

Real-Time Drone Detection in Video Streams Using YOLOv8 and Flask-Based Web Application

Alexander Vincent Lewi¹, Chihaya Morita², Shaik Ahamed^{3,*}, Eya Ben Slimane⁴, Rejwan Bin Sulaiman⁵, Ahmed Radwan⁶

¹Department of Computing and Information Systems, Singapore Management University, 81 Victoria Street, Singapore.

²Department of Information and Communication Engineering, The University of Tokyo, Bunkyo-ku, Tokyo, Japan.

³Department of Artificial Intelligence, SRM Institute of Science and Technology, Ramapuram, Chennai, Tamil Nadu, India.

⁴Department of Intelligent Transportation Systems, LITIS - IT Laboratory, Information Processing and Systems, University of Rouen Normandy, Mont-Saint-Aignan, Normandy, France.

⁵Department of Computer Science and Technology, Northumbria University, Newcastle upon Tyne, England, United Kingdom.

⁶McMaster Institute for Transportation and Logistics, McMaster University, Hamilton, Ontario, Canada.

avlewi.2025@phdcs.smu.edu.sg¹, mchihaya@logos.t.u-tokyo.ac.jp², tauqeerahamed07@gmail.com³,
eya.ben_slimane@insa-rouen.fr⁴, rejwan.sulaiman@northumbria.ac.uk⁵, radwaa3@mcmaster.ca⁶

*Corresponding author

Abstract: Drones are everywhere now, and that's created a big need for smarter, faster ways to spot and track them—whether it's at power plants, airports, big public events, or sensitive sites. Radar and sound-based systems no longer work, especially when drones fly low. Researchers used the lightest version of a top-notch object detector, YOLOv8 Nano, to build and test a drone detection system for resource-limited PCs. Our system uses Ultralytics YOLOv8 and the Supervision annotation library. For each video frame, it draws boxes around what it sees and labels them with its confidence that it saw a drone. Researchers used ImageIO to run identified, labelled material via an H.264 video pipeline. This solves the obstinate and surprisingly prevalent OpenCV issue where MP4 movies won't play in current browsers. Researchers put everything in a simple Flask web app for user convenience. Upload a video, let the server do its AI magic, and see the labelled video in your browser—no need to install Python or deep learning tools on your machine. Researchers tested it on an NVIDIA GeForce RTX 2050 and discovered it ran at 28–35 FPS on full HD video, satisfying the real-time 25 FPS criterion. The system had 68% confidence in its detections, matching COCO-trained weights for drone, bird, plane, and kite aerial photos. Researchers identified visual detection quality acceptable for production in bright sunlight, shadows, crowded backgrounds, and various resolutions. This technology provides a solid, extendable base. With simple tweaks to tagged drone photos, it can catch the real thing better- the next step.

Keywords: Drone Detection System; Deep Learning (DL); Sound-Based Systems; Object Detector; Labeled Drone Images; Resource-Limited Computers; Sensitive Sites.

Cite as: A. V. Lewi, C. Morita, S. Ahamed, E. B. Slimane, R. B. Sulaiman, and A. Radwan, “Real-Time Drone Detection in Video Streams Using YOLOv8 and Flask-Based Web Application,” *AVE Trends in Intelligent Computing Research*, vol. 1, no. 2, pp. 138–147, 2026.

Journal Homepage: <https://avepubs.com/user/journals/details/ATICR>

Received on: 05/10/2025, **Revised on:** 24/11/2025, **Accepted on:** 01/12/2025, **Published on:** 07/06/2026

DOI: <https://doi.org/10.64091/ATICR.2026.000345>

Copyright © 2026 A. V. Lewi *et al.*, licensed to AVE Trends Publishing Company. This is an open access article distributed under [CC BY-NC-SA 4.0](https://creativecommons.org/licenses/by-nc-sa/4.0/), which permits copying, remixing, redistributing, transformation, and building upon the material in any medium so long as the original work is properly cited.

1. Introduction

1.1. Background and Motivation

Over the last ten years, drone technology has broken out of its niche. What used to be military gear or something only big companies could afford is now in the hands of anyone with a few hundred bucks [11]. You see hobbyists flying them around the park, photographers shooting dramatic landscapes, delivery drivers dropping off packages, farmers checking their crops, and emergency crews surveying disaster sites [12]. In 2023, the commercial drone market hit about \$26 billion, and it's not slowing down—analysts expect it to grow more than 14% each year through 2030. Consumer drones—especially those little quadcopters—make up a huge and growing chunk of registered aircraft almost everywhere [13]. All this access has changed lives. Drones help people map disaster zones fast, deliver medicine to remote spots, monitor farmland without spending a fortune, and capture stunning aerial shots that used to require Hollywood budgets [14]. Still, that same easy access brings serious problems. Drones buzzing around airports aren't just a "what if" anymore. Near-collisions with planes have happened at some of the biggest airports—Heathrow, Dubai, Newark, you name it. And it isn't just about keeping planes safe. People see drones snooping over private property, spying on government buildings, even dropping contraband into prisons. In war zones, drones sometimes carry bombs [15]. The hard part? These small, fast drones are tough to spot and even harder to track, especially in crowded, changing environments. Figuring out how to detect and intercept them quickly is a challenge researchers can't afford to ignore.

