YOLO and LLM-Based Visual Solution for Automated Cargo Safety Detection and Alerts at Hong Kong International Airport

Ju Lin, Rouheng Ni, Ruilin Shi, Shuxuan Li, Jiayi Zuo
Department of Aeronautical and Aviation Engineering,
The Hong Kong Polytechnic University, Hong Kong
GitHub Repository

***Abstract*—This report presents an automated system for cargo safety detection, particularly designed for the Hong Kong International Airport (HKIA), with the purpose of reducing serious risks in cargo handling processes. The main idea of the system is to detect dangerous situations where goods are above the operator; hence, this is often the main reason for accidents in workplaces. Using the power of YOLO object detection combined with the power of multimodal Large Language Models (LLMs), the system will analyze visual information to classify potentially unsafe operations with high precision. Depth proximity estimation has been utilized during development to validate the LLM-based classification concerning its reliability and robustness. On detection, the present system generates real-time alerts through a Telegram notification platform by providing actionable insights like annotated images and video clippings that will facilitate quick corrective action. The approach thus assures improvement in operational safety with real-time monitoring and fast communication of hazards that improve the response time significantly. Further work will be directed at refining the system for wider application, extending its capability to more complex cargo operation scenarios.**

I. Background

With the development of the aviation industry and globalization, more and more goods will be transported in the cargo areThe aim of this project is to devise an automated system capable of detecting potential security threats in real-time, thereby augmenting operational safety and efficiency at airports. This system leverages advanced deep learning technologies, incorporating YOLO for object detection [1], [2], [3] and LLMs [4], [5] for hazard identification. Upon detecting potential hazards, the system automatically triggers alerts to relevant personnel through an integrated Telegram notification system, thereby bolstering the security protocols and operational efficacy at the HKIA.

II. Workflow

The workflow for this project was designed in such a manner that the whole process involved a number of key stages that helped in handling data and implementation in a very orderly manner.

First, raw data acquisition of almost 300GB in size involved a RealSense RGB-D camera at the HKIA, capturing different operational scenarios, thus forming the basis of further processing. Following the acquisition, RGB-D footage extraction from recorded rosbag files was done in order to allow access to visual and depth streams for further analysis.

The data was then labeled with the Roboflow platform to better structure a dataset to train machine learning models. After labeling them, a model could be trained to perform the desired tasks highly efficiently. Then, a proximity estimation method was developed in order to compare the heights of goods on trolleys with that of the operators themselves. The actual process entailed several attempts at various methods, including a straightforward numerical comparison of the heights, as well as examining other methods that might yield more accurate and efficient outcomes.

After that, an alert mechanism was built that would trigger in case of discrepancies or issues. That was made through a Python script with an integrated Telegram API, enabling the sending of real-time alerts through the use of Telegram. Each one of these workflow components played into accomplishing the project objectives in an effective and seamless way.

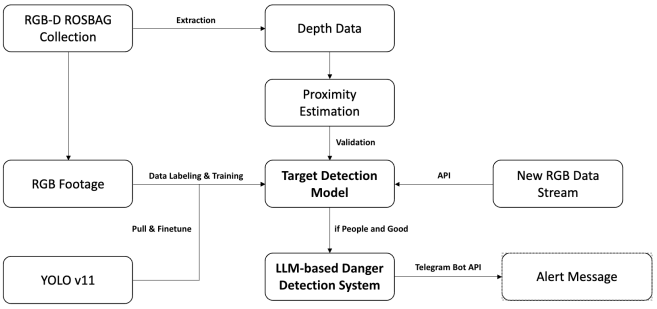

Fig. 1: Workflow

III. Hardware Setup

The automated cargo safety detection system at HKIA consists of the hardware setup of the RealSense 435i RGB-D Camera. The camera belongs to the Intel RealSense series, designed for different applications in robotics, 3D scanning, and augmented reality. Below is a detailed explanation of the RealSense 435i RGB-D Camera and its configuration:

- **Depth Sensing**: The RealSense 435i camera uses Time-of-Flight (ToF) technology to measure depth. It can capture depth data at speeds up to 30 frames per second (fps), which is adequately fast for real-time applications of depth perception.
- **RGB Imaging**: Supplementing the depth sensing, the 435i has a high-resolution RGB camera, which could be used to grab crisp color images at 1920x1080 pixels. These are suitable for visual detection and image processing.

