REACH - Retrieval Engine for Articles and Common Household items

Naveen Chandra Paladugu
University of Florida
npaladugu@ufl.edu

Jugal Ganesh Boddu
University of Florida
j.boddu@ufl.edu

***Abstract*—**

Many times, everyday objects are misplaced within a person's home, causing them a lot of frustration and wasted time during urgent situations. In fact, the objects are located inside the house, but we don't the exact location. The proposed project develops an object detection system that can record and retrieve the last location of an item. In this regard, the proposed system saves time and minimizes the effort a user needs to spend searching for their misplaced belongings by providing them with an easy-to-use interface for tracking.

This project detects objects using a YOLOv8 model, trained on a custom-made dataset whose images were created using a tool that extracts frames from video. The dataset was annotated using Roboflow to make it more diverse and robust for most scenarios that could happen in a household. Fine-tuning of YOLOv8 for object detection and tracking in real time is done from CCTV footage captured in a home environment. OpenCV techniques enhance object recognition under different conditions. A user-friendly interface built with Streamlit allows the user to quickly retrieve the last known location of items. TensorFlow Lite is used to optimize model inference for edge devices, ensuring fast performance.

***Keywords*—Object Detection, YOLO, Real-time Tracking, Edge Detection, Streamlit, Roboflow.**

I. INTRODUCTION

The problem here is that misplacing things around the house is one of those frustrating moments that always seem to waste time, stress people, and make them less productive, especially if they are urgently needed. This is an important issue because it involves everyday efficiency and well-being, causing unnecessary disruptions and inefficiencies in personal and professional life.

Traditional methods of searching for misplaced objects are time-consuming and ineffective. The key focus of the project is to provide an intelligent object detection system that enables users to locate misplaced items in their homes in the least amount of time. In this regard, the proposed system uses YOLOv8 to track the objects in real-time to identify household objects and save the last location where they were detected.

Key contributions include Custom YOLOv8 Model for detecting household items, which is trained with a custom dataset and applied augmentation for increasing robustness, Edge Detection Techniques use OpenCV and are utilized to enhance object recognition in poor conditions, Streamlit Interface helps in Designing a user-friendly interface for real-time interaction, which helped users get object locations efficiently and TensorFlow Lite for model inference optimization for low-latency performance on edge devices.

This report is structured as follows: it begins with an introduction to the problem and objectives, followed by a review of related work, detailed system design and implementation, an assessment of trustworthiness and risk management, evaluation and results, a discussion of strengths and limitations, future work and improvements, and concludes with key findings and references.

II. RELATED WORK

A. "YOLOv3: An Incremental Improvement"

1. Paper : J. Redmon, S. Divvala, R. B. Girshick, and A. Farhadi, "YOLOv3: An Incremental Improvement," arXiv:1804.02767 [cs.CV], 2018.
2. Authors: Joseph Redmon, Santosh Divvala, Ross B. Girshick, and Ali Farhadi

This paper introduces YOLOv3, which was able to revolutionize real-time object detection by offering a balance between speed and accuracy. It is known for its fast processing of images and the ability to detect objects in real time. However, YOLOv3 struggles with detecting smaller objects in cluttered environments.

Our project works on the improvements provided by YOLOv8 over YOLOv3. YOLOv8 outperforms YOLOv3 in handling small objects better, accuracy on complex scenes, and has been optimized for both speed and precision. Advanced architecture and training techniques make YOLOv8 more capable for practical applications like home object detection, where high accuracy of detecting occluded small objects is imperative.

B. "EfficientDet: Scalable and Efficient Object Detection"

1. Paper: M. Tan, R. Pang, and Q. V. Le, "EfficientDet: Scalable and Efficient Object Detection," in Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR), 2020, pp. 10781-10790.
2. Authors: Mingxing Tan, Ruoming Pang, and Quoc V. Le

EfficientDet aims at a good balance between model accuracy and computational efficiency by compound scaling: optimizing the model for different resource

constraints. It is very powerful for deployments on resource-constrained devices; however, it may not be the best suitable fit for applications that will require high precision real-time object detection.

In contrast, YOLOv8 focuses on real-time, high-speed object detection without sacrificing any accuracy. In this regard, your system utilizes YOLOv8 combined with Roboflow for augmentation and TensorFlow Lite for efficient inference, making sure that your system can handle both high accuracy and low-latency real-time detection, thus providing a better solution in smart home environments compared to EfficientDet.

