EdgeCube – An IoT Edge-sensor Node for Tiny Machine Learning Applications

Nathan Timmins and Cristinel Ababei
Electrical and Computer Engineering, Marquette University
Email: {nathan.timmins,cristinel.ababei}@marquette.edu

***Abstract*—We present a novel internet-of-things (IoT) edge device, EdgeCube, a compact embedded system that combines a microcontroller unit (MCU) and a field programmable gate array (FPGA). The proposed edge device represents a new tradeoff between how much performance can be pushed by the use of the FPGA device as an accelerator and the additional cost and increase in power consumption. Because it is intended for IoT vision applications, the proposed edge device includes a camera, specifically ESP32-CAM. We demonstrate EdgeCube on the classic person detection application, where the edge device must identify whether the camera sees a person or not in its frames. Such functionality is desired in applications where the edge device only needs to output one bit of information, 1 for person detected and 0 for no person in frame. No identifiable data is gathered from the edge device for privacy reasons, and no heavy communication traffic is necessary, thereby saving energy usage. The frames captured by the camera are sent by the MCU over SPI communication to the FPGA device, where a light convolutional neural network (CNN) runs realtime inference on the received frames. The CNN model is implemented on the FPGA to improve inference time by benefiting from the hardware acceleration and parallelism offered by the FPGA. In our hardware prototype, we use the popular ESP32-CAM MCU and an AMD Xilinx Artix 7 as an FPGA device. Working with a frame size of 96x96 pixels retrieved from the camera's OV2640 sensor, the proposed EdgeCube provides a performance of 235 fps and a detection accuracy of 71%.**

***Index Terms*—edge AI; tiny ML; IoT; microcontroller; FPGA accelerator; CNN; HLS;**

I. Introduction

We are witnessing two major developments in the AI area. On one hand, fueled by developments in supporting hardware (such as increasingly powerful manycore processors, graphics processing units (GPUs), and cloud/datacenter computing), increasingly sophisticated machine learning (ML) models have been developed to enable generative AI tools. On the other hand, there has been increased interest and research efforts in deploying AI/ML models and algorithms directly to internet-of-things (IoT) devices. This trend is motivated by the desire to shift AI/ML computations from the cloud to edge devices, which would eliminate high communication traffic of data from the edge to the cloud (reducing delay and power consumption) while at the same time enhancing security and privacy (because identifiable information need not be transmitted to the cloud, and, communication channels would be safer against side channel attacks). This shift is possible through a fast-growing new field of ML technologies and applications – referred to as tiny machine learning (TinyML) - capable of performing on-device sensor data analytics at very low power consumption [1]. Sitting at the intersection of embedded systems and machine learning, tinyML requires innovations at both software (SW)/algorithmic and hardware (HW) levels to be able to migrate complex and sophisticated ML algorithms from the cloud (where huge computational resources and vast amounts of memory are available) to embedded system devices sitting at the edge (with limited processing and memory resources).

Edge devices have constraints on size, memory, power, and latency - which render traditional cloud-based processing impractical. Therefore, deploying ML models to the edge is particularly challenging in embedded vision systems, where images and videos represent the primary input data into these models. For example, the computational complexity of convolutional neural network (CNN) models, which are very popular in computer vision, presents a significant challenge to performing inference solely on low-power embedded systems. Consequently, hardware designs that integrate two or more edge devices have emerged as a promising solution for accelerating vision workloads at the edge. One such hybrid approach involves combining microcontroller units (MCUs) ease of integration and networking capabilities while offloading intensive parallel computation to a field programmable gate array (FPGA), thereby achieving substantial improvements in throughput and energy efficiency.

Recent advancements in lightweight CNNs and hardware-efficient inference have empowered edge platforms to execute increasingly complex vision tasks without reliance on GPUs or external DRAM. In this context, MCU-FPGA co-design represents a promising intermediate approach between purely software-based TinyML methods and high-end System-on-Chip (SoC) accelerators. Existing designs vary widely in communication strategy, compute partitioning, and hardware scale, from large Zynq-class boards employing DDR-backed pipelines and AXI-Stream or DMA dataflows [2]–[7] to smaller, sensor-oriented FPGA systems that process streaming inputs directly from cameras or peripheral devices [8], [9]. Exploration of this design space reveals a fundamental trade-off between system complexity, performance, and deployability at the edge, motivating continued investigation into low-cost hybrid architectures such as the one presented in this work.

