AI

MDPI

Article

Does YOLO26 Truly Offer Advantages over Its Predecessors for Edge Deployment? A Benchmark Study in Aquaculture

Rakesh Ranjan *, Gajanan S. Kothawade, Kata Sharrer, Scott Tsukuda and Christopher Good

The Conservation Fund Freshwater Institute, Shepherdstown, WV 25443, USA
* Correspondence: rranjan@conservationfund.org; Tel.: +1-(304)-870-2203

Abstract

The You Only Look Once (YOLO) has been widely adopted in aquaculture monitoring and management due to its real-time performance and deployment flexibility. The recently introduced YOLO26 architecture incorporates Non-Maximum Suppression (NMS)-free end-to-end inference and is optimized for deployment on resource-constrained CPU-based devices, making it particularly relevant for edge deployment in commercial aquaculture applications. Nevertheless, its performance, operational efficiency, and deployment suitability compared with previous YOLO generations remain largely unvalidated in aquaculture-specific scenarios. This study benchmarks YOLO26 against three Ultralytics predecessors (YOLOv5u, YOLOv8, and YOLO11) across nano, small, and medium model scales for the detection of fish mortality, a critical indicator of fish population health and welfare, in recirculating aquaculture systems (RAS). Twelve model variants were evaluated for detection accuracy, training efficiency across seven dataset sizes, and inference performance on both high-performance NVIDIA A100 GPUs and the resource-constrained, CPU-only Raspberry Pi 5 edge device. All models achieved comparable performance on the full dataset, with mAP50 varying by only 1.25 percentage points across three independent training runs, indicating minimal influence of architectural generation on final mortality detection accuracy when sufficient training data are available. However, notable differences emerged in data efficiency and deployment performance. YOLOv8 demonstrated the strongest training efficiency, achieving 90% mAP50 with only 400 training images, whereas YOLO26 nano and small variants required 1000 images to reach comparable accuracy. In contrast, YOLO26 exhibited advantages during edge deployment, with YOLO26n achieving the highest inference speed on the Raspberry Pi 5 at 7.84 $\pm$ 0.13 FPS across three benchmark sessions, while YOLOv5mu outperformed all contemporary medium-scale architectures on CPU-based hardware. These results demonstrate that architectural novelty alone is an insufficient criterion for model selection. The findings support a deployment-oriented framework in which training data availability, target hardware, and inference requirements collectively inform model selection for aquaculture applications.

Keywords: YOLO; AI; object detection; edge deployment; fish mortality; precision aquaculture

Check for updates

Academic Editor: Miguel Angel Cazorla

Received: 2 July 2026
Revised: 20 August 2026
Accepted: 7 September 2026
Published: 9 September 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

1. Introduction

With the growing global population and rising per capita seafood consumption, demand for aquaculture production continues to increase [1]. Intensive aquaculture systems, such as net pens and recirculating aquaculture systems (RAS), have shown promise to meet this growing demand by supplying affordable, high-quality seafood [2,3]. However,

intensive systems also introduce significant challenges related to sustainability, fish health and welfare, and environmental impacts [4,5]. For RAS, the high capital and operational costs of these systems place strong pressure on profit margins and demand a high resource efficiency to ensure economic viability [6]. Recent advances in precision aquaculture technology, particularly Computer Vision and Artificial Intelligence (AI)-aided automated monitoring and decision-support systems, offer a promising tool for mitigating operational challenges and maximizing yield in intensive production environments [7,8]. These methods enable automated, non-invasive, and continuous monitoring of key biological and operational parameters in aquaculture systems that were previously accessible only through costly, labor-intensive manual observations.

Several studies have reported the adoption of convolutional neural networks (CNNs) for computer vision applications in aquaculture [9,10]. CNNs are machine learning models specifically designed for processing image and video data and are widely used for object detection and instance segmentation tasks. CNN-based methods have been applied to a wide range of aquaculture applications, including feed optimization [9,11], animal behavior monitoring [12,13], biomass estimation [14], disease detection [15], mortality monitoring [16,17], and system inspection [18]. While domain-specific adaptation of CNNs is being increasingly researched, their deployment in intensive aquaculture faces unique challenges, including high stocking densities, elevated water turbidity, optical distortions, and the rapid, unpredictable motion of aquatic animals. Unlike stationary targets in crop-based applications, key objects in aquaculture, such as small uneaten feed pellets randomly floating in a highly occluded complex background, complicate object detection and tracking. Furthermore, remote aquaculture facilities often face constraints on high-bandwidth data transmission, which limits the feasibility of cloud-based CNN deployment. Therefore, deploying CNN models on resource-constrained edge devices is critical in aquaculture to enable real-time, on-site monitoring and decision support without relying on continuous cloud connectivity or high-bandwidth communication infrastructure.

Inference speed is a key criterion in selecting CNN models for real-time aquaculture monitoring applications, such as tracking morphological features and welfare-related traits in freely swimming fish. While early two-stage CNN object detectors (e.g., R-CNN, Faster R-CNN, and Mask R-CNN) achieve high detection accuracy, their sequential pipelines introduce substantial computational overhead, resulting in increased latency and reduced inference speed [19,20]. These models first generate candidate Regions of Interest (ROIs) using a Region Proposal Network (RPN), followed by classification, bounding-box regression, and confidence re-scoring to produce final detections. A state-of-the-art (SOTA) two-stage Faster R-CNN achieves only 7–18 FPS on a Titan X GPU [21]. On embedded devices such as the Jetson TX2, reported inference speeds fall below 1 FPS, making these models unsuitable for real- or near-real-time detection in edge applications [22]. To address these challenges, single-stage CNN object detectors (e.g., EfficientDet, RetinaNet, SSD, YOLO) were introduced. These models eliminate the region proposal generation step in detection and perform object classification and bounding box regression simultaneously in a single forward pass [23]. Among these models, You Only Look Once (YOLO) has gained popularity in computer vision research for its optimal balance of detection accuracy and inference speed. The first YOLO model (YOLOv1), introduced by Redmon et al. [21] achieved real-time object detection with inference speeds of 45 FPS for the base model and up to 155 FPS for Fast YOLO on a Titan X GPU. This represented the first unified object detection framework to achieve real-time performance while maintaining decent accuracy. While YOLOv1 achieved superior speed, its accuracy was not on par with that of the two-stage Faster R-CNN model. The subsequent YOLO versions (e.g., YOLOv3, YOLOv5, YOLOv8), however, further optimized the architecture to achieve detection per-

formance comparable to SOTA two-stage CNN models [24–26]. Unlike other detection frameworks that have evolved primarily within academic research groups, YOLO has undergone rapid, community-driven iteration, from YOLOv1 through YOLO26, with each version introducing architectural innovations that have been quickly adopted, modified, and validated across applied domains, including aquaculture.

Mortality is a key metric for assessing the health and welfare of aquaculture populations. Unusual mortality patterns may indicate disease outbreaks, environmental stress, or system failures, among other causes. Real-or near-real-time mortality tracking can provide farm managers with actionable insights to inform management decisions and mitigate root causes before mass mortality events occur. Several research studies have refined the original YOLO architecture and introduced custom frameworks such as D-YOLOv4/E-YOLOv4 [27], Improved YOLOv8 [28], and Deadfish-YOLO [17] to effectively address domain-specific challenges in mortality detection. The incorporation of attention mechanisms, lightweight backbone architectures, redesigned loss functions, and transformer modules into optimized detection frameworks has improved the model's detection performance. Nevertheless, deployment-oriented performance analysis of YOLO models and the practical challenges in resource-constrained edge deployments remain largely underexplored [29]. The majority of reported studies benchmark detection performance exclusively on GPU-equipped servers or laboratory workstations, which do not reflect the constraints of real-world aquaculture deployments. Although GPU-enabled embedded platforms (e.g., NVIDIA Jetson) offer on-site inference capabilities, they remain cost-prohibitive and consume relatively high power, limiting their scalability for per-tank or per-pen deployment in commercial operations. Low-power embedded systems, such as Raspberry Pi and other CPU-based single-board computers, are cost-effective and well-suited for real-world deployment and increasingly adopted in small- to medium-scale aquaculture settings for sensing and automation tasks [30,31].

Despite growing adoption of YOLO-based detection in aquaculture, several practical bottlenecks continue to hinder real-world deployment. Insufficient on-site computing resources limit the feasibility of deploying computationally demanding models in commercial facilities [29]. Low-light conditions and optical distortions caused by water turbidity reduce detection reliability in underwater imaging systems [10,17]. Object occlusion due to high stocking densities and overlapping fish further complicates accurate bounding-box assignment. Additionally, detecting small targets, such as juvenile fish or early-stage mortality events, remains challenging for standard label assignment strategies. These deployment bottlenecks collectively underscore the need for lightweight, edge-deployable detection architectures that can operate reliably in the challenging imaging conditions of commercial aquaculture environments.

While most CNN-based detection models are optimized for GPU acceleration, recent advances in the YOLO family, particularly the latest YOLO26 [32], have introduced streamlined architectures that enable faster, more resource-efficient inference on CPU-constrained edge devices. YOLO26 has been reported to achieve improved small-object detection accuracy and up to 43% faster CPU inference compared to its predecessor, YOLO11 [33]. However, these performance gains have been demonstrated primarily on general-purpose benchmarks such as the Common Objects in Context (COCO) dataset, and their applicability to domain-specific aquaculture tasks has not been validated in the peer-reviewed literature. There is a critical gap in evaluating the deployment feasibility, performance trade-offs, and practical limitations of modern lightweight detection architectures under aquaculture-specific conditions on resource-constrained edge hardware. To address this gap, this study conducts a deployment-oriented benchmark that systematically evaluates YOLO26 against prior Ultralytics YOLO models in terms of detection accuracy, data ef-

ficiency, inference speed, and real-world edge-deployment feasibility for fish mortality monitoring in RAS. Given the significance of mortality monitoring in aquaculture, this study leverages mortality data collected from a semi-commercial scale RAS system to ensure real-world validation. The specific objectives of this study are to:

1. Quantify and compare the detection accuracy and model efficiency of YOLO26 for fish mortality detection against other Ultralytics YOLO architectures (YOLOv5, YOLOv8, and YOLO11) across various relevant model scales.
2. Assess training efficiency by determining the minimum data requirements for each architecture using learning curve analysis.
3. Benchmark inference performance on both high-performance GPU and resource-constrained CPU-based edge hardware to assess the practical deployment feasibility of each architecture for on-farm RAS mortality monitoring.

2. Materials and Methods

2.1. YOLO Model Architecture

YOLO has established itself as the standard for deployment-oriented computer vision applications across diverse domains, including agriculture and aquaculture. Ultralytics (Frederick, MD, USA) is a leading developer and maintainer of modern YOLO architectures and has played a key role in standardizing YOLO as a unified, production-ready framework to enhance accessibility, reproducibility, and adoption for real-time object detection. To ensure fair benchmarking of accuracy, efficiency, and edge-deployment performance, this study exclusively evaluates the Ultralytics YOLO (YOLOv5u [25], YOLOv8 [26], YOLO11 [33], and YOLO26 [32]) architectures, as these models share a consistent framework, training, and implementation pipeline. Non-Ultralytics and legacy YOLO variants were excluded to avoid inconsistencies. The details of key architectural innovations and contributions for each Ultralytics YOLO have been summarized in Table 1. While YOLOv5u, YOLOv8, and YOLO11 have been extensively documented in the literature, and their architectural details are well established [34–36], YOLO26 represents the most recent addition to the Ultralytics family, with limited peer-reviewed coverage and no reported application in the aquaculture domain to date.

