International Journal of Informatics, Data, and Emerging Applications
Vol 1, No 1, January – June, 2026, pp. 48-54
ISSN: 3164-5089 (Print) | E-ISSN: 3164-6921

Self-Hostable YOLOv11 Service for Marine Fish Data Curation

Jonathan Wuntu[1], Joshua Bernhard Tege[2]
[1]Dept. of Electrical Engineering, Sam Ratulangi University Manado, Kampus Bahu St., 95115, Indonesia
[2]Dept. Data Analyst CXM, dentsu Group, Menara Sentraya 35th Floor, South Jakarta, 12160, Indonesia
Corresponding e-mail: wuntujoy@gmail.com

***Abstract*— Marine fisheries sustainability requires accurate, real-time data on fish populations. This need is currently unmet due to the high costs of manual surveys and the slow, third-party monitoring solutions. This paper introduces a novel self-hostable web service designed for efficient aquatic monitoring and curation of YOLO format datasets. Utilizing FastAPI for asynchronous processing and a YOLOv11n architecture, this service enables rapid and high-accuracy fish detection, streamlining the creation of new training data. The self-reliant design eliminates reliance on third-party cloud annotation APIs, addressing privacy concerns and reducing long-term costs associated with data management. The YOLOv11n model's performance was benchmarked against YOLOv10n and YOLOv12n on the DeepFish and OpenImage V7-Fish datasets. It achieved superior detection on DeepFish with an mAP@50 of 97.2% and mAP@50:95 of 61.0%, demonstrating the robust capabilities crucial for effective fisheries management decisions. This work represents an improvement in autonomous marine monitoring technology and promotes data independence. Code is available at: https://github.com/jwuntu/YOLO_MFDDC.**

***Keywords*— Computer Vision, Deep Learning, Marine Fish, YOLO, Data Independence**

I. INTRODUCTION

Marine fisheries are a critical global resource, supplying essential protein to over 3.3 billion people and underpinning the economies of many coastal regions, approximately 35% of global fish stocks are overexploited, and marine biodiversity is decreasing rapidly [1], [2]. Effective, sustainable fisheries management is thus vital, as robust marine ecosystems are the foundation for long-term productivity [3], [4], [5].

Fisheries management requires precise and timely information on fish numbers, locations, and fishing intensity. Unfortunately, conventional monitoring methods are heavily restricted. Manual underwater surveys are expensive, labor-intensive, and can only cover small areas [6]. Relying on commercial catch data and logbooks poses an incomplete picture, only tracking harvested species and missing critical information about the whole ecosystem [7].

While underwater imaging and autonomous vehicles (AUVs) offer a pathway to large-scale data collection, interpreting these images still depends on specialists. Computer vision, particularly convolutional neural networks (CNNs), allows the best route to automation. CNN object detectors generally fall into two groups. Two-stage models, like Faster R-CNN, are highly accurate but too slow for resource-limited real-time AUV deployment [8]. This slowness can lead to missed detections and, consequently, inaccurate stock assessments.

The alternative is single-stage detectors, such as YOLO, which prioritize speed by detecting objects in a single pass. However, standard YOLO versions often suffer from a drop in accuracy, especially with the complex visuals and small sizes of marine life [9]. With YOLO, reduced accuracy, like slowness, also risks stock underestimation. Compounding this, many automated monitoring solutions force users into third-party data pipelines, requiring sensitive marine imagery to be used to external cloud services for annotation. This pipeline generates high costs, API dependencies, and significant concerns over data privacy and ownership.

This paper introduces a solution to both the speed-accuracy trade-off and the data dependence problem. Our system is built on the advanced, efficient YOLOv11n framework and utilizes a FastAPI backend for fast, asynchronous processing. This novel implementation creates a self-reliant, self-hostable web framework for automated YOLO format dataset curation, including video processing. This design ensures that users maintain comprehensive, independent control over their marine data assets.

II. METHODS

A. System Design Concept

The research focuses on developing a self-hosted tool that can automatically annotate and curate marine fish datasets in the YOLO format, aiming to streamline and simplify the dataset preparation process for object detection tasks in underwater environments. The tool functions to streamline the annotation process and achieve the best comparative performance between YOLOv10 and the latest YOLOv12 through a structured workflow. The core steps involve a comprehensive training and performance comparative study between YOLO variants, from YOLOv10 to the latest YOLOv12. This study, conducted with thoroughness and attention to detail, is crucial in preparing the backend processing for inference and automatic detection of marine fish based on the pretrained best-performance YOLO variants. To ensure fairness in the performance comparative study between YOLO variants, from YOLOv10 to the latest YOLOv12 in marine fish detection, this research will be using datasets from the DeepFish [10] and OpenImages v7-Fish [11] datasets.

