International Journal of Scientific Research & Engineering Trends
Volume 12, Issue 2, Mar-Apr-2026, ISSN (Online): 2395-566X

Corrosion Detection and Monitoring System: Yolo Based Real Time Deep Learning Framework

Mr. Prajwal Narayan Chaudhary, Mr. Pranav Prasad Kulkarni,
Mr. Chetan Ashok Bhalekar, Mr. Aditya Ganesh Gunjal, Prof. Kalyani Zirpe
Department of Artificial Intelligence & Data Science Engineering
Parvatibai Genba Moze College of Engineering Wagholi, India

Abstract- — Corrosion is a significant cause of damage in industrial infrastructure, transportation systems, marine equipment, pipelines, and metal parts. Traditional methods for inspecting corrosion mainly rely on manual observation and regular maintenance. These processes are time-consuming, labor intensive, and are subjective, which can lead to human error. Delays in spotting corrosion can lead to serious structural failures, higher maintenance costs, operational downtime, and safety risks. To address these issues, this paper introduces a real-time AI-based Corrosion Detection and Monitoring System. This system uses the YOLOv5 deep learning framework along with a modern web-based structure. The new system combines computer vision, deep learning, and web technologies to automate the detection of corrosion and assess its severity. It uses the YOLOv5s object detection model to find corrosion areas in uploaded images and live camera feeds. A React.js frontend offers an engaging and responsive user interface. Meanwhile, a FastAPI backend handles image processing, runs the necessary calculations, and communicates results. The system evaluates detected corrosion areas using bounding box calculations to estimate the amount of corrosion and categorize its severity as mild, moderate, or severe. It also features graphical visualizations, historical tracking, and repair suggestions to support preventive maintenance. This framework provides nearly real-time detection with higher accuracy and less reliance on manual inspection. Its modular and scalable design allows it to be used in various industries, including maritime, civil infrastructure, manufacturing, automotive, and aviation. Tests show that the system successfully identifies corrosion under different environmental conditions while maintaining good computational performance. This solution represents a cost- effective and smart way to monitor structural health and perform predictive maintenance.

Keywords- Corrosion detection, deep learning, YOLOv5, real-time monitoring, FastAPI, React.js, industrial safety, computer vision, automated inspection

I. INTRODUCTION

The phenomenon of corrosion is a major problem in industries that affects metallic materials slowly due to the interaction between the chemicals and electrochemicals found in the environment with metals. This causes a detrimental effect on the efficiency and durability of industrial assets, including bridges, pipelines, tanks, ships, aircraft parts, and machinery. It has been found through research in industries that billions of dollars' worth of maintenance and repairs in the industrial world have been caused by corrosion. The sectors of oil and gas, aerospace, and civil engineering are affected significantly if there is corrosion in their systems.[1]

Conventional approaches for corrosion testing rely primarily on visual inspections, ultrasonic inspection, radiographic examination, and chemical analysis. Although these processes are popular in industrial settings, their limitations include high costs, need for specialized personnel, lengthy duration, and inconsistency in inspection results [2]. Manual inspection is particularly problematic when working in dangerous or inaccessible areas, such as oil rigs, pipes under water, high-rise bridges, and congested manufacturing environments. Furthermore, conventional image processing algorithms using thresholding, edge detection, and tailored feature extraction do not consistently produce accurate results due to variations in lighting conditions and backgrounds [3].

Recently, advances in artificial intelligence, deep learning, and computer vision have transformed the field of automated industrial inspection and predictive maintenance. The use of object detection algorithms that rely on deep learning enables automatic identification of essential features from vast amounts of data, leading to increased detection performance. The YOLO family of object detectors is particularly efficient at providing an optimal trade-off between computational complexity and detection precision [4]. YOLO-based object detectors are widely applied in industrial sectors like crack detection, defect assessment, quality assurance, and structural health monitoring systems [5].

The YOLOv5 algorithm, which is developed by Ultralytics, boasts more advanced features than the previous versions because of faster processing capabilities and enhanced feature

extraction. It uses the single-shot object detection approach, which combines object localization and classification during just one step of neural network processing [6]. In terms of applications, the YOLOv5 algorithm is appropriate for real-time corrosion detection owing to its transfer learning capability and targeted corrosion dataset utilization [6],[7].

