



AI-BASED HELMET DETECTION PROJECT USING YOLOv11 AND STREAMLIT

¹Sams Tabrez, ²Keshav Kumar Mishra, ³Sachin Verma, ⁴Sachin Kumar, ⁵Komal Kusum, ⁶Rishav Raj, ⁷Himanshu Ranjan Das

¹⁻⁶UG Student, Department of Electronics & Communication Engineering, Haldia Institute of Technology, Haldia, Purba Medinipur, West Bengal

⁷Department of Electronics & Communication Engineering, Haldia Institute of Technology, Haldia, Purba Medinipur, West Bengal

¹tabrezmtp@gmail.com, ²Keshavkumarmishra35@gmail.com, ³sachinverma78760@gmail.com,

⁴sachin141ece@gmail.com, ⁵komalkusum18062005@gmail.com, ⁶rishavraj30420@gmail.com,

⁷das.himanshu69@gmail.com

ABSTRACT

In this paper we studied on how Wearing a safety helmet is one of the simplest ways to protect a worker's head at a construction site or factory, but checking this that all worker is wearing helmet by hand is very slow, tedious, and often uncertain. This paper presents an automatic helmet detection system built on the YOLOv11 deep learning model, trained on a labelled image dataset collected from the Roboflow platform. On unseen images, the model is showing precision of 77.7%, a recall of 90.1%, and a mean Average Precision of 91.3%, And for processing a single image it takes almost 11.8 milliseconds. The trained model was connected with web dashboard Streamlit, so that a safety officer with no technical background can upload a photo and instantly see whether the site is safe or not. The results confirm that how an AI model is used in human safety purpose.

KEYWORDS: Helmet Detection, YOLOv11, Deep Learning, Streamlit Dashboard

1. INTRODUCTION

Construction sites, factories, and mines are the places where workers face problem like falling hard object from height, heavy machinery part etc. A safety helmet is one of the simplest and most effective ways to protect a worker's head, that is why safety rules is compulsory in all industries. In practice, however, checking whether every worker is wearing a helmet is usually done by a human guard walking around the site or watching a wall of CCTV screens. This kind of manual checking are boring and exhausted, a man can never watched each corner of site at a time and often misses violations of rules, especially when it comes on large and busy sites [1].

Nowadays using artificial intelligence that lets a computer "see" and understand the pictures of construction sites, earlier needed a programmer to manually write rules for colour and shape to detect an object, and these rules broke down easily

whenever the lighting or camera angle changed. So, using Deep learning models, especially Convolutional Neural Networks (CNNs), instantly learn these visual patterns directly from thousands of example images [2], which makes them far easier for a busy, unpredictable construction site.

The goal of this project is to build a complete, working system that looks at a photo from a work site and decides whether the person in it is wearing a helmet or not for worker safety.

2. LITERATURE REVIEW

Before the idea of deep learning, helmet detection was done by using simple image processing tricks. In this System first located a person's head and then checked the colour of that region using techniques such as a Histogram of Oriented Gradients (HOG) with a Support Vector Machine (SVM), or fitted circular shapes using the Hough Transform [1]. Sometimes this method facing problem due to

camera angle, lightening effect, shadows, and other brightly coloured objects [3-20].

A major problem came from large number of image's classification competitions such as ImageNet [3], deep CNNs like AlexNet [4]. For worker safety researchers work on the field of deep learning. Early deep detectors, known as two-stage detectors, first proposed a set of candidate regions and then classified each one separately; the Faster R-CNN model [5] improved this approach by sharing computed features between the two steps. Two-stage detectors are good, but their live video analyses is usually slow. To increase the speed of analysing, the idea of single-stage detectors came, which predict an object's location and class in a single pass over the image. The You Only Look Once (YOLO) family, first introduced by Redmon et al. [6] and later improved through versions such

as YOLO9000 [7], became very popular because it is both fast and more accurately.

3. METHODOLOGY AND SYSTEM DESIGN

3.1 System Architecture

The complete system has two main parts: a backend that runs the trained YOLOv11 model, and a Streamlit dashboard that makes the interface between user and screen. When a photo is uploaded, firstly it stored in a memory and then passed through the trained network in a single forward pass, and the network give returns back a list of bounding boxes with class scores. Overlapping boxes are then removed using Non-Maximum Suppression (NMS), and the surviving boxes are drawn back onto the original photo before it is shown to the user, as summarised in Figure 1.

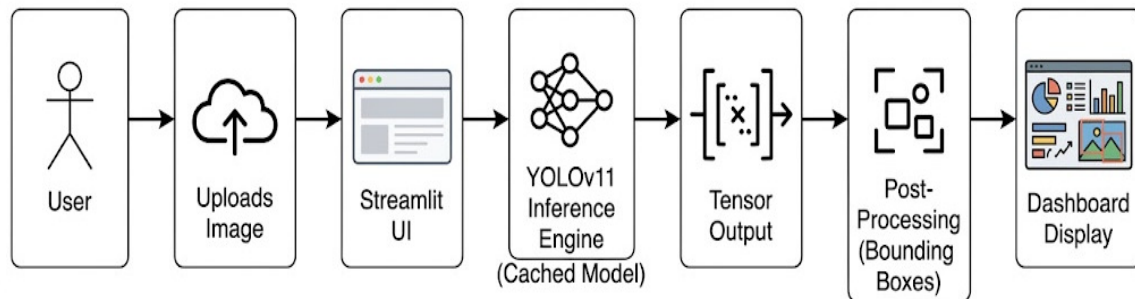


Figure 1: System Architecture Block Diagram.