Received: 10 August 2026, Revision: 26 September 2026, Accepted: 4 October 2026
available at: https://ejournal.unsrat.ac.id/v3/index.php/idea/

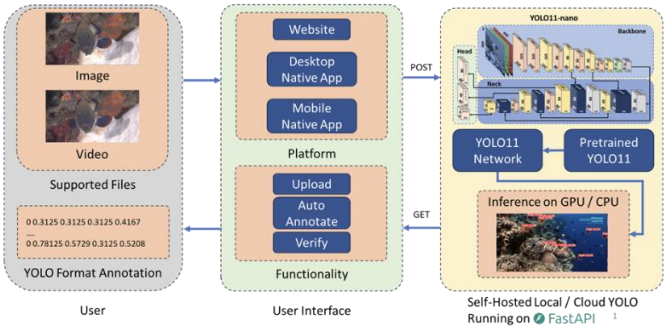


Fig. 1. System Design Concept

As illustrated in Figure 1, the system follows a client-server design centered on three components: the User, the User Interface, and the self-hosted backend. Users supply image or video files, from which YOLO-format bounding box annotations are ultimately derived. For this research, the primary interface is a web application, with the underlying design also supporting other platforms such as desktop and mobile native apps. The User Interface exposes two core functions: uploading media for automatic annotation via the POST method, and retrieving, reviewing, or verifying the resulting annotations via the GET method. These requests are served by a self-hosted backend running on FastAPI, which loads a pretrained YOLOv11 network to perform inference locally on available GPU or CPU resources, keeping all marine imagery and processing within the user's own infrastructure rather than relying on third-party cloud services. After YOLO infers the automatic annotations, the user verifies them: if the bounding boxes are sufficient, the user proceeds to download the YOLO-format annotation; otherwise, the user revises the annotation on the fly using the built-in annotation tool.

B. YOLO

This study employs the YOLO architecture as the main framework for object detection, specifically utilizing versions YOLOv10 through YOLOv12 to evaluate and compare their performance in detecting marine fish within curated datasets. All of these variants share a common structure consisting of a backbone serving as the primary feature extractor, a neck connecting the backbone to the detector, and a head serving as the detector, as illustrated in Figure 2.

1) Backbone

The Backbone section serves as the main feature extractor for the overall YOLO architecture. This process typically involves multiple convolutional layers that extract features from the input image by reducing its height and width while preserving essential information through the channel dimensions. While the backbone in YOLOv8 primarily consists of multiple convolutional layers, the more recent versions, YOLOv10 through YOLOv12, enhance this structure by integrating attention mechanisms.

YOLOv8 incorporates the C2f module, one of the main feature extractors, with the CBS (Convolution, Batch Norm, and SiLU) module. The C2f module uses the bottleneck module to extract features while reducing computational needs and maintaining feature extraction capabilities compared to the C3 module used in YOLOv5. Furthermore, YOLOv8 introduces the Spatial Pyramid Pooling Fast (SPPF) module, which improves upon the Spatial Pyramid Pooling (SPP) used in YOLOv5. Unlike

