI techniques at the technology level include novel process technologies and implementations such as memristors and analog approaches. For evaluation of the design targets and comparisons amongst the EI techniques, we propose and utilize five criterions: (1) Fast training time; (2) Fast decision-making time; (3) Energy efficiency; (4) Low area implementation; and (5) Scalability and supporting multiple EI algorithms. Some aspects for security and privacy are discussed in the later section of the paper. The different criterions will be further elaborated in Section II after discussions of illustrative EI use cases. Figure 1 shows the four techniques and five criterions for EI.

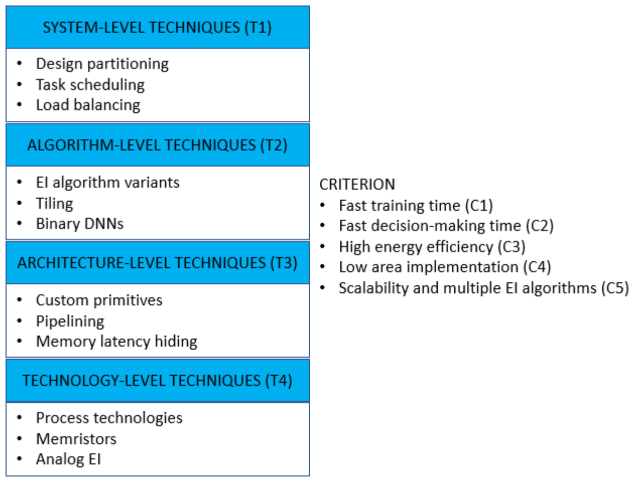

SYSTEM-LEVEL TECHNIQUES (T1)
Design partitioning
Task scheduling
Load balancing
ALGORITHM-LEVEL TECHNIQUES (T2)
EI algorithm variants
Tiling
Binary DNNs
ARCHITECTURE-LEVEL TECHNIQUES (T3)
Custom primitives
Pipelining
Memory latency hiding
TECHNOLOGY-LEVEL TECHNIQUES (T4)
Process technologies
Memristors
Analog EI
CRITERION
Fast training time (C1)
Fast decision-making time (C2)
High energy efficiency (C3)
Low area implementation (C4)
Scalability and multiple EI algorithms (C5)

FIGURE 1. EI techniques and criterions.

This paper presents a comprehensive survey on EI from four aspects: (1) First, the state-of-the-art for EI using a set of evaluation criteria is proposed and reviewed; (2) Second, EI for both cloud server accelerators and low-complexity edge devices are discussed; (3) Third, the challenges and research directions for EI are discussed; and (4) The paper concludes with the lessons learned and the future prospects are discussed in terms of the key role EI is likely to play in emerging technologies and applications such as Industry 4.0. This paper aims to give useful insights and motivate researchers towards the development of EI solutions for practical realizations and applications. The remainder of the paper is structured as follows. Section II presents EI use cases and the proposed evaluation criteria. Sections III, IV and V discusses EI architectures and accelerators for ASIC, FPGA and GPU platforms respectively. Sections VI discusses application specific architectures for EI for various scenarios and techniques such as visual attention, mobile multimedia, health and biomedical, swarm intelligence, evolutionary algorithms). The lessons learned and future prospects for EI are discussed in Section VII. Section VIII gives some concluding remarks.

A. EXISTING SURVEYS ON DNNS AND EI

This Some surveys on implementations of deep neural networks (DNNs) can be found in the literature [13]–[16]. Many of these works focus on implementations of convolutional neural networks (CNNs). There are fewer works which discuss other types of DNNs (e.g., Long short-term memory (LSTM) networks, recurrent neural networks (RNNs), restricted Boltzmann machine (RBM), etc.). The authors in [17], [18] provide an overview of the concepts behind embedded intelligence (EI) and illustrate with some representative survey works. There are some authors which focused on EI and ML implementations for different hardware platforms, such as custom microcontrollers [19], [101], GPUs [20], FPGAs [21], wearables [102]–[105] and IoT devices [106], [107]. Other works have focused on parallel approaches [108] or large-scale processing [109]. Compared to other and earlier works, this paper makes a distinctive and comprehensive contributions to EI from different aspects:

- The review covers general architectures for EI as well as application specific EI architectures. Different types of EI architectures are covered from CNNs to architectures for clustering algorithms, recommendation algorithms, swarm intelligence and evolutionary computing;
- The review for general architectures for EI is divided into EI architectures for three deployment platforms (ASICs, GPUs and FPGAs). A note here is that the various platforms possess different capabilities. Compared to ASICs and FPGAs, GPUs have larger computational capacity, memory and I/O bandwidth, making them more suitable for deployment as EI server accelerators. As a consequence, there are many more papers discussing GPUs as server accelerators than edge devices. We attempt to give a balanced coverage for the different platforms;
- The EI architectures are differentiated to be server accelerators (suitable for data centres and cloud deployments) and EI architectures for edge devices (suitable for low complexity IoT deployments);
- To aid designers, the paper gives a classification and discusses different techniques for EI implementations from four aspects or levels (system-level, algorithm-level, architecture-level, and technology-level techniques). These techniques are then used for comparative discussions throughout the paper;
- The paper proposes five criterions for EI architectures (fast training time, fast decision-making time, energy efficiency, low area implementation, scalability and supporting multiple EI algorithms). These criterions are then used for comparative discussions throughout the paper; and
- The paper includes several tables showing the various types of EI architectures and categorized using the different techniques levels and criterions proposed in this paper.

Furthermore, the paper discusses some lessons learned and the future prospects in terms of the key role EI is likely to play in emerging technologies and applications such as Industry 4.0.

II. EI USE CASES AND EVALUATION CRITERIA

This section introduces illustrative use cases that discusses some useful techniques for EI implementations, server accelerators and edge devices. This is followed by the various criterions derived from the use cases.

A. EI USE CASES

DianNao family of hardware accelerators for machine learning [22]. These are custom computing EI implementations which have been designed with a broad application scope. There are currently four architectures in the DianNao hardware family: (1) DianNao; (2) DaDianNao; (3) ShiDianNao; and (4) PuDianNao. Each of these architectures can serve as useful illustrative examples for EI and the various criterions and objectives to be achieved. In this section, we will first give an overview of the DianNao architecture. Next, we will discuss use cases for EI server accelerators using DaDianNao and PuDianNao, and EI edge devices using ShiDianNao.

1) DianNao EI ACCELERATOR

DianNao [23] was the first member of the EI family to accommodate advanced neural network architectures and requirements. Figure 2 shows the architecture for DianNao. The architecture consists of the following components: (1) Neural Functional Unit (NFU) – The NFUs implements the computational primitives to perform the computations. It includes multipliers, adder trees and non-linear functional units and is designed as a pipeline; (2) On-chip memory storage – The on-chip storage are implemented as three memory structures to correspond to the three types of storage requirements for input neurons, output neurons and synapses. In DianNao, these storage (buffer) structures are termed as NBin, NBout and SB. The memory design uses a custom design to minimize memory transfer latency and enhance system efficiency. The storage structures are implemented as scratchpad memory to avoid cache access overheads and cache conflicts. To exploit spatial locality, three DMAs are included to serve each of the buffers. At the algorithm level, DianNao uses loop tiling to minimize memory accesses and to accommodate large neural networks to fit in its buffers.

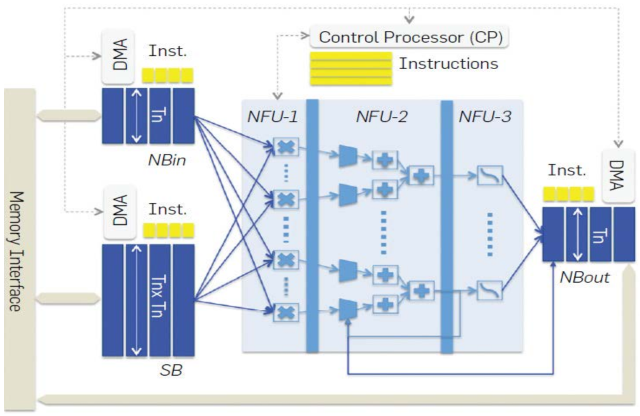


FIGURE 2. Architecture of DianNao [23].

2) USE CASE FOR EI SERVER ACCELERATORS – DaDianNao and PuDianNao

DaDianNao [24] was designed to accommodate very large-scale neural network architectures – i.e., to function effectively as a ML supercomputer. In this work, the authors identified the critical issue for accommodating large neural network architectures is the memory storage for the convolution and classifier layers. Figure 3 shows the architecture for DaDianNao which consists of a set of nodes (or tiles, processing elements (PEs)) arranged in a mesh topology. The design principle is to minimize data movement by storing

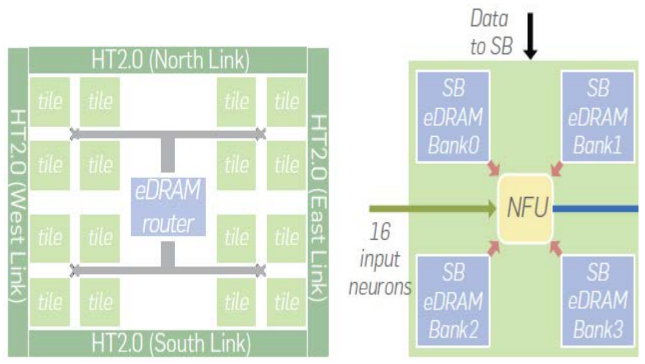


FIGURE 3. Architecture of DaDianNao [24].

synapses close to their respective neurons. A central idea is that the architecture is distributed and does not utilize a main memory storage. Each node contains its own NFU and four RAM banks to store the synapses. To increase the storage density, the memory banks are implemented using eDRAM technologies. The DaDianNao architecture was implemented in a 28 nm process. The architecture has an area of 67.73 mm^2 and a peak performance of 5585 GOP/s at 606 MHz. The performance is 21 times higher than a modern GPU (NVIDIA K20M) and demonstrates its high classification performance.

The DianNao and DaDianNao architectures were designed to only accommodate neural network architectures. Another member of the family, PuDianNao [25] was designed as a hardware accelerator to accommodate seven different ML techniques (*k*-means, *k*-NN, naïve Bayes, SVM, linear regression, classification tree and DNN). Figure 4 shows the architecture for PuDianNao. To accommodate different ML techniques, the functional units (FUs) or computational primitives of PuDianNao are designed as machine learning units (MLUs) for computation of primitives such as dot product, distance calculations and nonlinear functions. Each FU also has an arithmetic logic unit (ALU) to perform computations not supported by the MLU such as division and conditional assignment.

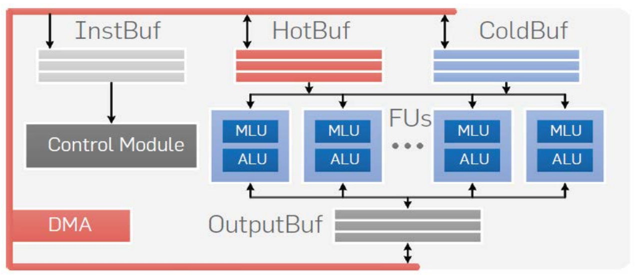


FIGURE 4. Architecture of PuDianNao [25].

3) USE CASE FOR EI EDGE DEVICE – ShiDianNao

ShiDianNao [26] was designed for low-complexity embedded vision systems and serves as a useful use case for EI edge devices. In particular, we will classify ShiDianNao as an application specific vision processor edge device. The ShiDianNao processor implements the CNN algorithm to achieve high energy efficiency. Figure 5 shows the architecture for ShiDianNao.

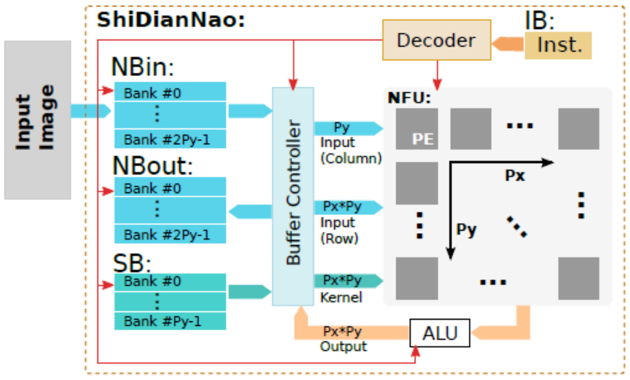


FIGURE 5. Architecture of ShiDianNao [26].

Similar to DianNao, there are three memory structures (NBin, NBout and SB). The input image pixels are stored in the NBin buffers. The NBout and SB buffers store the output neurons and synapses. For computational primitives, the architecture contains an NFU for neuron operations (multiplications, additions and comparisons) and an ALU for activation function operations. The computations are performed in 16-bit fixed point format instead of 32-bit floating point resulting in a reduced hardware area and cost for implementation. The ShiDianNao architecture is optimized to perform computations and exploit the locality present for two-dimensional (2D) data which is central to processing for the CNN. The NFU is implemented as a 2D ($P_x \times P_y$) mesh of processing elements (PEs) which fits the topology of 2D feature maps for CNN implementations. The architecture also contains a decoder for instructing the PEs on the operations to be performed. For completeness, the architecture of the ShiDianNao PE is shown in Figure 6.

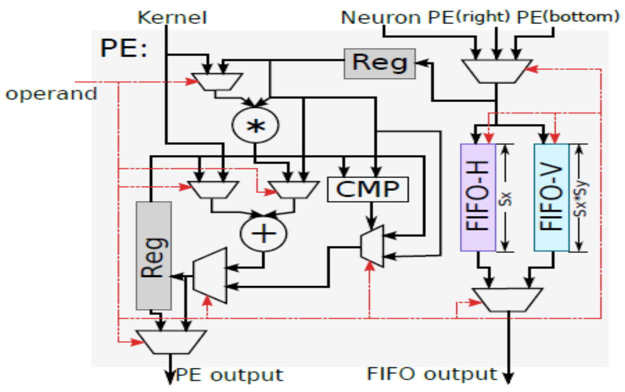


FIGURE 6. Architecture of ShiDianNao PE [26].

Each PE has the computational capability to implement different primitives such as a multiplication, an addition for a convolutional, classifier or normalization layer, an addition for pooling or max pooling layer, etc. Each PE has three inputs: (1) The control operations to be performed for the cycle; (2) The kernel values from the synapse buffer (SB); and (3) The neuron values from either the input or output neuron buffers (NBin or NBout) or from the right or bottom PE. Each PE has two outputs: (1) The output computation results to NBout or NBin; and (2) The output for propagating neurons to neighbouring PEs to achieve efficient data reusage. The ShiDianNao architecture was implemented in a 65 nm process. The architecture has an area of 4.86 mm^2 and a power consumption of 320.10 mW at 1 GHz. The power consumption is 60 times lower than a modern GPU (NVIDIA K20M) and demonstrates its energy efficiency.

