AI-Based Security Surveillance and Hazard Detection for Train Platform Safety

Alvaro Aparicio Serna, Xinrui Yu and Jafar Saniie
Embedded Computing and Signal Processing (ECASP) Research Laboratory (http://ecasp.ece.iit.edu/)
Department of Electrical and Computer Engineering
Illinois Institute of Technology, Chicago IL, U.S.A

***Abstract*—Safety in train platforms is a common concern in railway systems around the world. There are grave risks associated with the tracks and the crowds that often end up in injuries or deaths. Traditional approaches for ensuring safety in train transport networks rely on passive closed-circuit television (CCTV) monitorization that needs constant human attention. This paper proposes an AI-based alternative for detecting anomalies in surveillance videos, that is more efficient and cost-effective than traditional methods. Through Video Instance Segmentation (VIS), this work detects common risks such as overcrowding on the platform, people or objects standing on the edge of the platform, individuals or objects falling onto the tracks, the presence of firearms, and the presence of unattended baggage. The proposed algorithm combines state-of-the-art models like YOLOv8, ByteTrack, and Segment Anything Model (SAM) to classify, track, and segment object detections respectively. Additionally, this paper presents a custom-trained YOLOv8 model for gun detection. The results show that the system can successfully analyze video, create surveillance annotations, detect hazardous situations to alert authorities, and help prevent accidents and incidents on train platforms.**

Keywords—Railway Station, Passenger Safety, Surveillance Monitoring, Computer Vision, Artificial Intelligence

I. Introduction

User safety and security on train station platforms present significant challenges globally, with risks including slips, negligence, and assaults leading to accidents and fatalities [1] [2]. Despite the deployment of closed-circuit television (CCTV) for monitoring, their effectiveness is limited by the passive nature of surveillance and the overwhelming volume of video data for human operators to analyze, highlighting the inefficiency of current systems [3]. This necessitates the development of more efficient, automated surveillance solutions that leverage artificial intelligence (AI) techniques to enhance safety by detecting anomalies quickly and accurately. The paper aims to introduce an AI-based video processing algorithm focusing on video instance segmentation (VIS) to extract actionable insights for real-time response, aiming to mitigate common hazards and improve overall safety on train platforms.

II. Objectives

The paper aims to develop software for extracting key information from train platform surveillance footage to enhance safety by identifying risks such as overcrowding, individuals or objects near edges, falls onto tracks, firearm possession, and unattended baggage. It seeks to provide actionable insights for integration into an advanced AI-based surveillance system, improving station personnel's response to incidents. Additionally, the paper focuses on achieving VIS including object classification, segmentation, and tracking, thus facilitating comprehensive video analysis.

III. Video Instance Segmentation Approach

Video Instance Segmentation (VIS) extends the problem of instance segmentation from still images to videos, incorporating simultaneous classification, segmentation, and tracking of objects within video frames. This introduces the added complexity of associating data across frames. VIS offers the advantage of a more detailed scene understanding, allowing for precise object localization and improved management of object occlusion [4]. Building upon foundational work such as [5], which achieved breakthroughs in instance segmentation for still images, various methodologies have been developed to bridge the gap for video [6][7]. However, the approach of this paper distinguishes itself from others. This paper's strategy merges the capabilities of leading pre-trained models: You Only Look Once version 8 (YOLOv8) [8] for object detection, Segment Anything Model (SAM) [9] for instance segmentation, and ByteTrack [10] for instance tracking. As depicted in Fig 1, the process involves analyzing the video on a frame-by-frame basis, utilizing YOLOv8 to identify objects and their bounding boxes. These boxes then serve as inputs for ByteTrack to assign tracking identifiers, and for SAM to create pixel-level masks, thereby addressing the three core tasks of VIS.

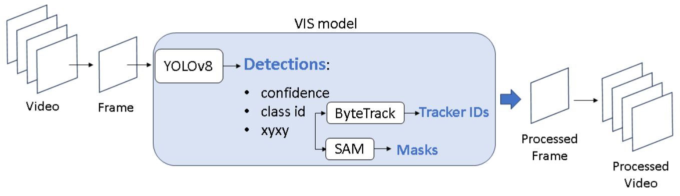


Fig 1. Video Instance Segmentation (VIS) Model Approach

IV. Model Description

This chapter focuses on describing the models that make up the video instance segmentation algorithm proposed in this paper.