With the RealSense 435i RGB-D Camera, one could control it programmatically to capture depth and RGB data. This enables the next processes of data preprocessing, YOLO Object Detection, and other processes of the project, which would be run on a highly-integrated Raspberry Pi 5.

IV. Data Collection

Data was collected from various zones within the cargo handling areas at the HKIA. Data is captured utilizing an intel realsense camera. Live depth, RGB streams, and 2D Views are turned on in Intel realsense app. The data of airport staff handling cargo of varying heights was captured. Various camera positions have been experimented with. All data are recorded into rosbag. The dataset includes images and videos of different cargo operations, annotated for object types and proximity information. The collected data ensures a comprehensive training set that reflects real-world scenarios.

V. Data Extraction

Python and the Pyrealsense2 library, which provides a binding for the Intel RealSense SDK (librealsense)[6], are used to analyze the recorded data from the Intel RealSense camera. The first objective was to preserve excellent data quality while extracting RGB and Depth streams from the recorded rosbag file. The YOLO model is then trained using the retrieved data.

A. RGB Data Extraction

Prior to starting the extraction, the Python bindings (pyrealsense2) and the Intel RealSense SDK (librealsense) were set up. This configuration made it possible to integrate with the rosbag file that contained the recorded data with ease. RGB data was accessed from the color frame stream after frames were extracted from the pipeline. After that, the frames were transformed into numpy arrays for processing and display. During extraction, frames that were noisy or incomplete were removed. This made guaranteed that only full and high-quality RGB frames were chosen for processing.

B. Depth Data Extraction

The process of extracting depth data from the RealSense stereo camera can be described step by step as follows:

- **Enable Depth and RGB Streams:** The RealSense pipeline was configured to enable both the depth and RGB streams to ensure synchronized recording of data during the session. Depth frames were accessed from the pipeline as raw pixel-wise depth values, typically measured in millimeters. These values were processed and converted into `numpy` arrays for compatibility with Python-based workflows.
- **Depth Calculation:** Each pixel in the depth frame represents the distance from the camera to the corresponding point in the scene. The depth value, $D(x, y)$, for a pixel (x, y) is calculated as:

$$D(x, y) = \frac{f \cdot B}{d(x, y)}$$

where:
  - f is the focal length of the camera,
  - B is the baseline distance between the stereo cameras, and
  - $d(x, y)$ is the disparity value for that pixel, derived from stereo matching.

Larger disparity values correspond to closer objects, while smaller values indicate farther objects.

- **Filtering Depth Data:** To improve the quality of the depth frames, noisy or inconsistent depth readings were filtered using techniques provided by the RealSense SDK, such as temporal and spatial smoothing or hole-filling. The filtering process can be represented as:

$$D_{\text{filtered}}(x, y) = \frac{\sum_{(i,j)\in\mathcal{N}} w_{i,j} \cdot D(i, j)}{\sum_{(i,j)\in\mathcal{N}} w_{i,j}}$$

where:
  - $\mathcal{N}$ is the neighborhood of pixel (x, y),
  - $w_{i,j}$ are the weights assigned to neighboring pixels based on spatial and temporal constraints.

This ensures that the filtered depth frame retains vital information while reducing noise and inconsistencies.

- **Aligning Depth and RGB Data:** To achieve spatial consistency, the depth frames were aligned with their corresponding RGB frames. This alignment ensures that both depth and RGB data share the same coordinate system, which is crucial for tasks such as object detection or proximity estimation. The transformation from depth coordinates (u_d, v_d) to RGB coordinates (u_r, v_r) is given by:

$$\begin{bmatrix} u_r \\ v_r \\ 1 \end{bmatrix} = K_{\text{RGB}} \cdot \left(R \cdot K_{\text{Depth}}^{-1} \cdot \begin{bmatrix} u_d \\ v_d \\ 1 \end{bmatrix} + T \right)$$

where:
  - K_{RGB} and K_{Depth} are the intrinsic matrices of the RGB and depth cameras, respectively,
  - R is the rotation matrix, and
  - T is the translation vector between the depth and RGB cameras.

