

IDD Report

Vehicle Tracking in Challenging Scenarios using YOLOv8

Irina Getman

Bachelor of Software Engineering, Yoobee colleges

CS301 Investigative Studio II

Supervisory team:

Main–Dr. Mohammad Norouzifard

Co-supervisor–Aisha Ajmal

April 21, 2023

Table of Content

Introduction	3
Pipeline diagram for Vehicle Tracking Model	5
Development and Deployment tools	6
YOLOv8 Model	8
YOLOv8 architecture	8
Data Collection	9
Data Augmentation	10
Roboflow	11
Features of Roboflow	12
Software Requirements Specification	12
Functional and non-functional requirements	13
Functional Requirements	13
Nonfunctional Requirements	13
Use case scenarios	13
General Use Case scenario for "Vehicle Tracking system using YOLOv8 algorithm"	13
Harsh weather conditions	14
Other challenging situations	18
UML diagrams	24
Class diagram	25
Activity diagram	28
UX/UI Modeling	30
User flow diagram	30
Lo-Fi Sketches	31
User testing 1	32
User testing 2	36
Design System	39
Hi-Fi prototypes	41
User testing 3	42
Conclusion	45
References	47
Appendix	49
Data Collection	49
YOLOv8 backbone diagram	50
Pretrained model prediction	52

Introduction

The advancements in computer vision and deep learning techniques have revolutionised the way we track objects in real-time scenarios. One such scenario is vehicle tracking in challenging environments where the accuracy of tracking is highly critical. This is where the YOLO (You Only Look Once) object detection algorithm comes into play. It is a state-of-the-art deep learning model that has shown remarkable performance in detecting and tracking objects in real-time.¹

The report starts by providing a pipeline diagram that serves as an overview of the key steps and components of our vehicle tracking model using the YOLOv8 algorithm. It provides a visual representation of the flow of data and processing from input to output in the system. The purpose of including a pipeline diagram in the report is to provide a clear and concise overview of the model's architecture and how it operates. The diagram helps to convey the complexity and functionality of the model in an easily understandable manner, providing a visual aid that complements the textual explanations in the report.

Then it continues with an overview of the YOLOv8 object detection algorithm and its architecture. It then discusses the dataset that will be used for training and testing the models and the preprocessing techniques used to prepare the data.

The report also includes a list of development and deployment tools, languages, and technologies used in the project. It further covers software requirements, functional and non-functional requirements, and use case scenarios with diagrams. A class diagram and activity diagram are also provided.

In the UX/UI modelling section of the report, we will be presenting a comprehensive overview of the user experience (UX) and user interface (UI) design for our vehicle tracking deployment model using Flask. Flask is a micro web framework written in Python that allows for the rapid development of web applications. It provides a simple and lightweight way to build web applications, making it an ideal tool for our project.

Based on the feedback from usability testing, we will then describe the alteration process, which includes any changes or refinements made to the initial design based on the feedback received. This process ensures that the final design meets the needs and expectations of the end-users.

Overall, this report presents a comprehensive study of the utilisation of YOLOv8 for vehicle tracking in challenging scenarios. Additionally, the report highlights the importance of the UX/UI modelling section, which includes low-fidelity wireframes, usability testing results, alteration process, and high-fidelity prototypes, as these elements play a crucial role in ensuring a user-friendly and

¹ Due to the length of the document, single spacing has been used instead of the standard double-spacing requirement as per APA 7th edition guidelines.

efficient design. The Software Requirements Specification (SRS) section also serves as a crucial reference, providing detailed specifications and guidelines for the development and deployment of the vehicle tracking model. Lastly, the report outlines future work and identifies constraints faced during the project, while also acknowledging the achievements and contributions made in advancing the field of vehicle tracking in challenging scenarios.

