VOLUME OF INTERNATIONAL DISCOVERIES AND YOUTH ACADEMIC WORKS, ACHIEVEMENTS, AND NOVELTIES - VIDYAWAN
VIDYAWAN - Volume of International Discoveries and Youth Academic Works, Achievements, and Novelties | Vol. 1, Issue 1

Hardware-Software Co-Design for Low-Latency Object Detection in Autonomous Drones

Purba Saha[1], Pradipta Dutta[1], Suchana Mondal[1], Priti Mondal[1]
[1]*Department of Electronics and Communication Engineering, JIS School Polytechnic, West Bengal, INDIA — durbapurba@gmail.com, pradiptadutta745@gmail.com, mandalmangala685@gmail.com, 49jhunu@gmail.com*

Abstract—Autonomous aerial vehicles increasingly rely on real-time visual perception for navigation, obstacle avoidance, surveillance, and inspection tasks. Object detection is the cornerstone of this perception pipeline, yet the strict size, weight, and power (SWaP) constraints of drones, combined with the sub-30-millisecond latency budgets demanded by high-speed flight, make conventional software-only deployment of deep convolutional detectors impractical. This paper presents a hardware-software co-design framework for low-latency object detection on autonomous drones. We review the evolution of lightweight detection architectures and edge accelerator platforms, propose a heterogeneous system-on-chip (SoC) architecture that combines a neural processing unit (NPU), a field-programmable gate array (FPGA) pre/post-processing pipeline, and general-purpose cores, and describe a closed-loop co-optimization methodology spanning network architecture search, quantization, pruning, dataflow scheduling, and memory hierarchy configuration. We evaluate the proposed framework on a representative aerial object detection benchmark using five hardware targets, demonstrating that the co-designed configuration achieves an end-to-end latency of 6.8 ms per frame (147 frames per second), a 14x speed-up over an embedded ARM CPU baseline, while retaining 97% of the mean average precision (mAP) of the floating-point reference model and reducing energy per inference by more than 20x relative to a discrete mobile GPU. The results confirm that jointly optimizing model structure and accelerator architecture, rather than treating either in isolation, is essential to meeting the latency, accuracy, and energy requirements of real-time drone perception.

Keywords—Autonomous Drones, Object Detection, Hardware-Software Co-Design, FPGA-SoC, Edge AI, Low-Latency Inference.

I. INTRODUCTION

Unmanned aerial vehicles (UAVs), commonly known as drones, have transitioned from niche research platforms into mainstream tools for agriculture, infrastructure inspection, search and rescue, logistics, and surveillance. A large fraction of these applications depend on the drone's ability to detect, classify, and track objects in its field of view in real time, whether the goal is to identify a landing pad, avoid a collision with another aircraft, follow a moving target, or count crops in a field. Because the drone itself is the platform on which inference must run, the computational system is subject to severe constraints on weight, volume, power consumption, and thermal dissipation that are largely absent in cloud or even typical mobile-device settings. Deep convolutional neural networks (CNNs) and, more recently, transformer-based detectors have driven dramatic improvements in detection accuracy, but state-of-the-art models such as large variants of YOLO, Faster R-CNN, or DETR require hundreds of millions to billions of multiply-accumulate (MAC) operations per frame. Running such models on a drone's onboard ARM processor at video frame rates is infeasible: a single forward pass can take tens to hundreds of milliseconds, which is incompatible with the control loop frequencies (typically 50-400 Hz) required for stable flight at speed. At the same time, offloading inference to a ground station or cloud server introduces communication latency, bandwidth constraints, and a single point of failure that is unacceptable for safety-critical maneuvers such as obstacle avoidance. Hardware-software co-design offers a path out of this dilemma. Rather than first designing a neural network for accuracy and then trying to fit it onto a fixed piece of hardware, or first