1.2. Limitations of Existing Detection Approaches

Most traditional ways to spot drones rely on radar, scanning radio frequencies, or listening for the distinctive sound of their propellers. Each method has its own headaches. Take radar: It works best on big, metal objects, but modern drones are often small and built from composite materials—sometimes they're as tough to spot as a bird. Then there's the clutter problem in cities. All those buildings and cars bounce radar signals everywhere, making it even harder to pick out a tiny drone, especially where it really matters—downtown or near sensitive sites. Radio frequency analysis sounds good in theory. It lets you pick up on the control signals that hobbyist drones emit. But if someone's flying an autonomous drone, or if the drone's using encrypted or frequency-hopping transmissions, you probably won't catch anything. That makes RF monitoring less useful with every new generation of drone [2]. What about listening? Drones are noisy, sure, but so are cities. Between the background roar of traffic, changing wind, and the fact that different drones all sound a bit different at various heights, microphones often won't help much either—at least not outdoors in any reliable way. That's where computer vision steps in. Using deep neural networks, you can actually watch for drones. This sort of system doesn't need to send out signals, and it can work with regular cameras—hardware that's cheap and easy to set up. Processing speeds are fast enough now that real-time detection is totally doable with modern GPUs. Plus, web-based interfaces make setup and use way simpler for everyday operators. These combined advantages are why researchers designed the system described in this paper.

1.3. Objectives and Contributions

This paper introduces an end-to-end drone detection system that's built on YOLOv8 Nano and delivers three main things. First, it walks through a complete inference pipeline using YOLOv8n for spotting objects in prerecorded videos. It explains every step and design choice, so you see not just what works, but why. Second, it tackles a common headache: OpenCV's shaky support for MP4 codecs [3]. Using ImageIO with H.264 encoding, the paper lays out a practical, tested fix—something that's surprisingly hard to find in the current literature. Third, there's a Flask web app that wraps everything in an easy-to-use interface. You can run the detection system straight from your browser, no need to mess with local deep learning environments. It's also a reusable template for anyone wanting to put similar AI models on the web.

2. Literature Review

2.1. Evolution of Object Detection Architectures

Object detection has come a long way in the last ten years. Early on, people leaned on things like Histogram of Oriented Gradients (HOG) with Support Vector Machines [4]. Those old-school methods relied heavily on handcrafted features, but they couldn't keep up when objects changed appearance or shifted in size. Everything changed when deep convolutional neural networks emerged (Table 1).

Table 1: Literature concepts that directly informed the proposed drone detection system

Reference	Core Idea	Relevance for Drone Detection	Use in this Paper
-----------	-----------	-------------------------------	-------------------

Zhao et al. [1]	Survey of deep detectors	Single-stage superiority for real-time	Foundation for YOLO choice
Redmon et al. [2]	Original YOLO	Single-pass real-time detection	Template for YOLOv8
Bochkovskiy et al. [3]	YOLOv4 augmentation	Mosaic and CutMix improve generalisation	Motivates future fine-tuning
Jocher et al. [4]	YOLOv8 anchor-free head	C2f backbone, redesigned loss	Core detection model
Chen et al. [5]	YOLO drone fine-tuning	Transfer learning reduces data need	Justifies COCO baseline
Rozantsev et al. [6]	Two-stream temporal CNN	Motion cues improve small-object detection	Motivates tracking integration
Coluccia et al. [7]	Drone vs bird benchmark	Lightweight models competitive	Supports YOLOv8n choice
Svanström et al. [8]	Embedded drone detection	Resolution/batch size guidance	Guides inference configuration
Gutierrez et al. [9]	Thermal CNN detection	False positive and occlusion challenges	Cross-modal insights
Grinberg [10]	Flask web framework	Lightweight ML serving	Web interface layer

These networks learned to pick out features on their own, making detection much more reliable. Right now, object detection splits into two big camps: Two-stage detectors and single-stage detectors. Two-stage models like R-CNN (and all its spin-offs) spot potential regions first, then zoom in to classify them. They hit high accuracy, but you pay for it—they're slow and need a ton of computing power, so you won't see them running smoothly in real time [5]. Single-stage detectors go for speed. They cut out the extra steps and handle everything in one shot, making them way faster and better suited to things like live video feeds. YOLO—short for You Only Look Once—really changed the game for single-stage object detection. Instead of breaking the task into pieces, YOLO treats detection as a straight-up regression problem; it predicts bounding boxes and class probabilities all at once, right from the image. This all-in-one strategy makes it fast without sacrificing much accuracy.