Pipeline diagram for Vehicle Tracking Model

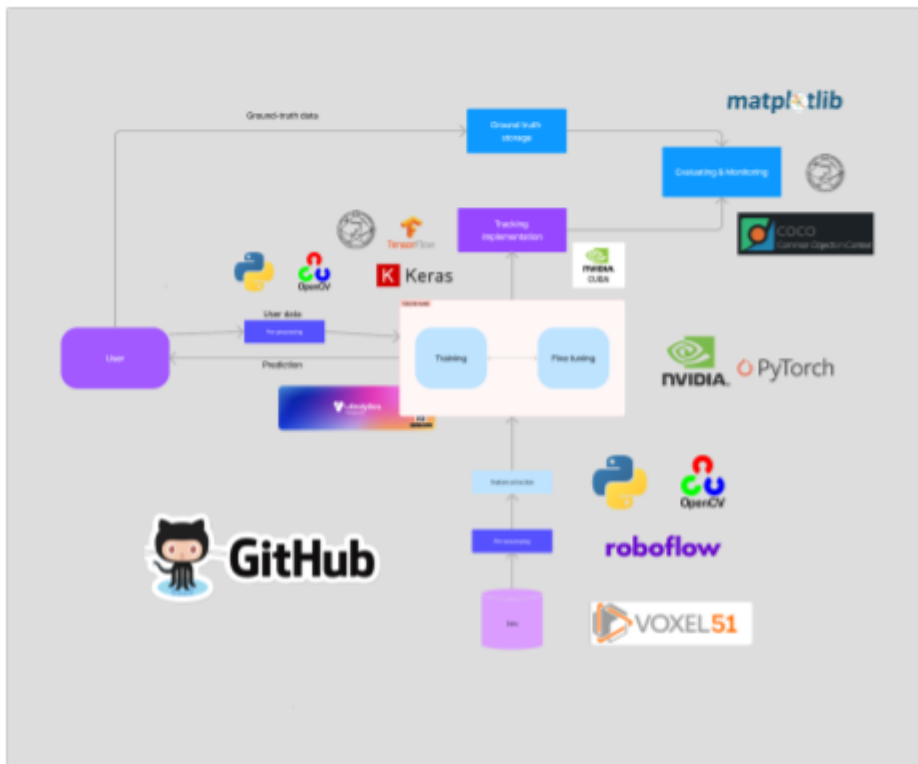


Figure 1. Pipeline diagram for Vehicle Tracking model using YOLOv8 algorithm

2

A pipeline diagram of "Vehicle Tracking in challenging scenarios using YOLOv8" deep learning based project(see Figure 1), is a visual representation of the different stages or steps involved in our project's workflow.

Here's are the steps of a pipeline diagram for our project:

1. **Data Collection:** This stage involves gathering the necessary data for vehicle tracking tasks. This task also includes video or image datasets that contain challenging scenarios such as adverse weather conditions, occlusions, or varying lighting conditions. We may also need to collect ground truth annotations or labels for the vehicles in the dataset to use for training and evaluation.

Tools/frameworks: OpenCV, FiftyOne, Roboflow.

2. **Data Preprocessing:** In this stage, the collected data is processed to prepare it for training. This may involve tasks such as resizing images or videos, converting formats, augmenting data to increase its diversity, and splitting the data into training, validation, and test sets.

Tools/frameworks: OpenCV, Python, Keras, TensorFlow.

3. **Model Training:** This stage involves training the YOLOv8 object detection model using the preprocessed data. This stage may include tasks such as

² By clicking on the image, you can access the link to Figma

configuring the YOLOv8 model architecture, setting hyperparameters, and training the model using GPU-accelerated hardware.

Tools/frameworks: Darknet (YOLOv8), CUDA, cuDNN, GPUs (such as NVIDIA GeForce or Tesla series), Deep Learning frameworks such as TensorFlow, PyTorch, or Keras.

4. Model Evaluation: Once the YOLOv8 model is trained, it needs to be evaluated to measure its performance. This may involve tasks such as testing the model on the validation and test datasets, calculating various performance metrics such as precision, recall, F1-score, or mean average precision (mAP), and analysing the results to assess the model's accuracy and robustness in challenging scenarios.

Tools/frameworks: Darknet (YOLOv8), Python, evaluation libraries such as COCO API, Matplotlib.

5. Tracking Implementation: This may involve tasks such as applying the object detection model on video frames or images in real-time, extracting the bounding box coordinates of the detected vehicles, associating the bounding boxes across frames to track the vehicles, and visualising the tracked vehicles on the output frames or images.

Tools/frameworks: Darknet (YOLOv8), OpenCV, Python, tracking libraries such as SORT (Simple Online and Real-time Tracking) or DeepSORT.