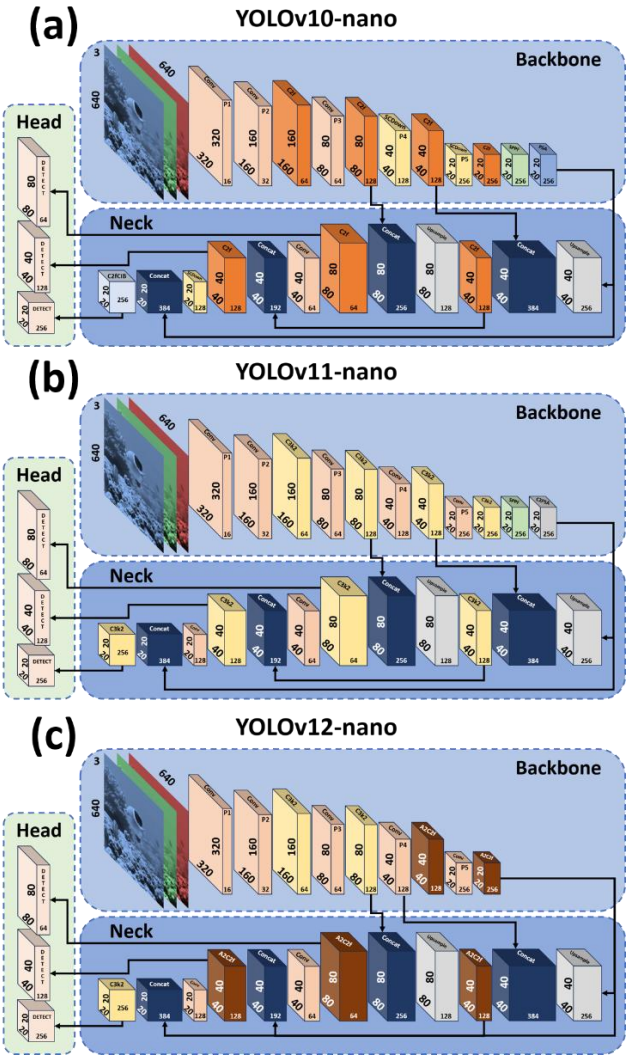


Fig. 2. YOLOv10 (a), YOLOv11 (b), and YOLOv12 (c) overall architecture

SPP, which applies three MaxPool layers with different kernel sizes (5×5, 9×9, and 11×11), SPPF uses three identical 5×5 MaxPool layers, achieving comparable multi-scale feature extraction with greater computational efficiency.

YOLOv10 builds on its predecessor, YOLOv8, while maintaining the use of the C2f module. As demonstrates in Figure 2 (a) the backbone of YOLOv10 adds a new attention mechanism called Partial Self-Attention (PSA) right after the SPPF layer, allowing the model to focus more effectively on important features while keeping computation efficient. The PSA attention mechanism processes only a portion of the channel dimensions through the attention module, while the remaining channels bypass it directly. Furthermore, some variants of YOLOv10 use a C2fCIB structure, which replaces the standard bottleneck inside the C2f module with a new component called the Compact Inverted Bottleneck (CIB) [12]. This design helps the network work more efficiently while capturing features in a clearer and more compact way.

YOLOv11 builds on its predecessor, YOLOv8, and introduces a novel C3k2 module replacing the C2f module, which is an updated version of the C2f module

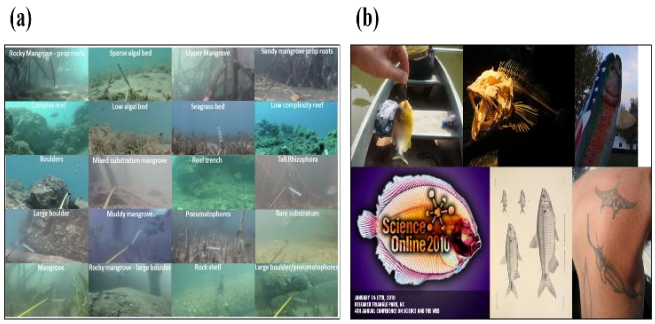


Fig. 3. Sample images from (a) DeepFish [10] and (b) OpenImages V7-Fish [11], illustrating the visual homogeneity of the former versus the heterogeneity of the latter.

itself [13]. The C3k2 module introduces a conditional component that replaces the bottleneck within the C2f module with another module called C3k, which itself contains a bottleneck structure. Furthermore, YOLOv11 incorporates an attention mechanism called the C2PSA, replacing the PSA attention mechanism. This C2PSA attention mechanism is an upgrade from the PSA module, which consists of two convolutional layers and one PSA module, hence the name C2PSA as shown in Figure 2 (b).

YOLOv12 builds on its predecessor, YOLOv11, while maintaining the use of the C3k2 module. As illustrates in Figure 2 (c), YOLOv12 introduces a new attention mechanism called A2C2f, which replaces the older C2PSA module to further enhance the model's ability to focus on important features. The A2C2f module is based on the C2f architecture, but instead of a single bottleneck, each bottleneck is replaced with two A2 modules, enhancing the model's ability to extract and combine features. This A2 attention is called area attention, which divides the feature of resolution (H, W) into L segments of size (H/L, W) or (H, W/L) [5]. Subsequently, this A2C2f module is present inside the Backbone and the Neck section. Furthermore, YOLOv12 removes the use of SPPF inside the Backbone section as shown in Figure 2 (c).