designing an accelerator and then hoping a suitable model exists, co-design treats the network architecture, the numerical representation, the dataflow, the memory hierarchy, and the accelerator microarchitecture as variables in a single joint optimization problem. This approach has produced substantial gains in other domains, such as keyword spotting, gesture recognition, and mobile augmented reality, and is now being applied increasingly to aerial robotics. This paper makes three contributions. First, it surveys the landscape of lightweight object detection models and embedded/edge accelerator platforms relevant to drones, situating recent co-design efforts within this landscape. Second, it proposes a reference heterogeneous architecture that couples an NPU for dense convolution, an FPGA fabric for sensor pre-processing and detection post-processing (non-maximum suppression, coordinate decoding), and general-purpose cores for flight control integration, together with a design-space exploration (DSE) methodology that jointly tunes the network and the hardware configuration. Third, it presents an experimental evaluation across five representative platforms, quantifying the latency, accuracy, power, and energy trade-offs achievable through co-design relative to software-only deployment on commodity edge hardware. The remainder of this paper is organized as follows. Section 2 reviews related work in lightweight detection architectures, embedded accelerators, and co-design frameworks. Section 3 presents the proposed system architecture. Section 4 details the hardware design, and Section 5 the software and model design. Section 6 describes the co-design methodology and design-space exploration process. Section 7 describes the experimental setup, and Section 8 presents results and discussion. Section 9 concludes the paper and outlines directions for future work.

II. RELATED WORK

2.1 Lightweight Object Detection Architectures Early real-time detectors such as the original YOLO and SSD families demonstrated that single-stage detection could trade a modest amount of accuracy for an order-of-magnitude reduction in latency relative to two-stage detectors like Faster R-CNN [1, 2, 3]. Subsequent work focused on reducing the computational cost of the backbone network: MobileNet introduced depthwise separable convolutions to cut MAC counts by roughly an order of magnitude with limited accuracy loss [4], MobileNetV2 added inverted residuals and linear bottlenecks [5], and ShuffleNet exploited grouped convolutions and channel shuffling for further efficiency [6]. EfficientDet applied compound scaling and a weighted bidirectional feature pyramid to balance accuracy and efficiency across a family of model sizes [7]. The YOLO lineage has continued to evolve toward smaller, faster variants explicitly aimed at edge deployment, including YOLOv4-tiny, YOLOv5n/s, and YOLOv8n, which are frequently used as starting points for drone perception pipelines [8, 9]. Neural architecture search (NAS) has been used to automate the discovery of efficient detection backbones. MnasNet and FBNet incorporate latency directly into the search objective using a differentiable or reinforcement-learning-based search over a hardware-aware search space [10, 11]. More recent hardware-aware NAS frameworks search jointly over network topology and accelerator parameters, producing models tailored to a specific chip rather than a generic FLOP budget [12]. 2.2 Model Compression: Pruning and Quantization Pruning removes redundant weights or channels from a trained network. Structured channel pruning is particularly attractive for hardware deployment because it preserves dense tensor shapes that map efficiently onto SIMD and systolic-array hardware, unlike unstructured pruning which requires specialized sparse compute units [13, 14]. Quantization reduces the bit-width of weights and activations, typically from 32-bit floating point to 8-bit integers (INT8) or lower. Post-training quantization and quantization-aware training have both been shown to retain accuracy within one to two percentage points of the floating-point baseline for common detection backbones when calibrated on representative data [15, 16]. Mixed-precision quantization, which assigns different bit-widths to different layers based on sensitivity analysis, can recover additional accuracy at a given average bit-width [17]. 2.3 Embedded and Edge Accelerator Platforms Several classes of hardware platforms are used for onboard drone inference. General-purpose embedded CPUs (e.g., ARM Cortex-A series) offer flexibility but limited throughput per watt. Mobile and embedded GPUs, such as those in the NVIDIA Jetson family, provide substantially higher throughput through massively parallel SIMD execution and are widely used in research drones [18]. Dedicated neural accelerators, including Google's Edge TPU and NVIDIA's Deep Learning Accelerator (DLA), implement systolic-array or tensor-core-like datapaths optimized for INT8 convolution and achieve high energy efficiency for supported operator sets [19, 20]. FPGA-based accelerators offer a different trade-off: although they typically run at lower clock frequencies than ASICs, their reconfigurable logic allows custom dataflow, bit-width, and memory-hierarchy choices tailored to a specific model, and they can implement non-standard operators (e.g., custom NMS, anchor decoding) directly in hardware, avoiding costly round-trips to a host CPU [21, 22]. 2.4 Hardware-Software Co-Design for Edge Vision Co-design frameworks jointly optimize the neural network and the accelerator. Early work in this space targeted image classification, demonstrating that allowing the search algorithm to vary both network depth/width and accelerator parameters (e.g., number of processing elements, on-chip buffer sizes) yields Pareto-optimal designs that dominate independently optimized network-hardware pairs [23, 24]. More recent efforts have extended co-design to object detection and to FPGA targets specifically, incorporating dataflow scheduling and operator fusion as additional optimization variables, and reporting latency