6. Visualization and Output: In this stage, we will generate visualisations or outputs to visualise the tracked vehicles in challenging scenarios. This may involve tasks such as drawing bounding boxes or other visual cues around the tracked vehicles, and saving the tracked vehicle information or visualisations to a file or displaying them in a graphical user interface (GUI).

Tools/frameworks: OpenCV, Matplotlib.

7. Post-processing: In this stage, the tracked vehicle information can be further processed or analysed. This may involve tasks such as counting tracked vehicles, performing some analytics on the tracked vehicle data.

Tools/frameworks: Python, NumPy, Pandas.

8. Deployment: Once the vehicle tracking pipeline is developed and tested, it can be integrated into our web application.

Tools/frameworks: Flask.

Development and Deployment tools

List of Collaborative Tools, IDEs, Programming Languages, Technologies, and Frameworks Used for the Project:

Collaborative Tools:

- GitHub: Used for version control, code sharing.
- Trello: Used for project management, task tracking.

- Google Docs: Used for collaborative document editing, sharing project-related information.
- Teams: Used for virtual meetings, discussions.
- Microsoft Word: for creating, editing, and formatting documents with text and graphics.

IDEs (Integrated Development Environments):

- Visual Studio Code: for front-end and back-end development.
- Jupyter Notebook: for data analysis, model training, and experimentation with machine learning models.
- Google Colab: for data analysis, model training. Extra plus - can use cloud based GPU for computer vision tasks.

Programming Languages:

- Python: Used for developing the YOLOv8 model, implementing data preprocessing, and integrating with other technologies.
- HTML/CSS: Used for designing and styling the user interface of the vehicle tracking app.
- SQL: Used for database management and data storage.

Technologies and Frameworks:

- YOLOv8: Object detection model used for vehicle tracking.
- Flask: Python web framework used for back-end development.
- OpenCV: Computer vision library used for image processing and object detection.
- NumPy: a useful tool in computer vision tasks, providing efficient and convenient operations for image processing, feature extraction, and deep learning applications.
- TensorFlow or PyTorch: used for training and deploying object detection models like YOLOv8.
- CUDA and cuDNN: NVIDIA libraries for GPU-accelerated computing, which can greatly speed up the training and inference processes of deep learning models like YOLOv8.
- Pandas or other data manipulation libraries: Used for data preprocessing, analysis, and manipulation tasks.
- Figma: A collaborative design tool for creating UI/UX designs.
- Lucidchart: A web-based diagramming and visual communication tool for creating flowcharts, wireframes, and other visual diagrams.
- Keras: User-friendly deep learning library for building and training neural networks.
- Darknet: Efficient open-source framework for object detection and image recognition.
- Matplotlib: Versatile Python library for creating various types of data visualisations.

YOLOv8 Model

YOLOv8, as the latest iteration of the YOLO (You Only Look Once) object detection algorithm, released by Ultralytics team earlier this year, outperforms previous versions by incorporating innovations has become current State-of-The-Art object detector reclaiming this title from YOLOv7 (Jain, 2023). Several unique features and advancements that make it a powerful option for vehicle tracking in challenging scenarios.

- **Improved Accuracy:** according to research (Photon, 2023), YOLOv8 incorporates advancements in backbone architecture and feature pyramid networks, which can enhance the accuracy of object detection and tracking compared to previous versions of YOLO.
- **Faster Inference Speed:** according to a research YOLOv8 Detection 10x Faster with DeepSparse—over 500 FPS on a CPU - Neural Magic (2023), YOLOv8 utilises optimised anchor box priors and prediction heads, leading to faster inference speeds without sacrificing accuracy, making it suitable for real-time applications.
- **Enhanced Robustness:** YOLOv8 is designed to handle challenging scenarios such as low light, occlusion, and cluttered environments, making it more robust in tracking vehicles in difficult conditions compared to previous YOLO versions. (Dorfer, 2023)
- **Improved Scalability:** YOLOv8 is scalable to different input resolutions, making it adaptable to various hardware capabilities and deployment scenarios. (OpenMMLab, 2023b)
- **Flexibility in Model Architecture:** according to Configuration - Ultralytics YOLOv8 Docs (2023), YOLOv8 allows for customization of the model architecture and hyperparameters, making it flexible for different use cases and performance requirements.