The main aims and contributions of this research are summarized as follows:

- Develop an automated real-time corrosion detection system using the YOLOv5 deep learning framework.
- Integrate live camera monitoring and image upload features within a web-based interface.
- Estimate corrosion severity using bounding-box- based calculations of corrosion area.
- Provide interactive visualizations, historical analysis, and maintenance suggestions.
- Create a scalable and cost-effective system suitable for industrial predictive maintenance.

The rest of the paper is structured as follows. Section II describes the proposed approach p. Section III presents the hardware architecture. Section IV provides the software architecture. Section V describes the operation and Section VI shows the results and analysis. The advantages and disadvantages are described in Section VII. Section VIII discusses future scopes. Section IX concludes paper.

II. PROPOSED SYSTEM AND METHODOLOGY

We design a two-tier system comprising a React.js frontend and a FastAPI backend.

Frontend: The React interface provides two tabs for input:
(1) Image Upload, where users select a file; (2) Live Camera, which streams video via WebRTC. We use HTML5 <video> and <canvas> elements: the video displays the live feed, and on user command, the current frame is drawn to an off-screen canvas. The canvas image is converted to a Blob and sent to the server. For file upload, we use an <input type="file"> element; the selected image is previewed locally via URL. create Object URL. On each request, the frontend shows a loading overlay, then displays results when ready. Results include the original image with bounding boxes (drawn on a canvas) and analytical charts (using the recharts library) for confidence scores. Detected results are also stored in window. Local Storage with a timestamp and compressed thumbnail for a detection history panel.

Backend (FastAPI): We implement a REST endpoint POST /detect to accept images. FastAPI is a modern, high-performance Python framework[7]. It handles CORS so our React app (likely served on a different port) can communicate with it. In Python, we read the uploaded image bytes, convert them to a PIL Image in RGB mode, and resize/normalize as needed. We load a pretrained YOLOv5s model via PyTorch Hub.

We select YOLOv5s because it is the smallest and fastest YOLOv5 model [3]. After the preprocessing of the image, we perform inference, which involves using results = model(img). We then convert the result (which is in PyTorch Tensor format) to a pandas dataframe using results.pandas().xyxy[0]. In the data frame, each entry consists of [xmin, ymin, xmax, ymax, confidence, class, name]. The data frame is filtered with the condition that confidence > 0.16. We then find the area of each bounding box (i.e., (xmax-xmin)*(ymax-ymin)), sum all of them, and divide by the area of the whole image to find corrosion_percent.

The corrosion detection and monitoring system is based on the development of a multi-layered architecture, which is modular and comprises layers related to image acquisition, image preprocessing, deep learning inference, communication, visualization, and storage. This architecture is created particularly to facilitate the use of the system for real-time corrosion monitoring in industries.

The frontend layer is built on React.js framework and allows users to upload images or capture images in real-time using a live camera feed. It interacts with the backend layer built on the FastAPI framework via REST APIs using the HTTP protocol.

The backend layer handles the processes of image preprocessing, YOLOv5 inference, estimation of the percentage of corrosion, and response formation. Moreover, it maintains the history of detections and generates results in a structured form.

The YOLOv5 framework works as the core intelligence layer of the system and facilitates the detection of corroded regions using bounding boxes and confidence values. Further processing is carried out on these regions for estimating the severity of corrosion. Visualization Layer displays corrosion percentage, confidence graphs, bounding boxes, and repair recommendation.

