YOLO-Based Hazard Detection and Safe Landing Zone Identification for Drones

PRATHAMESH CHATORIKAR, University of California, Santa Cruz, USA

ACM Reference Format:
Prathamesh Chatorikar. 2025. YOLO-Based Hazard Detection and Safe Landing Zone Identification for Drones. 1, 1 (March 2025), 19 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

This project focuses on helping drones find safe places to land during emergencies. I used YOLO, a fast object detection model, to detect important objects like rooftops, fire, smoke, and debris from aerial images. The goal is to avoid landing in dangerous areas.

I trained YOLO using a custom aerial dataset with labeled classes such as `rooftop`, `debris`, and `fire`. Based on YOLO's output, I developed a custom algorithm to check if a rooftop is safe. If no safe rooftop is found, the system uses a language model to suggest an alternate landing spot.

The next sections explain the detection method, the decision algorithm, and how results were analyzed.

2 Related Work

Recent advancements in drone-based fire detection have leveraged the YOLO (You Only Look Once) object detection framework to enhance the identification of fire and smoke in diverse environments.

Researchers have proposed lightweight models suitable for UAV deployment, focusing on early forest fire detection and real-time hazard identification:

- A study introduced **LUFFD-YOLO**, based on YOLOv8n, which incorporates attention mechanisms and multi-level feature fusion to improve detection accuracy while reducing computational complexity [MDPI, 2024].
- Another approach proposed an improved YOLOv8 variant by replacing full convolution with local convolution in the C2F module and adding an EMA module. This design enhanced feature channel interaction and improved utilization of contextual information [Nature, 2025].
- The development of **YOLOv8-EMSC**, a lightweight fire recognition model, has shown performance gains over traditional detectors. It adopts a deeper architecture for better detection across large-scale scenes [ScienceDirect, 2024].

These efforts collectively demonstrate that modern YOLO-based models can serve as reliable tools for integrating hazard detection into drone-based aerial scene understanding systems.

Author's Contact Information: Prathamesh Chatorikar, pchatori@ucsc.edu, University of California, Santa Cruz, USA.

Manuscript submitted to ACM

3 Design

3.1 Problem Definition

The goal is to find the safest place for a drone to land by analyzing aerial images. The drone must avoid landing on hazardous areas like fire, smoke, debris, or unstable rooftops. This needs to happen fast and accurately to work in real-time. The main challenge is that some areas may look safe but are not, so we must check them carefully.

3.2 Solution Approach

The system uses multiple steps to detect safe landing zones and avoid hazards. Each step helps to narrow down the best place to land. The process is as follows:

(1) **Dataset Creation and Model Training**
- A custom dataset of drone images was created using publicly available sources and manual image collection.
- All images were manually labeled using bounding boxes to mark rooftops, fire, smoke, debris, ships, and solar panels.
- Data augmentation was applied (e.g., rotation, scaling, cropping) to improve generalization.
- The model was trained using YOLOv11 nano on an NVIDIA T4 GPU for fast performance and lower memory usage.
- A unified dataset was built by modifying the `data.yaml` file to ensure consistent class labels across all images.

(2) **Input Image and Preprocessing**
- Aerial images taken from a drone are used as input.
- Images are resized and normalized to prepare them for detection.

(3) **Object Detection using YOLO**
- I used a fine-tuned YOLOv11 model to detect objects in the image.
- The model identifies safe zones like rooftops or ships and hazards like fire, smoke, debris, and solar panels.
- It returns bounding boxes for each object with class names and confidence scores.

(4) **Rooftop Safety Check (Algorithm 1)**
- For each rooftop found by YOLO, my algorithm checks if it is safe.
- A rooftop is safe if:
  - At least three out of four corners are outside any hazard box.
  - No hazard center lies completely inside the rooftop.
- If any rooftop passes these checks, it is marked as a safe landing zone.

(5) **LLM-Based Backup Plan**
- If no safe rooftop or ship is found, I call a Large Language Model (LLM).
- I send it a list of all detected objects and their positions using a structured prompt.
- The LLM finds a safe place to land that avoids all hazards.
- If no objects are detected, the full image is sent to the LLM for analysis.

(6) **Drawing Final Output**
- The final chosen landing point is shown using a green 4x4 box.
- I also draw red boxes for hazards and mark unsafe rooftops separately.

4 Challenges

4.1 Main Technical Challenges

The goal of this project is to combine object detection with reasoning to help drones find safe landing zones in emergency situations. Doing this requires building a reliable detection system using YOLO and integrating it with a Large Language Model (LLM) for decision-making when object detection alone is not enough. Several technical challenges were encountered during this work:

- **Training YOLO on a Balanced Dataset**
  The first challenge was to train YOLO effectively. Initial training was done on a single dataset with around 6,000 images. This led to overfitting—YOLO could detect rooftops well but failed to generalize to other objects like fire or debris. Fine-tuning the model sequentially on new datasets caused it to forget older knowledge (a problem known as catastrophic forgetting). To solve this, all datasets were merged into one, and the `data.yaml` configuration was updated to maintain consistent class labels.
- **False Positives in Hazard Detection**
  Detecting fire and smoke worked well, but debris was often misclassified due to limited training examples. This caused false positives—especially when similar-looking rooftop areas were detected as debris. The solution was to label more diverse samples and apply manual corrections, ensuring that the dataset included clear examples of safe vs. unsafe rooftops.

