International Journal of Engineering Research and Science & Technology

ISSN 2319-5991 www.ijerst.com
© 2026 IJERST. All Rights Reserved
Peer Reviewed, Referred & Indexed Journal

Research Paper

REAL TIME OBJECT DETECTION SYSTEM

Dusanapudi Bhanu Yaswanth
PG MCA Scholar, Department of Computer Science
Sir C R Reddy College, Eluru, Andhra Pradesh, India - 534005
bhanudusanapudi2002@gmail.com

Banisetti Jyothi Satya Bhavani
Professor, Department of Computer Science
Sir C R Reddy College, Eluru, Andhra Pradesh, India - 534005
jyothiganesh006@gmail.com

Musunuru Ratnakar
HOD, Department of Computer Science
Sir C R Reddy College, Eluru, Andhra Pradesh, India - 534005

Abstract*— This paper presents a modular Real Time Object Detection System that combines Ultralytics YOLOv8, OpenCV, a reusable Python inference class, and a FastAPI web service. The system accepts webcam frames, static images, or browser uploads; predicts multiple object classes; applies confidence filtering and non-maximum suppression; and returns colour-coded bounding boxes, labels, confidence scores, object counts, and real-time throughput. In addition to the 80 categories available in the pretrained COCO model, the project provides a custom-class pipeline that collects images through DuckDuckGo search, removes low-resolution and duplicate samples, and automatically produces YOLO annotations. Known categories are localized by the pretrained detector, while genuinely new categories are verified with CLIP before weak bounding-box assignment. YOLOv8n is then fine-tuned through transfer learning and evaluated on a held-out split. The implemented CLI and FastAPI interfaces share the same ObjectDetector core, preventing duplicated inference logic. Documented results include detection confidence above 85% for clearly visible objects, 45-60 FPS on an NVIDIA GPU, and 8-12 FPS on CPU-only execution. The resulting architecture offers a practical foundation for surveillance, retail analytics, inventory recognition, and assistive computer-vision applications.

Keywords*— YOLOv8, object detection, real-time inference, OpenCV, FastAPI, CLIP, transfer learning, custom-object training.

I. INTRODUCTION

Object detection requires a system to identify the category and spatial location of every relevant object in an image or video frame. Earlier pipelines based on manually designed edges, corners, colour histograms, and sliding-window classifiers are sensitive to scale, illumination, viewpoint, partial occlusion, and background clutter. Deep convolutional detectors instead learn hierarchical visual representations directly from annotated data and have become the dominant solution for surveillance, autonomous systems, retail analytics, robotics, and assistive technology [1]-[10].

A. Problem Context and Scope

The project addresses two connected requirements. The first is low-latency multi-object detection from webcam, image, and web-upload sources. The second is extensibility beyond the fixed vocabulary of a pretrained detector. YOLOv8 trained on COCO recognizes 80 general-purpose categories, but domain-specific targets such as earphones, earbuds, and TWS buds require new data and additional training. Manual bounding-box annotation is expensive, so the project treats data collection and labeling as part of the complete system rather than as an external prerequisite.

B. Objectives and Contributions

The implemented contribution is an end-to-end pipeline that connects data acquisition, hybrid automatic labeling, YOLO-format dataset generation, transfer-learning-based fine-tuning, validation, and deployment. A shared ObjectDetector class is consumed by both a command-line interface and a FastAPI service. This design provides a single location for model loading, threshold control, inference, box rendering, object counting, and FPS measurement. The project also exposes /health, /detect/image, and /detect/webcam endpoints and supports interactive webcam hot-keys for snapshots and confidence adjustment.

II. RELATED WORK

A. Two-Stage and Single-Stage Detection

R-CNN established the use of deep features for region-based detection, followed by SPPnet and Fast R-CNN, which reduced repeated convolution. Faster R-CNN introduced a learnable region-proposal network and became a strong accuracy-oriented baseline [1]-[4]. Single-stage methods removed the proposal-classification separation. SSD predicted boxes at multiple feature-map resolutions, while YOLO formulated detection as direct regression from the complete image [5]-[7]. Feature pyramids, focal loss, and instance-level extensions improved multi-scale recognition and class imbalance handling [8]-[10].

