Vol. 8, No. 3, June 2026

ISI Journal of Information Systems and Informatics

Published By Asosiasi Doktor Sistem Informasi Indonesia

Multi-Seed Robustness Benchmark of Lightweight YOLO Models for Young Crescent Moon Detection under Limited-Data Conditions

Bayu Krisna Murti[1], Kartika Firdausy[2], Murinto[3]

[1]Master Program of Informatics, Universitas Ahmad Dahlan, Yogyakarta, Indonesia

[2]Department of Electrical Engineering, Universitas Ahmad Dahlan, Yogyakarta, Indonesia

[3]Department of Informatics, Universitas Ahmad Dahlan, Yogyakarta, Indonesia

Received:
November 10, 2025
Revised:
May 17, 2026
Accepted:
May 30, 2026
Published:
June 24, 2026

Corresponding Author:
Author Name*:
Bayu Krisna Murti
Email*:
2408048018@webmail.uad.ac.id

DOI:
10.63158/journalisi.v8i3.1653

Abstract. Visual observation of the young crescent moon is challenging due to its thin and low-contrast appearance. Although YOLO-based object detectors are promising for image-based crescent localization, newer architectures do not automatically generalize well to small grayscale datasets, and prior studies rarely report robustness across repeated training runs. This study benchmarks YOLOv8n, YOLO11n, and YOLO26n for young crescent moon detection under limited-data conditions. A grayscale dataset of 697 images was resized to 640 × 640 pixels, annotated with the single class crescent_moon, and split into training, validation, and test subsets at a fixed 70:20:10 ratio. The three models were trained using the same configuration across five random seeds. Validation results were used to analyze multi-seed robustness, while the fixed 71-image test set was used for CPU-only inference evaluation. YOLO26n achieved the highest validation mAP@50-95 and fitness with the lowest variability, and also achieved the lowest CPU pipeline latency and highest throughput on the test set. These findings show that YOLO26n offers the best trade-off between accuracy and efficiency across the evaluated dataset and CPU-only inference setting. The reported throughput reflects low-frame-rate image-based inference, not real-time video performance. This study provides a reproducible benchmark protocol that combines fixed data splitting, grayscale preprocessing, data integrity checking, multi-seed robustness analysis, and CPU inference profiling.

Keywords: young crescent moon detection; lightweight object detection; YOLO benchmark; multi-seed reproducibility; CPU inference

ISI Journal of Information Systems and Informatics
Published By Asosiasi Doktor Sistem Informasi Indonesia

1. INTRODUCTION

The determination of the beginning of the Hijri month still relies on observing the young crescent moon, known as hilal, shortly after sunset. This process is important for determining the dates of Ramadhan, Syawal, and Dzulhijjah. However, visual observation remains technically challenging because the crescent often appears thin, dim, and low in contrast against the sky background [1]. Atmospheric conditions, twilight scattering, instrument quality, and the limitations of visual observation can also affect detection consistency [1], [2]. In Indonesia and Southeast Asia, the revised MABIMS criteria, which require a minimum crescent altitude of 3° and elongation of 6.4° [3], [4], further emphasize the need for a measurable, consistent, and reproducible vision-based decision-support tool for hilal observation. Such a tool is intended to support visual crescent localization from images or video frames and not to replace official religious or governmental decisions regarding the beginning of the Hijri month.

In this context, crescent visibility prediction and image-based crescent localization are distinct tasks. Crescent visibility prediction estimates the likelihood of the young crescent moon being visible based on astronomical or observational variables, such as altitude, elongation, age, or the arc of vision. In contrast, image-based crescent localization focuses on detecting and bounding the crescent object in an image or video frame. This study addresses the second task and positions the proposed method as a visual decision-support tool for crescent observation, not as an authority for determining the official beginning of the Hijri month.

Recent advances in deep learning, particularly in object detection, provide an opportunity to support vision-based observation of young crescent moons. Unlike image classification, object detection identifies the presence of an object and localizes it using a bounding box. This capability is relevant to young crescent moon detection because crescents are usually small, thin, low-contrast, and not always centered in the frame. Among object detection models, the YOLO family is widely used because it can locate objects in images or video frames with fast inference [5]. The Ultralytics YOLO ecosystem has continued to develop from YOLOv8 to YOLO11 and YOLO26, introducing architectural refinements such as anchor-free detection, C3k2 modules, and C2PSA attention [6], [7]. YOLO26n was included because it represents a newer lightweight YOLO variant in the

Ultralytics ecosystem with a more deployment-oriented design, making it relevant for evaluating whether a compact architecture can improve robustness and CPU inference efficiency in limited-data young crescent moon detection. However, YOLO26n is not treated as a universally validated standard architecture or as an automatically superior model. Its performance is therefore evaluated empirically under the same dataset split, training configuration, multi-seed evaluation protocol, and CPU-only inference setting as YOLOv8n and YOLO11n [8], [9]. Therefore, YOLO-based young crescent moon detection requires empirical evaluation under controlled experimental settings.

Previous studies on crescent-related computation can be grouped into visibility prediction and image-based detection. Visibility prediction studies commonly use astronomical and observational features, such as lunar altitude, elongation, and arc of vision, to estimate whether the crescent is likely to be visible [10], [11], [12]. Image-based detection studies process visual data directly to locate or identify the crescent in images or video frames, including contrast-enhancement preprocessing [1], [13], Haar-cascade and Support Vector Machine classifiers [14], video-based computer vision pipelines [15], and Mask R-CNN trained on robotic-telescope datasets [2]. However, existing studies remain limited in three aspects. First, no prior study has compared YOLOv8n, YOLO11n, and YOLO26n on the same benchmark for grayscale young crescent moon images [5]. Second, many evaluations still rely on single-run results, although deep learning performance can vary measurably due to random initialization, data sampling, and stochastic optimization [16]. Third, data integrity checking and CPU inference analysis are not always reported, even though both are important for evaluating model reliability and lightweight deployment feasibility [17].

Evaluation under limited-data conditions requires attention to robustness and data integrity. A model trained on a small dataset can be sensitive to random seed variation, augmentation sampling, and data partitioning [16], [18]. In addition, data leakage across training, validation, and test splits can inflate performance and reduce the reliability of evaluation results [17]. This issue arises when images are collected from multiple sources, leading to duplicate or visually similar samples appearing across partitions. Therefore, a reliable benchmark should control the data split, evaluate performance across multiple random seeds, and inspect possible explicit overlap across dataset partitions.

Based on these gaps, this study presents a controlled benchmark of YOLOv8n, YOLO11n, and YOLO26n for image-based young crescent moon detection using a limited grayscale dataset of 697 images. All images were converted to grayscale, resized to 640 × 640 pixels, manually annotated with one class, namely crescent_moon, and divided into fixed training, validation, and test subsets with approximately 70:20:10 ratios. The three nano-scale YOLO models were trained with the same configuration: 200 epochs, batch size 16, AdamW optimizer, an initial learning rate of 0.002, pretrained weights, and five random seeds: 42, 123, 7, 0, and 99. The validation set was used to evaluate multi-seed robustness, while the fixed test set was used for CPU inference and accuracy-efficiency analysis.

The main contribution of this study is a reproducible benchmark for limited-data image-based young crescent moon detection. Specifically, this study provides four measurable contributions: (1) a controlled grayscale dataset protocol with fixed training, validation, and test splits; (2) a multi-seed robustness evaluation of YOLOv8n, YOLO11n, and YOLO26n under the same training configuration; (3) explicit data integrity checking to reduce the risk of duplicate samples across splits; and (4) CPU inference profiling to assess lightweight deployment feasibility.

This study addresses three research questions: (1) how do YOLOv8n, YOLO11n, and YOLO26n perform in detecting young crescent moon objects on a limited grayscale dataset; (2) how stable is each architecture across different random seeds on the validation set; and (3) which model provides the best trade-off between detection performance and CPU inference efficiency on the fixed test set. This study is limited to single-class image-based crescent localization and does not determine the official beginning of the Hijri month.

2. METHODS

2.1. Research Workflow

This study followed an experimental workflow consisting of four stages: data preparation, experimental design, training and evaluation, and analysis and discussion. This workflow was used to ensure a controlled, consistent, and reproducible YOLO benchmark, as shown in Figure 1.

ISI Journal of Information Systems and Informatics
Published By Asosiasi Doktor Sistem Informasi Indonesia

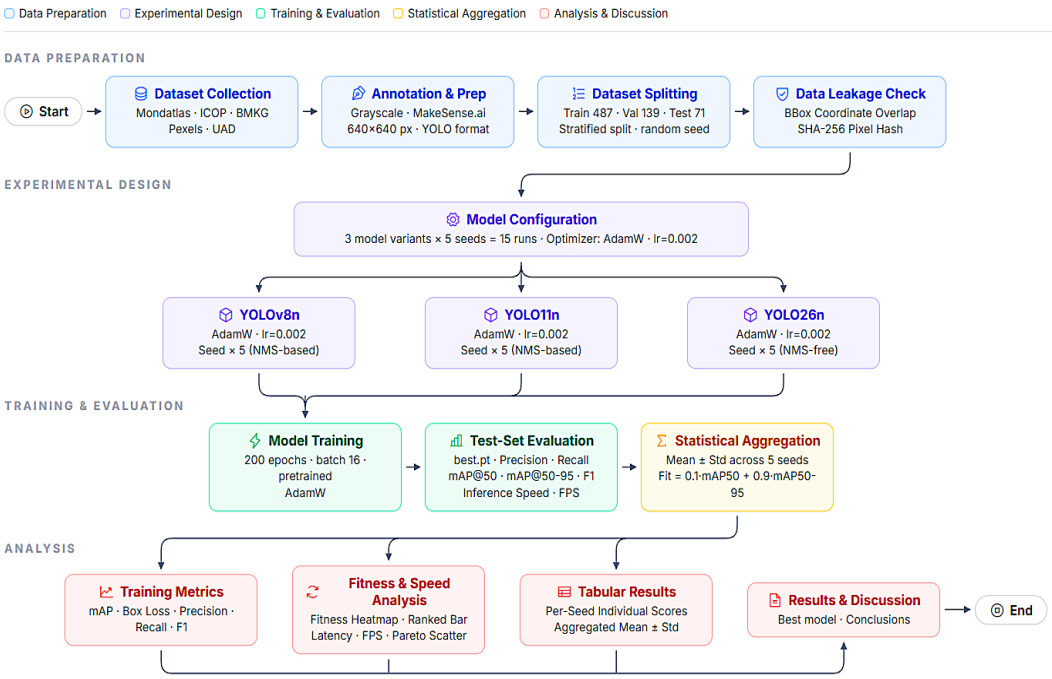


Figure 1. Research workflow of the proposed YOLO-based young crescent moon detection benchmark

The data preparation stage included dataset collection, manual annotation, preprocessing, data splitting, and data integrity checking. The experimental design stage defined the comparison of YOLOv8n, YOLO11n, and YOLO26n under the same training configuration. The training and evaluation stage measured object detection performance and CPU inference latency. The final stage aggregated the results statistically to analyze model performance, robustness, and the accuracy-efficiency trade-off.

2.2. Dataset And Data Preparation

2.2.1. Data Preparation

The dataset used in this study was compiled from five sources: Pexels, BMKG, Mondatlas, ICOP, and UAD Observatory, yielding 697 young crescent moon images under varying visual conditions. Mondatlas, ICOP, and BMKG were included to capture public crescent observation conditions, while Pexels was used to increase visual diversity after manual filtering to retain only authentic crescent moon configurations [19], [20], [21], [22]. UAD Observatory images were included as institutional observation data to strengthen the representation of local observation conditions. The distribution of dataset sources is shown in Table 1.