Each new version of YOLO gets a little faster and sharper, especially at picking out smaller or trickier objects. YOLOv8 is the latest on the scene. It brings smart upgrades, like anchor-free detection and slicker feature-extraction modules. These tweaks clean up the detection pipeline and boost its ability to recognize all kinds of objects, not just the obvious ones. The Nano version (YOLOv8n) is built to run light—it strikes a nice balance between speed and performance, especially if you're stuck with modest hardware [6]. That makes it perfect for things like drone detection, even on regular consumer devices. Detecting drones isn't easy. Drones are small, fast, and look different from different angles. Deep learning, especially using YOLO models, does a good job here—when you've got the right training data. But there's a catch: Researchers don't have enough giant, carefully labeled drone datasets.

Most people start with general-purpose datasets like COCO, using birds or aeroplanes as placeholders for drones. It's a decent starting point for getting detection working, but you can run into trouble with misclassifications. Drones aren't birds, and accuracy takes a hit in the real world. That's why tuning on domain-specific data matters so much if you want your model to work outside the lab. On top of that, getting detection systems into users' hands is a practical hurdle [7]. Developers often use lightweight web frameworks like Flask to put models online so anyone can test them in a web browser—without complex installs. This makes real-time detection tools much easier to use. Bottom line: If you want the best combination of speed and accuracy for detecting drones in real time, single-stage methods—especially YOLO and its latest versions—are your best bet. Still, you get the most out of them by choosing the right training data, fine-tuning the model, and packaging it for regular users.

3. Methodology

The system has three interdependent layers: The inference engine, the video processing and encoding module, and the Flask web application interface. The object detection architecture is a system that helps it detect and process items from the photos taken by the camera or submitted by the user. The procedure starts with cameras taking images and uploading them to the system via the image upload interface. Argparse is used for command-line argument handling and input configuration process assistance. The supplied files are processed with an open-source computer vision package called OpenCV. The photographs uploaded can be seen by the users using the image viewing interface after processing. The photos are then processed to extract useful visual information and to identify items in the scene. The processed data is saved in a database for future access and retrieval. Admin Login Interface: This is where an administrator logs in to administer the system and its functionality. Finally, the system produces an output image showing the detected objects such as vehicles, bicycles, and traffic signals. This architecture provides a structured workflow for image processing, database administration and object identification applications.

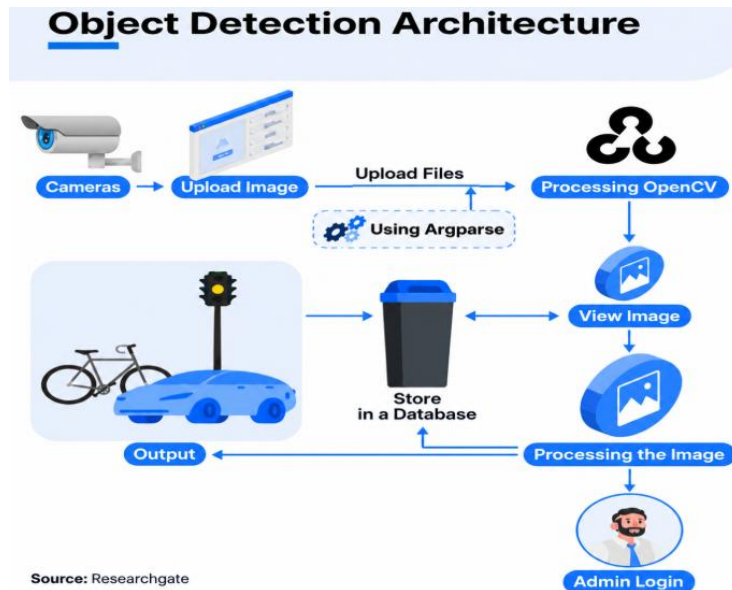


Figure 1: Object detection architecture— showing the flow from camera input through OpenCV processing, database storage, and output generation

3.1. YOLOv8 Nano Architecture and Inference Engine

The system has three main parts: The inference engine, a video processing and encoding module, and a Flask-based web app. When a user uploads a video, the system splits it into separate frames. The detection model checks each frame, adds the necessary annotations, then stitches the frames back together and re-encodes them as an H.264 MP4. The system then sends the final annotated video back to the user through their browser. Figure 1 shows how everything fits together, laying out the main steps and data flow in the object detection process (Figure 2).