YOLO26 [32,35] is the latest-generation Ultralytics architecture, officially released in January 2026. The model is explicitly engineered for edge and low-power devices, featuring a streamlined design that reduces architectural complexity while incorporating task-specific optimizations to enable faster, lighter, and more accessible deployment. YOLO26 is a native end-to-end model that eliminates the Non-Maximum Suppression (NMS) post-processing step, resulting in lighter and faster inference (Figure 1). The NMS-free end-to-end detection paradigm was first introduced in the YOLO family by YOLOv10 [37], which demonstrated that eliminating NMS through a dual-assignment training strategy can reduce inference latency without sacrificing detection accuracy. YOLO26 extends this paradigm with CPU-specific architectural optimizations to improve inference efficiency on resource-constrained edge hardware. NMS elimination is particularly beneficial for CPU-based edge deployment, where the sequential sorting and filtering operations inherent to NMS introduce greater computational overhead than on GPU hardware. The Distribution Focal Loss (DFL) module is also removed to simplify the bounding box regression pipeline, enabling seamless export to ONNX, CoreML, TensorRT, and TFLite formats for efficient deployment on resource-constrained edge devices. Training convergence is accelerated by the MuSGD, a hybrid Muon-SGD optimizer adapted from large language model training that improves update conditioning and promotes stable gradient updates. Progressive Loss (ProgLoss) gradually shifts training supervision from the one-to-many branch toward the one-to-one inference head, aligning training with the NMS-free inference path used at deployment. STAL

(Small-Target-Aware Label Assignment) promotes positive candidate coverage for small objects that would otherwise receive zero gradient signal under standard TAL assignment, improving detection of small and spatially sparse targets. These optimizations collectively improve small-object detection accuracy, simplify deployment across a broad range of hardware, and deliver CPU inference up to 43% faster than YOLO11 [32]. YOLO26 supports real-time performance across multiple vision tasks, including object detection, instance segmentation, classification, pose estimation, and oriented object detection.

Table 1. Summary of Ultralytics YOLO architectures evaluated in this benchmark study.

Model (Year)	Key Architectural Innovation and Contribution	Tasks	Framework	Citation
YOLOv5u (2020)	Ultralytics re-implementation of YOLOv5 with an anchor-free detection head and decoupled classification/regression branches. Employs CSP (Cross-Stage Partial) backbone with SiLU activation and PANet neck for multi-scale feature aggregation. The 'u' suffix denotes the updated anchor-free variant used in this study.	Object Detection, Instance Segmentation	PyTorch (Ultralytics)	[25]
YOLOv8 (2023)	Ultralytics next-generation model featuring a new C2f backbone and a fully anchor-free, decoupled detection head. Introduced improved training strategies, including mosaic augmentation, closure, and dynamic task-aligned assignment. Supports multi-task learning natively across detection, segmentation, pose estimation, and classification.	Object Detection, Instance Segmentation, Pose Estimation, Classification	PyTorch (Ultralytics)	[26]
YOLO11 (2024)	Introduced C3k2 CSP bottleneck blocks throughout backbone and neck for enhanced efficiency with fewer parameters than YOLOv8. Incorporated C2PSA (CSP with Spatial Attention) module to improve feature focus on salient regions. Achieves 22% fewer parameters than YOLOv8m while maintaining higher COCO mAP. Extended YOLO to oriented detection.	Object Detection, Instance Segmentation, Pose Estimation, Oriented Detection, Classification	PyTorch (Ultralytics)	[33,36]
YOLO26 (2026)	Edge-optimized end-to-end model eliminating Non-Maximum Suppression (NMS) at inference via a one-to-one prediction head trained using a dual-head strategy, removing post-processing latency. Removed Distribution Focal Loss (DFL) for simpler, hardware-friendly bounding box regression. Introduced MuSGD optimizer (SGD + Muon hybrid) for stable convergence, ProgLoss for progressive loss balancing, and Small-Target-Aware Label Assignment (STAL) for improved small-object detection accuracy. Delivers up to 43% faster CPU inference than YOLO11.	Object Detection, Instance Segmentation, Pose Estimation, Oriented Detection, Classification	PyTorch (Ultralytics)	[32,35]

YOLO26 was selected as the primary model for this benchmark study because its architecture is theoretically well-suited for deployment on resource-constrained on-farm hardware and for addressing real-world mortality detection challenges, including detecting overlapping targets during high-mortality events (Figure 1).

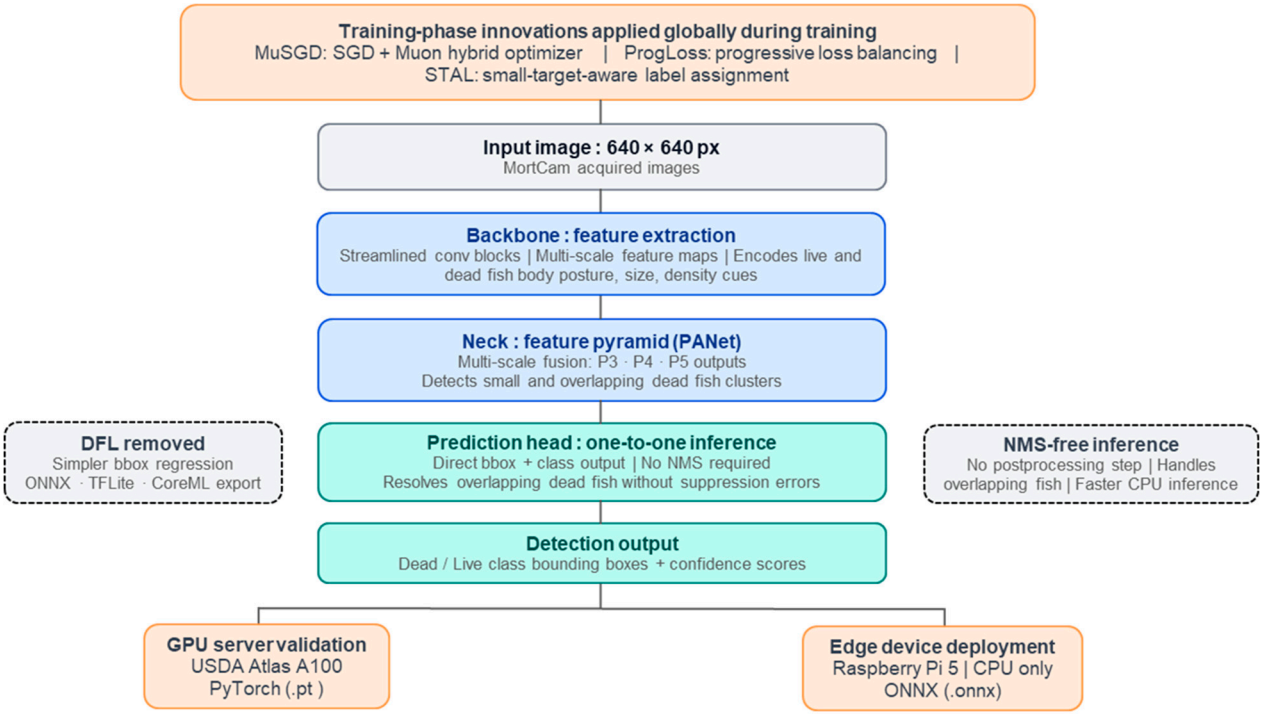


Figure 1. YOLO26 architecture, illustrating the end-to-end inference pipeline and key architectural innovations. The mortality model was trained and benchmarked on an NVIDIA A100 GPU (USDA SCINet Atlas) and deployed on a Raspberry Pi 5 to evaluate real-world inference performance on a resource-constrained edge device [NMS: Non-Maximum Suppression; DFL: Distribution Focal Loss].

2.2. Dataset

2.2.1. Image Acquisition

The mortality data were collected in a semi-commercial-scale RAS grow-out tank (volume = 150 m^3) stocked with 5000 Atlantic salmon (*Salmo salar*; stocking density = 54.3 kg m^{-3}; mean fish weight = 1.63 kg) at The Conservation Fund Freshwater Institute (Shepherdstown, WV, USA). A custom-built mortality imaging system (MortCam) [16] equipped with an RGB sensor (RPi HQ Camera, Raspberry Pi Foundation, Cambridge, UK; 12.3 MP resolution), a wide-angle lens (M23272M14, Arducam, Nanjing, China; 2.72 mm focal length, F2.5 aperture, 140° horizontal field of view), and an integrated single-board computer (Raspberry Pi 4B, Cambridge, UK; 1.5 GHz quad-core Cortex-A72 processor) was used for data collection (Figure 2). MortCam was deployed vertically downward, suspended from a steel cable at the center of the tank, positioned 3 m below the water surface and 0.6 m above the bottom drain plate. The camera was programmed to capture still images every 15 min. A pair of waterproof LED lights (LUMEN-LIGHT-R4-RP, Blue Robotics Inc., Torrance, CA, USA; pressure rating: 500 m; maximum brightness: 1500 lumens) was mounted on the MortCam housing to enhance illumination at the tank bottom. For the first 45 days, images were acquired under 'ambient light', with only the ceiling LED lights (model DLE-18-ST-W-5000–00, Digital Lumens, Boston, MA, USA) installed 7.9 m above the drain plate. Thereafter, for the next 45 days, underwater LED lights were switched on to capture images in 'supplemental light' conditions. During the data-collection window, mortality in the grow-out tank was marginal, and the acquired dataset could not capture high-mortality scenarios. To deal with this, euthanized culls from the same population were added to the tank on the last day to simulate a high-mortality event, and images were captured under both lighting conditions. Further details on MortCam development, installation, and data acquisition protocol can be found in a study by Ranjan et al. [16].



(a) (b) (c)

Figure 2. The (**a**) MortCam was installed above the bottom drain plate of a semi-commercial-scale grow-out RAS tank to image live and dead fish within the camera's field of view under (**b**) ambient and (**c**) supplemental lighting conditions.

The 90-day collection period, two distinct lighting conditions, minor positional variation in the suspended camera, dynamic repositioning of dead fish driven by water currents, and natural variation in live fish density, behavior, and mortality levels in the frame collectively ensure sufficient visual diversity across the dataset.