Table 1. Dataset splitting strategy.

Partition	Number of Images	Percentage of Dataset
Training Set	760	86.36%
Validation Set	120	13.64%
Test Set	0	0.00%
Total	880	100%

3.2 Data Description and Source

A good dataset of images is the foundation of any deep learning project, so the images which is used here were collected from the Roboflow platform

under the project workspace helmet_nohelmet. The dataset has two classes: Class 0 for a person who is not wearing helmet, and Class 1 for a person who is wearing a helmet, taken from real construction sites and factory area. Out of 880

labelled images, 760 were used to train the model and 120 were kept aside for validation, as listed in Table 1. Analysing the data of validation set and makes sure the reported accuracy reflects that how the model performs on images it has never seen before.

3.3 Image Pre-processing and Multi-Scale Augmentation

The real camera images differ a lot in lighting and angle, every image was set to a fixed 640×640 pixels before being given to the network. Contrast Limited Adaptive Histogram Equalization (CLAHE) and random brightness changes were applied so that the model learns how to handle both excess sunlight and dark areas. A mosaic augmentation step was also used, which stitches

four training images into one single frame, forcing the model to notice precisely, distant helmets in addition to large, close-up ones.

3.4 System Data Flow and Logic Modelling

Before writing any code, two simple Data Flow Diagrams (DFDs) were used to plan the logic of the system. The Level 0 diagram in Figure 2 treats the whole system as a single process and shows that a Site Safety Supervisor uploads an image and receives an annotated image with a safety status in return. The Level 1 diagram in Figure 3 breaks this single process into smaller steps: an upload module, the YOLOv11 inference engine, analytics and rendering module, and an export module, which made it much easier for the team to divide the work between members.

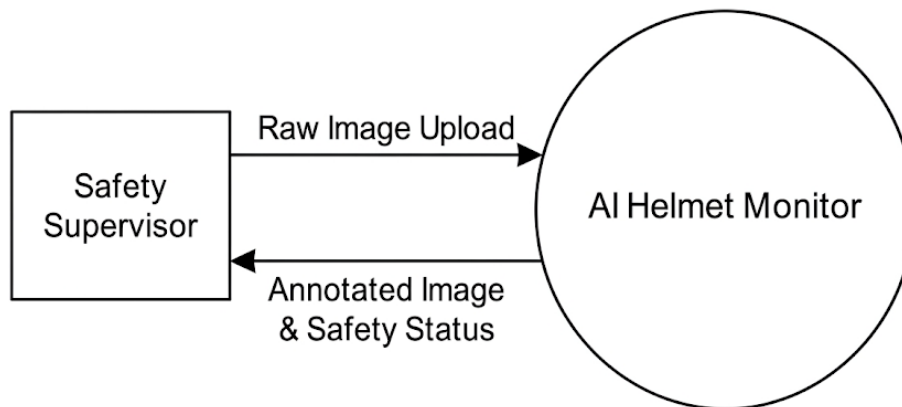


Figure 2: Context-Level Data Flow Diagram (DFD Level 0).

