Edge-Centric Real-Time Segmentation for Autonomous Underwater Cave Exploration

Mohammadreza Mohammadi
CSE, University of South Carolina
Columbia, SC 29208, USA
mohammm@email.sc.edu

Adnan Abdullah
ECE, University of Florida
Gainesville, FL 32611, USA
adnanabdullah@ufl.edu

Aishneet Juneja
CSE, University of South Carolina
Columbia, SC 29208, USA
junejaa@email.sc.edu

Ioannis Rekleitis
CSE, University of South Carolina
Columbia, SC 29208, USA
yiannisr@cse.sc.edu

Md Jahidul Islam
ECE, University of Florida
Gainesville, FL 32611, USA
jahid@ece.ufl.edu

Ramtin Zand
CSE, University of South Carolina
Columbia, SC 29208, USA
ramtin@cse.sc.edu

***Abstract*—This paper addresses the challenge of deploying machine learning (ML)-based segmentation models on edge platforms to facilitate real-time scene segmentation for Autonomous Underwater Vehicles (AUVs) in underwater cave exploration and mapping scenarios. We focus on three ML models—U-Net, CaveSeg, and YOLOv8n—deployed on four edge platforms: Raspberry Pi-4, Intel Neural Compute Stick 2 (NCS2), Google Edge TPU, and NVIDIA Jetson Nano. Experimental results reveal that mobile models with modern architectures, such as YOLOv8n, and specialized models for semantic segmentation, like U-Net, offer higher accuracy with lower latency. YOLOv8n emerged as the most accurate model, achieving a 72.5 Intersection Over Union (IoU) score. Meanwhile, the U-Net model deployed on the Coral Dev board delivered the highest speed at 79.24 FPS and the lowest energy consumption at 6.23 mJ. The detailed quantitative analyses and comparative results presented in this paper offer critical insights for deploying cave segmentation systems on underwater robots, ensuring safe and reliable AUV navigation during cave exploration and mapping missions.**

***Index Terms*—Edge Computing, Segmentation, Underwater Robots, Visual Servoing.**

I. Introduction & Background

Underwater cave exploration is a challenging frontier in robotics, offering significant potential for advancing our understanding of archaeology, hydrology, geology, and marine biology [1]. Cave formations, sediments, and water chemistry provide critical insights into historic climate conditions and geological events. They also play a crucial role in monitoring and tracking groundwater flows in Karst topographies, which supply freshwater to nearly a quarter of the global population [2]. However, these environments are often inaccessible and hazardous to human divers due to their complex, confined spaces and the absence of natural light [3], [4]. Hence, there is a growing emphasis on using Autonomous Underwater Vehicles (AUVs) and Remotely Operated Vehicles (ROVs) for safe and efficient exploration of underwater caves [5]–[7]. Fig. 1 shows a cave expedition scenario with an ROV inside an underwater cave system in Orange Grove, Florida.

The underwater caves explored by human scuba divers are marked with a single and continuous line termed *caveline* [8] that goes from open water (no overhead) to all the major parts of the cave. Along with other navigation markers such as *arrows* and *cookies*, the caveline provides the skeleton of the cave *i.e.*, a one-dimensional retraction of the 3D space [9] marking the depth and orientation of the main passages. Thus, detecting and following the caveline as navigation guidance is paramount for robots in autonomous cave exploration and mapping missions. Recently, Yu *et al.* [10] developed a robust Vision Transformer (ViT)-based learning pipeline named *CL-ViT* to detect and track cavelines by underwater robots for vision-based navigation. CL-ViT learning pipeline has two important features: **(i)** robustness to noise and image distortions [11]; and **(ii)** generalized model adaptation to data from new locations. Furthermore, identifying the ground plane, nearby obstacles, and navigational aids as well as human divers is critical for safe robotic operations inside underwater caves. Despite state-of-the-art (SOTA) detection performance, the proposed models are computationally demanding and do not run real-time on edge devices available onboard AUVs.

Fig. 1: A BlueROV2 is operating inside an underwater cave system by following its *caveline* in Orange Grove, FL.

In this paper, we analyze several distinct models, deploy mobile models on various edge AI accelerators, and compare their computational performance. Specifically, we focus on three models: **(i)** U-Net [12], a classic model for image segmentation, **(ii)** CaveSeg [13], a SOTA model developed for underwater cave segmentation, and (**iii**) YOLOv8 [14], one of the latest models from the YOLO family. We fine-tune these models for an underwater cave exploration application and deploy them on four edge platforms – Raspberry Pi-4, Intel™ Neural Compute Stick, Google Edge TPU, and Nvidia™ Jetson Nano – each with unique hardware characteristics that require different deployment strategies.

Application Scenario. Robotic exploration of underwater caves presents unique challenges, including navigating through narrow passages, avoiding obstacles, and making real-time operational decisions based on the surroundings objects and scene geometry. The lightweight cave segmentation models explored in this paper will enable AUVs to parse semantic information for detecting caveline and other navigation markers, avoiding obstacles, and maintaining interaction with companion divers if necessary. We will deploy our developed models onboard AUVs and investigate their utility in autonomous underwater cave exploration and mapping applications.

II. Related Work

A. Object Detection & Segmentation in Underwater Imagery

The scientific literature on subsea visual servoing explore the challenges and capabilities of underwater robots for scene understanding and safe navigation [7]. The fundamental task for visually-guided AUVs is to perceive the environment via cameras and other optical sensors, and then identify region-of-interests in the scene to plan effective navigational decisions [15]. Various data-driven models have been developed for vision tasks such as fast visual search [16], [17], visual enhancement [18], salient object localization [19], and oceanic resource monitoring [20], [21]. For instance, Koreitem *et al.* [16] developed an informed visual navigation system that first learns a similarity operator between the scene and a given exemplar image, and plans the visual searching path accordingly in an unconstrained environment. In contrast, model-free approaches are often better suited for autonomous exploratory applications [22]. Girdhar *et al.* [23] designed a low-dimensional semantic descriptor that encodes the environmental observations, referred to as topics, for safe autonomous control of AUVs. More recently, Modasshir *et al.* combined a deep learning-based classifier with Visual Inertial Odometry (VIO) to classify and count coral population density, via generating semantic maps [24] and volumetric models [25]. Additionally, Mohammadi *et al.* [26] developed three models for caveline detection: Vision Transformer, YOLOv8n, and U-Net; the models were deployed on edge devices and optimized for resource-constrained underwater robots in terms of energy efficiency and low latency.