It is in this hybrid category of approaches that this work introduces EdgeCube - a compact MCU-FPGA hardware platform designed for realtime image classification. It combines

Table I
SUMMARY OF RELATED ACCELERATOR IMPLEMENTATIONS

Reference	Approach / Model Used	Image Size	HLS / Manual	FPGA / Board Used	Speedup / Improve.
Hamdan & Rover [10]	VHDL Generation Tool	28×28; 32×32	Generated / Manual	Virtex-7	6.1×; 611.52 GOPS
Lu *et al.* [2]	Sparse-CNN	224×224	Manual	Zynq	309 GOPS; 12.9×
Ahmed *et al.* [11]	MAC Cell for CNN	N/A	Manual	Cyclone / Arria / Stratix	2.27–4.17×
Wang *et al.* [5]	CNN	28×28	Manual	Zynq	36.1×
Zhu *et al.* [6]	CNN	224×224	Manual	Zynq	1.5–6.7×
Zhang [7]	EdgeNet-IoT	224×224	Manual	Zynq	2.8–10.2×
Wi *et al.* [12]	CNN	224×224	Manual	Simulation Only	4.52–7.70×
Neris *et al.* [8]	Multiple CNNs	256×256; 512×512	HLS	Kintex	5.57–34.7×
Bjerge *et al.* [3]	CNN	224×224	HLS	Zynq	6.0–15.2 GOPS/W
Huang *et al.* [13]	CNN	224×224	HLS / RTL	Virtex-7	600–1000 GOPS
Srilakshmi *et al.* [14]	CNN	N/A	Both	Artix-7	198–339×
Chang *et al.* [4]	RNN	N/A	Not stated	Zynq	23×
Carpegna *et al.* [15]	SNN	28×28	Not stated	Artix-7	186×
Sahu *et al.* [16]	DNN	N/A	N/A	MSP430	1.12–8.45×
Lage *et al.* [9]	CNN	320×240	N/A	Raspberry Pi / UP Squared	2.1–7.3 FPS
EdgeCube (this work)	**CNN**	**96×96**	**Both**	**Artix**	**26.14×**

an ESP32-CAM microcontroller board for image acquisition and main control with an FPGA device that implements a CNN accelerator as the computational backend. The proposed architecture exploits the MCU's integrated camera interface and wireless communication, while the FPGA executes in parallel heavy dense convolutional operations required by the CNN model. This division of work enables the system to maintain a low power envelope while achieving realtime inference times, making it suitable for embedded applications such as low-power vision sensing, occupancy detection, and environmental monitoring.

II. PREVIOUS WORK

The problem of accelerating convolutional neural networks (CNNs) and deep neural networks (DNNs) on reconfigurable hardware has been studied in previous work - but more in the general context of using powerful (i.e., high-performance, but power hungry and expensive) computers and FPGA boards and not on resource-constrained edge hardware platforms. A summary of previous related work is presented in Table I. It is interesting and beneficial to classify previous studies into two main categories: *manual custom* approaches and *automatic high-level* approaches.

In the first category, early work emphasized traditional handcrafted register-transfer level (RTL) designs for CNNs. They included VHDL generators [10] and vendor-agnostic multiply-and-accumulate (MAC) architectures [11]. These approaches maximized throughput by fully pipelining convolution and pooling operations needed by CNNs and by relying heavily on the HW logic parallelization and embedded memory (used for CNN model weights storage) offered by FPGA devices. Later studies [2], [6] further improved this efficiency through structured or unstructured sparsity, skipping zero-valued operations to reduce bandwidth and energy. Parallel efforts on system-level optimization explored tile-based dataflows and scalable matrix-multiplication units to handle larger networks [5], [13] - both demonstrating high DSP utilization, but requiring large Zynq or Virtex FPGA devices with external high-capacity DDR memories.