2.2.2. Data Preprocessing

A total of 2000 images were manually sorted and selected for model training, representing diverse mortality levels [zero, low (<3 dead fish), and high (≥3 dead fish)], illumination conditions (ambient and supplemental), and varying water quality scenarios captured over three months of camera deployment. These images were uploaded to Roboflow (Roboflow Inc., Des Moines, IA, USA; https://roboflow.com) for data annotation and preprocessing. The images were manually annotated into two object classes ('Live': actively swimming fish; 'Dead': fish exhibiting belly-up floating behavior indicative of mortality) and randomly split in a 70:20:10 ratio for model training, validation, and testing. Each of the datasets was preprocessed in two steps: (1) auto-orientation by removing EXIF data, and (2) image resizing (416 × 416 pixels). Thereafter, the preprocessed training set was augmented using Roboflow's built-in augmentation pipeline [2× augmentation: brightness (±25%), rotation (±90°), and blur (±2 pixels)], yielding a final training set of 2800 images (i.e., 1400 original and 1400 augmented images). The training dataset had 2204 instances of the 'Dead' class and 3209 instances of the 'Live' class (Table 2). The validation and test sets were retained in their original form to provide unbiased performance estimates. All experiments used these fixed splits to ensure comparability across model evaluations.

Table 2. Summary of the mortality dataset used in this benchmark study.

Split (%)	Original Images	After Augmentation	Dead Instances	Live Instances
Training (70%)	1400	2800 (2×)	2204	3209
Validation (20%)	400	400 (none)	554	734
Test (10%)	200	200 (none)	311	310
Total	2000	3400	3069	4253

2.3. Model Training

Twelve object detection models from four Ultralytics YOLO generations (YOLOv5, YOLOv8, YOLO11, and YOLO26) across three size tiers [nano (n), small (s), and medium

(m)] were selected for this benchmark study. Larger model variants [large (l), and extra-large (x)] were excluded due to higher computational demands, larger memory footprint, and slower inference speeds, which limit their suitability for edge deployment scenarios. Each model was initialized with weights pretrained on the MS COCO dataset prior to fine-tuning.

2.3.1. Computing Hardware

All model training was conducted on the USDA Agricultural Research Service SCINet Atlas high-performance computing cluster (Mississippi State University, Starkville, MS, USA) with one NVIDIA A100 GPU (80 GB VRAM) and four CPU tasks per allocation. Interactive training sessions were accessed through Atlas Open OnDemand (OOD) using JupyterLab in a custom Conda environment. The software stack included Python (v3.10.20), PyTorch (v2.10.0) with CUDA 12.8 support, and Ultralytics (v8.4.21) for YOLO model training and validation.

2.3.2. Hyperparameter Tuning

All models were trained under identical hyperparameters to ensure fair cross-model comparison. Training was conducted for up to 100 epochs with early stopping patience of 50 epochs, an input image size of 640 × 640 px, and a batch size of 32. The optimizer was set to auto, which resolves to MuSGD for YOLO26 and SGD for YOLOv5u, YOLOv8, and YOLO11, consistent with each architecture's default training configuration in Ultralytics. The initial and final learning rates were both set to 0.01, with momentum of 0.937, weight decay of 0.0005, and a warmup period of 3 epochs. The IoU threshold was set to 0.7 for training label assignment. The default Ultralytics augmentation pipeline was applied uniformly, including mosaic augmentation, RandAugment, horizontal flipping (p = 0.5), HSV color jitter, scale and translation transformations, and random erasing. All models were initialized with MS COCO pretrained weights. Model performance was assessed across three independent training runs (seeds 1, 2, and 3) with deterministic training enabled. Complete training configuration files (args.yaml) for all runs are available in the public GitHub (https://github.com/PA-RRanjan/RAS-MortDB, accessed on 6 June 2026) repository.

2.3.3. Learning Curve Analysis

Learning curve analysis was performed to assess data efficiency, performance saturation, and the minimum dataset size needed for reliable detection accuracy. The analysis was performed across 12 selected models (4 model generations × 3 size tiers) using 7 training dataset sizes (100, 200, 400, 700, 1000, 1400, and 2800 images), resulting in 84 total training runs. Stratified random subsets were sampled from the entire 2800-image training set (hereafter termed as 'full dataset') using a fixed seed of 42 to generate datasets of different sizes. Since each original image had a corresponding augmented version, image pairs were sampled together to preserve augmentation balance across all subset sizes. The validation and test sets were kept constant across all experiments to ensure a fair performance comparison. Model performance at each dataset size was evaluated on the fixed validation set, while the test set was retained exclusively for final model evaluation and edge inference benchmarking. Unlike the full dataset, which was trained three times per model, learning curve models were trained once per dataset size, as repeating all 84 runs across multiple seeds was computationally infeasible. The strong training reproducibility demonstrated at the full dataset size (Section 3.1.1) supports the reliability of the single-run learning curve findings reported in this study.

2.4. Model Performance on GPU

The detection performance of all models was evaluated using mean average precision at an IoU threshold of 0.50 (mAP50), mean average precision across IoU thresholds from 0.50 to 0.95 in 0.05 increments (mAP50–95), precision, and recall [34]. All detection performance metrics are reported as the mean ± standard deviation (SD) across three independent training runs (seeds 1, 2, and 3), unless otherwise stated. Additionally, inference speed was assessed on the training GPU using two complementary metrics. First, raw GPU inference time was measured as the neural network forward-pass time (*Infer*, ms) per image using the Ultralytics validation pipeline on the test set. This metric reflects the true computational cost of the model by excluding system-level overhead and input/output (I/O) processes. Second, end-to-end average latency (*Avg*, ms) was evaluated to capture the full per-image inference pipeline, including image loading, preprocessing (resizing and normalization), forward-pass computation, and post-processing (e.g., NMS or equivalent). This metric was computed in Python across all 200 test images after three warm-up iterations to mitigate initialization effects and ensure stable timing estimates. The corresponding GPU throughput was then derived as the frame rate (GPU FPS) from the measured end-to-end latency. The postprocessing time (*Post*, ms) was reported separately to distinguish NMS overhead between NMS-based (YOLOv5u, YOLOv8, YOLO11) and NMS-free (YOLO26) architectures. Additionally, training time (hours, h), number of parameters (millions, M), PyTorch (PT) model size (megabyte, MB), and ONNX export size (MB) were recorded as supplementary efficiency indicators. Model size can be a limiting factor for deployment on edge devices with limited memory and storage. Therefore, model efficiency was quantified as the mAP50 per MB of PyTorch (PT) model size (×100) to evaluate the accuracy-to-storage trade-off, using the mAP50 averaged across the three training runs.

2.5. Edge Deployment Evaluation

Edge inference benchmarking was performed on a CPU-only single-board computer (Raspberry Pi 5, Cambridge, UK; Broadcom BCM2712), featuring a quad-core ARM Cortex-A76 CPU at 2.4 GHz and 8 GB LPDDR4X RAM, running Debian GNU/Linux 12 (Bookworm). The device was configured with Python (v3.13.5), Ultralytics (v8.4.23), ONNX Runtime 1.24.4 with CPUExecutionProvider, PyTorch (CPU build, v2.10.0), and psutil (v5.9.4). No GPU-accelerated co-processor was used. An active cooling system comprising an integrated fan, aluminum heatsink, and thermal pads was installed to mitigate thermal throttling during sustained inference workloads. The trained weights of 12 full dataset models were exported from PyTorch (.pt) to ONNX format on the Atlas HPC cluster using the Ultralytics export pipeline (format = 'onnx', imgsz = 640, opset = 17, simplify = True, FP32 precision) and transferred to the Raspberry Pi 5 for inference evaluation. Exported ONNX model sizes ranged from 9.8 MB (YOLO26n) to 103.6 MB (YOLOv8m).

The benchmarking was conducted in three independent sessions to assess measurement reproducibility and to eliminate any influence of the evaluation sequence on the reported latency. While a fixed model evaluation order grouped by size tier (nano, small, medium) was used for the first session, the second and third sessions used randomized model evaluation orders generated with fixed random seeds of 42 and 99, respectively. The Raspberry Pi 5 was rebooted between sessions to fully reset the thermal and memory state. A strict temperature gate of 50 °C was enforced before each model evaluation, and thermal throttle status was verified at the start of each session and confirmed as throttled = 0x0 in all cases. Mean inference latency and SD were reported across the three sessions.

Each model was evaluated on all 200 test images in a session-specific evaluation order following three warm-up inference runs, with standard preprocessing applied by the Ultralytics prediction pipeline (letterbox resizing to 640 × 640, pixel normalization to [0, 1]).

A confidence threshold of 0.25 and an IoU threshold of 0.45 were applied during prediction, consistent with Ultralytics defaults. For NMS-free models (YOLO26), postprocessing consists solely of confidence thresholding, eliminating the sequential sorting and filtering operations inherent to NMS-based architectures. The ONNX Runtime default thread configuration was applied through the Ultralytics prediction pipeline without explicit SessionOptions configuration, leading to intra-op threads equal to the number of physical CPU cores (i.e., 4 threads) with sequential inter-op execution on the quad-core ARM Cortex-A76. Evaluated inference metrics include mean inference time (*Avg*, ms), minimum (*Min*, ms), maximum (*Max*, ms), and 90th percentile (P90, ms) latency, frames per second (FPS), CPU utilization (%), peak RAM consumption (MB), and relative FPS compared to the same-tier YOLOv8 baseline.

The mortality dataset (RAS-MortDB) and trained model weights in PyTorch (.pt) and ONNX (.onnx) formats for all twelve evaluated architectures are publicly available on GitHub (https://github.com/PA-RRanjan/RAS-MortDB, accessed on 6 June 2026). The repository also includes training configuration files for all training runs and the edge inference benchmarking script to support reproducibility and advance open-access aquaculture computer vision research.

3. Results and Discussion

3.1. Mortality Detection

3.1.1. Overall Detection Accuracy

The detection performance of 12 YOLO models trained on the full dataset has been presented in Table 3. All models achieved competitive accuracy, with average mAP50 values across three independent training runs ranging from 93.56% (YOLOv5nu) to 94.81% (YOLO11m), representing a narrow spread of only 1.25 percentage points across all architectures and size tiers. SD across the three training runs was marginal, ranging from 0.10% (YOLO11m) to 0.41% (YOLO26m), confirming strong training reproducibility across all evaluated architectures. Similarly, the mAP50-95 metric, which applies stricter IoU thresholds, ranges from 67.60% (YOLOv5nu) to 71.96% (YOLO26m), indicating generally robust bounding-box localization across all models. The marginal differences in detection performance across the tested models and tiers indicate that, for the binary mortality detection task (dead vs. live), where dead fish exhibit strong visual indicators such as a belly-up posture and cessation of movement, architecture selection has minimal influence on overall detection performance. Even the smallest nano-tier models fully capture the discriminative features with adequate training data (Figure 3). The practical relevance of mAP50-95 in the context of RAS mortality monitoring is noteworthy. While loose detection at IoU of 50% may suffice for mortality count estimation, tighter bounding-box precision is important for distinguishing individual carcasses in high mortality events, where overlapping boxes can negatively affect detection count accuracy. Precision and recall metrics followed similar patterns, with minimal variation across architectures and model sizes. Overall, these findings indicate that newer-generation YOLO architectures, including the recently introduced YOLO26, do not offer a substantial advantage in detection accuracy for this specific application. Despite YOLO26's advanced loss optimization strategies, including ProgLoss and STAL, which are intended to enhance spatial precision for small and densely clustered objects, these improvements did not translate into measurable performance gains for mortality detection in larger fish. Notably, this benchmark dataset primarily comprised larger fish (1.5–2.5 kg), and juvenile mortality scenarios were not represented. Consequently, further investigation is needed to determine whether newer YOLO architectures may provide meaningful advantages for juvenile mortality detection, where smaller body size and potentially more challenging visual characteristics may alter model performance

differentials. Additionally, all reported performance metrics reflect training and evaluation on a single-species, single-facility dataset comprising adult Atlantic salmon in a grow-out RAS configuration. The generalizability of these findings to other species, production systems, or life stages remains to be established in future work.