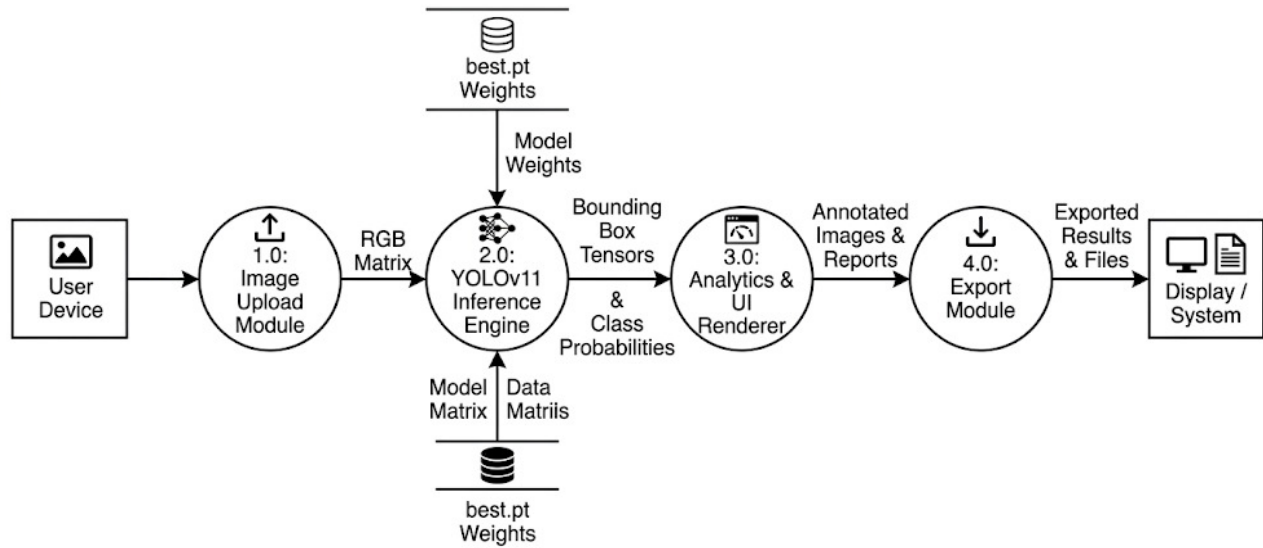


Figure 3: First-Level Data Flow Diagram (DFD Level 1)

3.5 Underlying Algorithm: YOLOv11 Architecture

The detection model of this project is based on YOLOv11, the newest member of the YOLO family. YOLOv11 treats detection as a single, direct regression problem, which is exactly what allows it to run in real time. Its backbone, shown in Figure 4, uses lightweight C3k2 blocks that keep the amount of computation low while still extracting strong features for small and medium-sized objects such as a helmet. The model learns to draw a tight, accurate box around each detected object by minimising a Complete Intersection over Union (CIoU) loss together with a Distribution Focal Loss (DFL); together, these make the predicted box edges sharper, even when a helmet is partly hidden behind scaffolding.

3.6 Tools and Technology Stack

The project was built entirely with free, open-source tools, which are listed in Table 2. Python was the programming language used throughout, the model itself was built and trained with PyTorch through the Ultralytics YOLO package [19], OpenCV handled routine image operations, and Streamlit was used to build the web dashboard. Training was carried out on an NVIDIA Tesla T4 GPU on a cloud notebook, which let the full 60-epoch training run finish in about fifteen minutes. Streamlit's caching feature was especially useful, since it keeps the large trained model loaded in memory instead of reloading it every time a new photo is uploaded.

Table 2: Tools and Technology Stack

Component	Technology Used
Programming Language	Python 3.12
Deep Learning Framework	PyTorch & Ultralytics
Computer Vision Library	OpenCV (cv2)
Frontend UI Framework	Streamlit
Matrix Processing	NumPy & PIL
Hardware Accelerator	NVIDIA Tesla T4 GPU
Code Editor / IDE	Jupyter Notebook & VS Code

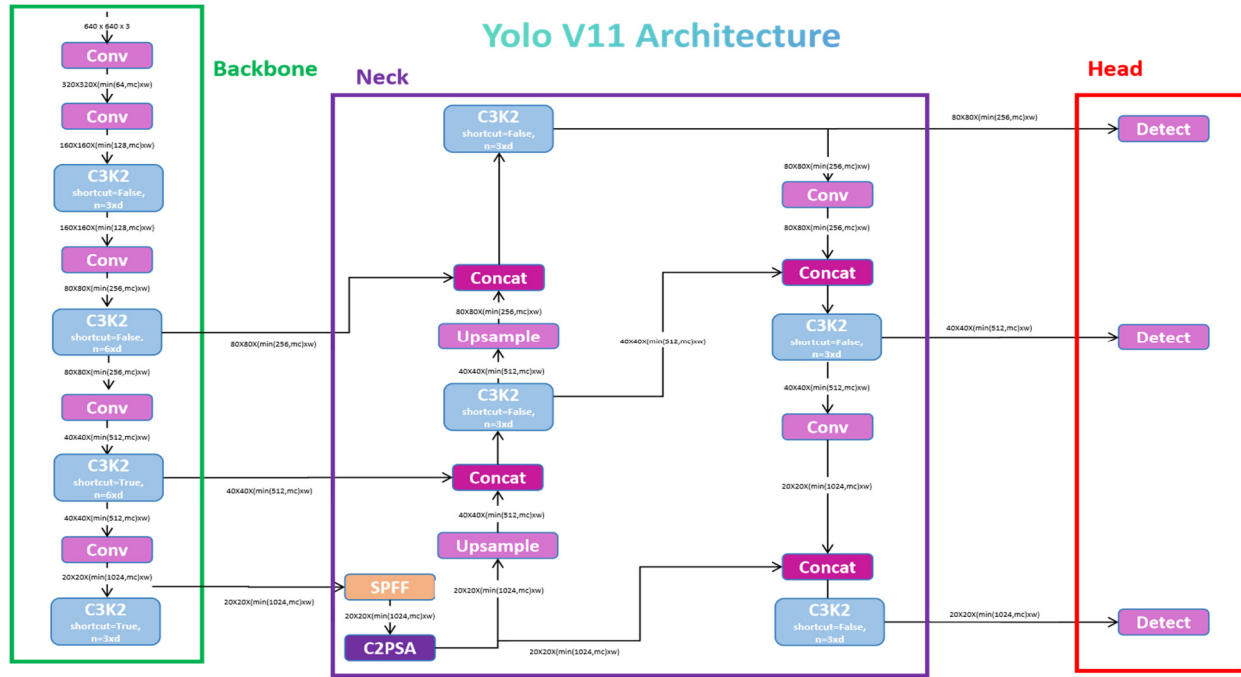


Figure 4: YOLOv11 Backbone, Neck and Head Architecture.

