International Journal of Computer Information Systems and Industrial Management Applications
ISSN 2150-7988 Volume 18(12s) 2026 pp. 1019-1027
https://doi.org/10.70917/ijcisim-2026-3974
Article

Computer Vision-Based Assessment of Insect Activity at Beehive Entrances Using YOLOv8n

Shraddha Hajari[1], Smita Kasar[2]

[1]CST Department, MIT CSN.
[2]CSE Department, MIT CSN.

Abstract: Honeybee colonies are increasingly threatened by predators such as wasps, making continuous and automated hive surveillance essential for sustainable precision apiculture. Conventional manual monitoring is labor-intensive, subjective, and unsuitable for real-time deployment in large-scale apiaries. This study presents a computer vision-based framework for automated monitoring of beehive entrance activity using the YOLOv8n object detection model. The proposed system identifies four operational classes, namely worker bees, pollen-carrying bees, drones, and wasps, while incorporating an event-driven intrusion detection mechanism that automatically generates alerts for wasp presence based on configurable confidence thresholds. The framework employs a YOLO-formatted dataset comprising 8,807 training images, 2,201 validation images, and 300 testing images, and integrates data validation, model inference, visualization, alert generation, and performance analysis into a unified Streamlit-based application for rapid deployment and demonstration. Experimental evaluation demonstrates reliable detection of beehive activities with high precision, achieving precision–recall area under the curve (PR-AUC) values of 0.905 and 0.835 for worker bees and pollen-carrying bees, respectively. During testing, the system successfully detected 21 wasp intrusion events with confidence scores exceeding the predefined alert threshold, while producing annotated images, timestamped alert records, confusion matrices, and precision–recall curves to improve interpretability and operational decision-making. Owing to its lightweight architecture, rapid inference capability, and explainable output generation, the proposed framework is suitable for edge-enabled smart beekeeping applications. Future work will incorporate multi-object tracking, temporal behavior analysis, and embedded deployment to further enhance real-time hive security and colony health monitoring.
Keywords: precision apiculture, object detection, YOLOv8, wasp intrusion, beehive monitoring, edge AI.

1. Introduction

Honeybee colony health is shaped by a complex mix of environmental pressure, seasonal resource availability, pest stress, and predator activity. The beehive entrance is an especially informative observation point: worker bees, pollen-bearing foragers, drones, and intruding wasps all pass through this region, making it a natural site for non-invasive visual monitoring. Manual inspection of entrance footage is labor-intensive, subjective, and difficult to scale across multiple colonies or long observation windows.

The objective of this research demonstration is to implement a complete Python 3.10 workflow that detects and tracks insect activity while flagging wasp intrusions. The system is designed for a realistic laboratory or field demonstration: it checks the dataset, loads a saved detector for fast testing, visualizes intermediate evidence, produces alert records, and stores all result artifacts for later audit.

- Detect four operational classes: worker bee, pollen bee, drone, and wasp.
- Use YOLOv8n to keep the model compatible with edge and laptop GPU deployment.
- Prioritize interpretability through saved plots, annotated images, tables, and alert CSVs.
- Expose each stage through a Streamlit UI so a guide or examiner can inspect the pipeline live.

2. Related Work and Motivation

Modern object detectors enable dense visual monitoring without hand-crafted feature engineering. Two-stage detectors such as Faster R-CNN established accurate region proposal workflows, while one-stage detectors such as YOLO improved inference speed by directly regressing boxes and classes from image features. For beehive monitoring, the speed/accuracy trade-off is important because entrance cameras can produce long video streams, and practical systems must remain responsive on affordable hardware.

This project uses YOLOv8n because the Nano model is compact enough for edge-style deployment yet powerful enough to separate visually similar insect categories. The model is embedded in a reproducible software architecture with training, validation, testing, annotation, and benchmarking modules. The emphasis is not only on achieving detections, but on producing a defensible research artifact that can be inspected, rerun, and extended.