B. Efficient, Transformer, and Open-Vocabulary Detectors

YOLOv3, YOLOv4, YOLOX, YOLOv6, YOLOv7, YOLOv9, and YOLOv10 progressively refined backbone design, label assignment, augmentation, anchor usage, and end-to-end inference [11]-[17]. Anchor-free detectors such as FCOS and CenterNet simplified box prediction, while EfficientDet optimized multi-scale fusion [18]-[20]. DETR and its successors replaced hand-designed matching components with transformer-based set prediction, and RT-DETR demonstrated that transformer detectors can also operate in real time [21]-[24]. CLIP, Grounding DINO, and YOLO-World broadened recognition through vision-language supervision and open-vocabulary prompting [25]-[27].

C. Research Gap

Many published detectors assume a fixed and manually annotated dataset. The present project focuses on the engineering gap between a pretrained detector and a

deployable custom-class application. It combines web-image collection, YOLO-based localization for known classes, CLIP verification for unseen classes, transfer learning, and a shared CLI/API inference layer. This integration is particularly useful for academic and small-team deployments that cannot maintain a separate annotation workforce.

III. MATERIALS AND METHODS

A. Data Collection

The project uses the pretrained COCO label space as the starting point and adds three custom concepts: earphones, earbuds, and TWS buds. The scraper.py module queries DuckDuckGo image search for each class, downloads candidate images, rejects files below a minimum resolution, removes content duplicates through hashing, and stores the remaining images by class. The implementation command targets approximately 200 candidate images per custom class, although the final report does not state the exact retained count after filtering.

B. Auto-Labeling and Dataset Preparation

The AutoLabeler first builds a combined vocabulary of 80 COCO categories and three custom categories. If a class is already represented by the detector or a configured alias, YOLOv8 inference supplies localized boxes. For a genuinely new category, CLIP ViT-B/32 evaluates image-text similarity; images above the documented 0.40 score threshold receive a normalized whole-image box. Labels are written as class_id, center-x, center-y, width, and height in YOLO format, and dataset. yaml defines training and validation paths.

Dataset element	Project specification
Pretrained source	COCO label space with 80 object categories
Custom concepts	Earphones, earbuds, and TWS buds
Total vocabulary	83 classes after combining pretrained and custom labels
Collection method	DuckDuckGo image search with resolution filtering and hash de-duplication
Annotation method	YOLO boxes for known classes; CLIP-verified weak boxes for new classes
Storage format	YOLO text labels with an automatically generated dataset.yaml

TABLE I. DATASET AND ANNOTATION CONFIGURATION

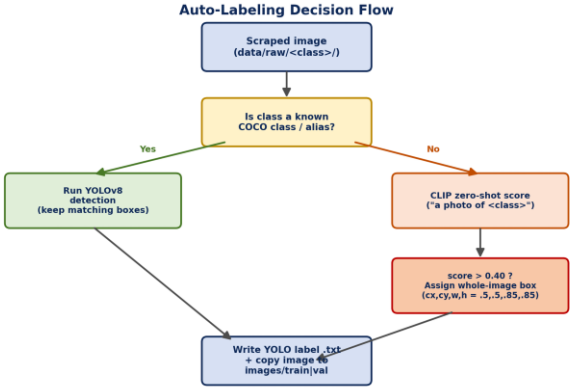


Fig. 1. Hybrid YOLO/CLIP auto-labeling decision flow.

C. Model Training and Validation

The CustomTrainer fine-tunes a pretrained YOLOv8n checkpoint rather than learning from random initialization. The documented training configuration uses AdamW, cosine learning-rate decay, early stopping with patience of 20 epochs, and augmentation that includes HSV jitter, rotation, translation, scaling, shear, horizontal flipping, mosaic, mixup, and copy-paste. The example implementation invokes 50 epochs with batch size 16. The best checkpoint is saved under the training run and is loaded directly by the inference class.

Validation is performed on a held-out split with the Ultralytics model.val() API. The evaluation script prints mAP@0.5, mAP@0.5:0.95, mean precision, and mean recall. The project report confirms that these values were generated but does not list their numerical outputs; therefore, the present paper does not fabricate project-specific mAP or precision/recall values.

D. System Architecture

The architecture separates a custom-training path from a shared real-time inference path. Webcam frames, static files, and web uploads enter detector.py, where a single ObjectDetector instance loads either the stock YOLOv8n checkpoint or the fine-tuned best.pt model. Confidence filtering and non-maximum suppression produce final detections. The CLI in main.py renders a local OpenCV window, while server.py provides REST and streaming access to a browser interface. Both deployment modes use identical model and drawing logic.