A. *YOLOv8*

YOLOv8, created by Ultralytics, is the eighth iteration of the popular YOLO single-stage object detector architecture. It introduces numerous improvements compared to the earlier versions and despite not having a published paper yet, it is considered the new state-of-the-art object detector [8]. YOLOv8 will be in charge of generating bounding boxes around objects of interest to pass them to further stages of the model and complete the video instance segmentation task. The choice of this particular model resides under the assumption that, given its results, it would provide the best performance in the task. This version is based on its predecessors, similar to YOLOv5 [8], it keeps the backbone, neck, and head structure. However, it introduces modifications that significantly improve efficiency and performance [8]. Changes in the backbone include the substitution of the Cross Stage Partial (CSP) layers by a new type of module named C2f. This module is a cross-stage partial bottleneck with two convolutions [8] that combine high-level features with contextual information. Another significant change takes place in the head, which is now decoupled so each branch (bounding box, class probabilities, and confidence scores) can focus on its specific task. Additionally, the model no longer uses fixed-size anchor boxes. This anchor-free detection speeds up the following non-maxima suppression (NMS) stage [11].

B. *ByteTrack*

ByteTrack is a state-of-the-art multi-object tracker that proposes a new association method that achieves better performance than older techniques for estimating bounding boxes and identities of objects in videos [10]. It receives the bounding box detections from YOLOv8 and uses them to generate consistent tracker IDs for objects throughout all video frames. Most Multi-Object Tracking (MOT) methods only use detection boxes with high-confidence scores as the input for data association. Those methods filter out low-confidence detection boxes because they consider they are not accurate enough and contain too much background, which can harm the overall performance. Nevertheless, the ByteTrack algorithm chooses to use almost every detection box, including low-confidence score ones, as they often represent occluded objects whose trajectories would otherwise get lost [10]. The process followed by the "Byte" association method is the following. After a previous detection stage, the bounding box detections are arranged using fixed threshold values. High-confidence score detections and low-confidence score detections are kept for future steps while backgrounds are discarded at that very moment. The algorithm then predicts new locations of tracks using a Kalman filter. Next comes the association process, which takes place in two steps. In the first step, the high-confidence detections are associated with the current tracks using IoU and cosine similarity between the detections and the track predictions. After this step, the high-confidence detections and the tracks that remain unassociated are left for the second association step. In the second association step, the low-confidence detections are associated with the remaining tracks, this time attending only to IoU criteria. Finally, the unmatched tracks are deleted and new tracks are initialized using the unmatched remaining detections.

C. *Segment Anything Model (SAM)*

The Segment Anything Model (SAM), created by Meta AI, is a promptable image segmentation model that can generate pixel-level masks for virtually any object contained in an image [9]. SAM is one of the three components of the Segment Anything (SA) project, a new initiative that aims to create a foundation for image segmentation that entails a segmentation model, a large dataset for image segmentation, and a data engine. In the context of this work, SAM is responsible for generating the segmentation masks given the detections from YOLOv8 to achieve the complete video instance segmentation goals. The rationale behind the choice of this model relies on the fact that traditional approaches require a lot of effort for training data collection and manual annotation. Other segmentation models must be trained extensively on specific data to only do one task and they require re-training when changing the dataset. SAM can be divided into three main components: the image encoder, the prompt encoder, and the mask decoder.

The image encoder is the first stage of the model. It is a version of the MAE pre-trained Vision Transformer [12], which has been adapted for high-resolution inputs. This choice is motivated by scalability and accessibility to pre-training. The image encoder gets fed the input image and generates an image embedding that is used as the basic element necessary for later segmentation. The next part is the prompt encoder which allows users to specify the locations for the desired segmentation using sparse (points, boxes, text) and dense (rough masks) prompts. Point and box prompts are represented by positional encodings and summed with the image embeddings, text prompts are encoded with CLIP [13], and mask prompts are embedded using convolutions and summed with the image embeddings.

Finally, the last stage of SAM is the mask decoder, which outputs the final segmented image. It maps the image embeddings from the image encoder with the prompt embeddings from the prompt encoder. The architecture of the mask decoder is based on a Transformer decoder block followed by a prediction head that uses self-attention mechanisms to update the embeddings. The final computation of the mask is done by a dynamic linear classifier.

V. Surveillance and Hazard Detection System

A. *Custom-trained YOLOv8 for Firearm Detection*

This project aimed to enhance surveillance through firearm detection using a custom-trained YOLOv8 model. Due to the absence of firearm classification in the COCO dataset's pre-trained YOLOv8 model, we compiled a custom dataset of 2228 images, combining original and augmented images to enhance robustness and minimize false positives.

