CONSTRUCTOR UNIVERSITY

YOLOvBub : Real Time Bubble Activity Detection

by

Stefan Dusnoki

Bachelor Thesis in Computer Science

Submission: May 3, 2025 Supervisor: Prof. Dmitry Kropotov

Constructor University | School of Computer Science and Engineering

Statutory Declaration

Family Name, Given/First Name	Dusnoki, Stefan
Matriculation number	30006235
Kind of thesis submitted	Bachelor Thesis

English: Declaration of Authorship

I hereby declare that the thesis submitted was created and written solely by myself without any external support. Any sources, direct or indirect, are marked as such. I am aware of the fact that the contents of the thesis in digital form may be revised with regard to usage of unauthorized aid as well as whether the whole or parts of it may be identified as plagiarism. I do agree my work to be entered into a database for it to be compared with existing sources, where it will remain in order to enable further comparisons with future theses. This does not grant any rights of reproduction and usage, however.

This document was neither presented to any other examination board nor has it been published.

German: Erklärung der Autorenschaft (Urheberschaft)

Ich erkläre hiermit, dass die vorliegende Arbeit ohne fremde Hilfe ausschließlich von mir erstellt und geschrieben worden ist. Jedwede verwendeten Quellen, direkter oder indirekter Art, sind als solche kenntlich gemacht worden. Mir ist die Tatsache bewusst, dass der Inhalt der Thesis in digitaler Form geprüft werden kann im Hinblick darauf, ob es sich ganz oder in Teilen um ein Plagiat handelt. Ich bin damit einverstanden, dass meine Arbeit in einer Datenbank eingegeben werden kann, um mit bereits bestehenden Quellen verglichen zu werden und dort auch verbleibt, um mit zukünftigen Arbeiten verglichen werden zu können. Dies berechtigt jedoch nicht zur Verwendung oder Vervielfältigung.

Diese Arbeit wurde noch keiner anderen Prüfungsbehörde vorgelegt noch wurde sie bisher veröffentlicht.

.....

Date, Signature

Contents

1 Abstract

As time goes by, the subsea technology industry continuously expands its operations, which necessitates additional funds, resources, and research into new technology. However, such growth often also brings negative aspects with it, with one concrete example being noise pollution. For each issue, there is at least one type of solution that is proposed and implemented, and in the case of underwater noise pollution, an efficient and low-cost proposition is the use of a bubble curtain. However, bubble curtains do not have inbuilt monitoring systems, despite the importance of keeping these systems running at maximum efficiency. Therefore, a gap in existing research unfolds.

We introduce YOLOvBub, a YOLOv10-based deep learning model specialized in detecting underwater bubble activity. Two additional accompaniments to our system include BubSET and BubSET-Testing, two datasets that facilitate the training and validation process, respectively.

Our approach yields reasonable results. The best performing model from our experiments reaches a mean average precision of 0.891 at an IoU threshold of 0.5, along with a precision value of 0.84 and a recall value of 0.88. The model also demonstrates a solid understanding of the provided material, with a large majority of the detections observed in the test footage having a confidence value of 0.9 and higher.

By fine-tuning the YOLOv10-M model for our specific task, shifting the model's focus more towards correct classification as opposed to bounding box placement and implementing a wide range of image augmentations, we create an original proposal for the issue, with the hopes of fostering further research into underwater object detection and computer vision as a whole.

Preamble: Context & Why This Matters

Industry expansion. As advances in the subsea technology domain continue to accelerate, the range of tasks in which these breakthroughs are applied develops at a similar pace. Studies have shown that industry sectors such as deep-sea mining, offshore wind, and submarine power cable laying have expanded in recent times. More specifically, the value of the global market for technologies used in deep sea mining was estimated at $650 million in 2020, and projections indicate a sharp increase to over $15 billion in the following 10 years [1]. The global offshore wind energy market exhibits a similar upward shift as mentioned previously. In 2023, more than 13,096 wind turbines in 319 projects brought the total capacity to 68,258 MW,

with the 6,326 MW deployed in 2023 representing the fourth largest annual deployment [2].

A further notable example comes from the British energy regulator Ofgem. Ofgem approved a $2.5 billion funding package for building approximately 200 kilometers of new subsea and underground cables. This package is part of the Eastern Green Link 1 (EGL1) project, which is designed to harness the power of the North Sea to reduce Britain's dependence on volatile international gas markets [3].

Environmental impact. Unfortunately, large-scale and rapid development as described above is not achieved without causing various long-term negative effects on the general environment. These repercussions are often direct results of industry-specific activities such as diving, mining, the maintenance of seaborne ships, and military activities, and they actively harm aquatic fauna and their habitat. Pertinent examples include mining contamination, habitat alteration and destruction, and noise pollution [4–6].

Environmental regulations. Efforts are underway to minimize these negative effects. Various solutions have been proposed, mainly in the form of regulatory mechanisms. For instance, the Environmental Impact Assessment (EIA) Directive adopted under European Union law defines an "EIA procedure", which is designed to use multiple metrics to measure the ramifications of a project. This assessment estimates both the direct and indirect impact the project in question would have on biodiversity, water, air, climate, and landscape [7, 8].

The evaluation includes three parties: The project developer, the competent authority, and the public. During the scoping stage, the competent authority informs the project developer of the information that must be provided. The environmental authorities, the concerned Member States, and the general public are then informed and consulted about the specifics of the project. Before reaching a verdict, the competent authority takes into account the previously held discussions before reaching a verdict. Finally, the general public is informed of the authority's decision, giving them the possibility to challenge the ruling [8]. These regulations are not limited in scope, as they are meant to address a wide variety of environmental issues.

Growing focus on noise pollution. Between the diverse environmental challenges that have emerged from the subsea industry, underwater noise pollution represents a unique subject of discussion, with recently introduced regulations placing special emphasis on the topic. Such laws are critical because numerous species of underwater fauna often rely on sound for communication and navigation [9], meaning that any kind of external

disturbance will have a negative impact on their delicate ecosystems. In an unprecedented move, the European Commission has introduced binding limits for underwater noise pollution in the Union, after years of advocacy from different organizations, such as the International Fund for Animal Welfare (IFAW) [10, 11].

Bubble curtains as a solution. One of the most practical methods used to combat the issue of underwater noise pollution during operations such as pile driving is the incorporation of a bubble curtain. In a general sense, a "bubble curtain" is a system in which compressed air or other gases are pushed through intentionally placed underwater pipes, with the ascent of the bubbles acting as a barrier for noise and therefore lowering the propagation of harmful sounds. Pile driving is particularly selected as an example, due to the fact that it is one of the loudest underwater activities associated with the subsea industry [12].

Lack of monitoring. Although well-validated hydrodynamic models aimed at predicting the effectiveness of sound attenuation have been described before in academic literature [13], few, if any, computational systems are capable of properly monitoring curtain efficiency. This oversight introduces an ideal opportunity for the creation of cutting-edge systems that measure the performance of bubble curtains in the realm of underwater noise mitigation. Our proposed approach utilizes state-of-the-art machine learning technology to address this challenge.

2 Introduction

Research gap. Our previous remark concerning the absence of studies on this matter is entirely abstract, as there are no mentions of the precise information that we are lacking. Therefore, we begin our introduction by addressing this subject.

First, we lack a comprehensive dataset that consists of annotated images which describe the particulars of what a bubble curtain is to a machine learning model. A possible solution could have been the dataset mentioned by Beelen et al. [13], however, to the best of our knowledge, this dataset was never released. Supplementary material for this research article, hosted on 4TU.ResearchData, only includes two non-annotated videos of bubble activity. As part of our efforts, we requested access to this dataset on 26 February 2025, but received no response.

We also used multiple search queries on Google, Google Scholar, and IEEE Xplore to verify the existence of any relevant datasets. Our searches

utilized the following queries:

- "bubble activity" AND ("paper" OR "machine learning" OR "dataset")
- "bubble curtain" AND ("paper" OR "machine learning" OR "dataset")
- "bubble curtain" AND "dataset"
- "bubble activity" AND "dataset"

We considered a result to be relevant if it provided an annotated dataset of bubble curtain imagery. However, none met this criterion. Instead, the results dealt with medical ultrasound bubbles, bubble activity in megasonic fields, and bubble-type behavior as part of economics, with the sources often showing duplicate results.