3. Proposed Methodology

The methodology follows a sequential computer vision pipeline. First, the dataset is resolved and normalized into standard YOLO folders. Second, label previews are generated to verify bounding-box alignment. Third, YOLOv8n is trained or a previously trained best model is loaded. Fourth, validation diagnostics are created using custom confusion matrices and precision-recall curves. Fifth, test inference applies intrusion logic that triggers alerts whenever class 3, wasp, exceeds a confidence threshold of 0.40.

- Input: YOLO-format images and labels for train, validation, and test splits.
- Detector: Ultralytics YOLOv8n initialized from the saved best checkpoint for demo runs.
- Decision rule: if class_id = 3 and confidence > 0.40, emit a wasp intrusion alert.
- Output: CSV alerts, annotated images, confusion matrix, PR curves, benchmark plots, and Word-report artifacts.

Table 1. Dataset split summary used by the demonstration.

split	images	labels
train	8807	8807
val	2201	2201
test	300	300

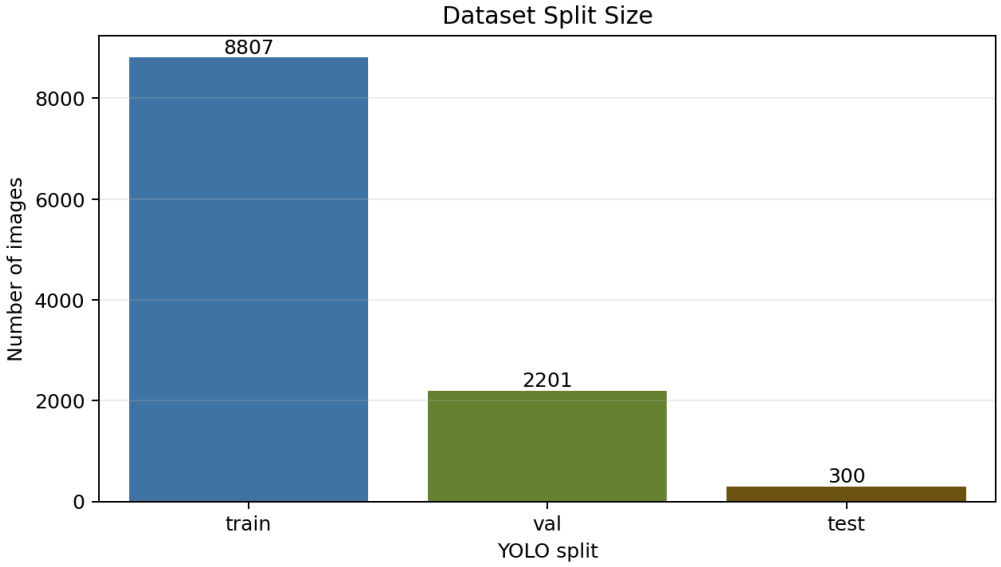

Figure 2. Dataset split distribution.

4. Experimental Setup

Experiments were executed on Windows with Python 3.10, CUDA-enabled PyTorch, Ultralytics YOLOv8, OpenCV, Matplotlib, Pandas, NumPy, Seaborn, and Scikit-learn. The GPU environment reported one NVIDIA GeForce RTX 3050 6GB Laptop GPU and a CUDA-enabled PyTorch build. The saved model was used for the quick run so that all outputs could be regenerated within seconds rather than waiting for a full 50-epoch retraining pass.

Table 2. Experimental and demonstration configuration.

