Sky to Edge-Cloud: Heterogeneous Computation Offloading for Energy-Efficient Drone Computing

Zhehang Zhang*
Electrical and Computer Engineering
University of Massachusetts Amherst
Amherst, MA, USA
zhehangzhang@umass.edu

Bharadwaj Madabhushi*
Electrical and Computer Engineering
University of Massachusetts Amherst
Amherst, MA, USA
bmadabhushi@umass.edu

Sandip Kundu
Electrical and Computer Engineering
University of Massachusetts Amherst
Amherst, MA, USA
kundu@umass.edu

Russell Tessier
Electrical and Computer Engineering
University of Massachusetts Amherst
Amherst, MA, USA
tessier@umass.edu

Abstract

Offloading computation from edge devices to an edge cloud can save energy and enhance performance, but managing workloads from heterogeneous devices such as CPUs, GPUs, and NPUs remains challenging. This paper presents a prototype heterogeneous offloading system using an FPGA-based edge cloud capable of handling diverse computation models. Using drone computing as a use-case, we demonstrate that applications such as depth estimation, object detection, and Simultaneous Localization and Mapping (SLAM) can efficiently offload CPU, GPU, and DSP tasks to a reconfigurable accelerator. By intelligently mapping heterogeneous kernels to the reconfigurable FPGA edge cloud, our system reduces drone energy by up to 90%, underscoring its *heterogeneous kernel offloading* capability and real-time performance and energy gains.

CCS Concepts

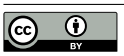
• **Computer systems organization → Cloud computing**; **Reconfigurable computing**; **Heterogeneous (hybrid) systems**; **Distributed architectures**.

Keywords

Autonomous drones, heterogeneous edge computing, FPGA offloading, kernel mapping, energy efficiency

ACM Reference Format:
Zhehang Zhang, Bharadwaj Madabhushi, Sandip Kundu, and Russell Tessier. 2026. Sky to Edge-Cloud: Heterogeneous Computation Offloading for Energy-Efficient Drone Computing. In *Great Lakes Symposium on VLSI 2026 (GLSVLSI '26), June 22–24, 2026, Canandaigua, NY, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3787109.3815304

*Both contributed equally to this work.



GLSVLSI '26, Canandaigua, NY, USA

ACM ISBN 979-8-4007-2431-2/26/06
https://doi.org/10.1145/3787109.3815304

1 Introduction

The proliferation of artificial intelligence (AI)-enabled devices at the edge has significantly improved various aspects of daily life. Edge devices, particularly in sensor applications, are becoming increasingly diverse, incorporating components such as CPUs, GPUs, and neural processing units (NPUs) to meet the computational needs of edge data processing. These devices are often deployed in mobile or field environments, where they must operate with limited power while performing real-time operations. Recently, the emergence of edge computing has enabled *efficient computational* offloading from edge devices to nearby edge clouds [42]. Such computational offloading allows edge devices to operate at low power, conserving energy while harnessing the edge cloud's high-performance processing capabilities to deliver real-time performance.

While homogeneous edge clouds offer several advantages, including: (*i*) *simplified management*, reducing the need for specialized resource allocation and complex load balancing, (*ii*) *cost efficiency*, with standardized hardware reducing maintenance, and support costs, (*iii*) *easier scalability*, with straightforward capacity expansion and (iv) *superior efficiency*, where any node can perform any task, they are ill-suited for executing heterogeneous workloads featuring diverse computational models. This paper explores an innovative approach to achieving the benefits of homogeneous edge clouds while supporting heterogeneous computational requirements. We propose an FPGA-based edge cloud, where the FPGA fabric can be dynamically reconfigured to accommodate diverse computational models and workloads. A reconfigurable edge cloud enables *edge devices to evolve and adapt their architectures over time*, while the edge cloud itself remains adaptable and relevant, thereby increasing its longevity and reducing overall costs. In this work, we develop and deploy a suite of methods to efficiently map diverse computational models onto FPGAs in edge cloud nodes. Diverse computation from CPUs, GPUs, and NPUs is dynamically offloaded from an edge device to our FPGA-based edge cloud platform.

We prototype the heterogeneous edge device workload using a Qualcomm Innovators Development Kit (QIDK) [37] that is equipped with a CPU, GPU and NPU. Edge cloud nodes are modeled using an AMD ZCU102 [7] platform. Performance on the computing components on the QIDK and UltraScale+ programmable logic on the ZCU102 board are characterized for application kernels and edge

device energy savings are quantified. The main contributions of this paper are:

- A homogeneous, dynamically reconfigurable FPGA edge-cloud that transparently offloads heterogeneous kernels from CPU/GPU/NPU workloads while preserving operational simplicity, cost efficiency, and scalable deployment.
- An offloading framework with efficient mapping methodologies that translate diverse edge-device computations onto FPGA-based edge-cloud nodes, validated on Qualcomm QIDK and AMD ZCU102.
- End-to-end evaluation of heterogeneous computation offload, demonstrating considerable performance and energy gains from (i) direct kernel offloading and (ii) avoiding costly NPU model reloads through selective offload.

2 Background

The advent of 5G and low-latency communication has enabled high-speed, seamless data transfer between edge devices and infrastructure. Offloading compute-intensive tasks to the edge cloud not only conserves resources on devices such as drones but also enables complex computations such as machine learning to be performed in real-time [23]. Edge computing extends the cloud to the edge of the network, closer to edge devices. By processing data closer to the source, the edge cloud reduces latency and enables real-time applications that were previously not possible. Edge cloud nodes service requests from edge devices which are diverse in their computational models. Examples of such offloading include volumetric streaming [29], simultaneous localization and mapping (SLAM) [8], and federated learning [12]. To accommodate the diversity of edge workloads, specialized edge cloud nodes mirroring the edge devices have been proposed [26]. However, this approach is not scalable, highlighting the need for more flexible and adaptable edge cloud solutions.

