2023 IEEE 20th International Conference on Mobile Ad Hoc and Smart Systems (MASS) | 979-8-3503-2433-4/23/$31.00 ©2023 IEEE | DOI: 10.1109/MASS58611.2023.00077

HeteroEdge: Addressing Asymmetry in Heterogeneous Collaborative Autonomous Systems

Mohammad Saeid Anwar[1], Emon Dey[1,2], Maloy Kumar Devnath[1], Indrajeet Ghosh[1,2],
Naima Khan[1,2], Jade Freeman[4], Timothy Gregory[4], Niranjan Suri[4], Kasthuri Jayarajah[2],
Sreenivasan Ramasamy Ramamurthy[3], Nirmalya Roy[1,2]
[1]Department of Information Systems, University of Maryland Baltimore County (UMBC), [2]CARDS, UMBC
[3]Department of Computer Science, Bowie State University, [4]DEVCOM Army Research Lab
[1](saeid.anwar, maloyd1)@umbc.edu, [1,2](edey1, indrajeetghosh, nkhan4, nroy)@umbc.edu, [2]kasthuri@umbc.edu,
[3]sramamurthy@bowiestate.edu, [4](jade.I.freeman2, timothy.c.gregory6, niranjan.suri).civ@army.mil

***Abstract*—Gathering knowledge about surroundings and generating situation awareness for autonomous systems is of utmost importance for systems developed for smart urban and uncontested environments. For example, a large area surveillance system is typically equipped with multi-modal sensors such as cameras and LIDARs and is required to execute deep learning algorithms for action, face, behavior, and object recognition. However, these systems are subjected to power and memory limitations due to their ubiquitous nature. As a result, optimizing how the sensed data is processed, fed to the deep learning algorithms, and the model inferences are communicated is critical. In this paper, we consider a testbed comprising two Unmanned Ground Vehicles (UGVs) and two NVIDIA Jetson devices and posit a self-adaptive optimization framework that is capable of navigating the workload of multiple tasks (storage, processing, computation, transmission, inference) collaboratively on multiple heterogenous nodes for multiple tasks simultaneously. The self-adaptive optimization framework involves compressing and masking the input image frames, identifying similar frames, and profiling the devices for various tasks to obtain the boundary conditions for the optimization framework. Finally, we propose and optimize a novel parameter *split-ratio*, which indicates the proportion of the data required to be offloaded to another device while considering the networking bandwidth, busy factor, memory (CPU, GPU, RAM), and power constraints of the devices in the testbed. Our evaluations captured while executing multiple tasks (e.g., PoseNet, SegNet, ImageNet, DetectNet, DepthNet) simultaneously, reveal that executing 70% (*split-ratio*=70%) of the data on the *auxiliary node* minimizes the offloading latency by $\approx$ 33% (18.7 ms/image to 12.5 ms/image) and the total operation time by $\approx$ 47% (69.32s to 36.43s) compared to the baseline configuration (executing on the *primary node*).**

***Index Terms*—Collaborative Systems, Deep Edge Intelligence, Autonomous Systems**

I. Introduction

In recent days, autonomous systems such as unmanned aerial or ground vehicles have been very popular in various applications, such as surveillance, photography and videography, mapping and surveying, agriculture, environmental monitoring, search and rescue, and delivery services. Unmanned vehicles equipped with sensors such as cameras, lidar, radar, and GPS [1] allow us to collect data and perform tasks (e.g., object detection, scene detection, and many more) in various environments. Performing these tasks while depending on the onboard computational unit (especially advanced deep learning models) involves limited power consumption, eventually affecting the autonomous systems' operation and safety [2]. Several recent studies have investigated the impact of operational time and power supply of autonomous systems [3]–[5], and suggest that the operational capacity of the systems is severely affected due to the execution of onboard sub-systems (e.g., navigation unit, cameras, communication systems) [5]. In addition to these sub-systems, accommodating Deep Neural Network (DNN) algorithms (usually power, memory, and computation hungry) to perform the tasks for situational awareness contributes to the already limited power availability [6]. Furthermore, as some of the systems' operations, such as navigation and communication, are more important for the safety of the expensive autonomous systems, optimizing the DNNs would be essential to conserve the limited available power.

One of the approaches to conserving power using DNNs can be achieved by offloading the inference task to a remote device (cloud server or a device connected to the same network) with surplus power and computational capability. However, such a solution is impacted by network availability, reliability, low bandwidth, and latency [7] caused due to the quality of communication links and the distance between the (*primary node*) (the device that will offload data) and the *auxiliary node* (either a remote server or edge device on the same network that can share the workload of the primary node). Some of the limitations of prior research on offloading include an offloading task to homogenous devices (e.g., MASA [8]) and smartphones [9] and expensive in the case of remote cloud services [10]. For scenarios such as situational awareness by autonomous systems, we hypothesize that leveraging another device within the system would mitigate latency and help conserve power compared to expensive cloud services. Keeping these discussions in mind, the overarching objective of this paper is to optimize and schedule tasks by offloading the data from a busy *primary node* to a relatively idle *auxiliary node*. To address this objective, we propose the following contribution.

(i) Data-Driven Resource-Aware Offloading Framework. This framework optimizes the system parameters, such as the processing complexity of the task, memory utilization, bandwidth, and power availability, to assert the *primary node*

2155-6814/23/$31.00 ©2023 IEEE
DOI 10.1109/MASS58611.2023.00077

to offload a portion of the data to an *auxiliary node*. Besides, we have introduced a novel parameter called *split-ratio*, which helps us efficiently offload the data to the *auxiliary node*. Our analysis indicates that offloading with a *split-ratio* of 0.7-0.8 enhances the performance of task execution at the expense of increased power and memory usage (*primary + auxiliary node*).

(ii) Testbed Development for System Evaluation. A system of two Nvidia Jetson devices and two UGVs was designed to evaluate the optimization framework. The devices were equipped with an MQTT-based publisher-subscriber protocol [11] to share the *auxiliary node's* system parameters to the *primary node* and offload the data to the *auxiliary node*. To show the effectiveness of our proposed optimization framework, we assess its performance on DNN applications with multiple data modalities, including posture estimation (identifying human postures like standing, sitting, or lying down), semantic segmentation (classifying each pixel in an image to the object it belongs to, providing a detailed understanding of the scene), and object detection.