C. *"TensorFlow Lite: Deploy Machine Learning Models on Mobile and IoT Devices"*

1. Paper: TensorFlow Team, "TensorFlow Lite: Deploy Machine Learning Models on Mobile and IoT Devices," 2020.
2. Authors: TensorFlow Team

This paper focuses on the deployment of machine learning models on mobile and IoT devices using TensorFlow Lite, optimized for low-latency inference on edge devices. It highlights the need for lightweight models to run efficiently on mobile devices.

Our project also uses TensorFlow Lite for real-time, on-device inference. However, in addition to edge optimization, you also use Docker for deployment, ensuring scalability and remote processing capabilities. This hybrid approach allows your system to scale while offering edge-optimized inference-a flexibility not discussed in the TensorFlow Lite paper.

D. *"Roboflow: Streamlining Computer Vision Dataset Preparation"*

1. Paper: "Roboflow: Streamlining Computer Vision Dataset Preparation"
2. Authors: Brad Dwyer and Adrian Rosebrock

Roboflow is a tool meant to make the preparation, augmentation, and management of computer vision datasets easier. It automates common augmentation techniques, such as rotation, flipping, and scaling, making the development of high-quality datasets for deep learning model training easier.

Our project uses Roboflow for targeted data augmentation, enabling you to improve your custom dataset for household objects and thereby increase robustness against different environmental conditions and occlusions of objects. While Roboflow is applied in many projects in computer vision, your tailored use for household-specific data stands out because it directly addresses real-world object detection needs within a smart home context.

E. *"Fast R-CNN: Real-Time Object Detection with Region-based CNNs"*

1. Paper: R. B. Girshick, "Fast R-CNN: Real-Time Object Detection with Region-based CNNs," in Proc. of the IEEE International Conference on Computer Vision (ICCV), 2015, pp. 1440-1448.
2. Authors: Ross B. Girshick

Fast R-CNN introduced a region-based approach for object detection that improved the detection speed by processing the regions of interest. At that time, this was a big improvement, but because of its complexity and slower inference time, it is less useful for real-time applications.

The opposite is that our project uses YOLOv8, which has optimized performance for both speed and accuracy, making it more appropriate for real-time object detection tasks. Given the architecture, YOLOv8 can run faster than Fast R-CNN with higher precision; thus, it can serve better in deployment within time-sensitive smart home environments.

III. System Design and Implementation

Here we are going to discuss regarding system design and architecture and few of the lifecycle stages such as Data collection and preprocessing, Model development and evaluation and Deployment Strategy used and the HCI considerations.

1. *System Overview*

Let's understand the overall computing architecture of this problem

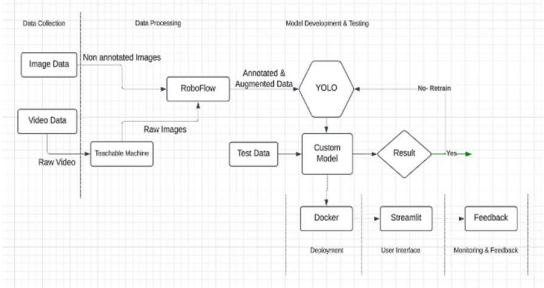

Fig 1: System Architecture

a. Data Collection:

Here we used 2 types of data. Firstly, Image data is found in Kaggle that contains of various objects in household and is named as "Home Office Dataset for Domain Adaptation" Dataset. Secondly, the video data is nothing but a video of a object in different angles.

b. Data Preprocessing:

Firstly, the video data that we get is converted into image data using the tool named Teachable Machine and then the Image data is combined with the dataset downloaded from Kaggle and given to the ROBOFLOW.

Here in RoboFlow we label the images, and the data augmentation is performed in order increase the data size. Augmentation includes adding noise, resizing, exploring light, cutting the images and few others. This

processed annotated data is sent to YOLO model for custom training.

c. Model Development:

YOLOv8 was selected due to its state-of-the-art performance in object detection, offering real-time processing, and efficient architecture suitable for deployment on edge devices. Its ability to handle multiple objects in a single frame and robust detection under varying conditions made it an ideal choice for home environments. This efficiency was further enhanced with the use of TensorFlow Lite for low-latency applications.

Because of this custom data training YOLO works better than any other model to detect the objects in our home environment. In future, we will implement the model in a way that it learns the new data by itself and predicts much more accurately.

d. Deployment:

The trained model is then deployed in a Docker container that encapsulates the entire application, including the model itself, all dependencies, and the runtime environment. Docker will ensure that the system is easy to scale and deploy across different machines or environments without compatibility issues.