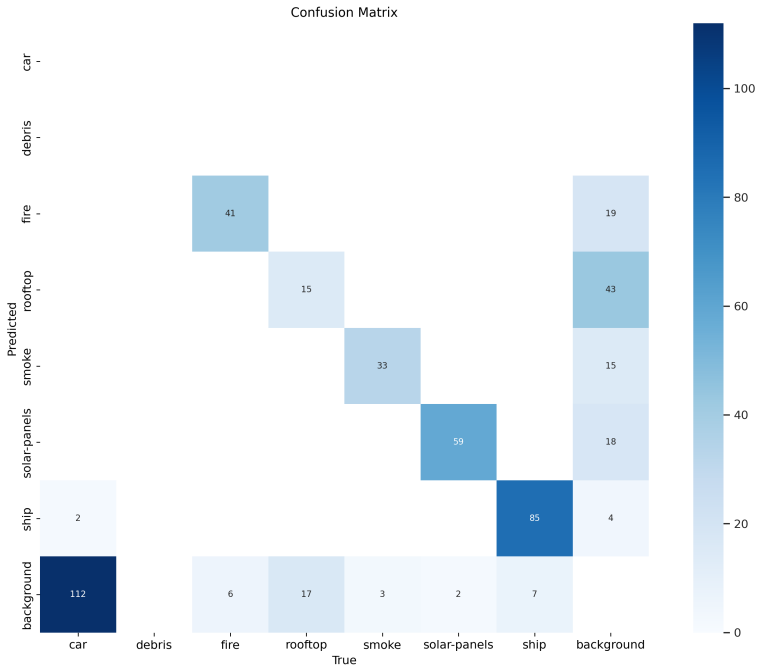


Fig. 1. Confusion matrix

- **Overlapping Bounding Boxes**
  Sometimes the YOLO model predicted fire or debris in the same location as a rooftop. This made it difficult to decide if a rooftop was really safe. To solve this, a custom filtering algorithm (Algorithm 1) was created. It checks each rooftop and ensures that:
  - At least three corners of the rooftop are outside any hazard box.
  - No hazard center is completely inside the rooftop.
- **Labeling and Augmentation Issues**
  Manual labeling was time-consuming. Augmenting the dataset (rotation, flipping, etc.) helped model generalization, but it made it harder to keep bounding boxes accurate. Custom scripts were written to fix bounding boxes after augmentation so that the training data remained reliable.
- **Efficient Use of the LLM for Decision-Making**
  YOLO is fast but can't reason. The LLM is good at reasoning, but slow and expensive to run. Initially, the LLM was asked to process full images, but this took too long. The solution was to pass only detection metadata (like bounding boxes and labels) to the LLM. The LLM is only called when:
  - No safe rooftop is found by YOLO.
  - All detected rooftops overlap with hazards.
  - No objects are detected at all (fallback to full-image LLM).

4.2 Solutions to Address These Challenges

To overcome the above problems, the following solutions were implemented:

- **Unified Dataset Creation and Labeling**
  All datasets were combined and the `data.yaml` file was edited to define a consistent label set. New labels such as `rooftop` and `hazardous rooftop` were added. This ensured the model learned both safe and unsafe scenarios. The labeled dataset is available at Roboflow Project Page and code is available on GitHub.
- **Algorithm 1 for Filtering Unsafe Rooftops**
  To improve decision quality, Algorithm 1 checks if a rooftop is far enough from hazards. It uses geometric rules to filter out rooftops that are too close to fire, smoke, or debris. This helps reduce the chances of choosing a dangerous landing zone.
- **Selective Use of LLMs**
  Instead of always sending the full image to the LLM, object metadata (YOLO outputs) is passed as JSON-like input. This reduces the amount of data processed by the LLM, improving speed. Full-image prompts are only used when detection completely fails.
- **Dataset Preprocessing and Label Fixing**
  A preprocessing pipeline was developed to fix label inconsistencies across datasets and align class names. This helps the YOLO model learn better from mixed sources without confusion.
- **Tradeoff Between Speed and Accuracy**
  Three different pipelines were compared:
  - **YOLO Only:** Fastest but not always safe.
  - **YOLO + LLM (Using Metadata):** Best overall tradeoff between speed and reasoning.
  - **LLM Only (Full Image):** Accurate in complex cases but too slow for real-time use.