In the second category, high-level synthesis (HLS) tools became a dominant trend for making FPGA acceleration more accessible to designers lacking hardware description language (HDL) skills. HLS tools can synthesize VHDL/Verilog implementations directly from design specifications in higher level of abstraction languages like C/C++, SystemC, or Matlab - making FPGAs accessible to a wider range of designers. Recent studies [14] quantified the area and timing trade-offs achieved by HLS-based and traditional HDL design approaches, showing that HLS-based approaches (which rely on careful/intelligent pragma usage) can provide performance comparable to the performance of handcrafted RTL solutions, but at drastically reduced design time. Other previous works [3], [10] leveraged C/C++ or SystemC-based toolflows to generate modular CNN layers, integrating convolution, pooling, and fully connected stages within reusable templates. Such workflows enable rapid parameter tuning and automatic quantization, but still depend on high-end FPGAs, such as the Zynq Ultra96 or Kintex UltraScale devices. Despite these advances, communication overheads and external DRAM access continue to dominate latency in many HLS-based accelerators, leaving a research-gap that could be filled by tighter coupling of hybrid MCU–FPGA designs, particularly when off-chip memory is limited or not available.

In addition to CNN acceleration, there is previous work that focused on other types of deep neural networks acceleration with FPGAs. These include spiking-neural-network accelerators [15] emphasizing energy efficiency through event-driven computation and recurrent neural networks implemented with weight-banked LSTM engines [4]. Clever optimization techniques - such as aggressive ternary weight compression and encoding [12] and co-design of FPGA–SoC architectures that integrate quantized depthwise CNNs with runtime stacks for IoT analytics [7] - further reduced model footprints below one bit per weight while preserving (or minimally degrading) performance. These studies highlight a broader recent push toward compact, energy-efficient ML models used for for real-time inference. However, generally they still depend on

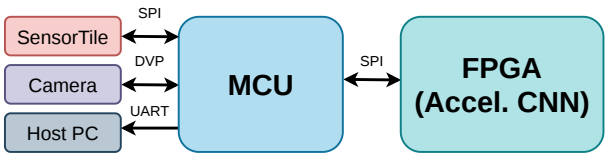

Figure 1. System level diagram of the proposed EdgeCube system.

sophisticated SoC or ASIC-like platforms with integrated ARM processors and high-bandwidth memory interfaces.

On a different line of research, previous studies examined distributed and heterogeneous execution strategies at the microcontroller level. Frameworks that partition neural inference across multiple constrained nodes have been shown to enable full-accuracy inference on memory-limited hardware [16], while local person-detection systems - implemented on single-board computers with optional hardware acceleration - demonstrated local processing for privacy-preserving applications [9]. Other approaches achieve high-resolution remote-sensing CNN inference on radiation-tolerant FPGAs through streaming-based data capture to reduce buffering overhead [8]. Collectively, these studies illustrate how partitioning, task allocation, and communication strategies impact system scalability and energy efficiency. However, they still target hardware platforms with substantial computational and memory resources or rely on distributed clusters rather than truly low-cost, single-device embedded systems.

In contrast, the proposed EdgeCube hardware platform targets the same design goals (high performance real-time inference, energy efficiency, hardware modularity, and reduced costs) but does so on a far smaller physical and computational scale. Whereas prior implementations often assume large SoC FPGAs, high-bandwidth DDR memories, or multi-board setups, our light system combines an ESP32-CAM microcontroller and a low-cost FPGA connected only through SPI. This hybrid hardware platform encapsulates the complete vision pipeline - needed for the person detection application targeted in this work - within a single, inexpensive embedded device. By demonstrating that real-time CNN inference can be achieved through a streaming MCU + FPGA hybrid approach, EdgeCube extends the idea of hardware acceleration and edge-AI technologies (such as TinyML) towards resource-minimal practical embedded systems.