Table 1. Dataset source distribution

Source	Number of images	Percentage	Notes
Pexels	167	24.0%	Public image platform, manually filtered
BMKG	65	9.3%	Institutional/public hilal observation source
Mondatlas	203	29.1%	Public crescent observation archive
ICOP	198	28.4%	Public crescent observation archive
UAD Observatory	64	9.2%	Institutional observation collection
Total	**697**	**100.0%**	Curated grayscale young crescent moon dataset

Images from public repositories were used for academic research and cited in accordance with the availability of their sources. Images from Pexels were manually filtered and used in accordance with the platform's public license terms [23]. Images from the UAD Observatory were used as institutional observation data for research purposes. This study reports aggregate experimental results and does not redistribute the full image dataset.

Before annotation and model training, all images were converted into grayscale because crescent moon detection primarily depends on luminance contrast against the twilight sky, where color information provides minimal discriminative value. All images were resized to 640 × 640 pixels with a 1:1 aspect ratio to match the YOLO input configuration. This standardization ensured that all models received input data with the same spatial resolution, enabling consistent training and evaluation across YOLOv8n, YOLO11n, and YOLO26n.

2.2.2 Annotation Process

After preprocessing, all images were manually annotated by the researcher using MakeSense.ai [24]. The annotation was performed using a single object class, namely crescent_moon (nc = 1). Bounding boxes were manually drawn to mark the visible location of the young crescent moon in each image. The annotation results were then exported in YOLO format, containing the class index and normalized bounding box coordinates.
After export, the annotation files were manually inspected to ensure that each visible crescent was correctly labeled and that no missing or incorrect class labels were present.

All bounding boxes were annotated and reviewed by a single researcher. No inter-annotator agreement was calculated. This is acknowledged as a limitation because bounding-box placement for thin and low-contrast crescent objects may involve subjective judgment, especially when the crescent boundary is faint or partially visible.

The current dataset contains positive young crescent moon images with one annotated class, namely crescent_moon. In YOLO-based object detection, the model learns to discriminate between objects and background using annotated bounding boxes and unannotated regions during training. Recent YOLO-based studies also show that positive and negative sample assignment, as well as object-background information, are important for improving detection learning and reducing confusion with background-like patterns [25], [26]. However, this dataset does not include fully negative images without crescent objects, such as clear sky, cloud edges, twilight gradients, or other non-crescent visual patterns. Negative-sky images were not included in this version because the dataset was constructed from confirmed young crescent moon observations from available crescent-positive sources, while non-crescent sky scenes were not systematically collected or verified during the initial dataset development. Therefore, this study evaluates single-class crescent localization on positive images, while false-positive behavior on fully negative sky images remains limited. Future dataset development should include negative images with empty label files or false-positive examples to improve false-positive evaluation and robustness against non-crescent background patterns. An example of the annotation process is shown in Figure 2.

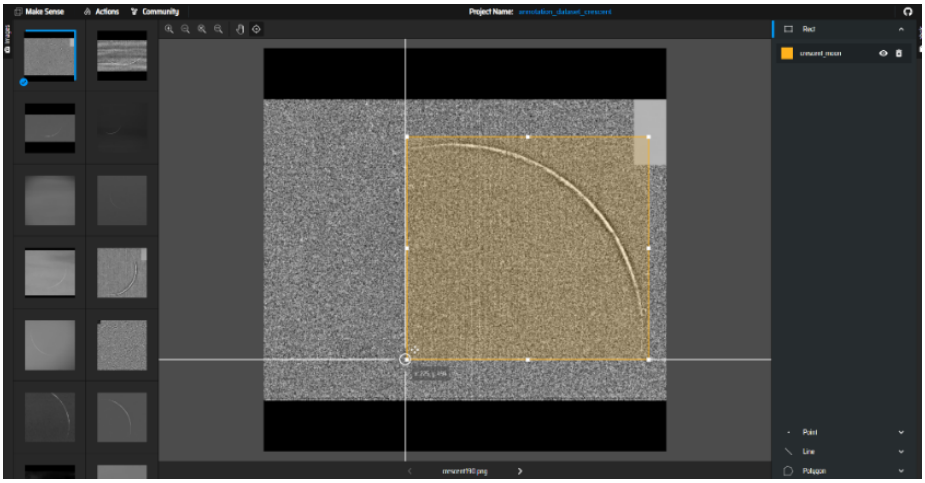


Figure 2. Example of crescent moon annotation using MakeSense.ai

As shown in Figure 2, each visible young crescent moon object was assigned a bounding box and the class label crescent_moon. These manually annotated bounding boxes were used as ground-truth labels during model training and evaluation.

2.2.3 Dataset Splitting

After all images from the five sources were merged, preprocessed, and annotated, the dataset was divided into training, validation, and test subsets using a fixed random split with seed 42 at approximately 70:20:10. The resulting split was saved as a fixed dataset directory in Google Drive and reused in Google Colab across all architectures and training seeds. This strategy ensured that each model was trained, validated, and tested on the same data partitions, allowing a fair comparison across models and training runs. However, the split was not stratified by image source or observation session. Therefore, images from the same source, or visually similar images from similar observation contexts, may still appear across different subsets.

2.2.4 Data Integrity Check

Before model training, a data integrity check was conducted to reduce the risk of explicit data leakage across the training, validation, and test splits [17]. The check was performed by constructing overlap matrices based on two representations: normalized bounding box coordinates and SHA-256 pixel-level image hashes [27]. The bounding box coordinate check was used to identify identical annotation patterns across different splits, while the SHA-256 hash check was used to detect pixel-identical images after preprocessing. This procedure was intended to detect explicit duplicate samples across splits. It does not claim to identify all possible near-duplicate images or visually similar crescent patterns.

2.3. Experimental Design

2.3.1 Model Architecture Overview

This study compares three nano-scale YOLO architectures from the Ultralytics framework [5]: YOLOv8n, YOLO11n [6], and YOLO26n [28], [29]. These models were selected because they represent different generations of lightweight YOLO detectors and share the same Ultralytics training and evaluation pipeline. YOLO26n was included as a newer lightweight YOLO variant with a more deployment-oriented design than earlier generations. In this study, YOLO26n is not treated as automatically superior, but as an architecture that requires empirical validation under the same dataset split, training

configuration, augmentation strategy, and multi-seed evaluation protocol as YOLOv8n and YOLO11n. This consistency allows the comparison to focus on architectural differences, multi-seed robustness, and CPU inference efficiency rather than differences caused by separate implementation frameworks. The computational complexity of each model was examined using the Ultralytics model summary generated in the same experimental environment. The summary includes the number of layers, parameters, and GFLOPs, as shown in Table 2.

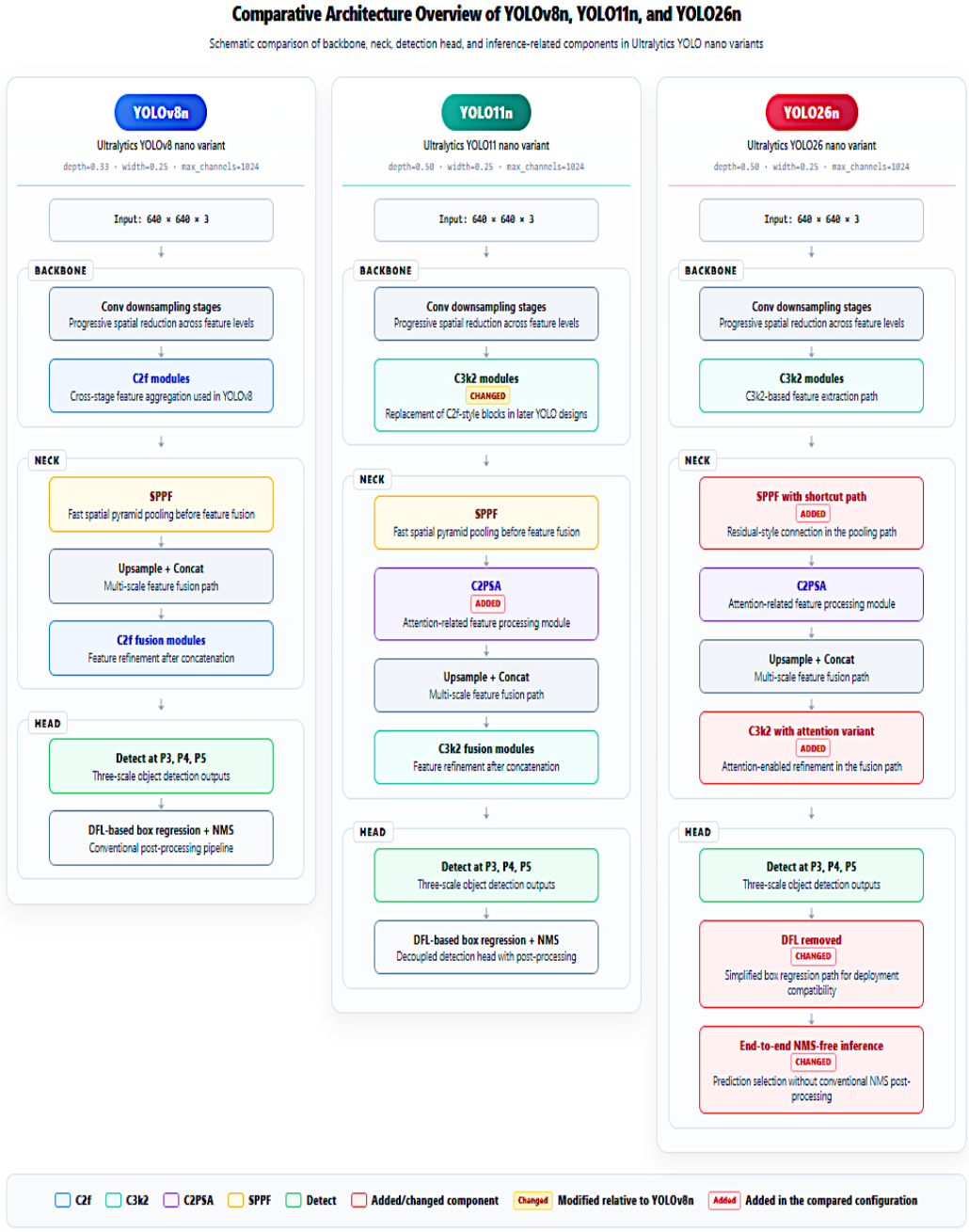
Table 2. Model complexity summary from Ultralytics model inspection

Model	Layers	Parameters	GFLOPs
YOLOv8n	129	3,157,200	8.9
YOLO11n	181	2,624,080	6.6
YOLO26n	260	2,572,280	6.1

As shown in Table 2, the three architectures differ substantially in depth, parameter count, and computational cost. YOLOv8n has the fewest layers (129) but the highest parameter count (3.16M) and GFLOPs (8.9). YOLO11n increases depth to 181 layers while reducing parameters to 2.62M and GFLOPs to 6.6. YOLO26n has the deepest structure (260 layers) yet the lowest parameter count (2.57M) and computational cost (6.1 GFLOPs). This profile supports its inclusion as a candidate lightweight detector for CPU-oriented young crescent moon detection. However, its lower computational profile does not automatically imply better detection performance, so its effectiveness must be verified through validation robustness, test-set performance, and CPU inference analysis. This pattern indicates that model depth does not necessarily translate into higher computational cost—a property relevant to lightweight young crescent moon detection. These differences suggest that YOLO26n may offer better inference efficiency. Still, this assumption requires empirical validation, with final model selection based on test-set performance, multi-seed robustness, and CPU inference latency. The architectural components of the three models are further illustrated in Figure 3.

As shown in Figure 3(a), YOLOv8n uses C2f-based modules for feature extraction and feature fusion, SPPF for spatial feature aggregation, and a multi-scale detection head at P3, P4, and P5 with DFL-based box regression and NMS post-processing [30]. Figure 3(b) shows that YOLO11n introduces C3k2 modules and C2PSA to improve feature processing

in newer YOLO designs, while still using multi-scale detection outputs and NMS-based post-processing [6], [30]. Meanwhile, Figure 3(c) shows that YOLO26n adopts a more deployment-oriented design, including SPPF with a shortcut path, C2PSA, an attention-based C3k2 variant, simplified box regression, and end-to-end NMS-free inference [28], [29].



