International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 10 Issue: 04 | April - 2026 SJIF Rating: 8.659 ISSN: 2582-3930

Find-It: Smart Forgotten Object Detection System

Prof. Mohammed. Juned Shaikh, Furkan Ahmad Farooqui, Aatika Khan, Mohammad Ali

Department of Computer Engineering, Rizvi College of Engineering
Mumbai, India.

Abstract - This project proposes an IoT-based forgotten object detection system that assists users in locating commonly misplaced household or personal items such as keys, wallets, and mobile phones. The system uses a camera to continuously monitor the environment and applies computer vision techniques to detect predefined objects. Whenever an object is detected, the system stores its last-seen image along with the date and time, creating a visual memory of the object's location. This approach reduces manual searching, saves time, and provides a cost-effective smart assistance solution suitable for homes and workplaces.

Keywords— Object Detection, YOLOv8, Forgotten Objects, FastAPI, React, Open Images Dataset, Last-Seen Tracking, Model Validation, Bias Testing

I. INTRODUCTION

People frequently lose track of small daily-use objects. The most common examples are phone, wallet, book, and remote. This causes stress and time loss in homes and workplaces. Many existing solutions depend on hardware tags, which are useful but require users to tag every object.

FindIt uses a camera-only approach with computer vision, so the user does not need to attach any hardware to objects.

The core idea is simple: detect target objects in camera frames and store the most recent evidence (image + time). This creates a practical visual memory that users can check from a dashboard.

This paper is based on the implemented code in this repository. The backend API is in `backend/main.py`, detection loop in `backend/camera.py`, and UI in `frontend/src/App.jsx`. Hence, the paper reports implementation-grounded behavior rather than hypothetical design.

II. RELATED WORK

Object detection research has evolved from region-based approaches to single-stage real-time approaches. YOLO introduced end-to-end single-pass detection and made real-time object detection practical. YOLOv3 improved multi-scale predictions and backbone strength.

YOLOv8 is widely adopted in practice due to good speed, simple workflow, and practical tooling.

In parallel, transformer-based detectors such as DETR showed a different direction: set-based prediction and cleaner post-processing. However, for lightweight deployment in a local desktop setup, YOLO-style models are often preferred due to easier real-time behavior.

Dataset selection also matters. COCO is widely used for pretrained models and baseline benchmarking. Open Images provides broader category coverage and supports custom class filtering workflows. For this project, Open Images resources (Google-hosted) are used in dataset workflow discussions, while YOLO pretrained behavior provides practical baseline support.

Related assistive systems for finding lost objects exist, including wearable and smart-home robotic approaches. FindIt differs by targeting a simple and reproducible camera + web-dashboard architecture with explicit object history for daily use.

III. METHODOLOGY

A. System Architecture

Find-It architecture has four layers: (1) detection loop, (2) API layer, (3) storage layer, and (4) frontend layer.
- Detection loop (`backend/camera.py`): reads camera frames using OpenCV, runs YOLO, maps labels to canonical class names, and writes files.
- API layer (`backend/main.py`): provides REST endpoints and MJPEG stream.
- Storage layer (`detections/<class>/`): stores `last_seen.jpg`, `meta.json`, `history.json`.
- Frontend (`frontend/src/App.jsx`): dashboard, history, alerts, and settings.

This separation keeps the system responsive: model inference stays in the background thread while API routes serve already prepared state.

B. Dataset and Preprocessing

Dataset Characteristics:

The project targets household object classes aligned with application use. In implementation, canonical classes are user-friendly names: cell phone, book, wallet, remote, and power plug. Class-name mapping is used to bridge detector output vocabulary and UI vocabulary.

Preprocessing Pipeline:

The workflow uses Open Images resources for class filtering and dataset preparation. Typical preprocessing steps include class filtering, label normalization, train/validation split, and YOLO-format annotation generation. For this paper, model trend analysis is based on the real metrics file `paper/training_metrics_real.csv` generated from training logs.

C. Model Architecture

Architecture Specifications:

The detection stack uses Ultralytics YOLO with image size 640. Runtime filtering uses confidence threshold 0.35 after model inference call with 0.3 confidence floor. A category mapping dictionary converts labels such as `handbag` to canonical `wallet`.

Training Configuration:

The system supports a custom checkpoint path (`backend/runs/detect/yolo_household_mvp/weights/best.pt`). If custom weights are present, they are loaded; otherwise default `yolov8n.pt` is used. This provides a practical fallback mechanism and smooth deployment.

IV. IMPLEMENTATION

A. Frontend Implementation

Key Features:

- Live camera feed with on/off switch
- Detected objects card sorted by most recent detection
- History page with search and per-entry image links
- Alerts for never-detected and not-detected-since cases
- Clear History control with backend delete call

The frontend polls status and refreshes data when the camera is active, giving near real-time user feedback.

B. Backend Implementation

API Architecture:
The backend exposes object list, image retrieval, history retrieval, merged history, camera status, camera start/stop, and stream endpoints. The stream endpoint returns MJPEG frames from shared in-memory bytes updated by the camera thread.

Security Measures:
- CORS policy is explicitly set for local frontend origins.
- Cache-control headers are used for fresh image retrieval.
- Current implementation does not include authentication/authorization; this is identified as a known limitation for production use.

- Data clearing endpoint allows user-controlled local retention management.

C. Model Explainability

Model Explainability:
Current explainability is operational rather than feature-attribution based. The user can inspect `last_seen.jpg`, timestamps, and ordered history for each class. This gives practical traceability: users can see when and where the system detected an object.