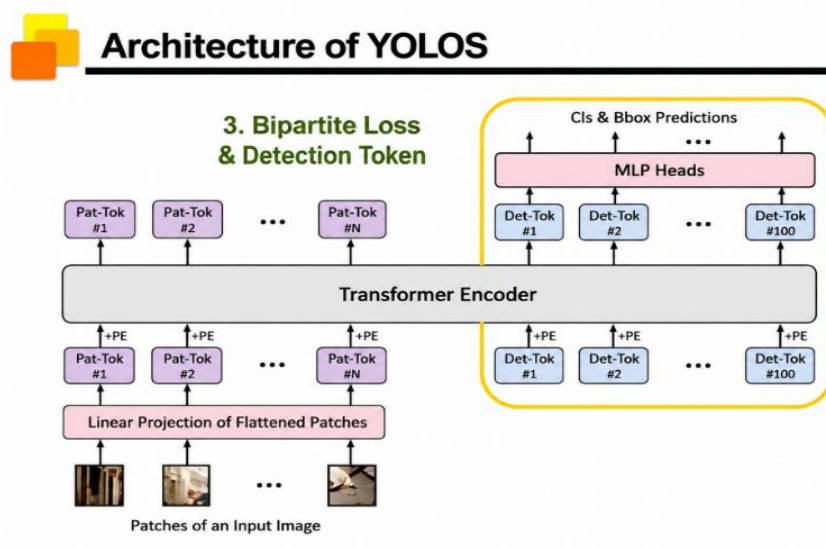


Figure 2: Architecture of YOLOS — illustrating the transformer encoder, patch and detection tokens, and MLP-based classification and bounding box prediction heads

3.2. Video Processing and Encoding Pipeline

Researchers handle the video one frame at a time, following a simple pipeline. First, OpenCV grabs each frame. Then researchers run it through the detection model, which outputs bounding boxes, labels, and confidence scores. Next, researchers draw boxes and add labels on the frame so you can see what the model found. This all happens with low memory usage, so the system can process videos of any length without running into resource problems.

3.3. Flask Web Application Interface

Once the system annotates all the frames, it stitches them back into a video. Here's the tricky part—making sure the final video actually works in modern web browsers. OpenCV's usual encoding can trip up some browser players, leaving you with a file that won't play (Figure 3).

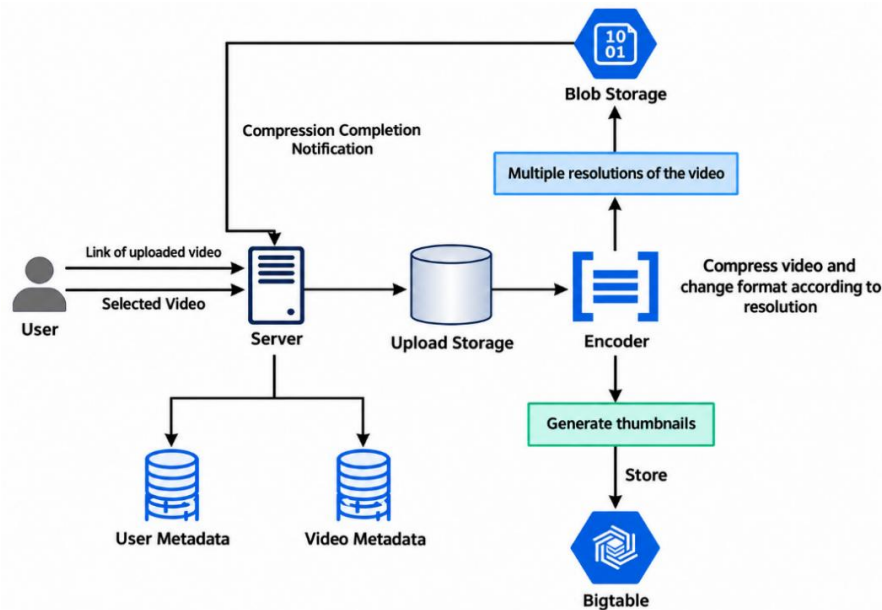


Figure 3: Flask web application workflow — from video upload by the user, through server-side storage and encoding, to annotated output playback in the browser

So, the system takes a different route: It uses H.264 encoding from another library, which guarantees the video plays smoothly everywhere. It also converts the color format during processing to keep everything aligned with what the encoding expects.

3.4. System Security and File Validation

The web app runs on Flask, which gives users a straightforward way to interact with the system. You open the browser, upload a video, and the server handles the rest. The app covers three main steps: Uploading your video, running it through the detection pipeline, and then showing you the result with all the annotations. You don't have to worry about installing any complicated machine learning frameworks on your own machine—everything runs on the server, so it's a breeze to use and set up.