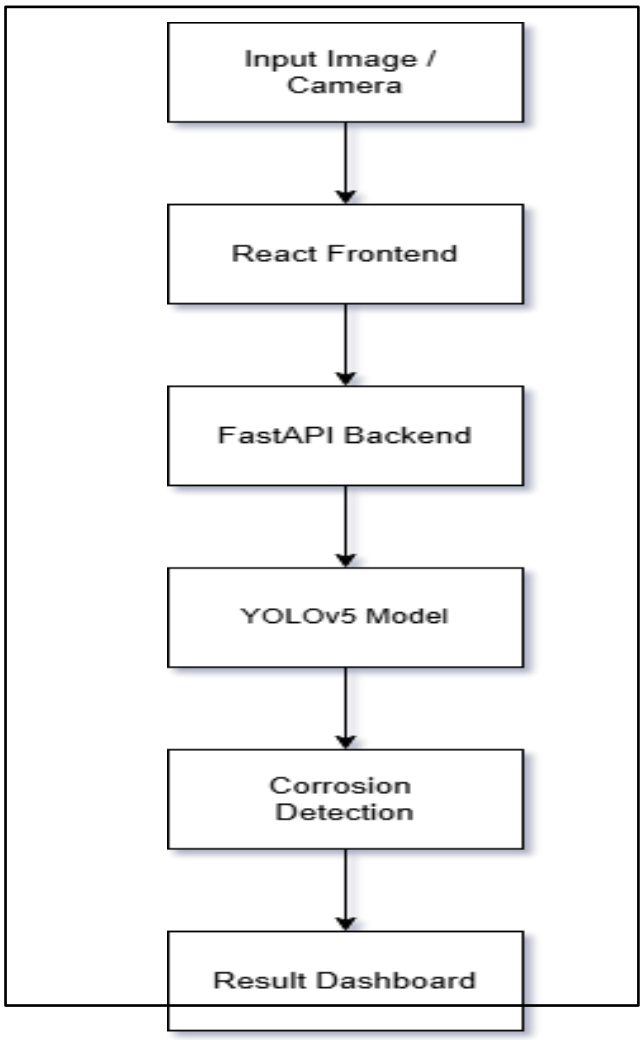

Fig. 1. System Architecture of YOLOv5-Based Corrosion Detection and Monitoring System

III. HARDWARE ARCHITECTURE

The system does not need specialized computing machines. The backend can be installed in any computer that supports Python and PyTorch; for real-time computation, one should consider using a GPU (for example, a desktop machine equipped with an NVIDIA GTX/RTX or a Jetson edge computer). YOLOv5 is capable of processing ~27 frames per second with TensorRT acceleration on an NVIDIA Jetson TX2[9]. Otherwise, the server will work without a GPU but with slower response times (probably 0.5–1s per frame). The frontend works on a normal PC/laptop/mobile phone with a built-in webcam. The only extra equipment that may be required is an Internet connection between frontend and backend. This architecture is depicted in Fig. 1 (see below): the communication takes place via

HTTP/HTTPS requests from users' devices to the central server running the FastAPI and YOLO services.

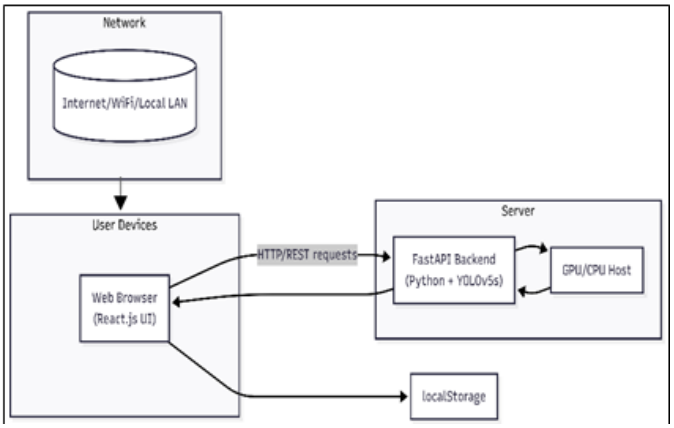

Fig. 2. Hardware Architecture

IV. SOFTWARE ARCHITECTURE

The software design is the classical pattern of web service [8]. User interaction and visualization is done via the React.js frontend. Main software components:

- **Input Components:** React provides support for file input (<input type="file">) and video stream capture (WebRTC APIs). State management can be used to send only the latest image/frame when a user initiates detection.
- **Canvas Overlay:** The frontend draws bounding boxes and labels on an HTML <canvas> over the image upon receiving detection results. This is done by looping through the detections array and using context.strokeRect() and the context.fillText() for drawing boxes and class names.
- **Charting and Alerts:** We use the recharts library to show bar charts of detection confidence scores. We also display the numeric corrosion_percent, and employ Bootstrap-like color coding (green/orange/red) to indicate severity.
- **History Logging:** Every detection (timestamp, percentage, thumbnail, JSON data) is saved in localStorage. It has an accordion UI that lets the user expand entries and see raw json if they want. When the user comes back later, the history is loaded from localStorage to keep the continuity.

Lightweight FastAPI backend service It exposes the /detect endpoint and does image I/O. The model is loaded once at startup to optimize throughput. FastAPI's concurrency model (ASGI) handles multiple simultaneous requests without blocking. CORS middleware is enabled to allow the React app (possibly on a different origin) to call the API. For simplicity no database, persistent storage is delegated to the client's browser.

This separation of front-end and back-end gives the system flexibility. The back-end, for instance, could be containerized and deployed in the cloud, and the React app could be hosted on a static site. Communication is plain JSON over HTTP, so it is easy to integrate with other components (e.g. alerting systems, databases).