Focused on gun detection, the dataset underwent preprocessing and augmentation (horizontal flips, ±15° shears, and resizing), resulting in diverse training, validation, and testing splits (Table I). The model, trained over 100 epochs with Stochastic Gradient Descent, applied regularization techniques (Blur, Median Blur, Grayscale, CLAHE) to reduce noise sensitivity and improve accuracy. The training process showed a consistent reduction in loss and stabilization in precision, recall, and mean average precision metrics (Fig 2).

The confusion matrix revealed an 86% accuracy in gun predictions. The noted 100% false positive rate stems not from a deficiency in the model but from the automatic inclusion of a background class in the matrix generation, despite the model's design does not predict a "background" label in the absence of other class detections (Fig 3). The evaluation demonstrated high precision for low to moderate recall values and a consistent F1 score for a wide range of confidence thresholds (Fig 4), indicating the model's effectiveness in firearm detection. The final model exhibited a mean average precision of 0.928 for IoU thresholds above 50% (Table II).

TABLE I. DATA SPLITS IN FIREARM DATASET

	Training	Validation	Testing
Before Augmentation	70 %	20 %	10 %
After Augmentation	87 %	8 %	5 %

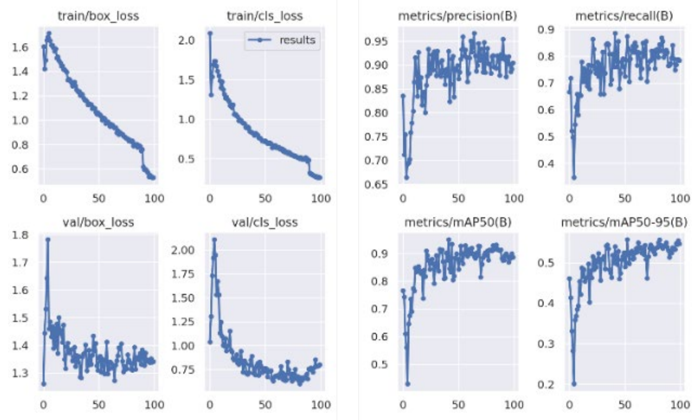


Fig 2. Custom YOLOv8 Training Graphs

TABLE II. CUSTOM YOLOV8 MODEL RESULTS

Metric	Value
Layers	268
Parameters	68,124,531
Gradients	0
GFLOPs	257.4
Class	All
Images	170
Instances	167
Precision	0.909
Recall	0.874
mAP50	0.928
mAP50-95	0.556

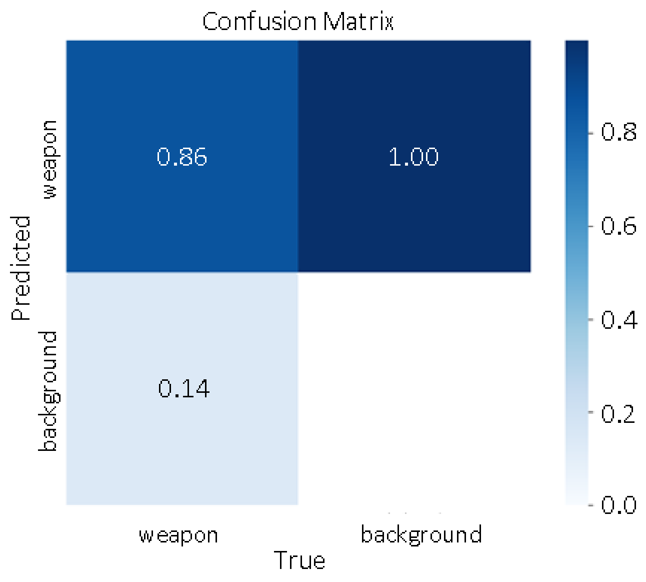


Fig 3. Custom YOLOv8 Confusion Matrix

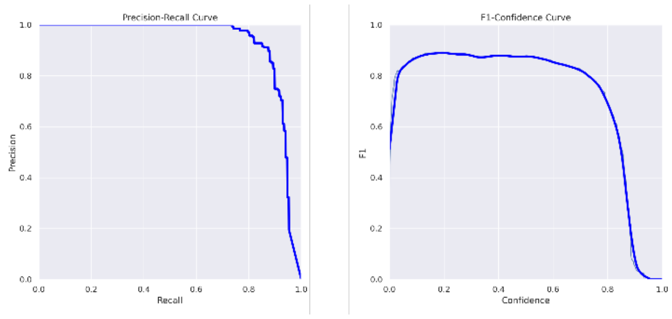


Fig 4. Custom YOLOv8 Precision-Recall (left) and F1 Score (right) Curves

B. *Utility Functions*

This section describes the utility functions developed to manage processes in the VIS algorithm.