2) Neck

In YOLO architectures from YOLOv8 up to the latest YOLOv12, the Neck section plays a crucial role as a bridge between the Backbone and the Head. Its main function is to combine and refine the feature information extracted at different spatial scales by the backbone, ensuring that both high-level semantic features and low-level detailed features are effectively utilized for object detection.

To achieve this, the Neck performs a series of upsampling and downsampling operations, which help to align the feature map dimensions and facilitate multi-scale feature fusion. This ensures that the model can accurately detect objects of varying sizes and shapes within an image.

All versions from YOLOv8 through YOLOv12 implement a hybrid structure that combines the strengths of both the Feature Pyramid Network (FPN) and the Path Aggregation Network (PAN) [14]. The FPN helps propagate strong semantic information from deeper layers upward, while the PAN improves localization accuracy by transmitting low-level spatial details downward. Together, these networks generate three distinct feature scales and pass them to the Head section, which then predicts the final object locations and classes.

3) Head

The Head section in YOLO architecture serves as the final detector for object detection, segmentation, and pose estimation; in this research, it will be using the object detection for drawing the bounding box. All versions from YOLOv8 through YOLOv12 introduce an anchor-free detection head with three scale feature maps of 20×20, 40×40, and 80×80. Furthermore, in all versions from YOLOv8 through YOLOv12, the Head section is designed to independently process objectness, classification, and bounding box regression. The loss function between the three processes utilizes BCE (Binary Cross Entropy) loss [15] for classification, DFL (Distributional Focal Loss) [16], and CIoU (Complete Intersection over Union) loss [17] for bounding box regression. The loss function can be expressed as follows:

$$L_{total} = \lambda_{CIoU} L_{CIoU} + \lambda_{DFL} L_{DFL} + \lambda_{Cls} L_{Cls}. \quad (1)$$

In YOLO, the default balance parameters for the loss components are set to 7.5 for box loss, 0.5 for DFL loss, and 1.5 for classification loss, respectively [12], [18].

C. FastAPI

This study uses FastAPI as the backend framework for developing the automatic annotation tool used in marine fish dataset curation. FastAPI manages data processing, YOLO model inference, and communication with the web-based user interface (see Figure 1), where users can upload images, view, and verify annotations. Built on the ASGI standard, FastAPI designed to support asynchronous operations, allowing faster and more efficient handling of multiple tasks compared to traditional frameworks. Its modular design also makes it adaptable for future development into desktop or mobile applications. Previous studies have shown that FastAPI and other ASGI-based frameworks offer better performance and scalability than older synchronous web frameworks [19], [20].

D. Dataset

This study uses two datasets: DeepFish [10] and OpenImages V7-Fish [11]. The DeepFish dataset serves as the primary benchmark, consisting of 4,505 images divided into training (3,596), validation (809), and test (100) sets [10]. The OpenImages V7-Fish dataset, extracted from the larger OpenImages V7 collection [11], contains only fish-class images and includes 5,699 images split into training (5,343) and validation (356) sets, and evaluation on this dataset was conducted using the validation split. As shown in Figure 3, OpenImages V7-Fish also includes non-biological or out-of-context depictions of fish, such as statues, tattoos, and logos, unlike DeepFish's consistent underwater imagery, a heterogeneity that likely contributes to its comparatively lower detection performance in Section III.A.

E. Hyperparameter setting

All experiments were conducted with an input resolution of 640 × 640 pixels to achieve an optimal balance between detection accuracy and computational efficiency. The model underwent training for 300 epochs with a batch size of 32. Stochastic gradient descent (SGD) was utilized as the optimizer, featuring an initial learning rate of 0.01, a momentum value of 0.937, and a weight decay factor of 0.0005. To ensure efficient data throughput, the number of worker threads for data loading was set to 4. These parameters were consistently applied across all experiments and baseline comparisons to ensure fairness and reproducibility.