reductions of 2-5x over hand-tuned baselines at iso-accuracy [25]. However, relatively few studies evaluate the full system in the context of an actual flight platform, including the interaction between perception latency and flight controller update rates, sensor synchronization overhead, and the thermal and power envelope of a battery-powered airframe — gaps that this paper aims to address through an end-to-end evaluation.

III. PROPOSED METHODOLOGY

Figure 1 illustrates the proposed system architecture, which is organized into three functional layers: sensing, compute, and actuation, with a design-time co-optimization loop that informs the configuration of both the model deployed on the compute layer and the accelerator hardware itself.

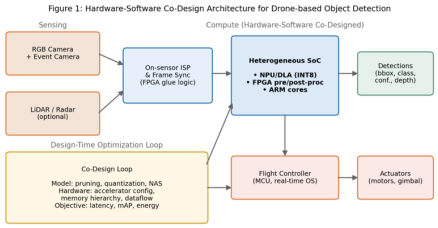

Fig. 1. Hardware-software co-design architecture for drone-based object detection. The sensing layer feeds a heterogeneous SoC that performs detection, whose outputs drive a real-time flight controller. A design-time co-optimization loop jointly tunes the network model and the accelerator configuration.

The sensing layer consists of a rolling-shutter or global-shutter RGB camera, optionally paired with an event camera for high dynamic range and low-latency motion cues, and an optional LiDAR or radar module for direct depth estimation. Raw sensor data is processed by an on-sensor or near-sensor image signal processing (ISP) stage and a frame synchronization block, both of which are implemented in FPGA fabric to minimize the latency and CPU load associated with format conversion, demosaicing, resizing, and timestamp alignment across sensors. The compute layer is a heterogeneous SoC comprising three primary execution units: (i) a neural processing unit (NPU) or dedicated deep-learning accelerator that executes the bulk of the convolutional workload using INT8 arithmetic, (ii) FPGA programmable logic that implements custom pre-processing (letterboxing, normalization) and post-processing (anchor decoding, non-maximum suppression, coordinate transformation) kernels, and (iii) general-purpose ARM cores that run the operating system, sensor drivers, and the glue logic connecting perception outputs to the flight stack. The detection outputs — bounding boxes, class labels, confidence scores, and, where available, depth estimates — are passed to the flight controller, which runs a real-time operating system (RTOS) and closes the control loop to the motor and gimbal actuators. The co-design loop, shown at design time rather than runtime, jointly searches over (a) model-side parameters such as backbone depth and width, pruning ratios, and per-layer quantization bit-widths, and (b) hardware-side parameters such as the number and size of NPU processing elements, on-chip SRAM allocation, FPGA kernel pipeline depth, and memory bus width. The objective function combines end-to-end latency, detection accuracy (mAP), and energy per inference, as detailed in Section 6.