(a) (b) (c)

Figure 3. Comparative architecture overview of (a) YOLOv8n, (b) YOLO11n, and (c) YOLO26n

The comparison in Table 2 and Figure 3 shows that the three models differ not only in version but also in structural design and computational profile. These differences provide a basis for evaluating whether a newer, more compact YOLO architecture can improve detection robustness and CPU inference efficiency on a limited grayscale Young crescent moon dataset. To ensure a fair comparison, all models were trained using the same dataset split, training configuration, augmentation strategy, and multi-seed evaluation protocol.

2.3.2 Training Configuration And Data Augmentation

All models were trained using the same experimental configuration to ensure a fair comparison across architectures. The experiments were conducted on YOLOv8n, YOLO11n, and YOLO26n using a standardized input size of 640 × 640 pixels and the same training protocol across five random seeds. Pretrained nano-scale YOLO weights were used because the dataset is relatively small and contains only one object class. This setting allowed the models to use general visual features learned from larger datasets while adapting to the specific characteristics of young crescent moon images.

Model training was performed on Google Colab using an NVIDIA Tesla T4 GPU. At the same time, CPU inference latency was measured separately on an Intel(R) Xeon(R) CPU @ 2.20 GHz with 2 cores using FP32 precision and without GPU acceleration. This separation was used to evaluate GPU-accelerated training while assessing lightweight deployment performance in a CPU-only inference scenario. The complete training configuration, software environment, and model weight sources are summarized in Table 3. The model weights and implementation pipeline were based on the official Ultralytics YOLO documentation [31].

Table 3. Training configuration, software environment, and model weight sources

Parameter	Value
Model	YOLOv8n, YOLO11n, YOLO26n
Model weights	yolov8n.pt, yolo11n.pt, yolo26n.pt
Number of classes	1
Class name	crescent_moon
Image size	640 × 640
Epochs	200

Parameter	Value
Batch size	16
Optimizer	AdamW
Initial learning rate	0.002
Pretrained weights	Yes
Random seeds	42, 123, 7, 0, 99
Total runs	15
Ultralytics version	8.4.48
Python version	3.12.13
PyTorch version	2.10.0+cu128
CUDA version	12.8
Training platform	Google Colab
Training device	NVIDIA Tesla T4 GPU
Evaluation checkpoint	best.pt for each seed
Inference precision	FP32
CPU inference device	Intel Xeon CPU @ 2.20 GHz, 2 cores

Data augmentation was applied during training using the built-in Ultralytics YOLO augmentation pipeline. The augmentation settings were adjusted to the characteristics of grayscale, low-contrast, and single-class young crescent moon images. Hue and saturation augmentation were disabled because color information was not used. Brightness variation, small geometric transformations, flipping, and mosaic augmentation were retained to increase visual diversity and reduce overfitting under limited-data conditions. Recent object detection studies have shown that augmentation improves robustness against variations in brightness, scale, object position, and orientation, especially in domain-specific datasets with small objects or limited visual samples [32], [33], [34]. Flipping was used as a robustness-oriented augmentation because the dataset was collected from heterogeneous sources and imaging setups, where apparent crescent orientation may vary due to camera rotation, optical configuration, mirroring, or post-capture formatting. However, this augmentation was applied only to improve image-based object localization, not to support astronomical interpretation of crescent orientation. Since this study does not estimate crescent visibility parameters, lunar geometry, or the official beginning of the Hijri month, vertical and horizontal flips were used to improve visual robustness while keeping the task limited to bounding-box

detection. Mosaic augmentation was retained because it combines multiple images into one training sample and exposes the detector to more varied object scales, backgrounds, and spatial contexts, as also used in recent YOLO-based object detection studies [33], [34]. The mosaic strategy was stopped during the last 20 epochs using close_mosaic = 20, allowing the models to adapt back to the original image distribution before training ended. The augmentation settings are summarized in Table 4.

Table 4. Data augmentation settings

Parameter	Value
hsv_h	0.0
hsv_s	0.0
hsv_v	0.6
degrees	10.0
translate	0.15
scale	0.5
flipud	0.6
fliplr	0.6
mosaic	0.8
close_mosaic	20
mixup	0.0

2.4. Training and Evaluation

2.4.1 Model Training

Model training was performed according to the configurations defined in Tables 3 and 4. Each architecture was trained 5 times with different random seeds (42, 123, 7, 0, and 99), resulting in a total of 15 training runs. The dataset split was fixed before training and reused across all models and seeds. This design ensured that the observed performance variation mainly reflected training stochasticity rather than changes in data partitioning.

To improve reproducibility, each random seed was applied to Python's random module, NumPy, PyTorch, and CUDA. The CuDNN deterministic mode was also enabled to reduce nondeterministic behavior during GPU computation. This setting is important because deep learning experiments can still exhibit variability across repeated runs due to random

initialization, stochastic optimization, data augmentation, and nondeterministic GPU operations [35], [36].

2.4.2 Test Set Evaluation

After training, each run was first evaluated on the fixed validation set to analyze multi-seed robustness. The validation metrics were aggregated using mean ± standard deviation across five random seeds. The fixed test set, consisting of 71 images, was then used for the final CPU inference and accuracy-efficiency evaluation. For each seed, both validation and test evaluations were performed using the best.pt checkpoint selected by the Ultralytics training process based on the best validation performance, rather than the last-epoch checkpoint. This ensured that all runs followed the same checkpoint selection rule before final evaluation.

The test set was not used during training or checkpoint selection, allowing the final evaluation to measure model performance on unseen data. During CPU inference evaluation, the confidence threshold was set to 0.25, and the IoU threshold for NMS-based post-processing was set to 0.45. For YOLOv8n and YOLO11n, NMS-based inference used max_det=1 because the evaluated task focuses on localizing a single young crescent moon candidate in each observation image. For YOLO26n, the NMS-free inference output was used without applying max_det, and the highest-confidence detection was selected from the model output. Model performance was evaluated using precision, recall, F1-score, mAP@50, mAP@50-95, fitness, and CPU inference latency. Precision, recall, F1-score, IoU-based mAP, and COCO-style mAP@50-95 are commonly used in object detection evaluation to measure detection correctness and localization quality [37], [38].

Precision measures the proportion of correct detections among all predicted detections, as shown in Equation (1).

$$Precision = \frac{TP}{TP + FP} \quad (1)$$

In Equation (1), TP denotes true positive detections, while FP denotes false positive detections. A higher precision value indicates that the model produces fewer incorrect detections.

Recall measures the proportion of correctly detected objects among all ground-truth objects, as shown in Equation (2).

$$Recall = \frac{TP}{TP + FN} \tag{2}$$

In Equation (2), FN denotes false negative detections. A higher recall value indicates that the model detects more ground-truth young crescent moon objects.

The F1-score combines precision and recall into a single harmonic mean, as shown in Equation (3).

$$F1\text{-}score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{3}$$

Equation (3) is used to evaluate the balance between precision and recall. A higher F1-score indicates that the model maintains a better balance between detecting correct objects and reducing false detections.

The mean Average Precision at IoU threshold 0.50, denoted as mAP@50, measures average precision when a prediction is considered correct if its Intersection over Union with the ground truth is at least 0.50. Meanwhile, mAP@50-95 averages the average precision across IoU thresholds from 0.50 to 0.95 with a step of 0.05, as shown in Equation (4).

$$mAP@50\text{-}95 = \frac{1}{10} \sum_{t \epsilon \{0.50, 0.55, \ldots, 0.95\}} AP_t \tag{4}$$

In Equation (4), AP_t denotes the average precision at IoU threshold t. The mAP@50-95 metric provides a stricter evaluation by considering multiple localization thresholds.

The fitness score was used as a summary indicator for model selection. In the Ultralytics detection setting, fitness gives greater weight to mAP@50-95 than mAP@50, as shown in Equation (5) [39], [40].

$$Fitness = 0.1 \times mAP@50 + 0.9 \times mAP@50\text{-}95 \quad (5)$$

Equation (5) shows that mAP@50-95 contributes more strongly to the final fitness value. Therefore, a higher fitness score indicates better overall detection performance, especially in terms of localization quality.

CPU inference latency was measured to assess the feasibility of lightweight deployment. Inference was evaluated using FP32 precision on an Intel(R) Xeon(R) CPU @ 2.20 GHz with 2 cores in Google Colab, without GPU acceleration. Latency was calculated as the average processing time per image, as shown in Equation (6).

$$Latency = \frac{T_{total}}{N} \quad (6)$$

In Equation (6), T_{total} denotes the total inference time, while N denotes the number of evaluated images. The latency value is reported in milliseconds per image. A lower latency value indicates faster inference.

2.4.3 Statistical Aggregation

The results from the five random seeds were aggregated for each architecture using mean ± standard deviation. The mean value represents the average performance across repeated runs, as shown in Equation (7).

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \quad (7)$$

In Equation (7), x_i denotes the metric value obtained from the i-th seed, while n denotes the number of runs. In this study, $n = 5$.

The standard deviation was used to measure performance variability across random seeds, as shown in Equation (8). A lower standard deviation indicates more stable performance across repeated training runs.

$$SD = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2} \quad (8)$$

In Equation (8), SD denotes the standard deviation, x_i denotes the metric value from each seed, $\bar{x}$ denotes the mean value, and n denotes the number of runs. A lower standard deviation indicates more stable performance across repeated training runs. The final comparison considered not only the average detection performance, but also the stability and CPU inference efficiency of each model.

2.5. Analysis Procedure

The analysis combined training behavior, detection performance, stability, and inference efficiency. Training curves were examined to observe the convergence behavior of each model based on loss, precision, recall, mAP, and fitness. The validation results were summarized in a table, with mean ± standard deviation across five random seeds to assess robustness. The fixed test set was used separately to evaluate CPU inference latency and the final trade-off between accuracy and efficiency. The fitness score was used as the main summary indicator because it gives greater weight to mAP@50-95, which reflects stricter localization quality. However, the final interpretation did not rely only on fitness. Precision, recall, F1-score, mAP@50, mAP@50-95, stability across seeds, and CPU inference latency were also considered. The most suitable model was determined by analyzing the trade-off between detection accuracy, robustness, and inference efficiency under the evaluated setting.

3. RESULTS AND DISCUSSION

3.1. Dataset Split and Integrity Analysis

This section presents the dataset characteristics after preprocessing, the final split distribution, and the data integrity check before model evaluation. The dataset consisted

of 697 young crescent moon images collected from multiple sources. All images were converted to grayscale and standardized to a 640 × 640, 1:1 aspect-ratio format before annotation and model training.

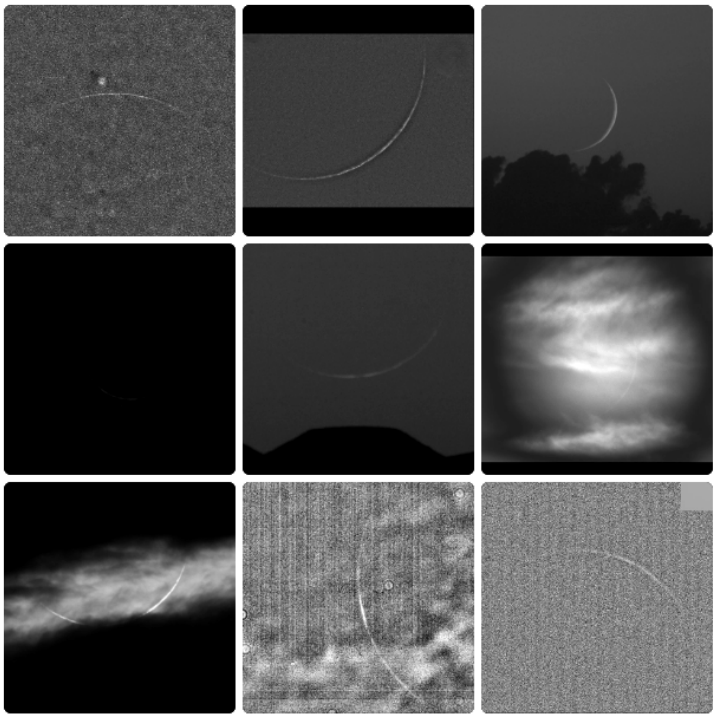


Figure 4. Preview of the preprocessed young crescent moon dataset in grayscale and standardized 1:1 format

Figure 4 shows that the preprocessed dataset contains young crescent moon images with diverse visual conditions, including variations in crescent thickness, brightness, contrast, and background appearance. These variations make young crescent moon detection challenging because the crescent is often thin and small, making it difficult to distinguish from the sky. The grayscale representation emphasizes intensity and contrast patterns, which are more relevant to crescent moon detection than color information.