The DianNao use cases discussed in this section has demonstrated the utilisation of various techniques for EI deployments at different levels. In particular, we can differentiate techniques applied at four levels: (1) System-level techniques (T1); (2) Algorithm-level techniques (T2); (3) Architecture-level techniques (T3); and (4) Technology-level techniques (T4). The first level techniques (T1) are system level techniques which aim to increase efficiency and performance when there are multiple EI objectives to be achieved. An example is the design of the PuDianNao MLU which has been designed to have the capability to execute EI operations for seven ML algorithms. Other examples would be EI server accelerators which are composed of multiple EI platforms or have the capability to compute algorithms from multiple data device streams. Techniques to optimize parallel processing will be classified as system-level techniques. In these cases, the importance is towards increasing the throughput of the EI accelerators for several tasks, and of lower importance would be the execution time for individual tasks. The second level techniques (T2) are algorithm level techniques which aim to aim to increase efficiency and performance by tuning the EI algorithms. An example is the usage of 16-bit fixed point format in ShiDianNao instead of 32-bit floating point. Other examples are loop unrolling techniques and performing computations using simpler activation functions such as rectified linear unit (ReLU). In T2 techniques, the designer makes adjustments to the EI algorithms to make it more suitable for hardware implementation. This may cause design tradeoffs between classification performance (which may decrease) and simpler hardware structures. Another example is the usage of loop tiling to minimize memory accesses and to accommodate larger neural networks to fit in the EI architecture buffers

The third level techniques (T3) are architecture level techniques which aim to increase efficiency and performance for specific EI operations or objectives. This is usually achieved by designing custom computational and hardware structures. An example is the NFU for DianNao which has been customized to perform efficient computations for neural network architectures. The usage of memory scratchpad instead of cache in DianNao is another example of an architecture level technique to avoid cache overheads and conflicts. The 2D mesh PE array utilized in ShiDianNao and the custom PE node also shows exploitation of T3 techniques. The fourth level techniques (T4) are technology or circuit level techniques which aim to increase efficiency and performance by exploitation of advanced technologies or circuit characteristics. For example, an EI architecture which is implemented using 28 nm process would have lower area and power consumption than an identical architecture implemented using 68 nm process. Other examples would be custom EI architectures using memristors [27] or the analogue characteristics of devices.

B. CRITERIONS FOR EI

In this section, we propose and discuss a set of criterions for EI server accelerators and edge devices which have been derived from the EI use cases. In particular, we can differentiate five criterions: (1) Fast training time (C1); (2) Fast decision-making time (C2); (3) High energy efficiency and low power consumption (C3); (4) Low area implementation (C4); and Scalability and flexibility to accommodate multiple EI algorithms (C5). The first criterion is the requirement for fast training. This criterion would be more applicable towards EI server accelerators than EI edge devices. In many implementations, the training for EI edge devices is performed offline. The trained network parameters are then hardcoded into the device memory banks to perform the inference tasks. The second criterion (C2) is the requirement and capability for rapid and fast decision making or inference. This criterion is important for EI architectures which are deployed as edge devices to increase safety operations such as the ADAS discussed in the introduction of this paper. This criterion is also important for EI edge devices working in tandem with EI server accelerators through communication networks to reduce and minimize the overall round trip latency time for decision making.

The third criterion (C3) is the requirement for high energy efficiency. This is important for EI edge devices which are powered by battery-powered and other limited energy sources, and plays a particularly important role for EI devices in IoT/fog and Industry 4.0 applications. The criterion is also applicable in the design of energy efficient accelerators used in large data centres. The fourth criterion (C4) is the requirement for low area implementations which is particularly critical for EI applications for wearable devices and body sensor networks. This criterion also relates to the model size and memory storage requirements for the EI implementations. The fifth criterion (C5) is the requirement for the EI architecture to have the capability to accommodate scalable implementations and/or have the flexibility to support multiple EI techniques or algorithms. Note that we have not included a criterion on classification accuracy. The assumption is that the classification accuracies for the EI architectures are sufficient to meet the needs of the specific deployments and applications. The next sections will present a review of EI architectures using these criterions as a basis for discussions throughout the paper.

TABLE 1. Summary of representative studies for EI architectures – ASIC.

					Techniques				Criterion					
Year	Focus of work	Algorithm	S	E	T1	T2	T3	T4	C1	C2	C3	C4	C5	Ref
2021	Hybrid precision CIM architecture	DNN/CNN		√				√		√	√			[29]
2021	CIM precision-programmable CNN inference	CNN		√				√		√	√			[28]
2021	ReRAM-based CIM architecture for LSTM computations	LSTM		√				√		√	√			[30]
2021	Hybrid digital-CIM architecture and novel dataflow	RNN		√				√		√	√			[31]
2018	Pipeline ReRAM-architecture for GAN	GAN		√				√		√	√			[32]
2019	Approximate computing reconfigurable architecture (ARA)	CNN		√		√	√			√	√	√		[37]
2021	CNN inference and approximate multipliers	CNN		√		√	√			√	√	√		[33]
2019	Approximate multipliers for CNN and MLP	CNN/MLP		√			√				√	√		[34]
2018	Low-power CNN SoC with DSPs	CNN		√			√				√			[38]
2020	Dual core deep learning accelerator for smartphone edge devices	Multiple		√			√				√		√	[39]
2018	Hardware acceleration for online ELM	ELM		√			√			√	√			[40], [41]
2017	Intelligence Boost Engine (IBE) for sensor hub SoCs	SVM		√			√				√		√	[42]
2017	Configurable and scalable neural coprocessor	ML		√			√				√			[43]
2017	Low-power robust deep tree search in AI SoCs	AI tree search		√			√				√			[44]
2017	HW-accelerated DCNN SoC	CNN		√			√				√			[45]

S – server accelerator E – edge device
T1 – system-level T2 – algorithm-level T3 – architecture-level T4 – technology-level
C1 – fast training C2 – fast decision-making C3 – low power C4 – small area C5 – scalability/flexibility

III. ARCHITECTURES AND TECHNIQUES FOR EI – ASIC

This section discusses architectures and techniques for EI for deployment on application specific integrated circuits (ASICs). The discussions will utilize the four categories (system-level – T1, algorithm level – T2, architecture level – T3 and technology level -T4) techniques and five criterions (C1 – C5) described in Section II. For ease of discussions, we have divided the topics into three categories: (1) Computing-in-Memory (CIM) approaches; (2) Approximate computing approaches; and (3) Low-power SoC approaches. Table 1 shows a summary of the ASIC EI architectures discussed in this section. The table also gives an indication of the suitability of the architecture for server accelerators – S and/or edge devices – E.

A. COMPUTING-IN-MEMORY APPROACHES AND TECHNIQUES

A recent trend for ASIC EI deployments is to exploit non-digital or novel technologies and characteristics of circuits to realize efficient EI implementations. Examples of these approaches can be categorized as computing in memory (CIM) or also termed as in-memory computing (IMC) architectures. CIM are examples of technology level techniques which in this case enables low power consumption and high energy efficiency for EI architectures. There are different types of CIM technologies based on non-volatile memories (NVMs) such as ReRAM or NOR flash. CIM can be characterized as non-Von Neumann computing structures. Figure 7 shows the key difference between a conventional von Neumann and the CIM computing structures [28]. The von Neumann structure requires data to be transported from buffer/memory storage to the computing elements for processing, resulting in increased energy requirements. In contrast, the key idea for the CIM structure is to perform the computations inside the memory structure to reduce the data movements and increase energy efficiency.

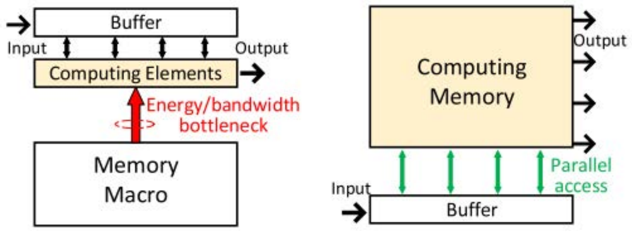


FIGURE 7. Von-Neumann and CIM computing structures [28].

The authors in [29] proposed a hybrid precision CIM architecture. The CIM architecture proposed by the authors consists of high precision layers and many binarized layers. The CIM computation scheme has the capability to support both high-precision multi-bit (MB) and binarized operations (BIN) schemes. The BIN operations trade off the classification performance for a simpler architecture and are examples of algorithm level techniques to reduce complexity. The MB datapath contains components like 8-bit DACs, 4-bit NOR flash cell arrays, current-to-voltage (ITV) converters and dual-mode ADCs. The BIN datapath contains components like a configurable wordline driver, a 1-bit cell array and ADCs. The network weights are stored in an MB array or a BIN array. The convolutional layers are computed and sent to the MB or BIN datapaths. The proposed architecture was implemented on a 0.22μm CMOS process. Their experimental results showed that the hybrid CIM architecture could achieve a 90% accuracy for various applications and a 2.15 TOPS/W power efficiency. Another example for the CIM approach for EI can be found in the work by authors in [28]. In this work, the authors designed a static RAM macro termed as CAP-RAM to leverage a specifically designed charge-domain multiply-and-accumulate (MAC) mechanism and circuitry to achieve energy-efficient and accurate inference for CNN. Their architecture was able to achieve 98.8% and 89.0% inference accuracy on the MNIST and CIFAR-10 datasets, and gave a peak throughput of 573.4 GOPS and energy efficiency of 49.4 TOPS/W.

The CIM approach has also been exploited to enable energy efficiency for other EI algorithms such as LSTM, RNN and GAN. The authors in [30] proposed a ReRAM-based CIM architecture for LSTM computations. The CIM architecture can be characterized as an externally-digital, internally-analog computational machine. The network weights are stored as conductances in the memory elements. The computationally intensive matrix vector multiplication (MVM) for the LSTM can be efficiently implemented by exploiting the CIM architecture. The DACs drive the bitlines by voltages which are proportional to the input activations and causes induced currents which is the sum of the activation weights. The current can be integrated onto a capacitance and final voltage values read out through the ADCs. Their architecture was able to achieve 90% inference accuracy on the Google speech dataset and energy efficiency of 25uJ/decision.

The authors in [31] proposed a CIM architecture for RNN computations. Their approach utilized a hybrid digital-CIM architecture and a novel dataflow to reduce 85.7% of memory accesses. Their architecture was able to achieve 90.2% inference accuracy, processing speed of 127.3 μ s/inference and energy efficiency of 5.1 pJ/neuron. The authors in [32] proposed a pipeline ReRAM-based CIM architecture for GAN computations termed as ReGAN. Their approach utilized a pipeline architecture for the layer-wise computations to increase the system throughput. Their architecture was able to achieve an average performance improvement of 240 times over a GPU implementation and a 94 times improvement in energy efficiency.

B. APPROXIMATE COMPUTING APPROACHES AND TECHNIQUES

Approximate computing approaches tradeoff computation precision for energy efficiency. These approaches can be applied at different levels of the design process (program, architecture and circuit) and has growing popularity for efficient EI implementations. The authors in [37] proposed an EI architecture framework termed as ARA (Approximate computing based Reconfigurable Architecture) to accelerate CNNs on edge/embedded devices and achieve high energy efficiency. The ARA architecture has three innovations: (1) At the algorithm level, the authors proposed hardware friendly compression framework, dynamic layered CNN structure and kernel shrinking methods to reduce the computing and convolution operations; (2) At the architecture level, the complexity of the computation primitives are reduced by utilizing approximate computing units to improve the energy efficiency. The approximate computing units include a multi-port SRAM LUT based multiplier, precision controllable approximate multiplier and approximate adder with error correction logic; (3) At the system level, the ARA architecture has the capability to accommodate different CNNs by having approximate computing units with a configurable datapath.

The ARA architecture consists of the following modules: (1) ARM7TDMI to function as system controller; (2) Scratchpad memory (SPM); (3) Convolution neuron processing unit (CNPU); and (4) Modules for system scheduling. The CNPU consists of neuron processing element arrays. Figure 8 shows the architecture of a neuron processing element (NPE) which can be reconfigured to operate in four modes: (1) Zero detection mode; (2) MAC mode – for calculation of multiplication, addition, MAC and activation operations; (3) Pooling mode; and (4) Activation mode. The ARA architecture was implemented in TSMC 45 nm process technology and achieved a power consumption of 204 mW@ 1.1 V, 200 MHz and 21.1 mW@ 0.9 V, 40 MHz. The authors remarked that the performance of ARA is 1.51 $\sim$ 4.36 times better than other state-of-the-art accelerators.

The authors in [33] proposed an approach for CNN inference using a precision controller and approximate multipliers. There are two innovations in this work: (1) At the algorithm level, the preprocessing precision controller is used to determine the adequate precision which is required for the EI classification task; (2) At the architecture level, a reconfigurable datapath designs are utilized to select the approximate multipliers (high precision or low precision) to be used based on the input from the precision controller. Figure 9 shows the reconfigurable design with the approximate multipliers for high-precision and low-precision computations. Their architecture was implemented using CMOS 15 nm process, and gave significant improvements in power, speed and area with a minimal loss in classification accuracy when compared

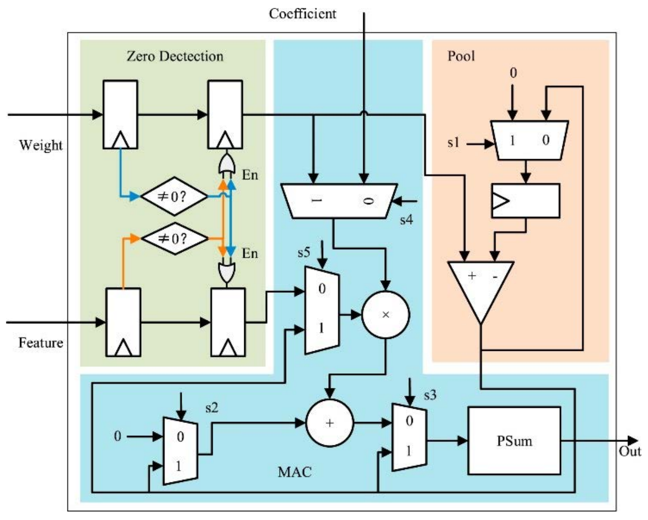


FIGURE 8. NPE architecture of approximate computing-based reconfigurable architecture (ARA) for CNN [37].

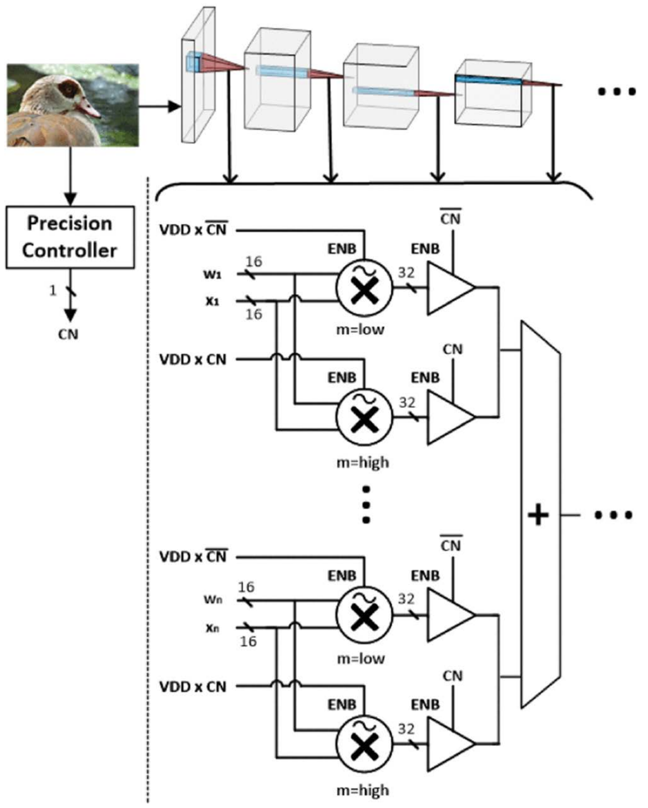


FIGURE 9. Reconfigurable design for approximate computing [33].

with exact multipliers. The classification accuracies for the designs were validated using simulations with VGG19, Xception and DenseNet201 models and the ImageNet dataset.

The authors in [34] performed a study to evaluate a large number of approximate multipliers for application in CNN and MLP neural networks. In this work, 100 deliberately-designed and 500 CGP-based (Cartesian genetic programing) approximate multipliers were evaluated using the MNIST and SVHN datasets. Their experimental results showed that CGP-generated multipliers are better used for application in neural network implementations. Another finding is that networks which use approximate multipliers can provide higher accuracies compared to networks that use the same number of exact multipliers. The approximate multipliers add small amounts of noise which helps to mitigate overfitting.