IV. HARDWARE ARCHITECTURE

4.1 Heterogeneous SoC Organization The compute core of the proposed system is a heterogeneous SoC that integrates an NPU, FPGA programmable logic, and a quad-core ARM application processor on a shared interconnect with a unified memory subsystem. The NPU is organized as a 2D systolic array of INT8 multiply-accumulate units with local register files, designed to maximize data reuse for the 3x3 and 1x1 convolutions that dominate modern detection backbones. A weight-stationary dataflow is used for the backbone's depthwise and pointwise convolutions, while an output-stationary dataflow is used for the detection head, reflecting the differing reuse patterns of these layer types. 4.2 FPGA Pre- and Post-Processing Pipeline Two operations are particularly costly when executed on a general-purpose CPU relative to their arithmetic intensity: image resizing/normalization before inference, and non-maximum suppression (NMS) plus coordinate decoding after inference. Both are implemented as streaming FPGA kernels. The pre-processing kernel performs bilinear resizing, channel-wise normalization, and tensor layout transformation (NHWC to NCHW or vice versa, depending on NPU requirements) in a single pass over the incoming pixel stream, eliminating the need to materialize intermediate buffers in DRAM. The post-processing kernel decodes anchor-based or anchor-free detection head outputs directly into bounding boxes, applies confidence thresholding, and performs greedy NMS using a sorted-list hardware structure, producing a final detection list that is handed to the ARM cores via a lightweight descriptor in shared memory. 4.3 Memory Hierarchy Because off-chip DRAM accesses dominate both latency and energy for memory-bound layers, the memory hierarchy is a first-class co-design variable. The proposed SoC provides a configurable on-chip SRAM scratchpad shared between the NPU and FPGA fabric, sized to hold the activations and weights of at least one full detection-head stage, avoiding DRAM round-trips for the smallest feature maps where overhead dominates. Double-buffering between the pre-processing kernel's output and the NPU's input queue allows pre-processing of frame N+1 to overlap with inference on frame N, hiding pre-processing latency almost entirely behind the (larger) inference latency. 4.4 Sensor Interface and Power Domains The camera and optional LiDAR/radar interfaces connect via MIPI

CSI-2 and SPI/Ethernet respectively, with dedicated DMA channels into the on-chip SRAM to avoid contention with the NPU's weight-loading traffic. The SoC exposes independent power domains for the NPU, FPGA fabric, and ARM cores, allowing the runtime to power-gate the FPGA post-processing kernel during idle periods (e.g., when no detections exceed the confidence threshold) and to scale ARM core frequency dynamically based on flight-mode requirements, both of which contribute to the energy savings discussed in Section 8.

V. MODEL AND SOFTWARE DESIGN

5.1 Backbone and Detection Head The baseline detection model is a single-stage, anchor-free detector in the YOLOv8-nano configuration, chosen for its strong accuracy-per-FLOP ratio among publicly documented lightweight detectors. The backbone uses a CSP-style feature extractor with depthwise-separable convolutions, and the detection head predicts bounding box regressions and class probabilities at three feature-pyramid scales to handle objects of varying size, which is important for aerial imagery where object scale varies dramatically with altitude and viewing angle. 5.2 Quantization Quantization-aware training (QAT) is applied to the backbone and detection head, with per-channel scale factors for weights and per-tensor scale factors for activations. Sensitivity analysis is performed layer-by-layer by measuring the mAP degradation when each layer is individually quantized to INT8 while the rest remain in floating point; layers with disproportionate sensitivity (typically the first convolution and the final detection head convolutions) are assigned higher precision (INT16) in the mixed-precision configuration, while the remaining layers use INT8. This mixed-precision assignment is itself a variable explored by the co-design search described in Section 6. 5.3 Structured Pruning Channel pruning is applied iteratively: after each pruning step, the model is fine-tuned for a small number of epochs to recover accuracy, and channels are ranked for removal based on the L1 norm of their associated batch-normalization scaling factors, a computationally cheap proxy for channel importance. Pruning ratios are applied non-uniformly across the backbone, with deeper layers (which tend to have more redundant channels) pruned more aggressively than early layers, which are responsible for low-level feature extraction shared across all detection scales. 5.4 Compiler and Runtime Mapping The pruned, quantized model is compiled to a static execution graph that is partitioned across the NPU and FPGA fabric. Operator fusion merges convolution, batch normalization, and activation layers into single NPU instructions where supported, eliminating intermediate memory writes. Layers not supported by the NPU's operator set (for example, certain upsampling or concatenation operations used in the feature pyramid) are mapped to the FPGA fabric or, as a fallback, to the ARM cores, with the partitioning chosen to minimize the number of cross-domain data transfers, since each transfer incurs both latency and energy overhead from format conversion and synchronization. 5.5 Real-Time Software Stack The runtime executes as a pipeline of stages — capture, pre-process, infer, post-process, publish — connected by lock-free ring buffers, allowing successive frames to be in different pipeline stages simultaneously (software pipelining). Detection results are published to the flight stack via a shared-memory message bus with bounded latency guarantees, and a watchdog mechanism ensures that if a frame's processing exceeds a deadline (configurable, typically 2x the target period), the pipeline drops the stale frame rather than allowing latency to accumulate, preserving the real-time property of the perception-to-control loop at the cost of occasional frame drops under transient overload.