B. Object Detection & Tracking on Edge Devices

In recent years, machine learning techniques for segmentation and object detection have been applied across various fields, including healthcare [27], [28], autonomous vehicles [29], space exploration [30], and manufacturing [31]. However, real-time systems, particularly those deployed on edge devices, face challenges such as latency, power dissipation, and energy consumption. Consequently, there has been a significant focus on developing tiny machine learning systems that are optimized for use on edge devices. In recent years, various TinyML techniques [32] and lightweight deep visual models [33] have been introduced such as SqueezeNet [34], MobileNet [35]–[37], ShuffleNet [38], [39], PeleeNet [40], MnasNet [41], Once-for-All (OFA) [42], GhostNet [43], MobileVit [44], and more. Iandola *et al.* [34] proposed a lightweight model called *SqueezeNet*, reaching AlexNet-level accuracy on ImageNet with $50\times$ fewer parameters and less than $0.5\,\text{MB}$ memory requirement. By employing streamlined architecture with depth-wise separable convolutions, Howard *et al.* [35] introduced the family of *MobileNet* models that are ideal for single-board embedded platforms. The successor MobileNetv2 [36] uses inverted residual structures to reduce computations, while MobileNetv3 [37] adopts a platform-aware automated neural architecture search in hierarchical search space along with NetAdapt [45], which further reduces the components of the network.

On the other hand, Zhang *et al.* [38] used the ResNet block coupling with innovations to devise *ShuffleNet*, which is compatible with mobile devices. ShuffleNetv2 [39] further improves the speed and accuracy by adopting a direct metric (speed) rather than indirect metrics like FLOPs. Moreover, Wang *et al.* [40] proposed a *PeleeNet* model by adopting an assortment of computation-conserving methods, making a compelling lightweight network. Tan *et al.* [41] presented a novel mobile CNN-based model using an automated neural architecture search approach. Besides, Cai *et al.* [42] introduced *OFA*, a lightweight network that can meet different hardware requirements. Furthermore, by merging the features of CNN and ViT, Mehta *et al.* [44] presented a lightweight and versatile vision transformer called *MobileVit* for edge devices. Previous works have also explored the deployment of these models on edge AI accelerators [46]–[49].

III. Methodology

A. Dataset Preparation

For data-driven training and evaluation on edge devices, we utilize visual data curated from three geographically diverse underwater cave systems: Devil's system in Florida, US; Dos Ojos Cenote in Quintana Roo, Mexico; and Cueva del Agua in Murcia, Spain [10]. The dataset introduced in [13] contains 3350 images with pixel-level annotations for the following 13 semantic objects: caveline, first/immediate obstacle layer, second/successive obstacle layer, obstacle-free open space, ground plane, scuba divers, caveline-attached rocks, navigation aids (arrows, reels, and cookies), and cave ornaments (stalactites, stalagmites, and columns). Figure 2 shows three sample images and corresponding labels from the dataset.

Among these categories, the caveline serves as the primary navigation guide, marking a continuous route toward the entrance/exit of a cave segment. Other navigation tools such as arrows and cookies are attached to the caveline to indicate the

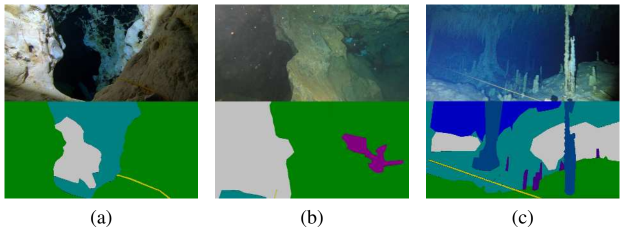


Fig. 2: Sample images and corresponding labels collected from three underwater cave systems in (a) Devil's Spring system, FL, USA; (b) Cueva del Agua, Murcia, Spain; and (c) Dos Ojos Cenote, QR, Mexico – are shown in each column.

nearest exit and the presence of other human divers, respectively. The caveline is securely fastened on rocks, referred to as attachment rocks, particularly found at corners and junctions where the caveline changes direction. The caveline and other navigation markers are annotated in yellow-red color tones to maximize visual contrast against the obstacles in background.

Open passages, obstacles, and the ground plane are crucial for path planning and autonomous navigation. The dataset categorizes the obstacles into first layer (green) and second layer (cyan) so that an AUV can anticipate its path and efficiently maneuver toward safe, open areas.

Human scuba divers are a critical consideration for safe cave exploration. Standard cave diving practices require the divers' route to the surface to remain unobstructed. Hence, an AUV would learn to detect scuba divers present in the scene and act accordingly, such as dimming the lights and yielding to other divers to ensure a clear path for their exit.

B. Targeted ML Models

1) YOLOv8: YOLOv8 [14] is a prominent model from the YOLO family, known for its impressive performance across various machine learning tasks, including object detection, image classification, and instance segmentation. Developed by Ultralytics, YOLOv8 is available in five different sizes, ranging from nano to x-large. The nano size is optimized for edge and mobile devices, while the x-large size is designed for high accuracy on large servers. Given the constraints of underwater robots, we selected YOLOv8n, which has only 3.4M parameters and is suitable for resource-constrained edge devices. For our application, we employed transfer learning, starting with a YOLOv8n model pre-trained on the COCO dataset and fine-tuning it for cave exploration using the targeted dataset.