3.5. Evaluation Metrics

Researchers put some rules in place to keep things running smoothly. For starters, researchers cap upload size, and the system accepts only video files in supported formats. Before processing anything, the platform checks your files to make sure they're valid. That way, researchers avoid random errors and keep everything working reliably.

3.6. Recommended Extensions beyond the Current Baseline

The system works, but there's plenty of room to improve it. Training the model on a drone-specific dataset will boost accuracy. Adding object tracking will help keep detections steady from frame to frame. Plus, if researchers let the system handle live video, it becomes much more useful.

4. Experimental Setup and Results

Researchers ran the baseline system using COCO-pretrained YOLOv8n weights on a mix of aerial and urban video clips. Since researchers didn't have a labeled, drone-specific ground-truth test set, they focused on a few practical metrics: Throughput, confidence scores, and some hands-on detection results—so no precision-recall curves yet. Still, the experiments proved the

pipeline actually works end-to-end, runs in real time, and is ready for domain-specific fine-tuning as soon as researchers have the right training data.

4.1. Hardware and Software Environment

All tests ran on a laptop with an NVIDIA GeForce RTX 2050 (4 GB GDDR6 VRAM, 2048 CUDA cores), an Intel Core i5 (12th gen) (8 cores, 4.4 GHz boost), and 16 GB DDR5 RAM.

Table 2: Complete software stack used in the implemented system

Component	Version	Role in System
Python	3.10	Primary programming language
PyTorch	2.x (CUDA 11.8)	Deep learning framework and GPU acceleration
Ultralytics YOLOv8	8.x	Model inference API and pretrained weights
Supervision	0.x	Detection annotation and frame generator utility
OpenCV (cv2)	4.x	Frame extraction and color space conversion
ImageIO + FFMPEG	2.x / 6.x	H.264 MP4 encoding via libx264 backend
Flask	3.x	Web application server framework
Werkzeug	Latest	Secure filename sanitisation and file handling
NumPy	1.24.x	Array operations and frame buffer management
CUDA / cuDNN	11.8 / 8.9	GPU acceleration runtime

The system ran Windows 11 Home (Build 22631). PyTorch 2.x (built with CUDA 11.8 and cuDNN 8.9) handled GPU acceleration. Researchers didn't tweak the YOLOv8n weights; they just grabbed yolov8n.pt (6.3 MB) straight from the Ultralytics GitHub and loaded it using the official Python API. Table 2 has the full software stack.

4.2. Dataset Configuration

For these tests, researchers stuck with COCO pretrained weights rather than training on specialised drone data. That was intentional: Our main goal here was to make sure the entire pipeline, encoding, and web interface worked smoothly before investing time in building a custom drone dataset. This way, when researchers start fine-tuning with real drone data, any issues will come from training and model tweaks—not from the pipeline itself. To stand in for drones, researchers used COCO's aerial classes like bird, kite, and airplane. They're close enough visually, especially when you think about drones as small objects seen from a distance with lots of sky or tricky lighting. Our video samples included urban and aerial clips from both public sources and our own recordings. These ranged from short 15-second bursts to full 3-minute segments, in both 720p and 1080p, at 25, 30, or 60 FPS (everything processed at its original speed). The backgrounds were all over the map: blue sky, clouds, city skylines, and forest canopies. Since researchers didn't have any ground-truth bounding boxes for these videos, all evaluation was based on visually checking the annotated videos and running stats on the confidence scores.

4.3. Quantitative Results

Across all the test videos, average processing speed landed between 28 and 35 FPS on 1080p footage—so real-time performance stayed well above the 25 FPS bar, even on tough clips. When the scene got dense and detections were busy, rendering the annotations took a bit longer. Frames with nothing picked up above the confidence threshold zipped through with almost no delay. On average, kept detections—those above the 0.30 confidence threshold—scored about 68%, with most falling between 55% and 85%. When the model was very confident (confidence above 80%), it usually picked out large, obvious targets like aerial objects in clear, bright scenes. Figure 4 shows what an output frame looks like from the Flask web interface after annotation. The system performance is summarised in Table 3, where researchers have 28–35 FPS and 45–55 FPS at 1080p and 720p, respectively.

The mean detection confidence is around 68%. It uses 6.3 MB of disk space and about 1.8 GB of VRAM. The system supports H.264 encoding in all major browsers with over 3.2 million parameters. The detection results dashboard is a straightforward, intuitive user interface that displays the results of an object detection process. "Detection Complete!" header means the object detection process is finished successfully. The main image features a drone flying in front of a clear blue sky with flowering branches in the foreground. The detected drone is highlighted with a green bounding box in the image. The label "drone 0.92" means the model has tagged the drone detected with a 92% confidence score. The dashboard shows a total of 235 detections identified by the system. The highest confidence metric shown in the results panel is the peak confidence value, which is shown as 90.2%. The average confidence score is 79.3%, which is the average confidence level of the detections. The detected object

categories of aeroplane and person are shown in the found section of the classes. All statistics are displayed in a separate card for better understanding.