The Docker container provides full portability and consistency so that the system can easily be deployed on a user's machine locally, as well as in cloud environments if that is desired.

e. User Interface:

A Streamlit interface provides front-end interaction with the system. It provides a simple interface for users to query and fetch the last known location of any detected object. The architecture also includes a local storage solution for saving object locations, ensuring that historical data can be accessed for future reference.

The inference engine runs inside the Docker container, where TensorFlow Lite is used for optimized and low-latency inference on edge devices. In this way, real-time object detection with minimal latency in processing offers fast feedback to the user. This system can support several requests concurrently, thus making it suitable for dynamic environments such as homes that have different objects in various locations.

f. Monitoring & Feedback:

The monitoring phase involves real-time tracking of the system's performance, focusing on key metrics such as MAP score, IOU and F1-score. These performance measures are important in assessing how well the model detects and classifies objects in dynamic home environments.

User feedback is integrated directly into the interface of Streamlit, where the users can provide input on the accuracy of detection and usability of the system. This is achieved by an easy-to-use interface through which the information is stored in a text file for later analysis. This feedback helps identify potential issues such as missed detections or false positives and will be helpful in continuous model improvements.

This allows for the ongoing refinement of the model. The feedback gathered through the interface, combined with performance metrics, guides model retraining and system updates, ensuring the system evolves to meet user expectations and performs optimally in real-world usage.

g. Data Storage:

The system can store the results of detection and object locations either locally on the device or in any cloud storage solution, depending on the deployment needs. This makes it easy to retrieve historical detection data to track objects over time.

Here at present, we are using it in the local system. So, there is more data privacy and easy access without use of internet.

2. Let us discuss the 3 life cycle stages that plays the entire role of the project in a detailed manner.

   a. Data Collection and Preprocessing:

   For Reach to work properly a proper dataset is quite crucial which resembles the real-life images of the desired objects. Through this, the model will be able to recognize the objects around the house.

   To achieve this, we found a dataset in Kaggle named “Home Office Dataset for Domain Adaptation”. This contains different folders as objects and each folder contains multiple images of that object category. In addition to this, we have also used Teachable Machine to create more generic data where we have taken images of our daily items used like mobile, watch, earbuds, etc.

   These images are then uploaded to Roboflow for data

annotation (Drawing a bounding box around the object in an image). Through roboflow we have annotated almost 2000 images. These annotated images are then sent to a pre-processing pipeline where we first re-sized every image to the shape 640 x 640. Then to generalize the data, data augmentation has been performed that includes flipping, shearing, rotation, and blurring. OpenCV techniques were employed for edge detection and noise reduction, enhancing object boundaries for better detection.

Even after doing all these, we had some problems in terms of diversity of data. Different lighting and background conditions were a significant challenge. A constant effort was required to identify and distinguish the overlapping objects and small objects. The custom training dataset was relatively small, limiting the model's ability to generalize effectively. This necessitated extensive data augmentation to simulate diverse conditions. Additionally, the small dataset size made it challenging to achieve high accuracy in detecting objects under varying lighting, occlusions, and cluttered environments, impacting initial performance.

b. Model Development and Evaluation

For this project, the main goal is to detect objects. There were 2 options i.e., YOLO and Resnet. We found out Yolo suits better for our objective as it defines a bounding box around the detected object. Out of all the yolo models we chose Yolov8l for its state-of-the-art performance in object detection, offering high accuracy, real-time processing, and efficient architecture suitable for deployment of edge devices. We have used pre-trained weights of Yolo with our custom dataset and transfer learning methods for testing. Its ability to handle multiple objects in a single frame and robust detection under varying conditions made it an ideal choice for home environments.

The training was done in 10 epochs with a batch size of 16. As it is an image dataset access to GPU is mandatory as it significantly reduces computational time. This model has been evaluated using metrics like mAP(mean average precision) though this we can calculate the accuracy of object detection models by calculating the mean average precision across different classes and IoU thresholds.

IoU(Intersection over Union) measures the overlap between the bounding box and ground truth(annotated image). Through this, we can say how well the model is predicting the bounding box. We are achieving almost mAP scores of 0.6.

We have generated 2 versions of REACH where in one the user needs to upload a video, and the output will be a video with all the objects detected in it and in the other version our model accesses the webcam and detects the object through the camera. If the desired object is detected the model returns the snapshot of the object, this acts as the location of the detected object and stores it in the local folder.

c. Deployment Strategy

We have implemented docker containerization and the model runs through the input video by detecting the objects. Output video contains all the objects detected in them. Docker ensures that the application is portable and can be run on any system with the Docker engine installed.