2) U-Net: U-Net [12] is a convolutional neural network (CNN) architecture designed for semantic segmentation tasks. It comprises three main parts: a contracting path (encoder), a bottleneck layer, and an expansive path (decoder). The contracting path is similar to that of a typical CNN. It includes blocks with two 3×3 convolutions (unpadded), each followed by a rectified linear unit (ReLU) and a 2×2 max pooling operation with stride 2 for downsampling. With each downsampling step, the number of feature channels doubles. At the center of U-Net, the bottleneck layer captures the most critical features while preserving spatial information, as it lacks pooling layers. The expansive path involves upsampling the feature map at each step, followed by a 2×2 convolution ("up-convolution") that halves the number of feature channels. This is concatenated with the cropped feature map from the contracting path, followed by two 3×3 convolutions, each with a ReLU. Cropping is necessary due to the loss of border pixels during convolution. The final layer uses a 1×1 convolution to map each feature vector to the desired number of classes. U-Net has 23 conv layers, resulting in over 31M parameters.

Due to the resource and power constraints of both robots and edge devices, we optimized the network by replacing the encoder with MobileNetV2. Additionally, we reduced the number of kernels per layer in the bottleneck and decoder sections. These modifications significantly decreased the network's parameters and MAC operations, resulting in a model with only 4.33 M parameters.

3) CaveSeg: The CaveSeg model used in this paper is inspired by the model developed in [13]. This model uses a light transformer backbone and a multi-level pyramid head module for feature extraction. Each RGB image is pre-processed into 4×4 patches, arranged into a linear representation of tokens, and then fed into the transformer network. The four-stage backbone applies multi-head self-attention [50] on the feature tokens using a windowed and shifted windowed module [51]. Each stage combines the patches with 2×2 neighboring pixels and doubles the linear representation to maintain the number of tokens. The window shifting technique is applied after each transformation to improve feature extraction. The pyramid head, inspired by [52], enhances higher-level features using multi-scale convolution blocks [53]. The features extracted by the backbone and the head module are upsampled and aggregated into a hierarchical map representation [54]. These features propagate into a dense convolution module that estimates the semantic class for each pixel. The hyperparameters are empirically tuned to achieve the best trade-off between accuracy and computational cost. This model is larger compared to the other two, with 35M parameters.

IV. Experimental Setup

In this study, we deploy and evaluate our ML models on a range of edge devices, including the Raspberry Pi, Intel Movidius Neural Compute Stick 2 (NCS2), Nvidia Jetson Nano, and Google Coral TPU, as shown in Figure 3. These devices are widely used for various ML tasks, such as image classification [48], semantic segmentation [26], and natural language processing [47].

1) Raspberry Pi-4: The Raspberry Pi-4 board features a Broadcom BCM2711 quad-core ARM Cortex-A72 CPU, with operating frequencies that vary by model: 1.5 GHz for the 2 GB and 4 GB RAM versions, and 1.8 GHz for the 8 GB RAM version. Available in multiple configurations, the Raspberry Pi-4 offers 2 GB, 4 GB, and 8 GB of LPDDR4 RAM. The device requires a 5V USB-C power supply, with the necessary current rating depending on the intended use and peripherals.

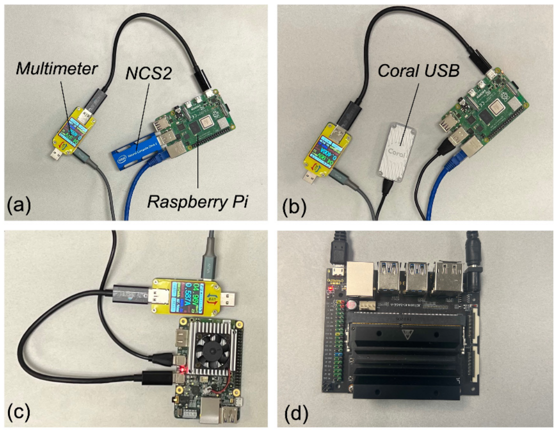


Fig. 3: Experimental setup. (a) Pi + NCS2 (b) Pi + Coral TPU Coral Dev board (d) Jetson Nano.

2) Intel Neural Compute Stick: The Intel Neural Compute Stick 2 (NCS2) is powered by the Intel Movidius-X Vision Processing Unit (VPU), which is featured by 16 programmable cores and a dedicated neural compute engine. The NCS2 operates at a base frequency of 700 MHz and includes 4 GB of RAM. Intel provides the OpenVINO library, available in both Python3 and C, to streamline the deployment of ML models on the NCS2. This library includes a model optimizer that converts models into a format suitable for NCS2 deployment. Following this, the OpenVINO inference engine API can be used to evaluate latency and power efficiency during inference.

3) Google Coral TPU: The Edge TPU serves as a co-processor on Coral's Dev Board, working alongside the NXP i.MX 8M system-on-chip (SoC) within their system-on-module (SoM) architecture. To run on the Coral Edge TPU, models must be converted to TensorFlow Lite format and quantized to 8-bit integer types. Coral also offers a USB accelerator that can be integrated with a CPU as a co-processor. With a power consumption of around two watts, this accelerator is well-suited for low-power environments.

4) Nvidia Jetson Nano: The Jetson Nano is a modular computer built around the Tegra X1 SoC, featuring a quad-core ARM A57 processor and four 32-CUDA core processing units. It comes with 4 GB of memory. Nvidia™ provides two operating modes for the Jetson Nano: in the low power mode (Jetson-L), only two cores of the ARM A57 are active, with a clock frequency of 0.9 GHz, and the GPU runs at 0.64 GHz. In the high power mode (Jetson-H), all four cores of the ARM A57 processor are active at a 1.5 GHz frequency, while the GPU operates at 0.92 GHz. The Jetson Nano utilizes NVIDIA TensorRT as its primary tool for ML model optimization.