Second, there are no already established machine learning models geared toward detecting bubble curtains. For this set of search queries, we considered a result to be relevant if it provided a ready-to-train computer vision model that makes use of training data geared towards bubble curtains. Our search queries for this were:

- "machine learning model" AND ("bubble activity" OR "bubble plume" OR "bubble curtain")
- "machine learning model" AND "bubble activity"
- "machine learning model" AND "bubble plume"
- "machine learning model" AND "bubble curtain"

Once again, no relevant results appeared here. However, in this case, we encountered an indirectly relevant research article, which deals with the assessment of bubble plumes and their efficiency in water circulation [14]. This further supports the idea that there are mathematical approaches to analyzing the issue at hand but no readily available machine learning solutions for it.

This absence of an appropriate dataset and machine learning model that specializes in bubble curtain activity makes up the research gap that we intend to fill.

Research questions. On the basis of our context and identified research gap, our main concerns for this research document are:

1. Given an image or video input, are we able to accurately detect any sort of bubble activity that takes place?
2. Given these bubble activity detections, are we able to utilize them to gauge the efficiency of a bubble curtain?

Relevant domain. Real-Time Object Detection (RTOD) is the area of computer vision that is particularly suited to support the efficient operation of bubble curtains. Real-Time Object Detection involves maintaining an equilibrium between speed and accuracy while identifying objects of interest in real-time video footage. Such models have been successfully applied in multiple domains, including medicine, by means of a deep learning model specialized in the automatic diagnosis of COVID-19 [15]. Another example is autonomous driving, with research into the development of three-dimensional object detection systems as part of modern autonomous driving pipelines [16]. Despite the continuous development of RTOD technologies, their application in the field of underwater research remains sparse. Consequently, RTOD in underwater scenarios presents uniquely complex challenges, such as limited visibility and the lack of a fixed structure for objects.

Strategy overview. We begin with a review of the established literature, where we corroborate our choice to use a YOLOv10-based model, followed by our methodology. There we describe how we formed our dataset, the specific YOLOv10 implementation we chose, and the training structure we employed. Our findings are then introduced in the "Results" section, leading to a detailed analysis in the "Discussion" section and a conclusion paired with remarks about future work.

3 Literature Review

3.1 Foundations of Modern Object Detection

Architecture types. In response to the rising demand for precise inference in real-time applications, several object detection architectures have been proposed. When it comes to real-time object detection, a central topic is whether our chosen models perform single-shot or two-stage detection. As the name implies, two-stage detectors employ two passes over an input image to find objects. The first pass creates likely candidates for object existence and position, while the second one refines the proposals into a final result. Although this approach is generally accurate, it is longer and more expensive from a computational point of view. Single-shot detection uses a single pass over a given image to detect objects. It is more suitable for real-time object detection than two-stage detection, as it is less computationally expensive and inference times are lower in comparison, but it struggles to detect small objects.

Notable models. Remarkable models include Region-Based Convolu-

tional Neural Networks/ R-CNN, of which we refer to R-CNN, Fast R-CNN and Faster R-CNN [17–19], the Single Shot MultiBox Detector (SSD) [20], the You Only Look Once (YOLO) variety (primarily v10 at the time of writing) [21], and the DEtection TRansformer (DETR) [22]. YOLO and SSD are representatives of the single-shot architecture, while the R-CNN models are two-stage detectors. Despite being an object detection model, DETR does not fit either category directly, as it uses a new method that removes procedures like non-maximum suppression and anchor generation. Regardless, it is worth mentioning DETR due to its relevance in the field of object detection and the strong performance it demonstrates.

Field analysis. Following the revitalization of object detection as a whole, attributable to the publication of AlexNet by Krizhevsky et al. [23] in 2012, one of the first relevant two-stage object detection models, R-CNN, was released in November 2013. This marked a shift away from traditional hand-crafted object detection strategies such as the Histogram of Oriented Gradients (HOG) and Haar cascades towards modern, CNN-based methods.

3.2 Development of Object Detection Models

R-CNNs. The R-CNN system starts with a selective search of candidate object locations, a strategy that applies the benefits of both segmentation and exhaustive search [24]. A large feature vector is extracted from each candidate region and fed into a version of the CNN presented by Krizhevsky et al. adapted by Girshick et al. Finally, the extracted features are scored using a class-specific SVM for each one. The results show that R-CNN outperformed every competing model in 16 of the 20 classes found in the PASCAL VOC 2010 dataset and achieved a value of 43.5% when measuring mean average precision (`mAP`). This constitutes an absolute improvement of 8.4 percentage points over the second-best model [17].

Fast R-CNN aims to improve three major drawbacks that R-CNN exhibits: The expensive and prolonged training process, slow object detection, and the multi-phase training pipeline [18]. These objectives are achieved by introducing the Region of Interest (RoI) Pooling Layer, which converts RoIs into small fixed feature maps. Another important strategy is the hierarchical sampling of Stochastic Gradient Descent (SGD) mini-batches, which expedites training speed. In addition, Fast R-CNN uses a training procedure that jointly optimizes its softmax classifier and bounding-box regressor. The results speak for themselves: On PASCAL VOC 2010, Fast R-CNN outpaces other models in 15 of the 20 classes and achieves a `mAP` of 68.8%, an increase of over 25 percentage points.

Faster R-CNN identifies the region proposal stage as a major bottleneck in object detection models, and addresses the issue by computing the proposals with a deep CNN, resulting in the introduction of Region Proposal Networks (RPNs) [19]. RPNs slide a small network over the final feature map, predicting objectness scores and bounding boxes at each position. Faster R-CNN was evaluated on PASCAL VOC 2012, where it attained a `mAP` of 75.9%, by leveraging both the COCO and 07++12 datasets. Despite using just 300 region proposals, Faster R-CNN outperforms classic selective search algorithms that use 2000 proposals by approximately 10 percentage points in `mAP`.

YOLOs. YOLOv1 [25] represents another major paradigm shift in the field of object detection, as it predicts class probabilities and bounding box coordinates directly in one pass. This makes it remarkably quick compared to R-CNNs, marking the progression from Less Than Real-Time to Real-Time object detection models. The base model achieves a `mAP` score of 63.4% while processing images at 45 frames per second. A fast version of the model, aptly named Fast YOLO, possesses a 52.7% `mAP` and processes images at a staggering 155 frames per second. Both of these metrics were measured on PASCAL VOC 2007.

A major difficulty of YOLOv1 is correctly localizing objects, and YOLOv2 [26] seeks to rectify this issue. By adding batch normalization, fine-tuning the classification network on ImageNet for 10 epochs at full resolution (448 x 448, compared to YOLOv1's 224 x 224), and using anchor boxes for prediction, v2 fixes the faults of its predecessor, attaining 78.6% `mAP` on PASCAL VOC 2007.

YOLOv3 [27] is the most distinctive entry in the series, particularly due to its tonal shift. In addition to discussing the upgrades that make v3 faster, the authors reflect on the state of object detection and raise concerns about the ethical implications of using such models, specifically when it comes to military and surveillance applications. The enhancements to the v3 architecture include a new feature extractor that uses residual connections, predicting objectness scores using logistic regression, and switching softmax for independent logistic classifiers. Quantifying its performance on the COCO dataset shows that in terms of `mAP` v3 lags behind other contemporary state-of-the-art models such as RetinaNet (33 and 40.8 respectively). Even so, it is preferable to compare it to models such as SSD, where performance is similar (30 and 31.2 respectively), but v3 is 3 times faster.

YOLOv4 through v9 [28–33] continued the trend of introducing incremental updates to increase the performance of the models. These improvements include, *inter alia*, advanced data augmentation and a trainable bag-of-freebies for accuracy improvement.

Though newer YOLO versions include v11 and v12, we have selected YOLOv10 [21] as our option, as the widespread adoption of this version by individuals in the domain and the comprehensive documentation resulted in a robust and well-optimized architecture. One of the main changes to v10 is the shift to NMS-free training, to simultaneously increase performance and lower latency. Results show that compared to previous YOLO incarnations and contemporary state-of-the-art models, v10 outperforms them.