- **Visualization:** A visualization of the extracted depth data is shown in the left part of Figure 2. This visualization provides an intuitive representation of the spatial relationships captured by the depth sensor, enabling inspection and verification of the data's quality.

Fig. 2: Extracted Data

VI. Data Labeling & Training

Data labeling and training are realized using Roboflow [7].

A. Data labeling using Roboflow

1) Upload the dataset: After the process of data extraction, we get some video data with higher quality. After that, we capture hundreds of images of both people and goods in the video, which is the dataset we are going to use, and then upload these image data to Roboflow.

2) Annotate the images: In Roboflow we manually draw bounding boxes around people and goods respectively. Then, we assign labels “goods” and “people” to each bounding box.

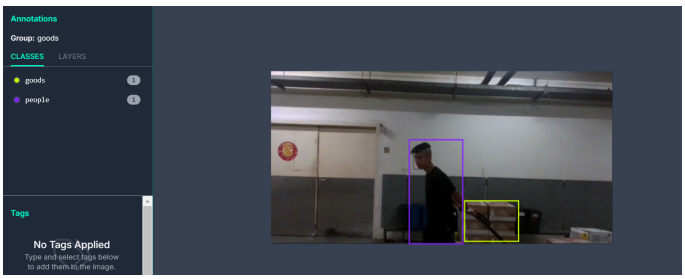

Fig. 3: Image Annotation

3) Split dataset: After the process of data labeling, we get a series of labeled datasets. Then we split the labeled dataset into training, validation, and test sets. Our splitting scheme as follows:

- **80% for training:** Used to train the YOLOv11 model.
- **10% for validation:** Used for fine-tuning hyperparameters and checking model performance during training.
- **10% for testing:** Used for evaluation of the model.

4) Export the Dataset: Export the dataset in YOLO format and download the dataset as a ZIP file, which was used for YOLO training.

B. Training the YOLO Model

YOLO is a popular object detection algorithm and widely used in computer vision tasks. That uses single neural network, grid system, real-time Inference to work.

YOLO has many advantages as follows:

- **High speed:** YOLO can achieve real-time detection.
- **End-to-end Learning:** YOLO performs object detection in a single pass, which makes YOLO easier to train and deploy.
- **High detection accuracy.**
- **Generalization capability:** It can effectively detect objects in different environments and conditions, making it suitable for real-world applications.
- **Unified architecture:** YOLO has a simple, unified architecture that can be easily modified or extended.
- **Versatility:** YOLO can be used in various situations.

Process of Model Training

- **Set up:** This step concludes the installation and import of Roboflow SDK, which is used to interact with the Roboflow platform. After that, we use the API key to locate the workplace.
- **Install YOLOv11:** Install YOLOv11 via Ultralytics.
- **Fine-tune YOLOv11 on dataset:** When training YOLOv11, locate our data in datasets.
- **Train the model:** Using the training set to train the model.
- **Display training results:** The plots show the performance of the training, using metrics including loss value, precision, recall rate, and mAP of each epoch.

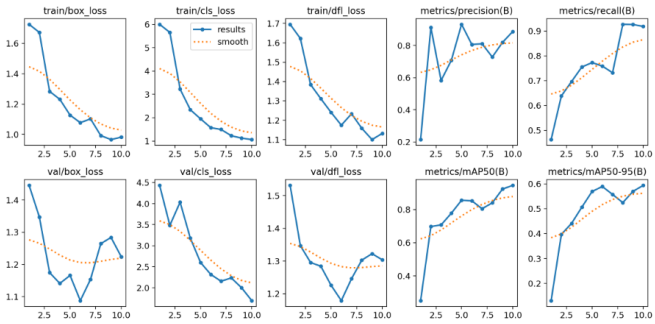

Fig. 4: Loss value, precision, recall rate, and mAP of each epoch

- **Validate the model:** Using the validation set to validate the trained model.
- **Display validation results:** Show the performance of validation.