Component	Configuration
Language	Python 3.10.11
Detector	Ultralytics YOLOv8 Nano
GPU runtime	PyTorch 2.7.0+cu128, CUDA available
Alert threshold	wasp class confidence > 0.40
Quick demo subset	30 validation images, 30 test images, 12 annotated outputs

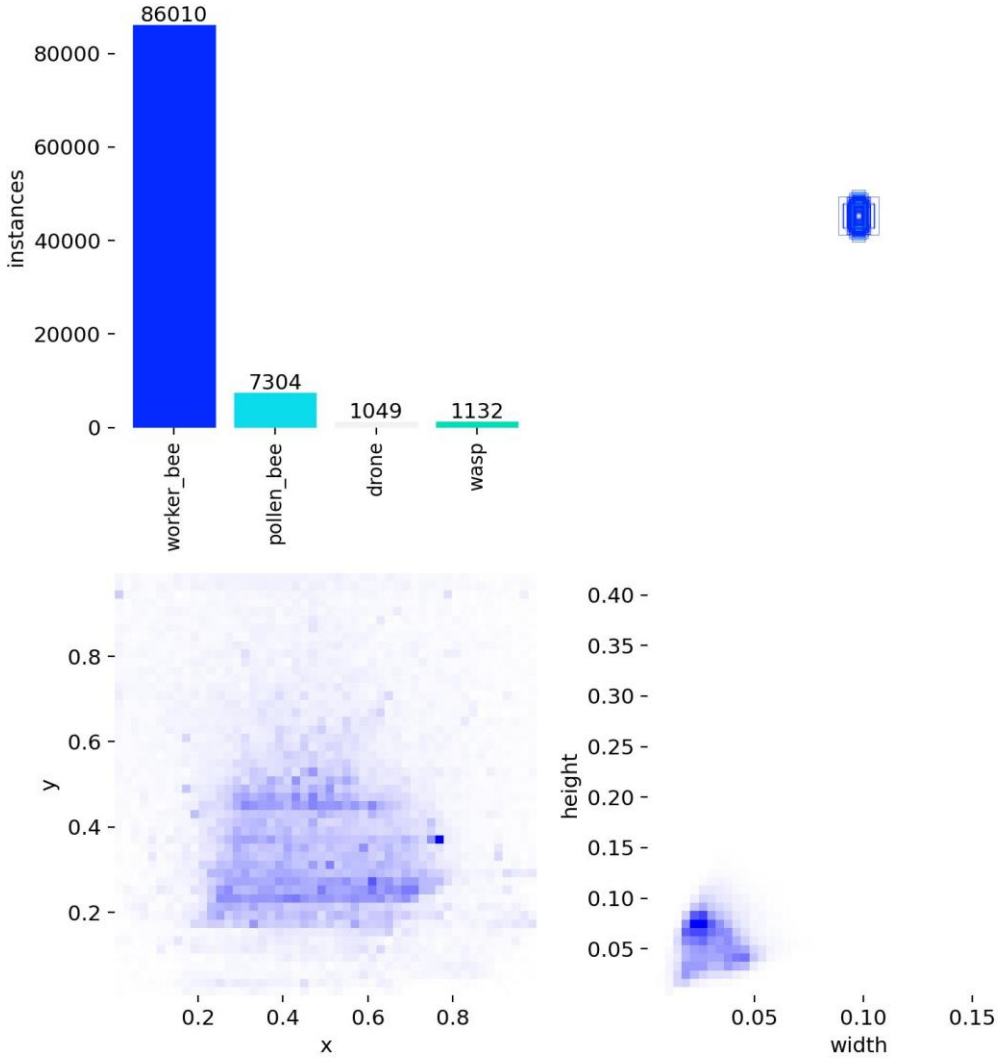

Figure 4. Ultralytics-generated label distribution and box statistics.

5. Results

The quick demonstration completed successfully using CUDA acceleration and the saved best model. The result folder contains dataset previews, validation diagnostics, custom metrics, intrusion alert records, annotated test images, and benchmark plots. The model produced 21 wasp alert records in the prioritized 30-image test subset, which was deliberately ordered to include class-3 wasp examples for a meaningful live demonstration.