C. LOW-POWER SOC APPROACHES WITH AI COPROCESSORS

A third trend for EI ASIC accelerators is to exploit the availability of low power SoC containing AI coprocessors. Examples of such SoCs with AI coprocessor capabilities are the Intel Myriad X with a Neural Compute Engine coprocessor [35], Google Coral with a TPU coprocessor [36] and Qualcomm Snapdragon with AIE DSP coprocessor [37]. This section discusses some studies which exploits the SoC capabilities for energy efficient EI architectures. Due to the focus on low power device implementations, these approaches and techniques are mostly used for implementation on EI edge devices.

The authors in [38] proposed integrating the Orlando Ultra Low-Power Convolutional Neural Network with hardware accelerated blocks, DSPs and on-chip memory to implement energy-efficient convolutions for DCNNs. The proposed architecture contains the following components and features: (1) ARM Cortex microcontroller; (2) DSP clusters; (3) Reconfigurable dataflow accelerator with sensor processing pipelines; and (4) Convolutional accelerators. Multiple chips can be connected together through chip-to-chip highspeed serial links of up to 8Gb/s to support larger networks. The Orlando SoC contains other components like configurable accelerator framework (CAF), DMA units, and IO interfaces.

The authors in [39] proposed a power-efficient deep learning accelerator (DLA) for smartphones which aims to minimize memory accesses. Their DLA has the following components and features: (1) Data processing engine with the capability to use different numerical representations; (2) Data-reuse techniques to reduce memory dependencies; (3) Weight compression and zero-activation skipping techniques to reduce DRAM data transfers; and, 4) Parallel execution of DLA cores with L2 SRAM to share each layer's activations and weights. The utilization of two cores is an example of an architecture-level technique. Each DLA core contains components like command engine (CMDE), modules for convolutional (CONV) operations, engines for non-convolutional operations, L1 SRAM shared buffer (SB), and a shared-buffer direct-memory-access (SBDMA) module.

The authors in [40], [41] proposed a SoC hardware implementation for embedded online sequential extreme learning machine (termed as OS-ELM) classification for real-time human action recognition. Their architecture was implemented on SoC FPGA (Zynq-7000) platform for efficient hardware acceleration. Their architecture consists of five modules: (1) Training Module; (2) Prediction Module; (3) Memory Module; (4) Communication Module; and (5) Clock Domain. Their experimental results showed that the OS-ELM implementation reduced the utilization of FPGA logic resources by 19% to 55%. Furthermore, the proposed architecture was two hundred times faster than an ARM Cortex-A9 for matrix inversion operations with a power consumption of 157 mW of on-chip power.

The authors in [42] proposed the IBE (Intelligence Boost Engine) which is a hardware accelerator to support applications for sensor hub SoCs. The IBE processes the algorithms for sensor fusion and machine learning (e.g., SVM) for the SLH200 sensor hub SoC. The IBE was fabricated in 55nm process. The IBE has components such as a control unit (CU), DMA, internal buffers and PEs (processing elements). An architecture-level technique used in the IBE is to skip multiplication steps for zero elements which are detected in the matrix. The CU stores information about the operational mode and size of input data. The DMA loads the input data and stores the data results. The authors in [43] proposed a configurable and programmable coprocessor core to compute ANNs in heterogeneous SoC. The proposed architecture cooperates with a interconnect IP layer to enable the seamless integration of heterogeneous resources. The prototype of the SoC scalable neural processor contains two coprocessors and two BRAMs connected to a companion MAI core. The architecture consists of a CPU, coprocessor units, Block RAMs (BRAMs) and UART for communication to external devices.

The authors in [44] proposed a low-power robust deep tree search transposition table (TT) with the following components and features: (1) On-chip/off-chip hybrid TT architecture to reduce the hit latency; (2) On-chip buffer cache to reduce the hit latency/off-chip memory access, and to prevent the write stalls of search cores; and (3) Progress-based entry replacement policy, are proposed to increase the hit rate, and to avoid the hash key collisions. The proposed hybrid TT is fabricated in a 65nm CMOS technology. The 64-bit bus is used for data communications between the 8 tree search cores and the L1-KTT. For the 2nd level communications between each tree search cluster and L2-BTT, the NoC (network-on-chip) is utilized. The L3-DTT is the off-chip DRAM connected through the external interface. The authors in [45] proposed a hardware accelerated DCNN SoC processor architecture with the following components and features: (1) DCNN convolutional accelerators for kernel compression; (2) On-chip reconfigurable data-transfer to reduce on-chip and off-chip memory traffic, (3) DSP array to support image processing and computer vision applications; (4) ARM-based host subsystem; (5) High-speed IO interfaces for transferring the image and sensor data; and (6) Chip-to-chip multilink to pair multiple devices. The SoC architecture includes the Convolutional Accelerator (CA) as shown in the figure. The authors in [4] surveyed the recent trends demonstrated in chip implementations of CMOS DNN accelerators. Their work aimed to look at real-chip implementations to present a contrast with discussions of unimplemented DNN chip architectures.

IV. ARCHITECTURES AND TECHNIQUES FOR EI – FPGA

Big This section discusses architectures and techniques for EI deployment on field programmable gate arrays (FPGAs). The discussions will utilize the four categories (system-level – T1, algorithm level – T2, architecture level – T3 and technology level -T4) techniques and five criterions (C1 – C5) described in Section II. For ease of discussions, we have divided the topics into four categories: (1) Approximate computing approaches; (2) Hardware-software co-design approaches; (3) Model reduction approaches; and (4) Design workflows. Table 2 shows a summary of the FPGA EI architectures. The table gives an indication of the suitability of the architecture for server accelerators – S and/or edge devices – E.

A. APPROXIMATE COMPUTING APPROACHES AND TECHNIQUES

The authors in [46] proposed the design and implementation of systolic neural network accelerator (termed as SNNAP) in programmable logic. The SNNAP is a flexible FPGA-based neural accelerator for approximate programs and automatically configures the topology of multi-layer perceptron (MLP) neural network and weights instead of the programmable logic itself. The design of the SNNAP is based on systolic arrays. The SNNAP architecture was implemented on the Zynq PSoC (programmable system-on-a-chip). The architecture of SNNAP consists of a cluster of Processing Units (PUs) connected through a bus. Each PU contains a chain of Processing Elements (PE). The PEs form a systolic array that feeds into the sigmoid unit. The authors in [47] (Wang, 2018) performed a study on the tradeoff between real-time processing (Quality of Service – QoS) and computation approximation (Quality of Result – QoR) for a CNN inference accelerator. The authors proposed an architecture termed as ELNA (ELastic Neural acceleration Architecture) to support multiple precision modes, optimization methods and accommodate different ranges of network parameters. The ELNA accelerator contains two main components: (1) Reconfigurable CNN accelerator; and (2) Software CNN synthesizer. The reconfigurable CNN accelerator can be switched to perform different operation modes. The CNN synthesizer is used to search the different CNN topologies to meet the QoR and QoS constraints. The ELNA accelerator utilizes a SIMD structure to fetch network weights and data for computation in the PEs. The architecture contains a 16 $\times$ 16 PE array, a weight buffer (375 KB) and two data buffers (128 KB). The ELNA architecture was implemented on a Xilinx Zynq-7020 clocked at 100 MHz. Their experimental results showed that ELNA could outperform conventional approaches by 316% when meeting the QoR requirement, and the architecture consumes 20% less energy when meeting both QoS and QoR requirements.

TABLE 2. Summary of representative studies for EI architectures – FPGA.

					Techniques				Criterion					
Year	Focus of work	Algorithm	S	E	T1	T2	T3	T4	C1	C2	C3	C4	C5	Ref
2015	Systolic neural network accelerator in programmable logic (SNNAP)	MLP		√			√			√	√			[46]
2018	Elastic Neural Acceleration Architecture (ELNA)	CNN		√			√			√	√			[47]
2020	FPGA accelerator for clustering	Clustering	√	√		√	√		√	√	√		√	[48]
2020	Custom instruction set architecture recommender	Collaborative filtering	√	√		√	√		√	√	√		√	[49]
2018	Systolic co-processor for DNN inference.	DNN		√			√			√				[50]
2020	High performance accelerator architecture with DSP and scheduling	CNN		√			√		√	√				[51]
2020	Energy efficient fast convolution for deep CNN	CNN		√			√			√	√			[52]
2020	Advanced feature point extraction for CNN	CNN		√		√	√			√				[53]
2018	Throughput optimization using batch processing and pruning			√			√			√				[54]
2020	Workflow for mapping & programming low-cost FPGA SoCs	CNN		√			√			√				[55]

S – server accelerator E – edge device
T1 – system-level T2 – algorithm-level T3 – architecture-level T4 – technology-level
C1 – fast training C2 – fast decision-making C3 – low power C4 – small area C5 – scalability/flexibility

B. HARDWARE-SOFTWARE CODESIGN APPROACHES AND TECHNIQUES

Hardware-software co-design approaches aim to exploit the reconfigurable capability of FPGAs and/or in-built components (e.g., DSP for computation, BRAM for storage) for custom based architectures (e.g., pipelined, systolic, parallel) for EI. The instruction set architecture (ISA) and dataflow may need to be re-designed (e.g., unrolling of loops for convolutions) to fully realize the advantages of the custom architectures.

Some examples of ISA customization approaches for EI can be found in [48], [49]. The work in [48] proposed an accelerator architecture which was deployed on FPGA for clustering algorithms. The proposed architecture has several novelties: (1) Combination of a basic ISA with a custom ISA tailored towards the clustering applications for increased computational performance and efficiency; (2) Capability and flexibility to accommodate four established clustering algorithms (k-means, DBSCAN, PAM and SLINK) and two distance measure computations (Euclidean and Manhattan distances) to support a wide variety of EI applications; and (3) Applying tiling techniques to improve the efficiency of memory accesses.

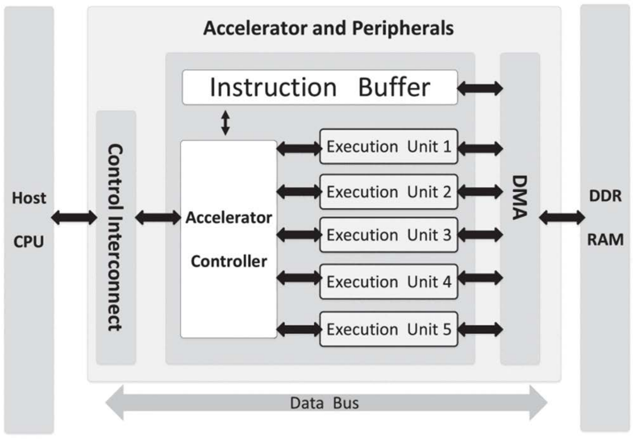
Figure 10(a) shows the custom ISA which has been proposed for the clustering accelerator. The authors derived the custom ISA by identifying the computational primitives for the various clustering algorithms to be supported (e.g., Euclidean distance computations can be accelerated by utilizing the VECTOR_SUB, VECTOR_MULT and SCALAR_SUM instructions, whereas Manhattan distance computations can be accelerated by utilizing the VECTOR_SUB, VECTOR_FAB and SCALAR_SUM instructions). Figure 10(b) shows the accelerator datapath architecture. The accelerator datapath contains the following components: (1) Instruction buffer; (2) Accelerator controller; and (3) Multiple execution units. The accelerator is operated in SIMD mode through running instructions on the execution units in parallel. For the memory accesses, the architecture uses tiling techniques to reduce the number of off-chip memory accesses for the distance calculations.

The objective is to exploit the data locality aspects in the algorithm for the memory accesses by utilizing a small on-chip memory. The loops of objects and cluster centroids are tiled and each tile is defined for the distance calculations between the T objects and the S cluster centroids. The EI clustering architecture was implemented on a Xilinx Zynq FPGA. Their experimental results showed that the accelerator could give a 23 times speedup compared to Intel Xeon processors, and an average energy efficiency ratio of 78.51 times and 9.46 times compared to CPU and GPU baselines. The overall design of this EI clustering architecture illustrates the various approaches and techniques available to the designer to meet the different criterions as discussed in Section II.

The authors in [49] demonstrates another example of hardware-software co-design approach using custom instruction sets for EI architectures. In this work, the authors

Instruction	Functionality
LOAD_OBJ	Read Data from DMA to Objects Arrays in FPGA
LOAD_CLU	Read Data from DMA to Means Arrays in FPGA
LOAD_TMP	Read Data from DMA to TMP Arrays in FPGA
LOAD_PARA	Read Data from DMA to PARA Arrays in FPGA
STORE_TMP	Send Data from TMP Arrays in FPGA to DDR
STORE_ID	Send Data from ClusterID Arrays in FPGA to DDR
STORE_DIST	Send Data from ClusterID Arrays in FPGA to DDR
STORE_LOCAL	Send Data from ClusterID Arrays in FPGA to DDR
VECTOR_SUB	Vector Sub
VECTOR_FAB	Vector ABS
VECTOR_MULT	Vector Sub
SCALAR_SUM	Vector Multiplication
FIND_MIN	Find Minimum in Arrays
VECTOR_ADD	Vector Sub
VECTOR_DIV	Vector Division
VECTOR_UPD	Vector Update Cluster ID

(a) Accelerator instruction set architecture (ISA).



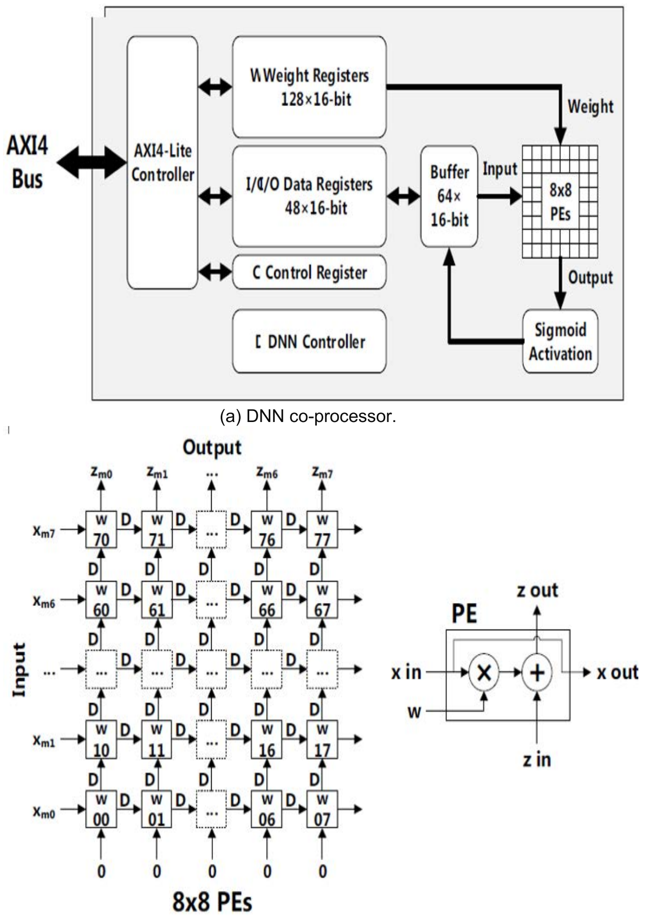
(b) Accelerator datapath architecture.