But for the version where we have used webcam to detect the objects runs locally only. The docker can't access the local hardware which is required to detect the webcam. This raises the problem, and we are unable to open the webcam for REACH to work. We are hosting this version through streamlit and this opens a webcam through the objective of the reach is achieved.

We have a feedback mechanism which logs the feedback of the user in a certain format, and this will be used to go back and rectify the problem whenever required. The last detected image is also being stored in the local file for the user to check where their objects are placed.

3. HCI Considerations:

Let us discuss the user interaction design and feedback mechanism in a detailed way separately

1. User Interaction Design:

The system's Streamlit-based interface offers an intuitive and user-friendly

experience, allowing users to search for objects and view their last detected location.

The user will interact through the UI designed using Streamlit for this project, in a way that is simple, accessible, and effective. Using the UI will enable users to interact with the object detection system by finding the objects, showing where in the environment of a house they were last detected.

The design employs a clean layout with minimal distractions, ensuring ease of navigation. Users can view real-time object detection results and explore details such as timestamps and object categories.

We developed 2 models, one takes video as a input and detects the required selected object and the other take live webcam footage and detect the result.

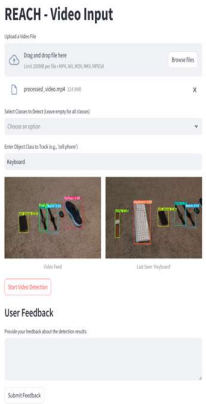

Fig-3 UI of Video taking model

Here we have given features like detecting only specific objects in the video given and another feature is to detect last seen on specific object type. This helps users to customize their preference in search.

The UI will be responsive and immediate to changes, providing timely and effective aid in times of critical junctures. This is all due to the user-centric approach driving adoption, trust, and satisfaction in using the system.

2. Feedback mechanisms:

The system contains a very strong feedback mechanism for improving performance continuously. It will allow users to give direct feedback through the Streamlit interface on detection accuracy, object misidentification, or any issues with the system.

We have a feedback mechanism that logs the user's feedback in a certain format, and this will be used to go back and rectify the problem whenever required. The last detected image is also being stored in the local file for the user to check where their objects are placed.

Here is the image of the UI that asks for feedback and that is stored in local disk concerning privacy of the user and will be collected by us in updating time.

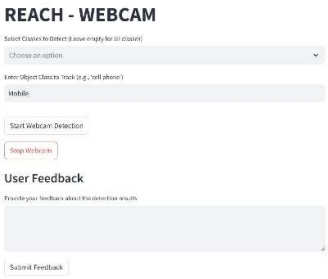

Fig-4 Feedback UI

This feedback collection, analysis, and integration continue in a loop, enabling the system to learn over time for the betterment of user needs. The emphasis on usability and transparency in the feedback mechanism helps it align with ethical AI practices and increase user satisfaction.

IV. TRUSTWORTHINESS AND RISK MANAGEMENT

Firstly, let us discuss different strategies used in each stage of lifecycle to manage security, privacy and ethical compliance

1. Data Collection:
    a. Security: All training data, including images and video frames, were treated as confidential. Any sensitive information is encrypted in transmission and storage to avoid unauthorized access.
    b. Privacy: Data used for training and testing is restricted to non-personally identifiable images of household objects only, so they meet privacy standards.
    c. Ethical Compliance: The data collection methodology was foreseen in a very open way, with the consent of the user when necessary. The treatment of data followed ethical directives to the effect that data were collected fairly and without intrusion into the private sphere.
2. Data Preprocessing:

a. Security: Data preprocessing, such as converting video frames into image data and augmentation, was done in a secure manner in Roboflow which is secure and only be accessed to correct members.
b. Privacy: The preprocessed pipeline did not use personally identifiable information. For enhancing the diversity of the dataset, techniques of data augmentation have been used that do not infringe on anonymity or invasiveness of the data.
c. Ethical Compliance: Preprocessing techniques included data augmentation that was designed to simulate real-world conditions without distorting or changing the original data in any way that would misrepresent the objects or violate ethical guidelines.

3. Model Development:
   a. Security: The training process and model code were stored in secure, version-controlled repositories
   b. with restricted access.
   c. Privacy: The model was trained using anonymized datasets that had no sensitive or personal information. Even if there is sensitive data it is blurred.

