© JAN 2026 | IRE Journals | Volume 9 Issue 7 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I7-1713245

SmartView: AI-Based Object Detection System

C M SUMANA[1], SUBANI D[2], ALUR MUSKAN MAHEK[3], N LAKSHMI[4], V ASHWINI[5]
[1]*Asst Prof, Dept of CSE (Artificial Intelligence and Machine Learning)*
[2,3,4,5] *Students of Dept of CSE (Artificial Intelligence and Machine Learning)*
Rao Bahadur Y Mahabaleswarappa Engineering college, Ballari, Karnataka, India.

Abstract - Recent advancements in computer vision have enabled automated systems to identify and localize multiple objects efficiently across diverse visual inputs. This paper presents Smart-view, an AI-based multi-source object detection system designed for both real-time and offline visual analysis. The system integrates the YOLOv8 deep learning model with a Flask-based web framework to support object detection from images, prerecorded videos, live webcam streams, and online video sources such as YouTube. Supporting tools including Open CV and FFMPEG are employed for frame acquisition, prepossessing and video conversion. To enhance usability, computationally intensive tasks are executed asynchronously, ensuring a responsive user interface. Detected objects are visually annotated and systematically logged in structured CSV format for further analysis. The proposed system demonstrates that efficient and scalable object detection can be achieved using lightweight models on CPU-based environments.

Keywords: Object Detection, YOLOv8, Computer Vision, Flask, Real-Time Processing, AI Applications.

I. INTRODUCTION

Object detection is an essential capability in modern computer vision systems, enabling automated analysis of visual data across various application domains. In this work, Smart View is introduced as a unified AI-based object detection system designed for practical deployment rather than theoretical experimentation. The system supports multiple visual input sources including images, prerecorded videos, live webcam streams, and online video links within a single processing framework Such multi-source object detection frameworks are widely adopted in modern computer vision applications [12].

Smart View integrates the YOLOv8 deep learning model with a Flask-based web interface to provide real-time and batch object detection through a standard web browser. Multimedia processing tools such as Open CV and FFMPEG are utilized to handle frame capture, prepossessing and video conversion. To improve usability, computationally intensive video detection tasks are executed in the background, ensuring that the user interface remains responsive while detection results are generated and stored for further analysis.

II. REVIEW OF LITERATURE

1. Joseph Redmon et al. Their work "YOLO: You Only Look Once" explains a real-time single-stage object detection method where objects are detected in one pass.

2. Alexey Bochkovskiy et al. Offered their work "YOLOv4" which improves both speed and accuracy using optimized training strategies for efficient object detection.

3.Tsung-Yi Lin et al. Presented their work "Focal Loss for Dense Object Detection" which handles class imbalance and improves the precision of single-stage detectors.

4.Xizhou Zhu et al. Their work "Deep Feature Flow" explains faster video processing by sharing features across frames instead of doing full inference every time.

5.Gary Bradski & Adrian Kaehler. Offered their work "Learning Open CV" which provides practical methods for image/video operations like frame capture, re sizing, drawing, and video writing.

6.Ultralytics Team. Presented their work on "YOLOv8" which introduces improved architecture, better training optimization, and faster inference, making it suitable for real-time image and video detection tasks used in this project.

III. SYSTEM ARCHTECTURE

The proposed Smart View system is designed as a modular, web-based object detection platform that integrates deep learning, video processing, and data logging into a single application. The architecture

© JAN 2026 | IRE Journals | Volume 9 Issue 7 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I7-1713245

follows a layered design, separating the presentation layer (web interface), application logic layer (Flask and processing modules), and AI/model layer (YOLOv8), supported by auxiliary components for media handling, logging, and deployment.