The dataset was then divided into training, validation, and test subsets using a fixed split. This split was defined before model training and reused across all architectures and random seeds to ensure that model comparisons were not affected by differences in data partitioning.

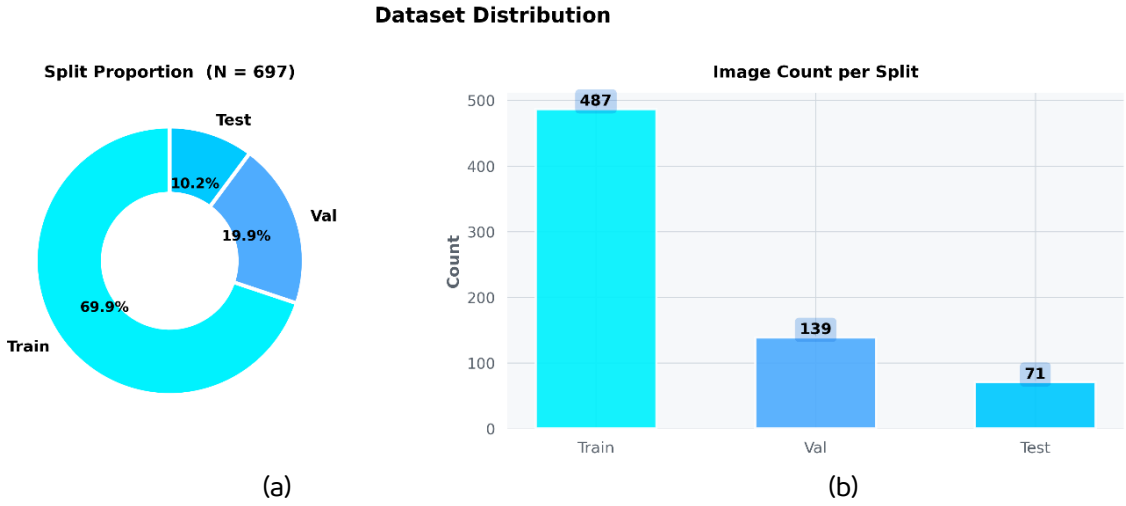


Figure 5. Dataset distribution: (a) split proportion and (b) image count per split

Figure 5(a) shows that the dataset was split at approximately 70:20:10, with 69.9% for training, 19.9% for validation, and 10.2% for testing. Figure 5(b) shows the number of images allocated for each purpose: 487 for training, 139 for validation, and 71 for testing. The training set optimized model parameters, the validation set monitored model performance during training, and the test set evaluated the final model.

The 71-image test set was used as a fixed held-out subset for controlled model comparison. Since the test images were drawn from the same curated dataset, they preserve part of the visual diversity shown in Figure 4, including variations in crescent thickness, brightness, contrast, and background appearance. However, the split was not explicitly stratified by image source, brightness level, crescent size, or background type. Therefore, the test set should be interpreted as a controlled held-out evaluation set rather than a complete representation of all possible young crescent moon observation conditions. Future work should apply stratified splitting based on source, observation session, brightness range, crescent size, and background category.

Before training began, a data integrity check was conducted to minimize the risk of data leakage across the splits. Two checks were performed: bounding box coordinate overlap and SHA-256 pixel-level hash overlap. The bounding box coordinate check detects identical annotation coordinates across different splits, and the SHA-256 pixel hash check detects pixel-identical images after preprocessing.

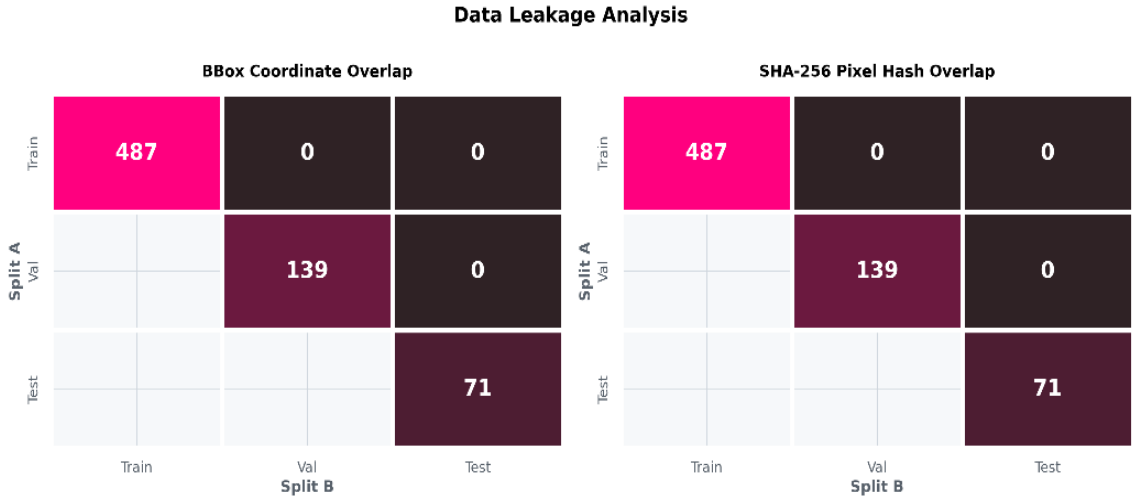


Figure 6. Data integrity : (a) bounding box coordinate overlap and (b) SHA-256 pixel hash overlap

Figure 6(a) shows zero off-diagonal values for bounding box coordinate overlap across all split pairs, including train-validation, train-test, and validation-test. Figure 6(b) also shows zero off-diagonal values for SHA-256 pixel hash overlap, indicating that no pixel-identical images were found across splits. The diagonal values indicate the number of samples per split: 487 for training, 139 for validation, and 71 for test.

These results indicate that no explicit overlap was detected based on identical bounding box coordinates or pixel-level hashes. Therefore, the dataset split reduces the risk of explicit data leakage and supports a fairer evaluation of YOLOv8n, YOLO11n, and YOLO26n. This check does not claim to eliminate all possible near-duplicate visual similarities, but it strengthens the reliability of the evaluation protocol.

3.2. Training Performance Analysis

Training performance was analyzed using the loss curves recorded over 200 epochs across five random seeds. The curves are presented as mean ± standard deviation to show both convergence behavior and variation across repeated runs. Since this study did not modify Ultralytics YOLO's internal loss function, the loss components were used as diagnostic indicators to assess training stability and potential overfitting.

ISI Journal of Information Systems and Informatics
Published By Asosiasi Doktor Sistem Informasi Indonesia

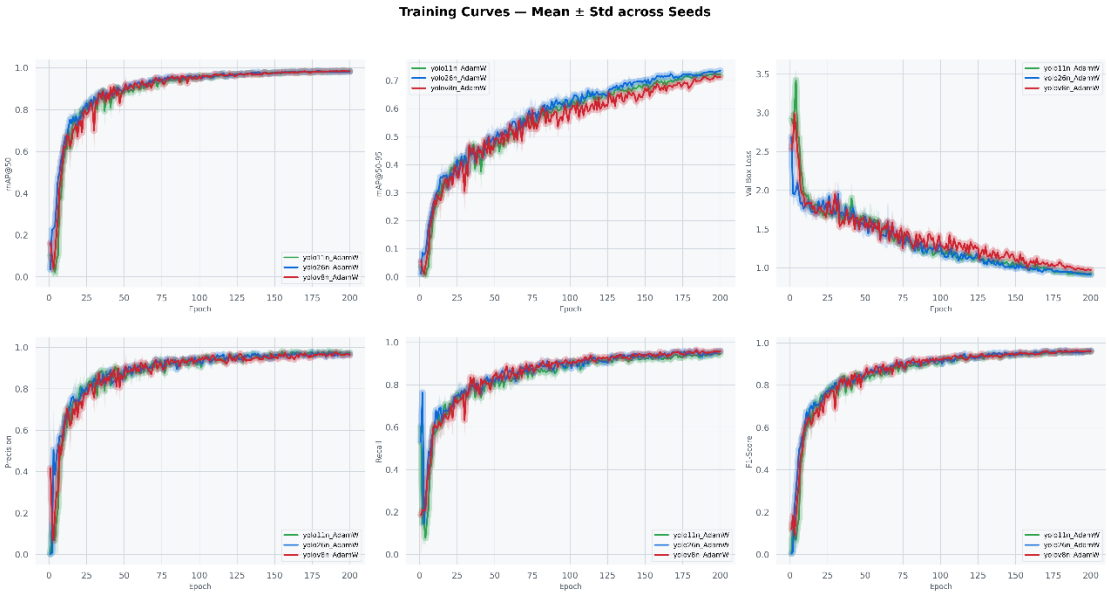


Figure 7. Training curves of YOLOv8n, YOLO11n, and YOLO26n across five random seeds

As shown in Figure 7, all three models exhibit consistent improvements in detection metrics during training. The mAP@50 curve increases rapidly during the early epochs. It approaches a high value near the end of training, indicating that the models quickly learned to localize young crescent moon objects under the standard IoU threshold. The mAP@50-95 curve increases more gradually, as expected, because this metric evaluates localization quality at stricter IoU thresholds. This pattern suggests that coarse detection ability was learned earlier, while more precise localization continued to improve throughout training.

The precision, recall, and F1-score curves also increase steadily and become more stable toward the final epochs. The shaded regions, which represent variation across seeds, are wider in the early training phase and become narrower as training progresses. This indicates that the models were more sensitive to random initialization and augmentation sampling at the beginning of training, but became more stable after learning the main crescent features. Among the three models, YOLO26n and YOLO11n show slightly stronger mAP@50-95 progression in the later epochs, while YOLOv8n remains competitive in mAP@50, precision, recall, and F1-score.

The validation box loss curve decreases throughout training for all models, indicating improved bounding box localization. The loss reduction is sharper in the early epochs and

ISI Journal of Information Systems and Informatics
Published By Asosiasi Doktor Sistem Informasi Indonesia

becomes smoother after the models adapt to the dataset. No clear indication of severe overfitting was observed in the aggregated training curves, as the validation metrics continued to improve or remained stable during training.