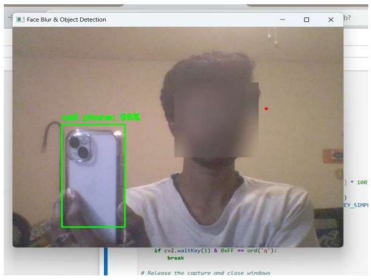

Fig 2: Blurred Image

   d. Ethical Compliance: The training dataset was balanced to ensure fair object detection across the various categories, and measures were taken to ensure that the model performs equitably.

4. Deployment:
   a. Security: The system is deployed in docker containers, isolating it and preventing unauthorized access and mitigating potential security risks. The deployment was kept secure by applying regular security updates to its dependencies.
   b. Privacy: The docker doesn't allow any user to see the data as the mechanism is private from others.
   c. Ethical Compliance: The deployment was ethical; in that it was transparent regarding the use of user data and feedback.

5. Monitoring:
   a. Security: Monitoring systems ensured that any sensitive data involved in the evaluation process was handled with the utmost care.
   b. Privacy: All user interactions with the Streamlit interface were securely managed.
   Feedback given through the interface was stored in text files, and data was processed locally to ensure privacy without the need for external data transmission.
   c. Ethical Compliance: The performance of the model was verified on a regular basis in a differential manner in all different object categories by means of fairness assessments continuously. By this, the system functioned properly within ethical boundaries by avoiding unwanted biases.

Let us discuss the various residual risks and the mitigation strategies to overcome

1. Data Collection Stage:
   a. Residual Risk: Insufficient diversity in the dataset could lead to biased or inaccurate detection for certain object types.
   b. Strategy: Various techniques of data augmentation (such as scaling, rotation, and flipping) were implemented and more images taken fr om different sources were added in the dataset to increase diversity.
2. Data Preprocessing Stage:
   a. Residual Risk: Preprocessing errors, such
   as mislabeled or poorly augmented data, could also lead to degraded model performance.
   b. Strategy: Automated preprocessing pipelines were implemented with validation steps to ensure data integrity. Random checks were done to identify and correct errors in augmented datasets.
3. Model Development Stage:
   a. Residual Risk: Overfitting on the training dataset
   could reduce the potential for generalization on unseen data.
   b. Strategy: Regularization techniques like dropout and early stopping were applied during the model training. Cross-validation was carried out to assess model

performance on unseen data, which could ensure robustness.

4. Deployment Stage:
    a. Residual Risk: Potential security vulnerabilities in the deployment environment (Docker containers) could lead to unauthorized access.
    b. Strategy: Potential security vulnerabilities in the deployment environment (Docker containers) could lead to unauthorized access.
5. Feedback Stage:
    a. Residual Risk: User feedback may not capture edge cases or rare object detection issues, hence leaving gaps in the improvement of the system.
    b. Strategy: Continuous monitoring was established whereby periodic updates were affected through feedback to improve the system's detection capabilities. Edge cases were flagged for additional testing and model retraining.
6. Monitoring Stage:
    a. Residual Risk: Gradual degradation of the model over time-for example, due to changes in the appearance of objects or lighting conditions-may degrade performance.
    b. Strategy: Scheduled re-evaluation and retraining of the model with updated datasets were incorporated. User feedback and performance metrics such as accuracy and F1-score were monitored to identify when retraining was necessary.

V. EVALUATION AND RESULTS

Here we will discuss various testing cases and their results regarding various testing cases.

1. Performance Metrics

The main concept of this project is Object Detection and for this the common metrics like accuracy can't be used directly. Instead, we use metrics like mAP. A score of 0.6 signifies that, at this IoU threshold, the model has reasonable localization and classification accuracy. Received a precision score of 0.7 which indicates that there might be some false positives.

We have used different types of loss functions, and it has been reduced with the increasing number of epochs

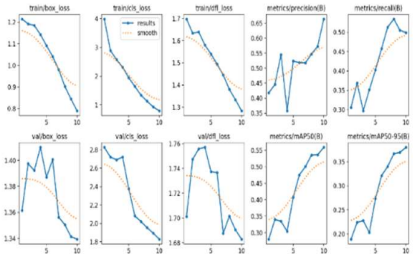

Fig-5 Various Loss Functions

Here we can clearly see that the loss is gradually decreasing over time.

2. Monitoring and Feedback

Monitoring the performance and resource usage of the system is crucial. We need to examine how the system behaves as Monitoring the performance and resource utilization of the system is an absolute necessity for keeping the system's efficiency and reliability. This section explains the analysis and results obtained from the captured monitoring metrics during the operation of the system, focusing on memory usage, disk usage, and CPU utilization.