Table 3. Detection performance of YOLO models trained on the full dataset (2800 images).

Model	Tier	mAP50 (Mean ± SD, %)	mAP50-95 (Mean ± SD, %)	Precision (Mean ± SD, %)	Recall (Mean ± SD, %)
YOLO26n	Nano	94.44 ± 0.38	**71.16 ± 0.70**	**90.94 ± 0.50**	88.02 ± 0.98
YOLO11n		**94.79 ± 0.32**	70.22 ± 0.67	89.13 ± 0.36	**90.10 ± 0.40**
YOLOv8n		94.15 ± 0.39	69.59 ± 0.19	90.02 ± 1.05	88.15 ± 0.94
YOLOv5nu		93.56 ± 0.30	67.60 ± 0.38	89.46 ± 1.34	88.28 ± 1.54
YOLO26s	Small	94.29 ± 0.26	**71.35 ± 0.13**	**91.17 ± 1.32**	88.50 ± 1.46
YOLO11s		**94.54 ± 0.40**	70.77 ± 0.67	90.67 ± 0.53	88.58 ± 1.42
YOLOv8s		94.34 ± 0.36	70.03 ± 0.40	90.85 ± 0.58	88.06 ± 1.19
YOLOv5su		94.09 ± 0.22	69.20 ± 0.25	88.66 ± 1.06	**89.94 ± 1.13**
YOLO26m	Medium	94.44 ± 0.41	**71.96 ± 0.85**	89.49 ± 1.56	89.22 ± 1.26
YOLO11m		**94.81 ± 0.10**	71.86 ± 0.24	90.19 ± 1.35	**89.27 ± 1.30**
YOLOv8m		94.63 ± 0.13	71.34 ± 0.46	90.25 ± 0.68	88.59 ± 0.69
YOLOv5mu		94.52 ± 0.37	70.88 ± 0.43	**90.54 ± 0.60**	88.32 ± 1.10

All performance metrics were reported as mean ± standard deviation (SD) across three independent training runs. Bold values indicate the best performance within each size tier for each metric.

3.1.2. Tier-Level Performance Analysis

Within the nano tier, YOLO11n achieved the highest mAP50 at 94.79 ± 0.32% (mean ± SD), marginally outperforming YOLO26n (94.44 ± 0.38%), YOLOv8n (94.15 ± 0.39%), and YOLOv5nu (93.56 ± 0.30%) [Table 3]. YOLO11s achieved the highest mAP50 (94.54 ± 0.40%) in the small tier, followed by YOLOv8s (94.34 ± 0.36%), YOLO26s (94.29 ± 0.26%), and YOLOv5su (94.09 ± 0.22%). For the medium tier, YOLO11m achieved the highest accuracy (94.81 ± 0.10%), followed by YOLOv8m (94.63 ± 0.13%), YOLOv5mu (94.52 ± 0.37%), and YOLO26m (94.44 ± 0.41%). Repeated training across three independent seeds confirmed that these tier-level rankings are reproducible. Given the narrow mAP50 spread of 1.25 percentage points and SD of up to 0.41%, these tier-level differences should be interpreted with caution as they may fall within normal training variability. Overall, these findings suggest that model architecture and size tier have minimal influence on detection accuracy. Therefore, under the conditions evaluated in this study, model selection should be guided by deployment-related factors rather than marginal differences in accuracy.

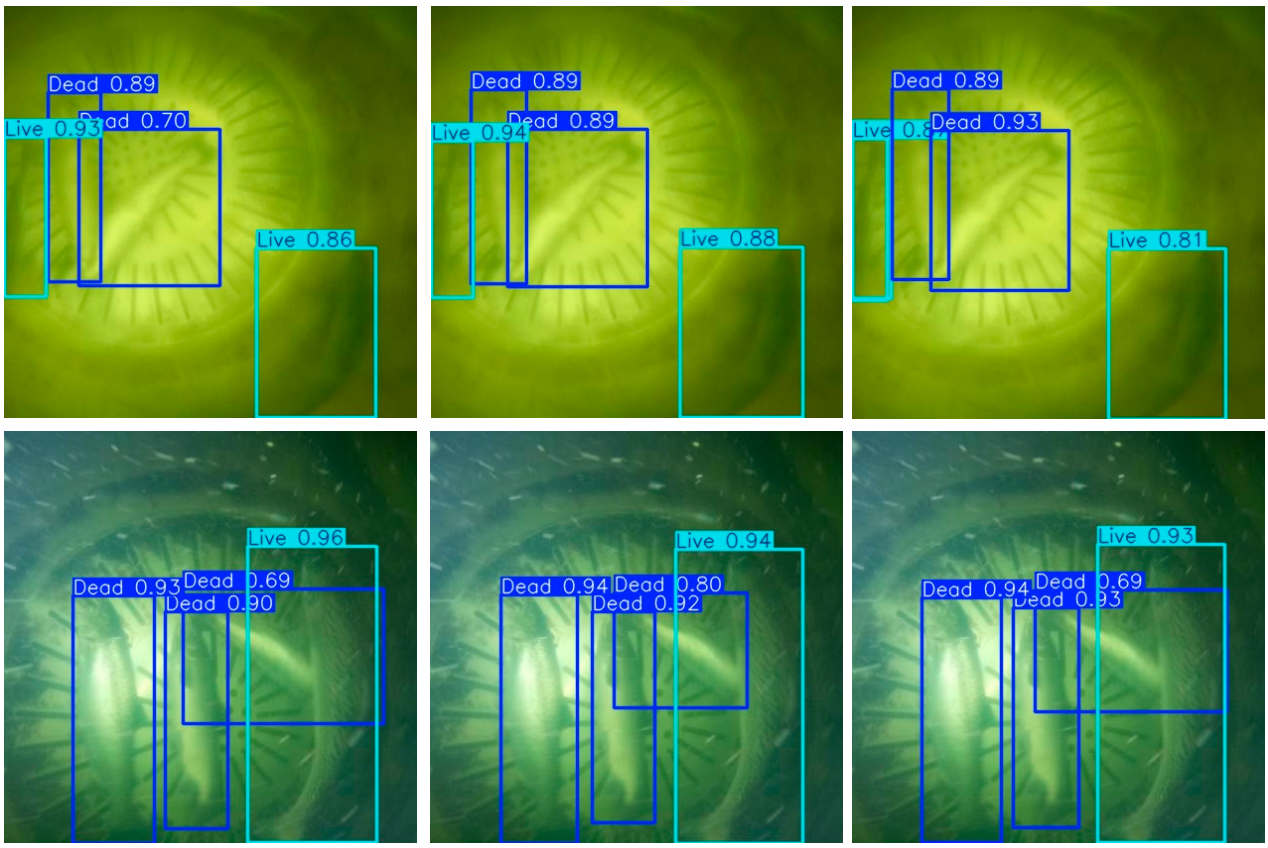


Figure 3. Representative dead and live fish detection output of (**a**) YOLO26n, (**b**) YOLO26s, and (**c**) YOLO26m, along with their detection confidence in ambient (top) and supplementary (bottom) light conditions. All three model variants successfully detected both dead and live fish under both lighting conditions, with confidence ranging from 0.69 to 0.94.

3.1.3. Per-Class Detection Analysis

Across all 12 evaluated models, detection performance for the Dead class (mAP50: 94.9–96.4%; mAP50-95: 71.7–76.0%) consistently exceeded the Live class (mAP50: 92.3–95.1%; mAP50-95: 68.1–74.0%) (Supplementary Table S1). This trend is particularly noteworthy given that Live instances were more abundant in the training dataset, indicating that class frequency alone did not determine detection success. The comparatively inferior performance in Live fish detection is primarily attributable to greater visual variability and image-quality challenges associated with live fish behavior. Live fish exhibited rapid, continuous, and unpredictable movement around the bottom drain plates, frequently resulting in motion blur observed in the training images that reduced feature clarity. Moreover, live fish often swam close to the camera, causing partial body captures and incomplete object representation within the frame. These factors introduced substantial within-class variability in orientation, body position, depth, and motion state, thereby increasing the complexity of precise object localization and bounding-box assignment. In contrast, dead fish displayed slower, more consistent movement driven primarily by water currents and were typically concentrated near the bottom drain plate region, which remained consistently within the camera's field of view. As a result, images of dead fish were generally clearer, with more complete body visibility and reduced positional variability. The higher visual consistency and improved image quality of Dead class samples likely contributed to the superior detection accuracy observed across all models. This per-class performance pattern is consistent with previous findings reported by Ranjan et al. [16], who similarly

observed stronger detection confidence for Dead class instances in a YOLOv7-based RAS mortality monitoring system. Notably, this Dead and Live accuracy pattern was consistent across all twelve evaluated architectures. From an operational perspective, this uniformity suggests that class-specific detection performance is unlikely to be a differentiating factor in model selection under the evaluated tank configuration and fish population. Therefore, for practical deployment, model selection may be better guided by computational efficiency and inference speed requirements rather than by marginal differences in class-specific detection performance.

3.2. GPU Inference Speed

YOLO26 demonstrated the lowest postprocessing (*Post*, ms) latency across all three size tiers. In the nano tier, YOLO26n achieved the fastest post-processing (0.179 ms), accounting for only 18.0% of the combined inference and post-processing time. In comparison, YOLOv5nu reported 0.706 ms (44.2%), YOLOv8n recorded a markedly higher post-processing time of 1.805 ms (67.8%), and YOLO11n reported 0.702 ms (47.1%), indicating substantially higher post-processing overhead in the latter models relative to YOLO26n. The postprocessing overhead of YOLO26n was 10.1 times lower than that of YOLOv8n. A similar advantage was observed in the medium tier, where YOLO26m achieved a near-negligible post-processing time (0.128 ms), accounting for only 4.2% of the combined inference and post-processing budget. Compared to YOLO11m, which required 1.587 ms (40.2%), and YOLOv8m, which required 0.840 ms (27.5%), YOLO26m exhibited a substantially lower overhead. YOLO26m exhibited a 6.6 times lower post-processing overhead than YOLOv8m. Similarly, for the small tier, the post-processing overhead of YOLO26s was 1.7 times lower than YOLOv8s. Such efficiency gain can be attributed to YOLO26's NMS-free end-to-end inference design, which handles duplicate suppression within the network, eliminating the need for an explicit NMS step and thereby reducing post-processing complexity and latency. Unlike post-processing time, the raw forward-pass inference time (*Infer*, ms) shows no consistent trend across model architectures and tested tiers. YOLO11n achieved the fastest forward-pass inference of 0.787 ms, followed closely by YOLO26n (0.813 ms) in the nano tier. In the medium tier, YOLOv5mu was fastest at 2.026 ms, while YOLO26m was slowest (2.919 ms), attaining a 44% longer forward pass relative to YOLOv5mu.