(iii) Data Compression for Enhanced Optimization Performance. As the data size grows, offloading data becomes more expensive. As a result, a frame compression and masking technique was leveraged to eliminate similar frames and extract the object of interest from the data to be offloaded, thereby reducing inference time and communication overhead, eventually enhancing overall performance in data-intensive environments.

(iv) Simulating Real-World Scenarios to Evaluate Offloading Strategy. In real-world autonomous system operations, the individual nodes would be in motion, suggesting that the distance between the nodes can affect the network parameters. As a result, we simulate a scenario where the nodes are constantly in motion. As the distance increases, offloading latency rises, prompting questions about when to stop offloading images. This exploration offers valuable insights such as understanding the distance-offloading-latency relationship, identifying offloading thresholds for efficiency, and optimizing image offloading strategies in dynamic environments, ultimately improving performance in real world scenarios.

II. Related work

Edge inference and Offloading of multiple concurrent DNN tasks. Inferencing Multiple DNN tasks concurrently is a critical salient feature for any autonomous system's real-time operation. Motivated by this, the authors of heimdall [6] developed a mobile GPU coordination platform for emerging augmented reality applications in which frame rates decrease and inference latency increases significantly due to multi-DNN GPU contention. It is designed with a pseudo-preemption mechanism that (i) breaks down the multi-DNN into smaller units and (ii) prioritizes and flexibly schedules simultaneous GPU tasks. Additionally, in BAND [12], the authors develop a mobile-based inference system. BAND dynamically enables the creation of DNN execution plans and schedules DNNs on processors according to stated scheduling goals. In contrast, we consider a broader range of edge devices and application scenarios in this work to eventually minimize bandwidth consumption and latency.

Task-based Scheduling: Task-based scheduling plays a vital role in optimizing and reducing inference with respect to the assigned tasks. The authors of [13] propose a novel approach to the constraint optimization problem and suggest a greedy heuristic approach to choosing the best subset of concurrent applications within the constrained fidelity and resource budget. Additionally, a real-time layer-level DNN scheduling framework [14] enables CPU/GPU scheduling of individual DNN layers with fine-grained CPU/GPU allocation schemes. This work tackles the schedulability of real-time DNN tasks, the asymmetric nature of DNN task execution on CPU and GPU, respectively, and the lack of task-based scheduling of CPU/GPU-aware allocation schemes. In contrast, our work involves running multiple DNN applications concurrently by splitting data into different nodes, considering each device's capabilities and task requirements.

Frame-based compression techniques. Prior research has shown that optical flow can be utilized to estimate object motion across multiple camera views, enabling the system to track objects moving between cameras [15]. This information helps schedule video frame processing, minimizing latency and prioritizing relevant frames to meet real-time video analytics demands. n the other hand, an adaptive frame masking approach [16] has been proposed to improve the efficiency of video streaming and processing in edge computing environments, focusing on lower communication overhead and accelerated DNN inference. However, both [15] and [16] aim at compressing frames acquired from static cameras. In contrast, we consider both static and mobile autonomous devices to optimize image offloading strategies, enhancing efficiency and effectiveness in various real-world scenarios.

III. System Overview

Our system architecture comprises a Device profiler and an online scheduler as shown in Fig. 1. The inputs are the image data and it is split according to resource availability.

We consider three important parts to design the framework 1. They are frame masking, profile engine, and optimization. This involves devising an efficient frame masking solution to ensure optimal performance during offloading between edge devices and creating a profiling engine that accurately evaluates primary and auxiliary nodes' performance, considering memory, power, and inference time while adapting to dynamic conditions like UGV movement. An optimization framework is also required to identify optimal split ratios for offloading decisions within specified bounds, resulting in a comprehensive and adaptive solution for multi-DNN systems.

A. HeteroEdge *Components*

Device profiler. In order to make memory and power-aware scheduling, we analyze and monitoring of the performance of both devices, with a focus on key metrics such as device memory, power usage, inference time, and network latency. By

gathering this information, we are able to evaluate the resource availability of each device in real-time and make informed decisions on the allocation of processing tasks.

Task scheduler. In our optimization framework, we incorporated a task scheduler that intelligently manages task offloading in a multi-node environment. By gathering profiling data about primary and auxiliary devices, it assesses resource availability and the Multi-DNN workload. The task scheduler then ascertains if the primary node requires offloading and calculates the optimal data-split ratio for efficient resource utilization. Acting as a smart decision-making system, It distributes workloads based on resource availability and performance capabilities, leading to reduced inference time, less memory utilization, and enhanced overall system efficiency.

The are some difficulties in designing an efficient offloading framework for multi-DNN execution systems in resource-constrained edge environments. This framework must address device heterogeneity, resource constraints, performance variability, and energy efficiency while adapting to dynamic conditions like varying velocities and distances. In moving conditions, the offloading latency may vary due to changes in distance between devices, leading to potential inefficiencies. Processing large image data in edge environments poses challenges, as increased data transmission consumes more bandwidth and results in higher latency. Additionally, processing larger images requires more computational power, straining resource-constrained devices. To tackle this issue, we introduce a frame compression technique, specifically frame masking which is crucial for ensuring optimal performance during offloading. Frame masking reduces image data size, minimizing bandwidth consumption, lowering latency, and improving energy efficiency.

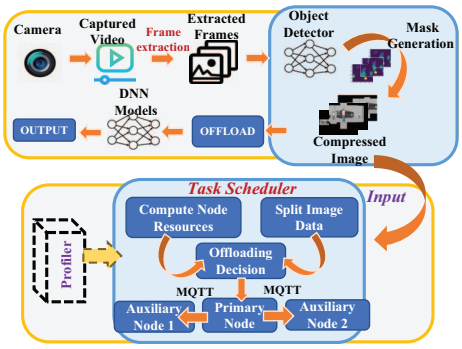


Fig. 1: Optimization framework overview: Frame compression technique and task scheduling process.