TABLE I. COMPARISON OF YOLO VARIANTS ON THE DEEPFISH DATASET [10]

Model	Params (M)	GFLOPs	mAP50 (%)	mAP50:95 (%)
YOLOv10n [9]	2.70M	8.4	96.6	60.6
YOLOv11n	2.59M	6.4	97.2	61.0
YOLOv12n	2.56M	6.5	96.1	60.7

TABLE II. COMPARISON OF YOLO VARIANTS ON THE OPENIMAGES V7-FISH DATASET [11]

Model	Params (M)	GFLOPs	mAP50 (%)	mAP50:95 (%)
YOLOv10n [9]	2.70M	8.4	50.5	34.4
YOLOv11n	2.59M	6.4	54.4	37.5
YOLOv12n	2.56M	6.5	46.1	29.3

F. Experimental platform

The study was conducted on the Kaggle platform. During training, a single NVIDIA Tesla P100 with 16GB of memory was used. This setup provided efficient computational power, accelerating the model's convergence speed and overall training process. For the FastAPI host, this study utilized a single NVIDIA GeForce RTX 3050 with 6GB of memory, along with an AMD Ryzen 7 5800H 8-core processor. This setup provided efficient computational power, accelerating the model's detection speed and the overall auto-annotation process, thereby simulating the performance of a small dedicated GPU server hosting this self-hostable web framework.

III. RESULT AND DISCUSSION

A. Evaluation on the Datasets

To determine the optimal model for the FastAPI-based auto-annotation web framework, three YOLO variants, YOLOv10n, YOLOv11n, and YOLOv12n, were systematically evaluated across two datasets. As shown in Table I, on the DeepFish benchmark dataset, YOLOv11n demonstrated superior performance with an mAP50 of 97.2% and mAP50:95 of 61.0%, outperforming YOLOv10n (96.6% mAP50, 60.6% mAP50:95) and YOLOv12n (96.1% mAP50, 60.7% mAP50:95). Despite YOLOv12n exhibiting the most lightweight architecture with 2.56M parameters and 6.5 GFLOPs, and YOLOv11n maintaining comparable computational efficiency at 2.59M parameters and 6.4 GFLOPs, the marginal differences in model complexity did not significantly compromise inference speed. When evaluated on the more challenging OpenImages V7-Fish dataset [11], as presented in Table II, all models exhibited reduced performance due to the increased complexity and diversity of the dataset. YOLOv11n maintained its leading position with 54.4% mAP50 and 37.5% mAP50:95, followed by YOLOv10n (50.5% mAP50, 34.4% mAP50:95) and YOLOv12n (46.1% mAP50, 29.3% mAP50:95). The consistent superiority of YOLOv11n across both datasets, coupled with its balanced trade-off between detection accuracy and computational efficiency, established it as the most suitable model for deployment in the real-time auto-annotation web framework for marine fish data curation.

B. System Architecture Overview

Following the model evaluation and selection of YOLO11n as the optimal detection model, a comprehensive web-based auto-annotation framework was developed to facilitate efficient fish detection and annotation workflows. The system architecture adopts a modular three-tier design comprising a FastAPI backend, a template-based frontend, and the YOLOv11n detection engine, as illustrated in Figure 1. This separation of concerns allows each layer to be developed, tested, and maintained independently, while communication between tiers is handled through RESTful API endpoints. Such a design also allows individual components to be upgraded or replaced in the future, for instance, swapping the detection engine for a newer YOLO variant, without requiring changes to the other layers.

The backend is built on FastAPI, selected for its asynchronous request handling, automatic API documentation, and efficient management of concurrent inference requests. These capabilities are essential for real-time annotation tasks that require responsive processing of images and video frames. The detection engine integrates the pre-trained YOLO11n model and manages the complete inference pipeline, including input preprocessing, object detection, and post-processing of bounding boxes, class labels, and confidence scores. The resulting predictions are converted into a standardized format for visualization and export. By performing inference directly within the backend rather than relying on external cloud-based services, the system reduces network-related latency, improves control over data privacy, and supports the self-hostable design objective of this study. Additionally, the backend structure allows the detection process to be maintained independently from the user interface, making future improvements to the inference pipeline easier to implement without significantly affecting other system components. This separation also simplifies debugging and helps ensure that detection results are processed consistently across different annotation tasks.