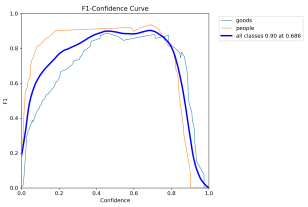

Fig. 5: F1-Confidence curve

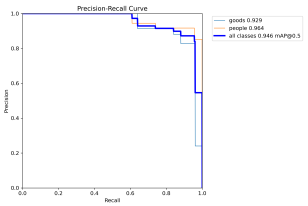

Fig. 6: Precision-Confidence curve

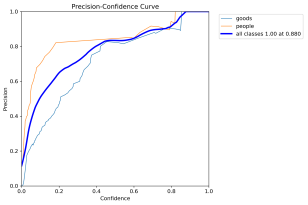

Fig. 7: Precision-Confidence curve

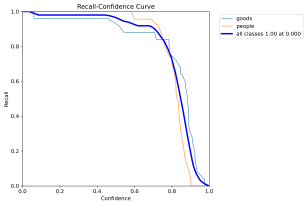

Fig. 8: Recall-Confidence curve

- **Infer with the model:** Using the model to inference with the test set.
- **Display the result of testing:** Show the result of inference with the model.

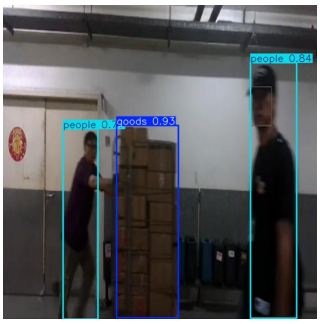

Fig. 9: Inference result

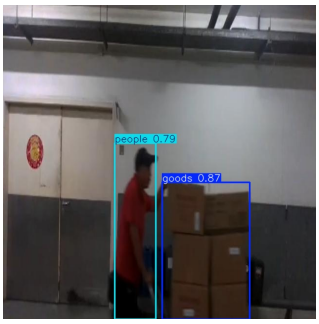

Fig. 10: Inference result

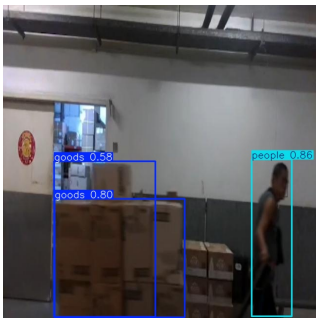

Fig. 11: Inference result

Fig. 12: Test result

- **Upload model on Roboflow:** After we finish training the YOLOv11 model, we can upload the model weights to Roboflow Deploy.

VII. Model Deployment and Application

Roboflow Hosted Video Inference was utilized to deploy the trained YOLO model above, to achieve that it can automatically detect and save the image that people and goods appeared in the inputting video simultaneously.

Roboflow Hosted Video Inference is a powerful tool that allows users to detect and analyze objects on the inputting video by using machine learning model trained on Roboflow, making video object detection much easier and more effective. Users can pay more attention to model training and application development by leveraging cloud computational resources without worrying about the underlying technical details and complicated local environment configuration. This technology makes it easier for developer to perform inference on video files. Moreover, this technology provides powerful support for a variety of applications, from security monitoring filed to object detection filed, and has a wide range of application prospects.

Specific implementations are as follows.

- **Development environment configuration.** Obtain API key, project ID and model version from Roboflow and install roboflow library on the python environment, then the model is accessible to python development environment through API key to connect to Roboflow.
- **Prediction data analysis.** Input video files for inference, utilize OpenCV library [8] to read video frames and process them frame by frame, call the trained model for inference to obtain prediction results, traverse and analyze the returned JSON data. The valuable parameters of prediction results are shown below.

```
'predictions': [
    {
    'x': 978.0,
    # x coordinates in the upper left
    corner of the bounding box
    'y': 703.0,
    # y coordinates in the upper left
    corner of the bounding box
    'width': 70.0,
    # The width of the bounding box
    'height': 466.0,
    # The height of the bounding box
    'confidence': 0.82,
    # The accuracy of model prediction,
    value range from 0 to 1
    'class': 'goods',
    # The detected object
    'class_id': 0,
    # The detected object ID
    }
]
```

- **Data extraction.** Extract class, confidence and bounding box information in each prediction result and save the frame when both 'people' and 'goods' are detected simultaneously with confidence greater than 0.7 to ensure only high-confidence detection results are saved.

By leveraging Roboflow Hosted Video Inference, this implementation effectively automates the detection of people and goods in video files, which contributes to a range of applications for object detection and analysis. The approach exhibits high efficiency of cloud-based solutions for machine learning tasks.