Description of the YOLOv5 Architecture

YOLOv5 is an advanced version of the single-shot object detection algorithm that can detect objects in real time with a high level of accuracy and efficiency.

There are three main parts of the YOLOv5 architecture, which include:

- Backbone: The backbone component performs feature extraction on input images through the CSPDarknet53 architecture.
- Neck: This part combines features extracted at various scales by the use of PANet and SPPF modules to improve object detection capability.
- Head: It makes predictions concerning class names, confidence scores, and bounding box positions. The YOLOv5 architecture is perfectly suited for real-time industrial inspection systems.

V. WORKING PRINCIPLE

During execution time, the workflow process is as follows:

Image Acquisition: Either a still image or a video frame is chosen by the user. For live streaming, WebRTC technology (getUserMedia) supplies a steady camera stream[8].

Transmitting: The image is encoded in JPEG Blob format and transferred via HTTP POST request to /detect endpoint.

Preprocessing: The FastAPI server converts the byte stream into memory (BytesIO), constructs a PIL Image object, and transforms it into RGB. Then, the image is resized to 640×640 pixels (e.g., for YOLOv5). It maintains the image aspect ratio.

Inference with YOLO: The loaded YOLOv5s model detects objects. As a single-stage detector, YOLOv5 generates bounding box locations and confidence levels in just one pass[2]. PyTorch Hub library is utilized to simplify the process, and the following line of code model = torch.hub.load('ultralytics/yolov5', 'yolov5s') loads the pre-trained weights[3]. Once the model completes processing, results.pandas().xyxy[0] returns a dataframe containing detection results (xmin, ymin, xmax, ymax, confidence, class).

Post-processing: We filter the table with confidence >= 0.16. For each box left, we calculate its area. We add them all up to obtain the total area covered by corrosion. We divide the sum by the area of the picture and obtain the corrosion_percent. All detection information, as well as this number, is included in the JSON response.

JSON Response & Visualization: The JSON response is returned to the frontend where it will be used to visualize the results. On the React app page, all the detected boxes are overlaid onto the image, charts are updated with confidences, etc. If the corrosion percent reaches certain values (for example, >= 40%, which makes it "severe"), then an appropriate alert message appears to recommend repairs. Detection information is logged (saved into localStorage) together with the current timestamp (for example, Date.now()) and a thumbnail picture in base64 format.

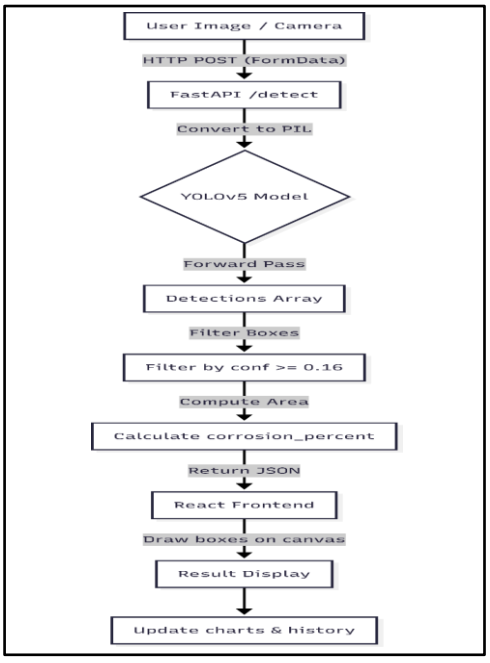

Fig. 3. Working Principle of The System

Since YOLOv5s takes tens of milliseconds to analyze a single image on a GPU[5], no significant lagging occurs on the side of the user (sub-second).

VI. RESULTS AND DISCUSSION

Performance evaluation was carried out using publicly available corrosion datasets and images that were collected individually. For instance, 6000 images of corrosion dataset (through augmentation) and Roboflow's corrosion dataset (with around 5500 images) were applied. The dataset was divided into a ratio of 80/20 training/test. Important performance indicators include mAP@0.5, precision, recall, IoU, and inference time. YOLOv5s (using torch.hub) attained mAP@0.5 of about 90%; larger models (YOLOv5).