The integration of ByteTrack with YOLOv8 for object tracking requires manual matching of detections to track objects due to model incompatibilities. This process involves three main steps: converting YOLOv8 detections to a format compatible with ByteTrack, preparing ByteTrack's output for subsequent matching, and then pairing detections with tracks based on the highest Intersection over Union (IoU) to ensure precise tracking. These steps use NumPy arrays for efficient data handling, with methodologies grounded in resources [14], [15] and [16].

Counting objects within predefined zones is vital for hazard detection. This involves assessing whether the midpoint of a detection falls within a zone, using Shapely [17] for spatial analysis, and updating object counts within these zones. A binary mask is employed to track whether detections are inside the zone, facilitating accurate monitoring and response to potential hazards.

The segmentation utility applies segmentation masks to video frames, enhancing visual outputs. It adjusts mask colors and transparency to blend segmentation overlays with video frames seamlessly. This function's process upon techniques discussed in [18].

Hazard detection utilities leverage the outputs from the VIS algorithm to identify potential safety threats within the monitored environment. These utilities encompass a series of processes designed to monitor critical aspects such as overcrowding in specific zones (like platforms), the presence of individuals or objects in critical or restricted areas (such as platform edges or tracks), the detection of firearms, and the identification of unattended baggage. For overcrowding, the system compares the current occupancy against a predefined maximum capacity, generating alerts if exceeded. In critical zones, it checks for the presence of persons or items that shouldn't be there, prioritizing alerts for human presence. The firearm detection process links detected weapons with nearby individuals, calculating spatial overlaps to identify potential suspects. Lastly, for unattended baggage, it assesses the proximity of people to bags, flagging any items without nearby owners.

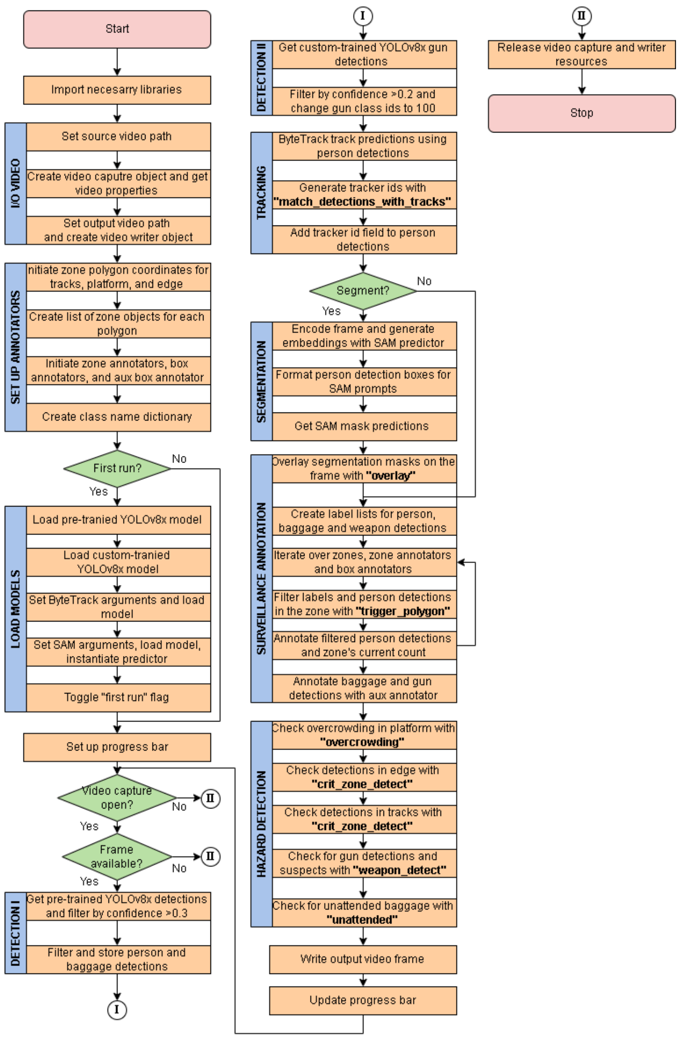
C. *VIS Surveillance and Hazard Detection Algorithm*