VIII. Height Comparison Using Depth Data

This project uses depth data from the Intel RealSense 435i to compare the heights of detected objects, such as goods and operators. The goal is to determine whether the goods exceed the height of the operator, and if so, issue an alert. The process is straightforward and involves extracting depth information and performing a height comparison.

A. Extracting Depth Data for Detected Objects

The first step is to extract depth information for the detected objects. This involves the following:

1) *Object Detection*: Objects, such as goods and operators, are detected using bounding boxes generated by algorithms like YOLO. Each bounding box is defined by its top-left corner (x_1, y_1) and bottom-right corner (x_2, y_2).
2) *Region Mapping*: Once the bounding boxes are obtained, the corresponding regions in the depth map are identified. The depth value $D(x, y)$ at each pixel inside the bounding box represents the distance from the camera to that point.
3) *Filtering Noise*: To ensure accuracy, invalid or noisy depth values are filtered out. For example, depth values that are zero or exceed the camera's reliable range are discarded:

$$D(x, y) > 0 \quad \text{and} \quad D(x, y) < D_{\text{max}}$$

Here, D_{max} is the maximum reliable depth range of the camera (e.g., 10 meters for the Intel RealSense 435i).

B. Estimating Object Heights

The height of each detected object is estimated by analyzing the depth values within its bounding box:

- *Finding the Topmost Point*: The topmost part of an object is represented by the pixel with the smallest depth value (closest to the camera) within its bounding box. This value is calculated as:

$$D_{\text{min}} = \min_{(x,y)\in B} D(x, y)$$

where B is the set of all pixels inside the bounding box.

- *Calculating Height*: The height of the object relative to the camera is then estimated using the camera's mounting height. Let h_{cam} represent the height of the camera above the ground. The height of the object is given by:

$$h_{\text{object}} = h_{\text{cam}} - D_{\text{min}}$$

This formula assumes the camera is mounted parallel to a flat ground surface.

C. Comparing Heights and Sending Alerts

After estimating the heights of both the goods and the operator, their heights are compared to determine if an alert needs to be issued:

- *Operator Height*: The height of the operator is calculated in the same way as the height of the goods:

$$h_{\text{operator}} = h_{\text{cam}} - D_{\text{operator-min}}$$

where $D_{\text{operator-min}}$ is the smallest depth value in the operator's bounding box.

- *Height Comparison*: The difference in heights between the goods and the operator is calculated as:

$$\Delta h = h_{\text{goods}} - h_{\text{operator}}$$

If the height of the goods exceeds the height of the operator ($\Delta h > 0$), an alert is triggered.

- *Alert Mechanism*: When the condition $\Delta h > 0$ is satisfied, an alert is sent to warn that the goods are taller than the operator, which ensures that potentially unsafe situations, particularly handling goods that are too tall, are flagged for further attention.

D. Use in This Work

In this project, proximity estimation, based on depth, was used during the development stage to cross-validate the model outputs created by our LLM-based solution. Using the proximity values that should have come out of the depth data in relation to the model outputs, we have checked the system for accuracy and reliability. This provided a good benchmark against which we have been able to make improvements and refine the solution.

However, depth data was not included in the final implementation for the following reasons:

- **High Computational Requirements**: Depth data analysis requires a great quantity of computation—especially at high resolutions—which is unfeasible for real-time applications.
- **Large Volume of Information**: Depth information is naturally volume-consuming, requiring high storage capacity and bandwidth; this makes it inappropriate for light or resource-limited systems.
- **Latency Issues**: The time taken to process the depth data incurs significant latencies, hence these are not applicable to applications that require fast responses.

While the estimation of proximity based on depth was invaluable during the development process, its limitation barred its use in real time. The final system is optimized such that it performs well, independent of depth information, hence assuring better performance with scalability.