SSD. The main contribution SSD introduced was the combination of multi-resolution feature maps to handle dynamic object sizes. When measured on PASCAL VOC 2012, SSD outperformed the models of its time and achieved a value of 80.0 `mAP`. Nevertheless, compared to newer frameworks, the monolithic architecture of the SSD represents one of its main drawbacks, as it innately restricts the model's ability to evolve. This has pushed the field in a different direction, focused on greater flexibility.

DETR. By shifting the focus towards an attention-based architecture [34] and away from the classical, CNN-based approach, transformers have impacted the field of object detection and deep learning as a whole. DETR [22] treats object detection as a set prediction problem. Its main features include the use of transformers with (non-autoregressive) parallel decoding and bipartite matching loss. As measured on the COCO dataset, DETR achieves a 44.9 `mAP`, outperforming various R-CNN configurations. Although DETR presents a robust and accurate approach to handling object detection, its main drawback comes from the very long training schedules that transformers require.

This account of how object detection as a field and its related models evolve points to a clear pattern: Novel ideas are put forward, and through peer contributions, the full potential for speed and accuracy of an architecture is brought out. This paradigm inspires our methodological decisions, which are outlined in the following section.

4 Methodology

4.1 Model Architecture

Why YOLO? Based on the research gap we are attempting to fill, we require a model that strikes an acceptable balance between three factors: Inference speed, computational complexity, and precision. The model must also be able to detect objects accurately under various conditions, such as several dimensions, distances, and environmental conditions. Taking every

addressed subject into account, we conclude that the implementation of a YOLOv10-based model, fine-tuned on a custom dataset, represents a suitable strategy for our goal.

Chosen YOLO configuration. YOLOv10 models are available in multiple versions. Their main differences arise from the increasing number of parameters, precision, and latency that come with each subsequent model. The smallest model, YOLOv10-N (Nano), has 2.3 million parameters and exhibits quick inference times. This comes at the cost of low average precision, which signals that, while this model is fast, it is also rather inaccurate. With this information in mind, we conclude that the Nano version of YOLOv10 does not align properly with our specification for a good balance between speed and precision. Therefore, it is imperative to upgrade to a different version. After analyzing the rest of the models, YOLOv10 - S / M / B / L / X (small, medium, big, large, extra), YOLOv10-M emerges as the optimal solution to our requirements. This decision is supported on two fronts.

Firstly, upgrading from Nano to Medium results in an absolute gain of 12.6% in precision, while increasing latency by 2.79 ms on average. In comparison, upgrading to YOLOv10-S only results in an absolute gain of 7.8%, with an average latency increase of 0.65%. As we get an additional 4.8% in precision after switching from Small to Medium at the cost of just 2.25 ms extra latency, it follows that upgrading to YOLOv10-M is a practical choice.

Secondly, further upgrades from YOLOv10-M to another higher-capacity model are not warranted, partly due to the diminishing benefit of increased precision and because of the increasing drawback of higher latency. For example, switching from Medium to Large only grants an absolute gain of 2.1%, while incurring a supplementary 2.54 ms in average latency. As the precision gained continues to shrink and be outweighed by latency in terms of raw values, for our use case upgrading shifts from a benefit to a hindrance.

Model	**#Params**	**FLOPs**	**AP^{val}**	**Latency**
YOLOv10-N	2.3M	6.7G	38.5%	1.84ms
YOLOv10-S	7.2M	21.6G	46.3%	2.49ms
YOLOv10-M	15.4M	59.1G	51.1%	4.74ms
YOLOv10-B	19.1M	92.0G	52.5%	5.74ms
YOLOv10-L	24.4M	120.3G	53.2%	7.28ms
YOLOv10-X	29.5M	160.4G	54.4%	10.70ms

Table 1: Performance comparison of YOLOv10 configurations. Data adapted from [21, 35].

We may also evaluate the benefits of upgrading to a more robust model from a different point of view. More specifically, our new perspective concerns the number of floating-point operations (FLOPs) that the model performs

and the number of parameters it has. FLOPs describe the number of basic arithmetic operations performed per input, and the number of parameters refers to all the weights and biases the model adjusts during its training procedure. Table 3 showcases that while the number of FLOPs and parameters swiftly grow with each upgrade, precision does not follow the same trend. Once again, YOLOv10-M stands out as a reliable model choice.

The final perspective to take into account for this section is that even though the number of parameters considerably increased, training times remained reasonable throughout our study. Practically speaking, training speed correlates to model complexity, and in real-world applications, training deep learning models may take anywhere from a few hours to a few days, with highly complex cases lasting for even longer periods of time. For our use, training sessions using the YOLOv10-M model take less than 12 hours to complete. To further expedite this process, we opted to freeze the first ten layers of the model, and to fine-tune our model instead of training from scratch. The rationale behind the immobilization of the first layers is that neural networks focus on detecting low-level, ubiquitous features such as edges and lines. As the parameters for these layers are well-established during the pretraining phase of the model, there is no justification for further altering them. Fine-tuning also has multiple advantages, such as reducing the requirements for abundant data while still allowing the model to adjust to task-specific information [36].

4.2 Dataset Formation

What is a bubble curtain? Although they were briefly described in the introductory section, a more precise definition is required to lead the dataset curation process. For the purposes of our study, bubble curtains are characterized by the combination of multiple columns of rising bubbles. Due to the small scale of the bubbles and the distance from which they were filmed, the curtains appear almost mist-like in aspect. We elected to leave out scenes that displayed solitary bubble plumes that did not have the typical width showcased when using such systems, or if the bubbles were conspicuously sparse. Keeping this definition in mind was integral to producing consistent and high-quality results. These specifications guided the process of image selection and formation of our dataset, entitled BubSET.

Dataset material sources. The primary source for the acquisition of relevant data was a series of experimental recordings conducted in our university's OceanLab. With an average of two minutes per video, the footage captures contrasting scenarios which are intended to optimize the model's

ability to detect target objects. The cloudiness of the water sometimes varies to reflect the different conditions in real-life scenarios. Filming angles also alternate, to improve the scale and rotation invariance of the model. The intensity of bubbles represents another variable, as distinct curtains may operate at different levels of strength. In some videos, emphasis was placed on separate parts of the curtain, such as the origin, center, and top. These variables provide the model with a holistic understanding of what a bubble curtain is and how to detect it accurately, regardless of the circumstances.

A secondary source for the procurement of relevant material for this study was YouTube. Performing a series of search queries with the intention of gathering learning material, such as "Underwater Bubble Activity", "underwater bubble curtains", and "aquarium bubble wall", provided many instances that would later serve as parts for the dataset. However, most of the results were of little significance, which prolonged the curation process.

The final source used in this research is represented by the supplementary material provided as part of Beelen et al. [13]. The material includes two videos, entitled *Stationary* and *Drone*. *Drone* includes footage obtained using a remotely operated vehicle, and shows an extensive curtain that expels a large number of bubbles, while *Stationary* is filmed using an immobilized camera that can change angles, and shows localized bursts of activity from an off-camera source. The supplementary material can be found on 4TU.ResearchData.

Gathering process and filtration. To isolate individual frames from the videos, we utilized two Python libraries, namely yt-dlp and FFMPEG [37, 38]. FFMPEG is a command-line tool that processes video and audio, while yt-dlp is a feature-rich tool that facilitated the video-collecting process. They were used jointly to download candidate videos when necessary and split them into individual frames, which were subsequently uploaded to Roboflow [39] for further processing. Roboflow is a website that is specifically geared toward easily uploading, organizing, and preprocessing large datasets, while also offering image annotation capabilities. We opted for manual annotation of the data points, as this strategy would provide the most control when it comes to precisely enclosing objects within bounding boxes.