V. Performance Evaluation

A. Deployment Considerations

After training and fine-tuning the ML models for the underwater cave segmentation application, as outlined in the previous section, we focus on deploying them across various edge platforms. Using PyTorch for training, we exported the models in ONNX format as an intermediary step. We then evaluated the performance of these models on well-known edge AI accelerators available in the market. Specifically, we investigated two experimental setups: *(1)* USB accelerators, where we compared the Intel NCS2 (Fig. 3a) with the Coral TPU USB accelerator (Fig. 3b), and *(2)* Development Boards, where we evaluated the Coral Edge TPU Dev Board (Fig. 3c) against the Nvidia Jetson Nano (Fig. 3d). The USB accelerators were integrated as co-processors with the Raspberry Pi.

Different configurations are required to run the models on each of these edge devices. The Raspberry Pi-4 and Coral TPU utilize TFLite models, with 32-bit floating-point (FP32) precision for the Raspberry Pi-4 and 8-bit integer precision for the Coral TPU. The Jetson platform uses TensorRT models with FP16 precision. Additionally, we integrated the NCS2 accelerator as a co-processor with the Raspberry Pi-4, using OpenVINO 2021 to convert the ONNX models into the required format, employing FP16 operations.

As mentioned in Section III, the CaveSeg model is larger than the other models and includes layers that are not supported by edge devices. Consequently, we deployed this model exclusively on the Raspberry Pi as the baseline edge device. Additionally, For the other models, due to the Edge TPU's inability to fully accommodate the entire architecture with larger inputs, we resized the input dimensions for the Yolo and U-Net models to 192×192 and 128×128, respectively. It is important to note that with larger input sizes, some operations are offloaded from the TPU to the CPU, which significantly impacts system performance. This modification affects the models' accuracy, which is discussed in the following section.

B. Performance Metrics

Here, we evaluate the performance of the ML models using a commonly used metric: Intersection Over Union (IOU). IOU assesses the accuracy of object localization by calculating the proportion of overlapping area between predicted and actual labels. It is defined as:

$$\mathtt{IoU} = \frac{\mathtt{Area\ of\ Overlap}}{\mathtt{Area\ of\ Union}} = \frac{\mathtt{TP}}{\mathtt{TP+FP+FN}} \quad (1)$$

where, TP and FP represent true positives and false positives, indicating the correct and incorrect selection of the class of interest pixels, resepectively. FN represents false negatives, indicating the pixels that are part of the target class in the ground truth but are missed by the prediction.

We compared the results and sizes of our proposed models with previous work, as shown in Table I. The YoLo outperforms other models with an IoU of 72.5%. The proposed U-Net model follows closely, achieving an IoU of 65.55%. Additionally, as illustrated in Table I, previous models, including CaveSeg, are significantly larger than Yolo and the proposed U-Net model, making them impractical for deployment on edge devices. When comparing the sizes of Yolo and U-Net to the CaveSeg model—which is already smaller than previous studies—they are $10.29\times$ and $8.08\times$ smaller, respectively.

TABLE I: Quantitative performance comparisons of all models in terms of IoU metrics and number of parameters.

Model	# of Parameters	IoU (%)
FastFCN	66 M	38.86
DeepLabV3+	42 M	38.46
Segmenter	98 M	30.81
Segformer	82 M	35.36
Swin Transformer	120 M	48.11
CaveSeg	35 M	40.22
U-Net	4.33 M	65.55
Yolo	**3.4**	**72.5**

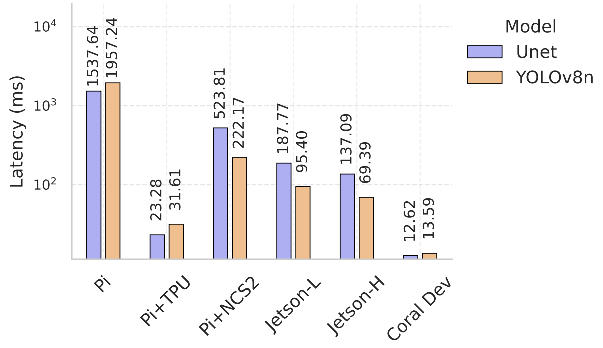


Fig. 4: Inference latency for all models and devices.

As mentioned in Section IV.A, deploying the Yolo and U-Net models on the Edge TPU requires reducing their input sizes to 192×192 and 128×128, respectively. This reduction impacts the models' accuracy. According to the results, the Yolo model running on the Edge TPU achieves an IoU of 69.01%, while the U-Net model on the same device reaches 62.76%. Compared to the original models, this represents a reduction in IoU of 3.49% for Yolo and 2.72% for U-Net. Despite this decrease, these models still offer significantly better accuracy than previous works. Table III provides the U-Net and YOLOv8n segmentation outputs for three sample images from the dataset.

C. Inference Latency Measurement

To evaluate inference latency, we first conducted one hundred inference operations for each model on each edge platform and measured the total latency. We then calculated the average inference time per image. Given the size and complexity of the CaveSeg model, we only ran it on the Raspberry Pi. The latency for processing a single input with the CaveSeg model is 35 seconds, which is significantly longer than the latency of the other two models on the same device.

Figure 4 displays the inference latency results for all platforms running YOLOv8n and U-Net. The results show that all edge accelerators offer significant speedup compared to the baseline Raspberry Pi 4 CPU for both models. The Coral Dev Board achieved the fastest inference times, with 12.62 ms per inference for U-Net and 13.59 ms for Yolo. Compared to the baseline Raspberry Pi, this represents a 144× speedup for Yolo and a 121.84× speedup for U-Net, highlighting the effectiveness of the Coral Dev Board for this application. For both Yolo and U-Net, the order of inference latency from shortest to longest is as follows: Coral Dev Board, Raspberry Pi + Coral USB, Jetson high power mode, Jetson low power mode, Raspberry Pi + NCS2, and Raspberry Pi.

