OshaSight: YOLOv8-based Real-Time Safety Monitor for PPE Violations and Smoking Behavior

Gokhan Ceylan · Lorenzo Porcelli
ACSAI, Sapienza University of Rome, Italy

Introduction

Workplace safety is a critical concern in industries such as construction and manufacturing, where failure to wear Personal Protective Equipment (PPE) or engaging in unsafe behaviors (e.g. smoking in prohibited areas) can lead to accidents. Traditional safety compliance checks (e.g. manual supervision or periodic audits) are labor-intensive and prone to human error. Which has led to the development of vision-based safety monitoring systems that use cameras and AI to detect PPE compliance and safety violations in real time.

Researchers have applied YOLO-based models to various safety gear detection problems. Hard hat and vest detection has been demonstrated using YOLOv3/v4 in construction sites, with one study (Nath et al., 2020) achieving ~72.3% mAP (mean Average Precision) for helmet/vest detection using a custom dataset of 1,500 images. Efforts thereafter having used larger datasets (e.g. the CHV dataset of six classes including helmets and vests) found that YOLOv5x could reach 86.55% mAP on PPE classes, with lighter models like YOLOv5s processing images at ~52 FPS. Beyond PPE, vision systems have tackled behavior violations such as smoking detection. As such, research with an enhanced YOLOv8 model reported about 85.9% mAP on a custom smoking behavior dataset, a significant improvement in comparison to earlier YOLO-based smoke detectors.

Method

Dataset Preparation and Annotation

Each class in OshaSight is trained on its own dedicated dataset, tailored to detect the presence of that particular PPE item or violation. To prepare the data for YOLOv8, all annotations were already downloaded in the YOLO format, besides mask data. For the face mask detector, the Kaggle dataset provided XML annotations in Pascal VOC format (with classes with_mask, without_mask, etc.). We wrote a custom conversion script to parse these XML files and generate YOLO labels. Using Python's ElementTree we extracted each object's bounding box (xmin, ymin, xmax, ymax) and image dimensions, then computed the YOLO normalized center coordinates (x_center, y_center) and box size (width, height) as fractions of image width/height. In the mask dataset's case there is only one class (with_mask, treated as class 0) for detection, since a person not wearing a mask will simply produce no detection. We split the mask dataset into train/val/test sets (approximately 80%/10%/10% by random selection). The conversion script automatically distributed the image files and label files into train/, valid/, and test/ directories accordingly.

Class	Train	Val	Test
Mask	690	77	86
Helmet	1850	77	48
Vest	676	66	33
Gloves	1018	102	50
Goggles	2571	121	246
Smoke	3852	37	50

Table 1: Train/Validation/Test split for each class.

For the other classes (helmet, vest, goggles, gloves, and smoke), datasets were obtained via Roboflow Universe in YOLO format. The helmet dataset (origin: "kask tanima") consisted of images with hardhats against no hardhats, labeled as one class "helmet" for those with hardhats. The high-visibility vest dataset contained only the class "vest" on persons wearing reflective vests. Each dataset had its own YAML configuration file specifying the paths to train/val/test directories and the single class name. Using distinct datasets per class simplified the annotation process (no need to annotate all classes in every image) but means that images without a given PPE item implicitly serve as negative examples for that item's model.

Model Training with YOLOv8

We trained one YOLOv8 model per class, opposed to a single multi-class model. This design choice was made to suit the different data sources.

We chose the YOLOv8-s architecture from Ultralytics as a good balance between speed and accuracy. Training was done using the Ultralytics YOLOv8 API. For each class, we initialized a model with pre-trained weights, yolov8s.pt (trained on COCO), to leverage transfer learning. We then tuned it on the specific PPE dataset. Training hyperparameters were kept consistent across all models: image size = 640 pixels; batch size = 8; epochs = 50. We used the default YOLOv8 training settings unless otherwise noted.

During training, YOLOv8 reports metrics on the validation set at each epoch. We monitored the mean Average Precision (mAP@0.5) and the mAP@0.5:0.95 for the target class. The best model checkpoint (determined by highest mAP on val) was saved as best.pt. All six models converged within 50 epochs, with training times ranging from ~1 to 3 hours per model on our hardware. Despite each model only having to learn one class, the variability in lighting, angles, and backgrounds in the datasets required augmentation to avoid overfitting. We observed the Mask and Helmet models reaching high precision quickly (their datasets have clear features). By contrast, the Gloves model, trained on a moderate-sized set (~1170 images), and the Goggles model despite having the largest dataset (~2900 images), still showed noisier learning curves and required careful tuning of learning rate and augmentation to stabilise performance. The diminishing gains beyond epoch 40 confirm that 50 epochs strike a good balance between accuracy and training time.