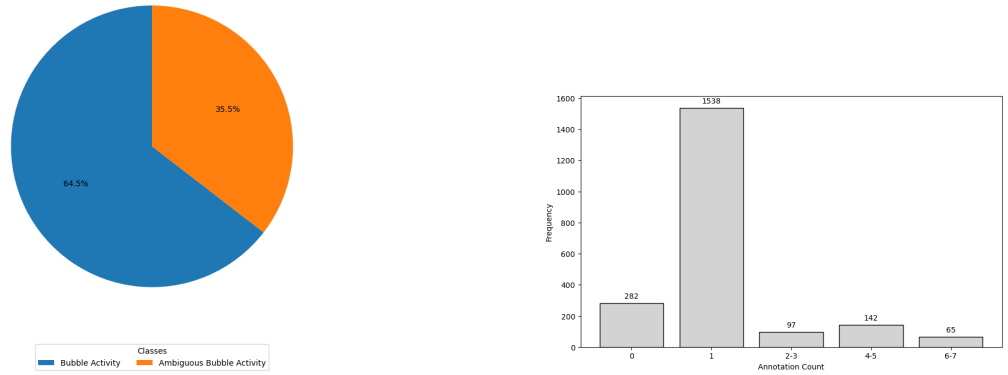
For annotating objects, we employed the following method: In the event that an object was clearly visible within the context of a given frame, it would be enclosed within a bounding box. Cases where the objects were obscured by environmental conditions or distance were relegated to the "ambiguous" class of objects, which is part of BubSET. The criteria for labeling an object as uncertain instead of ignoring it was the contrast between the candidate area and the background. If there was discernible contrast, the object would

then be classified as "ambiguous" instead of being overlooked. In such cases, it is useful to consider whether one must strain their eyes to see the activity. Should that be true, it becomes increasingly difficult for the model to extract meaningful features from the given objects. In these situations, a cautious approach to labeling objects is recommended. Despite ostensible subjectivity, the approach outlined above was used consistently throughout the dataset curation process.

Figure 1: Example of BubSET image.

Some frames gathered from YouTube videos could not be used as data points. This is because they did not include any sort of bubble activity or underwater scene, which meant that they could not provide any value as positive or negative examples. Such unusable frames were removed from the dataset.

The end result of this process contains 2124 images divided into 3 subsets: training (66%), validation (23%) and testing (11%).



(a) Annotation distribution by class.

(b) Histogram of Object Count by Image.

Figure 2: BubSET metrics.

Preprocessing steps and augmentations. Each image file contains

certain metadata that describes it. For instance, the EXIF (Exchangeable Image File) format describes how the image should be oriented. This information is stored and retrieved to encode the image at capture time [40]. Such metadata is useful for our image preprocessing stage, as it ensures a higher quality standard. In our case, we apply two key preprocessing steps to the dataset to enhance its reliability. Firstly, auto-orientation is enabled so that the position of the objects in the images match their respective bounding boxes. Secondly, the images are resized to 640 x 640 pixels, with the aim of accelerating the training process and improving model performance. In addition to the preprocessing steps, several augmentations were used to enhance the model's generalization ability and reduce overfitting during the training process.

We applied two types of augmentations, namely geometric transformations and quality changes. Transformations have the role of shifting image pixels to new positions with the intention of modifying the geometric structure, while quality adjustment altered the appearance of an image. The objective of flipping, cropping, rotating, and shearing the images is to instill variety into the model, while also instructing it that the bubble curtains we are looking for are not always perfectly in frame. Furthermore, changes to brightness are meant to replicate the dim conditions often found in videos of underwater scenes that include bubble curtains. Lastly, as the background color of the water is irrelevant to our task and should not be a deciding factor when the model detects bubble activity, grayscaling was applied as an augmentation to introduce color invariance.

Augmentation	**Values/Range**
Flip	Horizontal
Crop	0% Min Zoom, 20% Max Zoom
Rotation	Between -5° and +5°
Shear	±0° Horizontal, ±5° Vertical
Brightness	Between -10% and +10%
Grayscale	10% of images

Table 2: Overview of applied data augmentations and their parameter ranges.

As an additional point, during the preliminary stages of dataset construction, blurriness was also used as an augmentation. However, as a larger number of naturally blurry images were introduced into the collection, the need to apply it as a feature decreased. Applying further blurring to already blurred images could harm the model's ability to extract meaningful features from the pictures it uses as training material or skew its perception of what "bubble activity" means.

4.3 Auxiliary Software and Hardware

Hardware utilized. Hardware represents another important point of discussion. To gain access to satisfactory resources without being constrained by location or lack of physical hardware, we used Google Colaboratory [41]. Colab is a website that provides access to a variety of equipment, such as various GPUs and TPUs, while also allowing the user to work with their code inside of a Jupyter Notebook environment that requires little to no setup. Moreover, the equipment used for the experiments listed below is publicly available. This ensures that the training environment can be reproduced and our results easily verified.

The T4 GPU represents a cost-effective and readily available choice of equipment developed by NVIDIA using their Turing microarchitecture [42]. Typically speaking, it offers 16 GB of VRAM and includes 320 Tensor Cores, which are specialized units designed to decrease the time it takes to perform matrix operation. The L4 GPU [43] is a fast and energy-efficient hardware option that is also developed by NVIDIA. In contrast to the T4, however, the L4 is built on the Ada Lovelace architecture and possesses 24 GB of VRAM, which positions the L4 as a higher-throughput and lower-latency solution for training deep learning models.

The two Graphical Processing Units were used in conjunction, which presented an opportunity to monitor how differing GPUs affect training times for the object detection model. Our takeaway is that the L4 GPU is the preferable option, as it helps reduce the duration of the training process. Likewise, the High-RAM option available in Colab was another matter taken into consideration, as it helped mitigate any unwanted Out-Of-Memory (OOM) errors that may have resulted from the fine-tuning process.

Software utilized. Software structure is another important facet of this discussion. After multiple tests, we concluded that the optimal batch size for L4 GPUs is 32, while the optimal point for T4 GPUs is represented by a batch size of 16 with a nominal batch size of 32. Based on our experiments, lower sizes result in suboptimal resource usage, while larger sizes result in OOM errors. The particular setup for the T4 GPUs is possible thanks to gradient accumulation, which aids in training larger batch sizes despite possessing limited memory by accumulating multiple gradients before updating the weights of the model.

Another relevant instance of software is Automatic Mixed Precision, a specific implementation of Mixed Precision. This procedure elegantly combines FP16 and FP32, two floating point number formats meant to store gradient values and weights, to reduce the computational complexity of op-

erations such as convolutions and matrix multiplications [44].

In addition, the images in the dataset were cached in the memory of the program's runtime, with the objective of improving training speed by reducing disk I/O, but with the added liability of increased memory usage.

To reduce the number of false positives detected by the model, we introduced a post-processing mechanism in our model preparation pipeline. The process consists of two phases, identification and removal. In the first phase, all of the objects are identified on the basis of the ID that the model tracker gives each object. During the second phase, objects that have not appeared for at least the minimum number of frames (default value 5) are filtered out of the final visual output.

As ambiguous cases were distinguished from explicit bubble activity to ease the annotation process, the final version of the dataset included two object classes. This fact diverges from our initial intention of simply distinguishing between background and activity. As such, the `single_cls` parameter was toggled inside the training setup. This ensures that the different classes are treated as part of a singular category and bring our arrangement back in line with our original aims.

In order to further ensure reproducibility, the seed inside of our Jupyter Notebook which handles model training has been set to 10. Lastly, despite not encountering any issues during the training process, we opted to set checkpoints every 50 epochs so that the weights can be recovered should any setbacks occur.

4.4 Training Structure

Epoch number. During our trials, we conducted experiments that employed varying values for the total number of epochs, as it is imperative to identify the optimal value that this hyperparameter should have. If the value is too high, the model is prone to overfit to the training data, and if the value is too low, the model is exposed to the dangers of underfitting. Following our small-scale start of 150 epochs, we employed greater values such as 300, 500 and 1000 epochs. For our use case, the value of 700 epochs establishes itself as a solid option. At first glance, it may appear that this decision directly contradicts our previous considerations for a moderate value. The reason why it works is that, combined with the early stopping of the model, their combination results in a favorable training strategy. A supporting factor for this selection is the fact that the longer the model trains, the closer it is to achieving convergence. By the time the results gathered from late-stage

epochs start providing negligible improvements, early stopping takes effect, and the process ends before the model can overfit to the training data.