Despite YOLO26's NMS-free design, resulting in low post-processing overhead, this architecture had the highest end-to-end average latency (ms) and the lowest inference speed (FPS) across all size tiers. In the nano tier, YOLO26n had an average latency of 11.20 ms, resulting in an end-to-end inference speed of 89.31 FPS, compared to YOLOv8n at 9.18 ms (108.93 FPS), YOLOv5nu at 9.67 ms (103.41 FPS), and YOLO11n at 10.46 ms (95.60 FPS). The small and medium tiers followed the same trend. YOLOv8 consistently achieved the highest FPS and lowest average latency across all the size tiers. This discrepancy can be explained by examining the relationship between the measured inference components [i.e., forward-pass inference time (*Infer*) + Postprocessing time (*Post*)] and the reported average end-to-end latency (*Avg*). Beyond the *Infer* and *Post*, the *Avg* includes residual latency components related to additional pipeline overhead, such as image preprocessing, data loading, tensor allocation, and GPU-to-CPU memory transfers. YOLO26 variants incur the highest residual latency (i.e., *Avg-Infer-Post*) across all tiers, with the values of 10.21 ms (nano), 9.27 ms (small), and 8.52 ms (medium), compared to YOLOv8 at 6.52 ms, 6.83 ms, and 7.03 ms, respectively. This elevated residual overhead directly contributes to the higher end-to-end latency observed in YOLO26, offsetting the gains achieved through reduced post-processing. While the elimination of NMS reduces post-processing cost, YOLO26's one-to-one assignment head introduces additional tensor allocation and

output decoding operations within the inference pipeline, which contribute to higher end-to-end average latency relative to NMS-dependent architectures on GPU hardware [35,37].

Notably, all twelve models comfortably exceeded 30 FPS, the widely referenced minimum threshold for real-time video processing [21], by a margin of 2.8 to 3.8 times. For RAS mortality monitoring, where a fixed overhead camera observes a single tank and mortality events develop over minutes to hours, a detection rate of 5–10 FPS may provide sufficient temporal resolution for continuous monitoring, although alert-level performance was not formally evaluated in this study. All tested architectures, regardless of tier, provide substantially higher GPU throughput. While such high inference speeds achieved on high-end GPUs may be advantageous for more temporally demanding aquaculture tasks (e.g., feed waste monitoring, where small, fast-moving pellets require fine-grained temporal resolution and faster response for feeding optimization), the primary design constraint for aquaculture mortality monitoring shifts from maximizing FPS to optimizing cost efficiency, energy consumption, and deployment scalability. Therefore, low-power edge accelerators or CPU-based embedded systems may represent more practical and resource-efficient deployment choices without compromising operational effectiveness.

3.3. Computational Efficiency

Model efficiency (unitless), evaluated as average mAP50 per MB of model size to investigate accuracy-to-storage trade-off, revealed substantial variation across architectures and tiers. The nano tier achieved the highest efficiency (15.0–17.7), followed by the small (4.2–5.1) and medium (1.8–2.3) size tiers (Table 4). Therefore, nano-tier models may be best suited for edge devices with storage or memory constraints, such as microcontrollers or IoT devices with limited flash storage. For the nano tier, YOLOv5nu (PT model size = 5.3 MB; mAP50 = 93.56%; number of parameters = 2.5M) achieved the highest efficiency. However, maximizing model efficiency does not necessarily align with all deployment objectives. In scenarios where storage is not a constraint and peak detection accuracy is prioritized, larger models such as YOLO11m (PT size = 40.5 MB; mAP50 = 94.81%; 20.1 M parameters) may be preferable due to their slightly higher mAP. A similar trend was observed for training efficiency, with nano-tier models requiring the least training time, followed by the small and medium tiers. Across all tiers, YOLO26 variants did not demonstrate advantages in either model efficiency or training efficiency relative to earlier YOLO architectures.

Table 4. GPU inference speed and computational efficiency for YOLO models trained on the full dataset (2800 images).

Model	Tier	*Infer* (ms)	*Post* (ms)	*Avg* (ms)	GPU FPS	PT (MB)	mAP50/MB (×100)	Params (M)	Train (h)
YOLO26n	Nano	0.813	**0.179**	11.2	89.31	5.4	17.49	2.5	0.396
YOLO11n		**0.787**	0.702	10.46	95.6	5.5	17.23	2.6	0.328
YOLOv8n		0.857	1.805	**9.18**	**108.93**	6.3	14.95	3	**0.312**
YOLOv5nu		0.892	0.706	9.67	103.41	**5.3**	**17.65**	2.5	0.318
YOLO26s	Small	1.472	**0.461**	11.2	89.15	20.3	4.64	9.9	0.478
YOLO11s		1.78	0.792	10.91	91.66	19.2	4.92	9.4	0.373
YOLOv8s		**1.23**	0.791	**8.85**	**112.99**	22.5	4.19	11.1	**0.348**
YOLOv5su		1.539	0.523	9.46	105.71	**18.5**	**5.09**	9.1	0.353

Table 4. *Cont.*

Model	Tier	*Infer* (ms)	*Post* (ms)	*Avg* (ms)	GPU FPS	PT (MB)	mAP50/MB (×100)	Params (M)	Train (h)
YOLO26m	Medium	2.919	**0.128**	11.57	86.45	44	2.15	21.8	0.693
YOLO11m		2.365	1.587	11.8	84.77	**40.5**	**2.34**	20.1	0.592
YOLOv8m		2.214	0.84	**10.08**	**99.16**	52	1.82	25.9	0.524
YOLOv5mu		**2.026**	1.229	10.72	93.3	50.5	1.87	25.1	**0.514**

Infer (ms): raw GPU forward pass inference time; *Post* (ms): postprocessing time; *Avg* (ms): end-to-end average latency (ms) on A100 GPU; GPU FPS = end-to-end inference time (frame per second) on A100 GPU; PT (MB): PyTorch model size in Megabytes; Params (M): Number of model parameters in millions; Reported mAP50/MB evaluated from averaged mAP50 across three independent training runs (Table 3). Bold values indicate the best performance within each size tier for each metric.

3.4. Learning Curve Analysis

All learning curve dataset sizes refer to augmented training images sampled from the full training set comprising 1400 original and 1400 corresponding augmented images (Table 2). The strong training reproducibility demonstrated across three independent seeds for the full dataset (SD range: 0.10–0.41%) supports the reliability of the single-run learning curve findings reported here. Learning curve analysis revealed a consistent logarithmic improvement in mAP50, with most performance gains occurring between 100 and 1000 training images across all evaluated models (Figure 4). Of the 12 models evaluated, four achieved mAP50 $\geq$ 90% with only 400 training images (i.e., 200 original images and 200 corresponding augmented images), while six reached this threshold with 700 training images (i.e., 350 original images and 350 corresponding augmented images) (Figure 5). The YOLOv8 architecture demonstrated the highest training efficiency, as all variants (YOLOv8n, YOLOv8s, and YOLOv8m) attained $\geq$90% mAP50 with as few as 400 training images. In contrast, none of the YOLO26 models reached this performance level at 400 images. The YOLOv5u architecture showed comparable efficiency, requiring 400 to 700 images depending on the model size. Beyond 1000 images, the model performance gains across all tested models were marginal. This trend is consistent with COCO-pretrained transfer learning models applied to domain-specific datasets, wherein pretrained feature representations capture a substantial portion of the target domain's visual structure with limited fine-tuning data. Therefore, additional training images primarily refine decision boundaries rather than learning new feature representations [10,38]. Overall, YOLO26 was the least data-efficient architecture under the tested conditions, as its nano (YOLO26n) and small (YOLO26s) tier models required 1000 training images (500 original images and 500 corresponding augmented images) to surpass the $\geq$90% mAP50 threshold, approximately 2.5 times more original annotated images than their YOLOv8 counterparts (400 training images; 200 original and 200 corresponding augmented images). Only the medium tier of YOLO26 could reach this performance benchmark with 700 training images.

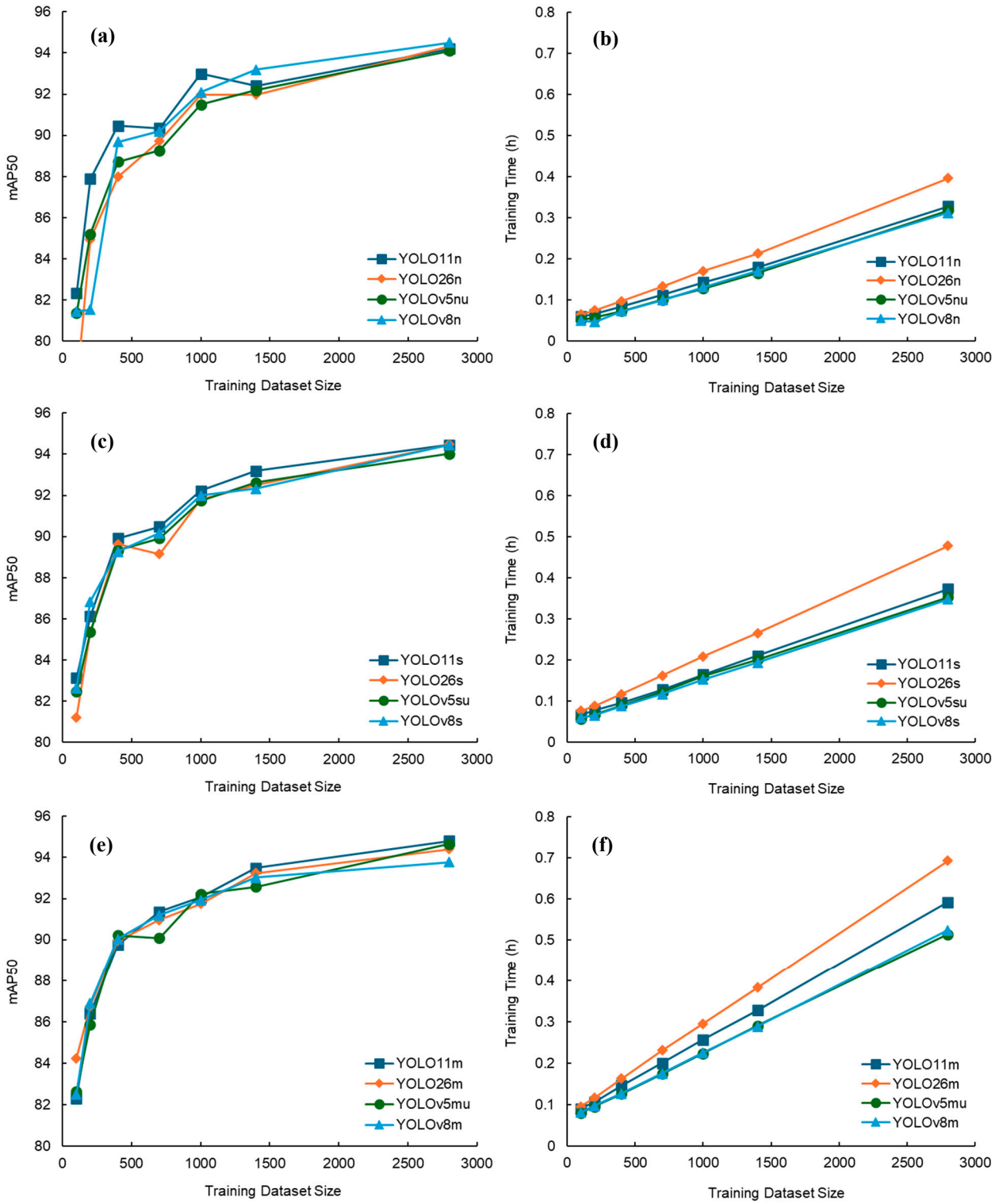


Figure 4. The learning curve analysis output illustrating the (**a**,**c**,**e**) mAP50 performance and (**b**,**d**,**f**) training time (hours) of YOLOv5u, YOLOv8, YOLO11, and YOLO26 across nano-, small-, and medium-tier models as a function of increasing training dataset size. The learning curve experiments were conducted with a single training run per dataset size.