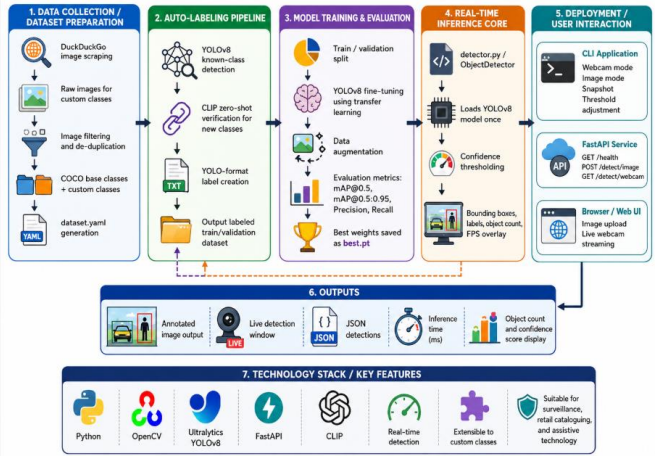


Fig. 2. System Architecture

E. Data Preprocessing and Organization

The collected image pool is prepared in a sequence that protects the training process from avoidable noise. Files that cannot be decoded, images below the configured minimum resolution, and repeated content hashes are removed before labeling. Accepted samples are grouped under class-specific directories so that every generated annotation can be traced to its source class. The auto-labeler writes one text file per image in YOLO format, where the class identifier and the center coordinates, width, and height of each box are normalized by image dimensions. This normalized representation is independent of the original resolution and is therefore suitable for multi-scale training.

The generated dataset.yaml stores the training and validation paths, the number of classes, and the ordered class-name list. Maintaining a single class order is essential because a label index has meaning only relative to this file. The project includes a label-repair utility for cases in which the custom

identifiers 80, 81, and 82 must be remapped for a custom-only configuration. Before training, the labeled data are divided into disjoint training and validation subsets. The validation subset is not used for gradient updates; it is reserved for measuring generalization through precision, recall, and mean average precision.

F. YOLOv8 Inference and Post-Processing

For each input frame, the detector converts the image into the representation expected by the Ultralytics model and performs a single forward pass. The network returns candidate boxes, objectness information, class probabilities, and class identifiers. Predictions below the selected confidence threshold are removed. Non-Maximum Suppression then ranks remaining predictions by confidence and suppresses lower-scoring boxes that overlap a stronger box beyond the IoU threshold. This step prevents one physical object from being counted multiple times and is especially important when nearby feature-map locations predict similar boxes.

The surviving detections are converted back to pixel coordinates and passed to the rendering functions. A reproducible class-dependent colour palette differentiates categories, while a contrasting text colour keeps labels readable over bright and dark regions. Each label contains the class name and confidence percentage. The frame also receives an object counter and, in webcam mode, a rolling FPS value. Because rendering is separated from model loading and prediction, the same annotated output format is produced by both the desktop application and the API service.

G. CLI and FastAPI Deployment

The command-line interface supports webcam and single-image modes. Webcam execution maintains a continuous capture-predict-render loop and accepts Q or ESC to stop, S to save an annotated snapshot, and plus or minus keys to change the confidence threshold during operation. Image mode reads one file, reports inference time, displays the result, and optionally saves the annotated image. These controls make the CLI suitable for local demonstrations and for tuning the threshold under different scene conditions without restarting the program.

The FastAPI layer exposes GET /health, POST /detect/image, and GET /detect/webcam. The health response identifies the loaded model and available class count. The image endpoint validates an uploaded file, decodes it with OpenCV, performs detection, and returns a response containing the annotated image, structured label-confidence-box records, total object count, original dimensions, and inference time. The webcam endpoint streams annotated frames from the server camera through Server-Sent Events. CORS support and a mounted browser front end allow the same inference service to be consumed without duplicating model code.