Model weights. To further mitigate the risks of overfitting, the model provides four types of weight files when training ends: The initial weights, the last set of weights, checkpoint weights which are dependent on how often the model saves its progress, and the weights that produced the best results during the entire process. The best weights are identified as such during the training process and are based on the mean average precision they produce. Specifically, the set of weights that produces the highest `mAP` is selected as the best. This rationale is tangibly implemented in YOLOv10-M through a fitness function. For every epoch, the algorithm calculates how well a set of weights performs by computing a dot product of precision, recall, `mAP@0.5` and `mAP@0.5:0.95`. The two `mAP` scores have the highest weight, 0.1 and 0.9 respectively, while precision and recall have no weight and therefore no influence on the final score [35].

Early stopping. To elaborate on our usage of this regularization technique, we used the default algorithm present inside the YOLOv10 architecture. The code puts this into effect through an `EarlyStopping` class, which checks if the newest training epoch achieves a fitness score equivalent to or better than the highest previous score. Our experiments applied patience values of 100 and 50. As the best set of weights is saved independently, we employed early stopping with a larger focus on reducing the elapsed training time and computational cost of the training loop.

Hyperparameters. YOLOv10 offers the possibility of automatically tuning hyperparameters by implementing Ray Tune, a Python library that handles hyperparameter optimization for various deep learning scenarios, in its framework [35].

For our implementation, the hyperparameters were either chosen by the default training logic or manually configured before training commenced. For our use case, SGD is determined as the ideal optimizer choice by the training algorithm, with a learning rate of 0.01 and momentum of 0.9. Further variables are grouped as follows: Biases and layers with batch normalization have no weight decay, while the remaining parameters have a standard weight decay of 0.0005.

The three loss values calculated across each epoch, localization, classification, and focal distribution (`box_loss`, `cls_loss`, `dfl_loss`) have particular weights attached to them by default. As part of our arrangement, we modified the `box_loss` and `cls_loss` values to 5.0, and 1.25 respectively. We increased the classification loss by 0.75 and decreased localization by 2.5 so

that the model would place more emphasis on accurately identifying objects during the training phase.

The `mosaic` hyperparameter refers to a type of augmentation that merges multiple data points into one organized mosaic, which can provide the dataset with further training examples. For our purposes, this setting was turned off, as all of the augmentations we provided and described in subsection 4.2 were enough for the model to achieve acceptable results.

4.4.1 Training Outcome

To briefly summarize this part before fully describing it in section 6, the Python implementation of YOLOv10 used in our study provides users with the ability to cleverly record all results in one directory, simply by toggling the `plots` flag before commencing a training loop. This way, we smoothly collected our training and validation metrics in one centralized location for review. These metrics include loss values for the training and validation sets, along with information about precision, recall, and `mAP`.

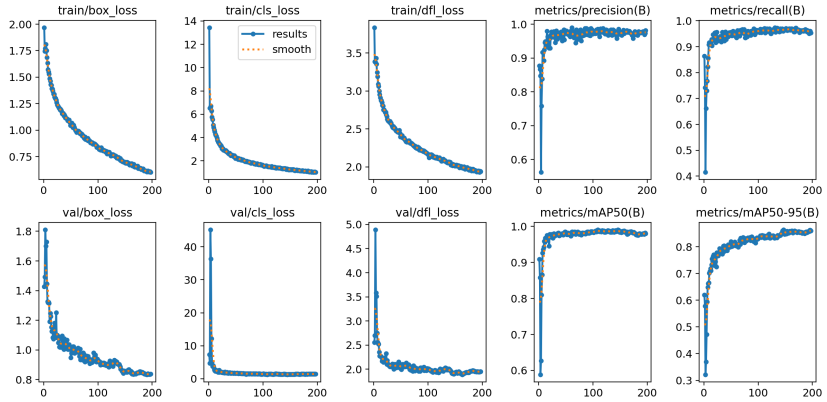


Figure 3: Automatically generated training metrics charts by the YOLOv10 implementation. Results are expanded on in section 5.

4.5 Testing Structure

Visual inspection. The procedure for evaluating a model's performance independently of its training component is largely similar to what we have described above. We begin by providing the model with the best weights from the training loop and instructing it to run inference on a video file. The result contains the model's annotations. The outcome is subjected to a visual analysis, combining our largely autonomous process with a Human-in-

the-Loop (HITL) element. This is done to ensure that the metrics recorded by the model are indeed of the quality they purport to be, as matching a visual confirmation with our benchmarks grants the whole more credibility. As observed by Redmon et al. [25], "Humans glance at an image and instantly know what objects are in the image, where they are, and how they interact." Our propensity to detect objects accurately justifies the inclusion of the HITL aspect.

Figure 4: Standard model detection.

The figure above exhibits a standard detection of our YOLOv10-M model. The detection includes the most definite area of bubble activity and shows a high confidence value, in this case 0.87. High confidence and the accurate bounding box are two factors that point towards a good generalization ability of the model, which will be subjected to in-depth analysis in Section 6, where we discuss our findings.

Metrics measurement. Beyond scanning the resulting material, what methods do we employ to verify that the model works as intended? This question has a threefold answer. First, we measure the confidence scores recorded during inference. We correlate a larger count of high values (i.e. greater than or equal to 0.8) with a better ability for the model to carry out its intended purpose well. Second, we measure the detection count between frames. In this case, consistency is our main interest, as successive frames with disparate detection counts indicate that the model detects multiple false positives. Our third and final method of verification is collecting and analyzing the same metrics used for training, such as precision, recall, mean average precision, etc. The three procedures allow us to get a better understanding of what the model detects and pinpoint amendable mistakes.

Below is a brief illustration of the entire process, as described in this section.

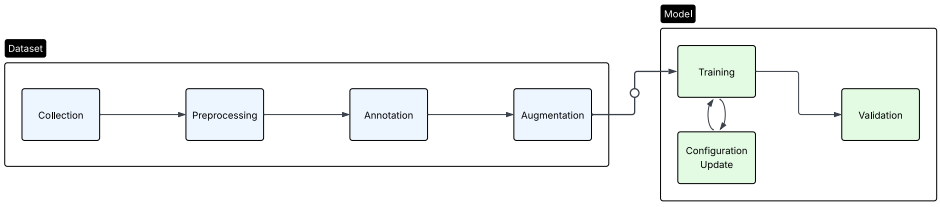


Figure 5: Dataset creation and model training pipeline.

5 Results

Overview. The setup for the experiments and the parameters used are the same as described in the methodology section. Furthermore, for each time we validated and gathered the results of a training configuration, we computed a fitness score in a fashion identical to the way described when analyzing the original YOLOv10 fitness function. The fitness value here has the objective of encompassing each individual model's performance into one value.

Config	Total Time	Origin	Precision	Recall	mAP@0.5	mAP@0.5:0.95	Fitness
14042025	6.1 hours	Pretrained	0.727	0.710	0.733	0.461	0.488
14042025-20finetune	23 min	14042025	0.824	0.854	0.864	0.607	0.632
14042025-40finetune	46 min	14042025	0.853	0.848	0.874	0.617	0.642
15042025	3.3 hours	Pretrained	0.855	0.832	0.880	0.619	0.645
15042025-20finetune	23 min	15042025	0.857	0.858	0.893	0.623	0.650
15042025-40finetune	46 min	15042025	0.863	0.838	0.874	0.608	0.630
16042025	1.3 hours	15042025	**0.895**	0.847	**0.898**	0.621	0.648
16042025-20finetune	23 min	16042025	0.838	0.873	0.890	0.624	0.650
16042025-40finetune	46 min	16042025	0.828	0.880	0.884	0.627	0.652
18042025	3.2 hours	Pretrained	0.840	0.883	0.891	**0.642**	**0.666**
18042025-20finetune	23 min	18042025	0.823	**0.889**	0.874	0.621	0.646
18042025-40finetune	46 min	18042025	0.868	0.860	0.890	0.624	0.650

Table 3: Comparison of training runs. Bold values indicate the best performance per metric. Grayed rows represent base training runs initialized from pretrained weights. Biggest results per column are highlighted.