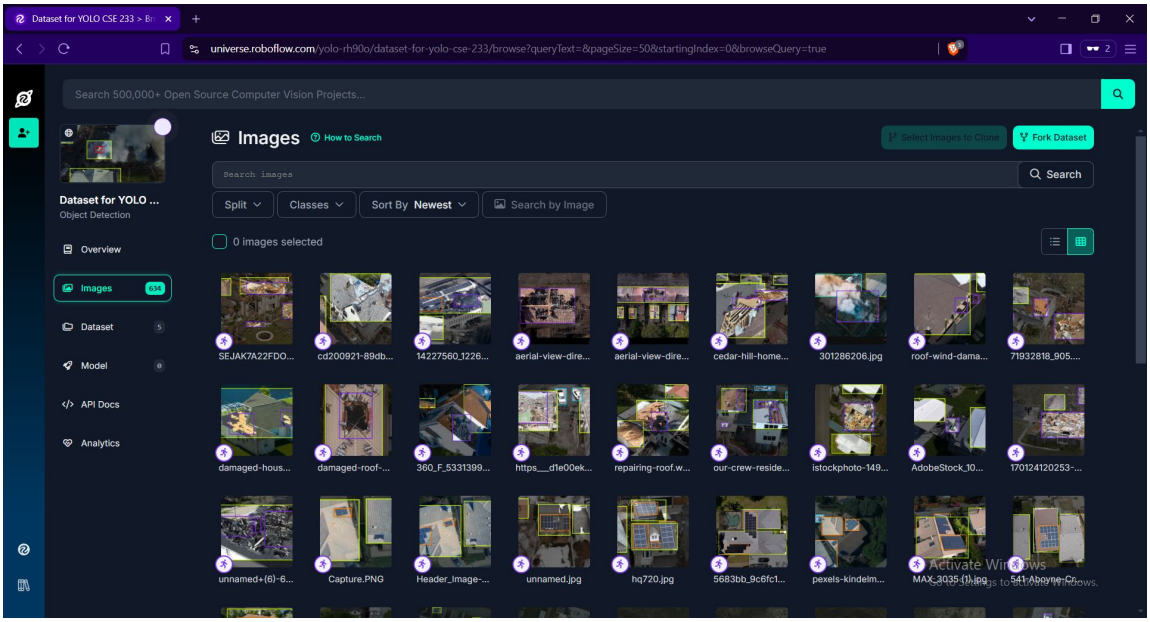


Fig. 2. Custom labeled dataset with safe and unsafe rooftop classes.

By solving these challenges, the final model is able to detect safe landing zones accurately, reason about hazards, and make decisions even when YOLO is uncertain. The hybrid approach ensures both speed and safety, making it suitable for real-time drone applications.

5 Feedback Implementation from Midterm report

5.1 Scenario Coverage

To address the feedback received, I focused on building a systematic dataset of diverse landing scenarios and validating object detection across these scenarios. The model used for these results is YOLOv11-Nano. The model was trained on a custom-labeled dataset consisting of both safe and hazardous rooftop conditions.

Training Details:

- Architecture: YOLOv11-Nano
- Training Set: Custom aerial dataset (rooftops, hazards, ships)
- Classes used: rooftop, solar-panels, `fire`, smoke, debris, ship
- Output: `best.pt` weights file saved after training and evaluation on validation set
- GitHub Link for Weights: https://github.com/Chatorikar/CSE233-YOLO

Model Limitations:

- Due to the limited number of labeled examples, the confidence score in some outputs is relatively low.
- Overlapping classes like `solar-panels` on rooftops created ambiguous detections during evaluation.

Scenarios Created and Labeled (as per feedback):

(1) Rooftop with fire and smoke

(2) Rooftop cluttered with debris
(3) Rooftop covered with solar-panels
(4) Ship floating in open water for emergency drone landing

Detection Output Samples:

(a) Aerial drone image (b) YOLO image processing

Fig. 3. Rooftop with Fire and Smoke

6 YOLO-Based Safe Landing Detection (Without LLM)

In this section, I explain how my system can find a safe landing spot using only the YOLO object detection model and my custom safety-check algorithm, without any help from the Large Language Model (LLM).

6.1 How YOLO Helps Identify Landing Zones

In my case, I fine-tuned YOLO to detect the following:

- **Safe Zones:** `rooftop`, `ship`
- **Hazardous Zones:** `fire`, `smoke`, `debris`, `solar-panels`

Once YOLO processes an image, it returns bounding boxes and labels for each object. These boxes tell me the position and size of each detected item.

6.2 My Algorithm: Determining if a Rooftop is Safe

After YOLO detects rooftops and hazards, my custom algorithm (Algorithm 1) checks each rooftop to decide whether it is safe to land on. Here's how it works in simple steps:

(1) For every detected `rooftop`, I calculate the four corners of its bounding box.
(2) For every detected `hazard` (like fire or debris), I check two things:

(a) Aerial drone image (b) YOLO image processing

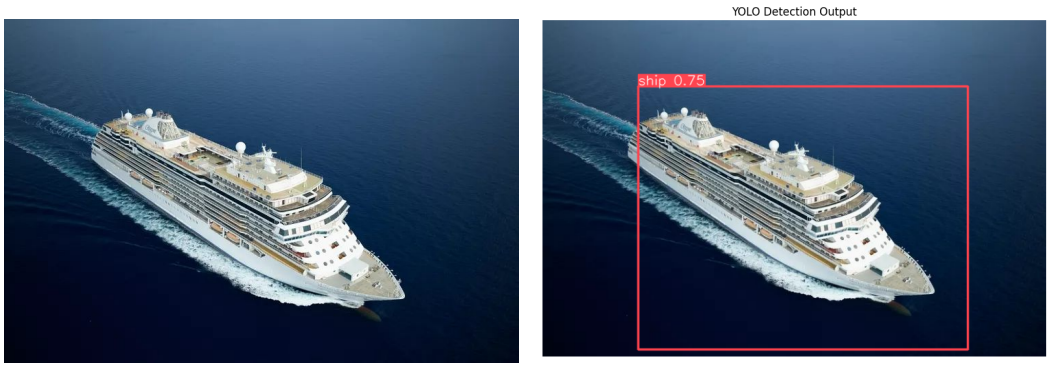
Fig. 4. Rooftop with Debris

(a) Aerial drone image (b) YOLO image processing

Fig. 5. Rooftop with Solar Panels

- **Corner Rule:** At least **three out of four** rooftop corners must be outside all hazard boxes.
- **Center Rule:** The center of any hazard must not be inside the rooftop box.

(3) If a rooftop passes both rules, I consider it **safe for landing**.