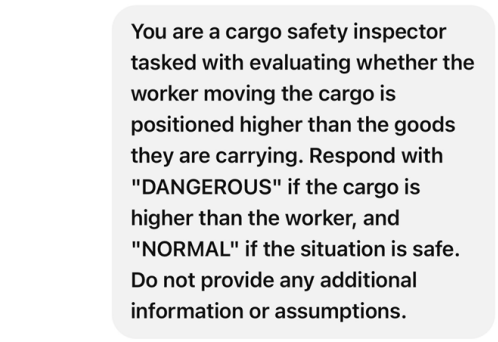

Fig. 13: Prompt used to instruct GPT-4o

IX. LLM-Based Danger Detection System

The idea for this system is inspired by the way humans assess whether an operation is dangerous. Humans rely on visual cues, understand the context of a situation, recognize dangerous patterns, and interpret instructions to make informed decisions. Similarly, many recent LLMs have advanced capabilities in understanding and reasoning. Multimodal models such as GPT-4 [5], LLaMA 3.2 [9], and Gemini [10] can process both textual and visual inputs, enabling them to comprehend their environment and make contextually appropriate judgments.

For our experiments, we tested with GPT-4o through the iOS application and employed prompt engineering to guide its responses. The prompt we designed is shown in Figure 13. It was carefully crafted to ensure clarity, specificity, and focus, enabling the model to perform the task effectively.

A. Analysis of the Prompt

The prompt is structured to maximize its clarity and effectiveness. It is divided into the following components:

1. **Role Definition:** The prompt begins with the statement, *"You are a cargo safety inspector tasked with evaluating..."*, which assigns a specific role to the model. This framing sets the context and ensures that the model approaches the task with the mindset of an inspector.

2. **Scenario Description:** The prompt clearly describes the situation to be evaluated: *"...whether the worker moving the cargo is positioned higher than the goods they are carrying."* This part eliminates ambiguity and ensures the model understands the specific task.

3. **Explicit Output Instructions:** The model is instructed to respond with specific keywords: *"Respond with 'DANGEROUS' if the cargo is higher than the worker, and 'NORMAL' if the situation is safe."* This ensures consistency in the format of responses and getting rid of elaborations that are not necessary.

4. **Guidance to Avoid Assumptions:** The last prompt, "Do not provide any additional information or assumptions," will stop the model from including extraneous details or conjectural data in the output, hence keeping it focused and concise, while avoiding hallucination.

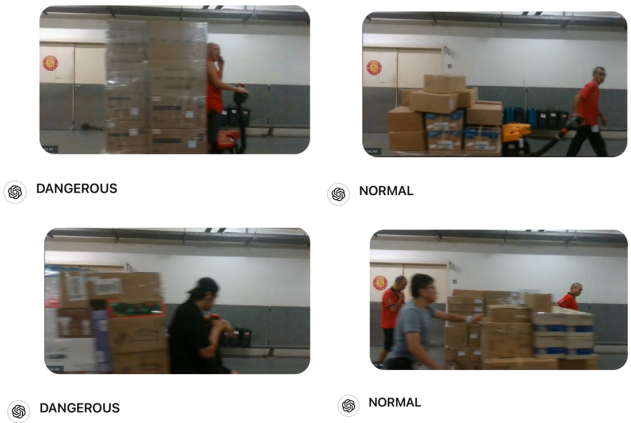

Fig. 14: Example of Response Provided

B. Performance and Observations

We evaluated the system in many different scenarios with both textual descriptions and visual inputs, emulating real-world conditions. The model gave accurate answers reliably, proving its ability in safety-evaluation of the environment. This can be attributed to the clarity and specificity of the prompt, which guided the reasoning process of the model well.

Figure 14 shows an example of the prompt and the model's output. These results demonstrate the potential of LLMs, guided by well-crafted prompts, for being effective tools in danger detection. Their ability to process and reason over multimodal data makes them particularly well-suited for such applications.

C. Model Deployment

This section describes the implementation of the danger detection system using both cloud-based and local environments. Each of these configurations has been specially implemented to test the efficacy of the system in different settings.

1) Cloud Deployment: Currently, it runs on the Azure AI, which provides the basic infrastructure so that the system seamlessly performs its tasks. We used GPT-4-0613 with specific prompts and parameters tailored to the task. The cloud configuration of Azure ensures that the system has the ability to make real-time evaluations and scale if needed, hence making it feasible for practical applications.

2) Local Deployment: We also experimented with running the system locally using the OLLAMA framework [11], as shown in Figure 15. For this, we deployed the `meta-llama/Llama-3.2-11B-Vision` model [12]. This version of Llama showed impressive multimodal capabilities, such as understanding the context of a scene and making reasonable judgments. However, it occasionally made incorrect assessments, which may be due to its smaller size and limited capacity compared to larger models.