III. Proposed EdgeCube System

A. System Level Description

The proposed system implements a modular hardware-software architecture that enables real-time image classification on resource-constrained embedded platforms. A simplified system-level block diagram of the proposed EdgeCube hardware platform is presented in Fig. 1. The design is composed of several cooperating subsystems: a microcontroller-based image acquisition and control unit, a serial data transport interface, an on-chip dual-port memory buffer, and a streaming CNN accelerator implemented on an FPGA.

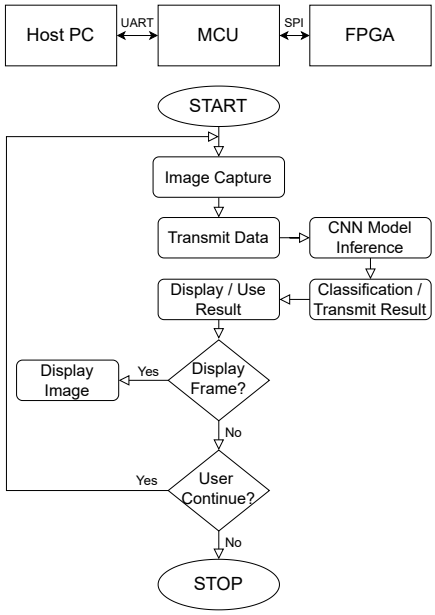

Figure 2. Flowchart of the top-level control algorithm of the proposed EdgeCube system.

System operation follows a deterministic pipelined and per-frame oriented workflow. The microcontroller acquires a grayscale image (i.e., a frame) from the camera sensor, formats it into a sequential stream of pixel data, which is synchronously transmitted to the FPGA device. The FPGA receives the pixel data into a dedicated memory buffer and processes the received image through the accelerated pipelined CNN model implementation. In other words, this accelerated CNN model executes efficiently real-time inference; that is, it predicts whether or not the input image contains or not a person. The result of the predicted classification is returned back to the microcontroller. The design eliminates the need for external memory or host-side processing and supports continuous operation under strict latency constraints. This top-level partitioning allows computation and communication to be managed independently, while maintaining a lightweight hardware-software interface.

B. Microcontroller Processing and Control

The microcontroller unit (MCU) executes a streamlined control loop responsible for image acquisition, pixel formatting, serial transmission, and inference retrieval. The main control algorithm run by the MCU is described with the help of the flowchart presented in Fig. 2. Processing begins with frame capture from the camera sensor, followed by linearization of the two-dimensional image data. If quantization or normalization is required, such operations are applied per pixel prior to transmission. A frame-start signal initiates a transaction with the FPGA. Pixel data are transmitted in a raster-scan order using a serial protocol with deterministic timing. Upon completion of the transfer, the MCU waits for an inference-ready indicator before performing a single-byte readback. The returned value encodes the predicted class (person present or

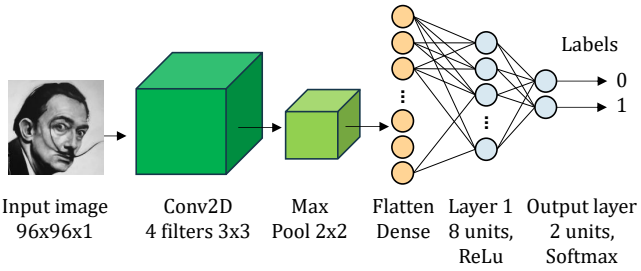

Figure 3. CNN model implemented on the FPGA device. The two classes corresponding to labels {0,1} are "No person" and "Yes person"' in frame.

not in frame) by the CNN model executed on the FPGA, and may be used for application-level decision making or performance logging. All MCU functions are implemented without dynamic memory allocation or asynchronous execution, ensuring predictable timing behavior and simplifying synchronization with the hardware accelerator. Because the FPGA accelerator processes the frame independently, the MCU remains responsible solely for data preparation and communication management.