The frontend employs a template-based rendering approach similar to Flask's templating system, separating the user interface from the backend's detection logic. This design supports the development of an accessible interface for uploading images and videos, reviewing detection results, and managing annotations. Responsive layouts also help the application adapt to different screen sizes and browser environments, providing a consistent user experience across devices. Communication between the frontend and backend occurs exclusively through well-defined RESTful API endpoints, enabling consistent data exchange and reducing dependencies between the two components. These endpoints also provide a structured mechanism for handling user requests and returning detection results, helping maintain predictable interactions between the interface and the inference engine. Consequently, the interface can be modified or replaced without substantially changing the underlying detection engine, provided that the API specifications remain consistent. This modular architecture improves maintainability, supports future extensions, and reduces duplicated development effort when adapting the system to additional platforms. Overall, the combination of template-based rendering and RESTful API communication provides a flexible foundation for a scalable, maintainable, and extensible annotation system.

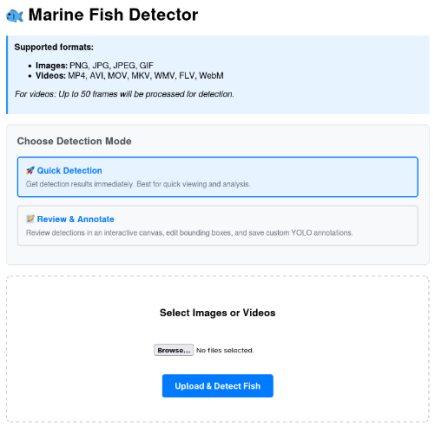


Figure. 4. Marine Fish Detector interface showing mode selection and file upload options.

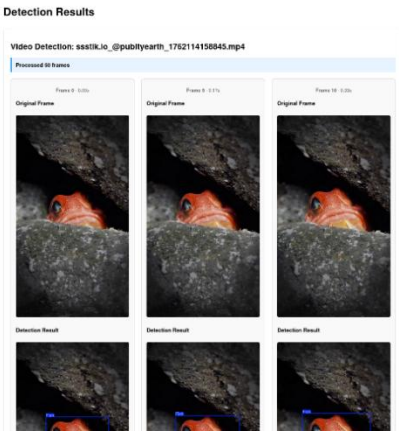


Figure. 5. Detection results displaying original and annotated frames from video processing with bounding boxes and confidence scores.



Figure. 6. Interactive annotation editor with bounding box editing tools and frame navigation controls.

Algorithm 1 Auto-Annotation Workflow

Require: Media M (image or video), confidence threshold θ
Ensure: Annotated images with bounding boxes and labels in YOLO format

User uploads media M through web interface
Send M to FastAPI backend via POST request
if M is video **then**
  Get total video frames N
  Calculate sampling interval $s = N/50$
  Extract 50 frames uniformly: $I = \{I_1, I_2, \ldots, I_{50}\}$
else
  $I = \{M\}$
end if
for each image I_j in I **do**
  Preprocess image I_j:
    Resize I_j to model input dimensions (640×640)
    Normalize pixel values to $[0, 1]$
  Execute YOLO11n inference on preprocessed I_j
  Obtain detection set $D_j = \{d_1, d_2, \ldots, d_n\}$
  for each detection d_i in D_j **do**
    if confidence$(d_i) > \theta$ **then**
      Add d_i to filtered detections D'_j
    end if
  end for
  Convert D'_j to YOLO annotation format:
  for each d_i in D'_j **do**
    Extract normalized $(x_{\text{center}}, y_{\text{center}}, \text{width}, \text{height}, \text{class}_i)$
  end for
  Store annotations $A_j = \{(x_{\text{center}}, y_{\text{center}}, \text{width}, \text{height}, \text{class}_i)\}$
end for
Return all annotations $A = \{A_1, A_2, \ldots, A_{50}\}$ (or single A for image) to frontend
Render bounding boxes on images in user interface
Allow user to edit, add, or remove annotations across frames
Export final annotations in YOLO format

C. *Auto-Annotation Workflow Algorithm*

The auto-annotation process transforms images or videos into annotated outputs, as presented in Algorithm 1. Users upload media to the FastAPI backend, which extracts 50 uniformly distributed frames for videos or processes images directly. Each image is resized to 640×640 pixels, normalized to [0, 1], and fed into YOLOv11n for detection. Outputs are filtered by confidence threshold θ and converted to YOLO format with normalized coordinates.

Annotations are displayed as visual overlays on the frontend, enabling users to assess quality and make corrections through interactive editing tools. Users can adjust bounding boxes, modify labels, delete errors, or add missed detections, with frame-by-frame navigation available for video sequences. Completed annotations are exported in YOLO format for immediate use in training pipelines.