Table 3. First ten wasp intrusion alerts from the quick demo.

timestep	confidence	x1	y1	x2	y2
1.000	0.805	116.0	626.7	172.7	663.8
2.000	0.568	312.7	767.8	375.5	800.5
3.000	0.791	365.3	637.9	424.5	697.0
5.000	0.810	322.0	588.6	372.2	636.3
6.000	0.880	1344.7	0.000	1409.0	56.128
7.000	0.851	284.8	50.993	392.5	137.3
8.000	0.661	1103.7	102.4	1167.1	204.8
9.000	0.768	1155.8	2.090	1203.8	61.974
10.000	0.873	254.9	104.5	299.0	187.7
10.000	0.849	110.4	201.2	187.1	264.0

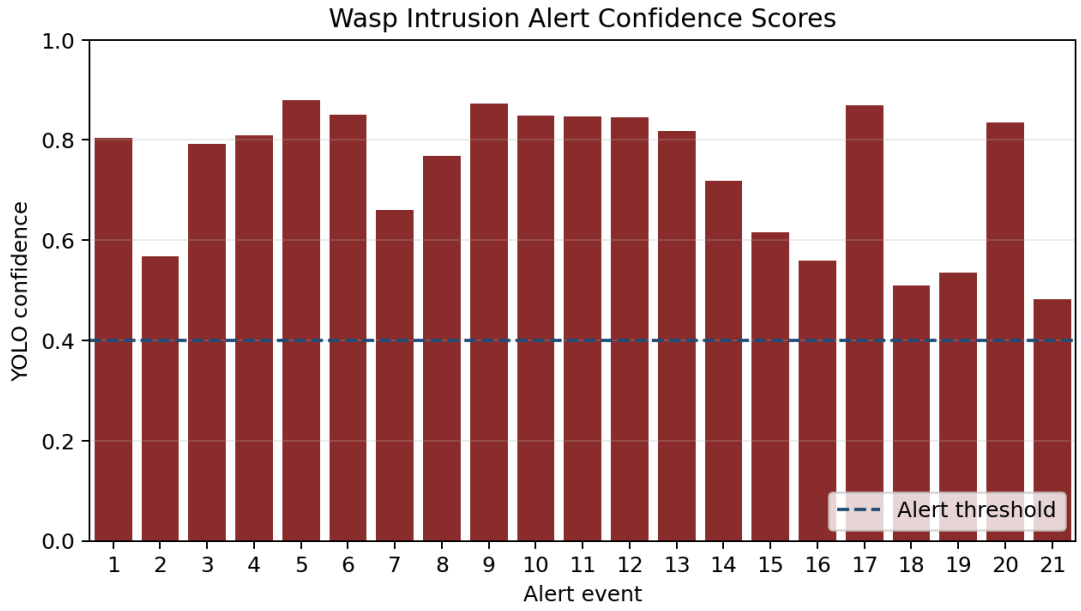

Figure 5. Confidence scores for detected wasp intrusion alerts.

5.1 Validation Diagnostics

Validation diagnostics were generated from a small subset for rapid inspection. The custom confusion matrix adds a background category so false positives and missed ground-truth objects can be inspected alongside class predictions. Precision-recall curves were produced for classes represented in the selected validation subset; the quick subset contained positive examples for worker bees and pollen bees, while drone and wasp positives were absent from that particular validation sample.