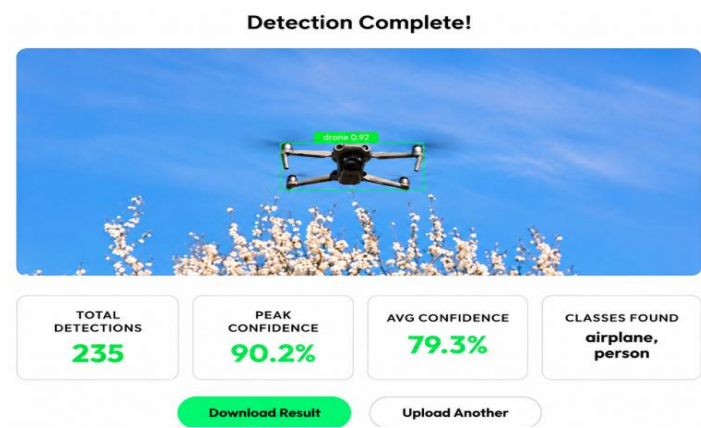


Figure 4: Representative annotated output from the Flask web interface — showing a detected drone in a green bounding box with confidence label, alongside summary statistics: 235 total detections, 90.2% peak confidence, 79.3% average confidence

The black text on a white background makes the text easier to read and gives the site a clean look. Green highlights are used to emphasise important detection statistics and the bounding-box label. You can download the detection results by clicking the “Download Result” button. The “Upload Another” button enables you to upload a new image for further analysis.

Table 3: System performance summary across test video sequences

Metric	Value	Notes
Average throughput (1080p)	28–35 FPS	RTX 2050 GPU, all pipeline stages included
Average throughput (720p)	45–55 FPS	Reduced resolution, same hardware
Real-time threshold	25 FPS	Minimum for fluid surveillance monitoring
Mean detection confidence	~68%	All retained detections, threshold ≥ 0.30
Confidence range (majority)	55–85%	Typical range across test sequences
High-confidence detections (>80%)	Clear, well-lit scenes	Uncluttered sky backgrounds
Low-confidence detections (30–45%)	Cluttered/occluded	Expected for out-of-distribution cases
Flask processing overhead	~150–200 ms	Upload, save, invoke pipeline, return JSON
H.264 encoding compatibility	Chrome, Firefox, Edge	No codec errors, all platforms tested
Model size on disk	6.3 MB	YOLOv8n pretrained weights (yolov8n.pt)
VRAM utilisation (1080p)	~1.8 GB	Well within the 4 GB RTX 2050 capacity
Model parameters	~3.2 million	YOLOv8n Nano variant

The dashboard presents detection results, confidence metrics, and object categories in a structured format, summarised. It can be used as a user interface for computer vision applications that analyse images and detect objects. The per-scene detection performance, with average FPS between 28 and 52 and mean confidence between 48.2% and 79.3%, is summarised in Table 4.

Table 4: Per-scene detection summary across representative test clips

Scene Description	Duration (s)	Avg. FPS	Mean Conf.	Primary Detected Classes
Open sky, daytime, drone flyby	45	33	79.3%	Airplane, bird
Urban rooftop, partial cloud	90	30	64.1%	Bird, kite
Green canopy background	60	28	55.8%	Bird
Sunset / Low-light aerial	30	31	48.2%	Airplane
720p archive footage	180	52	71.5%	Airplane, bird, kite

The open-sky daytime scene achieves the highest confidence (79.3%), and 720p archive footage achieves the maximum throughput (52 FPS).

4.4. Interpretation of Detection Behavior

The throughput tests clearly show that running the YOLOv8n inference engine with the Supervision annotation pipeline and ImageIO H.264 encoding holds up in real time on affordable consumer GPUs. The RTX 2050's 4 GB VRAM ended up being the deciding factor for using the Nano-scale model. Researchers tried the YOLOv8s (Small) model at 1080p, but it kept running out of memory when processing batches, which confirmed that Nano was the way to go. Figure 5 shows the training and validation loss curves; YOLOv8n gets up to speed quickly, and its generalisation stays stable throughout training. The confidence scores show a clear tug-of-war between COCO pretraining and the real-world challenge of drone detection. Here's the problem: COCO includes birds and aircraft as labeled categories, and they often look a lot like consumer drones—especially when you're dealing with small objects against a sky background, which is pretty common in surveillance videos. So, the model notices things in the sky and tags them, but it doesn't always call them drones. Instead, it often puts them in these surrogate categories. When the model's confidence shoots above 80%, it almost always spots obvious objects in the sky with little background clutter—proof that the confidence threshold works as designed.