Planned explainability upgrades include confidence trend view per class, threshold-sensitivity reports, and optional visual attention maps for deeper model interpretation.

V. EVALUATION

A. Model Performance

Performance is summarized from real training metrics. The model reaches best validation values around epoch 24 and then shows normal fluctuation in later epochs. The trajectory indicates useful learning progress compared to early epochs.

TABLE I: PERFORMANCE METRICS

Metric	Value (real log)
Epoch range	1 - 42
Best mAP50	0.57660 at epoch 24
Best mAP50-95	0.47298 at epoch 24
Final mAP50	0.38859
Final mAP50-95	0.33999
Mean mAP50 (std)	0.34991 (0.08392)
Mean mAP50-95 (std)	0.27404 (0.08222)
First 10-epoch mean mAP50	0.26519
Last 10-epoch mean mAP50	0.39604
Trend gain (Last10 - First10)	0.13086

B. Validation Methodology

Validation Methodology:

Metrics in this paper come from validation values logged during model training and exported into `paper/training_metrics_real.csv`. This supports checkpoint selection and development-stage comparison.

External Validation:

A separate locked external test benchmark is not included in the current repository artifact. Therefore, current values should be interpreted as in-project validation metrics, not final deployment guarantees. Future work should add external test splits and per-class benchmark reporting.

C. Bias Testing and Fairness Analysis

Bias Testing and Fairness Analysis:
In this project context, fairness risk mostly appears as environmental bias rather than demographic bias. Key sources are class imbalance, background bias, lighting differences, and viewpoint bias.

Current logs provide aggregate metrics; per-class fairness matrix is not yet available. To improve fairness testing, future protocol should include condition-wise evaluation (lighting, distance, angle, clutter), per-class precision/recall, and threshold sensitivity sweeps.

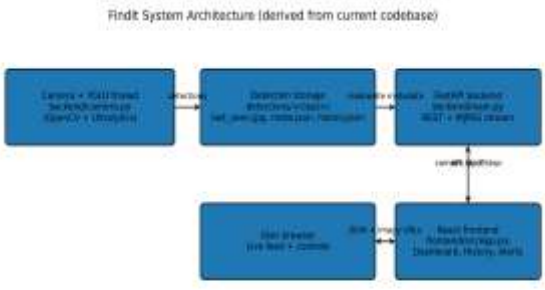

Figure 1. System architecture generated from current code modules.

Figure 2. Real training metrics plot from paper/training_metrics_real.csv.

VI. DISCUSSION

Explainability Benefits:
- Users can verify detections with saved evidence images
- Time-based logs support practical object retrieval decisions
- System behavior is easy to audit through API and files

Limitations:
- No external benchmark set in current artifact
- No built-in authentication for API endpoints
- Possible performance drop under low light, occlusion, and uncommon viewpoints
- File storage can grow over long usage periods

Practical Discussion:
FindIt is strong as a prototype and applied research system. It is especially useful for controlled indoor setups. For production-quality deployment, the system needs stronger validation, security controls, and long-term data lifecycle management.

VII. CONCLUSION

FindIt provides a practical camera-based forgotten-object detection workflow with live monitoring, history logging, and user-facing retrieval support. The implementation is modular, understandable, and easy to extend.

Future improvements should prioritize external validation benchmarks, per-class fairness reporting, authentication, and deployment optimization for larger real-world environments.

ACKNOWLEDGMENT

The authors thank the project guide Prof.Mohammed Juned Shaikh, department faculty, and open-source communities behind Ultralytics, FastAPI, React, and OpenCV for tools and learning resources used in this work.

RESOURCES USED

[1] Open Images V7 download page (Google-hosted): https://storage.googleapis.com/openimages/web/download_v7.html

[2] Ultralytics YOLOv8 Model for identification: https://github.com/ultralytics/ultralytics

[3] FastAPI documentation in backend system: https://fastapi.tiangolo.com

[4] React documentation for frontend application: https://react.dev

[5] OpenCV official website: https://opencv.org

[6] COCO dataset: https://cocodataset.org

REFERENCES

[1] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," CVPR, 2016. Available: https://openaccess.thecvf.com/content_cvpr_2016/html/Redmon_You_Only_Look_CVPR_2016_paper.html

[2] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement," arXiv:1804.02767, 2018. Available: https://arxiv.org/abs/1804.02767

[3] G. Jocher et al., "Ultralytics YOLOv8," GitHub repository, 2023. Available: https://github.com/ultralytics/ultralytics

[4] N. Carion et al., "End-to-End Object Detection with Transformers," ECCV, 2020. Available: https://arxiv.org/abs/2005.12872

[5] T.-Y. Lin et al., "Microsoft COCO: Common Objects in Context," ECCV, 2014. Available: https://cocodataset.org

[6] A. Kuznetsova et al., "The Open Images Dataset V4," IJCV, 2020. Available: https://storage.googleapis.com/openimages/web/download_v7.html

[7] H.-M. Liao et al., "GO-Finder: A Registration-Free Wearable System for Assisting Users in Finding Lost Objects via Hand-Held Object Discovery," 2021. Available: https://arxiv.org/abs/2101.07314

[8] C. Xie et al., "Finding misplaced items using a mobile robot in a smart home environment," Frontiers of Information Technology & Electronic Engineering, 2019. Available: https://link.springer.com/article/10.1631/FITEE.1800275