C. FPGA Based CNN Accelerator

We next present a brief description and specific details of the CNN model, which we developed for the person detection application, and which is entirely deployed onto the FPGA device for maximum acceleration. CNN models have been very popular in computer vision applications. As with other types of machine learning models, a challenging aspect regarding CNN model development and optimization is how to define various model hyperparameters, including network architecture, number of layers, number of units per layer, number of filters, batch size, number of epochs used for training, etc. In our case, we selected several of these hyperparameters based on our previous experience, on recent literature dealing with similar CNN models, and on an empirical trial-and-error process during which several different values for model parameters have been investigated as follows: we started from a CNN architecture with 3 layers and larger frame sizes; then, we incrementally decreased the number of layers and frame size in order for the synthesized VHDL description of the CNN model to fit within the resources of the given FPGA device while the classification accuracy was kept as high as possible. In addition to the model architecture parameters illustrated in this figure, we used during training: $sparse\_categorical\_crossentropy$ as loss function, $Adam$ as an optimizer, and a batch size of 32. The final optimized CNN model used in this paper is shown in Fig. 3.

During model optimization, training was done for a number of 30 epochs using the publicly available dataset called the Human Detection Dataset, which is available for download at [17]. This dataset contains CCTV footage images (both indoor and outdoor). It includes 362 images without humans and 559 images with humans. Once the CNN model is optimized in Tensorflow, we apply tinyML optimization to it using the TensorFlow Lite Micro toolchain from Google. This optimization employs quantization and significantly reduces the size of the model in order to be deploy-able on resource constrained edge devices. The model summary is shown in Table II, which shows the total number of parameters and the model size before and after the optimization using tinyML technologies, for comparison purposes.

Table II
CNN MODEL DATA IN TENSORFLOW AND TENSORFLOW LITE (TFL) MICRO LEVELS.

Characteristic	CNN Tensorflow	CNN TFL Micro
Total params	73,794	73,794
Model size	288.26 KB	75.5 KB

The CNN accelerator is implemented as a synthesizable hardware IP block that operates on quantized image data stored in dual-port memory. The accelerator consists of four principal components: (1) a sliding-window convolution engine, (2) a spatial pooling module, (3) a dense classification layer, and (4) output selection logic. The convolution engine employs a line-buffered sliding-window structure enabling an initiation interval of one cycle. Multiply–and-accumulate operations for each receptive field are executed in parallel, and intermediate results are forwarded to the pooling stage. The pooling module performs spatial downsampling to reduce feature-map dimensionality and improve translation robustness. The flattened pooled output is passed to a dense layer implemented using a small vector dot-product unit. The accelerator produces two logits, and an argmax operation selects the predicted class. The class bit is encoded into an 8-bit output format for compatibility with the serial transport interface. The accelerator is generated from high-level C/C++ descriptions using Vitis 2024.2 hardware synthesis tools with explicit pipeline and memory-partitioning directives. As a result, the core sustains continuous throughput and exhibits deterministic latency dominated by pipeline depth rather than input resolution.

IV. EDGECUBE PROTOTYPE

A custom PCB layout prototype of EdgeCube (see Fig. 4) was designed and assembled to evaluate the person detection application on real hardware under realistic embedded conditions. The prototype primarily combines an ESP32-CAM microcontroller board [18] with a Numato Lab Mimas A7 Mini FPGA board [19]. The ESP32-CAM board uses an ESP32-D0WD low power 32-bit CPU with up to 240 MHz clock speed and built-in 520 KB SRAM. The FPGA board uses an AMD Xilinx Artix 7 FPGA device capable of operating at 100 MHz clock speed. The board includes a 2Gb DDR3 memory chip, which we use for image buffering and storage.