(a) Aerial drone image (b) YOLO image processing

Fig. 6. Ship Detection in the Ocean for Emergency Landing

Algorithm 1 Safe Landing for Drone via YOLO + LLM

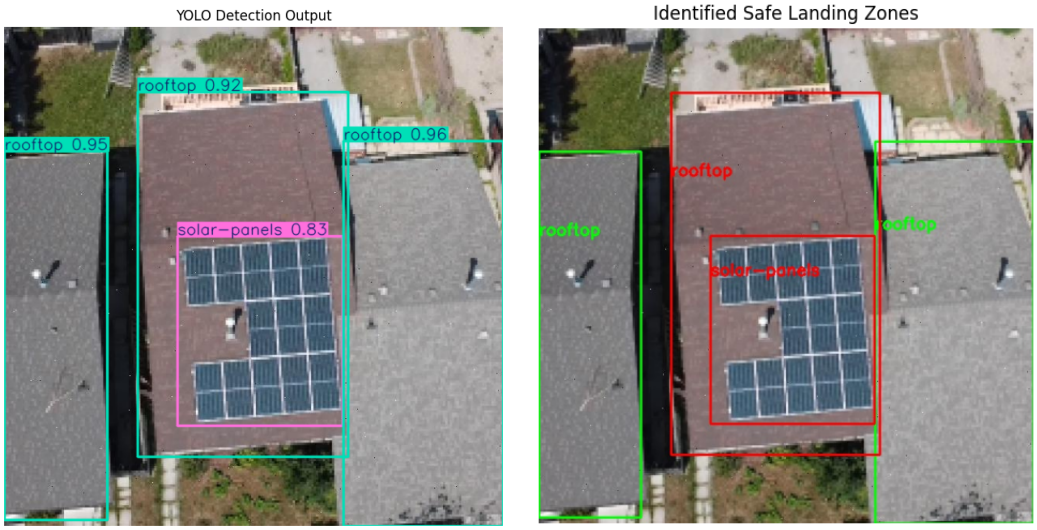
Require: Aerial images $\mathcal{I} = \{I_1, I_2, \ldots, I_n\}$, YOLO model M, LLM $\mathcal{L}$, class sets $\mathcal{S}$ (safe), $\mathcal{H}$ (hazardous)
Ensure: Predicted optimal landing coordinates (x^*, y^*) for each $I_i \in \mathcal{I}$

Initialize: Load model M and LLM $\mathcal{L}$; define $\mathcal{R} \leftarrow \emptyset$
for each input image $I_i \in \mathcal{I}$ **do**
  $D_i \leftarrow M(I_i)$ ▷ Run YOLO on I_i to obtain detections $D_i = \{d_1, d_2, ..., d_m\}$
  $\mathcal{S}_i \leftarrow \{d_j \in D_i \mid c_j \in \mathcal{S}\}$ ▷ Extract candidate safe zones (rooftops, ships)
  $\mathcal{H}_i \leftarrow \{d_j \in D_i \mid c_j \in \mathcal{H}\}$ ▷ Extract hazardous regions (fire, smoke, debris)
  $(x^*, y^*) \leftarrow \emptyset$
  if $\mathcal{S}_i \neq \emptyset$ **then**
    for each candidate $s \in \mathcal{S}_i$ **do**
      if `isSafe`$(s, \mathcal{H}_i)$ = `True` **then**
        $(x^*, y^*) \leftarrow$ `Center`(s)
        break
      end if
    end for
    if $(x^*, y^*) = \emptyset$ **then**
      $(x^*, y^*) \leftarrow \mathcal{L}(\mathcal{S}_i, \mathcal{H}_i)$ ▷ Fallback to LLM with detection metadata
    end if
  else
    $(x^*, y^*) \leftarrow \mathcal{L}(I_i)$ ▷ No objects detected; full image reasoning via LLM
  end if
  if $(x^*, y^*) \neq \emptyset$ **then**
    Render 10×10 green box centered at (x^*, y^*) on I_i
  end if
  Append $\mathcal{R} \leftarrow \mathcal{R} \cup \{(I_i, D_i, (x^*, y^*))\}$
end for
Postprocessing: Export $\mathcal{R}$ to CSV, generate visualizations and timing analytics

This logic is important because a rooftop may be partially covered by fire or debris, and a simple object label is not enough to say it's safe. My algorithm goes one level deeper and checks their spatial relationship.

6.3 Visualizing Safe Zones Detected by YOLO

If at least one safe rooftop is found, I draw a green box on the image to show the safest landing spot.



(a) YOLO-detected bounding boxes with hazards and rooftops

(b) My algorithm highlighting the final safe landing zone

Fig. 7. Safe Landing Zone Detected Using YOLO and Spatial Logic Only (No LLM)

As shown above, YOLO detects hazards and rooftops, and then my algorithm decides whether the rooftop is safe. In this example, the rooftop is not completely covered by fire or smoke, so the system marks it as a safe landing zone using a green box.

6.4 Why This Step Matters

This YOLO-only pipeline is fast, lightweight, and works well when the scene is clear and rooftops are easily distinguishable from hazards. It avoids the extra cost and time of calling a large language model. In many cases, this basic step is enough to guide a drone to land safely without any further reasoning.

In the next section, I describe how I use a Large Language Model (LLM) when YOLO alone is not enough — for example, when no safe rooftop is clearly visible.

6.5 Time Complexity and Performance

One of the biggest advantages of using YOLO is its speed. The YOLO model processes an entire image in a single forward pass through a neural network, making it extremely fast.