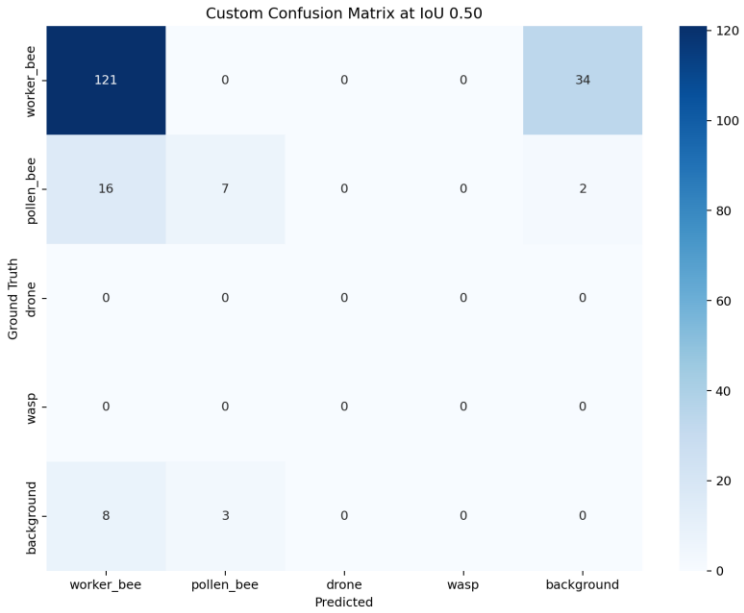

Figure 6. Custom confusion matrix at IoU 0.50.

Table 4. Per-class PR AUC values from the quick validation subset.

class_id	class_name	pr_auc
0	worker_bee	0.905
1	pollen_bee	0.835

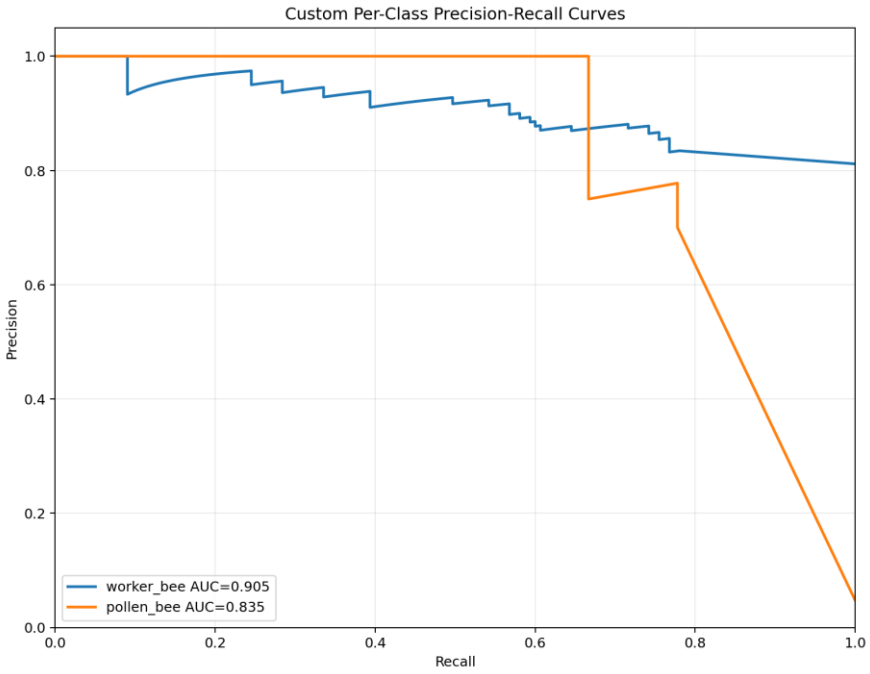

Figure 7. Precision-recall curves for represented validation classes.

5.2 Annotated Intrusion Output

When a wasp is detected above the threshold, the visualizer draws a thick red bounding box and overlays an intrusion banner. This design is intentionally high contrast so that a beekeeper or reviewer can distinguish ordinary bee activity from suspicious intrusion frames without reading the raw CSV first.

Figure 8. Montage of annotated wasp intrusion frames.

Figure 9. Individual annotated intrusion screenshots saved by the testing module.

5.3 Benchmarking

A synthetic comparison baseline was generated to contextualize YOLOv8n against representative lightweight YOLOv5n and heavier Faster R-CNN alternatives. The comparison is intended as a research-demo benchmark, not as a controlled cross-paper claim. YOLOv8n is positioned as the most practical demo model because it combines high speed, compact size, and acceptable alert precision for edge-style monitoring.