H. Training Protocol and Reproducibility

The documented training example begins with pretrained yolov8n weights, 50 epochs, batch size 16, and transfer learning on the generated dataset. AdamW optimization and cosine learning-rate decay provide stable updates, while early stopping with a patience of 20 epochs prevents unnecessary continuation when validation quality no longer improves. Mosaic, mixup, copy-paste, geometric transforms, and HSV variation enlarge the effective visual diversity of the relatively small scraped dataset. The best validation checkpoint, rather than the final epoch by default, is saved for evaluation and deployment.

Reproducibility depends on preserving the dataset.yaml file, class order, train-validation split, model name, image size, confidence and IoU thresholds, epoch count, batch size, optimizer configuration, software versions, and hardware description. The project already separates collection, labeling, training, evaluation, and inference into dedicated modules, which makes each stage repeatable. A stronger experimental record should additionally save random seeds, per-epoch loss curves, class-wise AP values, and the exact count of retained images after filtering.

E. Mathematical Formulation

A predicted box is matched to a ground-truth box through Intersection over Union (IoU). For a selected confidence and IoU threshold, true positives, false positives, and false negatives determine precision and recall. F1-score summarizes their balance, average precision integrates the precision-recall curve, and mAP averages AP over classes. Runtime throughput is measured in frames per second.

$$IoU = \frac{Area(Bp \cap Bg)}{Area(Bp \cup Bg)} \quad (1)$$

$$Precision = \frac{TP}{TP + FP} \quad (2)$$

$$Recall = \frac{TP}{TP + FN} \quad (3)$$

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall} \quad (4)$$

$$AP = \int_0^1 p(r)\, dr \quad (5)$$

$$mAP = (1/N)\, \Sigma\, APc \quad (6)$$

$$FPS = \frac{Number\ of\ processed\ frames}{Elapsed\ time\ (s)} \quad (7)$$

IV. RESULTS AND DISCUSSION

A. Detection and Interface Outputs

The CLI webcam mode produced annotated frames containing a class label, confidence percentage, and bounding box for each detected object, together with the live FPS and object-count overlays. The supplied result image shows a cell phone detected at 65% confidence while the pipeline operates at 15.3 FPS. The browser interface provides image upload and live-camera tabs and sends input to the FastAPI backend. The API returns an annotated JPEG, a structured list of labels, confidences, and box coordinates, the object count, image dimensions, and inference time.

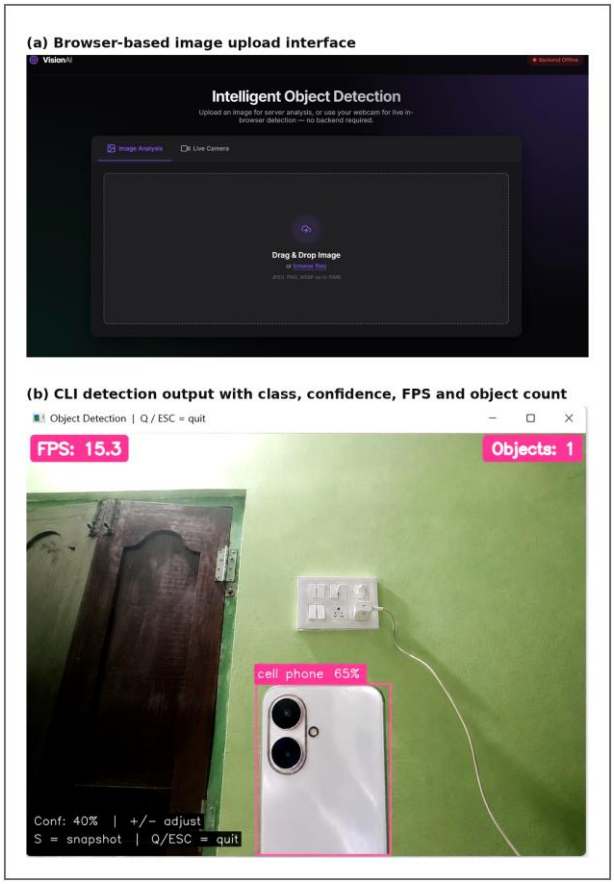


Fig. 3. Important outputs: browser upload interface and CLI object-detection result.

B. Runtime Performance

For clearly visible and well-lit COCO objects, the report records average detection confidence above 85%. YOLOv8n achieved 45-60 FPS with NVIDIA GPU acceleration and approximately 8-12 FPS on CPU-only hardware. The FastAPI single-image endpoint completed inference in the tens of milliseconds on GPU. These results support interactive operation and confirm the expected speed advantage of GPU inference. The chart uses the midpoint of each reported FPS range and error bars to retain the original minimum and maximum values.