B. System Assumptions

In this context, we assume that resource-constrained devices have limited processing power and memory resources compared to more powerful servers or computers. When running multiple DNN models simultaneously, higher energy consumption and potential performance degradation are expected. Model inference may also be a concern when executing multiple DNN models concurrently on these devices. Additionally, scalability concerns arise due to the UGV's limited capacity to handle an increasing number of DNN models or growing complexity. The system assumes a variety of UGVs with different processing capabilities, memory capacities, and energy consumption profiles. The UGVs are connected through a communication network that allows for data offloading and communication between devices. The profiling engine can accurately measure performance metrics like memory utilization, power consumption, and inference time. The system assumes that the UGVs may be in motion, causing the distance between them to change and affecting communication latency

IV. *HeteroEdge* Profiling Engine

In our proposed system, the individual nodes continuously monitor system variables under multi-DNN workloads to identify optimal collaborative configurations. We first describe the testbed setup we used in profiling multi-DNN workloads across heterogeneous systems and then present quantitative insights from profiling various device and network attributes.

A. Testbed setup

We consider a network consisting of two heterogeneous edge platforms akin to a pair of autonomous systems with heterogeneous resources: (i) a low-resource Jetson Nano with a quad-core ARM Cortex-A57 MPCore processor, 4GB of LPDDR4 memory, and a 128-core NVIDIA Maxwell GPU, and (ii) a Jetson Xavier embedded with an octa-core NVIDIA Carmel ARM v8.2 CPU, 8GB LPDDR5, and a 512-core Volta GPU. In all our experiments, we assume that the lower end device (i.e., Nano) constantly monitors system parameters to offload its workload to the more powerful device, for executing multiple DNNs for downstream applications. Fig. 2 shows the experimental setup of our testbed. While the Jetson Xavier device was positioned in a fixed location, UGVs mounted with Jetson Nanos were moved at different angles and velocities for emulating various mobility conditions. We adopted a publisher-subscriber architecture [17] (specifically, the Message Queuing Telemetry Transport (MQTT) protocol) for message passing between the two devices.

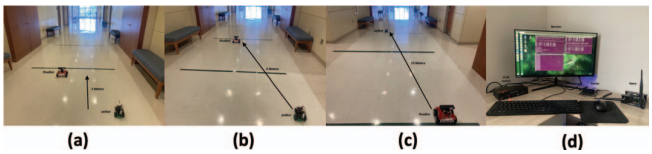


Fig. 2: Experimental setup for Testbed: (a) UGV setup for 2-meter distance (b) 6-meter distance (c) 10-meter distance (d) static device (Nano-Xavier) setup.

B. Device Profiling

HeteroEdge profiling engine runs on both primary and auxiliary nodes to continuously log memory utilization, power consumption, and inference time for both devices. While in our

TABLE I: Profiling results from testbed for semantic segmentation and posture estimation model

r (split ratio)	T_1 (Xavier) (s)	P_1 (Xavier) (w)	M_1 (Xavier) (%)	1-r	T_2 (Nano) (s)	T_3 (Offlatency) (s)	P_2 (nano) (w)	M_2 (nano) (%)
0	0	0.95	10.2	1	68.34	0	5.89	69.82
0.3	8.45	4.59	36.67	0.7	39.03	0.43	5.35	63.77
0.5	13.88	5.42	45.61	0.5	28.35	0.89	5.63	52.54
0.7	16.64	5.73	51.23	0.3	19.54	1.25	4.75	45.58
0.8	17.24	6.17	56.96	0.2	13.34	1.44	4.48	40.34
1	19.001	6.38	59.37	0	0	1.56	0.77	16

experiments we consider the low-resource device (i.e., Jetson Nano) as the primary node and Jetson Xavier as the auxiliary node for simplicity, in reality, all nodes in the network can assume primary and auxiliary roles. *HeteroEdge* uses Jetson Stats [18] to measure memory utilization and average power consumption.

DNN Workloads. As *HeteroEdge* is designed for autonomous systems that are required to run multiple concurrent compute-intensive tasks, we run two exemplar tasks, namely semantic segmentation and posture estimation, using a multiprocessing pool. *HeteroEdge* utilizes the Nvidia Jetson Inference Library [19] for profiling the various DNN models in order to optimize offloading decisions.

Split Ratio (r). In our work, we propose the **split ratio**, which represents the proportion of images offloaded to the auxiliary node. It ranges from 0 (all images processed locally) to 1 (all images offloaded to the auxiliary). The optimal split ratio maximizes the collaborative system's throughput while minimizing resource consumption. Notations $T1$, $P1$, and $M1$ represent operation time, power, and memory usage of the auxiliary, while $T2$, $P2$, and $M2$ represent those of the primary node. $Offlatency$ refers to the network latency resulting from offloading images.

In Table I, we report the measured performance of the two devices in processing a batch of 100 images, under various configurations – r ranging from 0 to 1. As anticipated, we observe that while the overall power consumption is comparable between the nodes, the processing latency is significantly lower for the auxiliary device for the same workload. For e.g., at a $r = 0.5$, while the processing time on the primary ($\approx$ 28.35 seconds) is double that of the auxiliary ($\approx$ 13.88 secs), the power consumption is comparable at 5.63 W and 5.42 W. At the same time, we also note that the offloading latency varies only minimally (between 0 and 1.56 secs) with r, supporting our premise for intelligent offloading.

C. Network Profiling

Furthermore, We investigated the network latency under two different network configurations- specifically, WiFi on two frequency bands, 2.4 Ghz and 5 Ghz. In Fig. 3, we plot the latency ($y-$axis) (a) for different sizes of images, (b) various split ratios, and (c) distance between the primary and auxiliary devices, on the $x-$axis. We note that the higher band offers lower latencies and we observe increasing latencies with both increasing split ratios as well as distances.