The offloading of computation from edge devices to the edge cloud has been widely studied [2, 14, 15, 40, 42]. Offloading can significantly extend the battery life of edge devices while providing robust computing capabilities for mobile edge devices, such as drones [17]. *The previous studies focus on task partitioning and offloading strategies based on CPU demand, network demand and latency requirement, but they do not consider heterogeneity of edge devices.* A diverse array of heterogeneous edge devices is currently being deployed, each offering distinct computational capabilities. NVIDIA introduced the Jetson Xavier NX [24, 38], while AMD provides a range of edge-oriented solutions, including the AMD EPYC Embedded 3000 Series Processors [21, 32]. Intel's portfolio includes the Movidius Vision Processing Units (VPUs) [18, 45], whereas Qualcomm offers AI accelerator co-processors, such as the DM.2e [22]. These devices differ significantly in their computational architectures, making task offloading from edge devices to edge clouds a complex challenge. *Existing research has not fully explored the intricacies of offloading strategies in heterogeneous edge environments.*

We examine the benefits and challenges of computational offloading from edge devices to the edge cloud using autonomous drones as a case study. Autonomous drones rely heavily on edge devices for sensing and actuation [1, 33]. These edge devices often execute specialized algorithms for real-time data compression, decompression, and intelligence preprocessing/postprocessing. Further, they must operate with minimal latency and energy consumption. To meet these constraints, edge devices used in drones may employ diverse computational models, necessitating that the edge cloud efficiently handle offloaded functions across heterogeneous architectures.

FPGA-based edge clouds have significant potential to support the offloading of varied computational workloads. While FPGA clouds have been explored in prior research for specific applications [39], they have mostly been used as co-processing accelerators. Liu et al. [28] and Biookaghazadeh et al. [9] describe the implementation of GPU operations on edge cloud FPGAs, although offloading is not considered. Xu et al. [44] examines the offloading of computer vision applications executed on CPUs to an edge cloud FPGA. Although these approaches highlight the promise of edge cloud FPGAs, *their role in facilitating heterogeneous computational offloading remains largely unexplored.* An effort towards this goal [47] introduces an offload management framework for heterogeneous edge devices.

By abstracting hardware complexities, FPGA application deployment enhances software programmability, significantly improving flexibility and accessibility. In our use case, energy efficiency and performance improvements versus edge devices are considered. Computations are offloaded to the edge cloud whenever the energy required for computing exceeds the energy required for communication that is related to offloading and at least a baseline of performance is maintained. In most cases performance *improvement* versus edge device execution is also achieved. The decision to offload can be made using application performance and energy characterization prior to deployment. This offloading approach provides an effective energy-saving mechanism [46].

3 System Design

3.1 Overview

Drones have limited energy and onboard computation resources. Offloading tasks to the edge cloud reduces onboard processing, thus conserving power for extended flight operations. The edge cloud in our system provides CPU and programmable logic (PL) computational support. Soft deep learning processing unit (DPU) intellectual property (IP) cores, instantiated in the PL, enable efficient execution of machine learning models. The cloud infrastructure dynamically selects and processes machine learning (ML) tasks. Non-ML-based image processing applications can be implemented using customized IP cores that are compiled to bitstreams. The DPU and other core bitstreams can be swapped into the FPGA on demand.

3.2 Metrics

Edge device and edge cloud performance and energy metrics determined via profiling enable the system to dynamically select the application execution environment. These metrics include *processing time*, the time required to execute a predefined model, and *communication time*, the time taken to transmit information from the edge device to the edge cloud and receive the results back.

Communication time is defined as:

$$T_{comm} = T_{\text{transmit}} + T_{\text{receive}} \quad (1)$$

in which T_{transmit} is the time required to send data from the edge device to the edge cloud and T_{receive} is the time required to transmit the processed results back to the edge device. *Round trip delay (RTD)* is the total time required to complete the entire computation, including both communication and execution on the edge cloud, is given by:

$$RTD = T_{comm} + T_{cloud} \quad (2)$$

in which T_{cloud} is the time taken to perform computation on the edge cloud. Communication time accounts only for the transmission delays, while RTD includes both transmission and edge cloud's processing time, providing a complete measure of the computation duration.

Processing energy, E_{proc}, is the average energy consumed by the edge device to perform computation locally on the edge device. This energy can largely be saved by offloading computation to the edge cloud. *Round-trip energy consumption* is the total energy consumed on the *edge device* (when offloaded) for task completion:

$$E_{\text{comm}} = E_{\text{transmit}} + E_{\text{receive}} \quad (3)$$

in which E_{transmit} is the energy consumed by the edge device while transmitting an image to the edge cloud and E_{receive} is the energy consumed by the edge device while receiving results from the edge cloud. Since energy availability on the edge cloud is not considered a limiting factor, round-trip energy effectively is limited to edge device communication energy. Our primary focus is on the energy consumed by the edge device for data transmission and reception. Edge device energy savings by offloading can be defined as:

$$E_{\text{savings}} = E_{\text{proc}} - E_{\text{comm}} \quad (4)$$

Throughput is used to measure processing performance for both local edge device execution and for computation which involves offloading to the edge cloud.

3.3 Drone Execution Scenarios

In this work, we examine the offloading of multiple kernels that form the core of typical drone applications. Drone operation broadly involves two classes of computation: localization and mapping (SLAM) and object analysis [34]. While these operations may execute sequentially in structured missions (*e.g.,* navigating toward a destination followed by object analysis), they can also operate concurrently or in alternating fashion depending on mission demands, environmental dynamics, and available resources.