TABLE II: FPS rates for models running on Nvidia™ A100 GPU, Coral Devboard, and Coral USB as a co-processor.

Model	Device	FPS
FastFCN	Nvidia™ A100 GPU	16.89
DeepLabV3+	Nvidia™ A100 GPU	15.04
Segmenter	Nvidia™ A100 GPU	13.73
Segformer	Nvidia™ A100 GPU	10.92
Swin Transformer	Nvidia™ A100 GPU	12.39
CaveSeg	Nvidia™ A100 GPU	19.78
U-Net	**Coral Dev**	**79.24**
	Pi + Coral USB	42.95
YOLOv8n	Coral Dev	73.58
	Pi + Coral USB	31.64

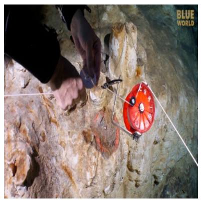
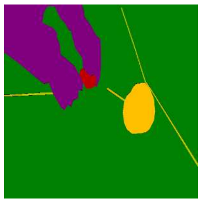
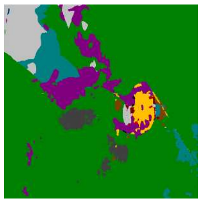
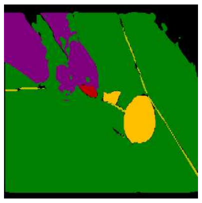
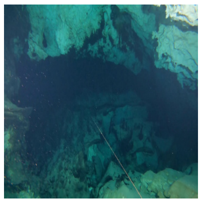
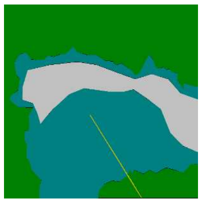
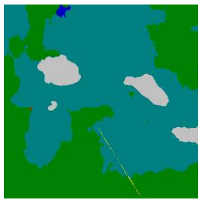
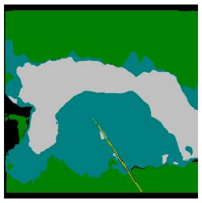
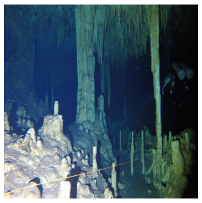
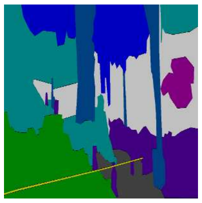
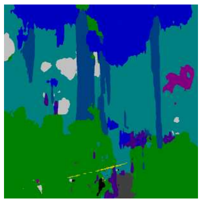
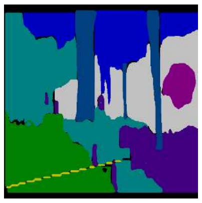
1) USB Accelerators: For the USB accelerators, both the NCS2 and Coral USB accelerators demonstrated improvements over the baseline Raspberry Pi 4 for both the Yolo and U-Net models. Specifically, running Yolo on the Coral USB resulted in a 61.91× speedup, while the NCS2 showed an 8.81× improvement. For the U-Net model, the Coral USB and NCS2 achieved a 66.05× and 2.93× improvement, respectively. These results indicate that integrating compact USB accelerators into the robot can significantly enhance system performance.

2) Development Boards: Both development boards provide significant speedups compared to the Raspberry Pi. For the Yolo model, we observe a 20.52× improvement in the Jetson low power mode and a 28.21× improvement in the Jetson high power mode over the baseline device. The Coral Dev Board offers an even greater improvement, with a 144.02× increase in performance. For the U-Net model, the Jetson board shows an 8.19× improvement in low power mode and an 11.22× improvement in high power mode, while the Coral Dev Board achieves a 121.84× improvement over the baseline device.

Table II presents a comparison between our proposed models and other studies in terms of frame per second (FPS) ratio. Except for Yolo and the proposed U-Net model, none of the models can be deployed on edge devices due to resource limitations. Therefore, we deployed the larger models on a Nvidia™ A100 GPU, while Yolo and U-Net were deployed on the Coral Dev Board, which demonstrated superior performance in terms of latency compared to other edge devices.

As listed in Table II, U-Net running on the Coral Dev Board achieves the highest frame rate at 79.24 FPS, followed closely by Yolo on the same device with 73.58 FPS. We also include the FPS ratio for running mobile models on the Coral USB as a co-processor, achieving 42.95 FPS for YOLOv8n and 31.64 FPS for U-Net. These results demonstrate that both models are well-suited for real-time underwater cave segmentation applications. Particularly, since many robots are equipped with their own processors, integrating a co-processor like the Coral USB makes real-time applications feasible. Among the larger models running on the Nvidia™ A100 GPU, CaveSeg achieves the highest FPS of 19.78.

TABLE III: Qualitative analysis indicates that YOLOv8n outperforms the U-Net model in pixel accuracy and localization. The results for CaveSeg are not shown, as our focus is on mobile models that are suitable for deployment on edge devices.

Original Image	Label	**Model Outputs**	
		U-Net	YOLOv8n
			
			
			

D. Inference Power Measurement

To assess power dissipation during inference for the various models deployed on Raspberry Pi-4, NCS2, and Edge TPU devices, we used the MakerHawk UM34C USB multimeter, as depicted in Figure 3a. For the Jetson Nano, we utilized its internal sensors to monitor CPU and GPU power dissipation. We ran each model for five minutes, with power dissipation measured at a rate of one sample per second. Figure 5 shows the dynamic power measurements for both models across all platforms. The Coral Dev Board exhibits the lowest power dissipation compared to the other devices. Additionally, the figure indicates that power dissipation is comparable for both models on all platforms except for Pi+Coral TPU.