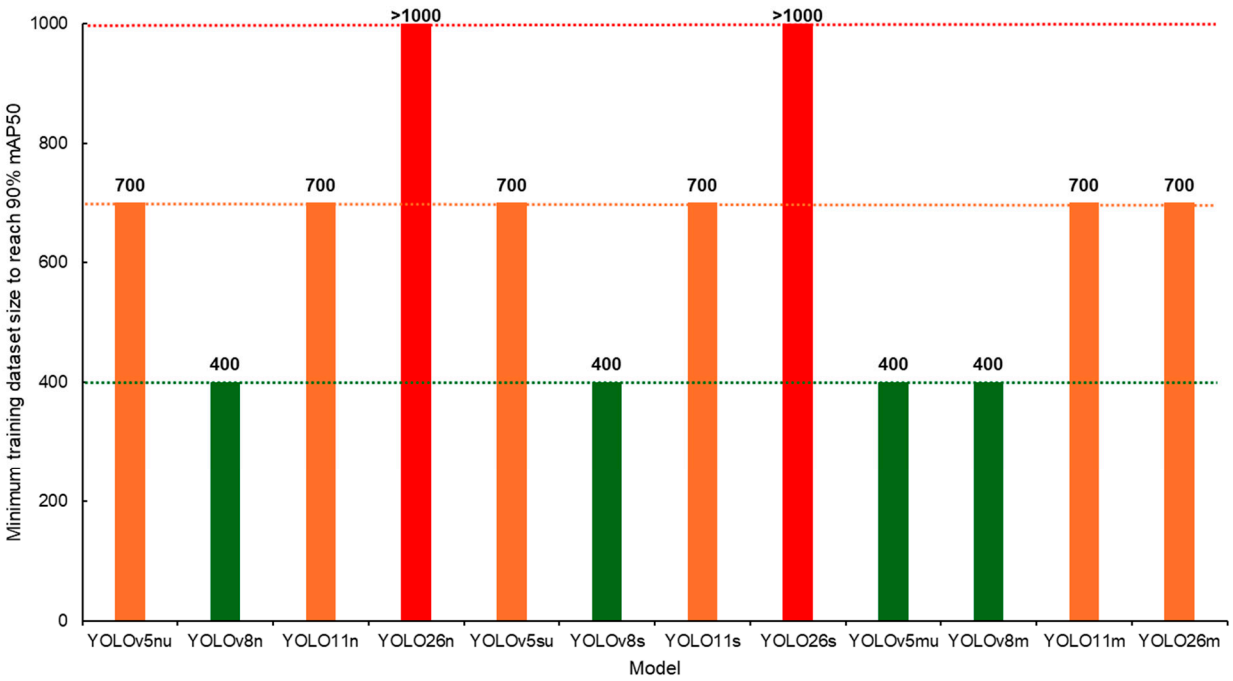


Figure 5. Minimum number of training images required to achieve ≥90% mAP50 on the validation set across different model variants. Models highlighted in green achieved this performance threshold with as few as 400 training images; those in orange required approximately 700 images, while models shown in red did not reach ≥90% mAP50 even with up to 1000 training images.

While performance improvements beyond 1000 training images were marginal, the associated training time increased substantially. For example, the YOLOv8 models achieved only a 1.8–2.5% increase in mAP50 when trained on the full dataset (2800 images) relative to 1000 images; however, training time increased by approximately 2.2–2.4 times across model size tiers. A similar trend was observed for other models. Notably, the time required to train the YOLO26 architecture was consistently higher than that of its predecessor models across all size tiers [Figure 4b,d,f]. The superior data and training efficiency of YOLOv8 can be attributed to its anchor-free, decoupled detection head, which simplifies label assignment and improves optimization of classification and localization tasks. These architectural advancements contribute to faster convergence and improved performance, particularly under limited data conditions [34,37]. In contrast, YOLO26 employs a dual-head training architecture comprising an auxiliary one-to-many head for dense supervision and a one-to-one inference head (Table 5) [32]. ProgLoss gradually shifts training emphasis toward the one-to-one inference head, reducing the effective supervision received by the inference head relative to predecessor architectures that use a single many-to-one assignment strategy for both training and inference. Additionally, the removal of DFL simplifies bounding box regression in YOLO26, which may provide less precise localization supervision during training compared to architectures that retain DFL [32,34]. These factors likely contribute to the higher volume of training data required for YOLO26 to achieve comparable convergence. Furthermore, the computationally intensive matching process during label assignment contributes to longer training times compared to architectures employing many-to-one assignment strategies, as reflected in the longer training durations observed for YOLO26 variants across all size tiers in this study.

Table 5. Comparison of label assignment, loss function, and supervision strategies across evaluated Ultralytics YOLO architectures.

Architecture	Label Assignment	Detection Loss	Supervision
YOLOv5u	Many-to-one (anchor-free)	BCE + CIoU	Dense
YOLOv8	Many-to-one (anchor-free, TAL)	BCE + DFL + CIoU	Dense
YOLO11	Many-to-one (anchor-free, TAL)	BCE + DFL + CIoU	Dense
YOLO26	Dual assignment (one-to-many + one-to-one, STAL)	BCE + CIoU (DFL removed; ProgLoss training)	Hybrid (dense + sparse)

TAL: Task-Aligned Learning; BCE: Binary Cross-Entropy; CIoU: Complete IoU; DFL: Distribution Focal Loss; STAL: Small-Target-Aware Label Assignment.

Overall, the observed logarithmic saturation, coupled with the increased training time associated with larger datasets, indicates that, despite the availability of extensive training data at the farm level, further annotation yields diminishing returns in accuracy. Therefore, additional resources may be more effectively allocated to improving image quality, increasing the diversity of mortality scenarios by collecting data for entire growth stages, or enhancing deployment infrastructure. Additionally, while the latest YOLO models promise deployment advantages for edge devices, this study demonstrates that these gains come at the cost of reduced training efficacy and higher data requirements. Therefore, architectural generation should not be the sole criterion for model selection in aquaculture applications.

3.5. Edge Deployment Performance

Initial deployment trials on the Raspberry Pi 5 without active cooling resulted in CPU throttling within minutes of sustained inference, with peak temperatures above 80 °C, indicating thermal soft-limit conditions that reduce the CPU operating frequency below the nominal 2.4 GHz. Following the active cooling integration described in Section 2.5, CPU temperature remained stable throughout all benchmark sessions (idle: 44.4 °C; peak: <70 °C), and no throttling was observed (throttle status: 0x0), confirming unthrottled operation at 2.4 GHz across all reported benchmark runs.

Inference latency and FPS measurements demonstrated high reproducibility across the three benchmark sessions, with marginal latency SD ranging from 0.9 to 11.6 ms and FPS SD ranging from 0.00 to 0.13 FPS across all twelve models, confirming measurement stability across sessions with randomized model evaluation order (Table 6). All nano-tier models achieved inference speeds exceeding 6 FPS on the Raspberry Pi 5, which provides sufficient temporal resolution for near-real-time mortality detection in RAS, where mortality events develop over minutes rather than seconds. YOLO26n achieved the highest inference speed in the nano tier at 127.5 ± 2.2 ms per image (7.84 ± 0.13 FPS), outperforming YOLOv5nu (139.9 ± 1.2 ms; 7.15 ± 0.06 FPS), YOLO11n (149.3 ± 2.5 ms; 6.70 ± 0.11 FPS), and YOLOv8n (155.2 ± 2.0 ms; 6.44 ± 0.08 FPS) by 9.7%, 17.0%, and 21.7%, respectively. This finding empirically validates YOLO26's claimed CPU inference advantages [32] in a domain-specific aquaculture deployment context, extending this claim beyond general-purpose COCO benchmarks. The observed speed advantage of YOLO26 can be primarily attributed to the elimination of NMS and DFL postprocessing steps, which reduce CPU computational burden and yield faster, more consistent per-image inference compared to NMS-dependent architectures [32,34,35,39]. Although inference was performed using ONNX Runtime, the observed 17.0% improvement over YOLO11n was considerably smaller than the up to 43% CPU ONNX inference speedup reported by Ultralytics. This difference is likely attributable to the use of a resource-constrained Raspberry Pi 5 ARM processor rather than

the Intel Xeon platform used in the official benchmark, as well as differences in deployment environment, model optimization, and application workload.

Table 6. Raspberry Pi 5 inference benchmark results for all 12 YOLO models using ONNX format on 200 test images.

Model	Tier	*Avg* (Mean ± SD, ms)	*Min* (ms)	*Max* (ms)	P90 (ms)	CPU FPS (Mean ± SD)	CPU (%)	Peak RAM (MB)	ONNX (MB)	Relative FPS % (vs. YOLOv8)
YOLO26n	Nano	127.5 ± 2.2	125.7	140.4	128.2	**7.84 ± 0.13**	49.9	531	9.8	+21.7%
YOLO11n		149.3 ± 2.5	145.2	188.3	150.3	6.70 ± 0.11	48.7	531	10.6	+4.0%
YOLOv8n		155.2 ± 2.0	151.4	201.2	156.7	6.44 ± 0.08	48.8	531	12.3	Baseline
YOLOv5nu		139.9 ± 1.2	136.4	156.6	145.1	7.15 ± 0.06	48.7	531	10.3	+11.0%
YOLO26s	Small	347.4 ± 4.2	343.7	389	348.8	**2.88 ± 0.03**	50.1	626	38.2	+16.6%
YOLO11s		361.4 ± 4.5	356.9	402.3	362.8	2.77 ± 0.04	49.6	626	37.9	+12.1%
YOLOv8s		405.1 ± 11.3	397.9	594.6	404.2	2.47 ± 0.07	49.6	626	44.7	Baseline
YOLOv5su		347.7 ± 3.5	343.1	390.4	349.4	2.88 ± 0.03	49.5	626	36.7	+16.6%
YOLO26m	Medium	955.4 ± 9.6	949.8	1005	957.1	1.05 ± 0.01	50.2	787	81.7	0.0%
YOLO11m		963.7 ± 10.8	957.6	1003.6	966.1	1.04 ± 0.01	49.9	787	80.4	−1.0%
YOLOv8m		949.5 ± 11.6	943.1	987.7	951.3	1.05 ± 0.01	49.9	787	103.6	Baseline
YOLOv5mu		792.5 ± 0.9	785.3	834.4	796.4	**1.26 ± 0.00**	50	787	100.5	+20.0%

Avg (ms): Mean end-to-end inference latency per image across 200 test images, reported as mean ± standard deviation (SD) across three independent benchmark sessions; *Min* (ms): Mean minimum single-image inference latency across the three sessions; *Max* (ms): Mean maximum single-image inference latency across the three sessions; P90 (ms): Mean 90th percentile inference latency across the three sessions, representing the worst-case latency experienced by 90% of processed images; CPU (%): Mean CPU utilization measured during inference; Peak RAM (MB): Peak process resident set size during inference, averaged across three sessions; FPS: Frames per second computed as 1000; *Avg* (ms), reported as mean ± SD across three sessions; ONNX (MB): File size of the exported ONNX model used for inference; Relative FPS %: Percentage improvement in FPS relative to the same-tier YOLOv8 baseline. Bold values indicate the highest FPS achieved within each size tier.