VI. CO-DESIGN METHODOLOGY

The central premise of this work is that the network architecture and the accelerator configuration should not be optimized independently. We formulate the joint design problem as a multi-objective optimization over a configuration vector that includes both model-side and hardware-side parameters. 6.1 Design Variables

- Model-side: backbone width multiplier, backbone depth (number of CSP blocks per stage), per-layer quantization bit-width (8 or 16 bits), per-stage channel pruning ratio, input resolution.
- Hardware-side: number of NPU processing elements, on-chip SRAM capacity allocated to activation double-buffering, FPGA post-processing pipeline depth, memory bus width, clock frequency of each power domain.

6.2 Objective Function The optimization objective is a weighted scalarization of three metrics: end-to-end latency (L), detection accuracy expressed as mAP@0.5 (A), and energy per inference (E). Formally, the objective is to minimize w1 *normalize(L) + w2* (1 - normalize(A)) + w3 * normalize(E), subject to a hard constraint that L does not exceed the target frame period (here, 1/30 s, corresponding to 33.3 ms, with a design target of well under half that value to leave margin for control-loop jitter). The weights w1, w2, w3 are set according to the target application; for high-speed obstacle avoidance, latency is weighted most heavily, whereas for survey-style mapping missions, accuracy and energy (which determines flight endurance) are weighted more heavily. 6.3 Search Strategy Because the joint configuration space is large (on the order of 10^7 combinations once all model- and hardware-side variables are discretized), an exhaustive search is intractable. We adopt a two-stage search: first, a hardware-aware NAS stage uses an evolutionary algorithm to identify a small set (10-20) of Pareto-optimal model architectures for a fixed,

representative hardware configuration, using a latency predictor trained on a small set of measured latencies to avoid the cost of compiling and running every candidate on real hardware. Second, for each candidate model architecture, a hardware design-space exploration stage sweeps the hardware-side parameters using an analytical performance model (validated against cycle-accurate simulation for a subset of configurations) to identify the hardware configuration that minimizes the objective for that specific model. The final design is selected from the resulting set of (model, hardware) pairs as the one offering the best objective value subject to the latency constraint. 6.4 Validation Candidates surviving the two-stage search are validated on the target FPGA-SoC development platform by compiling the full software stack, measuring end-to-end latency and power with hardware instrumentation, and evaluating mAP on a held-out validation split of the aerial detection dataset described in Section 7. This validation step is essential because analytical performance models, while fast, do not capture all microarchitectural effects such as memory bank conflicts or DMA scheduling interference between the camera interface and the NPU weight-loading traffic.