FIGURE 10. FPGA clustering accelerator [48].

proposed an EI accelerator architecture termed as WooKong which was deployed on FPGA for recommendation algorithms, focusing on neighborhood-based collaborative filtering (CF). The architecture has the capability to support different CF recommendation algorithms (user-based CF, item-based CF and SlopeOne) together with supporting various similarity metrics (Jaccard, Cosine, CosineIR, Euclidean and Pearson). The datapath architecture has capabilities for both learning/training and prediction. The architecture uses a custom instruction set architecture for mapping the three types of CF recommendation algorithms for the learning and prediction computations. The EI architecture was deployed on Xilinx Zynq FPGA. Their experimental results showed that the learning and prediction accelerators could achieve a computational speedup of 8.0 and 1.7 times respectively compared with an Intel i7, and energy efficiency of 137.4 times compared with an NVIDIA Tesla K40C GPU.

Some examples of architecture customization approaches for EI can be found in [50]–[53]. The authors in [50] proposed a systolic co-processor for DNN inference. The co-processor is utilized for matrix multiplication between the inputs on the DNN layer and the weight values. The co-processor is integrated with an ARM processor (Cortex-A9) through the standard AXI4 bus and deployed on Xilinx Zynq-7000 SoC. Figure 11 shows the architecture for the proposed systolic co-processor for DNN inference. The systolic array is utilized for one matrix multiplication, i.e., the matrix multiplication in one DNN layer. To compute the whole DNN layer, four matrix multiplications are implemented by time multiplexing method in DNN controller. The authors in [51] utilize the DSP of the FPGA and efficient scheduling mode to increase the performance of CNN accelerator. By doing so, the feature maps and weights can be stored in one FPGA chip and significantly reduce the cost on data transfer to the external memory. The two main techniques used in the research are adopting row pass scheduling and merging two weights together to improve the performance. Their experimental results showed that only 31% of DSP is utilized to run the convolutional and pooling part of AlexNet which is significantly higher performance compared to other approaches. There are three main categories of data flow scheduling mode and parallelism strategies. The first is known as "row column channel" and has the highest benefit of reuse efficiency with the trade-off that high storage is required to store the entire data as intermediate variables. The second is known as "channel-row-column" where a small memory is required but all the kernel is unable to reuse. The third is known as row pass or also known as "row-channel-column". The disadvantage of this approach is that the weights and intermediate storage are required. The advantage of this dataflow is that the architecture can be fully pipelined. The control module handles the weights extraction from RAM and improve the loading time of the PE. The researchers used the ping pong buffer techniques to utilize the calculation time for data transmission since the loading time of weights is typically shorter than calculation time. Thus, the time of updating the buffer can be decreased without increasing the storage space for the weights in the layer. Furthermore, the authors also set the register and line buffer to be of equal length, and able to use the cache to calculate the content spontaneously.

The multiplication and addition trees in PEs can be executed in full pipeline and pre-fill zeros into the pipeline calculation by referring to the convolutional mode. The PE architecture for pooling and convolution is similar where the FPGA multiplication and accumulation blocks are used to perform the operation for max pooling or average pooling. The architecture also utilizes dynamic quantization techniques based on the range of the data to prevent the precision of small range data to get impacted. The proposed architecture and associated techniques significantly improve the throughput and resource utilization to reduce the latency between the data transfers. The authors in [52] proposed an approach to improve the energy efficiency for high



(a) DNN co-processor.

(b) Systolic PE architecture.

FIGURE 11. Architecture for systolic co-processor for DNN inference [50].

dimensional convolutional computation and the associated data movements. Their approach had two main strategies: (1) Row stationary to optimize the data reuse for the spatial architecture; and (2) Computations using the one-dimensional Winograd minimal filter algorithm to reduce the multiplication. Their work showed that the dataflow was able to improve the factor up to 1.5 times and the on chip and off chip memory access was able to improve by 1.07 times and 1.46 times respectively.

The authors in [53] proposed an approach to implement advanced CNN-based feature point extraction to run in real time on FPGA embedded system. The FPGA hardware that used by the author is XILINX ZCU102 platform for both CNN and post processing operations. The algorithmic techniques used for the feature extraction is to improve the post processing operations with normalization, ranking and Non-Maximum Suppression (NMS) to reduce the computational complexity. Their experimental results utilized 8-bit fixed point binary representations in the softmax operation and gave insignificant accuracy loss for implementing state-of-the-art CNN based feature extraction methods. The Softmax module consists of three main categories which are adder tree, comparer tree and divider. In the architecture, 65 inputs will be read at once and the adder tree computes the inputs to the power of 2 by using shift register.

C. MODEL REDUCTION APPROACHES AND TECHNQIUES

The authors in [54] investigated throughput optimization using two orthogonal techniques which is batch processing and pruning. They proposed techniques to reduce the data transfer and speed up the network inference. The concept of batch processing is to reuse the weights that is already transferred and store for one section by processing input samples through time division multiplexing before proceeding to the next execution. Some researchers term this as mini-batch processing in the context of stochastic gradient descent which efficiently reduce the data transfer.

Figure 17 shows the architecture for throughput optimization. From the architecture, the researcher utilizes the matrix coprocessor that computes the transfer function to carry out all compute intensive function such as matrix vector that required pruning and batch processing is required. Their work showed that 16 bits are the most common used bit width that able to offer most accurate as single prevision precision floating point weights. Next module is the activation function which consist of large number of comparators and arithmetic operation.

The challenge of using this architecture is to balance between the latency vs the throughput on batch processing due to batch processing is reducing the number of weight transfer. On the analysis of this architecture, the authors highlighted that at least over 70% of parameters is being prune but there are accuracy drops is non-noticeable. This technique can be used in application that can accept a low amount of latency. In summary, the performance and speed able to accelerate the inference that learned fully connected DNN by using the FPGAs-based embedded Soc.

D. DESIGN WORKFLOWS AND TECHNIQUES

The authors in [55] proposed the CNN-Grinder workflow for mapping Caffe mobile-friendly CNNs, such as Squeeze Net and ZynqNet to HLS code which can be used for programming low-cost FPGA SoCs. CNN-Grinder provides developers an approach to map a CNN on an FPGA SoC for customizable CNN architectures and FPGA devices. The workflow is utilized by the SqueezeJet-2 accelerator which implements the convolutional, Rectified Linear Unit (ReLU), and max-pooling CNN layers. Their experimental results showed that their approach could achieve 14.18 frames/second (fps) and 11.69 fps for the SqueezeNet v1.1 and the ZynqNet CNN inference implementing stages on the FPGA SoC device for practical deployment and usage for real-time mobile systems.

V. ARCHITECTURES AND TECHNIQUES FOR EI – GPU

Graphics processor units (GPUs) were designed to support the acceleration of graphics capabilities in hardware. The GPU platform contain a large number (hundreds to thousands) of small computational units or cores structured in a

TABLE 3. Summary of representative studies for EI architectures – GPU.

					Techniques				Criterion					
Year	Focus of work	Algorithm	S	E	T1	T2	T3	T4	C1	C2	C3	C4	C5	Ref
2019	ML-based load balancing scheduler (Troodon)	ML	√		√				√	√				[59]
2020	Queuing model for DNN inference	DNN	√		√					√				[60]
2020	Performance and power analysis of GPU workloads	CNN	√		√				√		√			[61]
2020	Block circulant matrices (BCM) for DNN compression	DNN	√		√	√			√					[66]
2018	Batch orchestration algorithm (BOA) for deep learning models	DL	√			√			√					[67]
2017	Blocked LU decomposition algorithms for GPU training	ELM	√			√			√					[68]
2020	Machine learning for analysis of GPU kernel execution	Multiple	√		√								√	[69]
2019	Residual Gradient Compression (RGC) to accelerate DNN training	DNN	√			√			√					[70]

S – server accelerator E – edge device
T1 – system-level T2 – algorithm-level T3 – architecture-level T4 – technology-level
C1 – fast training C2 – fast decision-making C3 – low power C4 – small area C5 – scalability/flexibility

single instruction multiple data (SIMD) organization and has the capability to accelerate computationally-intensive tasks and workloads in a highly parallel architecture. In recent years, GPU platforms have become useful and is a popular option for the acceleration of EI capabilities in hardware. Some tutorials on GPU architectures and structures can be found in [9], [56]–[58]. This section discusses architectures and techniques for EI deployment on GPUs. The discussions will utilize the four categories (system-level – T1, algorithm level – T2, architecture level – T3 and technology level -T4) techniques and five criterions (C1 – C5) described in Section II. For ease of discussions, we have divided the topics into two categories: (1) Task scheduling and load balancing approaches; and (2) Algorithm and computation approaches. Table 3 shows a summary of the GPU EI architectures discussed in this section. The table gives an indication of the suitability of the architecture for server accelerators – S and/or edge devices – E.

A. TASK SCHEDULING AND LOAD BALANCING APPROACHES

The authors in [59] proposed a load-balancing scheduler termed as Troodon which is machine learning-based scheduling mechanism. The Troodon scheduler contains a scheduling mechanism (termed as E-OSched) for mapping and balancing jobs on CPU and GPUs. The Troodon architecture contains the following components and features: (1) Kernel Feature Extractor; (2) Device Suitability Classifier; and (3) Speedup Predictor. The Kernel Feature Extractor performs the extraction of the OpenCL kernel code-features. The Device Suitability Classifier module performs the classification of jobs based on the device suitability. The Speedup Predictor component performs the prediction of the job's speedup. Their experimental results showed the proposed scheduler could achieve a reduction of the execution time by 38% while giving higher system throughput and device utilization.

The authors in [60] proposed a formulation and model based on service queues for GPU servers. Their work focused on modelling the latency of GPU-based inference servers for different batch sizes and processing times. They performed experiments using three types of networks (MobileNet, ResNet50 and SSD-MobileNet) on the Tesla V100 and Tesla T4 GPUs. Their experimental results showed their approach could give high energy-efficiency within a latency requirement. The authors in [61] proposed a performance and power analysis of CNN workloads on different GPU platforms: (1) NVIDIA DGX-1 (eight Pascal P100 GPUs); and (2) Intel Knights Landing (KNL) CPUs. Their experiments used different CNN topologies such as CifarNet [62], AlexNet [63], GoogLeNet [64], and ResNet [65]. Their implementation gave theoretical equivalence to the sequential algorithm for batch gradient descent. Their experimental results showed that Pascal GPUs gave the highest overall performance.

B. ALGORITHM AND COMPUTATION APPROACHES

This section discusses some examples of algorithm and computation approaches for GPU. A critical issue to be addressed is to speed up the training time for the EI architectures. The authors in [66] proposed an approach to leverage Block Circulant Matrices (BCM) to perform compression of the linear transformation layers and acceleration of the DNN training by reducing the redundant computations for forward and backward propagation. In this work, the authors proposed techniques for: (1) Decomposition and interaction of individual operations; and (2) GPU kernel design and modifications to remove redundant computations, kernel optimizations, and data sharing patterns. Their experimental results using an NVIDIA Tesla V100 and the AlexNet topology gave performance improvements and speedups of 1.31 times and 2.79 times for the convolutional layers and fully-connected layers respectively. The authors in [67] proposed a batch-orchestration algorithm (BOA) to reduce the training time of deep learning model by improving hardware efficiency in GPU clusters. Their approach performs the coordination of mini-batch sizes for workers to reduce the iteration time for training. The batch orchestration process containing the two major stages: (1) GPU stage for computation time estimation; and (2) Coordination stage of the local mini-batch sizes.

The authors in [68] proposed efficient blocked algorithms to implement extreme learning machines (ELM) on GPUs. Their approaches were termed the blocked LU decomposition algorithm, blocked Cholesky decomposition algorithm, and blocked heterogeneous CPU-GPU algorithm. Figure 12 shows the proposed ELM-LRF architecture. The blocked LU decomposition algorithm was proposed to overcome the limitation of global memory size to enable the training of different sizes for the ELM-LRF models. The heterogeneous blocked CPU-GPU parallel algorithm exploits the resources on a GPU to further accelerate the performance of the blocked Cholesky decomposition for ELM-LRF. Their experimental results showed the blocked Cholesky decomposition algorithm could achieve two times performance improvements and speedup in comparison with blocked LU decomposition.

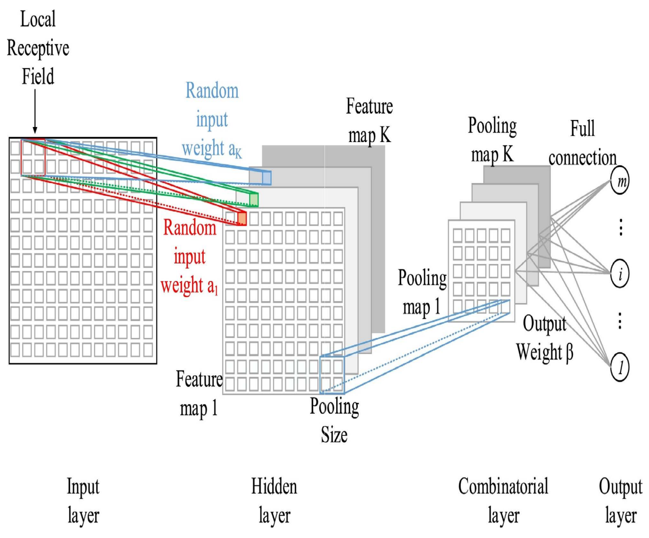


FIGURE 12. Architecture of heterogeneous blocked CPU-GPU for ELM-LRF [68].

The authors in [69] utilized several types of machine learning techniques (k-nearest neighbors, logistic regression, multilayer perceptron and XGBoost) to investigate how the kernel resource requirements could impact the GPU performance. In this work, the authors performed experiments on two different GPUs using the Rodinia, Parboil and SHOC benchmarks: (1) GPU Tesla (Pascal architecture); and (2) GPU RTX 2080 (Turing architecture). Their experimental results showed that features such as number of blocks/grid, number of threads/block and the number of registers are resource consuming parameters and affects the performance of the concurrent execution. The authors in [70] proposed a Residual Gradient Compression (RGC) system termed as RedSync to improve the performance speedup for DNN training by using the RGC to reduce the end-to-end training time on multi-GPU systems. Their approach gave good solutions and addressed two major challenges on multi-GPU systems: (1) High overhead incurred by GPU compression; and (2) Low support for sparse data structures and its data communications. Their experimental results showed that their approach gave significant speedups and performance improvements for high-intensity communication networks such as VGG and AlexNet.

VI. EI APPLICATION-SPECIFIC ARCHITECTURES

This section discusses EI for application-specific architectures from various perspectives: (1) Visual attention EI architectures; (2) Human computer interface (HCI) EI architectures; (3) EI architectures for mobile applications; (4) EI architectures for multimedia and computer vision; (5) Health and biomedical EI architectures; (6) EI architectures for swarm intelligence; and (7) EI architectures for evolutionary and genetic algorithms. Table 4 shows a summary of the EI application specific architectures discussed in this section.

A. VISUAL ATTENTION EI ARCHITECTURES

The authors in [71] proposed an embedded DNN (E-DNN) for energy efficient computation of visual attention (VA). The E-DNN VA processor architecture has two components and features: (1) CNN to realize the top-down VA; and (2) Configurable PE architecture for CNN and MLP computations. The EI algorithms use the same hardware resources for energy efficiency. The E-DNN processor was implemented in 65 nm CMOS process. The system consists of the E-DNN VA processor, a camera sensor, feature detectors for SIFT and filtering accelerator.

The authors in [72] proposed a VA engine (VAE) for saliency-based algorithm based on a digital cellular neural network architecture. The VAE is a hardware accelerator to reduce processing time and increase the energy efficiency. optimized for the saliency-based VA algorithm to speed up object-recognition. The proposed embedded VAE was implemented in 0.13-μ m CMOS process. The VAE contains the following hardware components: (1) Object recognition SoC with eight processing element clusters (PECs); (2) Matching accelerator (MA); and (3) Host RISC processor. The VAE performs saliency-based detection on the image and outputs a pixel map marking the ROIs. The object recognition SoC uses the ROI map to perform object-recognition for the regions of high saliency.