SLAM requires feature analysis during motion. These localization and mapping operations form an extensive part of the SLAM workload, consuming up to 70% of execution time [30]. SLAM typically operates directly on raw image frames and does not require input preprocessing, such as image resizing, format conversion, or normalization, nor output postprocessing. In Section 4.1, two implementations of localization and mapping kernels (*Lucas-Kanade* [19] and *FAST + BRIEF + Matching* [20]) are deployed on drone and edge cloud platforms allowing for the exploration of offloading.

Drone-based object analysis typically involves both object detection and depth estimation to identify objects and determine an appropriate response. Algorithms for these operations are often implemented as machine learning kernels that may execute sequentially, in parallel, or triggered reactively based on mission context [11]. In Section 3.4, we describe why the use of ML kernels motivates offloading regardless of their execution ordering. Both object detection and depth estimation require pre- and postprocessing. Preprocessing typically involves image resizing and pixel format conversion. Postprocessing involves results filtering.

3.4 Offloading Decisions

CPUs, GPUs, and NPUs have differing strengths in performing computation. CPUs are best suited to sequential applications while GPUs are ideal for fine-grained computation with wide parallelism. NPUs are generally optimized for the high-bandwidth multiply-accumulate operations typically performed by machine learning algorithms. We consider two specific offloading cases for our computation kernels:

Case 1: Kernel achieves energy and performance benefits via offloading: Often, there are performance and energy benefits to offloading kernels to the edge cloud whenever it is available. The FPGA in the edge cloud provides significant kernel parallelism and specialization, and the communication time and energy of the edge device are smaller than executing the kernel locally on the edge device. In this case, the edge device collects availability information from the edge cloud and assesses expected throughput and round-trip energy consumption. If there is a net energy benefit while at least maintaining a performance baseline, the system transfers the computation, including input data, to the edge cloud for processing. The software required for computation is available on the edge cloud and does not need to be transferred.

Case 2: Competing kernels in an application: In some cases, individual kernel performance and energy savings versus local execution cannot be obtained by offloading computation to the edge cloud. However, an application may require the execution of multiple kernels (e.g., in sequence) that are best suited to a specific unit (e.g., an NPU). Before an NPU can execute a kernel, it must load a model, a time-consuming exercise. If the model-loading overhead is considered, it can be energy and performance effective to offload the computation for the second NPU kernel rather than loading a new model into the NPU for every image frame. In this situation, offloading is also more efficient than executing the second kernel on the CPU or GPU.

4 Experimental Setup

Our experimental setup includes a QIDK with a Snapdragon Gen2 system-on-chip (SoC), commonly used in drones as the edge device, and an AMD ZCU102 FPGA card as the edge cloud node. An overview of this setup is shown in Figure 1.

4.1 Application Kernels and Datasets

Our selected kernels are widely used in drone applications. *YOLOv5s6* [25, 41] is a real-time object detection kernel that is frequently used in resource-constrained platforms. *Monodepth2* [16] is a single-camera depth estimator with consistent performance, making it suitable for lightweight drones without stereo hardware. The *Lucas-Kanade* method is an optical flow algorithm for motion tracking. The *FAST + BRIEF + Matching* algorithm provides feature extraction,

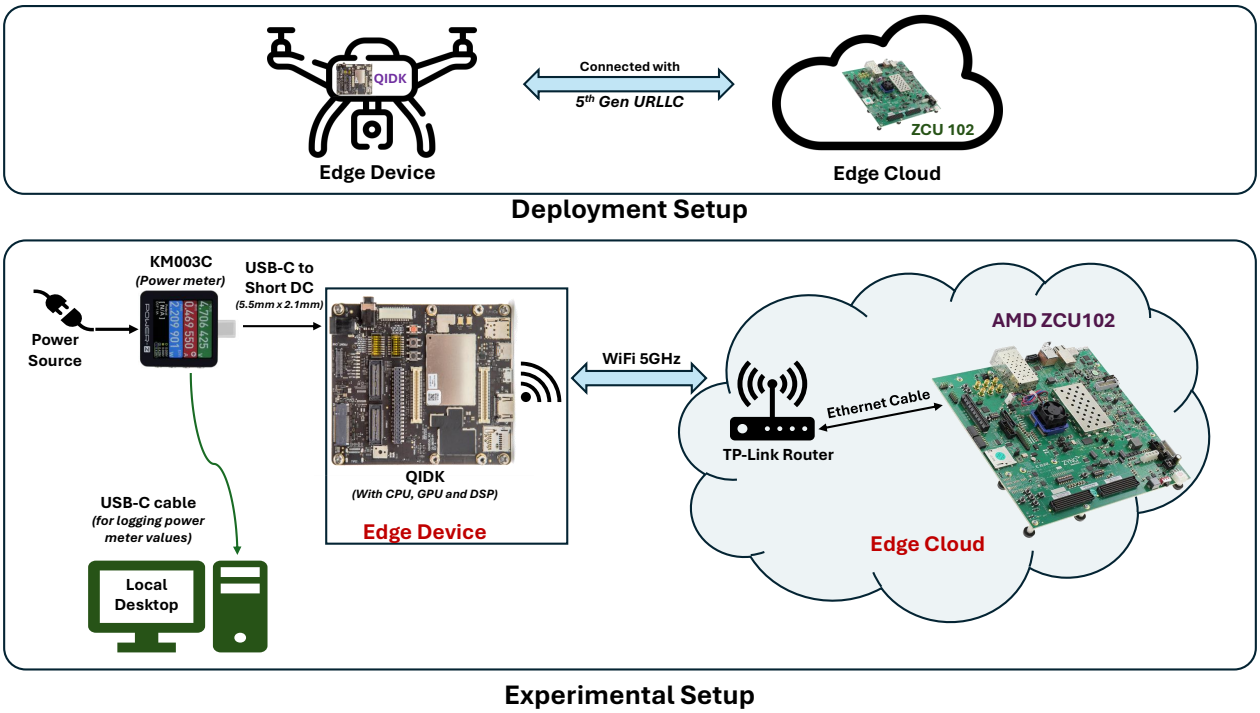


Figure 1: Experimental setup with QIDK as edge device and ZCU 102 as edge cloud connected via WiFi

description, and matching with low latency and limited memory requirements. These kernels represent heterogeneous workloads that can be mapped to CPU, GPU, NPU, and FPGA platforms.