For the Smoke detector the mAP@0.5 climbed rapidly, from roughly 0.55 after the 1st epoch to about 0.97 by the 20th, then edged up to 0.991 at epoch 50 before flattening out. The steep early gain and later plateau show that running the full 50 epochs captures the final few percentage points of accuracy, while training beyond that would deliver minimal return.

Real-Time Inference and GUI Interface

At runtime, OshaSight runs all the trained detectors in parallel on each frame from a webcam feed (in practice, the models are invoked sequentially per frame, as described below). We built a simple graphical interface using PySimpleGUI to display the video feed and allow the user to toggle which detections are active. The GUI (shown in Figure 1) consists of a video display panel on the left and a control panel on the right. The control panel contains a framed checklist of "Toggle Classes" (six checkboxes for Mask, Smoke, Gloves, Helmet, Goggles, Vest) and a table listing each class with an ID and a short description or requirement (for example, "Helmet – Head protection").

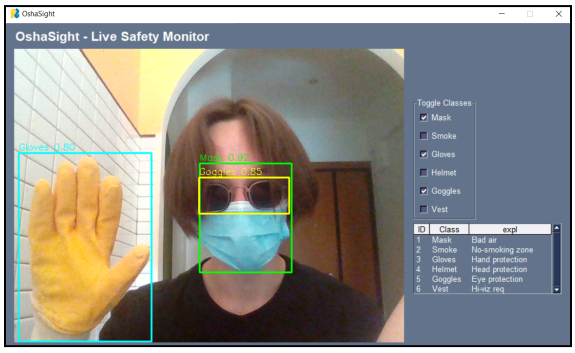

Figure 1: OshaSight GUI showing the live webcam feed and detection overlay.

Here, the system has detected "Goggles", "Mask", and "Gloves" (indicated by yellow, green, and cyan boxes respectively). The right-side panel allows enabling/disabling each class, and provides a brief explanation of each safety item. The application uses OpenCV to capture frames from the default webcam at ~640×480 resolution. Each frame is then passed through the YOLO models. We maintain a dictionary of loaded YOLO models (one for each class, each loaded from its best.pt weights) upon startup. For each video frame, the program iterates through each class/model pair, checks if its checkbox is enabled in the GUI, and if so, runs model.predict(frame) with a confidence threshold specific to that class. Based on validation results, we set custom confidence thresholds (e.g. 0.85 for Mask and Smoke, 0.50 for Helmet/Vest, etc.) to balance precision and recall for each detector. Thereby, classes with lower-performing models or small objects required lowering the threshold, whereas classes like Mask had a high threshold. The model's predictions (preds) are a list of bounding boxes with confidence scores. We filter out any boxes below the threshold (though the conf parameter in model.predict already helps) and then draw bounding boxes and labels on the frame using OpenCV. Each class has a distinct color for its bounding box (defined in a dictionary, e.g. green for Mask, red for Helmet, etc.) to make it easy to differentiate on video. The label text includes the class name and confidence (e.g. "Goggles 0.85"), drawn above the box.

Each enabled model processes the frame to detect its target object. Detections are

drawn on the frame, and if a cigarette is detected, an alert is triggered.

Whenever the Smoke model finds a cigarette in the frame (the smoke_detected flag is set), the system triggers a warning. We use a short PowerShell script call to play a cautionary WAV audio file in a separate process to avoid freezing the main loop while the sound plays. In parallel, the GUI overlay is changed: the video frame is tinted red (by blending a red transparent layer); a large "⚠ NO SMOKING" text is superimposed in bold red font at the top of the video view. This visual warning remains on-screen for a few seconds. We implemented a cooldown such that the alarm and overlay activate at most once every 3 seconds even if smoke is continuously detected, as a means to prevent excessive audio spam; the red overlay persists up to 5 seconds from the last detection, giving a clear indicator of the violation.

All of the following operations: model inference, drawing, and GUI update occur within the loop that runs approximately 10 to 15 times per second.

Implementation Details

OshaSight is implemented in Python and was tested on a Windows 10 machine. The PySimpleGUI framework proved convenient for rapidly creating the interface. However, it updates the image in the main thread, so maintaining a high frame rate requires careful tuning. We used window.read(timeout=1) to keep the GUI responsive to close or toggle events without a separate thread. The YOLOv8 models were run on an NVIDIA GPU (GeForce RTX-series laptop GPU) via the Ultralytics library, which defaults to using CUDA if available. This kept inference efficient despite our multiple models. When all six models are enabled, the system effectively runs six forward passes per frame (one per model). To optimize throughput, one could consider running models in parallel threads or combining them into one, but in our implementation, sequential processing was sufficient for a moderate frame rate.