B. HUMAN COMPUTER INTERFACE (HCI) EI ARCHITECTURES

The authors in [73] proposed a gesture and gait processing human computer interaction (HCI) architecture SoC

TABLE 4. Summary of representative studies for application specific EI architectures.

					Techniques				Criterion					
Year	Focus of work	Algorithm	S	E	T1	T2	T3	T4	C1	C2	C3	C4	C5	Ref
2015	Embedded DNN visual attention architecture	CNN, MLP		√		√		√			√		√	[71]
2010	VA engine for saliency-based CNN processing	CNN		√			√				√		√	[72]
2020	Gesture and gait processing SoC for rehabilitation	DNN		√		√	√				√			[73]
2020	EEG emotion classification SoC for autistic children	DNN		√			√				√			[74]
2020	Mobile inference SoC with multicore CPU, GPU, NPU	ANN		√			√			√				[75]
2019	Energy-efficient sparsity aware neural processing unit	Neural processing		√			√				√		√	[76]
2010	High performance machine learning SoC for multimedia	Image and feature stream		√			√			√				[77]
2010	Semantic analysis SoC for video processing and machine learning	Vector-level machine learning		√			√			√	√			[78]
2017	Intelligent Boost Engine (IBE)	Sensor fusion		√			√			√				[79]
2020	Parallel architecture for super resolution	DCNN		√		√	√				√			[80]
2015	Seizure onset and termination detection SoC	SVM		√			√				√			[81]
2015	SoC for FSCV neurochemical sensing	FSCV sensing		√			√	√				√		[83]
2013	Machine learning cardiac sensor SoC	VCG MI detection		√			√					√		[84]
2021	Deep learning COVID-19 chest detection	CNN	√				√			√				[86]
2017	Deep neural network on ultrasound	CNN	√				√			√				[87]
2021	3D semantic liver segmentation	DNN	√				√			√				[88]

S – server accelerator E – edge device
T1 – system-level T2 – algorithm-level T3 – architecture-level T4 – technology-level
C1 – fast training C2 – fast decision-making C3 – low power C4 – small area C5 – scalability/flexibility

for rehabilitation. In their work, the HCI SoC has the following components and features: (1) Mixed-signal feature extraction and integrated low-noise amplifiers; (2) Low-cost and low-power analog front end from training of the DNN classifier; (3) On-chip learning of DNN engine for user specific operations; and (4) On-chip training for user-specific DNN model and multi-chip networking capability. The neural network architecture contains four layers of fully-connected neurons with 12 input neurons, 24 second-layer neurons, 18 neurons for gesture classification and 18 neurons for gait classification.

The authors in [74] proposed an EEG-based emotion classification SoC architecture for autistic children. Their architecture combines a patient-specific (PS) DNN processor with a Analog Front-end (AFE). The proposed EEG-based emotion classification SoC architecture was implemented in 0.18 um CMOS process. The emotion classification processor incorporates the following components and features: (1) Feature Selection (FS) and Classification Engine (CE) for classification of valence and arousal features; and (2) Emotion Decision Logic (EDL) block for classification of four emotion states (happy, sad, relaxed and angry). The DNN consists of an input layer of size 4×8, two hidden layers of sizes 8×16 and 16×32 and an output layer of size 32×1.

C. EI ARCHITECTURES FOR MOBILE APPLICATIONS

The authors in [75] presented an evaluation of mobile SoCs focusing on its capabilities to perform inference. The architecture of the mobile SoC incorporates with multi-core CPU, GPU and NPU. In this work, the authors investigated the power-performance characteristics on the various neural network components within the mobile SoC. Their experimental results demonstrated that the mobile SoC could provide up to two times improvement with parallel inference.

The authors in [76] proposed a neural processing unit (NPU) with the focus on energy efficiency. The NPU has the following features and components: (1) Dual core accelerator with 1,024 MACs; (2) Parallel architecture in computing CLs and FCLs; (3) Feature-map selection and MAC operations; and (4) Bandwidth efficient network traversal. The proposed NPU was implemented in 8 nm CMOS technology.

The NPU has an area of 5.5mm^2, utilizes a supply voltage of 0.5-to-0.8V, and has a clock frequency of 933-MHz. The architecture contains one NPU controller (NPUC) and two NPU cores. The DMA has responsibility for managing the transfer of the weights and feature maps.

D. EI ARCHITECTURES FOR MULTIMEDIA AND COMPUTER VISION

The authors in [77] proposed a machine learning SoC (termed as MLSoC) architecture for accelerating the content analysis of machine learning and computer vision algorithms. The architecture was implemented with an area of16 mm area using 90 nm CMOS technology. The MLSoC contains two components: (1) Image stream processor (ISP); and (2) Feature stream processor (FSP). The ISP and FSP connects to the high-bandwidth dual memory (HBDM) by using the local media bus (LMB). Their experimental results showed that the architecture could achieve a performance throughput of 62.5 Gpixel/cycle and 16 vector/cycle for image processing and machine learning operations respectively.

The authors in [78] proposed a Semantic Analysis SoC (SASoC) architecture for performance speedup and acceleration of machine learning and video processing algorithms. Some example applications for the SASoC include concept-based image retrieval for scene recognition and photo classification in consumer electronics and face detection for camcorders. The architecture contains the following three components and features: (1) Image-Stream Processing System (ISPS) and Feature-Stream Processing System (FSPS). The ISPS performs the pixel-level task for feature extraction whereas the FSPS performs the vector-level machine learning tasks for semantic analysis; (2) Hierarchical memory organization and stream network design; (3) Dynamic frequency scaling and multiple clock domains to improve the energy efficiency and reduce the power consumption.

The authors in [79] proposed the IBE (Intelligence Boost Engine) which is a hardware accelerator to support sensor fusion for predicting the orientation and rotation information based on collected sensor data. The architecture has the following features and components: (1) Processing elements (PEs); (2) Control unit (CU). The CU stores information and performs the required operations for the IBE; and (3) DMA unit which loads and stores the result data independently. Their architecture was fabricated using 55nm technology. With the rapid evolution of CNNs, real time image super resolution (SR) is not commonly implemented in FPGA platform due to the long execution time. The authors in [80] proposed an energy efficient architecture with high parallelization methods for SR application. The research showed that by reducing the data bit width and number of parameters also able to generate high resolution of image.

E. HEALTH AND BIOMEDICAL EI ARCHITECTURES

The authors in [81], [82] proposed an architecture for a patient-specific (PS) seizure onset and termination detection SoC with machine-learning. The SoC architecture contains the following components and features: (1) PS seizure onset and termination detection classification processor. The machine learning classification is performed with a 16-channel feature extraction (FE) engine using a linear SVM; (2) Low-power pulsating voltage transcranial electrical stimulator (PVTES); and (3) On chip SRAM for EEG storage.

The authors in [83] proposed a SoC architecture for neurochemical sensing. The SoC architecture contains the following components and features: (1) DSP unit to perform real-time processing on neurochemical data; (2) Brain-implanted electrodes incorporating fast scan cyclic voltammetry (FSCV); (3) Wireless frequency shift keyed (FSK) transmitter; and (4) Clock generator. The authors in [84], [85] proposed a ML assisted cardiac sensor SoC for mobile healthcare applications. Figure 13 shows the architecture of the proposed CS-SoC. The architecture has the following features and components: (1) Cardiac signal acquisition; (2) Feature extraction and classification with DSP; and (3) Data management processor (DMP) to improve quality and perform data compression for storage. The proposed architecture is implemented in a 90-nm CMOS technology.

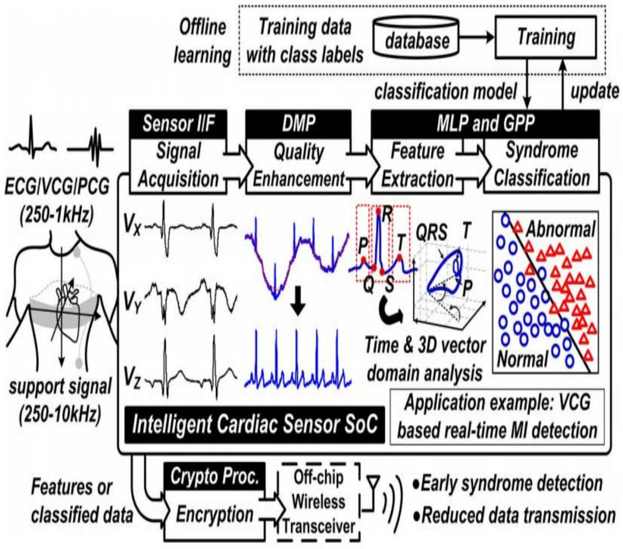
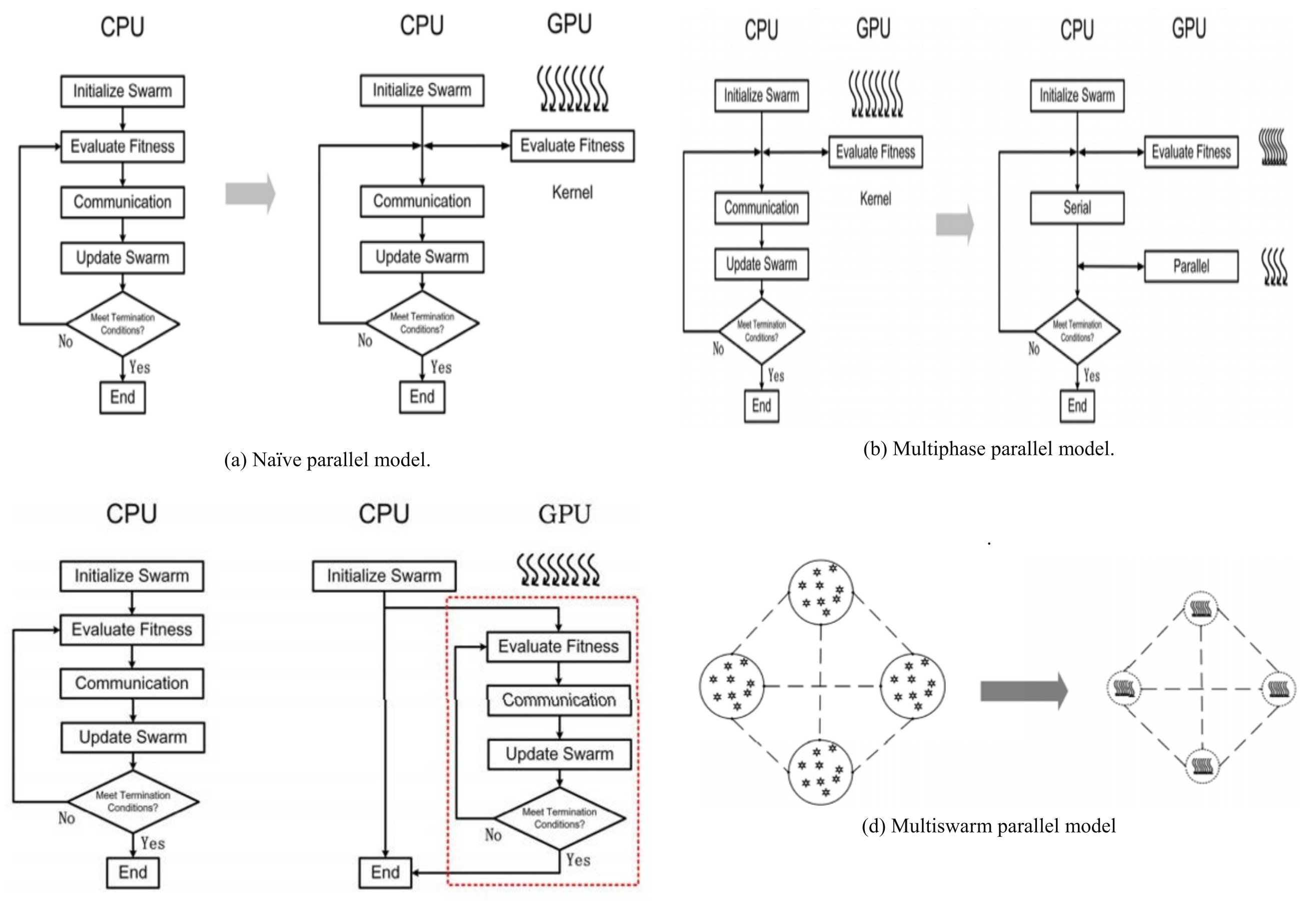

Offline learning
Training data with class labels
database
Training
classification model
update
Sensor I/F
Signal Acquisition
DMP
Quality Enhancement
MLP and GPP
Feature Extraction
Syndrome Classification
ECG/VCG/PCG (250-1kHz)
support signal (250-10kHz)
Vx
Vy
Vz
P R T Q S
QRS T
P
Time & 3D vector domain analysis
Abnormal
Normal
Intelligent Cardiac Sensor SoC
Application example: VCG based real-time MI detection
Features or classified data
Crypto Proc.
Encryption
Off-chip Wireless Transceiver
•Early syndrome detection
•Reduced data transmission

FIGURE 13. SoC architecture for cardiac sensor [84].

The authors in [86] proposed a deep learning architecture to automatically detect and identify the COVID-19 disease in chest X-ray images. They used a dataset containing CXR database of 659 COVID-19, 1660 healthy and 4265 non-COVID (viral and bacterial pneumonia) samples. They generated a robust Convolutional Neural Network (CNN) model for multi-class classification (COVID vs. normal vs. bacterial pneumonia vs. viral pneumonia) and binary classification (COVID-19 vs. non-COVID). Their model could perform rapid disease detection in 137 milliseconds per image in a system with NVIDIA GTX 1060 GPU for the online screening. The authors in [87] proposed an architecture to implement deep learning algorithms on a research scanner with GPU



(a) Naïve parallel model.

(b) Multiphase parallel model.

(c) GPU parallel model

(d) Multiswarm parallel model

FIGURE 14. Parallel SIA models for GPU architectures [89].

software beamforming. Two deep neural network architectures were implemented and evaluated on the ultrasound research platform termed as USPlatform. The authors used a six layer CNN architecture that was trained on two models. The first model used a dataset of 786 ultrasound images of baby body parts. The second model was used for brachial plexus localization and trained and evaluated on 5640 ultrasound B-mode frames. The architecture was deployed on a Nvidia Titan cluster integrated with the ultrasound platform. The authors in [88] proposed a deep learning model called 3D-DenseUNet-569 to perform fast training for semantic liver and tumor segmentation. The architecture also reduces the pooling layer by adopting a standard convolution with strides, and this decreases the memory consumption. The proposed architecture uses Depthwise Separable Convolution compared to traditional convolution. Their experimental results showed that DS-Conv could significantly decrease memory requirements and computational cost and achieve high performance.

F. EI ARCHITECTURES FOR SWARM INTELLIGENCE

The authors in [89] proposed an overview for accelerating swarm intelligence algorithms (SIAs) on GPU architectures. The authors discussed several considerations for implementing efficient parallel architectures of SIAs on GPUs. In this work, the authors discussed four categories for parallel implementation of SIAs on GPU architectures: (1) Naive parallel model; (2) Multiphase parallel model; (3) All-GPU parallel model; and (4) Multiswarm parallel model. These parallel SIA models for GPU architectures are shown in Figure 14.