To evaluate the performance of our edge cloud offloading system across the selected categories, we use benchmark datasets commonly used in the research community. For object detection and depth estimation, images from the COCO test set [27] were used. This sampling approach ensures a diverse distribution of real-world conditions, enabling reliable and meaningful hardware performance measurements. For Lucas-Kanade optical flow and FAST + BRIEF + Matching, images from the MPI-Sintel dataset [10] were used.

4.2 Edge Device

The QIDK includes a Snapdragon Gen2 platform with an eight-core Kyro CPU running an Android Tiramisu OS. The Snapdragon includes an Adreno 740 GPU [35]. The Qualcomm Snapdragon Neural Processing SDK (SNPE) uses this GPU path to accelerate deep learning models when the required operators and INT8 precision are supported. The Hexagon NPU [36] combines vector extensions with INT8 and FP16 inference processing.

4.2.1 Software Setup and Energy Profiler. Our QIDK software setup allows for application development and runtime monitoring of computation time and energy consumption without affecting system performance. Our edge device uses Termux [43], a Linux terminal emulator for Android.

For the QIDK, we use the ChargerLAB KM003C [13] to measure system-level power during local execution. The USB-C supply for the QIDK is connected through the KM003C, allowing continuous measurement of voltage, current, and instantaneous power. Measurement data is streamed to an external computer through the serial interface, sampled at 1 kHz to capture rapid changes during model execution. Baseline power is recorded with no applications running, and all kernel and communication power values are computed by subtracting this baseline.

Table 1: Resource utilization, load time, and size for FPGA bitstreams for the AMD XCZU9EG on the ZCU102. Percent values indicate utilization of the FPGA logic.

Resource	DPU		Lucas-Kanade		FAST-BRIEF	
		%		%		%
LUTs	51,115	18.6	24,477	8.9	97,068	35.4
FFs	97,855	17.9	33,979	6.2	78,796	14.4
BRAM	255	28.0	180	19.7	487	53.4
DSPs	710	28.2	78	3.1	139	5.5
Load time (ms)	354	–	195	–	321	–
Bitstream (MB)	18	–	13	–	17	–

4.3 Edge Cloud

4.3.1 Hardware and Software Setup. The AMD Zynq UltraScale+ MPSoC ZCU102 Evaluation Kit integrates a quad-core ARM Cortex-A53 processor which runs PetaLinux [6] at 1.2 GHz. The ZCU102 also includes a programmable logic (PL) fabric with 274,080 lookup

tables (LUTs). In our experimentation, the PL is configured with several different FPGA bitstreams, including a one-core DPU (for Yolov5s6 and Monodepth2) which operates at 330 MHz and other customized circuits synthesized by Vitis v2022.2 (for Lucas-Kanade and FAST + BRIEF + Matching). As shown in Table 1, these cores occupy a modest portion of the available FPGA resources.

4.3.2 FPGA Configuration. The PetaLinux OS leverages the PL to accelerate neural network inference by deploying the DPUCZDX8G B4096 soft IP core [4]. The DPU supports 8-way pixel and 16-way input/output channel parallelism, achieves up to 4,096 operations per clock cycle and implements compute-intensive convolution, activation, and pooling layers [5]. A53 processor cores manage orchestration, including memory transfers, task scheduling, and lightweight pre- and postprocessing. Due to the DPU's limited on-chip memory (4.8 MB of BRAM and URAM) [3], model execution follows a layer-by-layer streaming strategy, where the A53 incrementally loads each layer from DRAM.

Using our streaming model, inference for Yolov5s6 and Monodepth2 begins immediately as new images arrive. The A53 performs image *preprocessing* such as resizing, normalization, and format conversion on received images using OpenCV, and stores the processed data in DRAM for access by the DPU. During *inference*, the DPU fetches this data and executes the model sequentially, with each layer being partially loaded by the A53 into the DPU's limited on-chip memory (BRAM/URAM). Once inference is complete, the DPU writes the output logits to DRAM. These *postprocessed* values include pixel coordinates for YOLOv5s6 and 3D depth information for Monodepth2.

Custom bitstreams for Lucas–Kanade and FAST + BRIEF + Matching feature extraction and matching are also loaded into DRAM at boot. For Lucas–Kanade, a radius of 10 pixels and 10 iterations is used. For FAST + BRIEF + Matching, we track 800 keypoints, which aligns with SLAM settings that operate with several hundred to about one thousand features per frame [31]. For both Lucas-Kanade and FAST + BRIEF + Matching kernels, the bitstream is loaded into the FPGA and data is forwarded to the FPGA from the A53 via the AXI bus. After processing, data is collected by the A53 and returned to the edge device.

4.4 Communication Traffic Analysis

We assess communication performance using communication time for 160 kB COCO and 436 kB MPI-Sintel images and round-trip energy consumption on the QIDK. Communication traffic experiments were conducted using 5 GHz band (IEEE 802.11ac) communication. The average round-trip delay (RTD) is determined based on Equation 2. Communication energy consumption for the QIDK is measured using the KM003C. The energy consumed for image data transmission ($E_{transmit}$) and reception ($E_{receive}$) are calculated separately. The total round-trip energy (RTE) is then computed by substituting the transmission and reception energy values into Equation 3.