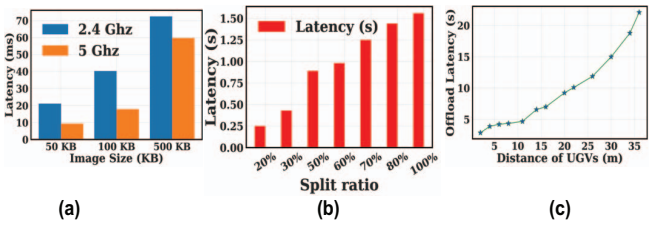


Fig. 3: MQTT latency for (a) different network bands & image sizes, (b) different split ratios, & (c) different distances with differing velocities of UGVs.

In the next section, we describe the *HeteroEdge* solver which takes into account the profiled variables to output an optimal collaborative system for vision tasks.

V. *HeteroEdge* Solver

We design the *HeteroEdge* solver such that it dynamically adjusts data-splitting ratios based on available resources, optimizing the collective throughput and resource utilization. This cost-effective and scalable solution is applicable to various edge computing scenarios, addressing resource limitations and energy constraints, while allowing efficient processing of large data volumes in a distributed manner. In the following subsections, we describe the steps in devising the optimization framework.

A. Latency and Energy Modeling

In Table II, we list the variables used in the optimization formulation.

TABLE II: Important Notation & Meaning.

Notation	Meaning
I	Input size of computational task
N	Number of cpu cycle needed to execute one bit of computation data
C_{cpu}	Total cycle needed to finish specific task
T_{exec}	Total execution time
S	Computation speed of a client device
E_{exec}	Total execution energy
B	Transmission Bandwidth
D_R	Data rate
T_0	Offloading latency
r	Split ratio
T_s	Time required to run ratio offloading code
E_0	Offloading energy
E_s	Energy required to run ratio offloading code

1) Execution Period: I denotes the input size of the computation task of the offloading device. N presents the number of CPU cycles needed to execute one bit of input computation data. Therefore, $C_{cpu} = NI$ denotes the cycles needed to finish the selected computation task. So, the executing latency can be expressed as $T_{exec} = \frac{C_{cpu}}{S}$ where S is the computation speed of the device and it is measured in cycles per second.

We model the power consumption of CPU as $P = \mu(S)^3$ as in [20] Thus, the energy consumption per cycle is $\mu(S)^2$. The energy consumption for deep model processing is then $E_{exec} = C_{cpu}\mu(S)^2$ where μ is a coefficient depending on

chip architecture. We assume that the computing speed of the server's CPU is limited to S^{max}, so we may have $0 \leq S \leq S^{max}$. Considering r is the split ratio,

$$E_{exec} = E_1 r + E_2(1-r)$$
$$T_{exec} = T_1 r + T_2(1-r)$$

Here, E_1 and E_2 are the execution energy of the two nodes participating to complete one task. Similarly, T_1 and T_2 are the execution time of the two devices for a specific split ratio.

2) Offloading Period: If B is transmission bandwidth, d is distance between two devices, e is path loss exponent, N_0 is the Gaussian noise power, the transmission data rate for offloading task can be found using the Shannon-Hartley algorithm [21].

$$D_R = B \ \log_2(1 + \frac{d^{-u} P_t}{N_0})$$

Here, P_t is the transmission power of the device during offloading. If the medium is lossless then we can put $u = 0$.

Then, the Offloading Latency is $T_o = \frac{C}{D_R}$. Here, C will depend on the selected split ratio.

Total latency can be given by $T = T_{exec} + T_o + T_s$. Energy required to run the split ratio selection code is then $E_s = P_k T_s$ where P_k is the power rating of the device which will run the solver code.

Offloading energy requirement can be divided into two parts,

$$E_0 = T_0 \sum_{i=0}^{N} P_i$$

Here, P_t is the power required of the sender device and P_r is the power drawn while receiving the sent data. Then the Total energy can be expressed as $E = E_{exec} + E_s + E_o$.

3) Solver: As our objective is to make the system both memory and energy-aware while minimizing the latency (i.e., improving the overall throughput), we have derived a relation between energy and memory during execution time. Specifically, we have considered the quadratic relation between energy and required memory during execution. While solving for optimization, we can use the variable substitution approach. We can work with the real part only of the following equation for a sub-optimal solution.

$$T = r(T_1 + T_3) + (1-r)T_2$$

Here T1 is the operation time for Jetson Xavier and T_2 is the operation time for Jetson Nano. T_3 is the round trip time for image transfer.

$$T_1 = a_1 r^2 + a_2 r + c_1 \tag{1}$$
$$T_2 = b_1(1-r)^2 + b_2(1-r) + c_2$$
$$E_1 = a_1 r^3 + a_2 r^2 + a_3 r + c_1 \tag{2}$$
$$E_2 = b_1(1-r)^3 + b_2(1-r)^2 + b_3 r + c_2$$
$$M_1 = a_1 r^2 + a_2 r + c_1 \tag{3}$$
$$M_2 = b_1(1-r)^2 + b_2(1-r) + c_2$$

Here, $E1$, and $E2$ are energy consumption for Jetson Xavier and Nano. $M1$ and $M2$ are also memory needed for Jetson Xavier and Nano the values of a_1, a_2, a_3, b_1, b_2, b_3 coefficients can be found through curve fitting feature with some experimental values. Problem formulation,

$$\begin{aligned} \min(T) \quad & \text{constraints :} \\ & C1 : T \leq \frac{\tau}{k}, && C2 : 0 \leq P_k \leq P^{\max}, \\ & C3 : 0 < r < 1, && C4 : 0 \leq S \leq S^{\max}, \\ & C5 : E^k_{\text{exe}} \leq W^k, && C6 : M^k_{\text{exe}} \leq M^k. \end{aligned} \tag{4}$$

Here, τ is the execution latency while the whole operation is done by one device and k is the total number of devices. W^k and M^k are the highest power and memory ratings of the individual devices respectively.