Table description. The table highlights the different values that each model achieves after a full training loop or after being fine-tuned. The naming conventions are as follows: For each base variant of a model, we refer to it on the basis of the date in which it was created, written down in DDMMYYYY format. The additional trials follow the same system, while also having a descriptor appended to their name, which indicates for how many epochs they were fine-tuned. The "Origin" column refers to where the execution of that particular run started. "Pretrained" means that we used the base YOLOv10-M weights, while a model name expresses that prior training was performed. Training times for each configuration can range from 23 minutes to just slightly more than 6 hours, and each base iteration includes more

data than its predecessors.

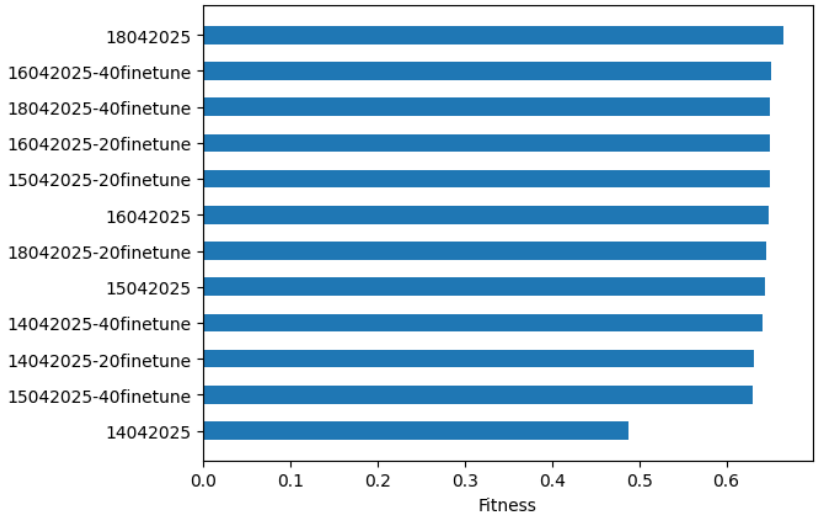
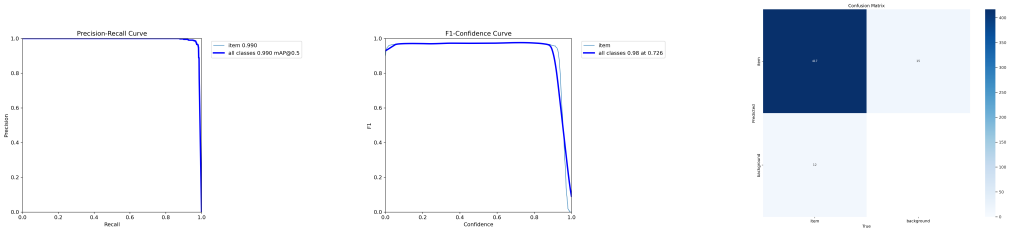


Figure 6: Training runs sorted based on overall fitness.

Model performance. The figure above highlights the general performance of each model while sorting them in descending order according to their fitness scores. The highest fitness value belongs to model 18042025, closely followed by several fine-tuned variants. Generally speaking, base models appear in the lower half of the chart, while most of the fine-tuned models position themselves higher on the list. The range of fitness values extends from 0.488 to 0.666.

Information about 18042025. The Precision-Recall curve of the model remains in the top right corner of the plot, which shows that this particular configuration reaches a value of 0.990 for `mAP@0.5`. Furthermore, 18042025 reaches a confidence value of 0.98 at a confidence threshold of 0.726, pointing to this as our optimal threshold. Lastly, the confusion matrix further outlines the strong performance of the model, as 417 of the 444 object instances were correctly identified, while having only 15 false positives and 12 false negatives.

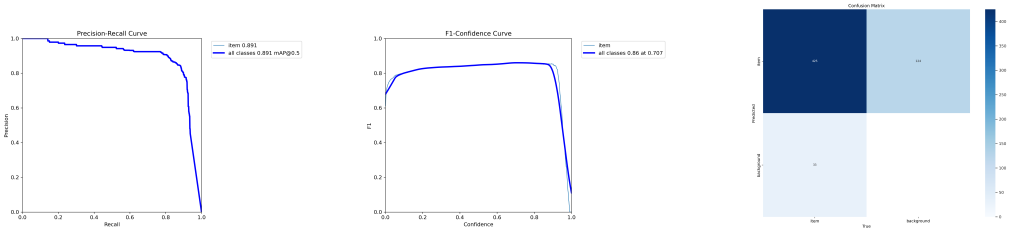
The final Precision-Recall curve, produced after examining this specific configuration on a test dataset, displays a smaller peak compared to the model training run, reaching a value of 0.891 for `mAP@0.5`. The same can be observed for the F1 measure of the model, which shows a maximum confidence value of 0.86 at a confidence threshold of 0.707. The errors displayed in the confusion matrix also increased, showing that while the vast majority of the model's 582 predictions are correct, the number of false negatives detected by the model more than doubled from 12 to 33, while the num-



(a) Precision-Recall Curve *(b) F1 Score vs. Confidence* *(c) Confusion Matrix*

Figure 7: Presentation of model 18042025's training metrics.

ber of false positives increased from 15 to 124, meaning that there are now approximately 8.27 times more false positives than before.

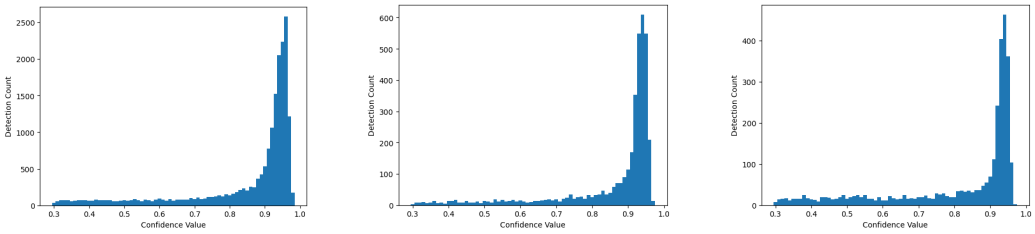


(a) Precision-Recall Curve *(b) F1 Score vs. Confidence* *(c) Confusion Matrix*

Figure 8: Presentation of model 18042025's testing metrics.

Testing dataset. In order to properly capture all the figures detailed above and appropriately assess the performance of each described model, we created and used a new test dataset. This set contains 499 frames, which were picked from our experimental recordings, along with supporting frames from the supplementary material provided in [13]. The name of this set is BubSET-Testing.

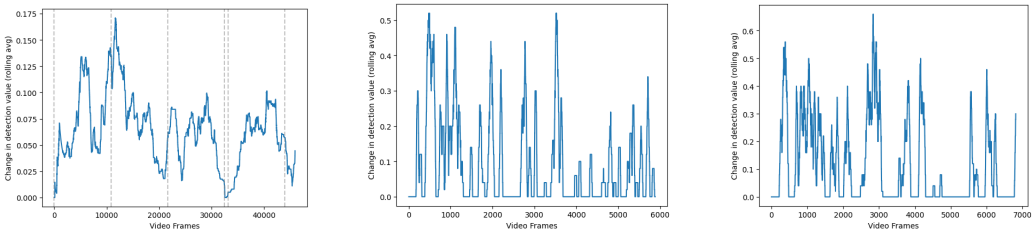
Confidence histogram. As part of our empirical measure of the performance of the model, we measured the confidence levels it exhibits when faced with new footage. For this we used three videos. The first two separately capture similar footage, but at high and low intensity, respectively. The final video shows footage recorded with the aid of a FiFish ROV V6 robot and includes a range of activity levels, angles, and movements. The final results show that the model is confident in its detections to a great extent, since most detections have a confidence value greater than 0.9.



(a) ROV footage. *(b) Video containing intense activity.* *(c) Video with non intense activity.*

Figure 9: Histograms of model confidence for testing footage.