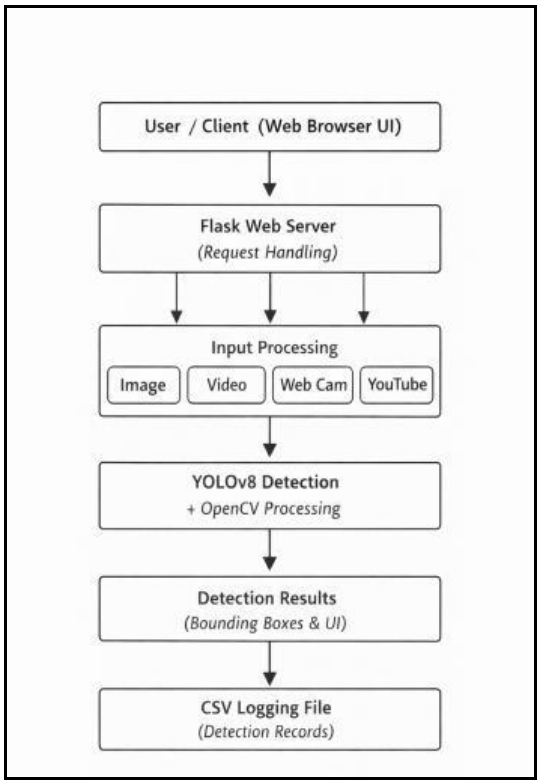


Fig 1: System Architecture

1. System Design & Architecture
→ The system follows a modular architecture using Flask for the web interface, YOLOv8 for detection, Open CV and FFMPEG for media processing, multi threading for background tasks, CSV logging for analytic, and Docker for deployment. This design ensures scalability, easy maintenance, and platform independence.

2. Model Selection (YOLOv8)
→ YOLOv8 is selected due to its high detection accuracy and real-time inference capability. The lightweight YOLOv8-nano model is loaded once during system startup, enabling fast and efficient object detection for all incoming requests, including live video streams and interactive visual inputs.

3. Input Handling
→ The system supports four modes of input:
Image Upload – Image is saved, processed by YOLOv8, and returned with bounding boxes. Video Upload – Video is uploaded and previewed instantly; full processing runs in a background thread. Webcam Stream – Live frames are captured using Open CV and processed in real time. YouTube Link – Videos are downloaded using YT-dlp, frame-processed with YOLOv8, and returned as browser-compatible output.

4. Frame Processing Workflow
→ During processing, each video frame is re sized to a resolution of 1280×720 and passed to the YOLOv8 model with a confidence threshold of 0.4. Detected objects are annotated with bounding boxes and class labels before being encoded for output visualization.

5. Background Processing (Multi-threading)
→ Multi-threaded background processing ensures that long-running detection tasks do not block the Flask web interface, allowing results to be displayed smoothly after processing completes., which is essential for processing live video streams and interactive visual inputs. Flask remains responsive during processing, and results are displayed automatically after completion, ensuring smooth user interaction.

6. Detection Logging (CSV Analytic)
→All detection outputs are saved in static/detection.CSV with details such as timestamp, input type, file/URL, frame number, detected objects, and object counts, enabling future analysis and comparison.

7. Web Interface (Flask)
→Flask manages file uploads, form handling, webcam streaming, displaying processed media, and presenting detection logs. The interface is designed to be simple, responsive, and user-friendly.

8. Deployment Using Docker
→Docker containerizes the entire system, eliminating dependency issues and enabling easy setup and cloud deployment on platforms like Render or AWS.

9. Testing & Validation
→Extensive testing was conducted on images, videos, online streams, and live webcam inputs to evaluate system performance. Evaluation focuses on detection accuracy, processing speed, playback smoothness, thread

© JAN 2026 | IRE Journals | Volume 9 Issue 7 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I7-1713245

stability, and correctness of CSV logs. YOLOv8 delivers fast and accurate detection across all inputs.

IV. IMPLEMENTATION

The project was implemented in a phased approach to build a unified object detection system supporting images, videos, webcam, and YouTube streams. The components interact in a pipeline where the user submits input (image, video, webcam, or YouTube URL), the server processes the input using the YOLOv8 model, generates processed outputs (images/videos with bounding boxes), logs metadata to a CSV file, and returns the results back to the browser for visualization.