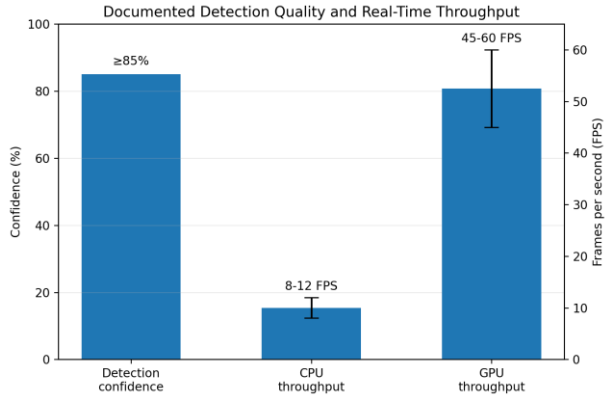


Fig. 4. Combined documented detection-confidence and throughput results.

Metric	Documented observation
Detection confidence	Above 85% for clearly visible, well-lit COCO objects
GPU throughput	45-60 FPS with YOLOv8n
CPU throughput	8-12 FPS with YOLOv8n
Custom vocabulary	83 classes: 80 COCO + 3 custom categories
API response	Annotated JPEG, structured detections, count and inference time
mAP / precision / recall	Computed by evaluate.py; numeric values were not included in the report

TABLE II. DOCUMENTED EXPERIMENTAL RESULTS

C. Result Analysis

The observed throughput demonstrates why the nano model was selected for the real-time default. GPU execution leaves sufficient headroom for higher-resolution inputs or additional tracking, whereas CPU operation remains usable for demonstrations and low-cost deployments. The hybrid labeling pipeline generated an 83-class dataset without manual box drawing and visually produced tight boxes for most sampled images. Transfer learning preserves generic low-level features and adapts later layers to the custom categories, reducing the training burden compared with full retraining.

The strongest architectural property is reuse: the same ObjectDetector that is validated offline is loaded by both the CLI and the web service. Consequently, thresholding, non-maximum suppression, colour assignment, counting, and model updates remain consistent across deployment modes. The absence of reported numerical mAP, precision, and recall values limits direct comparison with published benchmarks; these values should be exported from evaluate.py in the final experimental record.

D. API and End-to-End Verification

The output chapter verifies both deployment paths rather than evaluating only an offline notebook. In the browser interface, the user can drag and drop an image or select the live-camera tab. A successful request reaches the FastAPI endpoint, is decoded, passes through the shared ObjectDetector, and returns an annotated result. The structured response preserves machine-readable information in addition to the displayed JPEG, allowing downstream software to use labels, confidence values, coordinates, counts, dimensions, and timing. The health endpoint provides a simple readiness check before a client attempts inference.

The CLI result demonstrates that acquisition, prediction, non-maximum suppression, rendering, object counting, and FPS tracking operate together in a continuous loop. The reported examples include simultaneous detection of persons and everyday COCO objects, and the supplied frame shows a phone box with a 65% confidence label. The documented custom-class workflow also reached deployment: images were collected, automatically labeled, fine-tuned, and then loaded through the same model argument used by both front ends. This confirms that extending the vocabulary does not require changes to the visualization or API code.

E. Robustness and Real-Time Behaviour

Testing covered different object categories, camera distances, lighting conditions, and scene densities. Confidence threshold adjustment provides a practical mechanism for controlling the precision-recall trade-off. A lower threshold increases sensitivity but may admit weak detections, whereas a higher threshold suppresses uncertain

boxes at the risk of missing small, partly occluded, or poorly illuminated objects. NMS is applied after confidence filtering to maintain one representative box for each object. The combination is appropriate for interactive use because both controls are computationally inexpensive compared with the neural-network forward pass.

The 45-60 FPS GPU range substantially exceeds a conventional 25-30 FPS video rate, indicating that the nano model can process frames with real-time headroom on accelerated hardware. The 8-12 FPS CPU range is slower but remains useful for demonstrations, low-cost monitoring, and applications where the scene changes gradually. Throughput will vary with input resolution, processor generation, camera decoding, the number of detections, and competing system load. Therefore, these figures are correctly treated as documented observations rather than universal benchmark claims.