4) Battery and Charging Constraints: We pose additional constraints based on the remaining battery life (before a recharge is required), to specifically cater for our mobile autonomous systems. To account for this, we consider two UGVs RosBot[1] and JetBot[2] with a 4000 mAh battery capacity and a discharge rate of 70%. These run on computing platforms similar to those considered in our previous analyses. The UGVs have a driving time of about 20-25 minutes and lose around 15-20 watts during that time. Running an additional DNN model on the bot will reduce its power capacity even further. If our DNN model runs for 50-60 seconds, it will consistently consume 5 to 6 watts of power. Equations 5 and 6 show how we calculate available power given these additional constraints.

The optimization is then based on an additional power threshold constraint, and when the available power (P) falls below the threshold, the UGV starts offloading more aggressively to the auxiliary system (recall that the power consumption of running concurrent DNNs is more than that of offloading images, see Table I). This ensures that the UGV can complete its mission without running out of power.

$$E_{available} = C_0 k - E_{dnn} - E_{drive} \tag{5}$$
$$P_{available} = E_{available} / \left((1-k)\left(t_{dnn} + t_{drive}\right)/3600\right) \tag{6}$$

Here, $E_{available}$ represents the available energy, k is the discharge rate of the battery, while E_{dnn} and E_{drive} denote the energy consumption of running the DNN model and driving the UGV, respectively. t_{dnn} is the duration of the DNN model in seconds, t_{drive} is the duration of the UGV driving in seconds. Using curve fitting, we derive memory and power as functions of inference time. We found empirically that quadratic functions have the best fit with adjusted R^2 values of 0.976 and 0.989, respectively.

5) Mobility Constraints: To mitigate offloading latency in dynamic scenarios, we introduce a minimum threshold, denoted as β. This threshold efficiently manages the offloading process by considering the UGVs' evolving motion.

To calculate the distance between the UGVs, we employ the following equation:

$$d = (V_{\text{primary}} + V_{\text{auxiliary}}) \times t$$

[1]https://husarion.com/manuals/rosbot/

[2]https://jetbot.org/master/

This equation calculates the distance, d, between two UGVs based on their velocities, V_{primary} and $V_{\text{auxiliary}}$, and a given time interval, t. The equation takes into account the relative motion of the UGVs, and the distance increases as the UGVs move apart during the time interval. The relationship between latency (L) and distance (d) between the two UGVs is modeled using the following equation and we get this equation from curve fitting:

$$L = a_1 \times d^2 - a_2 \times d + a_3$$

This equation represents the time delay (latency) in sending images from one UGV to another. As the distance, d, between the devices increases, the latency, L, also increases, affecting the efficiency of the offloading process.

When the latency meets or exceeds the threshold, β, the system stops sending data:

$$\text{If } L \geq \beta, \text{ stop sending data}$$

This approach ensures that the offloading process is adapted to the dynamic changes in UGV motion, resulting in more efficient resource utilization and improved overall performance.

B. Implementation

We use Python's GEKKO library [22] to solve our optimization problems. The objective function, variables and constraints are all specified as we described earlier with a nonlinear optimization solver (i.e., IPOPT solver [23]).

Algorithm 1 Algorithm for Split Ratio Selection

Require: IPs of connected devices n, Memory profiling of the nodes M, Inference time of device 1 T_1, Inference time of device 2 T_2, Round trip time T_3

Ensure: Determine the split ratio r for optimal operation time

1: On the primary node:
2: Calculate the device availability factor λ based on the memory of both devices. **Compute** the coefficients a_1, a_2, b_1, b_2, c_1, c_2 from equations 3 using curve fitting
3: **if** $M_1, M_2 \geq \lambda$ and check latency, $L \leq \beta$ **then**
4: Assign constraints from equation 3 on the following objective:

$$T = r(T_1 + T_3) + (1 - r)T_2$$

5: Check battery capacity and available UGV power: 5 and 6

$$P_{available} = E_{available} / \left((1 - k) \left(t_{dnn} + t_{drive} \right) / 3600 \right)$$

if $P_{available} >= E_{max}$
6: Solve the formulated problem for the given constraints using Interior Point Optimizer method
7: Send the derived amount of data to the subscriber node

VI. Frame-level Compression

To further optimize performance, *HeteroEdge* teases out *regions of interest* in images prior to running downstream DNN inferences (e.g., pose detection, image segmentation, etc.). This step ensures that the network latency from offloading them to an auxiliary node, when feasible, is also lowered. *HeteroEdge* first uses a state-of-the-art object detection model that generates binary masks where pixels with *detected objects* are denoted by bit 1, and 0 elsewhere. Element-wise multiplication of the binary mask with the original image returns a *compressed image* (see Fig. 4) which isolates objects of interest and eliminate extraneous backgrounds.

To demonstrate the savings in bandwidth utilization and inference times, we conduct microbenchmark experiments using two downstream DNN models. We generated a virtual environment in the Gazebo simulator [24] generating 3100 images with a total of 9 common object classes such as persons and vehicles present. First, we use the faster-RCNN object detector [25] for generating compressed images which are then fed to exemplar downstream DNNs; semantic segmentation (SegNet [26]) and posture detection (PoseNet [27]) models, respectively. Figure 4 shows illustrative examples of the output on the compressed frames for the two tasks. Overall, we observe a 13% reduction (on a Jetson Nano device) in the total computational time corresponding to a savings in bandwidth by up to 28% (i.e., from 8 MB down to 5.8 MB through compression) if the images were to be offloaded. While we observe only a 2% drop in inference accuracy from compression, the astute reader will note that an imperfect object detector model may create artifacts for downstream computer vision tasks; we defer an in-depth study as future work.

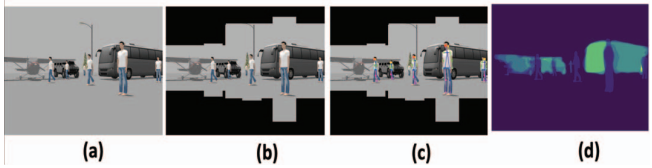


Fig. 4: (a) Original frame (b) Compressed frame & input (c) Results from compressed frame for Pose Estimation (d) Results from compressed frame for Semantic Segmentation.