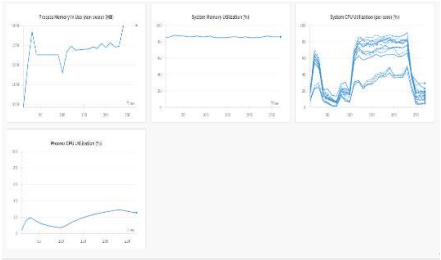

Fig-6 Monitoring of various metrics

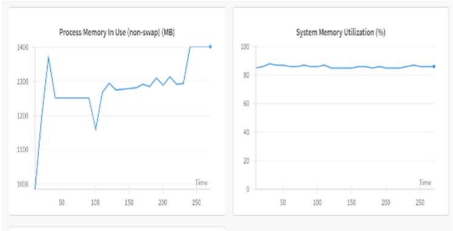

Fig-7 Memory Utilization

The process memory usage remained stable after an initial spike, averaging around 1400 MB. The sharp increase at the start indicates the memory being allocated during the model loading and initialization processes, and then the model is the most used memory when it runs data. The steady state is a result of effectively used memory during runtime and the fact that no memory leaks have occurred.

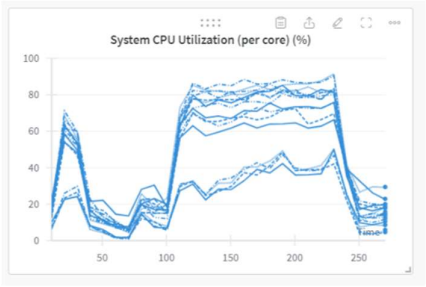

Fig-8 System CPU Utilization

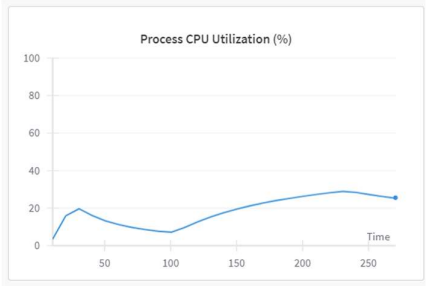

Fig-9 Process CPU Utilization

All cores of CPU usage showed a repetitive pattern with the peaks of utilization occurring during the operations that involve data pre-processing, object detection, and inference tasks. The cores have the same spikes, which signifies that multithreading and parallel processing are effectively used and the CPU utilization for the specific process started at a low level and progressively increased, reaching its maximum at about 30%. This is steady climbing that corresponds with the slow rise of computational load on the system due to the more frames or images it processed.

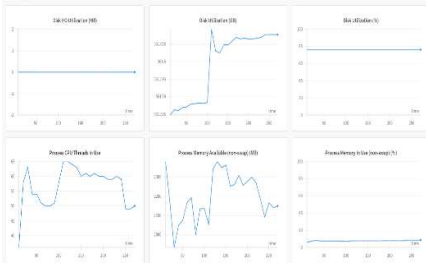

Fig-10 Various memory metrics

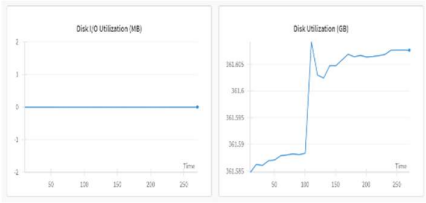

Fig-11 Disk Utilization

According to the graph, there is a significant rise in disk space utilization in the first quarter of runtime followed by stabilization. This could be because the system first uploads the data measurement content and model files, and then it runs the regular read/write procedures for logs or files. Thus, there is no disk I/O activity which means the system's work is mainly computer-bound rather than I/O-bound.

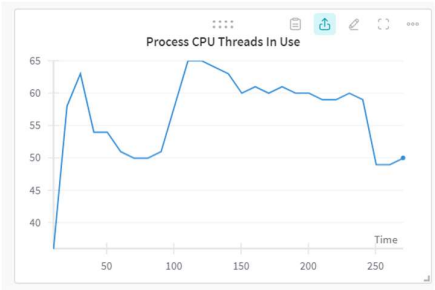

Fig-12 Threads and Process

The CPU thread count showed a constant increase, sometimes peaking at the beginning, which was associated with initialization, and later a steady decrease came about as the workload dropped. The number of units represents how well parallel tasks have been managed during the object detection process.