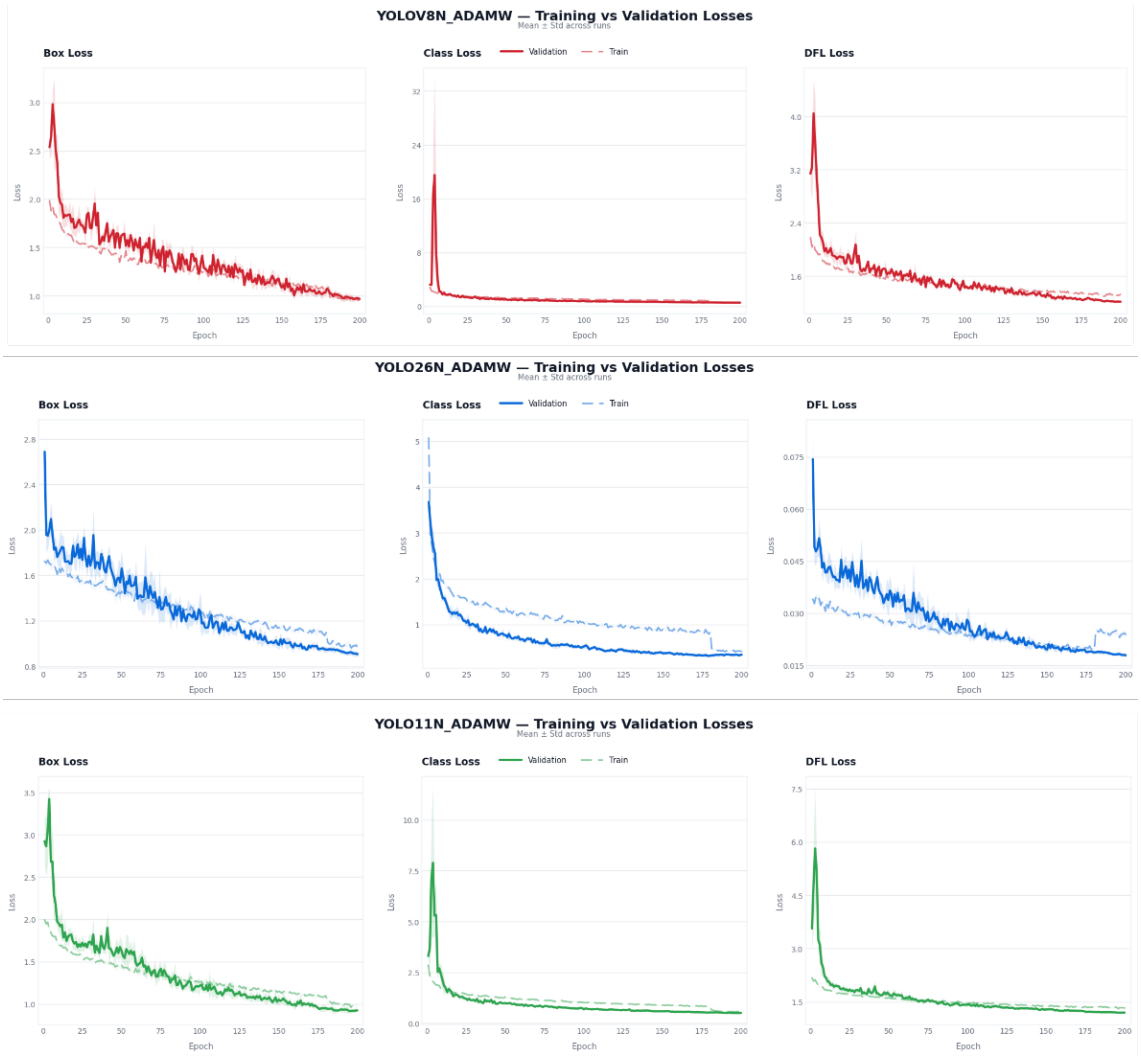


Figure 8. Detailed training and validation loss curves for YOLOv8n, YOLO26n, and YOLO11n

Figure 8 provides a more detailed view of the training and validation loss behavior for each architecture. For YOLOv8n, the box loss, class loss, and DFL loss decrease consistently across epochs. The validation class loss shows a sharp spike during early training, then quickly stabilizes and remains close to the training curve afterward. This indicates that YOLOv8n experienced early adaptation instability but did not show a persistent divergence between training and validation losses.

For YOLO11n, the loss curves show a similar decreasing pattern. The early validation spikes are visible in the box, class, and DFL losses, but the curves rapidly decline and stabilize after the initial epochs. The gap between the training and validation losses remains within control, suggesting that YOLO11n maintained stable generalization behavior throughout training. This is consistent with the aggregated training curves, which show that YOLO11n shows great and stable improvement after the early adaptation phase.

For YOLO26n, the box loss and class loss decrease steadily across epochs. The DFL-related loss appears at a much smaller numerical scale than in YOLOv8n and YOLO11n. This difference should be interpreted carefully. In the YOLO26n implementation used in this study, the simplified box regression setting with reg_max = 1 makes the DFL-related component behave differently from the full distribution-based regression used in earlier variants. Therefore, the DFL-related loss curve for YOLO26n is retained because it appears in the Ultralytics training log, but it should not be compared directly on the same numerical scale as the DFL losses of YOLOv8n and YOLO11n.

Overall, the training curves indicate that all models converged to a stable state under the same AdamW optimization and augmentation settings. Based on the aggregated training and validation curves, the models did not show a clear pattern of severe overfitting. The validation loss did not increase persistently, and the gap between training and validation behavior did not consistently widen during training. These findings support the use of the fixed test set and multi-seed statistical aggregation for the subsequent model comparison.

3.3. Validation-Based Multi-Seed Robustness Performance

After confirming stable convergence during training, each model was evaluated on the fixed validation set across five random seeds. The validation results were aggregated as mean ± standard deviation to assess both detection performance and robustness under repeated training runs. This validation-based evaluation was used because single-run results may not fully represent model behavior, especially under limited-data conditions. The fixed test set was kept separate and used only for final CPU inference and for the accuracy-efficiency evaluation. The validation-based multi-seed robustness performance is shown in Table 5.

ISI Journal of Information Systems and Informatics
Published By Asosiasi Doktor Sistem Informasi Indonesia

Table 5. Validation-based multi-seed robustness performance of YOLOv8n, YOLO11n, and YOLO26n

Model	Precision	Recall	F1-score	mAP@50	mAP@50-95	Fitness
YOLO26n	0.9673 ± 0.0120	0.9531 ± 0.0158	0.9601 ± 0.0106	0.9837 ± 0.0069	**0.7358 ± 0.0039**	**0.7606 ± 0.0041**
YOLO11n	**0.9800 ± 0.0054**	0.9330 ± 0.0284	0.9558 ± 0.0168	0.9839 ± 0.0048	0.7274 ± 0.0121	0.7530 ± 0.0112
YOLOv8n	0.9631 ± 0.0146	**0.9574 ± 0.0156**	**0.9602 ± 0.0120**	**0.9848 ± 0.0056**	0.7212 ± 0.0092	0.7475 ± 0.0085

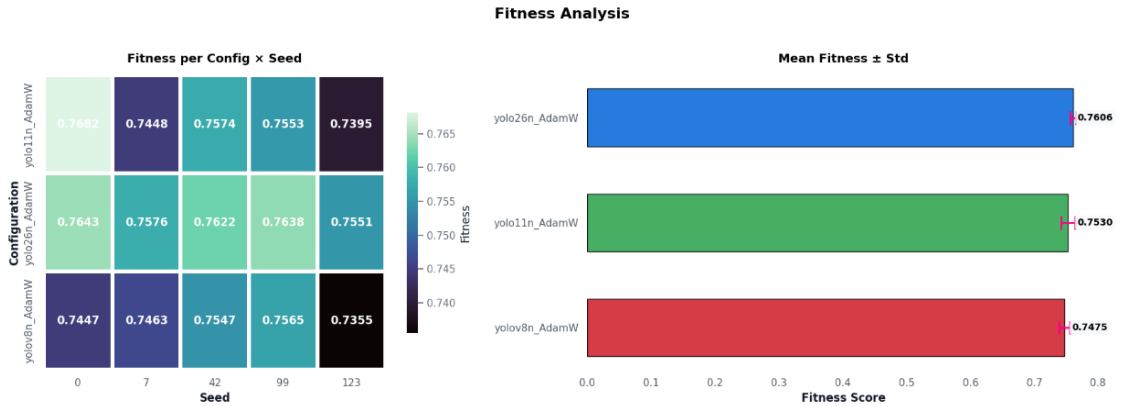
As shown in Table 5, all models achieved high average validation performance, but their robustness profiles differed across metrics. YOLO11n achieved the highest precision (0.9800 ± 0.0054), indicating that it produced fewer false positive detections across repeated validation runs. However, its recall showed the greatest variability (0.9330 ± 0.0284), suggesting that its ability to detect all ground-truth young crescent-moon objects was less stable across seeds. YOLOv8n achieved the highest validation recall (0.9574 ± 0.0156), F1-score (0.9602 ± 0.0120), and mAP@50 (0.9848 ± 0.0056). This result indicates strong detection capability under the standard IoU threshold on the validation set. However, its mAP@50-95 (0.7212 ± 0.0092) and fitness (0.7475 ± 0.0085) were lower than those of YOLO26n, suggesting weaker robustness under stricter localization thresholds.

YOLO26n achieved the highest validation mAP@50-95 (0.7358 ± 0.0039) and fitness (0.7606 ± 0.0041). It also produced the lowest standard deviation for these two key metrics. Since mAP@50-95 evaluates localization quality across stricter IoU thresholds and fitness gives stronger weight to this metric, YOLO26n demonstrated the most robust validation performance across five seeds. Therefore, from a validation-based multi-seed robustness perspective, YOLO26n provided the strongest balance between average detection performance and stability, even though it did not dominate every individual metric. Overall, the validation-based multi-seed results show that model selection should not rely only on the highest score in a single metric. YOLO11n was the most precise model, YOLOv8n achieved the strongest recall-oriented performance, and YOLO26n provided the most stable localization and fitness performance on the validation set. These findings

confirm the importance of reporting mean ± standard deviation when evaluating object detection models on a limited young crescent moon dataset.

3.4. Validation-Based Fitness Comparison

To further examine validation robustness, the fitness score was analyzed across all configurations and random seeds. In this study, fitness was used as a summary indicator because it gives greater weight to mAP@50-95, thereby emphasizing stricter localization quality rather than relying only on detection at a single IoU threshold. The per-seed validation fitness distribution and aggregated mean fitness values are shown in Figure 9.



(a) (b)

Figure 9. Validation fitness analysis: (a) fitness score for each configuration and random seed, and (b) mean fitness ± standard deviation

As shown in Figure 9(a), the fitness values vary across seeds for all three models, confirming that repeated training can produce slightly different outcomes even under the same experimental configuration. However, the pattern of variation differs across models. YOLO11n shows the widest spread, ranging from 0.7395 to 0.7682, indicating that its performance is more sensitive to seed variation. YOLOv8n also shows noticeable variation, with fitness values ranging from 0.7355 to 0.7565. In contrast, YOLO26n shows a narrower range, from 0.7551 to 0.7643, suggesting more consistent behavior across repeated runs.

This trend is further confirmed in Figure 9(b). YOLO26n achieved the highest mean fitness (0.7606) and the smallest standard deviation. YOLO11n ranked second with a mean fitness of 0.7530, while YOLOv8n obtained 0.7475. These results indicate that YOLO26n provides

the strongest overall balance between average performance and stability. Although YOLO11n produced the best single-seed fitness result, its larger variation suggests lower robustness across repeated runs. Therefore, from a fitness-based robustness perspective, YOLO26n can be considered the most reliable model among the three evaluated architectures. Overall, the fitness analysis strengthens the findings in Table 5. It shows that model ranking should not be based solely on the best single run, but also on performance consistency across seeds. This is particularly important for limited young crescent moon datasets, where stochastic training effects can meaningfully affect evaluation outcomes.

3.5. Cpu Inference And Accuracy-Efficiency Trade-Off

CPU inference performance was evaluated on the fixed test set under FP32 precision without GPU acceleration. Detection accuracy and inference efficiency were measured using the metrics defined in Section 2.4.2, with the complete results summarized in Table 6. Inference time refers to the time elapsed from input preparation to the model's forward pass. Total pipeline time includes preprocessing, model inference, post-processing, and result extraction. For YOLOv8n and YOLO11n, post-processing includes NMS-based filtering with max_det=1, while YOLO26n follows its end-to-end NMS-free inference behavior without max_det and uses the highest-confidence detection from the model output. Therefore, total pipeline time is a more practical indicator of CPU-only deployment cost.