VII. Evaluation

This section presents our in-depth evaluation of the proposed optimization framework on the Gazebo-based dataset we describe in Section VI, previously.

A. Deriving constraints for optimization

We derive constraints for optimization presented in equation 4 which provides the limitations and requirements of different resources that must be satisfied during the optimization process. We consider deriving constraints for important resources i.e., memory, power and inference time for optimizing the task offloading process.

From the experiments performed on the primary node, we obtain the base processing time for running multiple models on a single device. The total processing time for 100 images was found to be 68.34 seconds, as shown in Table I. We got 200 output images from two models for 100 image inputs. We use this as baseline for inference time, where optimized inference time must be less than 68.34 seconds overall. As we described in Section V, we use curve fitting to analyze the relationship

TABLE III: Results from the real-time system for static condition.

r (split ratio)	T3 (Offloading latency) (s)	P1 (Xavier) (w)	M1 (Xavier) (%)	1-r	T1+T2 (s)	P2 (nano) (w)	M2 (nano) (%)
0.2	.67	4.87	32.09	0.8	55.38	6.96	75.12
0.35	1.23	5.12	41.56	0.65	51.89	6.11	70.17
0.45	1.98	5.78	49.55	0.55	42.87	6.24	65.66
0.5	2.34	5.57	50.09	0.5	43.09	5.69	54.65
0.6	2.90	6.35	53	0.4	39.45	5.88	57.77
0.7	3.23	6.03	59.56	0.3	36.43	5.17	47.13
0.8	3.55	6.34	63.45	0.2	34.90	5.35	43.34
0.9	3.56	7.12	69.09	0.1	28.23	4.89	40.11

between inference time and data splitting ratio, memory and splitting ratio, and power and splitting ratio which represent equations 123. This enables us to predict the inference time for different splitting ratios and allows us to identify the optimal data-splitting ratio that will provide the lowest inference time while considering memory and power constraints for the task offloading process. Fig. 5(a) and Fig. 5(b) show the time, memory and power for different split ratios by our proposed *HeteroEdge* solver .

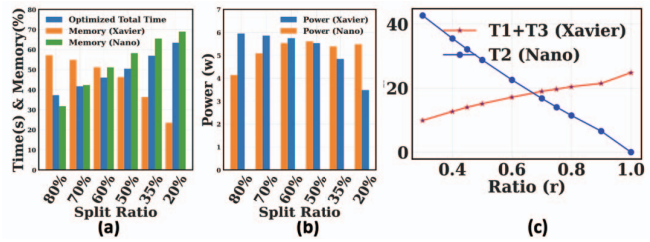


Fig. 5: Optimized results for different split ratios. (a) Total time and (b) memory usage for different split ratios and power usage for both devices, (c) Computational time for both devices changes with split ratio r.

From the solver we got the best value of the split ratio is 70% within our desired memory and power constraints. The total inference time for this split ratio is 17.72 seconds for Xavier (70 images) and 16.79 seconds for Nano (30 images). On average for two models and 200 outcomes, it takes a total of 34.51 seconds.

B. Empirical Evaluation

We perform empirical evaluations of static and dynamic conditions for our scenarios. We discuss the static and dynamic conditions in two case studies in the following.

Case-1: In this case, two UGVs are positioned at a fixed distance from each other, and their velocities are the same, leading to no relative movement between them. Since they are only 4 meters apart in our experiment, the communication overhead and latency remain constant throughout the evaluation. With a stable MQTT communication, the offloading of images from the primary UGV to the auxiliary UGV experiences a consistent latency based on their fixed distance. This constant latency allows an effective offloading process without additional communication overhead due to varying distances. The optimization framework can then be evaluated under this static condition to understand its performance in a controlled environment with minimal variations in latency.

The performance metrics for different ratios tested on the real-time testbed with the static condition are consistent with the optimization results from the solver, as illustrated in Table III. Here $T_1 + T_2$ represents the total operation time for both Jetson Nano and Xavier and T_3 stands for the offloading latency for UGVs with the static condition of 4 meters distance from each other. From the results, we notice there is a slight change in offloading latency with split ratios. In this case, offloading latency increases only with higher split ratios and the distance between UGVs is constant.

After using the IPOPT optimization solver, we estimate that offloading 70% of the images to a more powerful Xavier is the best optimal option given our specific memory and power constraints. We then evaluated the results in real-time systems (Table III) and tested different split ratios on a testbed and ended up with our desired results, which support our optimization framework. Overall, this demonstrates the efficacy of our framework in memory and power-aware task offloading, as well as the potential for further optimizations.

Case-2: In this case, the two UGVs are in motion with different velocities and/or directions, resulting in a dynamic distance between them. As the distance between the UGVs changes over time, the communication overhead and latency for offloading images from the primary to the auxiliary node also vary. Due to the dynamic nature of this scenario, the MQTT communication may experience fluctuations in latency based on the changing distance between the UGVs. This can lead to challenges in the optimization framework, as the offloading process may need to account for variations in communication overhead and latency. The optimization framework can be evaluated under this dynamic condition to understand its performance in a real-world environment with changing distances between the UGVs. This will help identify any potential issues and adjustments that may be needed to improve the framework's adaptability to varying communication conditions.

In Fig. 6 we record total operation time $(T_1 + T_2)$ for both UGVs and offloading latency T_3 for the varying distances between UGVs. In this setup, we use three different split ratio 30%, 70% and worst case 100% split and velocity for primary and auxiliary UGVs are respectively $V_{primary}$=1 m/s

TABLE IV: Overview of different deployed models in the testbed.