F. Dataset Quality and Custom-Class Learning

The custom data pipeline reduces the main practical cost of extending an object detector: drawing boxes manually. Known COCO-compatible concepts benefit from the localization ability of the pretrained YOLO model. For unseen concepts, CLIP provides open-vocabulary verification and accepts images above the documented 0.40 similarity threshold. Visual sampling of generated annotations indicated that most retained images received plausible labels, while ambiguous or low-quality examples could be discarded. The resulting vocabulary contains 83 classes, combining the original 80 COCO labels with earphones, earbuds, and TWS buds.

This approach is best interpreted as weak supervision rather than a replacement for expert annotation. A whole-image box can be acceptable when a product photograph contains one centered object, but it does not teach precise boundaries when the target occupies a small part of the scene. Transfer learning partially compensates by reusing generic visual features learned from COCO, yet final accuracy remains dependent on label quality, class balance, background diversity, and the separation between training and validation images. A future ablation study should compare YOLO-only proposals, CLIP-only verification, the hybrid method, and a manually corrected subset.

G. Comparison with Alternative Detectors

The choice of YOLOv8n is consistent with the project requirement for low latency on consumer hardware. Faster R-CNN remains a valuable accuracy-oriented architecture, but its proposal and refinement stages generally impose more inference cost. SSD also provides single-pass detection, although modern YOLO variants offer a convenient training and deployment ecosystem with integrated augmentation, validation, NMS, and export support. Transformer detectors such as DETR and RT-DETR simplify some matching design decisions and provide strong global reasoning, but model size and deployment constraints must still be considered for small academic systems.

The comparison does not imply that one detector is universally superior. A safety-critical application may accept lower speed for higher recall on small targets, while a live camera interface prioritizes predictable latency. The modular detector wrapper in this project makes model substitution possible because acquisition, API response formatting, and rendering are separated from the loaded checkpoint. A future comparative experiment can therefore evaluate several YOLO sizes, SSD, Faster R-CNN, or RT-DETR under the same input resolution and hardware, reporting mAP, latency, memory use, and energy consumption.

D. Limitations

The current custom-class labeling strategy assigns a whole-image box when CLIP verifies an unseen class, which is appropriate mainly when the scraped object dominates the image. Ambiguous backgrounds can create weak localization noise. The documented results also use a limited custom vocabulary and do not report class-wise AP, confusion analysis, dataset size after filtering, or repeated hardware trials. Real-time performance depends on camera resolution and available acceleration, while Server-Sent Events from the server webcam do not yet represent a general multi-camera streaming architecture.

V. CONCLUSION AND FUTURE SCOPE

The project demonstrates a complete real-time object-detection lifecycle rather than a stand-alone inference demo. It integrates web-based image collection, hybrid YOLO/CLIP auto-labeling, YOLOv8 transfer learning, validation, a reusable detection core, OpenCV visualization, and FastAPI deployment. The documented system detects standard COCO objects and supports new custom categories while maintaining interactive throughput on both GPU and CPU hardware.

Future development should replace weak whole-image boxes with tighter proposal-guided or human-verified labels, preserve class-wise evaluation outputs, and add persistent tracking through DeepSORT or ByteTrack. Additional improvements include active learning for low-confidence samples, quantization for edge devices, authenticated API access, persistent experiment logging, multi-camera ingestion, and rule-based alerts for selected classes or restricted regions.

A publication-oriented evaluation should report the exact retained dataset size, class distribution, train-validation split, per-class precision and recall, mAP at multiple IoU thresholds, confusion patterns, and repeated FPS measurements on clearly identified hardware. Export formats such as ONNX or TensorRT can be evaluated for edge deployment, while authentication, rate limiting, audit logging, and encrypted transport are needed before exposing the API outside a trusted local environment. These extensions preserve the present modular design while improving localization quality, reproducibility, scalability, and operational security.

A. Persistent Tracking and Temporal Analytics

The present detector treats every frame independently. A natural next stage is to associate detections across time so that each physical object retains a stable identity. Trackers such as DeepSORT or ByteTrack can combine YOLO boxes with motion and appearance information, enabling trajectory visualization, dwell-time estimation, line-crossing counts, and entry or exit events. Temporal association would also reduce flickering labels when an object is briefly occluded or its confidence falls below the threshold for one frame. The tracking module should remain downstream of ObjectDetector so that improvements to the detector automatically benefit all temporal analytics without coupling the training pipeline to one tracking algorithm.