1) USB Accelerators: For the USB accelerators, both the NCS2 and Coral USB accelerator generally exhibit higher power dissipation compared to the baseline Raspberry Pi, with the exception of the Coral USB when running YOLOv8n. In this case, there is a $1.53\times$ reduction in power dissipation. Additionally, when comparing the two USB accelerators, the Coral USB dissipates $2.52\times$ less power for running Yolo and $1.43\times$ less power for U-Net.

2) Development Boards: Compared to the Raspberry Pi, the Coral Dev Board offers a $2.67\times$ reduction in power dissipation for the Yolo model and a $2.71\times$ reduction for the U-Net model. In contrast, the Jetson Nano shows higher power dissipation compared to the Raspberry Pi. Specifically, in low power mode, the Jetson Nano's power dissipation increases by $1.32\times$ for the Yolo model and $1.39\times$ for the U-Net model. In high power mode, the increases are $2.77\times$ for the Yolo model and $2.5\times$ for the U-Net model.

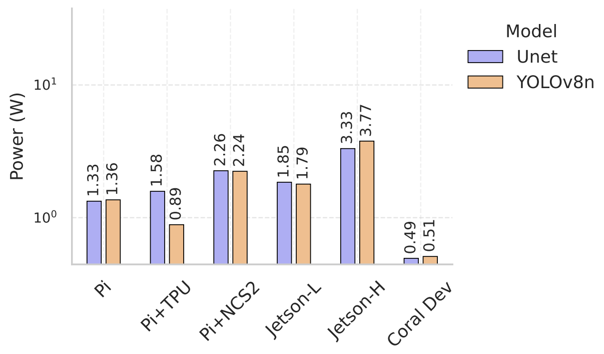


Fig. 5: Dynamic power comparison for all models and devices.

E. Inference Energy Measurement

Figure 6 provides a comparative analysis of inference energy results. All edge AI accelerators offer a significant improvement in inference energy efficiency compared to the baseline Raspberry Pi. For instance, the Coral Dev board outperforms other edge devices by consuming only $6.23\,mJ$ for running the U-Net model and $6.96\,mJ$ for the Yolo model.

1) USB Accelerators: The results show that the Coral USB consumes $95.14\times$ less energy for running the Yolo model and $55.71\times$ less for the U-Net model compared to the baseline Raspberry Pi. For the NCS2, the energy reduction is $5.36\times$ for the Yolo model and $1.73\times$ for the U-Net model when compared to the Raspberry Pi. Additionally, the Coral USB consumes $32.15\times$ and $17.76\times$ less energy than the NCS2 for running U-Net and Yolo models, respectively.

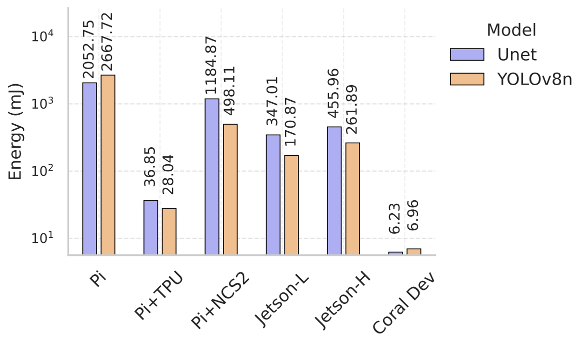


Fig. 6: Dynamic energy comparison for all models and devices.

2) Development Boards: Compared to the Raspberry Pi, the Coral Dev board achieves a $338.29\times$ reduction in inference energy for the Yolo model and a 329.49 reduction for the U-Net model. When comparing the Jetson-low and Jetson-high modes to the Raspberry Pi, the Jetson provides a $15.61\times$ and $10.19\times$ improvement in energy efficiency for the Yolo model, respectively. For the U-Net model, there is a $5.91\times$ improvement in energy efficiency with Jetson-low and a $4.5\times$ improvement with Jetson-high.

VI. Conclusion and Future Work

This paper addresses the challenge of real-time underwater cave segmentation, a critical task for autonomous underwater robots engaged in cave exploration and mapping. Given the limited connectivity of AUVs to the surface and the constraints of space, relying on cloud services or powerful GPUs for running deep learning-based segmentation models is not practical. Therefore, our study explores the trade-offs involved in deploying two different mobile segmentation models on various edge platforms, with a focus on evaluating accuracy, latency, power, and energy consumption.

Our experimental results reveal that the anticipated trade-offs are not always accurate. Contrary to expectations, larger models did not consistently deliver higher accuracy at the expense of increased inference times and energy consumption. Modern architectures like YOLOv8n, as well as specialized models such as U-Net for semantic segmentation, performed better than much larger models like encoder-decoder transformers. Our analysis uncovered several key insights: **(i)**YOLOv8n achieved notably high IoU values (72.5%), surpassing models that are 10 times larger, **(ii)** the FPS rate, crucial for real-time operation, is significantly influenced by the choice of edge platform. For example, running the U-Net model on the Coral Dev Board achieved an underwater cave segmentation speed of 79.24 FPS, meeting real-time criteria. Similarly, the YOLOv8n model, which demonstrated the highest accuracy, achieved 73.58 FPS on the same device. It is important to consider that many robots use their own processors and boards. In such cases, incorporating USB accelerators as co-processors can enhance real-time performance for computation-intensive tasks like semantic segmentation. Our findings indicate that a USB accelerator like the Coral USB also meets real-time requirements, delivering speeds of 42.95 FPS and 31.64 FPS for U-Net and YOLOv8n, respectively.

The comprehensive quantitative analyses presented in this paper serve as a valuable guide for making informed design decisions and managing trade-offs when integrating our cave segmentation systems with underwater robots for cave exploration and mapping missions. In future research, we plan to explore the practical implications of these trade-offs, including considerations like battery life, real-world accuracy, and the thermal effects resulting from computing power dissipation.