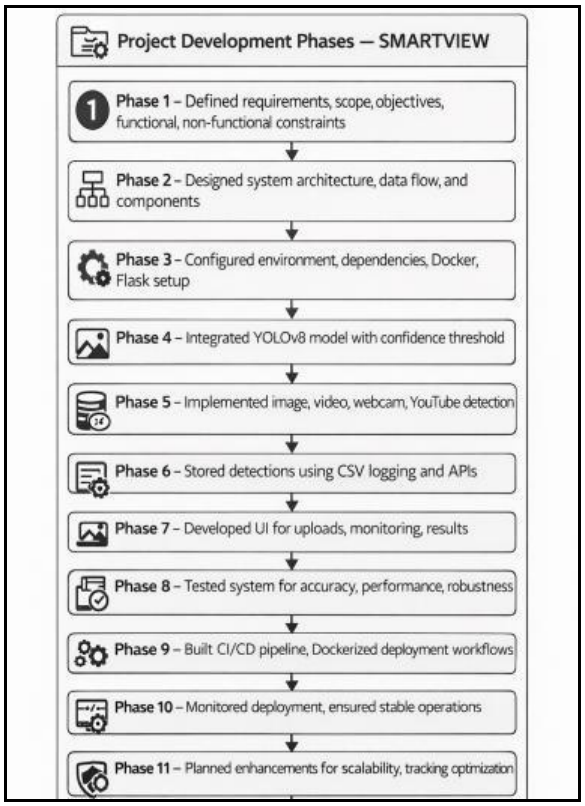


Fig 2: Different Phases of Object Detection System

Phase 1 — Requirement Analysis: Defined project scope, user goals, functional and non-functional requirements, and hardware suggestions. Key deliverable included requirements text and documentation.

Phase 2 — Architecture & Design: Designed a modular architecture with Flask back end, YOLO inference, video processing pipeline, storage, and UI. Module boundaries and deployment strategy (Docker) were finalized.

Phase 3 — Environment Setup: Configured reproducible environments with Python dependencies, Docker file, and optional docker-compose setup to ensure seamless local and containerized execution.

Phase 4 — Model Integration (Image): Integrated YOLOv8 for image detection with confidence threshold and generated annotated outputs saved for visualization.

Phase 5 — Video & Live Stream Processing: Implemented robust video handling for file uploads, webcam, and YouTube streams (VOD and live) using FFmpeg and background processing.

Phase 6 — Persistence & APIS: Detection metadata stored in CSV with REST endpoints for JSON retrieval and file download, ensuring easy access and traceability.

Phase 7 — Front end & UX: Developed intuitive UI for uploads, live monitoring, and detection results viewing with responsive design and client-side status updates.

Phase 8 — Testing & Validation: Performed unit and manual testing for all functionalities, including image, video, live streams, and API endpoints to ensure robustness.

Phase 9 — CI/CD & Container Build: Automated Docker image builds and deployment using GitHub Actions, optimizing reproducibility and release efficiency.

Phase 10 — Deployment & Operations: Deployed containerized application with persistent storage, GPU support, logging, and monitoring for reliable operation.

Phase 11 — Hardening & Future Enhancements: Planned migration to a database, authentication, job queues, and tracking extensions to scale and production the system.

V. RESULT

A. Image Detection

The system accurately identified objects in uploaded images using YOLOv8n. Output images (detected_<filename>.JPG) were generated, and

© JAN 2026 | IRE Journals | Volume 9 Issue 7 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I7-1713245

detected class names were logged into detection.CSV. High precision was observed for common objects such as persons, vehicles, and household items. Average processing time was less than 1 second per image.

B. Video Detection

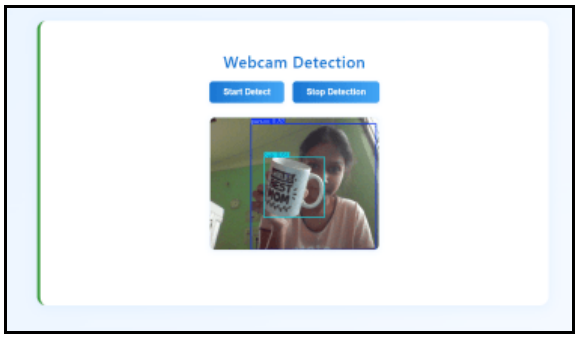
Videos were processed frame-by-frame with YOLO inference. A first-frame preview was generated in 1–2 seconds, while full video processing occurred in the background. Output videos were saved in .AVI and converted to browser-friendly .mp4 format. Detected objects per frame were logged with timestamps. Typical frame rates ranged from 20–30 FPS depending on resolution.

C. YouTube URL Detection

YouTube videos and live streams were downloaded or recorded using YTdlp and FFMPEG. Every 10th frame was processed to reduce computation. Output videos with bounding boxes were generated in both standard and web.mp4 formats for smooth playback. This feature allows detection on online media sources.