The MCU controls the camera's OV2640 sensor to retrieve grayscale frames of 96x96 pixels. The frames are streamed (using SPI communication standard operating at 80 MHz) as pixel sequences to the FPGA device using a frame-start strobe followed by raster-ordered transmission. On the FPGA side, the CNN model receives frames and performs realtime inference. The CNN model (discussed in section III-C) was trained

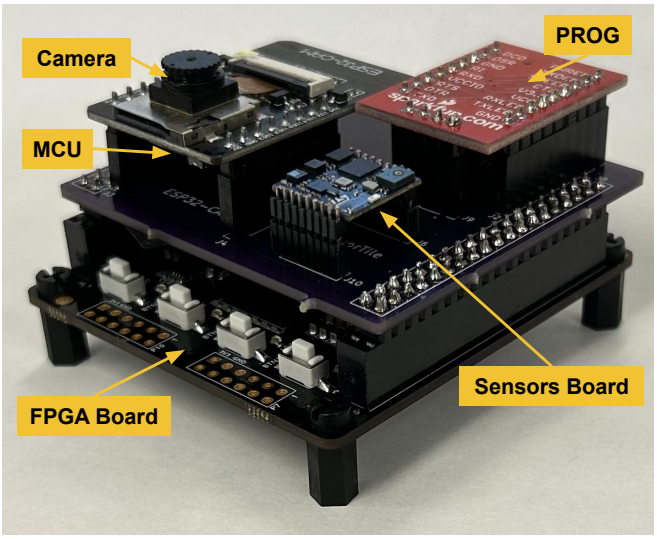

Figure 4. Photograph of the current prototype of EdgeCube edge node.

and quantized using the TensorFlow Lite Micro toolchain [20], and the resulting 8-bit parameters were embedded directly into the synthesized CNN accelerator deployed onto the FPGA device. The output of the CNN model's (i.e., person detected or not) is returned to the MCU as a single-byte class result via SPI with a ready/handshake line. Within the FPGA, the serial interface, dual-port frame buffer, and CNN core operate as a unified pipeline, allowing inference to begin immediately once a frame is received. The prototype requires no external memory or host processor and functions as a fully self-contained platform for performance evaluation, forming the basis for the experimental results presented in the next section.

As an optional diagnostic mode, the MCU can also send captured frames for visualization on the host PC over the USB-to-serial interface. On the host PC, a companion Python utility reconstructs incoming frames by detecting a frame marker, reading the image dimensions, and assembling the grayscale payload into a rendered image. This mode can be toggled at compile time and is not intended to be used during normal operation (in order to not slow down the MCU) of EdgeCube; it is used only for development and debug purposes.

V. Experimental Results and Discussion

A. Experimental Results

Using the hardware prototype described in the previous section, we conducted a series of experiments to evaluate the performance of EdgeCube with the person detection application deployed on it. In the first experiment, we have implemented an MCU-only approach, which serves as our base/reference for comparison purposes. In the second experiment, we deploy the person detection application on the EdgeCube platform, which uses the FPGA for acceleration of inferencing with the trained CNN model. In both experiments, we collect timing measurements measured directly at the MCU level in order to ensure a consistent comparison between the base and the proposed approaches. The timing measurements are done for each of the three main steps of the processing pipeline: image capture, pre-processing and transmission from MCU to FPGA, and inferencing on FPGA. The results of these experiments are summarized in Table III.

Table III
Comparison of latency achieved with the MCU-onnly baseline and with the FPGA-accelerated approach.

Step	**MCU-Only (μs)**	**MCU+FPGA (μs)**
Frame capture	5	6
Pre-processing & TX	863	3626
CNN-based inference	111483	627
Total latency	112351	4260
Speed-up	26.14×	

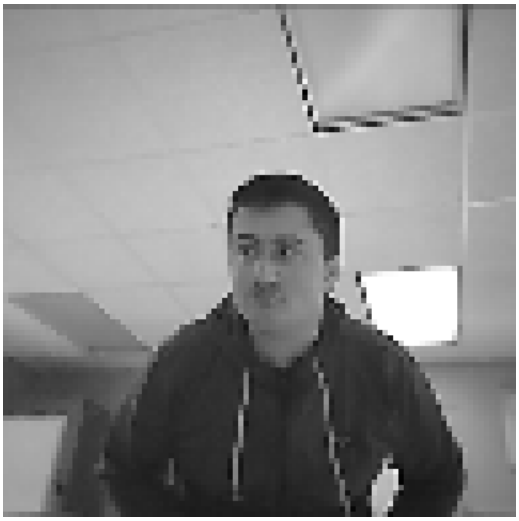

Figure 5. Example of a frame received on the host computer as a 96x96 grayscale image during real-time diagnostic mode of operation.