Results

Model Performance: Each of the six YOLOv8 models achieved reasonable detection accuracy on its respective test set, though performance varied with the amount of training data. Smoke and Goggles are the top performers, followed by Gloves and Helmet, reflecting their larger and cleaner datasets, while Mask and Vest trail slightly due to smaller sample sizes and greater visual variability. The Mask reached about 90.5% mAP@0.5 (61.9% at 0.5-0.95) on the test images, meaning it correctly identifies masked faces with high precision. The Helmet model also performed well at approximately 91.4% mAP@0.5 (81.0% at 0.5-0.95), as the helmet images were plentiful and distinctive. The Smoking detector achieved around 99.1% mAP@0.5 (87.0% at 0.5-0.95); most test images of people smoking were correctly flagged, though a few were missed if the cigarette was too small or not visible. Models for Gloves, Goggles, and Vest had lower but still usable accuracy, which were in the range of ~92.1% mAP@0.5 (61.5% at 0.5-0.95). For instance, the Gloves model

reached roughly 94.8% mAP@0.5 (70.1% at 0.5-0.95). It could usually spot brightly colored gloves on hands, but struggled with thin or skin-toned gloves, and the Goggles has high precision but recall issues are still possible despite its 98.9% mAP@0.5 (73.0% at 0.5-0.95) on its meagre dataset (it sometimes missed or confused eyewear).

Class	mP	mR	mAP @0.5	mAP @0.5-0.95
Mask	0.935	0.851	0.905	0.619
Gloves	0.919	0.925	0.948	0.701
Helmet	0.987	0.773	0.914	0.810
Goggles	0.983	0.981	0.989	0.730
Vest	0.980	0.782	0.921	0.615
Smoke	0.977	0.980	0.991	0.877

Table 2: mean Average Precision (mAP) scores for each class.

These mAP values confirm that more data or augmentation would help, but even at current levels the detectors can pick up most instances of their target objects when above confidence thresholds. We calibrated the confidence thresholds per class to ensure that, in practice, each model's precision was high. For example, the Goggles detector only reports a detection if it's fairly sure (≥0.80 confidence), otherwise it stays quiet to avoid false positives.

Real-Time Detection and Speed: During live webcam tests, OshaSight was able to detect PPE violations in real time with minimal latency. On a machine with an Intel Core i7-10750H CPU and an NVIDIA GeForce GTX 1660 Ti Laptop GPU, the per-frame processing time with all detection classes enabled was approximately 75 to 90 ms. This breaks down to roughly 12 to 15 ms inference per YOLOv8s model (for a 480×640 frame) plus some overhead for image processing and GUI rendering. In other words, the system ran at about 11 to 13 frames per second with all six detectors active. This frame rate is sufficient for monitoring a static camera scene for safety violations, though fast-moving violations might be missed if the frame rate drops. If fewer classes are toggled, the frame rate rises almost linearly; with only three models running, we measured roughly 25 to 30 FPS. The latency between an event (e.g. person removing their hardhat) and on-screen detection was primarily just one frame (on the order of 0.1s), plus the 0-3s intentional delay for audio alerts on repeated smoke detection. The webcam feed itself was the main source of any slight lag (standard webcam capture buffering ~30 ms). We did not observe the GUI becoming unresponsive as user inputs like unchecking a box would take effect on the next frame cycle, typically within 0.1 to 0.2 seconds.

The on-screen results were intuitively visualized. Figure 1 above shows a sample detection: the user on the right is wearing safety glasses which were correctly detected as "Goggles" (yellow bounding box and label). In another test, when the user wore a surgical mask, the Mask detector drew a green box around the face with the label "Mask". When a cigarette

was brought to the mouth, the Smoke model highlighted it with a blue box and the NO SMOKING alert flashed on the screen with a loud buzzer sound. This demonstrates the system's capability to not only detect issues but actively warn about critical violations like smoking. We also tested multiple objects simultaneously: for example, a user wearing a helmet and vest at the same time. Both the Helmet and Vest models fired and drew their respective boxes (red for helmet, magenta for vest) without interference. The independence of models means detections do not conflict, hence they can overlap if objects are close, but since each is labeled, it remains clear.