Similar trends were observed for the small tier. YOLO26s achieved the top inference speed of 347.4 ± 4.2 ms per image (2.88 ± 0.03 FPS), followed by YOLOv5su (347.7 ± 3.5 ms; 2.88 ± 0.03 FPS), YOLO11s (361.4 ± 4.5 ms; 2.77 ± 0.04 FPS), and YOLOv8s (405.1 ± 11.3 ms; 2.47 ± 0.07 FPS). The most unexpected finding from the edge inference benchmark emerged in the medium tier. Unlike the consistent dominance of YOLO26 in the nano and small tiers, YOLOv5mu substantially outperformed all medium-scale architectures, achieving 792.5 ± 0.9 ms per image (1.26 ± 0.00 FPS) compared to YOLOv8m (949.5 ± 11.6 ms; 1.05 ± 0.01 FPS), YOLO26m (955.4 ± 9.6 ms; 1.05 ± 0.01 FPS), and YOLO11m (963.7 ± 10.8 ms; 1.04 ± 0.01 FPS). The inference speed rankings observed on Raspberry Pi 5 suggest the interplay of two competing architectural factors on ARM CPU hardware. First, eliminating NMS postprocessing in YOLO26 likely reduces sequential CPU overhead, which may explain why YOLO26n and YOLO26s led their respective tiers. Unlike NMS, which performs sequential sorting and suppression operations that scale with scene complexity, YOLO26's one-to-one inference head produces a fixed number of outputs with constant latency regardless of scene content. Second, architectural complexity in the forward pass may introduce additional tensor operations, graph branching, and output decoding steps that are less efficiently executed on ARM processors compared to GPU hardware [34,40]. At medium scale, this graph complexity penalty may become the dominant factor, as YOLO26m, YOLO11m, and YOLOv8m all incorporate more complex backbone and neck

operations than their nano and small counterparts, resulting in slower CPU inference. YOLOv5mu, by contrast, retains a simpler convolutional graph without complex attention or one-to-one assignment mechanisms, which may explain its superior CPU inference performance at medium scale despite being the oldest architecture evaluated [34,40]. A direct profiling of ONNX operator execution, however, would be required to confirm these hypotheses. Regardless of the underlying mechanism, this finding reinforces that GPU inference rankings are unreliable proxies for edge deployment performance.

The inference speeds of small and medium-tier models (1.04–2.88 FPS) indicate that these architectures are better suited for periodic snapshot-based inspection at intervals of several seconds to minutes rather than continuous near-real-time monitoring on resource-constrained CPU-based edge hardware. Despite this, the edge-inference speed of all the tested models remains operationally viable for the mortality monitoring task. For tasks that require higher inference speeds, such as feed waste monitoring or behavioral analysis, GPU-accelerated embedded platforms may be a more suitable deployment solution.

ONNX model size across all tiers correlated closely with parameter count, with YOLOv8 consistently producing both the highest parameter count and the largest ONNX exports in every tier. YOLO26n produced the smallest ONNX export at 9.8 MB across all twelve models. For the Raspberry Pi 5 with 8 GB RAM used in this study, all twelve models loaded within available memory without issue. However, for deployment on more constrained embedded hardware with limited storage or RAM, the compact ONNX footprints of YOLO26n (9.8 MB) and YOLO11m (80.4 MB) at their respective tiers offer practical advantages in memory allocation overhead and model initialization speed for storage-limited on-farm edge devices.

CPU utilization was consistent at approximately 49–50% across all twelve models, regardless of architecture or model size tier, indicating that observed inference speed differences are attributable to computational graph efficiency rather than differences in CPU resource utilization. Peak RAM consumption scaled with model size tier rather than architecture, averaging 531 MB, 626 MB, and 787 MB for nano, small, and medium tier models, respectively. These findings confirm that the evaluated models operate comfortably within the 8 GB RAM of the Raspberry Pi 5.

Overall, the latest YOLO26 architecture met several of its intended design objectives, though the benefits were selective for this domain-specific benchmark study. The NMS-free and DFL-free design of YOLO26 produced substantial improvements in CPU inference performance on the Raspberry Pi 5. Among the nano and small model tiers, YOLO26n achieved the highest inference speed, validating its suitability for faster CPU-based deployment in practical aquaculture applications. In addition, the compact ONNX export size of YOLO26 supports deployment under resource-constrained on-farm conditions. The NMS-free inference design of YOLO26 improved CPU inference efficiency; however, its dual-head training strategy, which progressively shifts supervision toward the one-to-one inference head, introduced training-related trade-offs. The YOLO26 nano and small variants required approximately 2.5 times more annotated images than YOLOv8 to surpass the 90% mAP50 threshold and consistently incurred higher training-time costs. In contrast, YOLOv8 demonstrated balanced performance across the entire deployment pipeline. The model converged with comparatively limited training data, delivered strong GPU throughput, and transferred efficiently to the ONNX runtime without the pipeline overhead penalties observed in YOLO26. YOLOv8 also exhibited consistent behavior across both GPU and CPU deployment environments. While the anchor-free, decoupled head of YOLOv8 resulted in larger ONNX export sizes, this model produced a stable, well-optimized inference graph with predictable runtime characteristics. These advantages have likely contributed to the broader adoption of YOLOv8 in aquaculture applications [29], and in other computer

vision domains. The architectural trade-offs among YOLO generations are substantial, and model selection should be guided by application-specific constraints, including annotated data availability, deployment hardware, and real-time inference requirements.

4. Conclusions

The research question motivating this benchmarking study, whether the NMS-free architecture of YOLO26 represents a meaningful advancement over earlier YOLO generations for fish mortality detection in aquaculture, does not yield a definitive answer. The findings demonstrate that architectural generation alone is an insufficient criterion for model selection, and architectural novelty did not translate to model superiority under the conditions evaluated in this study. All 12 trained models across the architectural and size tiers converged within 1.25 percentage points of mAP50 for mortality detection, with training reproducibility confirmed across three independent seeds (SD: 0.10–0.41%). Therefore, the primary differentiators between the models are not detection accuracy but deployment-related factors such as the availability of annotated data, computational hardware constraints, and inference speed requirements. Practitioners with sufficient annotated datasets targeting real-time edge monitoring should prioritize YOLO26n, while those operating under data or computational constraints may find predecessor architectures more appropriate. Additionally, findings from this benchmark study highlight an important limitation of relying solely on GPU-based benchmarking as a proxy for edge deployment performance. The inference speed rankings observed on the A100 GPU did not translate to the CPU-based Raspberry Pi 5 and would have led to suboptimal model selection if GPU performance had been used as the sole deployment criterion. These results reinforce the need for deployment-oriented evaluation on target edge hardware in future studies, as processor architecture, memory constraints, and runtime-specific graph optimization can substantially affect edge performance relative to GPU benchmarks.

Although architectural advancements contribute incremental improvements in detection performance, greater emphasis should be placed on the more fundamental challenges of developing diverse, real-world training datasets and conducting rigorous optimization and validation under commercial-scale aquaculture conditions. For this benchmark study, sufficient training images were collected during the grow-out stage under two lighting conditions over a 90-day period; however, the dataset lacked mortality data from juvenile production phases and represented only a single grow-out tank configuration. The generalizability of the models to other species, facility designs, or production environments remains unverified. Furthermore, high-mortality scenarios in this dataset were simulated by manually introducing euthanized culls on the final day of the data collection period, rather than capturing natural mortality events triggered by disease outbreaks or environmental stress. In real production environments, mortality events may involve decomposing carcasses, fish piling, and elevated water turbidity that differ substantially from the controlled simulation used in this study. Model validation under such conditions remains unexplored and warrants investigation in future field-based studies. Factors such as varying water turbidity, camera positioning, and long-term field robustness under continuous commercial operation were also not systematically evaluated and may influence model performance in real-world deployment scenarios. Moreover, the edge inference benchmark was conducted on a single Raspberry Pi 5 device across three independent sessions and evaluated exclusively with ONNX Runtime. Alternative deployment frameworks, including TensorFlow Lite (TFLite) and OpenVINO, were not assessed, and the benchmark was limited to a single edge device in this study. Therefore, the findings of this study should be interpreted as context-specific, evidence-based guidance rather than universally generalizable deployment recommendations.

Future work should expand benchmarking efforts across various aquaculture production systems (e.g., net-pen, RAS), species, and growth stages to establish the cross-context generalizability required for industry-wide deployment recommendations. Furthermore, evaluation of the developed models on GPU-accelerated embedded platforms, such as the NVIDIA Jetson series, would provide a more comprehensive understanding of performance across diverse edge hardware environments and may reveal inference advantages of newer architectures that were not fully apparent under the CPU-constrained conditions evaluated in this study.

Supplementary Materials: The following supporting information can be downloaded at: https://www.mdpi.com/article/10.3390/ai7090354/s1, Table S1: Per-class detection performance (mAP50 and mAP50-95) for Dead and Live fish classes across all twelve evaluated YOLO model variants.

Author Contributions: R.R.: Conceptualization, Methodology, Investigation, Formal analysis, Funding acquisition, Writing—original draft; G.S.K.: Writing—review and editing, Data curation; K.S.: Writing—review and editing, Data curation; S.T.: Writing—review and editing, Data curation; C.G.: Writing—review and editing, Funding acquisition. All authors have read and agreed to the published version of the manuscript.

Funding: This study was supported by the USDA Agricultural Research Service agreement numbers 59-8082-5-002. Use of tradenames is solely to provide accurate information and does not imply endorsement by the USDA. The Conservation Fund and USDA are equal opportunity providers and employers. All experimental protocols and methods were approved by the Freshwater Institute's Institutional Animal Care and Use Committee (IACUC Protocol No. 1-2022, approved 13 January 2022). Culled fish used in the simulated high-mortality event were euthanized with an overdose of MS-222 before being introduced into the tank, in accordance with the approved IACUC protocol.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The fish mortality dataset (RAS-MortDB) used in this study, along with the trained model weights in PyTorch (.pt) and ONNX (.onnx) formats for all twelve evaluated architectures, training configuration files for all training runs, and the edge inference benchmarking script, are publicly available on GitHub (https://github.com/PA-RRanjan/RAS-MortDB, accessed on 6 June 2026) and archived on Zenodo (https://doi.org/10.5281/zenodo.20631780).