3.7 YOLOv11 Architecture

The YOLOv11 architecture is divided into three main parts: **Backbone**, **Neck**, and **Head**. The Backbone extracts important features from the input image using convolutional layers and special blocks that capture patterns at different scales. The Neck then refines and combines these features through up sampling and concatenation, making sure information from multiple resolutions is preserved. Finally, the Head performs the actual object detection at three different scales, allowing the model to identify both small and large objects effectively. In simple terms, YOLOv11 first learns useful details from the image, then enhances them, and finally detects objects with high accuracy.

4. IMPLEMENTATION

4.1 Environment Setup and Hardware Configuration

Model training was carried out on a cloud GPU instance fitted with an NVIDIA Tesla T4 card, whose availability was first checked using the standard NVIDIA system tool before training

began. The core Ultralytics package was then installed to provide all the YOLOv11 training and inference commands used through the rest of the project.

4.2 Dataset Acquisition via Roboflow API

Instead of manually downloading and unzipping files, the labelled dataset was pulled directly into the training notebook using the Roboflow Python API. This automatically downloaded the annotated images together with a data.yaml configuration file that tells the training script the class names and file locations to use.

4.3 Model Training Implementation

Training was started from the pre-trained yolo11n.pt weights so that the network did not have to learn basic shapes from zero, a common and effective trick known as transfer learning. The model was then trained for 60 epochs at an image size of 640x640, after which the best-performing weights were automatically saved for later use inside the dashboard.

4.4 Streamlit Dashboard Implementation

The dashboard was built with Streamlit because it turns a normal Python script into an interactive web page with very little extra code. The trained model is loaded once and cached in memory, an uploaded photo is passed to the model, and the returned bounding boxes are drawn back onto the image so the user can view the original and the annotated result side by side.

4.5 System Interface and Operational Walkthrough

Figure 5 shows the landing page of the dashboard, where a user can read a short project description and upload a photo for checking. Figure 6 shows the detection screen: the original photo appears on the left and the annotated result, complete with bounding boxes and confidence scores, appears on the right, together with a clear status card underneath so that even a non-technical viewer can understand the result immediately.

5. RESULT ANALYSIS

5.1 Training Loss Convergence

The model was trained for 60 epochs, and three kinds of loss were tracked throughout: box loss (how well a predicted box matches the true one), class loss (how well the model tells a helmet apart from a bare head), and distribution focal loss (how sharp the predicted box edges are). Table 3 lists these values at sample epochs, and Figure 7 plots them across the complete run. As training

progressed, all three losses steadily decreased while the mAP score climbed upward, which is exactly the pattern expected from a model that is genuinely learning rather than memorising the training images.

5.2 Performance Metrics (Precision, Recall and mAP)

On the 120 unseen validation images, the final model reached an overall precision of 77.7%, a recall of 90.1%, an mAP@0.5 of 91.3%, and an mAP@0.5:0.95 of 63.1%. The high recall value matters most for a safety tool, since it means the system rarely misses a worker who is actually present in the frame, keeping false "all-clear" signals to a minimum. Table 4 breaks these numbers down by class: the Helmet class, which had far more training examples, reached a strong 94.0% mAP@0.5, while the smaller No-Helmet class still reached a useful 88.7% mAP@0.5 even with only ten validation examples.

5.3 Confusion Matrix Evaluation

Figure 8 shows the confusion matrix, a simple grid comparing the model's prediction against the true label. Around 90% of both helmet and no-helmet cases landed on the correct diagonal cell, and most of the remaining mistakes were the model predicting "background" instead of a real object, rather than confusing a helmet with a bare head. This is a reassuring result, since it means the two safety classes are rarely swapped with one another.

Table 3: Training Performance Convergence Metrics Across Epochs

Epoch	Box Loss	Class Loss	DFL Loss	mAP@0.5	mAP@0.5:0.95
1	1.2510	2.6680	1.3880	0.2110	0.1180
10	1.2760	1.1450	1.3730	0.8320	0.4520
20	1.1100	0.8362	1.2590	0.8530	0.5190
30	1.0220	0.7013	1.2130	0.8970	0.5610
40	0.9612	0.6191	1.1650	0.9200	0.5730
50	0.8609	0.5340	1.0970	0.9240	0.6220
60	0.7704	0.4180	1.0330	0.8820	0.5970