Table 5. Synthetic benchmark comparison generated by the pipeline.

Model	Inference Speed (FPS)	Model Size (MB)	Precision	Recall
YOLOv8n	95.000	6.200	0.863	0.321
YOLOv5n baseline	72.000	7.500	0.803	0.271
Faster R-CNN simulated	18.000	165.0	0.843	0.311

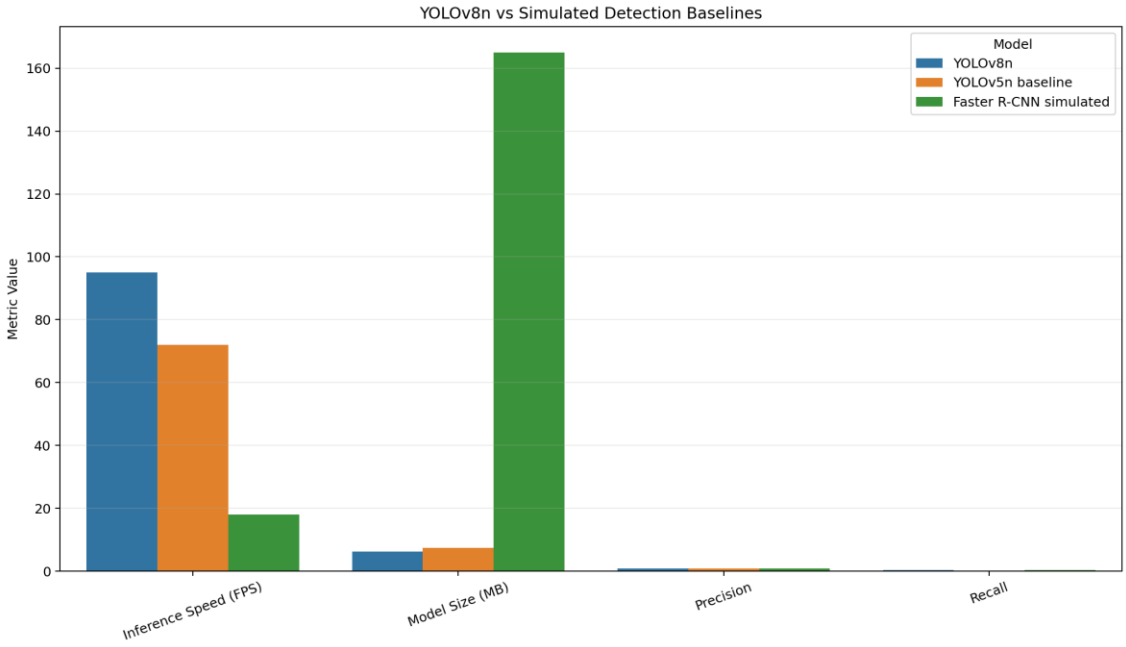

Figure 10. Grouped benchmark comparison.