D. *User Interface Design*

The web framework features a streamlined three-stage interface designed to minimize complexity while maximizing functionality for fish detection and annotation tasks. The interface architecture follows a progressive disclosure pattern, guiding users through the workflow sequentially from media upload to annotation export.

1) Stage 1: Input and Mode Selection (Figure 4)

The landing page, shown in Figure 4, displays supported formats (PNG, JPG, JPEG, GIF for images; MP4, AVI, MOV, MKV, WMV, FLV, WebM for videos) with a notification that video processing extracts 50 frames. Users select between two modes: "Quick Detection" for immediate viewing and analysis, or "Review & Annotate" for interactive editing and YOLO format export. After selecting files via the browse button, users click "Upload & Detect Fish" to initiate processing.

2) Stage 2: Detection Results Display (Figure 5)

Figure 5 shows the detection results interface. For videos, a grid displays all 50 extracted frames with timestamps, each showing the original frame alongside its detection result. Bounding boxes are overlaid with class labels and confidence scores, enabling quick assessment of detection quality. Images display a similar paired view focused on the single uploaded file.

3) Stage 3: Interactive Annotation Editor (Figure 6)

The annotation editor in Figure 6 provides a canvas with editable bounding boxes featuring resize handles for precise adjustments. Selected boxes are highlighted in green for clarity. An instruction panel explains Draw Mode for creating new boxes and Select Mode for editing existing annotations. Navigation controls enable frame-by-frame traversal for video annotations. The toolbar includes Draw/Select mode toggles, "Select Model" option, and "Clear All" button. Status indicators show total detections, selected boxes, and image dimensions. The interface maintains visual consistency across all stages through uniform typography and color schemes, ensuring smooth workflow transitions.

The cohesive design across all three stages maintains visual consistency through uniform typography, color schemes, and spacing, reducing cognitive load as users transition between workflow phases. The interface prioritizes immediate visual feedback, ensuring that all user actions, whether uploading files, viewing detections, or editing annotations, produce instant, unambiguous responses that confirm successful operation or clearly indicate errors requiring attention.

IV. Conclusion

This paper presented a self-hostable web framework for automated fish detection and YOLO dataset curation to support marine fisheries monitoring. The system integrates YOLOv11n with FastAPI, achieving 97.2% mAP@50 and 61.0% mAP@50:95 on DeepFish, outperforming YOLOv10n and YOLOv12n. On OpenImages V7-Fish, YOLOv11n achieved 54.4% mAP@50 and 37.5% mAP@50:95, demonstrating its effectiveness across diverse environments. The framework provides Quick Detection and Review & Annotate functionalities for efficient detection and annotation of images and videos. By eliminating third-party cloud dependencies, it enhances data privacy, reduces operational costs, and gives users greater control over marine imagery. Future work will focus on multi-species classification, active learning, and optimization for edge devices used in autonomous underwater vehicles, further supporting accessible marine monitoring and sustainable fisheries management.

References

[1] FAO, "The State of World Fisheries and Aquaculture 2022," *State World Fish. Aquac. 2022*, Jun. 2022, doi: 10.4060/CC0461EN.

[2] M. L. D. Palomares *et al.*, "Fishery biomass trends of exploited fish populations in marine ecoregions, climatic zones and ocean basins," *Estuar. Coast. Shelf Sci.*, vol. 243, p. 106896, Sep. 2020, doi: 10.1016/J.ECSS.2020.106896.

[3] E. Buonocore, U. Grande, P. P. Franzese, and G. F. Russo, "Trends and Evolution in the Concept of Marine Ecosystem Services: An Overview," *Water 2021, Vol. 13, Page 2060*, vol. 13, no. 15, p. 2060, Jul. 2021, doi: 10.3390/W13152060.

[4] C. Roberts *et al.*, "Rethinking sustainability of marine fisheries for a fast-changing planet," *npj Ocean Sustain. 2024 31*, vol. 3, no. 1, pp. 1–11, Sep. 2024, doi: 10.1038/s44183-024-00078-2.

[5] R. Hilborn *et al.*, "Effective fisheries management instrumental in improving fish stock status," *Proc. Natl. Acad. Sci. U. S. A.*, vol. 117, no. 4, pp. 2218–2224, Jan. 2020, doi: 10.1073/PNAS.1909726116/SUPPL_FILE/PNAS.1909726116.SAPP.PDF.