Unfortunately, we were unable to test the larger `meta-llama/Llama-3.2-90B-Vision` model due to computational constraints [9]. This version, with its significantly higher number of parameters, is expected to perform better, as larger models generally have greater accuracy and context comprehension. However, running such a model requires substantial hardware resources, which were not available during our testing.

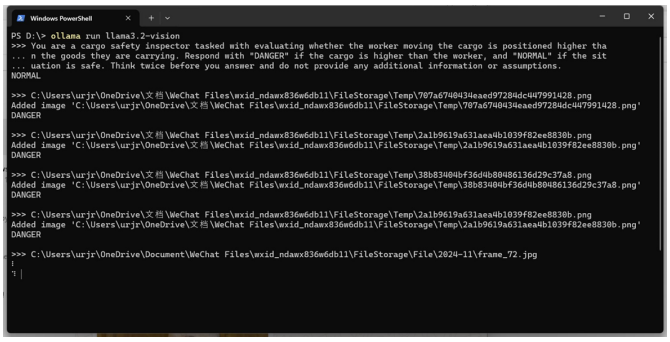

Fig. 15: Example of Local Deployment

We plan to explore the 90B version in the future, as it is likely to provide more reliable and accurate results compared to the 11B version. For now, the 11B model offers a good balance between performance and resource requirements while still demonstrating the potential of local deployment for tasks like danger detection.

X. Automated Telegram Alert System

The automated Telegram alert system, based on Telegram Bot API [13], serves as a crucial component for real-time notifications to ensure timely responses to detected dangers. The system is built to authenticate users, detect dangerous activities, and promptly notify the relevant staff with actionable information.

A. User Authentication

To ensure data security and prevent unauthorized access, the Telegram bot requires users to log in before receiving alerts. The authentication process involves entering a username and passcode, as shown in Figure 16. Upon successful login, the system greets the user and asks for confirmation to enable alert notifications.

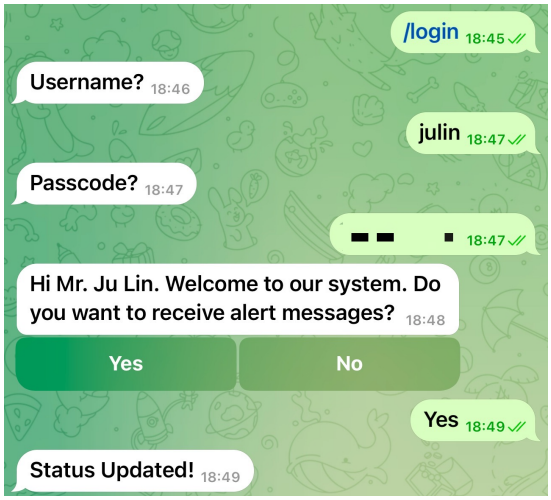

Fig. 16: Telegram bot authentication process.

B. Alert Notification System

Once the system detects a dangerous activity using the LLM and classifies it as hazardous, an alert message is sent

automatically to the corresponding staff. The alert provides the following details:

- **Time of detection:** The exact timestamp of the detected event.
- **Camera number:** The identification of the camera that captured the incident.
- **Screenshot:** A snapshot of the detected dangerous activity.
- **Video link:** A hyperlink to a 5-second video clip around the time of detection for further review.

Figure 17 shows examples of alert messages sent by the Telegram bot. Each message highlights the detected danger with clear and concise details, including a visual reference and a link for a video clip, which contains +- 5 seconds of the detected frame.