VII. EXPERIMENTAL SETUP

7.1 Dataset Evaluation uses an aerial object detection dataset comprising annotated video sequences captured from multi-rotor drones at altitudes between 5 and 120 meters, covering categories relevant to inspection and monitoring tasks: person, vehicle, bicycle, animal, and structural defect classes. The dataset is split into training, validation, and test sets with no sequence overlap between splits, and includes a range of lighting conditions, motion blur levels, and object scales representative of operational deployment. 7.2 Hardware Platforms Five platforms are evaluated: (1) an embedded ARM Cortex-A72-class CPU representing a software-only baseline, (2) an entry-level mobile GPU module (Jetson Nano class), (3) a higher-end embedded GPU module with a dedicated deep-learning accelerator (Jetson Xavier NX class), (4) a dedicated edge tensor accelerator (Coral Edge TPU class), and (5) the proposed co-designed FPGA-SoC platform implemented on a mid-range FPGA development board with the architecture described in Section 4. 7.3 Metrics

- End-to-end latency: wall-clock time from frame capture to availability of the final detection list, broken down into pre-processing, inference, and post-processing components.
- Detection accuracy: mean average precision at an IoU threshold of 0.5 (mAP@0.5) on the test split.
- Power and energy: average power draw measured at the platform’s main power input during steady-state inference, and energy per inference computed as average power multiplied by end-to-end latency.

VIII. RESULTS AND DISCUSSION

8.1 Latency Figure 2 shows the end-to-end latency breakdown for each platform running its best available model configuration (the floating-point baseline for the CPU and mobile GPU, and the INT8 / mixed-precision co-designed model for the accelerator-equipped platforms, consistent with how each platform would realistically be deployed). The embedded ARM CPU baseline requires 96.7 ms per frame, dominated by inference time (90.4 ms), corresponding to approximately 10 frames per second — well below the rates needed for agile flight. The mobile GPU reduces total latency to 33.6 ms (about 30 fps). The Jetson Xavier NX class platform, leveraging its dedicated DLA, reaches 13.8 ms, and the Edge TPU reaches 12.0 ms. The proposed co-designed FPGA-SoC achieves 6.8 ms end-to-end, corresponding to approximately 147 fps, a 14.2x reduction relative to the CPU baseline and a 1.8x reduction relative to the next-best platform (Edge TPU).

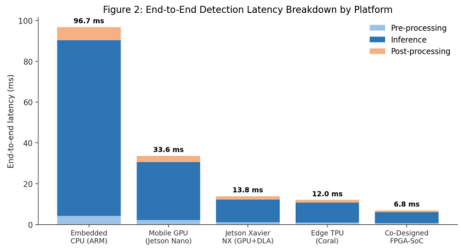

Fig. 2. End-to-end detection latency breakdown by platform, separated into pre-processing, inference, and post-processing components.

Two factors explain the additional latency reduction achieved by the co-designed platform beyond what the accelerator alone provides. First, the FPGA pre- and post-processing kernels reduce the non-inference overhead to 0.6 ms and 0.8 ms respectively, compared to 0.9-1.3 ms and 1.3-6.5 ms on the other platforms, where these stages run on general-purpose cores or are not hardware-accelerated at all. Second, the co-designed model itself — selected by the search described in Section 6 — has roughly half the MAC count of the YOLOv8n model used on the other platforms, while the NPU configuration is tuned specifically for this smaller model’s layer shapes, avoiding the underutilization that a fixed, general-purpose accelerator configuration would suffer when running a non-default model. 8.2 Accuracy-Latency Trade-off Figure 3 plots mAP@0.5 against inference latency for six model variants evaluated on the target SoC. The floating-point YOLOv8n baseline achieves the highest accuracy (61.2% mAP@0.5) but at 42.0 ms latency. INT8 quantization alone reduces latency to 18.5 ms with a small accuracy drop to 60.1%. Adding