In my case, each aerial image takes approximately **0.1 to 0.2 seconds** to run through YOLO on a GPU. This makes it suitable for real-time drone applications where fast decisions are necessary.

The total time complexity of the YOLO-based detection can be approximated as:

$$O(N) \quad \text{where } N \text{ is the number of grid cells processed during detection}$$

(a) YOLO-detected bounding boxes with hazards and rooftops (b) My algorithm highlighting the final safe landing zone

Fig. 8. Safe Landing Zone Detected Using YOLO and Spatial Logic Only (No LLM)

This is because YOLO scans the image in a grid-like structure, and each cell is checked only once.

My additional logic for determining safe rooftops (from Algorithm 1) compares each rooftop to every hazard, which results in:

$$O(r \cdot h) \quad \text{where } r \text{ is the number of rooftops, and } h \text{ is the number of hazards}$$

In practice, both r and h are small (usually less than 10), so the overall algorithm remains lightweight and fast.

6.6 Dataset Dependency

Although YOLO is fast, its performance depends on the quality and balance of the training dataset. My model was trained on a custom dataset containing aerial views of:

- Rooftops in different shapes and sizes
- Ships in water bodies
- Fire, smoke, debris, and solar panels on buildings

If the dataset does not contain enough variety—especially in lighting, angles, or rare hazards like rooftop fires—YOLO might:

- Miss detecting an object (false negative)
- Detect the wrong class (false positive)
- Give poor bounding box accuracy

To improve this, I carefully merged multiple datasets, corrected their labels, and used data augmentation. However, the YOLO-only pipeline still has some limitations when it encounters unseen or ambiguous objects.

6.7 Pros and Cons of YOLO-Only Safe Landing Detection

Pros:

- **Fast Inference:** YOLO runs in real time and does not need external computation (like cloud-based LLMs).
- **Lightweight:** Can run on edge devices like drones or Jetson boards.
- **Deterministic:** The decisions are repeatable and do not rely on stochastic language models.
- **Good for Clear Scenes:** Performs well when hazards and rooftops are clearly visible.

Cons:

- **Limited Reasoning:** YOLO cannot understand abstract relationships or make higher-level decisions.
- **Sensitive to Dataset Quality:** Performance drops if the training data lacks coverage of real-world edge cases.
- **Fails in Ambiguous Cases:** If a rooftop is partially occluded by smoke or debris, YOLO might detect it but not understand if it's actually safe.
- **No Fallback:** If no safe object is detected, YOLO cannot "guess" or recommend any alternative safe region.

Because of these limitations, I use YOLO as the first step. When it fails to confidently find a safe landing spot, I pass the results to a Large Language Model (LLM) for further reasoning and spatial decision-making, which is discussed in the next section.

7 YOLO + LLM: Final Hybrid Safe Landing System

Following the individual evaluations of YOLO and LLM-based approaches, I developed a combined solution that integrates the strengths of both methods. The goal was to address the limitations observed in standalone systems—YOLO's lack of reasoning and the LLM's relatively high inference time—and to create a reliable pipeline for identifying safe drone landing zones under varying environmental conditions.

7.1 Post-Detection Reasoning: Step-by-Step Decision Making Using Prompt Engineering

After all objects in an aerial image are detected using YOLO—including both potential safe zones (e.g., rooftops, ships) and hazardous ones (e.g., fire, smoke, debris)—the next task is to find a safe landing coordinate for the drone. In situations where no safe rooftop is confidently identified, I use a Large Language Model (LLM) to help make that decision. Instead of scanning the full image directly, I pass a structured summary of detected objects as metadata to the LLM, which improves performance and reduces cost.

(1) **Step 1: Detection Metadata Preparation**
YOLO provides detection results in nearly constant time $O(1)$ due to its single-shot architecture. Each detected object includes a class label (e.g., "rooftop," "fire") and a normalized bounding box. I group these results into:
   - **Safe candidates:** rooftops, ships
   - **Hazardous zones:** fire, smoke, debris, solar panels

   If no safe candidates are found—or if all of them are flagged as unsafe by my custom Algorithm 1—this metadata is passed to the LLM.

(2) **Step 2: Constructing the LLM Prompt**
I crafted a prompt using this metadata. The prompt instructs the LLM to:
   - Avoid placing landing coordinates inside any hazardous bounding boxes.
   - Prefer rooftops or ships if they are available and not affected by hazards.

- Return only the (x, y) coordinates of a safe location in normalized format (values between 0 and 1).

An example of this prompt is shown in Figure 9.

(3) **Step 3: Sending Metadata to LLM**
The detection metadata is serialized into JSON format and submitted to GPT-4 Turbo through an API call. This metadata-driven approach avoids full image scans and uses textual spatial understanding instead.

(4) **Step 4: LLM Response Processing**
The LLM replies with a suggestion in the form of coordinates:
`"x": 0.52, "y": 0.33`
I use JSON parsing or regex as a fallback to extract this information.