Detection count change. For this section, we measured the detection count per frame. Using those results, we computed the variation in detection counts between two frames, at which point these differences were smoothed using rolling operations from the pandas Python library. The objective is to check how "jittery" the video footage is. That is, the amount of flickering for detections in a video. In the case of our footage, periods of both stability and flickering are present.

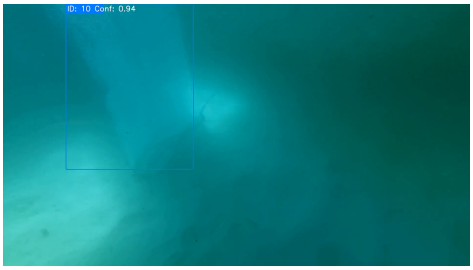


(a) Change in ROV footage detections. *(b) Change in high intensity detections.* *(c) Change in low intensity detections.*

Figure 10: Change in detection counts across videos.

Visual inspection. To accompany our quantitative assessment of the model performance, we performed a qualitative evaluation of 18042025's results.

As is visible in Figures 11 and 12, the model is able to accurately detect bubble activity under different conditions. Under less certain conditions, such as Figure 12 (b), the confidence of the model decreases, despite correctly identifying the activity that takes place in the given frame.

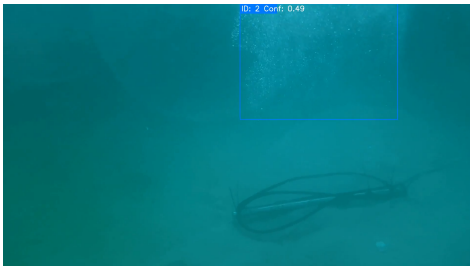


(a) Detection with a wide bounding box.

(b) Detection with reduced width, more height.

Figure 11: The model is capable of detecting objects across a range of situations. - 1

(a) Clear model detection.

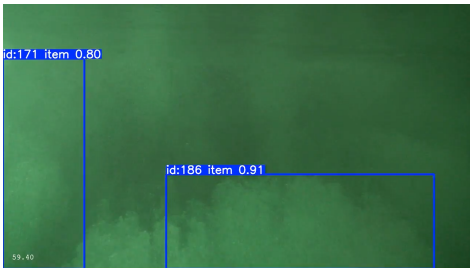


(b) Detection with a low bubble count.

Figure 12: The model is capable of detecting objects across a range of situations. - 2

Furthermore, when handling difficult examples, performance decreases compared to straightforward examples. Concretely, bounding boxes may capture the majority of the activity but not all, or miss instances of activity in ambiguous cases.

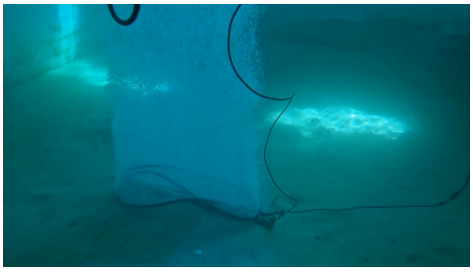
(a) Bounding boxes not fully covering activity.



(b) Results in an ambiguous environment.

Figure 13: Model handles ambiguous cases with increasing difficulties.

However, the model is prone to errors. For example, the figure below displays two types of errors that the model can come across. The first subfigure shows an instance of missed activity, despite the fact that the bubble activity in the frame is visible. The second subfigure includes a frame with no activity that was labeled regardless by the model.



(a) False Negative detection.

(b) False Positive detection.

Figure 14: Examples of errors the model encounters during testing.

6 Discussion

18042025 performance metrics. Recall the main figures that our model achieved. On BubSET-Testing, 18042025 attained a value of 0.891 for a `mAP@0.5`, a maximum F1 score of 0.86 at a confidence threshold of 0.707, and a fitness value of 0.666. Although Figure 8 shows that most of the detections fall within a high confidence threshold, with most values 0.9 and higher, the confusion matrix shows that the model still has a lot of false positives. This points to a possible observation: There exist some scenarios that the model has not learned to fully generalize. More specifically, while the model has a fair awareness of the given material, there are certain gaps in its understanding.

Performance-inhibiting factors. Although the model demonstrates adequate performance, there are still certain drawbacks that inhibit the full performance of the model. Observing Figure 13 and other test material, we can conclude that obstructions prevent the model from correctly identifying objects. This could have a negative impact on frame-by-frame detection counts, increasing the height of the spikes and their frequency, while causing flickering during visual inspection.

Closing. Per our research questions, we were able to create an object detection model that positively answers our inquiries. As a supplement, we may point out another facet of our inquiry: Generally, if a model lacks certain knowledge, it may receive additional fine-tuning for a short period of time. This might provide the model with a boost in stats, with as little as 20 epochs required in half of our experiments.

7 What This All Means

We introduce YOLOvBub, a real-time object detection system with the goal of identifying underwater bubble activity, based on YOLOv10 architecture. Furthermore, we introduce BubSET and its testing complement, BubSET-Testing. The datasets, the code used during this research, and extra inference footage can be found on GitHub. The system is designed to aid an object detection model in learning what constitutes bubble activity and what does not, as well as providing evaluation metrics for particular configurations.

By using YOLOv10-M, with the first 10 layers frozen, a batch size of 32 on an L4 GPU, and an increased focus on proper object classification, we take steps forward towards developing a robust system for monitoring bubble curtains. Additional enhancements to the architecture, such as providing a wide range of image augmentations, serve to further improve the final results yielded by the model.

Although YolovBub already exhibits acceptable performance levels, additional improvements can be made to the model's pipeline to hone its results. One such improvement concerns the version of our architecture. We detailed the reasons why we selected v10 as our version. Perhaps, as time advances, the factors presented may change and shift the balance in favor of using a newer YOLO version, or even a different architecture altogether.

Another improvement concerns the values given to our augmentations and hyperparameters. While our current values are justified, further testing of different value combinations could potentially yield stronger results.

As it currently stands, the model handles the binary task of detecting whether a certain position in a given frame contains bubble activity. However, a possible improvement in this regard would be to implement the ability to distinguish between intensity levels for detected activity. This could make use of the already established differentiation of activity in BubSET, and help the model reach a more nuanced understanding of what bubble activity is.

We hope that by presenting our findings, as documented in this paper, we can provide an incremental improvement to the field of underwater object detection and computer vision as a whole.

References

[1] M. Garside, "Global market value of deep sea mining technologies 2020–2030," May 2024, accessed: 2025-03-23. Available: https://www.statista.com/statistics/1276574/global-market-value-deep-sea-mining-technologies/.

[2] A. McCoy, W. Musial, R. Hammond, D. M. Hernando, P. Duffy, P. Beiter, P. Pérez, R. Baranowski, G. Reber, and P. Spitsen, "Offshore wind market report: 2024 edition," National Renewable Energy Laboratory, 15013 Denver W Pkwy, Golden, CO 80401, United States, Tech. Rep. NREL/TP-5000-90525, 2024, accessed: 2025-03-24. Available: https://www.nrel.gov/docs/fy24osti/90525.pdf.

[3] Y. Shabong, "Uk energy regulator approves $2.5 billion funding for new subsea cable," November 2024, accessed: 2025-03-24. Available: https://www.reuters.com/business/energy/uk-energy-regulator-green-lights-253-bln-funding-new-subsea-cable-2024-11-15/.

[4] Deep Sea Conservation Coalition, "Key threats to the deep sea," 2025, accessed: 2025-03-24. Available: https://deep-sea-conservation.org/key-threats/.

[5] S. C. Watson, P. J. Somerfield, A. J. Lemasson, A. M. Knights, A. Edwards-Jones, J. Nunes, C. Pascoe, C. L. McNeill, M. Schratzberger, M. S. Thompson, E. Couce, C. L. Szostek, H. Baxter, and N. J. Beaumont, "The global impact of offshore wind farms on ecosystem services," *Ocean & Coastal Management*, vol. 249, p. 107023, 2024, accessed: 2025-03-24. Available: https://www.sciencedirect.com/science/article/pii/S0964569124000085.