Table 6. CPU inference performance and test-set fitness on the fixed test set

Model	Val Fitness	Test Precision	Test Recall	Test F1-score	Test mAP@50	Test mAP@50-95	Test Fitness	Inference Time (ms)	Total Pipeline Time (ms)	FPS
YOLO26n	**0.7606 ± 0.0041**	0.9612 ± 0.0249	0.9542 ± 0.0228	0.9575 ± 0.0188	**0.9827 ± 0.0108**	**0.7818 ± 0.0121**	**0.8019 ± 0.0113**	**258.346 ± 10.633**	**261.468 ± 10.933**	**3.82 ± 0.13**
YOLO11n	0.7530 ± 0.0112	0.9774 ± 0.0125	0.9712 ± 0.0149	0.9742 ± 0.0120	0.9700 ± 0.0151	0.7610 ± 0.0093	0.7820 ± 0.0091	334.346 ± 89.609	340.884 ± 90.487	3.06 ± 0.60
Model	**Val Fitness**	**Test Precision**	**Test Recall**	**Test F1-score**	**Test mAP@50**	**Test mAP@50-95**	**Test Fitness**	**Inference Time (ms)**	**Total Pipeline Time (ms)**	**FPS**
YOLOv8n	0.7475 ± 0.0085	**0.9854 ± 0.0093**	**0.9774 ± 0.0127**	**0.9814 ± 0.0093**	0.9755 ± 0.0131	0.7471 ± 0.0212	0.7699 ± 0.0200	297.580 ± 13.955	302.922 ± 15.022	3.34 ± 0.15

As shown in Table 6, YOLOv8n achieved the highest test precision (0.9854 ± 0.0093), recall (0.9774 ± 0.0127), and F1-score (0.9814 ± 0.0093). YOLO26n achieved the highest test mAP@50 (0.9827 ± 0.0108), mAP@50-95 (0.7818 ± 0.0121), and fitness (0.8019 ± 0.0113), while also producing the lowest total pipeline time and highest throughput. Since fitness weights mAP@50-95 more strongly than mAP@50, this metric primarily drives the final ranking under stricter IoU thresholds.

Table 7. Best-performing model for each validation and test metric

Metric	Best model on validation set	Best model on test set	Metric focus
Precision	YOLO11n, 0.9800 ± 0.0054	YOLOv8n, 0.9854 ± 0.0093	Best false-positive control
Recall	YOLOv8n, 0.9574 ± 0.0156	YOLOv8n, 0.9774 ± 0.0127	Best object coverage
F1-score	YOLOv8n, 0.9602 ± 0.0120	YOLOv8n, 0.9814 ± 0.0093	Best precision-recall balance
mAP@50	YOLOv8n, 0.9848 ± 0.0056	YOLO26n, 0.9827 ± 0.0108	Best standard IoU detection
mAP@50-95	YOLO26n, 0.7358 ± 0.0039	YOLO26n, 0.7818 ± 0.0121	Best stricter localization
Fitness	YOLO26n, 0.7606 ± 0.0041	YOLO26n, 0.8019 ± 0.0113	Best weighted overall score
Pipeline time	Not applicable	YOLO26n, 261.468 ± 10.933 ms	Fastest CPU pipeline
FPS	Not applicable	YOLO26n, 3.82 ± 0.13 FPS	Highest CPU throughput

Table 7 shows that no single model dominated all evaluation metrics. YOLOv8n remained strongest in test precision, recall, and F1-score. At the same time, YOLO26n provided the best accuracy-efficiency trade-off by achieving the strongest localization performance, the highest fitness, the shortest CPU pipeline time, and the highest throughput. Because the 71-image test set was not source- or condition-stratified, the test-set results should be interpreted as a controlled held-out comparison rather than as evidence of general robustness across all young crescent moon observation conditions.

ISI Journal of Information Systems and Informatics
Published By Asosiasi Doktor Sistem Informasi Indonesia

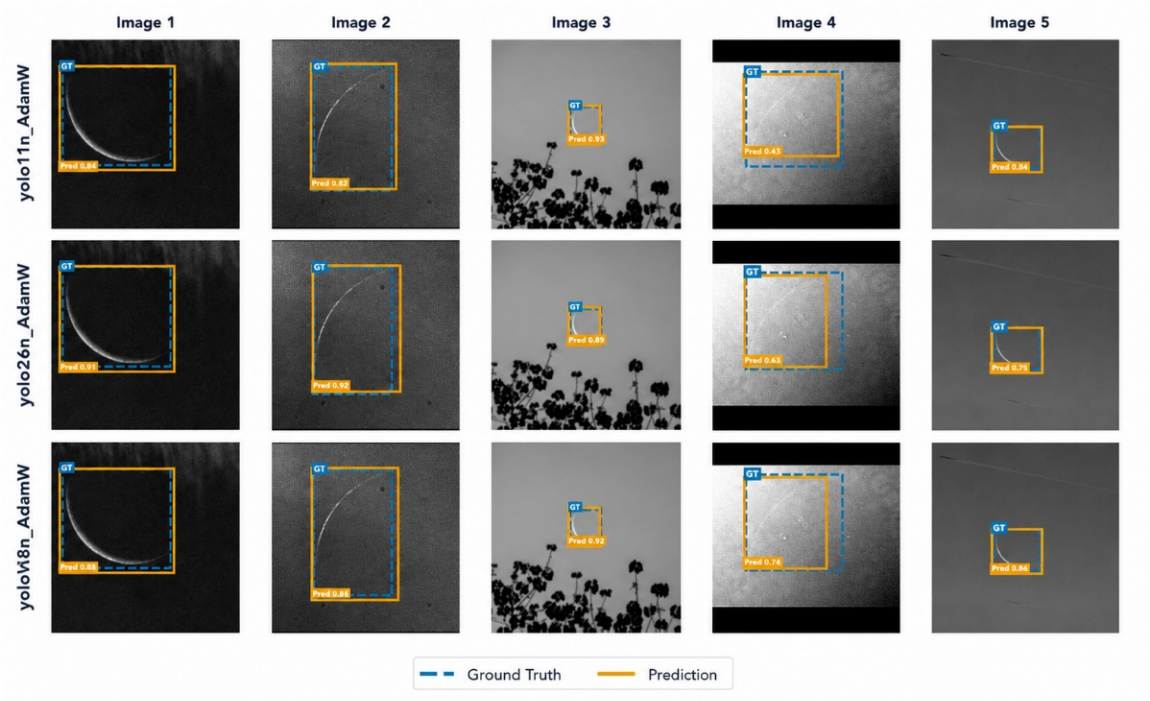


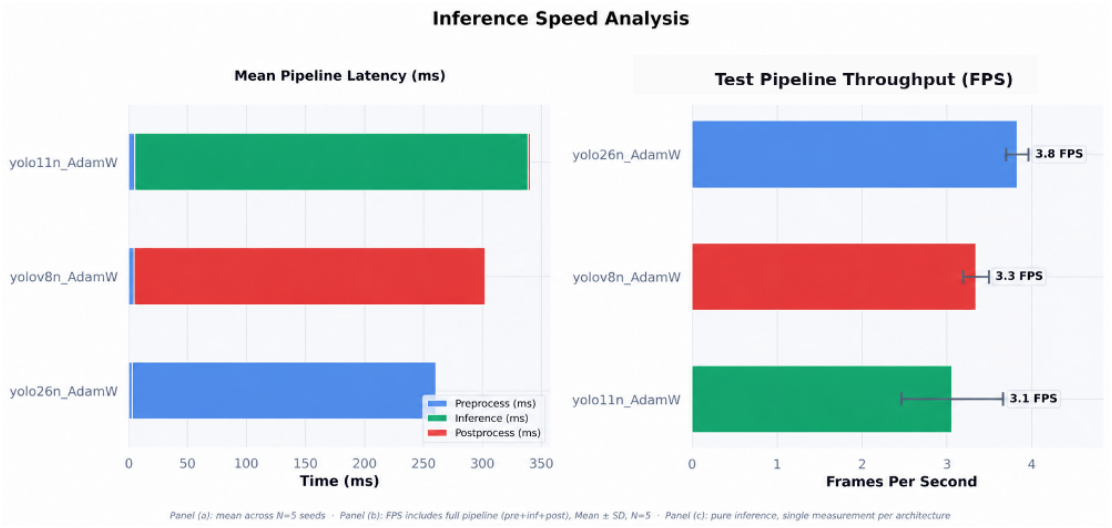
Figure 10. Representative qualitative predictions on fixed test images

Figure 10 shows representative qualitative predictions from several images in the fixed 71-image test set. Blue dashed boxes indicate ground-truth annotations, while orange boxes indicate predicted bounding boxes. This visualization is used solely for supporting evidence, while the formal model comparison remains based on the quantitative metrics in Tables 5, 6, and 7.

A separate missed-detection or inaccurate-localization panel was not included because confirmed failure-case outputs were not preserved as curated experimental records in the current experiment. Therefore, the qualitative analysis is limited to representative prediction examples and metric-level error interpretation. Source-wise or condition-wise test analysis was also not performed because detailed metadata for source distribution, contrast level, crescent size, and background category were not fully preserved at the test-split level. Future work should include structured metadata, predefined difficulty categories, curated qualitative error panels, and source-wise or condition-wise analysis to support more detailed evaluation.

Regarding the detection-error analysis, a separate confusion matrix or visual failure-case panel was not included because curated examples of false positives, false negatives, and

poor localization were not available in the current experiment. Since the dataset contains only positive young crescent moon images and does not include fully negative sky images, false-positive behavior in crescent-free scenes cannot be comprehensively evaluated. Therefore, error interpretation in this study is limited to metric-level evidence: precision reflects the false-positive rate, recall reflects the missed-detection rate, and mAP@50-95 reflects stricter localization quality.



(a) (b)

Figure 11. CPU inference speed analysis on the fixed test set: (a) mean pipeline latency and (b) pipeline throughput

Figure 11 visualizes the inference timing and throughput trends. YOLO26n achieved the lowest total pipeline time (261.468 ± 10.933 ms) and the highest throughput (3.82 ± 0.13 FPS), followed by YOLOv8n (302.922 ± 15.022 ms; 3.34 ± 0.15 FPS) and YOLO11n (340.884 ± 90.487 ms; 3.06 ± 0.60 FPS). The notably larger latency variation in YOLO11n (±90.487 ms) suggests greater sensitivity to runtime conditions across seeds. The trade-off analysis in Figure 12 shows the relationship between test-set fitness and total pipeline time. The ideal model is located in the upper-left region (higher fitness, lower latency). YOLO26n is positioned closest to this region, achieving both the highest test fitness and the lowest pipeline latency. In contrast, YOLOv8n had lower latency than YOLO11n but lower fitness, while YOLO11n achieved higher fitness at the cost of longer inference time. Overall, YOLO26n provided the most favorable accuracy-efficiency profile for lightweight image-based young crescent moon detection under the evaluated CPU-only inference setting,

achieving the highest test fitness, the lowest pipeline latency, and the highest throughput simultaneously.