5 Results

In this section, we compare the throughput for kernels implemented on both the edge device and offloaded to the edge cloud (Case 1). Edge device energy savings of offloading is also reported. Then, we discuss the possibility of offloading kernels to prevent model reloading on the edge device (Case 2).

5.1 Performance Metrics on Edge Device

Following the procedure outlined in Section 4.2, we measured the average processing time and average per-image energy consumption for each kernel deployed on the QIDK. These metrics were computed using the corresponding datasets described in Section 4.1. These experiments indicate which processing component on the edge device is best suited to each kernel.

The results in Table 2 list the kernels tested on the QIDK processing components: the Kyro CPU, Adreno GPU, and Hexagon NPU. The columns labeled T report the average execution time per image frame, while E captures the corresponding energy consumption. For ML applications, image pre- and postprocessing steps execute exclusively on the Kyro CPU and are reported separately in the caption to isolate inference costs; communication time is excluded throughout as all measurements reflect local on-device execution.

The data highlights that computation type strongly determines platform suitability, and no single processor is universally optimal. YOLOv5s6 and Monodepth2 perform best on the NPU, whose MAC-array architecture is purpose-built for dense tensor operations such as convolutions and matrix multiplications, enabling high parallelism and throughput. This sustained utilization translates to higher instantaneous power draw compared to non-ML workloads. Lucas-Kanade, dominated by dense pixel-level operations that map naturally to SIMD (Single Instruction Multiple Data) execution, achieves highest performance on the GPU. FAST + BRIEF + Matching contains a significant sequential and branching component — feature detection involves data-dependent control flow — making it better suited to the multi-core Kyro CPU.

Non-ML kernels draw lower instantaneous power across all platforms. These algorithms are control-flow-heavy, memory-bound, and less amenable to sustained parallel execution, resulting in lower processor utilization and reduced dynamic power consumption. This mismatch is most pronounced on the NPU: Lucas-Kanade takes over 120 seconds to execute at near-idle power, confirming that the NPU's fixed tensor dataflow circuits are largely bypassed and the workload falls back to a slow software path. In contrast, the same kernel completes in approximately 169 ms on the GPU, underscoring the importance of correct kernel-to-platform mapping.

Finally, ML kernels carry a one-time model loading overhead, handled by SNPE prior to first inference. This loading phase, reported separately in the table, involves parsing and compiling the model into a platform-specific representation and can be substantial — on the order of hundreds to over a thousand milliseconds. This motivates the selective offloading strategy discussed in Section 3.4, where avoiding repeated model reloads constitutes a key source of energy and latency savings.

5.2 Edge Cloud Performance Evaluation on ZCU102

To identify if edge cloud processing time is a limiting factor in round trip delay, we measured the average execution times for each step in the processing workflow for the four kernels, as described in Section 4.3, for images already stored on the edge cloud. Our evaluation considered both single-core DPU implementation for

Table 2: Model load and processing time/energy on Snapdragon. Average pre-processing time (energy) on the CPU for YOLOv5s6 is 16.42 ms (42.10 mJ). Average pre-processing time (energy) for Monodepth2 is 7.07 ms (31.78 mJ). Average post-processing time (energy) on the CPU for YOLOv5s6 is 28.65 ms (87.24 mJ). Average post-processing time (energy) for Monodepth2 is 1.71 ms (7.83 mJ).

Application	QIDK Execution											
	CPU				GPU				NPU			
	Model Load		Avg Proc		Model Load		Avg Proc		Model Load		Avg Proc	
	T (ms)	E (mJ)	T (ms)	E (mJ)	T (ms)	E (mJ)	T (ms)	E (mJ)	T (ms)	E (mJ)	T (ms)	E (mJ)
YOLOv5s6	159.25	1,165.59	792.77	5,802.62	1,177.49	6,359.37	151.61	818.83	100.48	491.87	24.01	117.52
Monodepth2	172.47	1,156.78	130.30	873.91	613.60	2,422.82	34.36	135.68	27.30	157.36	3.32	19.14
Lucas–Kanade	–	–	15,096.39	56,617.49	–	–	168.73	562.98	–	–	120,413.58	45,452.36
FAST + BRIEF	–	–	46.90	179.53	–	–	51.95	190.48	–	–	1,315.49	496.55

Table 3: Application performance on ZCU102

Category	Application	Avg Init T (ms)	Avg Pre-proc. T (ms)	Avg Proc. T (ms)	Avg Post-proc. T (ms)	Total T (ms)
Object Detect	YOLOv5s6	654.24	15.25	85.95	29.18	130.38
Depth Estimate	Monodepth2	422.72	15.42	26.15	1.75	43.32
Motion Track	Lucas-Kanade	–	–	89.90	–	89.90
Feature Extraction and Matching	FAST + BRIEF	–	–	32.70	–	32.70

Table 4: Communication Analysis

Dataset	Metric	Data Sent	Data Received	802.11ac 5GHz
COCO	Communication T (ms)	160kB per image	4kB	9.99
	$(T_{transmit}\ (ms) + T_{receive}\ (ms))$			9.58 + 0.41
	Round Trip E (mJ)			13.95
	$(E_{transmit}\ (mJ) + E_{receive}\ (mJ))$			13.32 + 0.63
MPI-Sintel	Communication T (ms)	436kB per image	4kB	12.12
	$(T_{transmit}\ (ms) + T_{receive}\ (ms))$			11.71 + 0.41
	Round Trip E (mJ)			16.91
	$(E_{transmit}\ (mJ) + E_{receive}\ (mJ))$			16.28 + 0.63

Table 5: Case 1: End-to-end comparison between QIDK on-device processing and edge-cloud offloading to FPGA for 3,030 frames. Communication costs are included.