(5) **Step 5: Visual Confirmation**
The selected point is drawn on the original image using a small green bounding box to indicate the recommended safe landing zone. This visual output helps verify the result and cross-checks whether the chosen location overlaps any hazard.

```python
# LLM Decision Function (Safe Landing Selection)
def ask_llm_for_safe_landing(safe_rooftops):

    print("checking if calling")
    prompt = f"""
     We have detected multiple hazardous areas in the image, including fire, smoke, debris, and other obstructions.
    The detected hazardous coordinates are as follows:
    {json.dumps(safe_rooftops, indent=4)}

    Your task is to determine the safest possible landing spot for a drone while strictly following these conditions:

    1. Avoid Hazardous Zones:
      - The landing spot must not be within any of the provided hazardous coordinates.
      - It must be at a safe distance from fire, smoke, or debris to ensure a secure landing.

    2. elect a Rooftop or Ship (if available):
      - If rooftops or ships exist in the image, prioritize landing on them.
      stricly outside hazardous coordinates
    3. Final Output - Coordinates for Safe Landing:
      - Provide the final (x, y) coordinates of the best possible landing spot.
      - The coordinates must be in normalized format (ranging between 0 to 1) relative to the image size.

    Ensure that the final coordinates strictly meet the above safety criteria and do not overlap with any hazardous region provided in the list.
    and those final coordinates strictly outside hazardous bounding box coordinates, and only give there coordinates with no explination, retrun coordinates
    """

    try:
        response = client.chat.completions.create(
            model="gpt-4-turbo",
            messages=[{"role": "system", "content": "You are an expert drone navigation assistant."},
                      {"role": "user", "content": prompt}]
        )
        response_text = response.choices[0].message.content
        print(response_text)
        return response_text, extract_coordinates(response_text)
    except Exception as e:
        return f"Error: {str(e)}", None
```

Fig. 9. LLM prompt for identifying safe landing zones outside detected hazard areas.

Benefits of Using Prompt Engineering. Prompt engineering helps reduce the need for passing full drone images to the LLM. Instead, I extract only essential object-level metadata (like location and type of hazard) and shape it into a task-focused prompt. This approach has the following advantages:

- **Faster Inference:** LLM processes structured metadata faster than raw image descriptions.
- **Lower Cost:** Avoids high compute cost tied to full image tokenization or vision-LM integration.

- **Better Accuracy:** By guiding the model with explicit instructions, the LLM focuses only on the relevant problem—safe landing selection.
- **Cleaner Output:** I instruct the model to return just coordinates—without any explanation—making downstream processing easier.

This hybrid reasoning process combines YOLO's fast detection with LLM's reasoning abilities. The result is a safe landing zone decision that is accurate, computationally efficient, and visually explainable. The actual evaluation results and comparisons will be discussed in the next section.

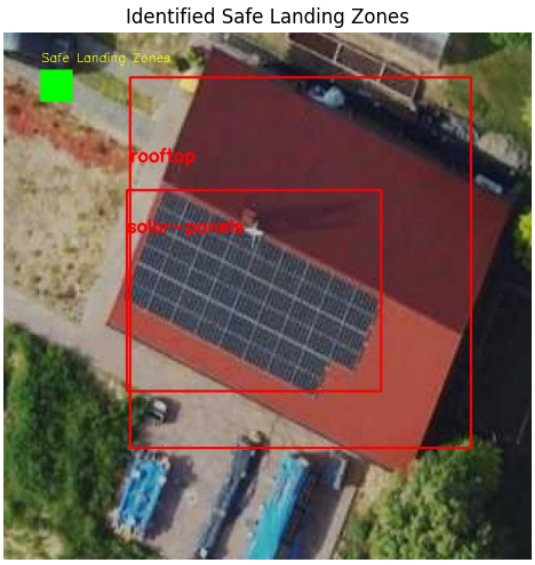


Fig. 10. YOLO+LLM: Final decision with safe landing spot (green box).

Fig. 11. YOLO+LLM: Final decision with safe landing spot (green box).

7.2 Motivation and Design Rationale

Earlier implementations that relied solely on YOLO performed well in simple scenarios. However, they struggled in more complex cases—such as when hazards overlapped rooftops. Meanwhile, using only the LLM added unnecessary delay and resource overhead in straightforward images.

The hybrid system was designed to overcome these challenges by:

- Reducing reliance on LLMs unless absolutely necessary
- Leveraging YOLO for fast, structured detection
- Using the LLM's reasoning ability to interpret ambiguous or complex scenes

7.3 Advantages of the Hybrid System

- **Efficient Resource Use:** The LLM is only queried when YOLO cannot confidently identify a safe rooftop. This minimizes unnecessary computational overhead.

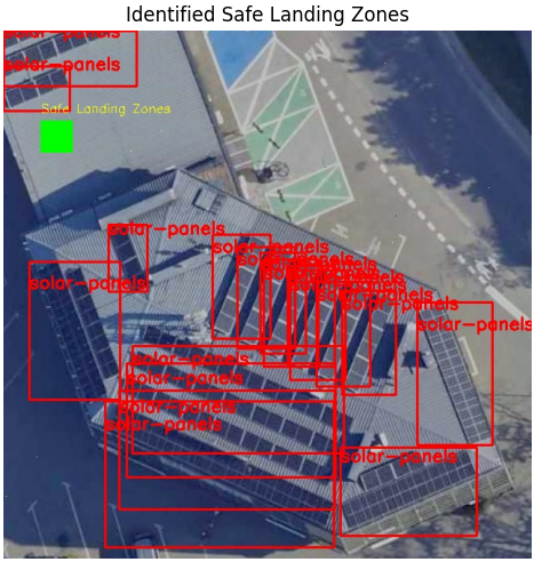


Fig. 12. YOLO+LLM: Final decision with safe landing spot (green box).

Fig. 13. YOLO+LLM: Final decision with safe landing spot (green box).

- **Improved Accuracy:** The combination of perception (YOLO) and reasoning (LLM) improves the system's robustness in a wide variety of scenarios.
- **Adaptability:** The system performs reliably across both clean and cluttered aerial scenes.
- **Explainability:** By incorporating LLM responses, the system offers a level of interpretability and justification for its decisions.