Acknowledgment

This research has been supported in part by the NSF grants 1943205, 2024741, 2340249, and the UF research grant #132763. The authors also acknowledge the help of the Woodville Karst Plain Project (WKPP), El Centro Investigador del Sistema Acuífero de Quintana Roo A.C. (CINDAQ), Global Underwater Explorers (GUE), and Ricardo Constantino, Project Baseline in collecting data, providing access to underwater caves, and mentorship in underwater cave exploration. Also, we thank Halcyon Dive Systems for their equipment support, Teledyne FLIR LLC for cameras, and KELDAN GmbH for underwater lighting solutions.

References

[1] M. J. Lace and J. E. Mylroie, “The biological and archaeological significance of coastal caves and karst features,” in *Coastal Karst Landforms*, pp. 111–126, Springer, 2013.

[2] D. Ford and P. Williams, *Introduction to Karst*, ch. 1, pp. 1–8. John Wiley & Sons, Ltd, 2007.

[3] L. Potts, P. Buzzacott, and P. Denoble, “Thirty years of american cave diving fatalities,” *Diving Hyperbaric Medicine*, vol. 46, pp. 150–154, 2016.

[4] P. L. Buzzacott, E. Zeigler, P. Denoble, and R. Vann, “American cave diving fatalities 1969-2007,” *International Journal of Aquatic Research and Education*, vol. 3, no. 2, p. 7, 2009.

[5] B. Joshi, M. Xanthidis, S. Rahman, and I. Rekleitis, “High definition, inexpensive, underwater mapping,” in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1113–1121, 2022.

[6] K. Richmond, C. Flesher, N. Tanner, V. Siegel, and W. C. Stone, “Autonomous exploration and 3-D mapping of underwater caves with the human-portable SUNFISH® AUV,” in *Global Oceans 2020: Singapore–US Gulf Coast*, pp. 1–10, IEEE, 2020.

[7] M. J. Islam, A. Quattrini Li, Y. A. Girdhar, and I. Rekleitis, “Computer vision applications in underwater robotics and oceanography,” pp. 173–204, CRC Press, 2024.

[8] S. Exley, *Basic cave diving: A blueprint for survival*. Cave Diving Section of the National Speleological Society, 1986.

[9] K. Siddiqi, S. Bouix, A. Tannenbaum, and S. W. Zucker, “Hamilton-jacobi skeletons,” *International Journal of Computer Vision*, vol. 48, no. 3, pp. 215–231, 2002.

[10] B. Yu, R. Tibbetts, T. Barna, A. Morales, I. Rekleitis, and M. J. Islam, “Weakly Supervised Caveline Detection For AUV Navigation Inside Underwater Caves,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 9933–9940, IEEE, 2023.

[11] M. J. Islam, Y. Xia, and J. Sattar, “Fast Underwater Image Enhancement for Improved Visual Perception,” *IEEE Robotics and Automation Letters (RA-L)*, vol. 5, no. 2, pp. 3227–3234, 2020.

[12] O. Ronneberger, P. Fischer, and T. Brox, “U-net: Convolutional networks for biomedical image segmentation,” in *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, pp. 234–241, Springer, 2015.

[13] A. Abdullah, T. Barua, R. Tibbetts, Z. Chen, M. J. Islam, and I. Rekleitis, “Caveseg: Deep semantic segmentation and scene parsing for autonomous underwater cave exploration,” in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3781–3788, 2024.

[14] "Ultralytics: YOLOv8 Docs, official website = https://docs.ultralytics.com/." Accessed: 2023-09-01.

[15] M. J. Islam, C. Edge, Y. Xiao, P. Luo, M. Mehtaz, C. Morse, S. S. Enan, and J. Sattar, "Semantic Segmentation of Underwater Imagery: Dataset and Benchmark," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020.

[16] K. Koreitem, F. Shkurti, T. Manderson, W.-D. Chang, J. C. G. Higuera, and G. Dudek, "One-shot informed robotic visual search in the wild," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5800–5807, IEEE, 2020.

[17] M. J. Islam, R. Wang, and J. Sattar, "SVAM: Saliency-guided Visual Attention Modeling by Autonomous Underwater Robots," in *Robotics: Science and Systems (RSS)*, (NY, USA), 2022.

[18] J. Zhu, S. Yu, L. Gao, Z. Han, and Y. Tang, "Saliency-Based Diver Target Detection and Localization Method," *Mathematical Problems in Engineering*, vol. 2020, 2020.

[19] M. J. Islam, P. Luo, and J. Sattar, "Simultaneous Enhancement and Super-Resolution of Underwater Imagery for Improved Visual Perception," in *Robotics: Science and Systems (RSS)*, 2020.

[20] M. Modasshir, A. Quattrini Li, and I. Rekleitis, "Deep neural networks: a comparison on different computing platforms," in *Canadian Conference on Computer and Robot Vision (CRV)*, (Toronto, ON, Canada), pp. 383–389, May 2018.

[21] T. Manderson, J. C. G. Higuera, R. Cheng, and G. Dudek, "Vision-based Autonomous Underwater Swimming in Dense Coral for Combined Collision Avoidance and Target Selection," in *IEEE/RSJ Int. Conference on Intelligent Robots and Systems (IROS)*, pp. 1885–1891, 2018.

[22] Y. Girdhar and G. Dudek, "Modeling Curiosity in a Mobile Robot for Long-term Autonomous Exploration and Monitoring," *Autonomous Robots*, vol. 40, no. 7, pp. 1267–1278, 2016.

[23] Y. Girdhar, P. Giguere, and G. Dudek, "Autonomous Adaptive Exploration using Realtime Online Spatiotemporal Topic Modeling," *Int. Journal of Robotics Research (IJRR)*, vol. 33, no. 4, pp. 645–657, 2014.