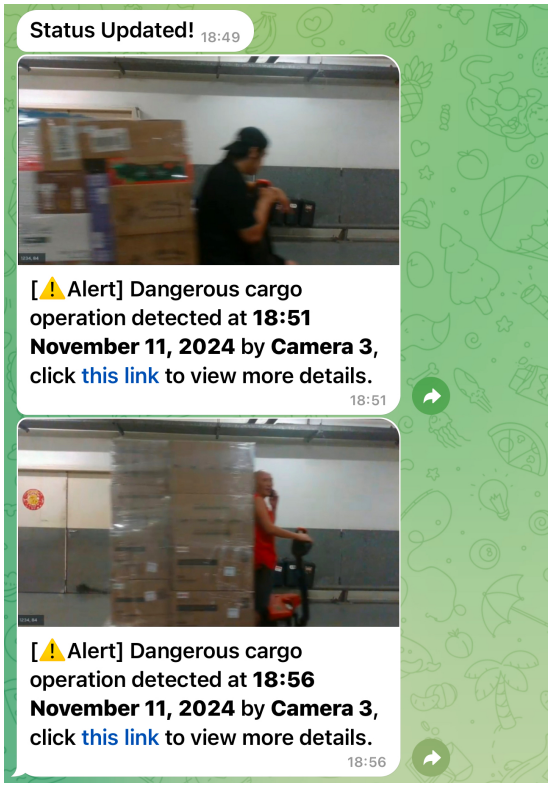

Fig. 17: Examples of Telegram alert messages containing detection details.

This notification system enhances the operational safety of the cargo area by providing real-time monitoring and alerts, enabling staff to take immediate corrective actions when necessary.

XI. Summary

This project realizes an automated cargo safety detection at the HKIA. It combines YOLO for object detection, depth estimation for validation, LLM-based danger detection and Telegram alert system. However, due to the limited time, we do not have enough dataset to train the model,so there are still some error in the object detection. Our future work will focus on improving the model performance so that it can handle more complex cargo situation with greater accuracy. We will conduct multiple experiments at the airports to ensure that the system can be used in the airports.

References

[1] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 779–788.
[2] Ultralytics, "Ultralytics yolo11," https://github.com/ultralytics/ultralytics, 2024, accessed: 2024-11-21.
[3] R. Khanam and M. Hussain, "Yolov11: An overview of the key architectural enhancements," *arXiv*, 2024. [Online]. Available: https://arxiv.org/html/2410.17725v1
[4] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," *OpenAI*, 2018, available: https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf.
[5] OpenAI, J. Achiam, S. Adler, S. Agarwal, L. Ahmad *et al.*, "Gpt-4 technical report," *arXiv*, vol. cs.CL, 2024, arXiv:2303.08774. [Online]. Available: https://doi.org/10.48550/arXiv.2303.08774
[6] Intel Corporation, "Intel®realsense™sdk," https://github.com/IntelRealSense/librealsense, 2024, accessed: 2024-11-21.
[7] B. Dwyer, J. Nelson, T. Hansen *et al.*, "Roboflow (version 1.0)," 2024, computer vision software. [Online]. Available: https://roboflow.com
[8] OpenCV Team, "Open Source Computer Vision Library," 2024, accessed: 2024-11-21. [Online]. Available: https://github.com/opencv/opencv
[9] Meta Platforms, Inc., "Llama 3.2 90b vision · hugging face," 2024, accessed: 2024-11-21. [Online]. Available: https://huggingface.co/meta-llama/Llama-3.2-90B-Vision
[10] R. Anil, S. Borgeaud, J.-B. Alayrac *et al.*, "Gemini: A family of highly capable multimodal models," *arXiv preprint arXiv:2312.11805*, 2023. [Online]. Available: https://arxiv.org/abs/2312.11805
[11] Ollama Contributors, "Github - ollama: Get up and running with llama 3.2, mistral, gemma 2, and other large language models," 2024, accessed: 2024-11-21. [Online]. Available: https://github.com/ollama/ollama
[12] Meta Platforms, Inc., "Llama 3.2 11b vision · hugging face," 2024, accessed: 2024-11-21. [Online]. Available: https://huggingface.co/meta-llama/Llama-3.2-11B-Vision
[13] Telegram, "Telegram bot api," n.d., accessed: 2024-11-21. [Online]. Available: https://core.telegram.org/bots/api

XII. Contributions

All team members contributed equally to the project. As requested, the detailed distribution of tasks is listed below:

- **Ju Lin**: Data Collection, Depth Extraction & Analysis, LLM-Based Danger Detection System Development and Deployment, Telegram Alert System Implementation.
- **Rouheng Ni**: Data Collection, RGB Data Extraction, Depth Data Extraction.
- **Ruilin Shi**: YOLO Deployment, Depth Data Analysis.
- **Shuxuan Li**: Data Collection, YOLO Deployment.
- **Jiayi Zuo**: Data Collection, YOLO Labeling and Training.