7.4 Limitations

Despite its benefits, the hybrid system has certain drawbacks:

- **Inference Time Variability:** LLM queries introduce latency, particularly in fallback cases where full-image processing is required.
- **Model Dependency:** The system relies on the correct functioning of both models. A failure in either YOLO or the LLM pipeline can affect output quality.
- **Prompt Sensitivity:** The output of the LLM may vary depending on the phrasing and structure of the prompt, requiring strict formatting.

8 Experimental Design and Results

8.1 Experimental Setup

To evaluate the effectiveness and efficiency of my hybrid safe landing system, I controlled experiments across four aerial images with varying levels of rooftop visibility and hazard density. The objective was to compare three configurations:

Source Code: The complete implementation, including the dataset, YOLO model weights, and LLM integration scripts, is available at: https://github.com/Chatorikar/CSE233-YOLO

(1) **YOLO Inference Only** – Direct object detection without reasoning.
(2) **YOLO → LLM (Metadata)** – YOLO detects objects and passes hazard metadata to an LLM for final decision-making.
(3) **LLM on Full Image** – The entire image is passed to the LLM for direct spatial inference.

Each image followed the same pipeline and was evaluated independently. Timing measurements were recorded for each configuration using Python's `time` module.

8.2 Algorithmic Flow

- **Step 1: Load image** from predefined test image paths.
- **Step 2: Run YOLO** object detector to classify all visible regions (rooftops, fire, smoke, debris, etc.).
- **Step 3: Measure YOLO detection time** using `measure_processing_time()`.
- **Step 4: Pass detected object metadata** to LLM via prompt engineering.
- **Step 5: Run LLM on metadata only**, and record time taken to generate coordinate response.
- **Step 6: Run LLM on full image** without any prior information to simulate worst-case fallback.
- **Step 7: Overlay bounding boxes and mark landing spot**.
- **Step 8: Store results and timings**.

8.3 Evaluation Metrics

Two metrics were used to analyze performance:

- **Processing Time (Seconds):** Measures computational delay from input to decision.
- **Landing Accuracy (Qualitative):** Assessed visually based on green box placement in output images.

8.4 Results and Discussion

Key observations based on Figure 14:

- **YOLO-only inference is the fastest**, with average times under 0.5s. This is expected since YOLO operates in a single forward pass.
- **YOLO + LLM (Metadata)** is significantly faster than full image prompting:
  - Because metadata is structured and narrowed down, the LLM reasoning is bounded and faster.
  - Average latency for this mode is 0.8s–1.0s.
- **LLM (Full Image)** is the slowest:
  - The model must analyze pixel-level data without prior knowledge.
  - Takes up to 1.8s per image on average.
- **Hybrid advantage:** YOLO+LLM (Metadata) offers the best trade-off:
  - Faster than full scan.
  - Safer than YOLO-only when hazards are present.

9 Analysis of Experimental Results

The focus is on processing time and decision accuracy across three methods: (1) YOLO-only, (2) YOLO + LLM (Metadata), and (3) LLM-only (Full Image Scan).

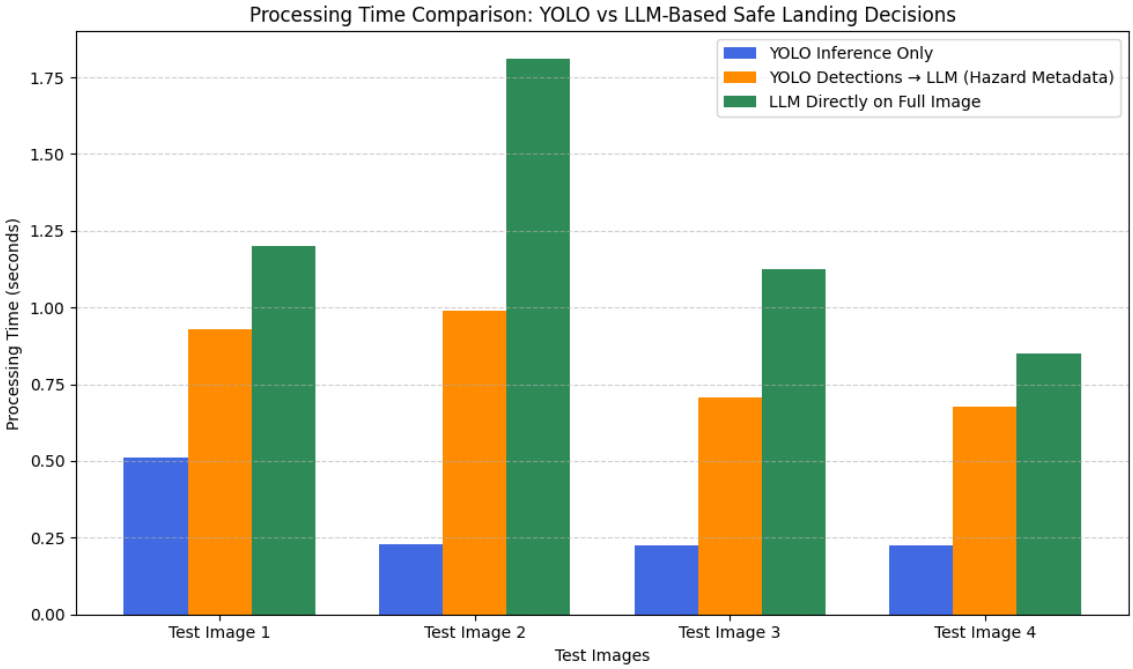


Fig. 14. Processing Time Comparison: YOLO vs LLM-Based Safe Landing Decisions across 4 Test Images