In the naïve parallel model, the SIAs offload the fitness evaluations onto the GPU for parallel execution. The implementation can be coarse grained (task parallel) or fine-grained (data parallel). In the multiphase parallel model, computation from different phases with explicit or implicit parallelism are offloaded onto GPUs. One important factor for the SIA implementation is that the communication between GPU and CPU is very slow. The All-GPU parallel model combines multiple kernels into a single one and executes a whole program on the GPU. Two strategies are commonly used to overcome the need for hardware synchronization by the GPU: (1) Coarse-grained strategy where all threads are organized into a single block (i.e., the swarm is mapped on a single thread, and each particle is mapped on a single thread); and (2) Fine-grained strategy where data dependency is removed, or software synchronization is utilized. The multiswarm parallel model is useful for implementing high-dimensional and large-scale SIA problems as it reduces the search time and improves the quality of the solutions provided. In this architecture, the swarm is divided into a few subswarms, and each of them evolves separately utilizing different threads.

The authors in [90] proposed a GPU-based SIA architecture for the exploration and mining of association rules in big data. Their approach used a bee SIA optimization method (termed as GSum-BSO) in the meta-rules discovery process which is implemented using GPU-based parallel programming. In this work, a parallel algorithm for GPU architecture is proposed. The neighborhood search is performed on CPU while the tasks for evaluating the potential solutions are implemented in parallel on the GPU architecture.

The authors in [91] proposed an algorithm termed as Particle Swarm Stepwise Algorithm (PaSS) for deployment on CPU-GPU clusters. Although PaSS can outperform some existing techniques, the issues for target optimization is a significant challenge. In this work, the authors aimed to shorten the computational time by proposing a parallel architecture on CPU and GPU clusters. The initial state is when multiple particles are selected. The particles update the values of their objective function by utilizing the information collected from other particles. Their experimental results showed that the approach achieved scalability on multiple threaded CPU and seven times speedup and performance improvements on GPU.

The authors in [92] proposed an improvement for the efficiency of PSO for parallel implementation on multicore processors with GPU acceleration. Their experimental results with high-dimensional and complex functions showed that significant performance improvements and speedups could be obtained. The authors in [93] implemented pseudorandom number generators (PRNGs) on CPUs and GPUs. They demonstrated the effect of PRNGs on a parallel architecture of the PSO on a GPU. The performance of PSO algorithms is affected by the quality of the PRNGs running on a GPU. Their experimental results showed that the proposed parallel implementation of SPSO could give up to 307 times performance improvements and speedup compared to a serial CPU implementation. The authors in [94] proposed a multi-swarm parameter estimation of biological systems architecture for GPU. Their proposed methodology is based on PSO and termed as MS2PSO. The MS2PSO enables significant speedup for the computation by utilizing ODE numerical integrators which are accelerated on GPUs. Their approach utilizes the master-slave distributed model to partition calculations required by MS2PSO on multi-core CPUS and GPUs. The master-slave approaches utilize parallel fitness evaluations. The master process performs the communication tasks among the slave processes.

Other works for SIA implementation on GPUs can be found in [95], [96]. The authors in [95] proposed a GPU-Accelerated PSO for the SHE in multilevel converters with unequal DC levels. In this work, the authors implemented the PSO in parallel on the GPU with CUDA. Their experimental results showed that the execution efficiency improved by hundreds of times faster compared to its CPU based counterpart. The authors in [96] proposed a novel parallel approach to run standard PSO on GPUs for application to the travelling salesman problem (TSP). Their approach showed that the correct choice of PSO settings applied to the TSP could give a solution within a reasonable time.

G. EI ARCHITECTURES FOR EVOLUTIONARY AND GENETIC ALGORITHMS

The authors in [97] presented an overview for performance speedup and acceleration of genetic algorithms (GAs) on GPU. In this work, the authors discussed coarse-grained and fine-grained parallel architectures, and proposed three categories for parallel implementation of GAs on GPU architectures: (1) Master-slave or panmictic model; (2) Island model; and (3) Cellular model. These parallel GA models for GPU architectures are shown in Figure 15. The models can be divided based on how the mating mechanisms for the GAs are performed. On the one hand, the panmictic model is a global model where individuals mate freely within the entire population. On the other hand, the cellular model is a local model where individuals mate only with its neighbors. The island model is a combination of the panmictic and cellular models. In this model, the population is separated into sub groups (termed as islands). The individuals on an island can mate freely while restrictions are placed for mating across islands. A migration operator is used to bring a part of a sub-population amongst islands for genetic diversity. In the implementation, each island could be allocated to a GPU and the migration process is specified amongst the GPUs.

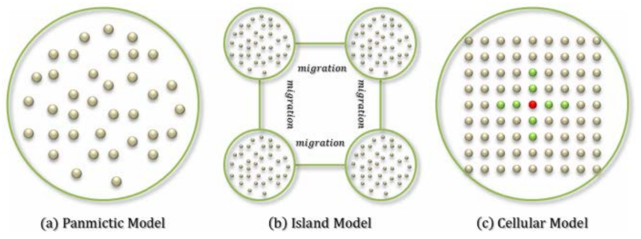


FIGURE 15. Parallel GA models for GPU architectures [97].

The authors in [98] proposed a GPU-based parallel implementation of a multi-objective evolutionary algorithm to train artificial neural networks (ANNs). They proposed a methodology for the prediction of energy consumption in buildings for increased energy efficiency. In their work, two machine learning techniques were combined. The first technique is ANN which were used to model and predict energy usage. The second technique is the multi-objective evolutionary algorithm to obtain the optimal ANN models for forecasting. Their experimental results compared a sequential implementation of the evolutionary algorithm (NSGA-II) with their architectures developed in parallel and implemented on GPU. Their approach showed better and faster execution time.

The authors in [99] proposed an approach termed as ERulesD2S for performing the induction of classification rules which are evolved with genetic programming. In this work, the authors proposed a highly efficient GPU-parallel architecture which could give performance improvements on data streams. The authors showed that ERulesD2S could scale to accommodate data streams with high dimensions while giving fast update and classifications. Their experimental results compared ERulesD2S with state-of-the-art classifiers and showed that their approach gave better performance than other approaches. The authors in [100] developed an optimization tool for wind turbine TMD using a combination of RBF neural networks and genetic algorithms. The authors developed a tool termed as PyGAOWT. Their experimental results showed that their proposed architecture gave good performance for the TMD application.

VII. LESSONS LEARNED, FUTURE PROSPECTS AND CHALLENGES

The previous sections have discussed EI architectures, the techniques for increasing their performances, the various criterions to target, and the challenges and potential solutions to address these main criterions. In this section, we discuss the lessons learned, future prospects, challenges and research directions for EI architectures. We identify a critical challenge and research direction towards addressing security, privacy and trust challenges for EI and discuss future prospects and applications for EI such as Industry 4.0.

A. LESSONS LEARNED FOR EI

The discussions in Sections III, IV and V have enabled us to derive several useful observations and lessons relevant to EI. In particular, Tables 1, 2, 3 and 4 have shown a snapshot of the broad spectrum of representative studies and the techniques (system-level, algorithm-level, architecture-level, technology level) which are being utilized to achieve the various criterions (fast training, fast decision-making, low power, small area and scalability) to meet the different EI applications and requirements. For example, the requirements for EI architectures (or accelerators) for servers/data centers will be different for EI architectures for edge devices.

We give the following observations. First, many authors have proposed techniques to exploit the fact that EI algorithms and applications can tolerate imprecision in the computations. This tolerance for imprecision can be exploited at different levels. At the algorithm level, approximate computing approaches can be utilized such as binarized operations/neural networks, and processing units (e.g., approximate multipliers/MAC) using approximate computing. At the architecture level, the designer has different tools for hardware-software co-design, design of custom instruction sets and datapaths for EI modules, and a variety of different hardware platforms such as ASIC, FPGA and GPU to select from. This tolerance for imprecision in EI can also be exploited at the technology level. The authors in [27] demonstrated an approach using Conductive Bridging RAM (CBRAM) devices and stochastic computing (i.e., computation is usually non-deterministic) to optimize deep learning parameters and reduced the size of Multiply and Accumulate (MAC) units by five orders of magnitude without incurring a notable performance decrease for classification task. A significant challenge which remains to be resolved would be the optimal tradeoff between the architecture complexity and the classification performance of the EI which can be tolerated.

Second, an analysis of the literature has shown that many EI architectures have been focused on the CNN architecture. There are far fewer architectures which have been proposed for other types of EI. The continuing challenge for the research community is to develop a portfolio of EI schemes and architectures to meet different application requirements and scenarios which may evolve in the future. Some examples of these EI architectures would be for recommendation systems, adversarial machine learning, intelligent swarms, etc. These EI architectures will also need to be designed to meet different criterions such as the ones proposed in Section II. Third, some authors have proposed to exploit new technologies and move away from the conventional Von Neumann and digital computing approaches to realize efficient EI architectures. Examples of these approaches would be the CIM computing structures which have the advantage of performing computations inside the memory structure to reduce data movements and increase energy efficiency. The authors in [111] discuss the potential of analog computing for deep learning hardware. The authors remarked that analog computing for deep learning is not expected to drive a fundamentally new ecosystem but will augment the existing digital platforms.

Fourth, the full realization and potential for EI requires a concerted effort by the research community to address the unique technical and interoperability challenges for EI. A step towards this direction is the development of the IEEE P2805 Standards [110]. Although these standards were developed for industrial and shop floor environments, the standards define protocols for self-management, data acquisition, and machine learning through cloud-edge collaboration on edge computing nodes (ECNs), and give guidelines for the applying machine learning algorithms for low-powered embedded devices. Figure 16 shows a schematic of the IEEE P2805 Standard for ECNs; and (5) Fifth, there are significant challenges which remain to be addressed for security, privacy and trust. New technologies such as blockchains may offer useful solutions. This will be discussed in the next sub-section for future prospects and research directions for EI.

B. FUTURE PROSPECTS AND RESEARCH DIRECTIONS FOR EI

The expectation is that EI would continue to play an important role in emerging technologies for social advancements and applications such as Industry 4.0 [112]. The integration of EI into smarter systems for manufacturing and production will enable improvements in areas such as predictive

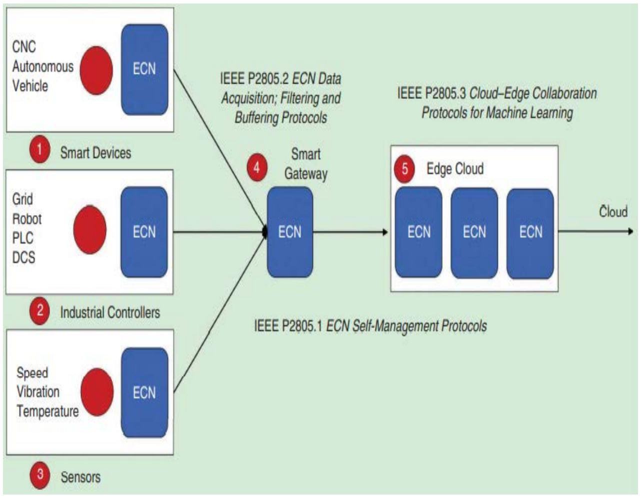

CNC
Autonomous
Vehicle
ECN
1 Smart Devices
Grid
Robot
PLC
DCS
ECN
2 Industrial Controllers
Speed
Vibration
Temperature
ECN
3 Sensors
IEEE P2805.2 ECN Data Acquisition; Filtering and Buffering Protocols
4 Smart Gateway
ECN
IEEE P2805.3 Cloud–Edge Collaboration Protocols for Machine Learning
5 Edge Cloud
ECN
ECN
ECN
Cloud
IEEE P2805.1 ECN Self-Management Protocols

FIGURE 16. Schematic of IEEE P2805 Standard for ECNs [110].

maintenance of machinery, early detection of product failures and optimized production lines. This leads on to improvements in manufacturing for better utilization of resources such as materials, machinery, labor and energy. Another trend that we see will impact practical applications and deployment of EI for society is that several other diverse technological trends are also emerging at the same time such as IoT, immersive visualization technologies (virtual and augmented reality (VR/AR)), digital twins and blockchains. While IoT technologies and smart sensing systems have been well investigated for integration with EI, the other trends for VR/AR, blockchains and digital twins have not been as well investigated. Some discussions for integrating EI and machine learning with VR/AR can be found in [113] and for integrating EI with digital twin technologies can be found in [114], [115]. A critical challenge which has been identified is the need for EI systems which addresses security, privacy and trust issues. One approach is to modify existing machine learning algorithms to be able to handle these issues. An example is the development of federated machine learning [116], [117] techniques to strengthen data privacy and security. A second approach is to utilize new blockchain technologies to achieve these goals. The authors in [118], [119] gives some discussions and challenges on the potential of integrating blockchains for edge computing systems.

Finally, we give remarks on future directions for EI research: (1) There remains significant challenges to continuously improve EI systems in terms of the five criterions which have been discussed. This could be in terms of developing new or improved system-level, algorithm-level, architecture-level or technology-level techniques; (2) Compared with applications for Industry 4.0 environments which have conditions which can be controlled, we see EI progressing and moving towards being more and more deployable in spontaneous scenarios and even unknown environments (EI in the wild); (3) A critical challenge is towards developing security, privacy and trust techniques for EI systems. These techniques can be incorporated into EI systems on a level-by-level basis and/or cross-layer approaches can be developed; and (4) The areas for integrating EI with other emerging trends such as blockchains, VR/AR and digital twins remain as significant challenges to be addressed for the research community and serve as useful future research directions.

VIII. CONCLUSION

This article addressed the convergence of increasingly complex embedded intelligence (EI) techniques and deployments into hardware architectures for server accelerators and edge devices. Our discussion began with a set of use cases and proposing some criterions for EI deployments and architectures. Using the criterions, we then presented discussions for EI architectures and accelerators for ASIC, FPGA and GPU platforms. Next, we discussed application specific EI architectures for various applications and scenarios. The paper gives a classification and discusses different techniques for EI implementations from different aspects or levels. We also discussed the lessons learned and the future prospects and broader perspectives for EI. This paper has presented the various issues and challenges for practical EI architectures and deployments and aims to open new avenues for future research directions in this area.

REFERENCES

[1] Y.-L. Lee, P.-K. Tsung, and M. Wu, "Techology trend of edge AI," in *Proc. Int. Symp. VLSI Design, Autom. Test (VLSI-DAT)*, Apr. 2018, pp. 1–2.

[2] S. Greengard, "AI on edge," *Commun. ACM*, vol. 63, no. 9, pp. 18–20, Aug. 2020.

[3] H. Flores, P. Nurmi, and P. Hui, "AI on the move: From on-device to on-multi-device," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun. Workshops (PerCom Workshops)*, Mar. 2019, pp. 310–315.

[4] H.-J. Yoo, "Intelligence on silicon: From deep-neural-network accelerators to brain mimicking AI-SoCs," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2019, pp. 20–26.

[5] T. Baji, "Evolution of the GPU device widely used in AI and massive parallel processing," in *Proc. IEEE 2nd Electron Devices Technol. Manuf. Conf. (EDTM)*, Mar. 2018, pp. 7–9.

[6] T. Baji, "GPU: The biggest key processor for AI and parallel processing," *Proc. SPIE*, vol. 10454, Jul. 2017, Art. no. 1045406.

[7] Z. Li, Y. Zhang, J. Wang, and J. Lai, "A survey of FPGA design for AI era," *J. Semicond.*, vol. 41, no. 2, Feb. 2020, Art. no. 021402.

[8] K. P. Seng, P. J. Lee, and L. M. Ang, "Embedded intelligence on FPGA: Survey, applications and challenges," *Electronics*, vol. 10, no. 8, p. 895, 2021.

[9] J. Nickolls and W. J. Dally, "The GPU computing era," *IEEE Micro*, vol. 30, no. 2, pp. 56–69, Mar./Apr. 2010, doi: 10.1109/MM.2010.41.