Application	Platform	Total RTD (s)	Avg RTD (ms)	Speedup per image	Total RTE (J)	Avg RTE (mJ)	Savings (mJ) per image
Lucas–Kanade	QIDK GPU only	520.31	171.72	–	1,907.82	629.64	–
	QIDK CPU only	46,269.51	15,270.47	–	173,532.29	57,271.38	–
	FPGA (802.11ac)	302.95	99.98	1.72 ×	329.45	108.73	520.91
FAST + BRIEF	QIDK CPU only	160.82	53.08	–	633.92	209.21	–
	QIDK GPU only	193.92	64.01	–	711.05	234.67	–
	FPGA (802.11ac)	115.11	37.99	1.40×	125.46	41.40	167.81

YOLOv5s6 and Monodepth2 and synthesized custom RTL circuits for Lucas-Kanade and FAST and BRIEF. This analysis effectively represents a *best case* analysis since image communication time is not included. The results, summarized in Table 3, indicate the processing time taken at each execution. For the DPU, the model requires initialization by loading configuration information into the core (*Avg Init*). Initialization is only performed once.

For the ML kernels, the time taken by the A53 to initialize the `xmodel` file is substantial, particularly for YOLOv5s6. It can be noted that model loads are infrequent operations that can be amortized over many inference operations, a common case for drone usage. All kernel implementations are faster on the FPGA than their QIDK counterparts, except for ML kernel implementations on the NPU.

Table 6: Case 2: End-to-End comparison between QIDK on-device processing and edge cloud offloading to FPGA for YOLOv5s6 for 303 frames. Communication costs are included.

Application	Platform	Total RTD (s)	Avg RTD (ms)	Speedup per image	Total RTE (J)	Avg RTE (mJ)	Savings (mJ) per image
Monodepth2	QIDK (NPU+GPU)	66.45	219.31	–	560.98	1,851.41	–
+ YOLOv5s6	QIDK (NPU+NPU)	259.82	857.50	–	213.72	705.35	–
	FPGA (802.11ac)	40.26	132.86	1.65×	62.18	205.21	1,646.20

5.3 Communication Performance Analysis

Communication between the edge device and the edge cloud is a critical link that enables offloading of computation to the FPGA, allowing the system to benefit from its acceleration capabilities. Two key factors that influence the feasibility of offloading are communication latency and the energy required to transmit data. Following the procedure described in Section 4.4, we measured both the upload and download latencies in hardware using our 802.11ac setup. As shown in Table 4, the communication time for transmitting and receiving a 160KB image (436KB image) averages 9.99 ms (12.12 ms). Ignoring model load time and instantiation, this value is significantly lower than the average processing times of most of the kernels running on the edge device (Table 2) indicating a potential benefit of offloading.

5.4 Case 1 Analysis

In this subsection, we consider kernels that can achieve performance and energy benefits via offloading. In a set of experiments, we measured the throughput of the applications offloaded from the edge device to the edge cloud. Case 1 results, which include communication time, are shown in Table 5. The Lucas-Kanade and FAST+BRIEF+Matching kernels always obtain a performance and energy benefit through offloading to the edge cloud. The results in the table illustrate total and average end-to-end round trip delays (RTDs) for 3,030 image frames. The *QIDK* values indicate no offloading, which serves as a baseline. Even considering communication delays and energy, significant speedup and energy savings can be achieved versus the best-case QIDK processing component. Only QIDK energy is considered in these end-to-end experiments. YOLOv5s6 and Monodepth2 are not considered for offloading in this context due to their efficient performance and energy consumption on the QIDK NPU.

5.5 Case 2 Analysis

As noted in Section 3.4, an application may require the execution of multiple kernels that are best-suited to a specific component (e.g., the NPU). Since the NPU requires a model load, which increases delay and energy, if two kernels are present, performance and energy benefits can be obtained via offloading. Table 6 examines the results of offloading YOLOv5s6 for an application in which Monodepth2 and YOLOv5s6 are executed in sequence for each image. The *QIDK* rows do not include offloading, and model loading is repetitively performed for both kernels for the NPU-only case. The other QIDK row indicates that a GPU is used for Yolov5s6 instead of the NPU. Average and total performance and energy results are obtained over 303 image frames. Offloading YOLOv5s6 results in significant performance and energy benefits (over 1.6J per image versus the NPU+GPU implementation). This represents a 90% energy reduction versus the NPU+GPU case. RTD values include communication delays.

5.6 Discussion

Tables 5 and 6 shows the clear benefits of offloading diverse computation from drones to the edge cloud. For computational heavy, high-accuracy models such as Monodepth2 and YOLOv5s6, energy savings in processing are significant, even after accounting for communication energy. This illustrates the advantages of offloading diverse computation from edge devices, not only in terms of energy efficiency but also in terms of overall performance, making it an effective strategy for enhancing drone operations and real-time applications.

6 Conclusion and Future Work

This paper presents a solution for offloading heterogeneous edge workloads to an FPGA-based edge cloud. In a drone use case, offloading from a Qualcomm QIDK to an AMD ZCU102, we reduced onboard computation energy by about 90% while sustaining real-time throughput across diverse workloads. Future work includes smarter workload partitioning, integration with emerging AI accelerators, expansion beyond drones, and improved dynamic resource allocation and communication (*e.g.,* multi-access edge computing) to enhance performance and robustness, advancing energy-efficient, high-performance edge/cloud co-managed systems. Multiple drones with variable communication latencies to multi-node edge clouds will also be evaluated.

Acknowledgments

This research was funded by National Science Foundation grants CNS-2247059 and CNS-2402382.

References

[1] Muhammad Abrar, Ushna Ajmal, Ziyad M Almohaimeed, Xiang Gui, Rizwan Akram, and Roha Masroor. 2021. Energy efficient UAV-enabled mobile edge computing for IoT devices: A review. *IEEE Access* 9 (2021), 127779 – 127798.