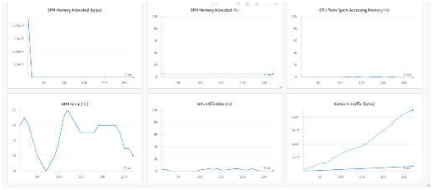

Fig-12 GPU Utilization

The charts are able to monitor GPU performance in real time as well as provide feedback. The overall feedback suggests a low-challenge situation for GPU resources with a few operations that briefly raise the temperature. The regular network traffic shows a process of steady data transfer that is not related to this. These understandings are very important for making the best out of the GPUs and keeping the systems healthy as well.

Through this, we can say that the model runs maximum time on the CPU. This local machine is unable to provide GPU access to the model and all the monitoring has been mentioned above.

3. Real-World Testing

REACH underwent extensive real-world testing to evaluate its performance, usability, and reliability in practical scenarios. The interface is layman-friendly where the user can detect the objects immediately after enabling the camera.

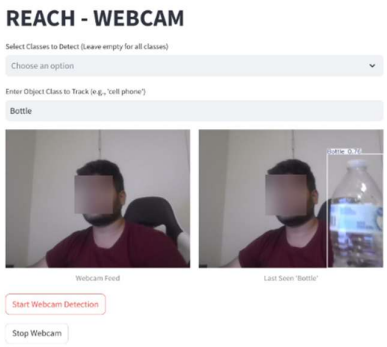

Fig-13 Main Demo

This is the live example where the user is detecting his water bottle immediately after enabling the camera

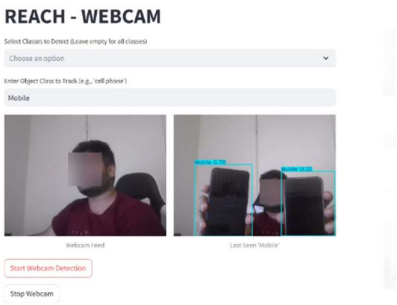

Fig-14 Main Demo

This is proof that our system can detect multiple objects while blurring the face at the same time through this the user is not required to worry about having multiple objects with him. Users can check the last detected object's image to know where he kept them.

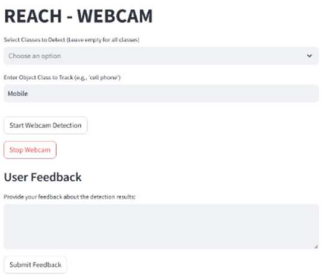

Fig-15 Feedback UI

After detection if the user thinks something wrong is getting detected then user can input his feedback in this and can save the feedbacks locally. As this is sensitive data, we are not uploading it to any cloud platform, rather we take these logs privately to address the issue.

VI. DISCUSSION

Here we talk about the challenges, strengths and limitations and the way we resolve them and highlighting the novelty and broader implications of this work.

1. Discussion between the strengths and limitations
    a. Strengths
        1. Real-Time Object Detection: The system performs well in real-time to give immediate feedback about object locations, which is critical in time-sensitive situations, such as looking for misplaced items.
        2. User-Friendly Interface: The Streamlit interface is intuitive, making it accessible for non-technical users, and provides a simple, seamless experience for querying object locations.
        3. Edge Deployment: Deploying the model on Docker with TensorFlow Lite and optimizes the inference speed, and it will be apt for deployment on low-resource edge devices without loss of performance.
    b. Limitations
        1. Object Occlusion and Cluttered Environments: The model fails to provide quite accurate detection of objects in the case of partial occlusion or when the object is hidden behind other items, which can result in missed detections in cluttered areas.
        2. Small Training Dataset: The dataset used for training was quite small, and this limited the generality of the model over all household objects and environments, hence probably affecting the detection in more complicated scenarios.
        3. Dependence on Lighting Conditions: While the system works well under normal lighting, its performance degrades under low light or overly bright conditions in which object boundaries are poorly defined.
2. Challenges
    a. Small Training Dataset
        1. Resolution: The model initially could not generalize well for different household objects because of the limited dataset. For this, the data augmentation techniques of rotation, flipping, and varying lighting conditions were used to artificially increase the diversity of the dataset. We further used Roboflow for augmenting the dataset to let the model generalize

well over the appearance of different objects and scenes.

b. Object Occlusion and Cluttered Background
    1. Resolution: The objects being partially hidden by other objects gave the biggest challenges of detections, especially in cluttered environmental setups. To counteract this, we added more examples of training for objects with partially occluded conditions. Further, edge detection, together with image preprocessing enhancements on object boundaries, was performed for higher accuracy of object detection in complicated scenarios.