A relationship between YOLO detection results and real corroded area was evaluated. Corrosion percent (corrosion_percent) is determined by the formula (sum of the bounding-box areas / image area) * 100%. A rectangular bounding box usually tends to exaggerate the real corroded area if corrosion is dispersed or overlapping. It may result in an increased value of the corrosion percent. Overlapping detections will also lead to double counting of pixels. Illumination or background disturbances like shadows or rust-like debris may result in false positives or negatives. To minimize those inaccuracies, non-maximum suppression and box merging were performed, and data augmentation such as mosaic transformation, flips, and color jitter was introduced.
The corrosion_percent values determine corresponding maintenance steps: 0% (no corrosion) requires no maintenance action; 1-14.9% (mild corrosion) indicates cleaning and priming; 15-39.9% (moderate corrosion) means abrasion sanding and applying heavy primer; 40%+ (severe corrosion) implies prompt metal repair or replacement.

Tests for robustness involved changing the angle, distance, and illumination of the camera. Robustness to moderate changes in light and scale was high; however, glare and occlusion led to a decrease in the level of precision. It would be beneficial to train the model with augmentations such as random brightness, contrast, and occlusions.

As a result of its asynchronous nature, FastAPI only serves one request per worker. Although FastAPI is optimized for high through put, it is recommended to deploy multiple asynchronous workers, batching frames, or using an inference server when operating under high loads. Using TensorRT or ONNX for conversion will help boost GPU inference throughput significantly. This architecture allows us to scale out horizontally.

The system consistently detects corrosion through automation, delivering quick and consistent findings along with maintenance recommendations. The method greatly minimizes the need for inspections. The future work will involve acquiring data from different fields for verification, incorporating segmentation models for better area estimation, and using hardware devices like GPUs or edge TPUs for real-time operations.

Advantages and Limitations

Advantages

- **Strengths:** Real-time monitoring: With YOLOv5 being a single-stage model, the inference process is highly efficient [5]. Field engineers can point their camera towards the target object and get immediate results with annotations displayed on their screen.
- **Scalability:** Decoupling the web architecture enables several clients and devices. The server-side application can be easily scaled horizontally (i.e.via Docker or Kubernetes). Adding a new location would mean launching another frontend instance without altering the core algorithm.
- **User-friendliness:** The web interface is designed to be simple enough for non-specialists. They can view clear
- **Visual Notifications** – alert icons (Green/Orange/Red depending on severity levels), bounding boxes, and bar charts. The accordion section with raw JSON data addresses tech-savvy users or those troubleshooting the system.
- **Cost-effectiveness:** There's no need for special equipment for end-users; any device that can access the internet and capture video footage is acceptable. Open-source libraries (e.g. PyTorch, YOLO, FastAPI, React) are used to minimize expenses.
- **Offline logging:** By using localStorage for keeping track of the detection log, users can access previous inspection reports without relying on the backend database.

Limitations

- Bounding Box Area Estimation: As already mentioned, rectangular bounding boxes may result in overestimation of the extent of corrosion. To get an exact estimation in terms of number of pixels, one needs a segmentation neural network (e.g. YOLOv8-seg or Mask R-CNN), which is beyond the scope of the assignment.
- Data Limitations: The size of the browser's local storage is capped at around 5 MB.[11] This effectively limits the number of entries to a few dozen. There are no login credentials required – all data is kept on one computer only.

- Deployment Considerations: As getUserMedia works only over HTTPS connection, a secure server must be used for field deployments of this application.[8] Otherwise, the camera will not work.
- Concurrent Requests Problem: The current approach relies on sequential inference. If there are many users simultaneously requesting analysis, CPU/GPU can become a bottleneck. Unless more concurrent processes are spawned, performance will suffer from high concurrency.
- Domain Shift: Ideally, the model should be retrained using corrosion samples for accurate detection. However, it may not work well on other materials or under completely different lighting conditions without

Future Scope

The proposed optimizations of the system include making it more scalable, fast, and accurate in detecting corrosion. One of the most important optimizations of the system includes cloud scaling of the FastAPI backend by deploying it using Docker in the cloud environment such as AWS or GCP that supports auto- scaling. With such deployment, the system would be able to process multiple requests concurrently while also balancing GPU and CPU loads. Another optimization is associated with implementing a real-time video stream using WebSockets instead of a single frame per request approach currently used by the system.

In order to make the corrosion detection more precise, the system can be supplemented with instance segmentation models such as YOLOv8-seg or DeepLab. While bounding-box detection is sufficient for locating corrosions, the use of segmentation will allow detecting corrosion boundaries accurately which results in more accurate corrosion area estimation. Finally, one can also consider the upgrade of AI models from the older ones to more advanced such as YOLOv8, YOLOv10, or even transformer detectors such as DETR.