The VIS algorithm for surveillance and hazard detection utilizes a combination of models and utility functions to monitor train platforms, identifying hazardous situations as illustrated in Fig 5. Initially, the algorithm sets up video input and output through OpenCV, capturing essential details like frame dimensions and rate, and prepares for video creation with an MP4 codec. Annotation tools for zones and detections are then prepared. Zones—train tracks, the platform, and its edge—are defined with polygons to count and annotate people's presence. Bounding box annotators for people, weapons, and baggage are established, along with a dictionary mapping class names to detections for clear labeling. Model loading follows, with pre-trained and custom-trained YOLOv8x models for detecting people, baggage, and guns. ByteTrack is utilized for tracking people, optimized for precision, while the SAM model is employed for segmentation, enhancing visual output on a GPU configuration. In the video analysis phase, the algorithm processes frames to detect, track, and segment objects. YOLOv8x models identify and sort people, baggage, and weapons, filtering by confidence. ByteTrack updates tracking IDs for people detections, and SAM generates segmentation masks, adding depth to the analysis. Finally, segmentation masks are overlaid, and detections are annotated with labels including class names, confidence levels, and tracking IDs. The algorithm assesses hazards like overcrowding, unauthorized entries, firearm presence, and unattended baggage, applying alert messages on frames as necessary. Each frame, once analyzed and annotated, contributes to the output video.

VI. Results and Discussion

A. *Results*

This section shows the results obtained by the VIS surveillance and hazard detection system demonstrating how it analyzes and detects risks when posed with common safety and security hazardous situations that can occur on a train platform. The system consists of a combination of state-of-the-art pre-trained models and a custom-trained model whose performances are well-known or have been discussed in previous sections.



Start
Import necesarry libraries
I/O VIDEO
Set source video path
Create video caputre object and get video properties
Set output video path and create video writer object
SET UP ANNOTATORS
nitiate zone polygon coordinates for tracks, platform, and edge
Create list of zone objects for each polygon
Initiate zone annotators, box annotators, and aux box annotator
Create class name dictionary
First run?
No
Yes
LOAD MODELS
Load pre-tranied YOLOv8x model
Load custom-tranied YOLOv8x model
Set ByteTrack arguments and load model
Set SAM arguments, load model, instantiate predictor
Toggle "first run" flag
Set up progress bar
Video capture open?
No
II
Yes
Frame available?
No
II
Yes
DETECTION I
Get pre-trained YOLOv8x detections and filter by confidence >0.3
Filter and store person and baggage detections
I
I
DETECTION II
Get custom-trained YOLOv8x gun detections
Filter by confidence >0.2 and change gun class ids to 100
TRACKING
ByteTrack track predictions using person detections
Generate tracker ids with "match_detections_with_tracks"
Add tracker id field to person detections
Segment?
No
Yes
SEGMENTATION
Encode frame and generate embeddings with SAM predictor
Format person detection boxes for SAM prompts
Get SAM mask predictions
SURVEILLANCE ANNOTATION
Overlay segmentation masks on the frame with "overlay"
Create label lists for person, baggage and weapon detections
Iterate over zones, zone annotators and box annotators
Filter labels and person detections in the zone with "trigger_polygon"
Annotate filtered person detections and zone's current count
Annotate baggage and gun detections with aux annotator
HAZARD DETECTION
Check overcrowding in platform with "overcrowding"
Check detections in edge with "crit_zone_detect"
Check detections in tracks with "crit_zone_detect"
Check for gun detections and suspects with "weapon_detect"
Check for unattended baggage with "unattended"
Write output video frame
Update progress bar
II
Release video capture and writer resources
Stop