[6] A. Al Muksit, F. Hasan, M. F. Hasan Bhuiyan Emon, M. R. Haque, A. R. Anwary, and S. Shatabda, "YOLO-Fish: A robust fish detection model to detect fish in realistic underwater environment," *Ecol. Inform.*, vol. 72, p. 101847, Dec. 2022, doi: 10.1016/J.ECOINF.2022.101847.

[7] D. S. Murray, V. Campón-Linares, C. M. O'Brien, R. B. Thorpe, R. P. Vieira, and F. Gilmour, "Emerging issues in fisheries science by fisheries scientists," *J. Fish Biol.*, vol. 105, no. 2, pp. 557–563, Aug. 2024, doi: 10.1111/JFB.15683.

[8] H. Wang and N. Xiao, "Underwater Object Detection Method Based on Improved Faster RCNN," *Appl. Sci. 2023, Vol. 13, Page 2746*, vol. 13, no. 4, p. 2746, Feb. 2023, doi: 10.3390/APP13042746.

[9] J. Wuntu, M. D. Putro, and R. Syahputra, "Real-Time Fish Detection in Indonesian Marine Ecosystems Using Lightweight YOLOv10-nano Architecture," Sep. 2025, Accessed: Nov. 03, 2025. [Online]. Available: https://arxiv.org/abs/2509.17406v1

[10] A. Saleh, I. H. Laradji, D. A. Konovalov, M. Bradley, D. Vazquez, and M. Sheaves, "A realistic fish-habitat dataset to evaluate algorithms for underwater visual analysis," *Sci. Rep.*, vol. 10, no. 1, pp. 1–10, 2020, doi: 10.1038/s41598-020-71639-x.

[11] A. Kuznetsova *et al.*, "The Open Images Dataset V4: Unified image classification, object detection, and visual relationship detection at scale," *Int. J. Comput. Vis.*, vol. 128, no. 7, pp. 1956–1981, Feb. 2020, doi: 10.1007/s11263-020-01316-z.

[12] A. Wang *et al.*, "YOLOv10: Real-Time End-to-End Object Detection," no. NeurIPS, pp. 1–21, 2024, [Online]. Available: http://arxiv.org/abs/2405.14458

[13] R. Khanam and M. Hussain, "YOLOv11: An Overview of the Key Architectural Enhancements," Oct. 2024, Accessed: Nov. 04, 2025. [Online]. Available: https://arxiv.org/abs/2410.17725v1

[14] S. Liu, L. Qi, H. Qin, J. Shi, and J. Jia, "Path Aggregation Network for Instance Segmentation," *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, pp. 8759–8768, Mar. 2018, doi: 10.1109/CVPR.2018.00913.

[15]. Usha Ruby Dr.A, "Binary cross entropy with deep learning technique for Image classification," *Int. J. Adv. Trends Comput. Sci. Eng.*, vol. 9, no. 4, pp. 5393–5397, Aug. 2020, doi: 10.30534/IJATCSE/2020/175942020.

[16] X. Li *et al.*, "Generalized Focal Loss: Learning Qualified and Distributed Bounding Boxes for Dense Object Detection," *Adv. Neural Inf. Process. Syst.*, vol. 2020-December, Jun. 2020, Accessed: Oct. 08, 2025. [Online]. Available: https://arxiv.org/abs/2006.04388v1

[17] Z. Zheng, P. Wang, W. Liu, J. Li, R. Ye, and D. Ren, "Distance-IoU Loss: Faster and Better Learning for Bounding Box Regression," *AAAI 2020 - 34th AAAI Conf. Artif. Intell.*, pp. 12993–13000, Nov. 2019, doi: 10.1609/aaai.v34i07.6999.

[18] Y. Tian, Q. Ye, and D. Doermann, "YOLOv12: Attention-Centric Real-Time Object Detectors," Feb. 2025, doi: 10.0.

[19] A. Novikau, "Evaluative Comparison of ASGI Web Servers: A Systematic Review," *Int. J. Sci. Eng. Appl.*, vol. 13, no. 3, pp. 30–35, Mar. 2024, doi: 10.7753/IJSEA1303.1007.

[20] B. Bednarz and M. Miłosz, "Benchmarking the performance of Python web frameworks," *J. Comput. Sci. Inst.*, vol. 36, pp. 336–341, Sep. 2025, doi: 10.35784/JCSI.7738.