D. Webcam Stream Detection

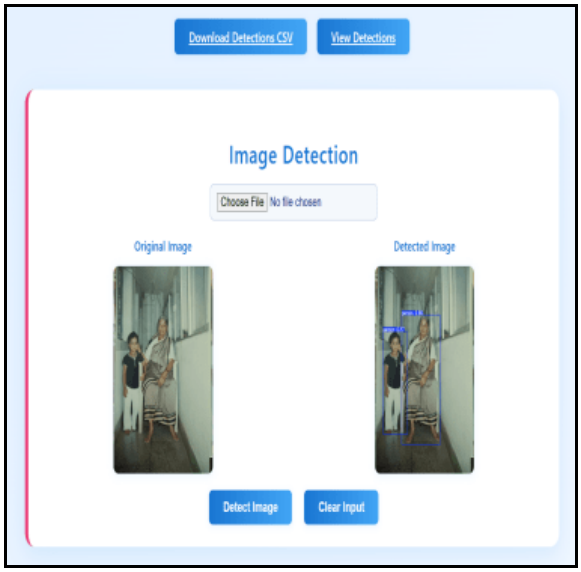
Real-time webcam detection produced live bounding boxes and MJPEG streamed frames with minimal lag. Continuous detection achieved 8–15 FPS on CPU hardware, demonstrating suitability for surveillance applications.



E. Detection Logging

All detection across images, videos, YouTube, and webcam streams were logged in detection's.CSV, including: timestamp, source type, source name, frame number, detected class names, and object counts. The CSV enables further analytic, heat maps, and model evaluation.

© JAN 2026 | IRE Journals | Volume 9 Issue 7 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I7-1713245



VI. CONCLUSION

The Smart View system successfully demonstrates a robust and versatile object detection platform built using YOLOv8, Flask, Open CV, and FFMPEG. The project achieves its goal of providing a unified interface for detecting objects in images, videos, webcam streams, and YouTube sources, making it suitable for multiple real-world applications such as surveillance, monitoring, and analytic.

By combining real-time inference with intuitive visual outputs, the system offers both practicality and efficiency. The integration of background processing, automatic video conversion, and detection logging enhances usability and reliability. Overall, Smart View shows that advanced computer vision capabilities can be deployed effectively even on CPU-based environments, without requiring high-end GPU resources. The proposed architecture demonstrates that scalable and multi-source object detection systems can be effectively implemented using lightweight models and open-source frameworks.

REFERENCES

[1] Ultralytics, “YOLOv8: Next-Generation Object Detection,” Ultralytics Documentation.[Online].Available: https://docs.ultralytics.com/
[2] Ultralytics “Ultralytics/ultralytics – YOLOv8 Official GitHub Repository,”GitHub.[Online].Available: https://github.com/ultralytics/ultralytics
[3] PyTorch Foundation, “PyTorch Documentation”, PyTorch.org. [Online]. Available: https://pytorch.org/
[4] G. Bradski, “The Open CV Library,” Open CV Documentation. [Online]. Available: https://docs.opencv.org/
[5] A. Ronacher, “Flask: A Python Micro framework,” Flask Documentation.[Online].Available: https://flask.palletsprojects.com/
[6] FFMPEG Team, “FFMPEG Documentation,” FFmpeg.org. [Online]. Available: https://ffmpeg.org/documentation.html
[7] yt-dlp Developers, “yt-dlp: A Feature-Rich YouTube Downloader,” GitHub. [Online]. Available: https://github.com/yt-dlp/yt-dlp
[8] C. R. Harrisetal., “Array programming with Numpy,” Numpy Documentation. [Online]. Available: https://numpy.org/doc/
[9] Python Software Foundation, “Python 3 Documentation,” Python.org. [Online]. Available: https://docs.python.org/
[10] Docker Inc., “Docker Documentation,” Docker Docs. [Online]. Available: https://docs.docker.com/
[11] Pallets Projects, “Werkzeug & Jinja2 Documentation,” PalletsProjects.com.[Online].Available: https://palletsprojects.com/
[12] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You Only Look Once: Unified, Real-Time Object Detection,” arXiv:1506.02640,2015.[Online].Available: https://arxiv.org/abs/1506.02640
[13] Apple Inc., “HTTP Live Streaming (HLS) Specification,” Apple Developer Documentation. [Online]. Available: https://developer.apple.com/streaming/