[10] Q. Shang, L. Chen, J. Cui, and Y. Lu, "Hardware evolution based on improved simulated annealing algorithm in cyclone V FPSoCs," *IEEE Access*, vol. 8, pp. 64770–64782, 2020.

[11] D. Ignatov and A. Ignatov, "Controlling information capacity of binary neural network," *Pattern Recognit. Lett.*, vol. 138, pp. 276–281, Oct. 2020.

[12] H. Qin, R. Gong, X. Liu, X. Bai, J. Song, and N. Sebe, "Binary neural networks: A survey," *Pattern Recognit.*, vol. 105, Sep. 2020, Art. no. 107281.

[13] D. Moolchandani, A. Kumar, and S. R. Sarangi, "Accelerating CNN inference on ASICs: A survey," *J. Syst. Archit.*, vol. 113, Feb. 2021, Art. no. 101887.

[14] A. Khan, A. Sohail, U. Zahoora, and A. S. Qureshi, "A survey of the recent architectures of deep convolutional neural networks," *Artif. Intell. Rev.*, vol. 53, no. 8, pp. 5455–5516, Dec. 2020.

[15] S. I. Venieris, A. Kouris, and C.-S. Bouganis, "Toolflows for mapping convolutional neural networks on FPGAs: A survey and future directions," *ACM Comput. Surv.*, vol. 51, no. 3, pp. 1–39, May 2019.

[16] M. Z. Alom, T. M. Taha, C. Yakopcic, S. Westberg, P. Sidike, M. S. Nasrin, M. Hasan, B. C. Van Essen, A. A. S. Awwal, and V. K. Asari, "A sate-of-the-art survey on deep learning theory and architectures," *Electronics*, vol. 8, no. 3, p. 292, Mar. 2019.

[17] K. L.-M. Ang and J. K. P. Seng, "Embedded intelligence: Platform technologies, device analytics, and smart city applications," *IEEE Internet Things J.*, vol. 8, no. 17, pp. 13165–13182, Sep. 2021.

[18] R. P. Dick, L. Shang, M. Wolf, and S. W. Yang, "Embedded intelligence in the Internet-of-Things," *IEEE Design Test*, vol. 37, no. 1, pp. 7–27, Feb. 2020.

[19] J. Lin, W.-M. Chen, Y. Lin, J. Cohn, C. Gan, and S. Han, "MCUNet: Tiny deep learning on IoT devices," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 11711–11722.

[20] A. Fonseca and B. Cabral, "Prototyping a GPGPU neural network for deep-learning big data analysis," *Big Data Res.*, vol. 8, pp. 50–56, Jul. 2017.

[21] S. Branco, A. G. Ferreira, and J. Cabral, "Machine learning in resource-scarce embedded systems, FPGAs, and end-devices: A survey," *Electronics*, vol. 8, no. 11, p. 1289, 2019.

[22] Y. Chen, T. Chen, Z. Xu, N. Sun, and O. Temam, "Diannao family: Energy-efficient hardware accelerators for machine learning," *Commun. ACM*, vol. 59, no. 11, pp. 105–112, 2016.

[23] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, "Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," *SIGARCH Comput. Archit. News*, vol. 42, no. 1, pp. 269–284, Feb. 2014.

[24] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, and O. Temam, "DaDianNao: A machine-learning supercomputer," in *Proc. 47th Annu. IEEE/ACM Int. Symp. Microarchitecture*, Dec. 2014, pp. 609–622.

[25] D. Liu, T. Chen, S. Liu, J. Zhou, S. Zhou, O. Teman, X. Feng, X. Zhou, and Y. Chen, "Pudiannao: A polyvalent machine learning accelerator," *ACM SIGARCH Comput. Archit. News*, vol. 43, no. 1, pp. 369–381, Mar. 2015.

[26] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam, "ShiDianNao: Shifting vision processing closer to the sensor," in *Proc. 42nd Annu. Int. Symp. Comput. Archit.*, Jun. 2015, pp. 92–104.

[27] C. Lammie, J. K. Eshraghian, W. D. Lu, and M. R. Azghadi, "Memristive stochastic computing for deep learning parameter optimization," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 68, no. 5, pp. 1650–1654, May 2021.

[28] Z. Chen, Z. Yu, Q. Jin, Y. He, J. Wang, S. Lin, D. Li, Y. Wang, and K. Yang, "CAP-RAM: A charge-domain in-memory computing 6T-SRAM for accurate and precision-programmable CNN inference," *IEEE J. Solid-State Circuits*, vol. 56, no. 6, pp. 1924–1935, Jun. 2021.

[29] R. Xu, L. Tao, T. Wang, X. Jin, C. Li, Z. Li, and J. Ren, "A hybrid precision low power computing-in-memory architecture for neural networks," *Microprocessors Microsyst.*, vol. 80, Feb. 2021, Art. no. 103351.

[30] C. J. Schaefer, M. Horeni, P. Taheri, and S. Joshi, "LSTMs for keyword spotting with ReRAM-based compute-in-memory architectures," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2021, pp. 1–5.

[31] R. Guo, Y. Liu, S. Zheng, S.-Y. Wu, P. Ouyang, W.-S. Khwa, X. Chen, J.-J. Chen, X. Li, L. Liu, M.-F. Chang, S. Wei, and S. Yin, "A 5.1 pJ/neuron 127.3 μs/inference RNN-based speech recognition processor using 16 computing-in-memory SRAM macros in 65 nm CMOS," in *Proc. Symp. VLSI Circuits*, Jun. 2019, pp. C120–C121.

[32] F. Chen, L. Song, and Y. Chen, "ReGAN: A pipelined ReRAM-based accelerator for generative adversarial networks," in *Proc. 23rd Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Jan. 2018, pp. 178–183.

[33] I. Hammad, L. Li, K. El-Sankary, and W. M. Snelgrove, "CNN inference using a preprocessing precision controller and approximate multipliers with various precisions," *IEEE Access*, vol. 9, pp. 7220–7232, 2021.

[34] M. S. Ansari, V. Mrazek, B. F. Cockburn, L. Sekanina, Z. Vasicek, and J. Han, "Improving the accuracy and hardware efficiency of neural networks using approximate multipliers," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 28, no. 2, pp. 317–328, Feb. 2020.

[35] *Enhanced Visual Intelligence at the Network Edge*. Accessed: Feb. 2022. [Online]. Available: https://www.intel.com/content/www/us/en/products/details/processors/movidius-vpu.html

[36] *Build Beneficial and Privacy Preserving AI*. Accessed: Apr. 2022. [Online]. Available: https://coral.ai/

[37] Y. Gong, B. Liu, W. Ge, and L. Shi, "ARA: Cross-layer approximate computing framework based reconfigurable architecture for CNNs," *Microelectron. J.*, vol. 87, pp. 33–44, May 2019.

[38] A. Erdem, C. Silvano, T. Boesch, A. Ornstein, S.-P. Singh, and G. Desoli, "Design space exploration for orlando ultra low-power convolutional neural network SoC," in *Proc. IEEE 29th Int. Conf. Appl.-Specific Syst., Architectures Processors (ASAP)*, Jul. 2018, pp. 1–7.

[39] C.-H. Lin, C.-C. Cheng, Y.-M. Tsai, S.-J. Hung, Y.-T. Kuo, P. H. Wang, P.-K. Tsung, J.-Y. Hsu, W.-C. Lai, C.-H. Liu, S.-Y. Wang, C.-H. Kuo, C.-Y. Chang, M.-H. Lee, T.-Y. Lin, and C.-C. Chen, "A 3.4-to-13.3 TOPS/W 3.6 TOPS dual-core deep-learning accelerator for versatile AI applications in 7nm 5G smartphone SoC," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2020, pp. 134–136.

[40] A. Safaei, Q. M. J. Wu, T. Akilan, and Y. Yang, "System-on-a-chip (SoC)-based hardware acceleration for an online sequential extreme learning machine (OS-ELM)," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 38, no. 11, pp. 2127–2138, Nov. 2019.

[41] A. Safaei, Q. M. J. Wu, Y. Yang, and T. Akilan, "System-on-a-chip (SoC)-based hardware acceleration for extreme learning machine," in *Proc. 24th IEEE Int. Conf. Electron., Circuits Syst. (ICECS)*, Dec. 2017, pp. 470–473.

[42] M. Kee, S.-J. Lee, H.-S. Seon, J. Lee, and G.-H. Park, "Intelligence boosting engine (IBE): A hardware accelerator for processing sensor fusion and machine learning algorithm for a sensor hub SoC," in *Proc. IEEE Symp. Low-Power High-Speed Chips (COOL CHIPS)*, Apr. 2017, pp. 1–3.

[43] U. Martinez-Corral and K. Basterretxea, "A fully configurable and scalable neural coprocessor IP for SoC implementations of machine learning applications," in *Proc. NASA/ESA Conf. Adapt. Hardw. Syst. (AHS)*, Jul. 2017, pp. 125–132.

[44] D. Shin, Y. Kim, and H.-J. Yoo, "A 1.41 mW on-chip/off-chip hybrid transposition table for low-power robust deep tree search in artificial intelligence SoCs," in *Proc. 30th IEEE Int. Syst.-on-Chip Conf. (SOCC)*, Sep. 2017, pp. 138–142.

[45] G. Desoli, N. Chawla, T. Boesch, S.-P. Singh, E. Guidetti, F. De Ambroggi, T. Majo, P. Zambotti, M. Ayodhyawasi, H. Singh, and N. Aggarwal, "A 2.9 TOPS/W deep convolutional neural network SoC in FD-SOI 28 nm for intelligent embedded systems," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2017, pp. 238–239.

[46] T. Moreau, M. Wyse, J. Nelson, A. Sampson, H. Esmaeilzadeh, L. Ceze, and M. Oskin, "SNNAP: Approximate computing on programmable SoCs via neural acceleration," in *Proc. IEEE 21st Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2015, pp. 603–614.

[47] Y. Wang, H. Li, L. Cheng, and X. Li, "A QoS-QoR aware CNN accelerator design approach," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 38, no. 11, pp. 1995–2007, Nov. 2019.

[48] C. Wang, L. Gong, F. Jia, and X. Zhou, "An FPGA based accelerator for clustering algorithms with custom instructions," *IEEE Trans. Comput.*, vol. 70, no. 5, pp. 725–732, May 2021.

[49] C. Wang, L. Gong, X. Ma, X. Li, and X. Zhou, "WooKong: A ubiquitous accelerator for recommendation algorithms with custom instruction sets on FPGA," *IEEE Trans. Comput.*, vol. 69, no. 7, pp. 1071–1082, Jul. 2020.

[50] E. Setiawan and T. Adiono, "Implementation of systolic co-processor for deep neural network inference based on SoC," in *Proc. Int. SoC Design Conf. (ISOCC)*, Nov. 2018, pp. 36–37.

[51] Q. Yin, Y. Li, H. Huang, H. Li, Q. Zhang, B. Cao, and J. Zhang, "FPGA-based high-performance CNN accelerator architecture with high DSP utilization and efficient scheduling mode," in *Proc. Int. Conf. High Perform. Big Data Intell. Syst. (HPBD&IS)*, May 2020, pp. 1–7.

[52] W.-J. Li, S.-J. Ruan, and D.-S. Yang, "Implementation of energy-efficient fast convolution algorithm for deep convolutional neural networks based on FPGA," *Electron. Lett.*, vol. 56, no. 10, pp. 485–488, May 2020.

[53] Z. Xu, J. Yu, C. Yu, H. Shen, Y. Wang, and H. Yang, "CNN-based feature-point extraction for real-time visual SLAM on embedded FPGA," in *Proc. IEEE 28th Annu. Int. Symp. Field-Program. Custom Comput. Mach. (FCCM)*, May 2020, pp. 33–37.

[54] T. Posewsky and D. Ziener, "Throughput optimizations for FPGA-based deep neural network inference," *Microprocessors Microsyst.*, vol. 60, pp. 151–161, Jul. 2018.

[55] P. G. Mousouliotis and L. P. Petrou, "CNN-grinder: From algorithmic to high-level synthesis descriptions of CNNs for low-end-low-cost FPGA SoCs," *Microprocessors Microsyst.*, vol. 73, Mar. 2020, Art. no. 102990.