Additional components that could be added to the platform include analytics tools and enterprise integrations. For instance, adding an enterprise-grade database like PostgreSQL or MongoDB would enable storage of inspection information, facilitate multi-device history tracking, and authenticate users. Additionally, the platform can be integrated with maintenance management solutions to streamline workflows by sending alerts or creating work orders every time significant corrosion is observed. Ultimately, implementing these changes would assist in turning the existing prototype into a full-fledged maintenance and inspection solution.

VIII. CONCLUSION

The proposed YOLOv5-based Corrosion Detection and Monitoring System successfully demonstrates the application of Artificial Intelligence, Deep Learning, and Computer Vision technologies for automated industrial inspection and structural health monitoring. The developed framework effectively addresses the limitations of conventional manual corrosion inspection methods by providing a faster, more accurate, and real-time detection solution. By integrating the YOLOv5 deep learning model with a React.js frontend and FastAPI backend, the system achieves seamless communication between image acquisition, inference processing, and result visualization modules.

The implementation of the YOLOv5s object detection model enabled efficient identification of corrosion regions from both uploaded images and live camera feeds. The single-stage architecture of YOLOv5 significantly reduced inference latency while maintaining reliable detection performance. The model successfully detected various corrosion patterns including rust patches, oxidation spots, and surface degradation areas under different environmental conditions. The corrosion percentage estimation mechanism further enhanced the system by quantitatively analyzing the affected structural area and classifying corrosion severity into multiple categories. Additionally, the web-based dashboard improved usability through real-time visualization of bounding boxes, confidence scores, severity indicators, and graphical analytics. A live camera detection module enabled constant monitoring, while a history of detections facilitated chronological monitoring of the deterioration of the structure's condition.

Several significant advantages of the proposed system compared to conventional inspections methods can be noted. Namely, the system provides lower dependency on skilled inspectors, lower susceptibility to human errors, shortened inspection periods, and consistent corrosion detection and estimation. Due to its modularity, the client-server approach enables easy scalability of the system and its application in various spheres, such as maritime industry, civil infrastructures, automobiles, factories, oil and gas pipelines, and airplane maintenance. Nevertheless, some issues were discovered during testing, including inaccurate bounding box-based detection leading to overestimating corroded areas and low precision in poor lighting, shadowed or reflective environments. Also, browser localStorage may provide limited options for storage of information. In summary, the proposed framework provides a solid base for the further development of intelligent systems for corrosion inspections.

REFERENCES

1. Q. Yu et al., “Enhancing YOLOv5 Performance for Small-Scale Corrosion Detection in Coastal Environments Using IoU-Based Loss Functions,” J. Mar. Sci. Eng., vol. 12, no. 12, 2024, Art. no. 2295, doi:10.3390/jmse12122295.[1][2]
2. C. Fu and M. Abisado, “An Integrated CNN, YOLOv5 and Faster R-CNN Framework for Real-Time Water Pipe Defect Detection,” Int. J. Adv. Comput. Sci. Appl., vol. 16, no. 10, 2025, pp. 151–157.[4]
3. S. Abid et al., “Advancing Structural Health Monitoring with Lightweight Real-Time Deep Learning-Based Corrosion Detection,” Stat., Optim., Inf. Comput., vol. 14, no. 6, pp. 3833–3856, 2025, doi:10.19139/soic-2310-5070- 2919.[6]
4. J. Nelson and J. Solawetz, “YOLOv5 is Here: State-of-the-Art Object Detection at 140 FPS,” Roboflow Blog, June 2020.[Online].Available: https://blog.roboflow.com/yolov5-is-here[5]
5. Ultralytics, ultralytics YOLOv5Architecture, YOLOv5docsavailable:https://docs.ultralytics.com/yolov5/tutorials/architecture_description/[12]
6. FastAPI Team, FastAPI Documentation (accessed 2025)[online],Available:https://fastapi.tiangolo.com/[13]
7. Mozilla Developer Network, MediaDevices.getUserMedia() Web API Documentation (accessed 2025). [Online]. Available:https://developer.mozilla.org/en-US/docs/Web/API/MediaDevices/getUserMedia[8]
8. Mozilla Developer Network (MDN), Web Storage — Storage Limits, Web API Documentation (accessed 2025). [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/Storage_API/Quota_management[11]