Application	DNN Model	Split Ratio (r=0) Run 100% on primary node T2 (Nano) (s) (Original image)	Split Ratio(r=0) T2 (nano) (s) (Masked image)	Split Ratio (r=0.5) T1 (Xavier)+ T2 (Nano) (s) (Original image)	Split Ratio (r=0.5) T1 (Xavier)+ T2 (Nano) (s) (Masked image)	Split Ratio (r=0.7) T1(Xavier)+ T2 (Nano) (s) (Original image)	Split Ratio (r=0.7) T1 (Xavier)+ T2 (Nano) (s) (Masked image)
Image recognition+ Object Detection	ImageNet, DetectNet	74.68	69.90	56.74	49.78	44.13	38.98
Object detection+ Depth Sensing	DetectNet, DepthNet	76.90	71.34	64.2	57.89	43.17	40.32
Semantic Segmentation + Depth Sensing	SegNet, DepthNet	71.25	65.56	58.43	53.66	48.37	43.2
Image recognition+ Depth Sensing	ImageNet, DepthNet	69.66	61.47	50.64	46.45	43.54	38.43
Object Detection+ Pose estimation	DetectNet, PoseNet	67.28	64.89	51.59	46.89	39.69	35.9

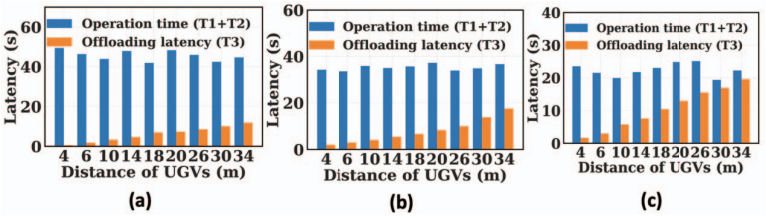


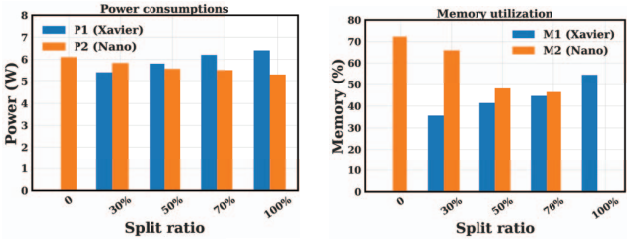
Fig. 6: The experimental setup for the testbed is used to evaluate the performance of the system at different distances. (a) 30% split ratio (b)70% split ratio (c) 100% split ratio.

and $V_{auxiliary}$=3 m/s. The evaluation results reveal a positive correlation between the distance among the UGVs and the offloading latency. As the distance increases, the offloading latency also rises, affecting the optimization system's effectiveness. For instance, at a distance of 26 meters, the average offloading latency from the primary node to the auxiliary node is 13.9 seconds, which increases the communication overhead and compromises the system's performance.

In order to address this challenge, we propose an offloading latency threshold. The system can effectively track the offloading latency during the operation. If the latency surpasses the threshold, the primary node stops offloading images to the auxiliary node and searches for a more suitable split ratio lower than the previous one. If the search for an optimal split ratio within the bounds is unsuccessful, the primary node performs all processing tasks locally. This adaptive approach maintains optimal performance by minimizing communication overhead and avoiding excessive latency.

C. Evaluation with Model Heterogeneity

In order to thoroughly evaluate the performance of the *HeteroEdge*, it is critical to validate it with a diverse range of models that represent different use cases and applications. To accomplish this, we carefully selected a range of models that represent different types of computational requirements from object detection to image classification and depth estimation. As an exemplar, we selected computer vision model ImageNet [28] for object detection, DetectNet [29] for object localization and DepthNet [30] for monocular depth estimation. We deployed each model including the previous two models PoseNet and SegNet on our *HeteroEdge* testbed and ran them concurrently while measuring the key performance metrics such as total operation time, power consumption, and resource utilization.



(a) (b)

Fig. 7: (a) Average power consumption (b) Average memory utilization.

Table IV depicts the results from multiple DNN models simultaneously running on primary and auxiliary nodes. We tested it for 100 images and when the split ratio became zero, the entire processing was carried out only on the primary node as shown in Table IV. We observe that the primary node sends 50% and 70% of the images to the auxiliary node for split ratio 0.5 and 0.7, respectively. It is worthwhile to note that there is an additional runtime on the primary node for object detection and mask generation. We record on average 3-4 ms latency per image with a lightweight faster-rCNN model [25]. We notice that the total operating time is lower (on average 9%) in case of masked frames compared to the original frames though there was a notable change in power consumption and memory utilization. Fig. 7(a) presents a slight increase in power consumption which is on average 4-5% more compared to the baseline where all processing is done locally for split

ratio = 0. On contrary, Fig.7(b) depicts a significant reduction in memory usage compared to the baseline memory usage i.e., $\approx$ 72.23% at split ratio = 0. For example, for a 70% split ratio, both devices use an average of 47% of memory, which is almost a 34% decrease compared to the baseline configuration.

VIII. Conclusion & Future work

In conclusion, our optimization work focused on reducing the latency of DNN model inference by offloading image processing to more powerful computing devices. We proposed framewise optimization a split ratio metric to determine the proportion of images to offload and used a solver to determine the optimal split ratio based on the memory and power constraints of UGV. Overall, our work provides a practical solution for reducing DNN model inference latency on resource-constrained devices. Our results show that offloading with MQTT and dynamic adjustment of the split ratio based on available power can further reduce latency and improve performance. In this work, we utilize a primary-auxiliary node setting which is hierarchical offloading in nature. We want to extend the current research to consider a star topology for offloading tasks in future work. In a star topology, a central node (the "hub") manages the communication and coordination among multiple edge devices (the "spokes"), allowing for more efficient resource allocation and data sharing.

Acknowledgment

This work has been partially supported by NSF CAREER Award #1750936, U.S.Army Grant #W911NF2120076, NSF CNS Grant #2233879 and NSF REU Site Grant #2050999

References

[1] Muntasir Mahmud, Mohamed Younis, Gary Carter, and Fow-Sen Choa. Underwater node localization using optoacoustic signals. In *ICC 2022 - IEEE International Conference on Communications*, pages 4444–4449, 2022.

[2] Syed Agha Hassnain Mohsan, Nawaf Qasem Hamood Othman, Yanlong Li, Mohammed Alsharif, and Muhammad Khan. Unmanned aerial vehicles (uavs): Practical aspects, applications, open challenges, security issues, and future trends. *Intelligent Service Robotics*, 01 2023.