As discussed in the previous section, EdgeCube platform was designed with a real-time diagnostic mode, which is very useful for real-time functional validation. This feature can be enabled only during testing and verification and not during normal operation. In this diagnostic mode, frames from the camera (which during normal operation are streamed only to the FPGA) are sent over the serial-to-USB to the host computer as well. On the host computer, a Python program receives the frame data and simply displays them as images. An example frame received and displayed on the host PC in this mode is shown in Fig. 5. This visualization confirmed correct pixel ordering, frame integrity, and consistent grayscale capture across lighting conditions, providing an additional verification mechanism for accelerator behavior.

B. Discussion

As we see in Table III, the MCU-only approach required an average of 112.35 ms per inference. As expected, this time is dominated by the time it takes for the processing of all convolutional operations, which must be done on the MCU's processor. In contrast, the proposed MCU+FPGA implementation achieved a total latency of 4.26 ms. The

accelerator's internal compute tail measured only 627 μs, confirming that computation runs substantially faster than data can be streamed. The MCU+FPGA implementation achieved a testing accuracy of 71.1%, compared to 72.3% for the MCU-only implementation, indicating that transferring the CNN model to hardware resulted in negligible information loss. Overall, the FPGA-accelerated implementation provided a 26.14x speedup and consistent model accuracy compared to the baseline.

In both experiments, we measured the average power consumption with the help of a USB power tester. The power consumed by the MCU-only implementation was 0.48 W while the power consumed by the MCU+FPGA implementation was 1.62 W. In terms of FPGA device resource utilization, the MCU+FPGA implementation used 7.8% of LUTs, 4.0% of FFs, and 60% of DSP slices. The primary constraint was on-chip memory, with 84% of BRAM consumed by the dual-port frame buffer and line-buffer components. Despite this, the device supports considerable scaling potential for deeper CNNs or multi-layer pipelines.

These results demonstrate that performance is constrained primarily by the serial-transfer time rather than by computational throughput. Because the accelerator completes inference within only a few hundred microseconds, deeper or more complex CNN architectures could be integrated to increase performance without exceeding real-time latency bounds. Logic and DSP utilization remain low, while memory is the dominant limiting factor; however, tiling or compressed buffering schemes can mitigate BRAM pressure. Overall, the EdgeCube prototype validates that real-time embedded inference can be achieved using a compact hybrid MCU–FPGA hardware platform without external memory or host processing.

VI. Conclusion and Future Work

We presented EdgeCube, a compact hybrid edge device for the Internet of Things (IoT) that combines an MCU with a camera sensor and an FPGA device for real-time embedded image classification. The proposed hardware platform is designed as a generic smart sensor platform that could be extended with additional sensors. Currently, it was prototyped using an ESP32-CAM MCU board and an AMD Xilinx Artix 7 FPGA device and was used to demonstrate the classic person detection application. The FPGA device is used to accelerate a pretrained CNN model for person detection, while the MCU captures frames from the camera sensor, transmits data to the FPGA and executes the system level controls. Experimental evaluation demonstrated a 26.14x speed-up over a microcontroller-only execution. As future work, it would be interesting to investigate how to generalize the accelerator architecture to support multiple model types, such as multi-layer CNNs, depthwise networks, or even non-CNN models. Second, integrating additional sensors, including environmental, audio, or motion modules, would allow multimodal fusion to expand the system's utility and performance in a wider range of edge applications.

References

[1] Pete Warden and Daniel Situnayake, "TinyML: Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers," O'Reilly Media, 2020.

[2] L. Lu, J. Xie, R. Huang, J. Zhang, W. Lin, and Y. Liang, "An efficient hardware accelerator for sparse convolutional neural networks on fpgas," in *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2019, pp. 17–25.