[6] T. A. Mooney, M. H. Andersson, and J. Stanley, "Acoustic impacts of offshore wind energy on fishery resources: An evolving source and varied effects across a wind farm's lifetime," *Oceanography*, vol. 33, no. 4, pp. 82–95, December 2020, available: https://doi.org/10.5670/oceanog.2020.408. [Online]. Available: https://doi.org/10.5670/oceanog.2020.408

[7] E. Commission, "Environmental impact assessment : Evaluating the effects of public and private projects on the environment," accessed: 2025-03-24. Available: https://environment.ec.europa.eu/law-and-governance/environmental-assessments/environmental-impact-assessment_en.

[8] E. P. R. Service, "Environmental impact assessment : Directive 2011/92/eu," European Parliament, Tech. Rep. PE 627.136, November 2018, accessed: 2025-03-24. Available: https://www.europarl.europa.eu/cmsdata/226402/EPRS_ATAG_627136_EIA_Directive-FINAL.pdf.

[9] N. Fisheries, "Understanding sound in the ocean," National Oceanic and Atmospheric Administration, Tech. Rep., n.d, accessed: 2025-03-25, Available: https://www.fisheries.noaa.gov/insight/understanding-sound-ocean.

[10] I. F. for Animal Welfare, "Eu takes historic step to protect marine life: mandatory limits on underwater noise pollution communicated to member states," March 2024, accessed: 2025-03-25, Available: https://www.ifaw.org/international/press-releases/eu-historical-madatory-limits-underwater-noise-pollution.

[11] E. Union, "Commission notice on the threshold values set under the marine strategy framework directive 2008/56/ec and commission decision (eu) 2017/848," European Commission, Tech. Rep., March 2024, accessed: 2025-03-25, Available: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX%3A52024XC02078&qid=1710328433367.

[12] P. H. Dahl, C. A. F. de Jong, and A. N. Popper, "The underwater sound field from impact pile driving and its potential effects on marine life," pp. 18–25, 2015, accessed: 2025-03-25. Available: https://acousticstoday.org/wp-content/uploads/2015/06/The-Underwater-Sound-Field-from-Impact-Pile-Driving-and-Its-Potential-Effects-on-Marine-Life-Peter-H.-Dahl-Christ-A.-F.-de-Jong-and-Arthur-N.-Popper.pdf.

[13] S. Beelen and D. Krug, "Planar bubble plumes from an array of nozzles: Measurements and modelling," 2024, accessed: 2025-03-25. Available: https://arxiv.org/abs/2401.09771.

[14] S. Choi and D. H. Kim, "Assessing operational efficiency of bubble plumes for water circulation enhancement," *Water*, vol. 16, no. 23, 2024. [Online]. Available: https://www.mdpi.com/2073-4441/16/23/3538

[15] T. Ozturk, M. Talo, E. A. Yildirim, U. B. Baloglu, O. Yildirim, and U. Rajendra Acharya, "Automated detection of covid-19 cases using deep neural networks with x-ray images," *Computers in Biology and Medicine*, vol. 121, p. 103792, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0010482520301621

[16] J. Mao, S. Shi, X. Wang, and H. Li, "3d object detection for autonomous driving: A comprehensive survey," 2023. [Online]. Available: https://arxiv.org/abs/2206.09474

[17] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," 2014. [Online]. Available: https://arxiv.org/abs/1311.2524

[18] R. Girshick, "Fast r-cnn," 2015. [Online]. Available: https://arxiv.org/abs/1504.08083

[19] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," 2016. [Online]. Available: https://arxiv.org/abs/1506.01497

[20] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, *SSD: Single Shot MultiBox Detector*. Springer International Publishing, 2016, p. 21–37. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-46448-0_2

[21] A. Wang, H. Chen, L. Liu, K. Chen, Z. Lin, J. Han, and G. Ding, "Yolov10: Real-time end-to-end object detection," 2024. [Online]. Available: https://arxiv.org/abs/2405.14458

[22] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," 2020. [Online]. Available: https://arxiv.org/abs/2005.12872

[23] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, p. 84–90, May 2017. [Online]. Available: https://doi.org/10.1145/3065386

[24] J. R. R. Uijlings, K. E. A. van de Sande, T. Gevers, and A. W. M. Smeulders, "Selective search for object recognition," *International Journal of Computer Vision*, vol. 104, no. 2, pp. 154–171, 2013. [Online]. Available: https://ivi.fnwi.uva.nl/isis/publications/2013/UijlingsIJCV2013

[25] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," 2016. [Online]. Available: https://arxiv.org/abs/1506.02640

[26] J. Redmon and A. Farhadi, "Yolo9000: Better, faster, stronger," 2016. [Online]. Available: https://arxiv.org/abs/1612.08242

[27] ——, "Yolov3: An incremental improvement," 2018. [Online]. Available: https://arxiv.org/abs/1804.02767

[28] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," 2020. [Online]. Available: https://arxiv.org/abs/2004.10934

[29] R. Khanam and M. Hussain, "What is yolov5: A deep look into the internal features of the popular object detector," 2024. [Online]. Available: https://arxiv.org/abs/2407.20892

[30] C. Li, L. Li, H. Jiang, K. Weng, Y. Geng, L. Li, Z. Ke, Q. Li, M. Cheng, W. Nie, Y. Li, B. Zhang, Y. Liang, L. Zhou, X. Xu, X. Chu, X. Wei, and X. Wei, "Yolov6: A single-stage object detection framework for industrial applications," 2022. [Online]. Available: https://arxiv.org/abs/2209.02976

[31] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, "Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors," 2022. [Online]. Available: https://arxiv.org/abs/2207.02696

[32] G. Jocher and Ultralytics, "Yolov8: Open-source object detection model (version 8.0)," https://github.com/ultralytics/ultralytics, 2023, accessed: 2025-04-26.

[33] C.-Y. Wang, I.-H. Yeh, and H.-Y. M. Liao, "Yolov9: Learning what you want to learn using programmable gradient information," 2024. [Online]. Available: https://arxiv.org/abs/2402.13616

[34] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2023. [Online]. Available: https://arxiv.org/abs/1706.03762

[35] Ultralytics, "Ultralytics yolov10 github repository," https://github.com/ultralytics/ultralytics, 2024, accessed: 2025-04-17.

[36] X. Han, Z. Zhang, N. Ding, Y. Gu, X. Liu, Y. Huo, J. Qiu, Y. Yao, A. Zhang, L. Zhang, W. Han, M. Huang, Q. Jin, Y. Lan, Y. Liu, Z. Liu, Z. Lu, X. Qiu, R. Song, J. Tang, J.-R. Wen, J. Yuan, W. X. Zhao, and J. Zhu, "Pre-trained models: Past, present and future," *AI Open*, vol. 2, pp. 225–250, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2666651021000231

[37] yt-dlp, "yt-dlp: A youtube-dl fork with additional features and fixes," https://pypi.org/project/yt-dlp/, n.d., accessed: 2025-04-02.

[38] F. Developers, "Ffmpeg: A complete, cross-platform solution to record, convert and stream audio and video," https://www.ffmpeg.org/, n.d., accessed: 2025-04-02.

[39] B. Dwyer, J. Nelson, T. Hansen, and et al., "Roboflow (version 1.0) [software]," https://roboflow.com, 2024, computer vision.

[40] B. Dwyer. (2020, May) When should i auto-orient my images? Roboflow Blog. [Online]. Available: https://blog.roboflow.com/exif-auto-orientation/

[41] Google. (2024) Google colaboratory. Accessed: 2025-04-05. [Online]. Available: https://colab.google/

[42] N. Corporation, "Nvidia turing gpu architecture," September 2018, accessed: 2025-04-05. [Online]. Available: https://images.nvidia.com/aem-dam/en-zz/Solutions/design-visualization/technologies/turing-architecture/NVIDIA-Turing-Architecture-Whitepaper.pdf

[43] NVIDIA Corporation, "Nvidia l4 tensor core gpu," 2023, accessed: 2025-04-06. [Online]. Available: https://www.nvidia.com/en-us/data-center/l4/

[44] P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. Elsen, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh, and H. Wu, "Mixed precision training," 2018. [Online]. Available: https://arxiv.org/abs/1710.03740