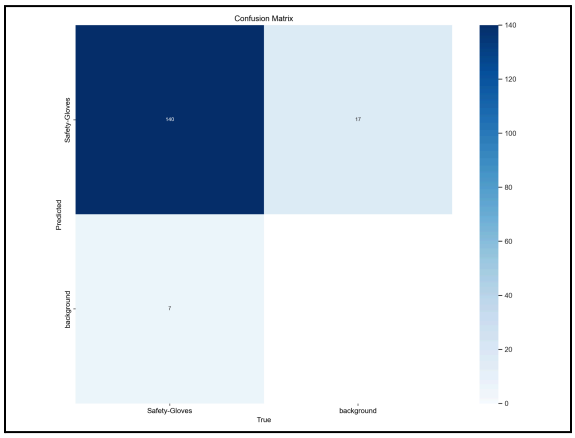

Figure 2: Confusion matrix for Gloves.

False Positives/Negatives: We encountered relatively few false positives during testing, thanks to the tuned confidence thresholds. Nonetheless, some occurred. The Gloves detector occasionally produced a false positive (as seen in Figure 2) when it saw a hand or skin patch with a color similar to the gloves it trained on (e.g. a hand near a white sleeve might trigger a low-confidence glove detection). Similarly, the Goggles model at times misidentified ordinary eyeglasses or even a reflection on a cabinet door as "Goggles" when the threshold was set too low. By raising the threshold to 0.80, we eliminated most of those false detections at the expense of missing a few true positives in difficult conditions. The smoke detector was generally precise for actual cigarettes; however, it could miss a cigarette if the person's hand fully covered it or if the lighting was poor (false negative). In one trial, the model also triggered what turned out to be a small thin marker pen held near the mouth, which is an understandable confusion given the object's shape and position mimicked a cigarette. These issues highlight the trade-offs in detection sensitivity.

GUI and System Behavior: The PySimpleGUI interface proved user-friendly. The toggle buttons worked reliably to turn certain detections on or off on the fly. The explanation table helped interpret the classes for non-technical users, e.g. if "Vest" is detected, the table reminds that a vest is required for high visibility ("Hi-viz req"). We did notice that rapidly toggling options or resizing the window could briefly pause the video, but it recovered quickly. The program's memory footprint remained modest (each YOLO model is ~20 MB), and the CPU usage stayed low thanks to GPU offloading of inference. One practical issue was that the application did not implement a graceful exit on window close.

Sources:

Bochkovskiy, A., Wang, C.-Y., & Liao, H.-Y. M. (2020). *YOLOv4: Optimal speed and accuracy of object detection* (arXiv:2004.10934) [Preprint]. arXiv. https://arxiv.org/abs/2004.10934

Ferdous, M., & Ahsan, S. M. M. (2022). PPE detector: A YOLO-based architecture to detect personal protective equipment for construction sites. *PeerJ Computer Science, 8*, e999. https://doi.org/10.7717/peerj-cs.999

Jocher, G., Chaurasia, A., Qadir, A., & Stoken, A. (2023). *Ultralytics YOLOv8* (Version 8.0) [Computer software]. GitHub. https://github.com/ultralytics/ultralytics

Mvd, A. (2020). *Face Mask Detection Dataset* (Version 1) [Dataset]. Kaggle. Retrieved June 1, 2025, from https://www.kaggle.com/datasets/andrewmvd/face-mask-detection

Roboflow. (n.d.). *Glasses Detection Dataset* (Version 2) [Dataset]. Retrieved June 1, 2025, from https://universe.roboflow.com/noriel-vlmtq/glassesdetection-lwja6

Roboflow. (n.d.). *Hazard Vest Dataset* (Version 3) [Dataset]. Retrieved June 1, 2025, from https://universe.roboflow.com/tello-8ckdt/hazard-vest

Roboflow. (n.d.). *Kask Tanima (Helmet) Dataset* (Version 3) [Dataset]. Retrieved June 1, 2025, from https://universe.roboflow.com/sametataysamet/kask-tanima

Roboflow. (n.d.). *Moye Moye (Gloves) Dataset* (Version 1) [Dataset]. Retrieved June 1, 2025, from https://universe.roboflow.com/manoj-mava/moye-moye

Roboflow. (n.d.). *Sigara Deneme (Smoke Detection) Dataset* (Version 2) [Dataset]. Retrieved June 1, 2025, from https://universe.roboflow.com/deneme-yz/sigara-deneme

Wang, Z., Lei, L., & Shi, P. (2023). Smoking behaviour detection algorithm based on YOLOv8-MNC. *Frontiers in Computational Neuroscience, 17*, 1243779. https://doi.org/10.3389/fncom.2023.1243779