[3] Pedram Beigi, Mohammad Sadra Rajabi, and Sina Aghakhani. An Overview of Drone Energy Consumption Factors and Models. *arXiv e-prints*, page arXiv:2206.10775, June 2022.

[4] *New Intelligent Battery Management System for Drones*, volume Volume 6: Energy of *ASME International Mechanical Engineering Congress and Exposition*, 11 2019. V006T06A028.

[5] Aakash Khochare, Yogesh Simmhan, Francesco Betti Sorbelli, and Sajal K. Das. Heuristic algorithms for co-scheduling of edge analytics and routes for uav fleet missions. In *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*, pages 1–10, 2021.

[6] Juheon Yi and Youngki Lee. Heimdall: Mobile gpu coordination platform for augmented reality applications. *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, 2020.

[7] H. Chen, W. Qin, and L. Wang. Task partitioning and offloading in IoT cloud-edge collaborative computing framework: a survey. *Journal of Cloud Computing*, 11(1), dec 3 2022.

[8] Bart Cox, Jeroen Galjaard, Amirmasoud Ghiassi, Robert Birke, and Lydia Y. Chen. Masa: Responsive multi-dnn inference on the edge. In *2021 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pages 1–10, 2021.

[9] Quang-Huy Nguyen and Falko Dressler. A smartphone perspective on computation offloading—a survey. *Computer Communications*, 159:133–154, 2020.

[10] Erol Gelenbe, Ricardo Lent, and Markos Douratsos. Choosing a local or remote cloud. In *2012 Second Symposium on Network Cloud Computing and Applications*, pages 25–30, 2012.

[11] Urs Hunkeler, Hong Linh Truong, and Andy Stanford-Clark. Mqtt-s — a publish/subscribe protocol for wireless sensor networks. In *2008 3rd International Conference on Communication Systems Software and Middleware and Workshops (COMSWARE '08)*, pages 791–798, 2008.

[12] Joo Seong Jeong, Jingyu Lee, Donghyun Kim, Changmin Jeon, Changjin Jeong, Youngki Lee, and Byung-Gon Chun. Band: coordinated multi-dnn inference on heterogeneous mobile processors. In *Proceedings of the 20th Annual International Conference on Mobile Systems, Applications and Services*, pages 235–247, 2022.

[13] Nirmalya Roy, Vasanth Rajamani, and Christine Julien. Supporting multi-fidelity-aware concurrent applications in dynamic sensor networks. In *2010 8th IEEE International Conference on Pervasive Computing and Communications Workshops*, pages 43–49. IEEE, 2010.

[14] Woosung Kang, Kilho Lee, Jinkyu Lee, Insik Shin, and Hoon Sung Chwa. Lalarand: Flexible layer-by-layer cpu/gpu scheduling for real-time dnn tasks. In *2021 IEEE Real-Time Systems Symposium (RTSS)*, pages 329–341. IEEE, 2021.

[15] Shengzhong Liu, Tianshi Wang, Hongpeng Guo, Xinzhe Fu, Philip David, Maggie B. Wigness, Archan Misra, and Tarek F. Abdelzaher. Multi-view scheduling of onboard live video analytics to minimize frame processing latency. In *42nd IEEE International Conference on Distributed Computing Systems,ICDCS 2022, Bologna, Italy, July 10-13, 2022*, pages 503–514. IEEE, 2022.

[16] Shengzhong Liu, Tianshi Wang, Jinyang Li, Dachun Sun, Mani Srivastava, and Tarek Abdelzaher. Adamask: Enabling machine-centric video streaming with adaptive frame masking for dnn inference offloading. In *MM '22: The 30th ACMInternational Conference on Multimedia,Lisboa, Portugal, October 10 - 14, 2022*, pages 3035–3044. ACM, 2022.

[17] Emon Dey, Jumman Hossain, Nirmalya Roy, and Carl Busart. Synchrosim: An integrated co-simulation middleware for heterogeneous multi-robot system. In *2022 18th International Conference on Distributed Computing in Sensor Systems (DCOSS)*, pages 334–341. IEEE, 2022.

[18] Jetson stats. https://https://github.com/rbonghi/jetson_stats. Accessed: 2023-02-03.

[19] A. Kurniawan. Deep-Learning Computation. https://link.springer.com/chapter/10.1007/978-1-4842-6452-2_7, dec 12 2020.

[20] Weiwen Zhang, Yonggang Wen, Kyle Guan, Dan Kilper, Haiyun Luo, and Dapeng Oliver Wu. Energy-optimal mobile cloud computing under stochastic wireless channel. *IEEE Transactions on Wireless Communications*, 12(9):4569–4581, 2013.

[21] Shannon–hartley theorem. https://https://en.wikipedia.org/wiki/Shannon-Hartley_theorem. Accessed: 2022-08-06.

[22] Logan D. R. Beal, Daniel C. Hill, R. Abraham Martin, and John D. Hedengren. Gekko optimization suite. *Processes*, 6(8), 2018.

[23] L.T. Biegler and V.M. Zavala. Large-scale nonlinear programming using ipopt: An integrating framework for enterprise-wide dynamic optimization. *Computers and Chemical Engineering*, 33(3):575–582, 2009. Selected Papers from the 17th European Symposium on Computer Aided Process Engineering held in Bucharest, Romania, May 2007.

[24] Gazebo. https://gazebosim.org/. Accessed: 2022-08-06.

[25] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[26] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(12):2481–2495, 2017.

[27] Alex Kendall, Matthew Grimes, and Roberto Cipolla. Posenet: A convolutional network for real-time 6-dof camera relocalization. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[28] Agus Kurniawan. *Deep-Learning Computation*, pages 107–119. Apress, Berkeley, CA, 2021.

[29] Andrew Tao, Jon Barker, and Sriya Sarathy. Detectnet: Deep neural network for object detection in digits. *Parallel Forall*, 4, 2016.

[30] Arun CS Kumar, Suchendra M. Bhandarkar, and Mukta Prasad. Depthnet: A recurrent neural network architecture for monocular depth prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2018.