[24] M. Modasshir, S. Rahman, O. Youngquist, and I. Rekleitis, "Coral Identification and Counting with an Autonomous Underwater Vehicle," in *IEEE International Conference on Robotics and Biomimetics (ROBIO)*, (Kuala Lumpur, Malaysia, (Finalist of T. J. Tarn Best Paper in Robotics)), pp. 524–529, Dec. 2018.

[25] M. Modasshir, S. Rahman, and I. Rekleitis, "Autonomous 3D Semantic Mapping of Coral Reefs," in *12th Conference on Field and Service Robotics (FSR)*, (Tokyo, Japan), pp. 365–379, Aug. 2019.

[26] M. Mohammadi, S.-E. Huang, T. Barua, I. Rekleitis, M. J. Islam, and R. Zand, "Caveline detection at the edge for autonomous underwater cave exploration and mapping," in *International Conference on Machine Learning and Applications (ICMLA)*, pp. 1392–1398, 2023.

[27] A. B. Rajeoni, B. Pederson, A. Firooz, H. Abdollahi, A. K. Smith, D. G. Clair, S. M. Lessner, and H. Valafar, "Vascular system segmentation using deep learning," *Artificial Intelligence: Machine Learning, Convolutional Neural Networks and Large Language Models*, vol. 1, p. 85, 2024.

[28] A. Bagheri Rajeoni, B. Pederson, D. G. Clair, S. M. Lessner, and H. Valafar, "Automated measurement of vascular calcification in femoral endarterectomy patients using deep learning," *Diagnostics*, vol. 13, no. 21, p. 3363, 2023.

[29] M. Mohammadi, M. Morsali, S. Tabrizchi, B. C. Reidy, A. Roohi, S. Angizi, and R. Zand, "Pixelprune: Optimizing aiot vision systems via in-sensor segmentation and adaptive data transfer," *Authorea Preprints*, 2024.

[30] Y. Dai, T. Zheng, C. Xue, and L. Zhou, "Segmarsvit: Lightweight mars terrain segmentation network for autonomous driving in planetary exploration," *Remote Sensing*, vol. 14, no. 24, p. 6297, 2022.

[31] M. K. Ferguson, A. Ronay, Y.-T. T. Lee, and K. H. Law, "Detection and segmentation of manufacturing defects with convolutional neural networks and transfer learning," *Smart and sustainable manufacturing systems*, vol. 2, 2018.

[32] P. P. Ray, "A review on tinyml: State-of-the-art and prospects," *Journal of King Saud University-Computer and Information Sciences*, vol. 34, no. 4, pp. 1595–1623, 2022.

[33] S. S. A. Zaidi, M. S. Ansari, A. Aslam, N. Kanwal, M. Asghar, and B. Lee, "A survey of modern deep learning based object detection models," *Digital Signal Processing*, vol. 126, p. 103514, 2022.

[34] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size," *arXiv preprint arXiv:1602.07360*, 2016.

[35] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[36] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4510–4520, 2018.

[37] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, *et al.*, "Searching for mobilenetv3," in *IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 1314–1324, 2019.

[38] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 6848–6856, 2018.

[39] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, "Shufflenet v2: Practical guidelines for efficient cnn architecture design," in *Proceedings of the European conference on computer vision (ECCV)*, pp. 116–131, 2018.

[40] R. J. Wang, X. Li, and C. X. Ling, "Pelee: A real-time object detection system on mobile devices," *Advances in neural information processing systems*, vol. 31, 2018.

[41] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, "Mnasnet: Platform-aware neural architecture search for mobile," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 2820–2828, 2019.

[42] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, "Once-for-all: Train one network and specialize it for efficient deployment," *arXiv preprint arXiv:1908.09791*, 2019.

[43] K. Han, Y. Wang, Q. Tian, J. Guo, C. Xu, and C. Xu, "Ghostnet: More features from cheap operations," in *IEEE/CVF conference on computer vision and pattern recognition*, pp. 1580–1589, 2020.

[44] S. Mehta and M. Rastegari, "Mobilevit: light-weight, general-purpose, and mobile-friendly vision transformer," *arXiv preprint arXiv:2110.02178*, 2021.

[45] T.-J. Yang, A. Howard, B. Chen, X. Zhang, A. Go, M. Sandler, V. Sze, and H. Adam, "Netadapt: Platform-aware neural network adaptation for mobile applications," in *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 285–300, 2018.

[46] B. C. Reidy, M. Mohammadi, M. E. Elbtity, and R. Zand, "Work in progress: Real-time transformer inference on edge ai accelerators," in *2023 IEEE 29th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pp. 341–344, 2023.

[47] B. C. Reidy, M. Mohammadi, M. E. Elbtity, and R. Zand, "Efficient deployment of transformer models on edge tpu accelerators: A real system evaluation," in *Architecture and System Support for Transformer Models (ASSYST ISCA)*, 2023.

[48] M. Mohammadi, H. Smith, L. Khan, and R. Zand, "Facial expression recognition at the edge: Cpu vs gpu vs vpu vs tpu," in *Proceedings of the Great Lakes Symposium on VLSI 2023*, GLSVLSI '23, (New York, NY, USA), p. 243–248, Association for Computing Machinery, 2023.

[49] H. Smith, J. Seekings, M. Mohammadi, and R. Zand, "Realtime facial expression recognition: Neuromorphic hardware vs. edge ai accelerators," in *2023 International Conference on Machine Learning and Applications (ICMLA)*, pp. 1547–1552, 2023.

[50] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[51] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin transformer: Hierarchical vision transformer using shifted windows," in *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 10012–10022, 2021.

[52] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, "Pyramid scene parsing network," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2881–2890, 2017.

[53] T. Xiao, Y. Liu, B. Zhou, Y. Jiang, and J. Sun, "Unified perceptual parsing for scene understanding," in *Proceedings of the European conference on computer vision (ECCV)*, pp. 418–434, 2018.

[54] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2117–2125, 2017.