9.1 Key Insights from Experiments

- **YOLO-only is fastest:**
  - YOLO performs object detection in a single pass (O(1) complexity per image).
  - Since it does not perform any additional reasoning, it provides the fastest inference.
  - Limitation: Fails when all detected rooftops are deemed unsafe.
- **YOLO + LLM (Hazard Metadata) balances speed and reasoning:**
  - Metadata from YOLO (e.g., bounding boxes and class labels) is passed to the LLM.
  - LLM performs spatial reasoning only on known hazard zones, avoiding full-image analysis.
  - This drastically reduces response time (50–60% faster than full scan) and cost.
  - It improves accuracy in edge cases (e.g., partially occluded rooftops).
- **LLM-only (Full Image Scan) is slowest:**
  - The LLM receives no structured input; it must analyze the full image path contextually.
  - Processing time increases significantly due to higher token count and image context parsing.
  - Still useful as a last resort when no bounding boxes are available from YOLO.

9.2 Lessons Learned

- Passing structured detection metadata to the LLM reduces cost and improves speed.
- YOLO is effective when clear rooftops exist, but fails under occlusion or hazard overlap.
- Prompt engineering is crucial to guide the LLM effectively using bounding box context.

- A hybrid approach ensures maximum coverage: fast detection with fallback reasoning.

9.3 Sources of Errors and Failure Cases

- **YOLO misclassification:**
  - Rooftops with heavy debris or overlapping fire were still marked safe in a few cases.
  - This led to false-positive landing zones.

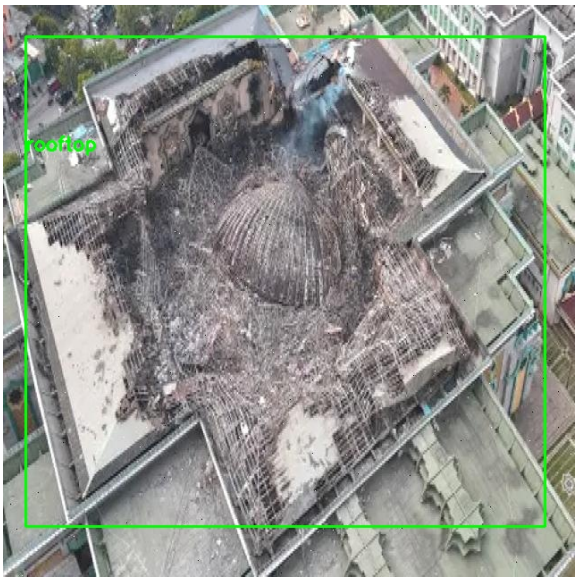


Fig. 15. False Positive due to limited dataset of debris

- **Imbalanced training datasets:**
  - Class imbalance (e.g., more clean rooftops than damaged ones) affected detection bias.
  - YOLO sometimes failed to detect less frequent classes like solar panels or collapsed roofs.
- **Incorrect LLM outputs in full-image mode: Fig 16**
  - When given no metadata, LLM occasionally returned coordinates inside hazard zones.
  - This was due to lack of visual structure and relying only on ambiguous image path descriptions.
- **Latency due to large prompt context:**
  - Full image prompts with no bounding box filtering caused GPT-4 Turbo to take longer to respond.
  - This adds delay without guarantee of better output.

10 Conclusion

- Trained a YOLO-based model to detect rooftops, ships, and hazards (fire, smoke, debris, solar panels).
- Curated and merged multiple datasets to prevent label imbalance and overfitting.
- Designed a rule-based post-processing algorithm (Algorithm 1) to verify rooftop safety using bounding box overlap checks.
- Integrated YOLO with a Large Language Model (LLM) for decision-making when no safe rooftop is found.

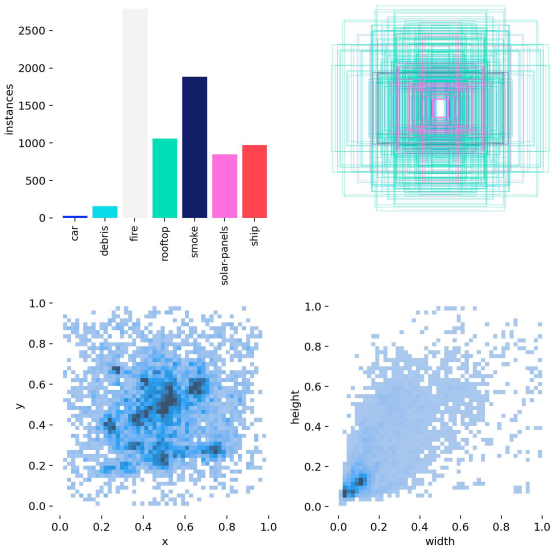


Fig. 16. Labels Distribution

Fig. 17. LLM-only: Incorrect landing zone selected over solar panels.

Fig. 18. YOLO+LLM: Correct safe zone selected using metadata.

- Used prompt engineering to pass object metadata to LLM, reducing full-image reasoning overhead.
- Evaluated system on multiple aerial images and compared three approaches: YOLO-only, YOLO+LLM (metadata), and LLM-only.
- YOLO+LLM showed optimal balance between speed and reliability for identifying safe drone landing spots.