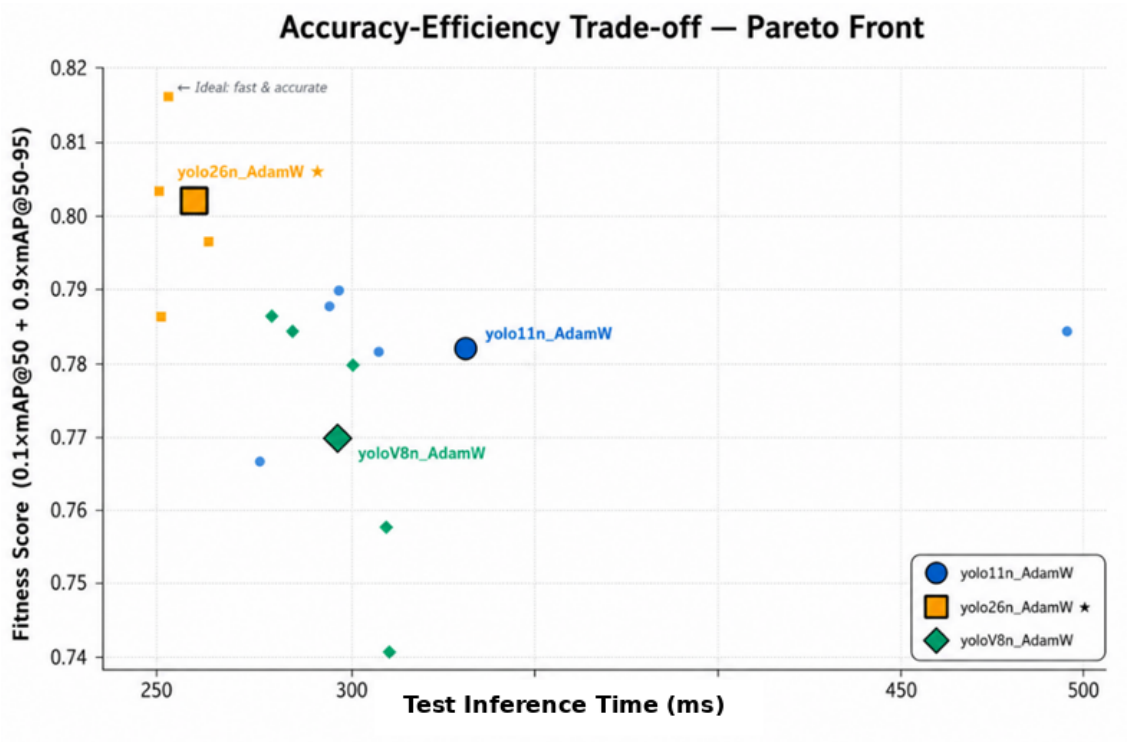


Figure 12. Accuracy-efficiency trade-off on the test set based on fitness score and pipeline inference time

3.6. Discussion

Overall, all three YOLO architectures detected young crescent moon objects effectively under limited-data conditions, demonstrating that lightweight detectors can learn useful crescent features from a small grayscale dataset. However, their performance profiles differed across metrics: YOLO11n achieved the highest validation precision, YOLOv8n the highest validation recall, F1-score, and mAP@50, while YOLO26n achieved the highest validation mAP@50-95 and fitness with the lowest seed variability. This indicates that model selection in limited-data young crescent moon detection should consider multiple metrics and seed-level stability rather than a single best run, since stochastic training effects can meaningfully shift outcomes when the dataset is small.

The localization-oriented advantage of YOLO26n stems from its compact yet deeper architectural design. Although YOLO26n has more layers than YOLOv8n and YOLO11n, it

has the lowest parameter count and GFLOPs (Table 2), suggesting efficient hierarchical feature extraction. This property is particularly relevant for young crescent moon detection because small-object localization is sensitive to bounding-box deviation — minor errors cause larger IoU drops on small objects than on large ones. Recent lightweight detection studies similarly report that efficient feature extraction, attention mechanisms, and multi-scale enhancement improve accuracy-efficiency balance in challenging small-object scenarios [33], [41], [42], [43].

A notable observation emerges when comparing the validation results in Table 5 with the test-set results in Table 6, as summarized in Table 8. The test fitness was consistently higher than the validation fitness for all three models. This increase was mainly driven by mAP@50-95, which improved on the test set, while mAP@50 remained stable or slightly decreased. Since the fitness score assigns greater weight to mAP@50-95, the higher test mAP@50-95 directly increased the final fitness value. This pattern suggests that the 71 test images were, on average, easier to localize at stricter IoU thresholds than the 139 validation images. Possible explanations include an imbalance in split difficulty, source-level bias, or visual similarity among images collected from the same observation sessions. Therefore, the higher test fitness should not be interpreted as evidence that the test set is more representative or more reliable, but as an outcome of the fixed split used in this benchmark.

Table 8. Comparison of validation and test-set mAP and fitness

Model	Metric	Validation	Test	Test – Val ±
YOLO26n	mAP@50	0.9837 ± 0.0069	0.9827 ± 0.0108	-0.0010
YOLO26n	mAP@50-95	0.7358 ± 0.0039	0.7818 ± 0.0121	+0.0460
YOLO26n	Fitness	0.7606 ± 0.0041	0.8019 ± 0.0113	+0.0413
YOLO11n	mAP@50	0.9839 ± 0.0048	0.9700 ± 0.0151	-0.0139
YOLO11n	mAP@50-95	0.7274 ± 0.0121	0.7610 ± 0.0093	+0.0336
YOLO11n	Fitness	0.7530 ± 0.0112	0.7820 ± 0.0091	+0.0290
YOLOv8n	mAP@50	0.9848 ± 0.0056	0.9755 ± 0.0131	-0.0093
YOLOv8n	mAP@50-95	0.7212 ± 0.0092	0.7471 ± 0.0212	+0.0259
YOLOv8n	Fitness	0.7475 ± 0.0085	0.7699 ± 0.0200	+0.0224

The data integrity check in this study was limited to explicit duplicate detection using SHA-256 pixel-level hashing and bounding-box coordinate overlap. This procedure is useful for detecting pixel-identical images and identical annotation patterns. Still, it cannot identify near-duplicate images caused by similar capture sessions, small cropping differences, compression changes, brightness changes, or repeated observations of the same crescent. Therefore, the absence of explicit overlap should not be interpreted as a guarantee that all visual similarity across splits has been removed. Future benchmarks should add perceptual hashing, feature-embedding similarity checks, and source- or session-based stratified splitting to reduce the risk of near-duplicate leakage.

The CPU inference results align with the efficiency interpretation. YOLO26n achieved the shortest pipeline time and highest throughput, consistent with its lower parameter count and GFLOPs. Prior lightweight detection studies report similar trends, where reduced computational complexity translates into faster inference [32], [33]. However, the obtained 3.82 FPS should be interpreted as support for image-based or low-frame-rate observation of the young crescent moon under CPU-only deployment, not as real-time video performance.

In a video-based observation workflow, the evaluated models may face additional challenges that are not fully represented by the current single-frame image dataset. Real observation videos may contain motion blur, atmospheric turbulence, sensor noise, compression artifacts, and frame redundancy. Motion blur and atmospheric noise can weaken crescent boundaries and reduce localization confidence, while frame redundancy can lead to repeated detections across similar frames. Therefore, applying the detector to video should involve frame selection, temporal filtering, and human verification rather than treating every frame as an independent final decision.

Although CPU-only evaluation on Google Colab provides an initial indication of the feasibility of lightweight deployment, it does not fully reflect performance on real edge devices. Actual deployment on devices such as Raspberry Pi, Jetson Nano, or other low-power embedded platforms may be affected by memory bandwidth, processor architecture, thermal throttling, camera input latency, and runtime optimization. The lower pipeline time of YOLO26n is likely due to a combination of a lower parameter count, lower GFLOPs, a deployment-oriented architecture, and NMS-free inference.

However, runtime implementation and hardware execution behavior can also affect measured latency. Therefore, the latency advantage should be interpreted as an empirical result under the evaluated Ultralytics implementation and CPU-only environment, not as a universal guarantee across all hardware and runtimes.

Compared with previous crescent detection studies, the present study differs in evaluation focus and deployment setting. The Mask R-CNN-based approach trained on robotic-telescope images [2] demonstrates the feasibility of deep instance-based crescent detection. Still, it does not focus on multi-seed robustness or lightweight CPU-only inference across YOLO nano architectures. The video-based computer vision approach [15] is closer to practical observation workflows because it processes temporal visual input. Still, its focus differs from the present single-frame benchmark and does not directly evaluate the stability of repeated training. Compared with the Haar-Cascade and SVM approach by Muztaba et al. [14], this study uses a different detection paradigm. Their crescent detection pipeline used contrast-enhancement preprocessing, Cascade Classifier training with positive and negative images, and SVM-based object validation. In contrast, this study uses YOLO-based single-class bounding-box localization, where non-annotated regions outside the crescent bounding boxes contribute to background learning during training. These differences show that the present contribution lies not only in crescent localization but also in controlled multi-seed benchmarking, leakage checking, and CPU inference profiling.

In practical crescent observation, the proposed detector should be integrated as a decision-support component within the existing observation workflow. The system can highlight candidate crescent locations, provide bounding-box evidence, and support observer review. Still, the final interpretation should remain under human verification by trained astronomical observers and relevant religious or governmental authorities. This integration is important because crescent visibility is affected not only by image-level detection, but also by observation context, astronomical parameters, instrument setup, and official decision procedures.

These findings indicate that YOLO26n is suitable for lightweight image-based young crescent moon detection on the evaluated dataset, with the evaluated training configuration and CPU-only inference setting. However, this study has several limitations.

First, the dataset contains 697 grayscale positive images and remains limited in its ability to cover all possible conditions for observing young crescent moons. Second, the task was formulated as single-class crescent localization on positive young crescent moon images. The absence of fully negative sky images remains an important limitation, as false-positive behavior in crescent-free sky scenes cannot be comprehensively evaluated. Third, the fixed 71-image test set was useful for controlled comparison, but it was not explicitly stratified by source, brightness level, crescent size, or background condition. Fourth, the integrity check only detected explicit duplicates using SHA-256 hashing and bounding-box coordinate overlap, while near-duplicate visual similarity may still exist. Fifth, CPU inference was measured in a Google Colab CPU-only environment using FP32 precision, so the reported 3.82 FPS should be interpreted as low-frame-rate image-based analysis rather than real-time video deployment. Future work should use larger, stratified datasets, include negative images with empty label files, apply perceptual hashing or source/session-based splitting, evaluate on real observation videos, and test deployment on dedicated edge hardware.

4. CONCLUSION

This study presented a reproducible benchmark protocol for image-based young crescent moon detection using 697 grayscale images, a fixed data split, five random seeds, and CPU-only inference evaluation. Under the evaluated setting, YOLO26n provided the most favorable accuracy-efficiency trade-off among the tested lightweight YOLO models, showing stronger localization-oriented robustness and faster CPU pipeline performance. However, YOLOv8n and YOLO11n still showed advantages in several individual metrics. The benchmark protocol combines grayscale preprocessing, fixed data partitioning, single-class annotation, data integrity checking, multi-seed robustness analysis, and CPU inference profiling. The findings should be interpreted within the scope of this dataset and experimental design, since the study used only positive crescent images, did not include fully negative sky images, used a non-stratified test set, and measured inference in a Google Colab CPU-only environment rather than real video or edge-device deployment. Future work should expand and stratify the dataset, include negative samples with empty label files, apply near-duplicate detection through perceptual hashing or source/session-based splitting, evaluate real observation videos, and test deployment on dedicated edge devices.

REFERENCES

[1] W. N. J. Hj Wan Yussof, M. Man, R. Umar, A. N. Zulkeflee, E. A. Awalludin, and N. Ahmad, "Enhancing Moon Crescent Visibility Using Contrast-Limited Adaptive Histogram Equalization and Bilateral Filtering Techniques," *Journal of Telecommunications and Information Technology*, vol. 2022, no. 1, pp. 3–13, Mar. 2022, doi: 10.26636/JTIT.2022.155721.

[2] R. Muztaba, H. L. Malasan, and M. Djamal, "Deep learning for crescent detection and recognition: Implementation of Mask R-CNN to the observational Lunar dataset collected with the Robotic Lunar Telescope System," *Astronomy and Computing*, vol. 45, p. 100757, Oct. 2023, doi: 10.1016/J.ASCOM.2023.100757.

[3] Maskufa, Sopa, S. Hidayati, and A. Damanhuri, "Implementation of the New MABIMS Crescent Visibility Criteria: Efforts to Unite the Hijriyah Calendar in the Southeast Asian Region," *AHKAM : Jurnal Ilmu Syariah*, vol. 22, no. 1, pp. 209–236, Jun. 2022, doi: 10.15408/AJIS.V22I1.22275.