[56] S. Mittal and J. S. Vetter, "A survey of methods for analyzing and improving GPU energy efficiency," *ACM Comput. Surv.*, vol. 47, no. 2, pp. 1–23, Jan. 2015.
[57] K. Raju and N. N. Chiplunkar, "A survey on techniques for cooperative CPU-GPU computing," *Sustain. Comput., Informat. Syst.*, vol. 19, pp. 72–85, Sep. 2018.
[58] L. M. Ang and K. P. Seng, "GPU-based embedded intelligence architectures and applications," *Electronics*, vol. 10, no. 8, p. 952, Apr. 2021.
[59] Y. N. Khalid, M. Aleem, U. Ahmed, M. A. Islam, and M. A. Iqbal, "Troodon: A machine-learning based load-balancing application scheduler for CPU–GPU system," *J. Parallel Distrib. Comput.*, vol. 132, pp. 79–94, Oct. 2019.
[60] Y. Inoue, "Queueing analysis of GPU-based inference servers with dynamic batching: A closed-form characterization," *Perform. Eval.*, vol. 147, May 2021, Art. no. 102183.
[61] N. A. Gawande, J. A. Daily, C. Siegel, N. R. Tallent, and A. Vishnu, "Scaling deep learning workloads: NVIDIA DGX-1/Pascal and Intel knights landing," *Future Gener. Comput. Syst.*, vol. 108, pp. 1162–1172, Jul. 2020.
[62] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Univ. Toronto, Toronto, ON, Canada, Tech. Rep., 2009.
[63] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.
[64] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 1–9.
[65] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.
[66] S. Dong, P. Zhao, X. Lin, and D. Kaeli, "Exploring GPU acceleration of deep neural networks using block circulant matrices," *Parallel Comput.*, vol. 100, Dec. 2020, Art. no. 102701.
[67] E. Yang, S.-H. Kim, T.-W. Kim, M. Jeon, S. Park, and C.-H. Youn, "An adaptive batch-orchestration algorithm for the heterogeneous GPU cluster environment in distributed deep learning system," in *Proc. IEEE Int. Conf. Big Data Smart Comput. (BigComp)*, Jan. 2018, pp. 725–728.
[68] S. Li, X. Niu, Y. Dou, Q. Lv, and Y. Wang, "Heterogeneous blocked CPU-GPU accelerate scheme for large scale extreme learning machine," *Neurocomputing*, vol. 261, pp. 153–163, Oct. 2017.
[69] P. Carvalho, E. Clua, A. Paes, C. Bentes, B. Lopes, and L. M. D. A. Drummond, "Using machine learning techniques to analyze the performance of concurrent kernel execution on GPUs," *Future Gener. Comput. Syst.*, vol. 113, pp. 528–540, Dec. 2020.
[70] J. Fang, H. Fu, G. Yang, and C.-J. Hsieh, "RedSync: Reducing synchronization bandwidth for distributed deep learning training system," *J. Parallel Distrib. Comput.*, vol. 133, pp. 30–39, Nov. 2019.
[71] I. Hong, S. Park, J. Park, and H.-J. Yoo, "A 1.9 nJ/pixel embedded deep neural network processor for high speed visual attention in a mobile vision recognition SoC," in *Proc. IEEE Asian Solid-State Circuits Conf. (A-SSCC)*, Nov. 2015, pp. 1–4.
[72] S. Lee, M. Kim, K. Kim, J.-Y. Kim, and H.-J. Yoo, "24-GOPS 4.5-mm^2 digital cellular neural network for rapid visual attention in an object-recognition SoC," *IEEE Trans. Neural Netw.*, vol. 22, no. 1, pp. 64–73, Jan. 2011.
[73] Y. Wei, Q. Cao, J. Gu, K. Otseidu, and L. Hargrove, "A fully-integrated gesture and gait processing SoC for rehabilitation with ADC-less mixed-signal feature extraction and deep neural network for classification and online training," in *Proc. IEEE Custom Integr. Circuits Conf. (CICC)*, Mar. 2020, pp. 1–4.
[74] A. R. Aslam, T. Iqbal, M. Aftab, W. Saadeh, and M. A. B. Altaf, "A 10.13 μJ/classification 2-channel deep neural network-based SoC for emotion detection of autistic children," in *Proc. IEEE Custom Integr. Circuits Conf. (CICC)*, Mar. 2020, pp. 1–4.
[75] S. Wang, A. Pathania, and T. Mitra, "Neural network inference on mobile SoCs," *IEEE Design Test*, vol. 37, no. 5, pp. 50–57, Oct. 2020.
[76] J. Song, Y. Cho, J.-S. Park, J.-W. Jang, S. Lee, J.-H. Song, J.-G. Lee, and I. Kang, "An 11.5 TOPS/W 1024-MAC butterfly structure dual-core sparsity-aware neural processing unit in 8 nm flagship mobile SoC," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2019, pp. 130–132.
[77] T.-W. Chen, C.-S. Tang, S.-F. Tsai, C.-H. Tsai, S.-Y. Chien, and L.-G. Chen, "Tera-scale performance machine learning SoC (MLSoC) with dual stream processor architecture for multimedia content analysis," *IEEE J. Solid-State Circuits*, vol. 45, no. 11, pp. 2321–2329, Nov. 2010.
[78] T.-W. Chen, Y.-L. Chen, T.-Y. Cheng, C.-S. Tang, P.-K. Tsung, T.-D. Chuang, L.-G. Chen, and S.-Y. Chien, "A multimedia semantic analysis SoC (SASoC) with machine-learning engine," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2010, pp. 338–339.
[79] M. Kee, L. Seung-Jin, S. Hyun-Su, L. Jongsung, and G.-H. Park, "Intelligence boosting engine (IBE): A hardware accelerator for processing sensor fusion and machine learning algorithm for a sensor hub SoC," in *Proc. IEEE Symp. Low-Power High-Speed Chips (COOL CHIPS)*, Apr. 2017, pp. 1–3.
[80] J.-W. Chang, K.-W. Kang, and S.-J. Kang, "An energy-efficient FPGA-based deconvolutional neural networks accelerator for single image super-resolution," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 30, no. 1, pp. 281–295, Jan. 2020.
[81] M. A. B. Altaf, C. Zhang, and J. Yoo, "A 16-ch patient-specific seizure onset and termination detection SoC with machine-learning and voltage-mode transcranial stimulation," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2015, pp. 1–3.
[82] M. A. Bin Altaf, C. Zhang, and J. Yoo, "A 16-channel patient-specific seizure onset and termination detection SoC with impedance-adaptive transcranial electrical stimulator," *IEEE J. Solid-State Circuits*, vol. 50, no. 11, pp. 2728–2740, Nov. 2015.
[83] B. Bozorgzadeh, D. R. Schuweiler, M. J. Bobak, P. A. Garris, and P. Mohseni, "Neurochemostat: A neural interface SoC with integrated chemometrics for closed-loop regulation of brain dopamine," *IEEE Trans. Biomed. Circuits Syst.*, vol. 10, no. 3, pp. 654–667, Jun. 2016.
[84] S.-Y. Hsu, Y. Ho, P.-Y. Chang, C. Su, and C.-Y. Lee, "A 48.6-to-105.2 μW machine learning assisted cardiac sensor SoC for mobile healthcare applications," *IEEE J. Solid-State Circuits*, vol. 49, no. 4, pp. 801–811, Apr. 2014.
[85] S.-Y. Hsu, Y. Ho, P.-Y. Chang, P.-Y. Hsu, C.-Y. Yu, Y. Tseng, T.-Z. Yang, T.-F. Yang, R.-J. Chen, C. Su, and C.-Y. Lee, "A 48.6-to-105.2 μW machine-learning assisted cardiac sensor SoC for mobile healthcare monitoring," in *Proc. Symp. VLSI Circuits*, Jun. 2013, pp. C252–C253.
[86] R. C. Joshi, S. Yadav, V. K. Pathak, H. S. Malhotra, H. V. S. Khokhar, A. Parihar, N. Kohli, D. Himanshu, R. K. Garg, M. L. B. Bhatt, R. Kumar, N. P. Singh, V. Sardana, R. Burget, C. Alippi, C. M. Travieso-Gonzalez, and M. K. Dutta, "A deep learning-based COVID-19 automatic diagnostic framework using chest X-ray images," *Biocybern. Biomed. Eng.*, vol. 41, no. 1, pp. 239–254, 2021.
[87] P. Jarosik and M. Lewandowski, "The feasibility of deep learning algorithms integration on a GPU-based ultrasound research scanner," in *Proc. IEEE Int. Ultrason. Symp. (IUS)*, Sep. 2017, pp. 1–4.
[88] N. Alalwan, A. Abozeid, A. A. ElHabshy, and A. Alzahrani, "Efficient 3D deep learning model for medical image semantic segmentation," *Alexandria Eng. J.*, vol. 60, no. 1, pp. 1231–1239, Feb. 2021.
[89] Y. Tan and K. Ding, "A survey on GPU-based implementation of swarm intelligence algorithms," *IEEE Trans. Cybern.*, vol. 46, no. 9, pp. 2028–2041, Sep. 2016.
[90] Y. Djenouri, A. Bendjoudi, D. Djenouri, A. Belhadi, and N. Nouali-Taboudjemat, "New GPU-based swarm intelligence approach for reducing big association rules space," in *Proc. IEEE SmartWorld, Ubiquitous Intell. Comput., Adv. Trusted Comput., Scalable Comput. Commun., Cloud Big Data Comput., Internet People Smart City Innov. (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*, Aug. 2017, pp. 1–6.
[91] M. Yang, R.-B. Chen, I.-H. Chung, and W. Wang, "Particle swarm stepwise algorithm (PaSS) on multicore hybrid CPU-GPU clusters," in *Proc. IEEE Int. Conf. Comput. Inf. Technol. (CIT)*, Dec. 2016, pp. 265–272.
[92] M. P. Wachowiak, M. C. Timson, and D. J. DuVal, "Adaptive particle swarm optimization with heterogeneous multicore parallelism and GPU acceleration," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 10, pp. 2784–2793, Oct. 2017.
[93] M. M. Hussain and N. Fujimoto, "Effect of the pseudorandom number generators on the standard particle swarm optimization on a GPU," in *Proc. Int. Conf. Comput. Sci. Comput. Intell. (CSCI)*, Dec. 2018, pp. 295–300.

[94] A. Tangherloni, L. Rundo, S. Spolaor, P. Cazzaniga, and M. S. Nobile, "GPU-powered multi-swarm parameter estimation of biological systems: A master-slave approach," in *Proc. 26th Euromicro Int. Conf. Parallel Distrib. Netw.-Based Process. (PDP)*, Mar. 2018, pp. 698–705.

[95] K. Yang, H. Li, Y. Huang, Q. Zhang, and G. Zhao, "GPU-accelerated particle swarm optimization for selective harmonic elimination in multilevel converters with unequal DC levels," in *Proc. 43rd Annu. Conf. IEEE Ind. Electron. Soc. (IECON)*, Oct. 2017, pp. 1186–1191.

[96] O. Bali, W. Elloumi, P. Kromer, and A. M. Alimi, "GPU particle swarm optimization applied to travelling salesman problem," in *Proc. IEEE 9th Int. Symp. Embedded Multicore/Many-Core Syst.-on-Chip*, Sep. 2015, pp. 112–119.

[97] J. R. Cheng and M. Gen, "Accelerating genetic algorithms with GPU computing: A selective overview," *Comput. Ind. Eng.*, vol. 128, pp. 514–525, Feb. 2019.

[98] J. R. S. Iruela, L. G. B. Ruiz, M. C. Pegalajar, and M. I. Capel, "A parallel solution with GPU technology to predict energy consumption in spatially distributed buildings using evolutionary optimization and artificial neural networks," *Energy Convers. Manage.*, vol. 207, Mar. 2020, Art. no. 112535.

[99] A. Cano and B. Krawczyk, "Evolving rule-based classifiers with genetic programming on GPUs for drifting data streams," *Pattern Recognit.*, vol. 87, pp. 248–268, Mar. 2019.

[100] Z. Liu, Y. Wang, X. Hua, H. Zhu, and Z. Zhu, "Optimization of wind turbine TMD under real wind distribution countering wake effects using GPU acceleration and machine learning technologies," *J. Wind Eng. Ind. Aerodyn.*, vol. 208, Jan. 2021, Art. no. 104436.

[101] F. Sakr, F. Bellotti, R. Berta, and A. De Gloria, "Machine learning on mainstream microcontrollers," *Sensors*, vol. 20, no. 9, p. 2638, May 2020.

[102] R. V. Shah, G. Grennan, M. Zafar-Khan, F. Alim, S. Dey, D. Ramanathan, and J. Mishra, "Personalized machine learning of depressed mood using wearables," *Transl. Psychiatry*, vol. 11, no. 1, pp. 1–18, Jun. 2021.

[103] C. Voss, N. Haber, and D. P. Wall, "The potential for machine learning–based wearables to improve socialization in teenagers and adults with autism spectrum disorder—Reply," *J. Amer. Med. Assoc. Pediatrics*, vol. 173, no. 11, p. 1106, Nov. 2019.

[104] P. Kumari, L. Mathew, and P. Syal, "Increasing trend of wearables and multimodal interface for human activity monitoring: A review," *Biosensors Bioelectron.*, vol. 90, pp. 298–307, Apr. 2017.

[105] B. M. Meyer, L. J. Tulipani, R. D. Gurchiek, D. A. Allen, L. Adamowicz, D. Larie, A. J. Solomon, N. Cheney, and R. S. McGinnis, "Wearables and deep learning classify fall risk from gait in multiple sclerosis," *IEEE J. Biomed. Health Informat.*, vol. 25, no. 5, pp. 1824–1831, May 2021.

[106] M. Merenda, C. Porcaro, and D. Iero, "Edge machine learning for AI-enabled IoT devices: A review," *Sensors*, vol. 20, no. 9, p. 2533, 2020.

[107] T. Lin, "Deep learning for IoT," in *Proc. IEEE 39th Int. Perform. Comput. Commun. Conf. (IPCCC)*, Nov. 2020, pp. 1–4.

[108] S. R. Upadhyaya, "Parallel approaches to machine learning—A comprehensive survey," *J. Parallel Distrib. Comput.*, vol. 73, no. 3, pp. 284–292, Mar. 2013.

[109] J. Qiu, Q. Wu, G. Ding, Y. Xu, and S. Feng, "A survey of machine learning for big data processing," *EURASIP J. Adv. Signal Process.*, vol. 2016, no. 1, pp. 1–16, 2016.

[110] W. Dai, H. Nishi, V. Vyatkin, V. Huang, Y. Shi, and X. Guan, "Industrial edge computing: Enabling embedded intelligence," *IEEE Ind. Electron. Mag.*, vol. 13, no. 4, pp. 48–56, Dec. 2019.

[111] W. Haensch, T. Gokmen, and R. Puri, "The next generation of deep learning hardware: Analog computing," *Proc. IEEE*, vol. 107, no. 1, pp. 108–122, Jan. 2019.

[112] C. Lee and C. Lim, "From technological development to social advance: A review of industry 4.0 through machine learning," *Technol. Forecasting Social Change*, vol. 167, Jun. 2021, Art. no. 120653.

[113] E. Bastug, M. Bennis, M. Médard, and M. Debbah, "Toward interconnected virtual reality: Opportunities, challenges, and enablers," *IEEE Commun. Mag.*, vol. 55, no. 6, pp. 110–117, Jun. 2017.

[114] Y. Jiang, S. Yin, K. Li, H. Luo, and O. Kaynak, "Industrial applications of digital twins," *Phil. Trans. Roy. Soc. A, Math., Phys. Eng. Sci.*, vol. 379, no. 2207, 2021, Art. no. 20200360.

[115] F. Flammini, "Digital twins as run-time predictive models for the resilience of cyber-physical systems: A conceptual framework," *Phil. Trans. Roy. Soc. A, Math., Phys. Eng. Sci.*, vol. 379, no. 2207, Oct. 2021, Art. no. 20200369.

[116] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Trans. Intell. Syst. Technol.*, vol. 10, no. 2, pp. 1–19, 2019.

[117] X. Lu, Y. Liao, P. Lio, and P. Hui, "Privacy-preserving asynchronous federated learning mechanism for edge network computing," *IEEE Access*, vol. 8, pp. 48970–48981, 2020.

[118] R. Yang, F. R. Yu, P. Si, Z. Yang, and Y. Zhang, "Integrated blockchain and edge computing systems: A survey, some research issues and challenges," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 2, pp. 1508–1532, 2nd Quart., 2019.

[119] Z. Xiong, Y. Zhang, D. Niyato, P. Wang, and Z. Han, "When mobile blockchain meets edge computing," *IEEE Commun. Mag.*, vol. 56, no. 8, pp. 33–39, Aug. 2018.

KAH PHOOI SENG (Senior Member, IEEE) received the B.Eng. and Ph.D. degrees from the University of Tasmania, Australia. She was a Professor and the Department Head of computer science and networked system with Sunway University. Before joining Sunway University, she was an Associate Professor with the School of Electrical and Electronic Engineering, University of Nottingham. She has worked or attached to Australian-based and U.K.-based universities including Monash University, Griffith University, the University of Tasmania, the University of Nottingham, Sunway University, Edith Cowan University, and Charles Sturt University. She is currently a Professor in artificial intelligence at Xi'an Jiaotong-Liverpool University and an Adjunct Professor at the School of Computer Science, Queensland University of Technology. Prior to joining XJTLU, she was an Adjunct Professor with the School of Engineering and Information Technology, UNSW. She has a strong record of publications and has published over 250 papers in journals and international refereed conferences. She has participated in over 1.8 million dollars research grant projects from government and industry in Australia and overseas. She has supervised or co-supervised 15 Ph.D. students to completion and more than 25 higher degree research students. Her research interests include computer science and engineering including artificial intelligence (AI), data science and machine learning, big data, multimodal information processing, intelligent systems, the Internet of Things (IoT), embedded systems, mobile software development, affective computing, computer vision, and the development of innovative technologies for real-world applications. She is an Associate Editor of IEEE Access. She also serves on the editorial board or committees of several journals and international conferences.

LI-MINN ANG (Senior Member, IEEE) received the B.Eng. (Hons.) and Ph.D. degrees from Edith Cowan University, Australia. He was an Associate Professor of networked and computer systems with the School of Information and Communication Technology (ICT), Griffith University. He has worked at Australian and U.K. universities including Monash University, the University of Nottingham, ECU, CSU, and Griffith University. He is currently a Professor of electrical and computer engineering at the School of Science, Technology and Engineering, University of the Sunshine Coast (USC), Australia. His research interests include computer, electrical, and systems engineering, including the Internet of Things, intelligent systems and data analytics, machine learning, visual information processing, embedded systems, wireless multimedia sensor systems, reconfigurable computing (FPGA), and the development of innovative technologies for real-world systems including smart cities, engineering, agriculture, environment, health, and defence. He is a fellow of the Higher Education Academy, U.K.

• • •