3. Novelty and Broader Implications
    a. The novelty of this project comprises seamless integration of custom-trained YOLOv8 models, Streamlit-based user interaction, and real-time detection of objects, returning the snapshot where that object is located. Unlike traditional systems of object detection, this project focuses mainly on the localization of house objects and solves a common problem that is not much explored: helping people find things they misplace in their home. This hasn't been solved by anyone till date. Thus, making our model unique.
    b. The broader implications include the possibility of scaling this system for broader IoT integrations that could enable smart homes to track and manage household items in a dynamic fashion. This could also potentially inspire further innovation in personalized AI-driven tools, catering to niche yet impactful real-world challenges.

VII. Future Work and improvements

Here we are going to discuss the potential improvements and the areas where you further research your application in and the future challenges of the system.

1. Potential Improvements and Extensions
    a. Enhanced Dataset Diversity: This is expanding the custom dataset further with more objects, scenes, and lighting conditions to make it more generalizable. It may also include other publicly available datasets.
    b. Improved Real-Time Feedback: This can be enhanced further by allowing the user to query or respond with their voice through speech recognition, providing a hands-free and interactive way. This will be helpful in cases where typing may not be convenient, especially for differently abled people.
    c. Adaptive Learning: Introduce mechanisms for online learning in which the system updates new objects or user-specific preferences in runtime, thus offering better personalization.
2. Areas for Further Research and Application
    a. Smart Home Integration: Extend the system to smart home applications, such as Alexa and Google Home, for smooth interaction and control of devices.
    b. Assistive Technology for Accessibility: Develop applications targeting visually impaired or elderly users in which object detection can be combined with audio cues or haptic feedback to support navigation and object retrieval.
3. Future Evolution of the System
    a. Scalability for Larger and Complex Environments: The system can be further enhanced with multi-camera integration and spatial mapping to address challenges posed by multi-room or multi-level homes and provide more accurate object localization over wider areas.
    b. Dynamic Adaptation to New Objects: Introduce methods of continual learning that will effectively allow the system to understand and detect new objects in real time without requiring an overhaul of the training; keep the model updated with user-specific items.

VIII. Conclusion

This project effectively addresses the everyday challenge of locating misplaced household items through a high-performing, AI-powered object detection system. Utilizing YOLOv8, the system delivers real-time, precise detection with minimal latency, significantly reducing the time and effort required to find objects. The integration of an intuitive Streamlit interface ensures accessibility for non-technical users, making the system practical for real-world, everyday use. By focusing on a commonly overlooked yet impactful problem, this project contributes to enhancing household productivity and convenience.

The success of this project lies in its comprehensive and well-structured AI lifecycle management. Data collection and augmentation were optimized through Roboflow, ensuring a diverse dataset representing real-life scenarios. The fine-tuning of YOLOv8, paired with robust evaluation metrics like mAP and IoU, demonstrated superior detection accuracy and reliability. Deployment via Docker infrastructure with TensorFlow Lite ensured scalability and efficiency, enabling seamless real-time performance even in resource-constrained environments. Continuous improvement was supported by monitoring and feedback mechanisms, including real-time logging and user-reported insights.

IX. References

[1] J. Redmon et al., "You Only Look Once: Unified, Real-Time Object Detection," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 779-788.

[2] K. He et al., "Deep Residual Learning for Image Recognition," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770-778.

[3] M. Zehtabian, N. Zamani, and M. Ahmadi, "Privacy-Preserving Deep Learning Using Secure Multiparty Computation," *IEEE Access*, vol. 9, pp. 29748–29758, 2021.

[4] H. Fan, H. Ling, and S. Zhu, "Real-Time Object Detection With Online Learning," *IEEE Transactions on Image Processing*, vol. 25, no. 3, pp. 1274-1289, 2016.

[5] *J. Redmon, S. Divvala, R. B. Girshick, and A. Farhadi, "YOLOv3: An Incremental Improvement," arXiv:1804.02767 [cs.CV], 2018*.

[6] *M. Tan, R. Pang, and Q. V. Le, "EfficientDet: Scalable and Efficient Object Detection," in Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR), 2020, pp. 10781-10790.*

[7] TensorFlow Team, "TensorFlow Lite: Deploy Machine Learning Models on Mobile and IoT Devices," 2020.

[8] "Roboflow: Streamlining Computer Vision Dataset Preparation"

[9] R. B. Girshick, "Fast R-CNN: Real-Time Object Detection with Region-based CNNs," in Proc. of the IEEE International Conference on Computer Vision (ICCV), 2015, pp. 1440-1448.