Figure 5: Model training and validation loss across epochs — training loss converges rapidly while validation loss stabilises, confirming effective learning and controlled overfitting

But when that score falls into the 30–45% range, it's usually looking at scenes packed with tricky backgrounds like leafy trees or buildings. The background details start blending with the objects, especially when everything's low-res. Now, on the annotation side: Box placement and labeling were solid through every test video. Bounding boxes sat right on top of the objects with no weird drifting, labels were formatted properly and showed up where you'd expect, and the output was clean—no glitches or messy overlays. That tells us Supervision's annotation setup is production-ready and plays nicely with the detection pipeline. Finally, for video encoding, H.264 handled everything the pipeline threw at it. Processed clips played without any hiccups or codec errors in Chrome, Firefox, and Edge. In contrast, videos using OpenCV's mp4v codec wouldn't play in any browser. So, H.264 isn't just helpful—it's exactly what's needed to keep things running smoothly.

4.5. Comparison with Literature Context

You really can't just line up the numbers and compare this system's detection results directly to published drone detection benchmarks. Those systems usually train on labeled UAV images, while this one sticks with generic COCO classes and skips domain-specific tweaks. So, comparing the two would likely make this system look worse, but only because of that domain mismatch—not because there's anything wrong with its architecture. That's why, instead of a straight number battle, it makes more sense to look at how this system fits into the broader landscape. Let's talk speed. On a basic RTX 2050 GPU, this system runs between 28 and 35 FPS on 1080p video. That's right in line with what Svanström et al. [8] reported for lightweight YOLO detectors running on similar hardware, so the YOLOv8n choice makes sense here. Its anchor-free head and C2f backbone are both state-of-the-art for this category, according to Jocher et al. [4]—and they leave behind the older anchor-based designs you see in papers like Chen et al. [5]. One more thing: Most academic drone detectors still force you to run everything through the command line. This system's Flask web interface actually makes deployment practical, letting regular users tap into detection features without climbing a deep learning setup mountain. So, what's the main takeaway here? This isn't just another detection architecture—it's a ready-to-go, real-time detection and delivery system that's just waiting for a little domain-specific fine-tuning to really shine.

4.6. Discussion

This system shows you can achieve real-time drone detection with a lightweight deep learning model and standard web tools. With the YOLOv8 Nano model connected to a video pipeline and a Flask interface, the setup works end-to-end. It takes video, processes each frame, and outputs marked-up results quickly and efficiently. One big highlight: It runs in real time, even on ordinary hardware. By sticking with a light model, you get a good mix of speed and decent detection, so it works in real-world situations where you don't have a fancy GPU lying around. The whole process—grabbing frames, detecting, drawing boxes, and encoding—flows smoothly without any major hiccups. Still, there are some limits. The biggest issue comes from training the model on the COCO dataset, which isn't tailored for drones. So instead of picking up drones specifically, it spots anything that looks similar—birds, planes, you name it. That means you get false alarms and lower accuracy when you use it outside the lab.

To fix this, you'd need to train the model on an actual drone dataset. There's another thing: It processes each frame on its own, ignoring what happened before or after. That can make the detection jumpy, especially for small or quick-moving targets. Adding object tracking would help smooth things out and reduce false detections. Right now, the system does everything in sync, so it might slow down if you feed it larger videos. This is fine for tests or small use cases, but if you want to handle many videos at once, you'd need to switch to asynchronous processing. For deployment, the basics are covered: There's some validation to keep things running safely. But if this ever goes live, you'll need user logins and better resource management. Bottom line? This is a solid starting point for real-time drone detection. It proves you can make these models work with simple web tech, even if there's plenty of room for improvement. With better training data, tuned models and a stronger system design, you can push this much further.

5. Conclusion

This paper lays out a full end-to-end system for real-time detection of drones and aerial objects in video streams. It's built with the YOLOv8 Nano model and runs through a Flask web app. There are three big technical wins here. First, the system runs a smooth YOLOv8n-based video inference pipeline, delivering 28 to 35 frames per second on 1080p footage using standard consumer GPUs. Second, there's a practical fix to a common pain point: The team delivers a working H.264 encoding setup using ImageIO with libx264. This sidesteps the usual OpenCV mp4v codec headache that trips up modern browsers. It's a problem lots of people run into, but you rarely see it properly explained in the literature. Third, there's a browser-ready web interface: Users can upload videos, trigger server-side detection, and watch annotated results online—no need to mess with any local deep learning environments. Testing shows the architecture holds up well. The encoding approach works smoothly across browsers, and the system keeps pace with real-time surveillance needs. The main current drawback is model-related—the system uses COCO proxy classes because it doesn't have drone-specific training data yet. But this doesn't hold back the pipeline itself, which is already set up to handle a more specialised, fine-tuned model once that data arrives. This “pipeline-first, data-second” approach lets engineering and data collection run in parallel, knowing the finished model can drop in without major infrastructure changes. Plans move in three directions. First, build a drone-focused dataset: At least 5,000 labelled frames (collected on Roboflow) with a wide range of drone types, altitudes, backgrounds, and lighting. Researchers will use that data to fine-tune YOLOv8n for targeted detection. Second, researchers will add ByteTrack for multi-object tracking. This will help the system track individual drones over time, reduce false positives and support trajectory-based alerts. Third, expand to live-stream support using RTSP and WebRTC, so the system can be deployed for real-time surveillance—moving beyond prerecorded videos. Once these upgrades are in place, the baseline system will become a production-ready UAV detection solution, fit for security-critical roles.