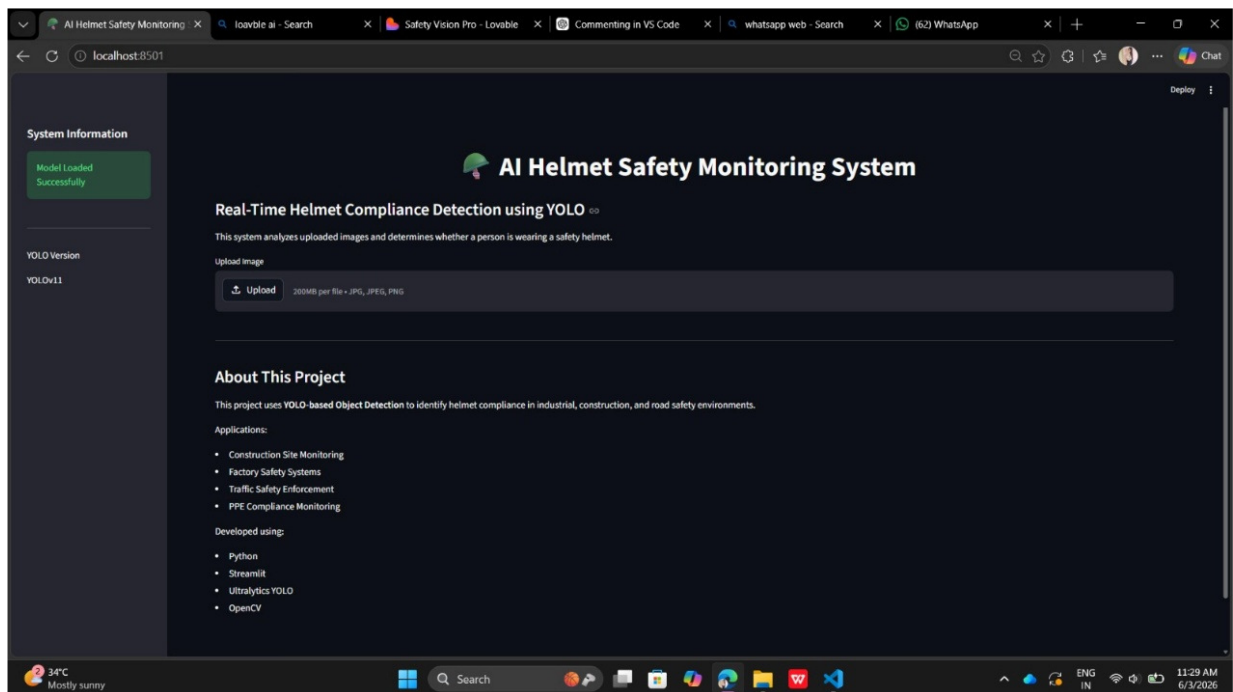


Figure 5: Dashboard Landing Page.

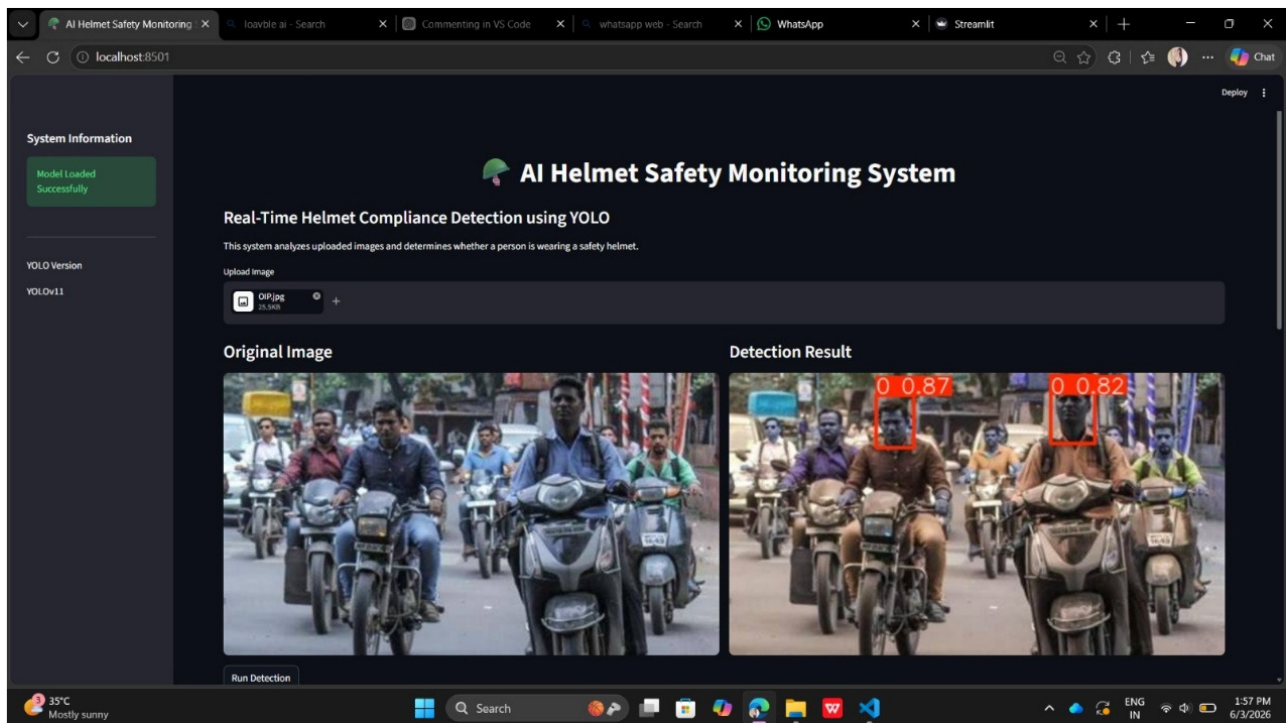


Figure 6: Real-Time Detection Result View.