Acknowledgments: The author extends their sincere thanks to Brian Vinci and Travis May for their valuable suggestions and assistance with mortality data collection. During the preparation of this work, Generative AI tools [Claude AI (Sonnet 4.6) and ChatGPT (4o)] were used solely to improve the readability and language of the manuscript. All AI-assisted revisions were thoroughly reviewed and edited for accuracy and relevance.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Bartley, D.M. *World Aquaculture 2020—A Brief Overview*; FAO Fisheries and Aquaculture Circular No. 1233; Food and Agriculture Organization of the United Nations: Rome, Italy, 2022. [CrossRef]
2. Ahmed, N.; Turchini, G.M. Recirculating aquaculture systems (RAS): Environmental solution and climate change adaptation. *J. Clean. Prod.* **2021**, *297*, 126604. [CrossRef]
3. Rust, M.B.; Amos, K.H.; Bagwill, A.L.; Dickhoff, W.W.; Juarez, L.M.; Price, C.S.; Morris, J.A., Jr.; Rubino, M.C. Environmental performance of marine net-pen aquaculture in the United States. *Fisheries* **2014**, *39*, 508–524. [CrossRef]
4. Bostock, J.; McAndrew, B.; Richards, R.; Jauncey, K.; Telfer, T.; Lorenzen, K.; Little, D.; Ross, L.; Handisyde, N.; Gatward, I.; et al. Aquaculture: Global status and trends. *Philos. Trans. R. Soc. B Biol. Sci.* **2010**, *365*, 2897–2912. [CrossRef] [PubMed]
5. Miao, W.; Lal, K.K. (Eds.) *Sustainable Intensification of Aquaculture in the Asia-Pacific Region: Documentation of Successful Practices*; Food and Agriculture Organization of the United Nations: Bangkok, Thailand, 2016.
6. Engle, C.R. The economics of recirculating aquaculture systems. *J. World Aquacult. Soc.* **2023**, *54*, 782–785. [CrossRef]

7. O'Donncha, F.; Grant, J. Precision aquaculture. *IEEE Internet Things Mag.* **2019**, *2*, 26–30. [CrossRef]
8. Føre, M.; Frank, K.; Norton, T.; Svendsen, E.; Alfredsen, J.A.; Dempster, T.; Berckmans, D. Precision fish farming: A new framework to improve production in aquaculture. *Biosyst. Eng.* **2018**, *173*, 176–193. [CrossRef]
9. Hu, H.; Tang, C.; Shi, C.; Qian, Y. Detection of residual feed in aquaculture using YOLO and Mask RCNN. *Aquac. Eng.* **2023**, *100*, 102304. [CrossRef]
10. Ranjan, R.; Sharrer, K.; Tsukuda, S.; Good, C. Effects of image data quality on a convolutional neural network trained in-tank fish detection model for recirculating aquaculture systems. *Comput. Electron. Agric.* **2023**, *205*, 107644. [CrossRef]
11. Hu, X.; Liu, Y.; Zhao, Z.; Liu, J.; Yang, X.; Sun, C.; Chen, S.; Li, B.; Zhou, C. Real-time detection of uneaten feed pellets in underwater images for aquaculture using an improved YOLO-V4 network. *Comput. Electron. Agric.* **2021**, *185*, 106135. [CrossRef]
12. Wang, H.; Zhang, S.; Zhao, S.; Wang, Q.; Li, D.; Zhao, R. Real-time detection and tracking of fish abnormal behavior based on improved YOLOV5 and SiamRPN++. *Comput. Electron. Agric.* **2022**, *192*, 106512. [CrossRef]
13. Wang, Z.; Zhang, X.; Su, Y.; Li, W.; Yin, X.; Li, Z.; Ying, Y.; Wang, J.; Wu, J.; Miao, F.; et al. Abnormal behavior monitoring method of *Larimichthys crocea* in recirculating aquaculture system based on computer vision. *Sensors* **2023**, *23*, 2835. [CrossRef] [PubMed]
14. Zhang, T.; Yang, Y.; Liu, Y.; Liu, C.; Zhao, R.; Li, D.; Shi, C. Fully automatic system for fish biomass estimation based on deep neural network. *Ecol. Inform.* **2024**, *79*, 102399. [CrossRef]
15. Cai, Y.; Yao, Z.; Jiang, H.; Qin, W.; Xiao, J.; Huang, X.; Pan, J.; Feng, H. Rapid detection of fish with SVC symptoms based on machine vision combined with a NAM-YOLO v7 hybrid model. *Aquaculture* **2024**, *582*, 740558. [CrossRef]
16. Ranjan, R.; Sharrer, K.; Tsukuda, S.; Good, C. MortCam: An artificial intelligence-aided fish mortality detection and alert system for recirculating aquaculture. *Aquac. Eng.* **2023**, *102*, 102341. [CrossRef]
17. Zhou, C.; Wang, C.; Sun, D.; Hu, J.; Ye, H. An automated lightweight approach for detecting dead fish in a recirculating aquaculture system. *Aquaculture* **2025**, *594*, 741433. [CrossRef]
18. López-Barajas, S.; Sanz, P.J.; Marín-Prades, R.; Gómez-Espinosa, A.; González-García, J.; Echagüe, J. Inspection operations and hole detection in fish net cages through a hybrid underwater intervention system using deep learning techniques. *J. Mar. Sci. Eng.* **2024**, *12*, 80. [CrossRef]
19. Ren, S.; He, K.; Girshick, R.; Sun, J. Faster R-CNN: Towards real-time object detection with region proposal networks. *IEEE Trans. Pattern Anal. Mach. Intell.* **2017**, *39*, 1137–1149. [CrossRef] [PubMed]
20. Girshick, R.B.; Donahue, J.; Darrell, T.; Malik, J. Rich feature hierarchies for accurate object detection and semantic segmentation. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Columbus, OH, USA, 23–28 June 2014; pp. 580–587.
21. Redmon, J.; Divvala, S.; Girshick, R.B.; Farhadi, A. You only look once: Unified, real-time object detection. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 27–30 June 2016; pp. 779–788.
22. Jung, J.K. Making Faster R-CNN Faster! Available online: https://jkjung-avt.github.io/making-frcn-faster/ (accessed on 15 April 2026).
23. Tasnim, S.; Wang, Q. Progress in object detection: An in-depth analysis of methods and use cases. *Eur. J. Electr. Eng. Comput. Sci.* **2023**, *7*, 39–45. [CrossRef]
24. Redmon, J.; Farhadi, A. YOLOv3: An incremental improvement. *arXiv* **2018**, arXiv:1804.02767.
25. Jocher, G. *Ultralytics YOLOv5*; Ultralytics: Frederick, MD, USA, 2020. Available online: https://docs.ultralytics.com/models/yolov5 (accessed on 15 April 2026).
26. Jocher, G.; Qiu, J.; Chaurasia, A. *Ultralytics YOLOv8*; Ultralytics: Frederick, MD, USA, 2023. Available online: https://docs.ultralytics.com/models/yolov8 (accessed on 15 April 2026).
27. Zhang, P.; Zheng, J.; Gao, L.; Li, P.; Long, H.; Liu, H.; Li, D. A novel detection model and platform for dead juvenile fish from the perspective of multi-task. *Multimed. Tools Appl.* **2024**, *83*, 24961–24981. [CrossRef]
28. Zhang, H.; Tian, Z.; Liu, L.; Liang, H.; Feng, J.; Zeng, L. Real-time detection of dead fish for unmanned aquaculture by YOLOv8-based UAV. *Aquaculture* **2025**, *595*, 741551. [CrossRef]
29. Ranjan, R. YOLO in precision aquaculture: A decadal bibliometric and systematic review of applications, architectural adaptations, and deployment challenges. *J. Agric. Food Res.* **2026**, *28*, 102982. [CrossRef]
30. Mohd Rahman, M.I.; Abdul Razak, A.F.; Abdul Majeed, A.P.P.; Musa, R.M.; Abdul Jalil, A.A.; Mohd Khairuddin, I.; Abdullah, M.A.; Mohd Razman, M.A. Analyzing fish detection and classification in IoT-based aquatic ecosystems through deep learning. *PeerJ Comput. Sci.* **2026**, *12*, e3496. [CrossRef]
31. Veerappan, S.; Arvinth, N. IoT-enabled communication network for real-time disease alerts in smart aquaculture systems. *J. Intell. Assist. Commun. Technol.* **2025**, *1*, 1–7.
32. Jocher, G.; Qiu, J.; Liu, M.; Lyu, S.; Akyon, F.C.; Kalfaoglu, M.E. Ultralytics YOLO26: Unified Real-Time End-to-End Vision Models 2026. *arXiv* **2026**, arXiv:2606.03748.
33. Jocher, G.; Qiu, J.; Jocher, G.; Qiu, J. *Ultralytics YOLO11*; Ultralytics: Frederick, MD, USA, 2024. Available online: https://docs.ultralytics.com/models/yolo11 (accessed on 15 April 2026).

34. Sapkota, R.; Karkee, M. Ultralytics YOLO evolution: An overview of YOLO26, YOLO11, YOLOv8 and YOLOv5 object detectors for computer vision and pattern recognition. *arXiv* **2025**, arXiv:2510.09653.
35. Sapkota, R.; Cheppally, R.H.; Sharda, A.; Karkee, M. YOLO26: Key architectural enhancements and performance benchmarking for real-time object detection. *arXiv* **2025**, arXiv:2509.25164.
36. Khanam, R.; Hussain, M. YOLOv11: An overview of the key architectural enhancements. *arXiv* **2024**, arXiv:2410.17725.
37. Wang, A.; Chen, H.; Liu, L.; Chen, K.; Lin, Z.; Han, J.; Ding, G. YOLOv10: Real-time end-to-end object detection. *Adv. Neural Inf. Process. Syst.* **2024**, *37*, 107984–108011. [CrossRef]
38. Dalal, S.; Lilhore, U.K.; Sharma, N.; Arora, S.; Simaiya, S.; Ayadi, M.; Almujally, N.A.; Ksibi, A. Improving smart home surveillance through YOLO model with transfer learning and quantization for enhanced accuracy and efficiency. *PeerJ Comput. Sci.* **2024**, *10*, e1939. [CrossRef]
39. Chakrabarty, S. YOLO26: An analysis of NMS-free end to end framework for real-time object detection. *arXiv* **2026**, arXiv:2601.12882.
40. Ashfaq, S.; AskariHemmat, M.; Sah, S.; Saboori, E.; Mastropietro, O.; Hoffman, A. Accelerating deep learning model inference on Arm CPUs with ultra-low bit quantization and runtime. *arXiv* **2022**, arXiv:2207.08820.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.