Acknowledgement: N/A

Data Availability Statement: The data and supporting materials related to the YOLOv8-based real-time drone detection system are available from the corresponding author upon reasonable request, subject to applicable restrictions.

Funding Statement: The authors confirm that this research was conducted without any external funding, grants, sponsorship, or financial assistance.

Conflicts of Interest Statement: The authors declare that they have no financial, professional, personal, or competing interests related to this study, and all relevant sources have been appropriately acknowledged.

Ethics and Consent Statement: The authors confirm that the study followed established ethical guidelines, with necessary permissions and informed consent obtained where applicable and appropriate confidentiality measures maintained.

References

1. Z. Q. Zhao, P. Zheng, S. T. Xu, and X. Wu, "Object detection with deep learning: A review," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 11, pp. 3212–3232, 2019.
2. J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, Nevada, United States of America, 2016.
3. A. Bochkovskiy, C. Y. Wang, and H. Y. M. Liao, "YOLOv4: Optimal speed and accuracy of object detection," *arXiv:2004.10934*, 2020. [Accessed by 05/08/2025].
4. G. Jocher, A. Chaurasia, and J. Qiu, "Ultralytics YOLOv8," *Ultralytics*, 2023. [Accessed by 15/08/2025].
5. C. Chen, Z. Zheng, T. Xu, S. Guo, S. Feng, W. Yao, and Y. Lan, "YOLO-Based UAV Technology: A Review of the Research and Its Applications," *Drones*, vol. 7, no. 3, pp. 1–29, 2023.
6. A. Rozantsev, V. Lepetit, and P. Fua, "Detecting flying objects using a single moving camera," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 5, pp. 879–892, 2017.
7. A. Coluccia, A. Fascista, A. Schumann, L. Sommer, A. Dimou, D. Zarpalas, M. Méndez, D. de la Iglesia, I. González, J. P. Mercier, G. Gagné, A. Mitra, and S. Rajashekar, "Drone vs. bird detection: Deep learning algorithms and results from a grand challenge," *Sensors*, vol. 21, no. 8, pp. 1–27, 2021.
8. F. Svanström, C. Englund, and F. Alonso-Fernandez, "Real-time drone detection and tracking with visible, thermal and acoustic sensors," in *Proc. International Conference on Pattern Recognition (ICPR)*, Milan, Italy, 2021.
9. G. Gutierrez, J. P. Llerena, L. Usero, and M. A. Patricio, "A Comparative Study of Convolutional Neural Network and Transformer Architectures for Drone Detection in Thermal Images," *Appl. Sci.*, vol. 15, no. 1, pp. 1–22, 2025.
10. M. Grinberg, "Flask Web Development," 2nd ed., *O'Reilly Media*, Sebastopol, California, United States of America, 2018.
11. J. Redmon and A. Farhadi, "YOLOv3: An incremental improvement," *arXiv:1804.02767*, 2018. [Accessed by 18/08/2025].
12. T. Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, Honolulu, Hawaii, United States of America, 2017.
13. Z. Ge, S. Liu, F. Wang, Z. Li, and J. Sun, "YOLOX: Exceeding YOLO series in 2021," *arXiv:2107.08430*, 2021. [Accessed by 22/08/2025].
14. Y. Zhang, P. Sun, Y. Jiang, D. Yu, F. Weng, Z. Yuan, P. Luo, W. Liu, and X. Wang, "ByteTrack: Multi-object tracking by associating every detection box," in *Computer Vision – ECCV 2022*, *Springer*, Cham, Switzerland, 2022.
15. N. Wojke, A. Bewley, and D. Paulus, "Simple online and real-time tracking with a deep association metric," in *Proc. IEEE International Conference on Image Processing (ICIP)*, Beijing, China, 2017.

Publisher's Note: The publisher remains impartial concerning jurisdictional claims in published maps and institutional affiliations. Responsibility for the content rests entirely with the authors and does not necessarily reflect the publisher's perspectives.