Event logic can then operate on tracks instead of isolated boxes. Examples include detecting an object left unattended for a specified duration, counting products moving through a region, or raising an alert when a selected class enters a restricted area. Such rules require configurable regions of interest, clock synchronization, duplicate-event suppression, and a persistent event store. They also require careful threshold design: overly sensitive rules produce alert fatigue, while conservative rules may miss important events. Evaluation should therefore report event precision, event recall, detection-to-alert delay, and the stability of object identities across occlusion and re-entry.

B. Edge Deployment and Model Optimization

Although YOLOv8n already provides useful CPU performance, edge deployment can be improved through model export, quantization, and hardware-specific acceleration. The trained checkpoint can be exported to ONNX and then evaluated with ONNX Runtime, TensorRT, OpenVINO, or mobile inference engines. Half-precision and integer quantization may reduce latency, memory use, and energy consumption, but the compressed model must be revalidated because small or low-contrast custom objects can lose accuracy after aggressive quantization. A fair comparison should use the same validation images and report mAP, per-class recall, latency, peak memory, model size, and power consumption.

Edge devices also impose constraints that are not visible on a development workstation. Camera capture, image resizing, inference, post-processing, rendering, and network transmission compete for limited CPU, GPU, and memory resources. The software should support adjustable input resolution, frame skipping, asynchronous capture, and bounded queues so that old frames are discarded instead of building delay. For embedded surveillance, the important quantity is not only raw FPS but end-to-end latency from photon capture to displayed or transmitted result. Measuring each pipeline stage separately would identify whether optimization is required in decoding, inference, NMS, drawing, or API serialization.

C. Security, Privacy, and Responsible Deployment

The current FastAPI service is suitable for a trusted demonstration environment, but a network-accessible deployment should add authentication, authorization, encrypted transport, file-size limits, content-type validation, request-rate controls, and structured audit logs. Uploaded images may contain faces, identity documents, locations, or confidential workplace information. A production policy should define why images are processed, whether they are retained, who can access detections, and when logs are deleted. Whenever possible, inference should occur locally and the service should return results without storing original frames. Administrative interfaces should expose only the minimum information needed for the intended task.

Responsible deployment requires bias and failure analysis because web-scraped data may overrepresent studio images, popular brands, or narrow backgrounds. Performance on clear COCO objects does not guarantee reliability in dim, crowded, or unusual scenes. The model should therefore be validated in its actual operating environment, with human oversight for consequential decisions.

D. Reproducible Evaluation and Benchmarking

A stronger experimental protocol should preserve the exact filtered image count for every class, class-frequency distribution, duplicate-removal statistics, train-validation assignment, random seed, software versions, model checkpoint, input size, augmentation settings, and hardware configuration. Training curves should include box, classification, and distribution-focal losses together with validation precision, recall, mAP@0.5, and mAP@0.5:0.95. Class-wise AP and qualitative error examples are particularly important for the three custom categories because semantically similar labels such as earphones, earbuds, and TWS buds may be confused with one another. Repeated runs would indicate whether improvements are stable or caused by one favorable split.

Benchmarking should separate detector quality from deployment speed. Accuracy comparisons must use an identical validation set and IoU policy, whereas runtime comparisons must use the same image size, batch size, warm-up procedure, and measurement window. CPU and GPU trials should report model-loading time separately from steady-state inference. The API should additionally report encoding and network overhead, because browser-visible latency can differ from neural-network inference time. These practices would allow the project to be compared fairly with YOLO variants, SSD, Faster R-CNN, and real-time transformer detectors without overstating results that the source report does not numerically provide.

E. Application Extension

The modular architecture can support surveillance, retail inventory, assistive vision, traffic observation, industrial inspection, and classroom demonstrations by changing the trained vocabulary and downstream rules. Retail applications may combine detection with shelf regions and stock counts; assistive systems may convert selected detections into speech; industrial applications may identify missing components or protective equipment. Each domain requires its own representative dataset, acceptance criteria, and error-cost analysis. The main reusable contribution is therefore not one fixed model but the connected scrape-label-train-evaluate-deploy workflow and the shared inference interface through which a domain-specific checkpoint can be introduced without rewriting the CLI or web service.