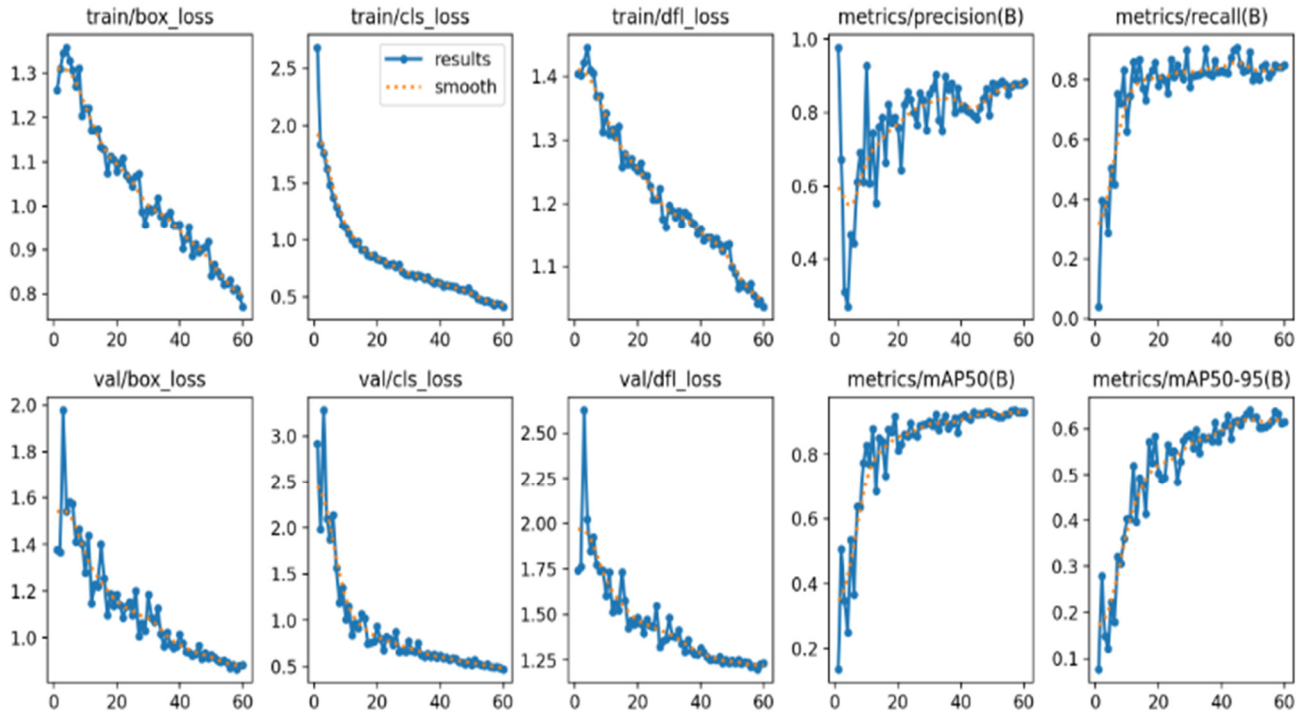


Figure 7: Training and Validation Loss Curves Across 60 Epochs.