[4] A. Mufid and T. Djamaluddin, "The implementation of new minister of religion of Brunei, Indonesia, Malaysia, and Singapore criteria towards the Hijri calendar unification," *HTS Teologiese Studies / Theological Studies*, vol. 79, no. 1, p. 8, Jun. 2023, doi: 10.4102/HTS.V79I1.8774.

[5] J. Terven, D.-M. Córdova-Esparza, and J.-A. Romero-González, "A Comprehensive Review of YOLO Architectures in Computer Vision: From YOLOv1 to YOLOv8 and YOLO-NAS," *Mach. Learn. Knowl. Extr.*, vol. 5, no. 4, pp. 1680–1716, Nov. 2023, doi: 10.3390/make5040083.

[6] Z. Wang *et al.*, "PC-YOLO11s: A Lightweight and Effective Feature Extraction Method for Small Target Image Detection," *Sensors*, vol. 25, no. 2, p. 348, Jan. 2025, doi: 10.3390/s25020348.

[7] W. D. Jima, S. F. Desta, T. A. Tarekegn, G. S. Gebremedhin, A. B. Gutema, and T. G. Debelee, "Tsetse Fly Detection and Sex Classification Model Enrichment Employing YOLOv8 and YOLO11 Architecture," *Applied AI Letters*, vol. 6, no. 3, p. e70004, Oct. 2025, doi: 10.1002/ail2.70004.

[8] H.-V. Nguyen, J.-H. Bae, Y.-E. Lee, H.-S. Lee, and K.-R. Kwon, "Comparison of Pre-Trained YOLO Models on Steel Surface Defects Detector Based on Transfer Learning with GPU-Based Embedded Devices," *Sensors*, vol. 22, no. 24, p. 9926, Dec. 2022, doi: 10.3390/s22249926.

ISI Journal of Information Systems and Informatics
Published By Asosiasi Doktor Sistem Informasi Indonesia

[9] D. Dai, J. Gao, S. Parsons, and E. Sklar, "Small datasets for fruit detection with transfer learning," *UKRAS21 Conference: Robotics at home Proceedings*, vol. 4, pp. 5–6, Jul. 2021, doi: 10.31256/NF6UH8Q.

[10] M. Al-Rajab, S. Loucif, and Y. Al Risheh, "Predicting new crescent moon visibility applying machine learning algorithms," *Sci. Rep.*, vol. 13, no. 1, p. 6674, Apr. 2023, doi: 10.1038/s41598-023-32807-x.

[11] S. Loucif, M. Al-Rajab, R. Abu Zitar, and M. Rezk, "Toward a globally lunar calendar: a machine learning-driven approach for crescent moon visibility prediction," *J. Big Data*, vol. 11, no. 1, p. 114, Aug. 2024, doi: 10.1186/s40537-024-00979-6.

[12] Z. T. Allawi, "Crescent Moon Visibility: A New Criterion using Deep learned Artificial Neural-Network," *Iraqi Journal of Science*, pp. 2332–2343, Apr. 2024, doi: 10.24996/ijs.2024.65.4.45.

[13] A. L. A. Mohd Nasir *et al.*, "Comparative Analysis of Image Processing Technique in Determining the New Crescent Moon Visibility," *J. Phys. Conf. Ser.*, vol. 2915, no. 1, p. 012004, Dec. 2024, doi: 10.1088/1742-6596/2915/1/012004.

[14] R. Muztaba, H. L. Malasan, and M. Djamal, "A Self-Construction of Automatic Crescent Detection Using Haar-Cascade Classifier and Support Vector Machine," *J. Phys. Conf. Ser.*, vol. 2734, no. 1, p. 012007, Mar. 2024, doi: 10.1088/1742-6596/2734/1/012007.

[15] J. A. Utama *et al.*, "Young lunar crescent detection based on video data with computer vision techniques," *Astronomy and Computing*, vol. 44, p. 100731, Jul. 2023, doi: 10.1016/J.ASCOM.2023.100731.

[16] D. Picard, "Torch.manual_seed(3407) is all you need: On the influence of random seeds in deep learning architectures for computer vision," *arXiv:*2109.08203, May 2023. Accessed: May 06, 2026. [Online]. Available: http://arxiv.org/abs/2109.08203

[17] M. Rosenblatt, L. Tejavibulya, R. Jiang, S. Noble, and D. Scheinost, "Data leakage inflates prediction performance in connectome-based machine learning models," *Nat. Commun.*, vol. 15, no. 1, p. 1829, Feb. 2024, doi: 10.1038/s41467-024-46150-w.

[18] E. P. Silmina, S. Sunardi, and A. Yudhana, "COMPARATIVE ANALYSIS OF YOLO DEEP LEARNING MODEL FOR IMAGE-BASED BEEF FRESHNESS DETECTION," *JITK (Jurnal Ilmu Pengetahuan dan Teknologi Komputer)*, vol. 11, no. 1, pp. 250–265, Aug. 2025, doi: 10.33480/JITK.V11I1.6784.

[19] Pexels, "Young Crescent Search Results," 2026. Accessed: May 24, 2026. [Online]. Available: https://www.pexels.com/search/young%20crescent/

[20] Badan Meteorologi Klimatologi dan Geofisika, "Galeri Pengamatan Hilal," 2026. Accessed: May 24, 2026. [Online]. Available: https://hilal.bmkg.go.id/gallery

[21] M. Elsässer, "Mondbeobachtungen / Observations of the Moon," 2025. Accessed: May 24, 2026. [Online]. Available: https://www.mondatlas.de/beob_sicheln.html

[22] International Astronomy Center, "Islamic Crescents' Observation Project (ICOP)," 2026. Accessed: May 24, 2026. [Online]. Available: https://www.icoproject.org/

[23] Pexels, "Free Stock Photo and Video License," 2026. Accessed: May 24, 2026. [Online]. Available: https://www.pexels.com/license/

[24] P. Skalski, "Makesense.ai," 2019. Accessed: May 24, 2026. [Online]. Available: https://www.makesense.ai/

[25] C. Wang, Q. Wang, Y. Qian, Y. Hu, Y. Xue, and H. Wang, "DP-YOLO: Effective Improvement Based on YOLO Detector," *Applied Sciences*, vol. 13, no. 21, p. 11676, Oct. 2023, doi: 10.3390/app132111676.

[26] Q. Xu, Y. Li, and Z. Shi, "LMO-YOLO: A Ship Detection Model for Low-Resolution Optical Satellite Imagery," *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.*, vol. 15, pp. 4117–4131, 2022, doi: 10.1109/JSTARS.2022.3176141.

[27] X. Li, L. Chang, and X. Liu, "CE-Dedup: Cost-Effective Convolutional Neural Nets Training based on Image Deduplication," in *Proc. IEEE ISPA/BDCloud/SocialCom/SustainCom*, IEEE, Sep. 2021, pp. 11–18. doi: 10.1109/ISPA-BDCloud-SocialCom-SustainCom52081.2021.00017.

[28] R. Sapkota, R. H. Cheppally, A. Sharda, and M. Karkee, "YOLO26: Key Architectural Enhancements and Performance Benchmarking for Real-Time Object Detection," *arXiv:*2509.25164, Mar. 2026. Accessed: May 06, 2026. [Online]. Available: http://arxiv.org/abs/2509.25164

[29] P. Hidayatullah and R. Tubagus, "YOLO26: A Comprehensive Architecture Overview and Key Improvements," *arXiv:*2602.14582, Feb. 2026. Accessed: May 06, 2026. [Online]. Available: http://arxiv.org/abs/2602.14582

[30] P. Hidayatullah, N. Syakrani, M. R. Sholahuddin, T. Gelar, and R. Tubagus, "YOLOv8 to YOLO11: A Comprehensive Architecture In-depth Comparative Review," *Jurnal RESTI (Rekayasa Sistem dan Teknologi Informasi)*, vol. 10, no. 2, pp. 341–354, Apr. 2025, doi: 10.29207/RESTI.V10I2.6598.

[31] Ultralytics, "Ultralytics YOLO Documentation," 2026. Accessed: May 24, 2026. [Online]. Available: https://docs.ultralytics.com/

[32] P. Ruiz-Ponce, D. Ortiz-Perez, J. Garcia-Rodriguez, and B. Kiefer, "POSEIDON: A Data Augmentation Tool for Small Object Detection Datasets in Maritime Environments," *Sensors*, vol. 23, no. 7, p. 3691, Apr. 2023, doi: 10.3390/s23073691.

[33] L. Liao, L. Luo, J. Su, Z. Xiao, F. Zou, and Y. Lin, "Eagle-YOLO: An Eagle-Inspired YOLO for Object Detection in Unmanned Aerial Vehicles Scenarios," *Mathematics*, vol. 11, no. 9, p. 2093, Apr. 2023, doi: 10.3390/math11092093.

[34] J.-H. Kim, N. Kim, Y. W. Park, and C. S. Won, "Object Detection and Classification Based on YOLO-V5 with Improved Maritime Dataset," *J. Mar. Sci. Eng.*, vol. 10, no. 3, p. 377, Mar. 2022, doi: 10.3390/jmse10030377.

[35] E. Thibeau-Sutre *et al.*, "ClinicaDL: An open-source deep learning software for reproducible neuroimaging processing," *Comput. Methods Programs Biomed.*, vol. 220, p. 106818, Jun. 2022, doi: 10.1016/J.CMPB.2022.106818.

[36] A. Ghezal and A. König, "A Comparative Study of Hybrid Machine-Learning vs. Deep-Learning Approaches for Varroa Mite Detection and Counting," *Sensors*, vol. 25, no. 16, p. 5075, Aug. 2025, doi: 10.3390/s25165075.

[37] D. Kolosov, V. Kelefouras, P. Kourtessis, and I. Mporas, "Anatomy of Deep Learning Image Classification and Object Detection on Commercial Edge Devices: A Case Study on Face Mask Detection," *IEEE Access*, vol. 10, pp. 109167–109186, 2022, doi: 10.1109/ACCESS.2022.3214214.

[38] L. Wood and F. Chollet, "Efficient Graph-Friendly COCO Metric Computation for Train-Time Model Evaluation," *arXiv*:2207.12120, Jul. 2022. Accessed: May 06, 2026. [Online]. Available: http://arxiv.org/abs/2207.12120

[39] D. Einsiedel, M. Vita, F. Kaltenecker, B. Dunnewind, J. Meulendijks, and C. Krupitzer, "Automated Detection of Quality Deviations in Poultry Processing Using Step-Specific YOLOv12 Models," *Foods*, vol. 15, no. 6, p. 1019, Mar. 2026, doi: 10.3390/foods15061019.

[40] Ultralytics, "Reference for ultralytics/utils/metrics.py - Ultralytics YOLO Docs." Accessed: May 06, 2026. [Online]. Available: https://docs.ultralytics.com/reference/utils/metrics/#ultralytics.utils.metrics.ClassifyMetrics.fitness

[41] M. Zhang, S. Xu, W. Song, Q. He, and Q. Wei, "Lightweight Underwater Object Detection Based on YOLO v4 and Multi-Scale Attentional Feature Fusion," *Remote Sens. (Basel).*, vol. 13, no. 22, p. 4706, Nov. 2021, doi: 10.3390/rs13224706.

ISI Journal of Information Systems and Informatics
Published By Asosiasi Doktor Sistem Informasi Indonesia

[42] Z. Huang, J. Wu, L. Su, Y. Xie, T. Li, and X. Huang, "SP-YOLO-Lite: A Lightweight Violation Detection Algorithm Based on SP Attention Mechanism," *Electronics (Basel)*, vol. 12, no. 14, p. 3176, Jul. 2023, doi: 10.3390/electronics12143176.

[43] J. Cao, W. Bao, H. Shang, M. Yuan, and Q. Cheng, "GCL-YOLO: A GhostConv-Based Lightweight YOLO Network for UAV Small Object Detection," *Remote Sens. (Basel)*, vol. 15, no. 20, p. 4932, Oct. 2023, doi: 10.3390/rs15204932.