Fig 5. VIS Surveillance and Hazard Detection Algorithm Flowchart

Therefore, the delivery of results focuses on demonstrating real-case scenarios through demos. The first demo handles a video of a train platform during normal operation. The video captures the moment a train arrives at the station and the passengers get in and out of the train. As seen in the frame sequence shown in Fig 6, the system achieves the object detection, segmentation, and tracking objectives described for the VIS problem, focusing on people detections. Each detected individual receives a very precise segmentation mask and a tracker ID that sticks with them throughout the whole video. There are occasional frames in which one detection loses the tracker ID, but it is recovered shortly after. Moving on to the video analysis capabilities, the system correctly filters and counts the person detections for the platform, platform edge, and train tracks. Different alert messages pop up when the system identifies risks. For instance, the overcrowding alert is triggered when more than 5 people stand on the platform. The system also alerts when someone steps on the edge. On the other hand, people detections in the tracks zone are not considered because the train is at the station. Another demo, Fig 7, recreates the situation of a person pacing on the edge of the platform for an extended period. In this case, the system alerts of that critical detection, specifying the tracker ID of the subject. Receiving frequent alerts of this nature all referred to the same tracker ID is valuable information that can be leveraged by the authorities to intervene. To put the system's capabilities to the test, the demo in Fig 8 simulates the situation where an individual leaves an unattended backpack behind. The sequence shows that while the backpack is being held by the individual, there is no alert message. It is when the individual steps away from the backpack that the alarm is triggered.

Finally, to try out the custom-trained YOLOv8x model for gun detection, the demo in Fig 9 recreates the event of a suspect drawing a gun. The results show that the system is capable of detecting the presence of a gun and sending an alert message with the tracker ID of the suspects involved (in this case only one). Unfortunately, this demo could not be carried out at a real train station for safety reasons and lack of authorization. Similarly, the event of objects or people falling onto the tracks could not be recreated either. However, the detection of those situations uses the same functions (although with different parameters) as in Fig 7, which was tested correctly.

B. *Discussion*

The demonstration results present a functional system that accurately detects, tracks, and segments person detections while analyzing potential risks and alerting when a hazard is identified. However, during the development of the system, there have been challenges that are worth commenting on in this section.

The first comment is regarding the decision to display alert messages on the processed frame when an event is triggered. To deliver the results, this option was more visual than, for example, printing a message on the terminal. Besides, this paper focuses on extracting insights that can later be fed downstream to a more complex system. Hence, showing the messages on the frame is sufficient to test that the system works as intended. Needless to say, in the scope of a more complex application, the surveillance system could interact with a database log and send messages to the authorities, who can receive them on their devices. Having cleared that out, let us focus on the challenges that the system may encounter, and that can undermine its performance. One crucial factor is the camera's location and orientation. It needs to be positioned at a height with the correct tilt angle to provide a top-down view that allows seeing the extent of the platform and tracks. It is key to consider the alignment of the camera's axis with the direction of the tracks. Rather than having a complete alignment, the projection of the camera's axis onto the ground plane (where people stand) shall be slightly tilted. This avoids occlusion between the people standing on the platform and helps with weapon and unattended baggage detection. Of course, occlusion can also happen in the direction perpendicular to the tracks, but since the width of the platform is significantly shorter than its length, occlusion in this direction is less aggravating. Additionally, experience has shown that the framing of the area under surveillance shall show the whole body of the detections. The detection filtering by zone is done by assessing the top or bottom middle point of the detection's bounding box. Given the top-down angle of the camera, it would be best to use the top middle point as in Fig 6. However, with tilted camera angles that show the extension of the platform horizontally like in Fig 7 or Fig 8, it is better to use the bottom middle point to asses where the people are stepping. Therefore, it is recommended to use the top middle point only when the camera is aligned with the tracks or at very zenith angles. Considering the above, the best camera location would be at an almost zenith angle and slightly tilted to show the horizontal extension of the tracks and platform. Moreover, if the resources allow it, it would be ideal to have two cameras, one with a zenith-side view of the platform and one with a front view of the platform. The first would focus on supervising the zones and the second on detecting unattended packages and weapons. Another worth noting aspect is the performance of the gun detection model. While the validation results denote a fairly good model, the testing demos show excessive false negatives that prevent a good tracking of the firearm. After analysis, it was concluded that most false positives happen when the gun is held at odd angles and not pointed. This suggests a bias in the training and validation datasets, that do not include enough sample images with atypical gun positions and orientations.