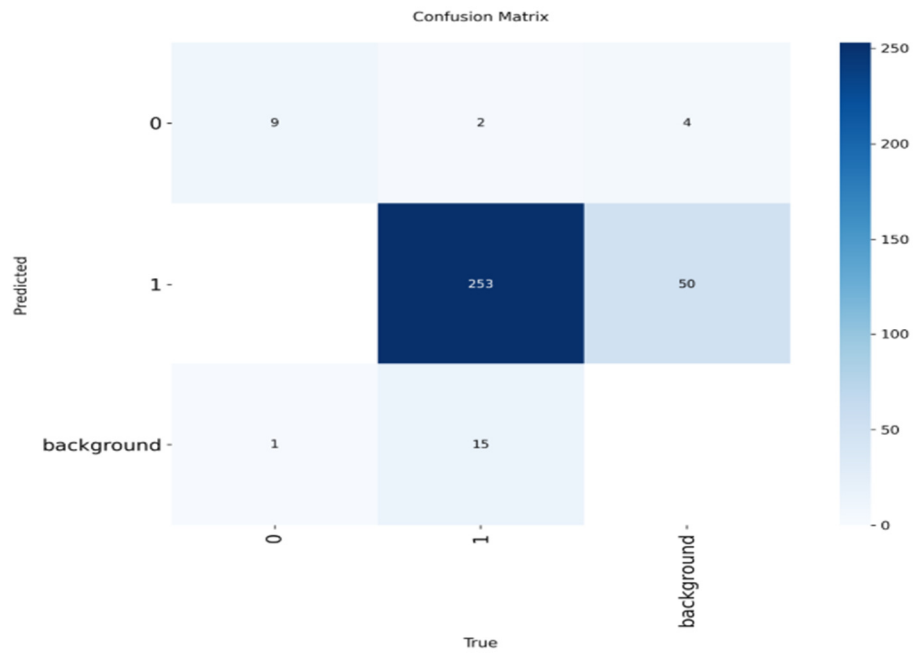


Figure 8: Confusion Matrix on the Validation Set.

Table 4: Class-Wise Validation Performance Metrics

Object Class	Instances	Precision	Recall	mAP@0.5	mAP@0.5:0.95
Class 0 (No Helmet)	10	0.642	0.898	0.887	0.663
Class 1 (Helmet)	270	0.913	0.904	0.940	0.599

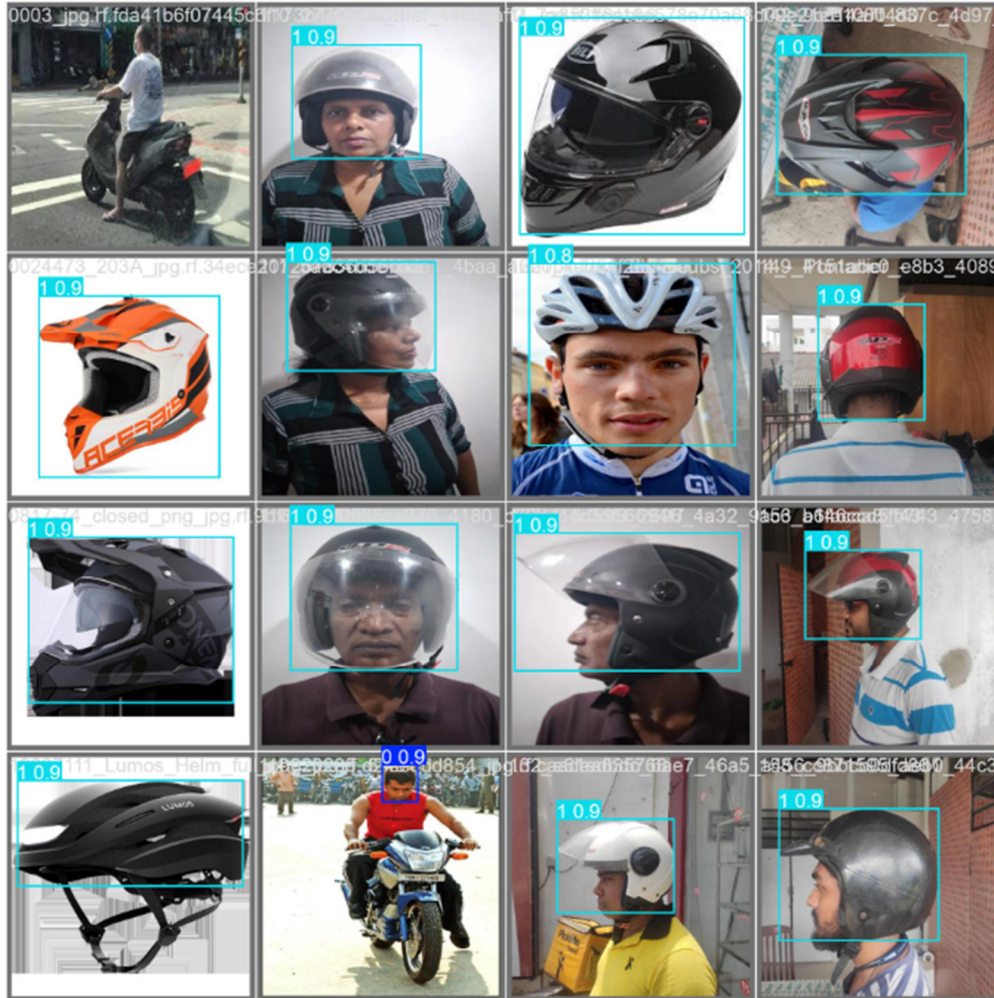


Figure 9: Sample Helmet Detection Results (0: No Helmet, 1: Helmet).

Table 5: Processing Latency Breakdown (Per Frame)

Pipeline Phase	Execution Time
Image Pre-processing	0.3 ms
YOLOv11 Deep Convolutional Inference	5.2 ms
Post-processing (NMS)	6.3 ms
Total Pipeline Latency	11.8 ms

5.4 Detection Result Samples

Figure 9 shows a sample of test images with the model's predicted boxes drawn on them. The system correctly located helmets of different colours and shapes, on people at different distances from the camera, and even inside a busy traffic scene, showing that it generalises reasonably well beyond simple, posed photographs. Therefore, from the pictures shown in the figure, it is clear that utmost detection is possible using the described techniques.