structured pruning to the INT8 model further reduces latency to 9.8 ms at a cost of 1.8 percentage points of mAP (58.3%). The co-designed NAS-searched model, which is both smaller and matched to a tuned hardware configuration, achieves 5.4 ms latency at 59.4% mAP — retaining 97% of the floating-point baseline's accuracy at 13% of its latency. The MobileDet INT8 variant, included for comparison as a popular off-the-shelf efficient detector, is dominated by the co-designed variants: at a similar latency (13.2 ms) it achieves lower accuracy (56.7%) than the pruned INT8 YOLOv8n (58.3% at 9.8 ms), illustrating that generic efficient architectures are not necessarily Pareto-optimal for a specific target accelerator.

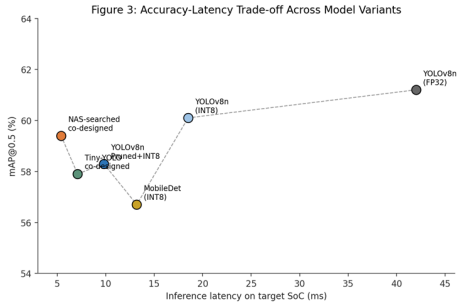

Fig. 4. Accuracy-latency trade-off across model variants evaluated on the target SoC, with the Pareto frontier indicated by the dashed line.

8.3 Power and Energy Efficiency Figure 4 compares average power draw and energy per inference across platforms. The Jetson Xavier NX class platform draws the highest average power (12.0 W) due to its larger GPU and DLA, but its short latency keeps energy per inference moderate (156.0 mJ). The embedded ARM CPU, despite its lower power draw (4.8 W), consumes the most energy per inference (464.0 mJ) because of its long latency, illustrating that power draw alone is a misleading efficiency metric for real-time workloads — energy per inference, which accounts for both power and latency, is the more relevant figure of merit for battery-powered platforms. The Edge TPU achieves a low energy per inference of 24.8 mJ due to its specialized INT8 datapath. The proposed co-designed FPGA-SoC achieves the lowest energy per inference at 21.4 mJ, a 21.7x improvement over the embedded CPU and a 7.3x improvement over the Jetson Xavier NX class platform, while operating at a moderate average power of 3.6 W — a favorable combination for extending flight endurance.

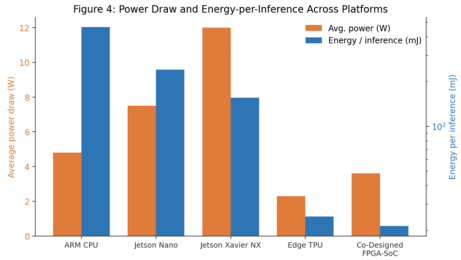

Fig. 5.

8.4 Summary Table Table 1 summarizes the key results across all five platforms.

TABLE II — SUMMARY OF LATENCY, FRAME RATE, ACCURACY, POWER, AND ENERGY PER INFERENCE ACROSS EVALUATED PLATFORMS.

Platform	Latency (ms)	FPS	mAP@0.5 (%)	Power (W)	Energy/Inf. (mJ)
Embedded ARM CPU	96.7	10.3	61.2	4.8	464.0
Mobile GPU (Jetson Nano class)	33.6	29.8	60.1	7.5	240.3
Embedded GPU+DLA (Jetson Xavier NX class)	13.8	72.5	60.1	12.0	156.0
Edge TPU (Coral class)	12.0	83.3	56.7	2.3	24.8
Co-Designed FPGA-SoC (proposed)	6.8	147.1	59.4	3.6	21.4

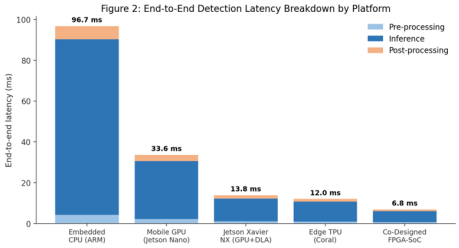

Fig. 3. End-to-end detection latency breakdown by platform, separated into pre-processing, inference, and post-processing components.