Finally, one big issue of the system is its feasibility of use in real-time. The limiting factor is the approach toward segmentation. SAM creates very detailed segmentation masks of virtually any object without the need for a custom-trained segmentation model. That and its ease of use with sparse prompts made it an attractive choice when thinking about the VIS task. However, according to this work's experience, the issue with SAM is the time spent encoding the image frames and generating the embeddings. That process is the most time-consuming of the whole algorithm and it needs to be repeated in every frame for the model to work correctly. That translates into having to wait around 15 minutes to process a 15 seconds long video. Of course, this depends on the hardware, but it is too much time and inefficiency regardless. SAM is still a very recent model and its feasibility to run in real-time is being discussed currently in the issues tab of the Segment Anything project GitHub [19]. Having said that, if we exclude the segmentation part, the video processing times match the duration of the video, hence real-time processing would be feasible. Further supporting this, our related research using Nvidia Jetson Orin Nano achieved a framerate of 22 FPS. This configuration, which utilized 75% of GPU capacity and 25% of CPU, had an inference time of 28 ms and a total power consumption of 8300 mW [20]. These results highlight the potential for real-time operation with appropriate hardware and more efficient segmentation models.

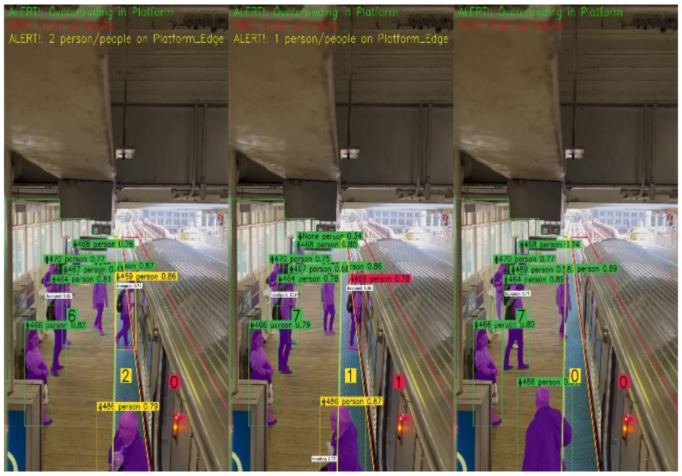

ALERT!: Overcrowding in Platform
ALERT!: 2 person/people on Platform_Edge
ALERT!: Overcrowding in Platform
ALERT!: 1 person/people on Platform_Edge
ALERT!: Overcrowding in Platform