5.5 Inference Latency and Computational Benchmarks

Real-time video needs a processing speed faster than 30 frames per second, which means less than about 33 milliseconds per frame. Table 5 shows that the trained model, with only about 2.6 million parameters, completed the full pipeline (pre-processing, inference, and box clean-up) in around 11.8 milliseconds per image on the Tesla T4 GPU - roughly 85 frames per second, well within the real-time requirement.

5.6 Failure Mode Observations

A few edge cases were also noticed during testing. Workers who were very far from the camera

sometimes occupied too few pixels for the model to notice them, and a helmet covered more than about three-quarters by a pipe or another worker was occasionally missed altogether. Strong glare from direct sunlight or a welding arc also sometimes washed out the colour of a helmet enough to push the model's confidence score below the detection threshold.

6. CONCLUSION AND FUTURE WORK

This project developed and successfully tested a real time helmet detection system using YOLOv11. It also worked very fast for live video, at around 85 frames per second. This model is connected with streamlit dashboard. This helps for user who didn't have any coding background to check whether workers are wearing helmet or not and also this model reduces the time for checking construction sites. It has one drawback that it's difficult to detect when object is far from the camera, in crowd and when there is strong light or glare. Overall, this project shows that a lightweight deep learning model with a simple web interface can provide a fast and adorable solution for safety monitoring at construction sites.

REFERENCES

- [1] K. Dahiya, D. Singh and C. K. Mohan, "Automatic detection of bike riders without helmet using surveillance videos in real-time," in Proc. IJCNN, Vancouver, BC, Canada, July 2016, pp. 3046-3051.
- [2] Y. LeCun, Y. Bengio, and G. Hinton, "Deep Learning," *Nature*, vol. 521, no. 7553, pp. 436-444, May 2015.
- [3] J. Deng, W. Dong, R. Socher, L. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in Proc. IEEE CVPR, Miami, FL, USA, 2009, pp. 248-255.
- [4] A. Krizhevsky, I. Sutskever, and G. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in Advances in NIPS 25, Lake Tahoe, NV, USA, 2012, pp. 1097-1105.
- [5] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," *IEEE TPAMI*, vol. 39, no. 6, pp. 1137-1149, June 2017.
- [6] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," in Proc. IEEE CVPR, Las Vegas, NV, USA, 2016, pp. 779-788.

- [7] J. Redmon and A. Farhadi, "YOLO9000: Better, Faster, Stronger," in Proc. IEEE CVPR, Honolulu, HI, USA, 2017, pp. 6517-6525.
- [8] Tushar Kale, et al., "Helmet Detection and Number Plate Recognition," IJRASET, vol. 11, issue V, pp. 4038-4041, May 2023.
- [9] Manisha Pise et al., "Helmet and Number Plate Detection Using YOLOv5," IJARST, 2024.
- [10] N. Gosavi, et al., "Helfine: Real-Time Helmet Violation Detection with Automated Fine System," IJISRT, vol. 10, issue 2, 2025.
- [11] Hari Krishnan, R., "Enhancing Road Safety with Real-Time Helmet Detection and Challan Issuance Using YOLO and OCR," IRJMETS, 2024.
- [12] J. Majumder, "Real-time YOLO-based monitoring for helmet and number plate detection," 2022.
- [13] D. Singh, C. Vishnu and C. K. Mohan, "Visual big data analytics for traffic monitoring in smart city," in Proc. IEEE ICMLA, Anaheim, CA, USA, 2016, pp. 18-20.
- [14] A. S. Kapse, S. Shreevamshi, P. Ravichandra, R. Reddy, and R. Reddy, "A Survey on Helmet Detection by CNN Algorithm," ITM Web of Conferences (ICDSAC 2023), vol. 56, 05004, 2023.
- [15] P. Tripathi, et al., "A Review on Helmet and Number Plate Detection," IJRPR, vol. 4, no. 5, pp. 2473-2484, May 2023.
- [16] P. Ahuja and K. Bhavsar, "Microcontroller Based Smart Helmet Using GSM & GPRS," in Proc. ICOEI, Tirunelveli, India, 2018, pp. 917-921.
- [17] M. Uniyal, H. Rawat, M. Srivastava and V. K. Srivastava, "IOT based Smart Helmet System with Data Log System," in Proc. ICACCCN, Greater Noida, India, 2018, pp. 1-6.
- [18] M. Jeong, H. Lee, M. Bae, D. B. Shin, S. H. Lim and K. B. Lee, "Development and Application of the Smart Helmet for Disaster and Safety," in Proc. ICTC, Jeju, Korea, 2018, pp. 1060-1062.
- [19] Jocher, G., Stubs, F., et al., "Ultralytics YOLO" (Computer software), GitHub.
- [20] Y. Wu, M. Chen, G. Li, W. Wang, and G. Zhang, "A comprehensive survey on the application of deep learning in intelligent transportation systems," IEEE Access, vol. 9, pp. 32812-32837, 2021.