[3] K. Bjerge, J. H. Schougaard, and D. E. Larsen, "A scalable and efficient convolutional neural network accelerator using hls for a system-on-chip design," *Microprocessors and Microsystems*, vol. 87, p. 104363, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0141933121005160

[4] A. X. M. Chang and E. Culurciello, "Hardware accelerators for recurrent neural networks on fpga," in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2017, pp. 1–4.

[5] C. Wang, L. Gong, Q. Yu, X. Li, Y. Xie, and X. Zhou, "Dlau: A scalable deep learning accelerator unit on fpga," *IEEE TCAD*, vol. 36, no. 3, pp. 513–517, 2017.

[6] C. Zhu, K. Huang, S. Yang, Z. Zhu, H. Zhang, and H. Shen, "An efficient hardware accelerator for structured sparse convolutional neural networks on fpgas," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 9, Jan 2020.

[7] X. Zhang, "Edge-optimized neural network architecture for real-time iot data processing: A hardware-software co-design approach," in *2025 4th International Conference on Artificial Intelligence, Internet of Things and Cloud Computing Technology (AIoTC)*, 2025, pp. 409–412.

[8] R. Neris, A. Rodríguez, R. Guerra, S. López, and R. Sarmiento, "Fpga-based implementation of a cnn architecture for the on-board processing of very high-resolution remote sensing images," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 15, pp. 3740–3750, 2022.

[9] E. S. Lage, R. L. Santos, S. M. Junior, and F. Andreotti, "Low-cost iot surveillance system using hardware-acceleration and convolutional neural networks," in *2019 IEEE 5th World Forum on Internet of Things (WF-IoT)*, 2019, pp. 931–936.

[10] M. K. Hamdan and D. T. Rover, "Vhdl generator for a high performance convolutional neural network fpga-based accelerator," in *2017 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, 2017, pp. 1–6.

[11] H. O. Ahmed, M. Ghoneima, and M. Dessouky, "Concurrent mac unit design using vhdl for deep learning networks on fpga," in *2018 IEEE Symposium on Computer Applications Industrial Electronics (ISCAIE)*, 2018, pp. 31–36.

[12] H. Wi, H. Kim, S. Choi, and L.-S. Kim, "Compressing sparse ternary weight convolutional neural networks for efficient hardware acceleration," in *2019 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, 2019, pp. 1–6.

[13] W. Huang, H. Wu, Q. Chen, C. Luo, S. Zeng, T. Li, and Y. Huang, "Fpga-based high-throughput cnn hardware accelerator with high computing resource utilization ratio," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 8, pp. 4069–4083, 2022.

[14] S. Srilakshmi and G. Madhumati, "A comparative analysis of hdl and hls for developing cnn accelerators," in *2023 Third International Conference on Artificial Intelligence and Smart Energy (ICAIS)*, 2023.

[15] A. Carpegna, A. Savino, and S. Di Carlo, "Spiker: an fpga-optimized hardware accelerator for spiking neural networks," in *2022 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2022, pp. 14–19.

[16] R. Sahu, R. Toepfer, M. D. Sinclair, and H. Duwe, "Denni: Distributed neural network inference on severely resource constrained edge devices," in *2021 IEEE International Performance, Computing, and Communications Conference (IPCCC)*, 2021, pp. 1–10.

[17] K. Verner, "Human detection dataset," Feb 2022. [Online]. Available: https://www.kaggle.com/datasets/constantinwerner/human-detection-dataset

[18] AI-Thinker, "ESP32 Cam WiFi Bluetooth Development Board with OV2640 Camera Module," https://media.digikey.com/pdf/Data%20Sheets/DFRobot%20PDFs/DFR0602_Web.pdf, 2024.

[19] Numato Lab, "Mimas A7 Mini FPGA Development Board," https://numato.com/product/mimas-a7-mini-fpga-development-board/, 2024.

[20] Google, "TensorFlow Lite for Microcontrollers," https://ai.google.dev/edge/litert/microcontrollers/overview, 2025.