REFERENCES

[1] R. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation,” in Proc. IEEE CVPR, 2014, pp. 580-587.
[2] K. He, X. Zhang, S. Ren, and J. Sun, “Spatial pyramid pooling in deep convolutional networks for visual recognition,” in Proc. ECCV, 2014, pp. 346-361.
[3] R. Girshick, “Fast R-CNN,” in Proc. IEEE ICCV, 2015, pp. 1440-1448.
[4] S. Ren, K. He, R. Girshick, and J. Sun, “Faster R-CNN: Towards real-time object detection with region proposal networks,” in Proc. NeurIPS, 2015, pp. 91-99.
[5] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” in Proc. IEEE CVPR, 2016, pp. 779-788.
[6] W. Liu et al., “SSD: Single shot MultiBox detector,” in Proc. ECCV, 2016, pp. 21-37.
[7] J. Redmon and A. Farhadi, “YOLO9000: Better, faster, stronger,” in Proc. IEEE CVPR, 2017, pp. 7263-7271.
[8] T.-Y. Lin et al., “Feature pyramid networks for object detection,” in Proc. IEEE CVPR, 2017, pp. 2117-2125.

[9] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in Proc. IEEE ICCV, 2017, pp. 2980-2988.
[10] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," in Proc. IEEE ICCV, 2017, pp. 2961-2969.
[11] J. Redmon and A. Farhadi, "YOLOv3: An incremental improvement," arXiv:1804.02767, 2018.
[12] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "YOLOv4: Optimal speed and accuracy of object detection," arXiv:2004.10934, 2020.
[13] Z. Ge, S. Liu, F. Wang, Z. Li, and J. Sun, "YOLOX: Exceeding YOLO series in 2021," arXiv:2107.08430, 2021.
[14] C. Li et al., "YOLOv6: A single-stage object detection framework for industrial applications," arXiv:2209.02976, 2022.
[15] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, "YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors," arXiv:2207.02696, 2022.
[16] C.-Y. Wang, I.-H. Yeh, and H.-Y. M. Liao, "YOLOv9: Learning what you want to learn using programmable gradient information," arXiv:2402.13616, 2024.
[17] A. Wang et al., "YOLOv10: Real-time end-to-end object detection," in Proc. NeurIPS, 2024.
[18] Z. Tian, C. Shen, H. Chen, and T. He, "FCOS: Fully convolutional one-stage object detection," in Proc. IEEE ICCV, 2019, pp. 9627-9636.
[19] X. Zhou, D. Wang, and P. Krähenbühl, "Objects as points," arXiv:1904.07850, 2019.
[20] M. Tan, R. Pang, and Q. V. Le, "EfficientDet: Scalable and efficient object detection," in Proc. IEEE CVPR, 2020, pp. 10781-10790.
[21] N. Carion et al., "End-to-end object detection with transformers," in Proc. ECCV, 2020, pp. 213-229.
[22] X. Zhu et al., "Deformable DETR: Deformable transformers for end-to-end object detection," in Proc. ICLR, 2021.
[23] H. Zhang et al., "DINO: DETR with improved denoising anchor boxes for end-to-end object detection," in Proc. ICLR, 2023.
[24] Y. Zhao et al., "DETRs beat YOLOs on real-time object detection," in Proc. IEEE/CVF CVPR, 2024, pp. 16965-16974.
[25] A. Radford et al., "Learning transferable visual models from natural language supervision," in Proc. ICML, 2021, pp. 8748-8763.
[26] S. Liu et al., "Grounding DINO: Marrying DINO with grounded pre-training for open-set object detection," in Proc. ECCV, 2024.
[27] T. Cheng et al., "YOLO-World: Real-time open-vocabulary object detection," in Proc. IEEE/CVF CVPR, 2024, pp. 16901-16911.
[28] T.-Y. Lin et al., "Microsoft COCO: Common objects in context," in Proc. ECCV, 2014, pp. 740-755.
[29] M. Everingham et al., "The Pascal Visual Object Classes (VOC) challenge," Int. J. Comput. Vis., vol. 88, no. 2, pp. 303-338, 2010.
[30] A. Dosovitskiy et al., "An image is worth 16×16 words: Transformers for image recognition at scale," in Proc. ICLR, 2021.