[2] Jaber Almutairi and Mohammad Aldossary. 2021. A novel approach for IoT tasks offloading in edge-cloud environments. *Journal of Cloud Computing* 10, 1 (2021), 28.

[3] AMD. 2021. *UltraScale Architecture Memory Resources User Guide (UG573).* Available: https://0x04.net/~mwk/xidocs/ug/ug573-ultrascale-memory-resources.pdf.

[4] AMD. 2023. *DPUCZDX8G for Zynq UltraScale+ MPSoCs Product Guide (PG338).* Advanced Micro Devices. https://docs.amd.com/r/en-US/pg338-dpu/Introduction?tocId=3xsG16y_QFTWvAJKHbisEw

[5] AMD. 2025. DPU IP Details and System Integration. https://xilinx.github.io/Vitis-AI/3.5/html/docs/workflow-system-integration.html. [Accessed: 10-Mar-2025].

[6] AMD. 2025. PetaLinux Tools. https://www.amd.com/en/products/software/adaptive-socs-and-fpgas/embedded-software/petalinux-sdk.html. Accessed: 2025-03-02.

[7] AMD. 2025. ZCU102 Evaluation Kit. https://www.amd.com/en/products/adaptive-socs-and-fpgas/evaluation-boards/ek-u1-zcu102-g.html. Accessed November 2025.

[8] Ali J Ben Ali, Marziye Kouroshli, Sofiya Semenova, Zakieh Sadat Hashemifar, Steven Y Ko, and Karthik Dantu. 2022. Edge-SLAM: Edge-Assisted Visual Simultaneous Localization and Mapping. *ACM Trans. Embed. Comput. Syst.* 22, 1 (Oct. 2022), 1–31.

[9] Saman Biookaghazadeh, Ming Zhao, and Fengbo Ren. 2018. Are FPGAs Suitable for Edge Computing?. In *HotEdge*.

[10] D. J. Butler, J. Wulff, G. B. Stanley, and M. J. Black. 2012. A naturalistic open source movie for optical flow evaluation. In *European Conf. on Computer Vision (ECCV) (Part IV, LNCS 7577)*, A. Fitzgibbon et al. (Eds.) (Ed.). Springer-Verlag, 611–625.

[11] Emre Çetin, T Tolga Sari, Mert Assoy, and Gökhan Seçinti. 2024. Monodepth-based object detection and depth sensing for vehicle vision systems. In *World Forum on Internet of Things (WF-IoT)*. 696–701.

[12] Zheng Chai, Ahsan Ali, Syed Zawad, Stacey Truex, Ali Anwar, Nathalie Baracaldo, Yi Zhou, Heiko Ludwig, Feng Yan, and Yue Cheng. 2020. TiFL: A Tier-based Federated Learning System. In *Proceedings of the International Symposium on High-Performance Parallel and Distributed Computing*. 125–136.

[13] ChargerLAB. 2023. POWER-Z KM003C USB-C Power Meter. https://www.power-z.com/products/262. Accessed: 2025-05-16.

[14] Haiming Chen, Wei Qin, and Lei Wang. 2022. Task partitioning and offloading in IoT cloud-edge collaborative computing framework: a survey. *Journal of Cloud Computing* 11, 1 (2022), 86.

[15] Jie Chen, Yajing Leng, and Jiwei Huang. 2023. An intelligent approach of task offloading for dependent services in Mobile Edge Computing. *Journal of Cloud Computing* 12, 1 (2023), 107.

[16] Xue-Zhi Cui, Quan Feng, Shu-Zhi Wang, and Jian-Hua Zhang. 2022. Monocular depth estimation with self-supervised learning for vineyard unmanned agricultural vehicle. *Sensors* 22, 3 (2022), 721.

[17] Naqqash Dilshad, JaeYoung Hwang, JaeSeung Song, and NakMyoung Sung. 2020. Applications and challenges in video surveillance via drone: A brief survey. In *International Conference on Information and Communication Technology Convergence*. 728–732.

[18] Emily Dunkel, Jason Swope, Zaid Towfic, Steve Chien, Damon Russell, Joseph Sauvageau, Douglas Sheldon, Juan Romero-Canas, Jose Luis Espinosa-Aranda, Léonie Buckley, et al. 2022. Benchmarking deep learning inference of remote sensing imagery on the Qualcomm Snapdragon and Intel Movidius Myriad X processors onboard the International Space Station. In *IEEE International Geoscience and Remote Sensing Symposium*. 5301–5304.

[19] Arda Surya Editya, Tohari Ahmad, and Hudan Studiawan. 2022. Direction estimation of drone collision using optical flow for forensic investigation. In *2022 10th International Symposium on Digital Forensics and Security (ISDFS)*. IEEE, 1–6.

[20] Ahmed Elamin, Ahmed El-Rabbany, and Sunil Jacob. 2024. Event-based visual/inertial odometry for UAV indoor navigation. *Sensors* 25, 1 (2024), 61.

[21] Kamil Halbiniak, Roman Wyrzykowski, Lukasz Szustak, Adam Kulawik, Norbert Meyer, and Pawel Gepner. 2022. Performance exploration of various C/C++ compilers for AMD EPYC processors in numerical modeling of solidification. *Advances in Engineering Software* 166 (2022).

[22] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. 2018. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2704–2713.

[23] Bongjae Kim, Hong Min, Junyoung Heo, and Jinman Jung. 2018. Dynamic computation offloading scheme for drone-based surveillance systems. *Sensors* 18, 9 (2018), 2982.

[24] Yassin Kortli, Souhir Gabsi, L Voon, Maher Jridi, Mehrez Merzougui, and Mohamed Atri. 2022. Deep embedded hybrid CNN–LSTM network for lane detection on NVIDIA Jetson Xavier NX. *Knowledge-based Systems* 240 (2022).