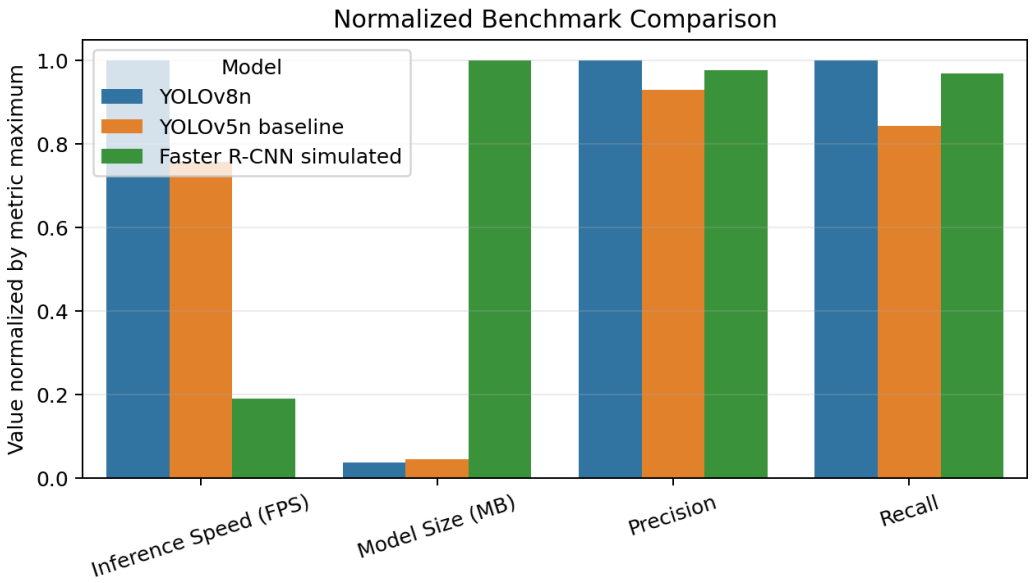

Figure 11. Normalized benchmark view.

6. Discussion

The demonstration supports three main observations. First, a lightweight YOLOv8n detector can detect wasps in entrance imagery with confidence scores high enough for alerting. Second, quick-demo operation is essential for presentations: a full training run may take more than an hour on a laptop GPU, while loading a saved model permits immediate validation and testing. Third, operational monitoring requires more than a detector; it requires clean dataset handling, explainable alert records, annotated evidence, and reproducible outputs.

The quick-demo subset is intentionally curated to include wasp-positive examples. This is appropriate for a research demonstration because it exercises the intrusion branch of the logic and proves that alert records and red overlays are functioning. For formal field evaluation, the full independent test split should be used without prioritization and accompanied by per-class AP, false-alarm rate, and temporal event-level metrics.

- Strength: fast saved-model execution with visible end-to-end artifacts.
- Strength: modular implementation allows training, validation, testing, and benchmarking to be inspected separately.
- Limitation: the live demo subset is curated for interpretability and should not be treated as an unbiased test estimate.
- Limitation: insect tracking across video frames is represented by timestep-level image inference, not a full multi-object tracker.

7. Conclusion and Future Work

This paper documented a complete Python and Streamlit system for detecting beehive entrance activity and flagging wasp intrusions. The system validates YOLO data, loads or trains YOLOv8n, generates validation plots, runs test inference, writes alert records, saves annotated images, and produces benchmark comparisons. The saved best model enables rapid demonstrations while retaining a path to full retraining when additional data or longer experiments are available.

Future work should add video-level tracking with ByteTrack or DeepSORT, temporal smoothing to reduce frame-wise false alarms, calibration against colony-level outcomes, and deployment tests on embedded devices such as Jetson-class hardware. The Streamlit interface can also be extended with video upload, live camera streams, and exportable daily intrusion summaries for beekeepers.

References

1. Bochkovskiy, A., Wang, C.-Y., and Liao, H.-Y. M. (2020). YOLOv4: Optimal Speed and Accuracy of Object Detection. arXiv:2004.10934.
2. Jocher, G., Chaurasia, A., and Qiu, J. (2023). Ultralytics YOLOv8. Ultralytics software documentation and model implementation.
3. Lin, T.-Y., Goyal, P., Girshick, R., He, K., and Dollar, P. (2017). Focal Loss for Dense Object Detection. IEEE ICCV.
4. Paszke, A. et al. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. NeurIPS.
5. Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. CVPR.
6. Ren, S., He, K., Girshick, R., and Sun, J. (2015). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. NeurIPS.
7. Van Rossum, G. and Drake, F. L. (2009). Python 3 Reference Manual. CreateSpace.
8. Zhou, X., Wang, D., and Krahenbuhl, P. (2019). Objects as Points. arXiv:1904.07850.