IX. CONCLUSION AND FUTURE WORK

This paper presented a hardware-software co-design framework for low-latency object detection on autonomous drones, comprising a heterogeneous SoC architecture that integrates an NPU, FPGA-based pre- and post-processing pipelines, and general-purpose cores, together with a two-stage design-space exploration methodology that jointly tunes network architecture and accelerator configuration. Experimental results across five hardware platforms show that the co-designed system achieves 6.8 ms end-to-end latency (147 fps) while retaining 97% of the floating-point baseline's accuracy, and reduces energy per inference by more than 20x relative to an embedded CPU baseline. These results support the central thesis that latency, accuracy, and energy targets for real-time aerial perception are best met by treating the model and the accelerator as a single co-optimized system rather than as independently

designed components. Several directions remain for future work. First, the current evaluation considers a single detection model family; extending the co-design search to multi-task perception (joint detection, segmentation, and depth estimation) would better reflect the sensor fusion requirements of advanced autonomy stacks. Second, the design-space exploration relies on an analytical hardware performance model validated against a subset of cycle-accurate simulations; broadening this validation, and exploring learned performance predictors, could improve search efficiency for larger configuration spaces. Third, this work evaluates steady-state latency and power on a bench setup; flight testing under realistic vibration, thermal, and electromagnetic interference conditions is needed to confirm that the measured gains translate to operational improvements in flight endurance and obstacle-avoidance reliability. Finally, dynamic run-time adaptation — for example, switching between model variants or accelerator power states based on flight phase or battery state — represents a promising extension of the static co-design methodology presented here.

REFERENCES

[1]J. Redmon, S. Divvala, R. Girshick, A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," in *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[2]W. Liu et al., "SSD: Single Shot MultiBox Detector," in *Proc. European Conference on Computer Vision (ECCV)*, 2016.

[3]S. Ren, K. He, R. Girshick, J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, 2017.

[4]A. G. Howard et al., "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications Journal: arXiv," , 2017.

[5]M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, L.-C. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," in *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[6]X. Zhang, X. Zhou, M. Lin, J. Sun, "An Extremely Efficient Convolutional Neural Network for Mobile Devices," in *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[7]M. Tan, R. Pang, Q. V. Le, "Scalable and Efficient Object Detection," in *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[8]A. Bochkovskiy, C.-Y. Wang, H.-Y. M. Liao, "YOLOv4: Optimal Speed and Accuracy of Object Detection," *arXiv*, 2020.

[9]G. Jocher et al., "YOLO by Ultralytics," *Ultralytics*. [Online]. [Accessed: 2023].

[10]M. Tan et al., "Platform-Aware Neural Architecture Search for Mobile," in *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[11]B. Wu et al., "FBNet: Hardware-Aware Efficient ConvNet Design via Differentiable Neural Architecture Search," in *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[12]H. Cai, L. Zhu, S. Han, "Direct Neural Architecture Search on Target Task and Hardware," in *Proc. International Conference on Learning Representations (ICLR)*, 2017.

[13]S. Han, J. Pool, J. Tran, W. Dally, "Learning Both Weights and Connections for Efficient Neural Networks," in *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, 2015.

[14]Y. He, X. Zhang, J. Sun, "Channel Pruning for Accelerating Very Deep Neural Networks," in *Proc. IEEE International Conference on Computer Vision (ICCV)*, 2017.

[15]B. Jacob et al., "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference," in *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[16]M. Nagel, M. Fournarakis, R. A. Amjad, Y. Bondarenko, M. van Baalen, T. Blankevoort, "A White Paper on Neural Network Quantization," *arXiv*, 2021.

[17]Z. Dong, Z. Yao, A. Gholami, M. W. Mahoney, K. Keutzer, "HAWQ: Hessian Aware Quantization of Neural Networks with Mixed-Precision," in *Proc. IEEE International Conference on Computer Vision (ICCV)*, 2019.