Fig 6. System Demo: Normal Platform Operation

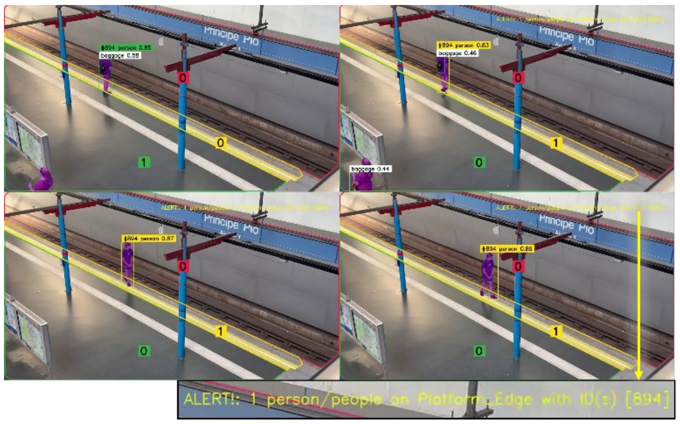


Fig 7. System Demo: Person Pacing on Platform Edge

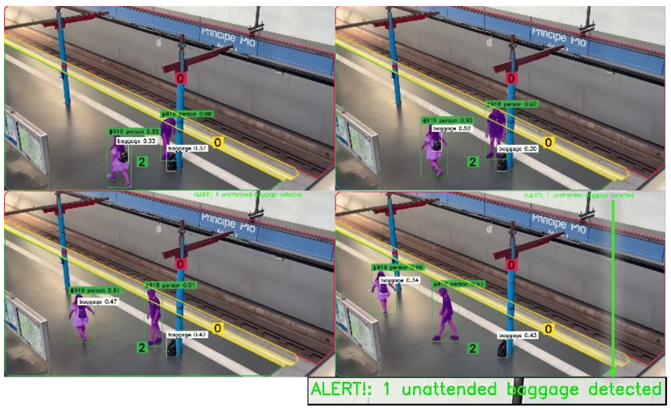


Fig 8. System Demo: Unattended baggage

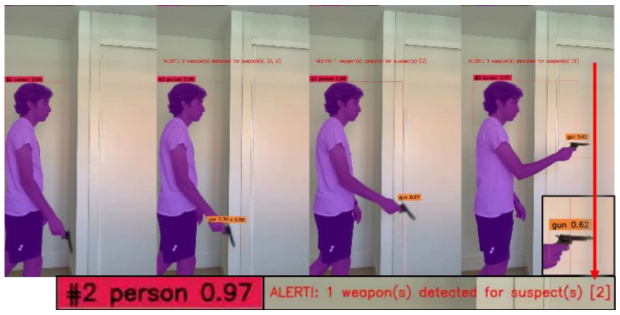


Fig 9. System Demo: Gun Detection

VII. Conclusions

This work presents a Video Instance Segmentation (VIS) algorithm for surveillance and hazard detection in train platforms. The algorithm combines three state-of-the-art models (YOLOv8x, ByteTrack, and SAM) to achieve classification, segmentation, and tracking of detected objects. The system focuses on detecting common risks that take place in train stations such as overcrowding on the platform, standing people or objects on the edge of the platform, fallen people or objects on the tracks, the presence of firearms, and the presence of unattended baggage. The results show that the system can successfully analyze video, create surveillance annotations and detect those hazardous situations to alert authorities and help in preventing accidents and incidents in train platforms. The algorithm in this work aims to collect insights that can then be fed downstream into a more complicated system. The station employees, including monitoring operators, security officials, and train drivers, may react to incidents and take the appropriate actions to respond to safety and security hazards using the analyzed video data. As a result, this paper can be viewed as part of a larger attempt to realize a more complicated AI-based surveillance system for railway station safety.

References

[1] S. Oh, S. Park, and C. Lee, "A platform surveillance monitoring system using image processing for passenger safety in railway station," Jan. 2007.

[2] C. Poirier, S. Adelé, and J.-M. Burkhardt, "Individual accidents at the interface between platform, train and tracks (PT2I) in the subway: a literature review," Cognition, Technology & Work, Feb. 2020.

[3] J. Black, S. Velastin, and B. Boghossian, "A real-time surveillance system for metropolitan railways," Jan. 2006.

[4] L. Yang, Y.-C. Fan, and N. Xu, "Video Instance Segmentation," Oct. 2019.

[5] K. He, G. Gkioxari, P. Dollar, and R. Girshick, "Mask R-CNN," 2017 IEEE International Conference on Computer Vision (ICCV), Oct. 2017.

[6] H. Lin, R. Wu, S. Liu, J. Lu, and J. Jia, "Video Instance Segmentation with a Propose-Reduce Paradigm," 2021 IEEE/CVF International Conference on Computer Vision (ICCV), Oct. 2021.

[7] Y. Wang et al., "End-to-End Video Instance Segmentation with Transformers," arXiv (Cornell University), Jun. 2021.

[8] J. Terven and D. Cordova-Esparza, "A Comprehensive Review of YOLO: From YOLOv1 to YOLOv8 and Beyond," arXiv (Cornell University), Apr. 2023.

[9] A. Kirillov et al., "Segment Anything," arXiv (Cornell University), Apr. 2023.

[10] Y. Zhang et al., "ByteTrack: Multi-object Tracking by Associating Every Detection Box," pp. 1–21, Oct. 2021.

[11] D. Reis, J. Kupec, J. Hong, and A. Daoudi, "Real-Time Flying Object Detection with YOLOv8," May 2023.

[12] K. He, X. Chen, S. Xie, Y. Li, Piotr Dollár, and R. Girshick, "Masked Autoencoders Are Scalable Vision Learners," Nov. 2021.

[13] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, and others, "Learning transferable visual models from natural language supervision," International Conference on Machine Learning, 2021.

[14] Roboflow, "Supervision," GitHub, Oct. 27, 2023. https://github.com/roboflow/supervision

[15] Y. Zhang, "ByteTrack," GitHub, Nov. 09, 2023. https://github.com/ifzhang/ByteTrack

[16] "roboflow/notebooks," GitHub, Apr. 27, 2023. https://github.com/roboflow/notebooks

[17] "The Shapely User Manual — Shapely 1.6 documentation," Readthedocs.io, 2018. https://shapely.readthedocs.io/en/stable/manual.html

[18] "Ultralytics," GitHub. https://github.com/ultralytics

[19] "Segment Anything," GitHub, Apr. 17, 2023. https://github.com/facebookresearch/segment-anything

[20] D. Rodriguez Garcia, A. Aparicio Serna, D. Sancho Crespo, X. Yu, and J. Saniie, "AI Smart Security Camera for Person Detection, Face Recognition, Tracking and Logging," Apr. 2024.