[25] Haijun Liang, Xiangwei Zhang, Jianguo Kong, Zhiwei Zhao, and Kexin Ma. 2024. SMB-YOLOv5: a lightweight airport flying bird detection algorithm based on deep neural networks. *IEEE Access* 12 (2024), 84878–84892.

[26] Qianlin Liang, Prashant Shenoy, and David Irwin. 2020. AI on the Edge: Characterizing AI-based IoT Applications Using Specialized Edge Architectures. In *IEEE International Symposium on Workload Characterization*. 145–156. doi:10.1109/IISWC50251.2020.00023

[27] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. 2014. Microsoft COCO: Common Objects in Context. In *European Conference on Computer Vision*. 740–755.

[28] Xing Liu, Jianfeng Yang, Chengming Zou, Qimei Chen, Xin Yan, Yuao Chen, and Chenran Cai. 2022. Collaborative Edge Computing With FPGA-Based CNN Accelerators for Energy-Efficient and Time-Aware Face Tracking System. *IEEE Transactions on Computational Social Systems* 9, 1 (Feb. 2022), 252–266.

[29] Yu Liu, Bo Han, Feng Qian, Arvind Narayanan, and Zhi-Li Zhang. 2022. Vues: practical mobile volumetric video streaming through multiview transcoding. In *Proceedings of the Annual International Conference on Mobile Computing And Networking*. 514–527.

[30] Qinwei Luo, Jianfeng Zhu, and Hailong Pei. 2024. ORB-SLAM3 Front-End Acceleration System Based on ZYNQ Platform. In *Proceedings of the CAC)*. 2203–2208.

[31] Raul Mur-Artal, J. M. M. Montiel, and Juan D. Tardos. 2015. ORB-SLAM: A Versatile and Accurate Monocular SLAM System. *IEEE Transactions on Robotics* 31, 5 (Oct. 2015), 1147–1163.

[32] Samuel Naffziger, Noah Beck, Thomas Burd, Kevin Lepak, Gabriel H Loh, Mahesh Subramony, and Sean White. 2021. Pioneering chiplet technology and design for the AMD Epyc and Ryzen processor families: Industrial product. In *ACM/IEEE International Symposium on Computer Architecture*. 57–70.

[33] Mozhgan Navardi, Edward Humes, and Tinoosh Mohsenin. 2022. E2EdgeAI: Energy-efficient edge computing for deployment of vision-based DNNs on autonomous tiny drones. In *IEEE/ACM Symposium on Edge Computing*. 504–509.

[34] Mutagisha Norbelt, Xiling Luo, Jinping Sun, and Uwimana Claude. 2025. UAV localization in urban area mobility environment based on monocular VSLAM with deep learning. *Drones* 9, 3 (2025), 171.

[35] Qualcomm. 2025. Qualcomm Documentation: GPU Architecture Overview. https://docs.qualcomm.com/bundle/publicresource/topics/80-78185-2/gpu.html?product=1601111740035277. Accessed November 2025.

[36] Qualcomm. 2025. Qualcomm Hexagon Processor Overview. https://www.qualcomm.com/processors/hexagon. Accessed November 2025.

[37] Qualcomm Technologies. 2025. Snapdragon Overview. https://www.qualcomm.com/developer/hardware/qualcomm-innovators-development-kit. Accessed: 2025-02-21.

[38] Prashanthi S, S Kesanapalli, and Yogesh Simmhan. 2022. Characterizing the performance of accelerated Jetson edge devices for training deep learning models. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 6, 3 (2022), 1–26.

[39] Chung-An Shen, Ding-Yuan Lee, Chung-An Ku, Meng-Wei Lin, Kuo-Cheng Lu, and Shao-Yu Tan. 2019. A Programmable and FPGA-accelerated GTP Offloading Engine for Mobile Edge Computing in 5G Networks. In *Proceedings of the IEEE Conference on Computer Communications Workshop*.

[40] Ihsan Ullah, Hyun-Kyo Lim, Yeong-Jun Seok, and Youn-Hee Han. 2023. Optimizing task offloading and resource allocation in edge-cloud networks: a DRL approach. *Journal of Cloud Computing* 12, 1 (2023), 112.

[41] Ultralytics. 2025. YOLOv5 Models. https://docs.ultralytics.com/models/yolov5/. Accessed November 2025.

[42] Jianyu Wang, Jianli Pan, Flavio Esposito, Prasad Calyam, Zhicheng Yang, and Prasant Mohapatra. 2019. Edge Cloud Offloading Algorithms: Issues, Methods, and Perspectives. *Comput. Surveys* 52, 1 (2019), 1–23.

[43] Wiki. 2025. The Termux Wiki. https://wiki.termux.com/wiki/Main_Page. Accessed: 2025-03-02.

[44] Chenren Xu, Shuang Jiang, Guojie Luo, Guangyu Sun, Ning An, Gang Huang, and Xuanzhe Liu. 2022. The Case for FPGA-Based Edge Computing. *IEEE Transactions on Mobile Computing* 21, 7 (July 2022), 2610–2619.

[45] Qian Xu, Victor Li, et al. 2023. Neural Architecture Search for Intel Movidius VPU. *arXiv preprint arXiv:2305.03739* (2023).

[46] Ashkan Yousefpour, Genya Ishigaki, Riti Gour, and Jason P. Jue. 2018. On Reducing IoT Service Delay via Fog Offloading. *IEEE Internet of Things Journal* 5, 2 (April 2018), 998–1010.

[47] Zhehang Zhang, Bharadwaj Madabhushi, Sandip Kundu, and Russell Tessier. 2025. Managing Computation Offloading from Edge Devices to a Reconfigurable Edge Cloud. In *2025 IEEE 36th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE, 123–130.