YOLOv8 architecture

YOLOv8 is a state-of-the-art object detection model that is based on the YOLO (You Only Look Once) family of models. YOLOv8 is an improved version of YOLOv5, which is itself an improvement of the original YOLO model. YOLOv8 is built on top of the deep learning framework PyTorch and has the following architecture:

1. **Backbone network:** YOLOv8 uses a modified CSPDarknet53 as the backbone network. CSPDarknet53 is a convolutional neural network that consists of a series of convolutional layers with residual connections and cross-stage partial connections. This network is designed to extract features from the input image.

2. Neck network: YOLOv8 uses a modified YOLOv5 neck network that consists of a series of convolutional layers and upsampling layers. The neck network is used to combine the features extracted by the backbone network and to generate feature maps of different resolutions.
3. Head network: YOLOv8 uses a modified YOLOv5 head network that consists of several convolutional layers and prediction layers. The head network is used to predict the bounding boxes and class probabilities of the objects in the input image.
4. SPP (Spatial Pyramid Pooling): YOLOv8 uses APP to process the feature maps at different resolutions. SPP divides the feature maps into multiple grids and applies pooling operations to each grid. This allows the model to capture object features at different scales.
5. PAN (Path Aggregation Network): YOLOv8 uses PAN to combine the features from different scales. PAN consists of several convolutional layers and upsampling layers. It is used to merge the features from the SPP layers and to generate a unified feature map.
6. Detector layer: The detect layer is the final layer of the YOLOv8 model. It is used to predict the bounding boxes and class probabilities of the objects in the input image. The detect layer uses anchor boxes to predict the size and aspect ratio of the objects.

To summarise, YOLOv8 is designed with a strong focus on speed, size, and accuracy, making it a compelling choice for various vision AI tasks. It outperforms previous versions by incorporating innovations like a new backbone network, a new anchor-free split head, and new loss functions. These improvements enable YOLOv8 to deliver superior results while maintaining a compact size and exceptional speed.

YOLOv8 is capable of performing tasks such as detection, segmentation, and classification. The models available by YOLOv8 are YOLOv8n, YOLOv8s, YOLOv8m, YOLOv8l, and YOLOv8x.

Installing and using YOLOv8 is straightforward. According to the documentation, there are two ways to install and use YOLOv8. One can use the pip command to install Ultralytics or clone the official repository and then install it from there. Thus, making deployment of the YOLOv8 model for various tasks extremely user friendly.

Data Collection

Collecting and preparing data for challenging scenarios such as low visibility and occlusion can be a complex process. Here are some steps and tools we are going to use to collect and prepare data for such scenarios:

1. Identify the specific scenarios for the model: Since our model is designed to track vehicles, we will need to collect data to train our model to recognise vehicles. We are planning to start with two classes: "car" and "truck". Then we need to obtain data for challenging scenarios such as fog, rain, or snow, or situations where objects are partially occluded by other objects.
2. Collect data: begin collecting data that represents these scenarios. This could involve taking photos or videos in low visibility conditions or setting up controlled experiments where objects are partially occluded.
3. Use data augmentation techniques: Data augmentation techniques can be used to artificially increase the size of your dataset and make it more representative of the challenging scenarios. We are planning to use image processing techniques to add fog or rain to the images, or use 3D modelling software to generate synthetic images of objects with varying levels of occlusion(Blance, 2022).
4. Label the data accurately: After collecting and augmenting our data, it is important to label it accurately and consistently. This involves assigning ground truth labels to each image or video frame that indicate the presence and location of objects of interest. There are many tools available for labelling data, including open-source tools such as labelling and commercial platforms such as Amazon SageMaker Ground Truth and Roboflow.
5. Split the data into training, validation, and test sets: Finally, it is important to split the data into training, validation, and test sets. The training set is used to train our model, the validation set is used to monitor its performance during training, and the test set is used to evaluate its performance after training is complete.

In addition to data augmentation techniques, there are several public datasets available that contain images or videos taken in challenging scenarios such as low visibility or with objects partially occluded. For example, the OVIS dataset is a large-scale benchmark dataset for video instance segmentation with severe object occlusions. (OVIS | Occluded Video Instance Segmentation, 2022)(Qi et al., 2022)

Data Augmentation

Data augmentation is a technique used to artificially increase the size and diversity of a dataset by applying various transformations to the original data. Some popular data augmentation techniques for image data include:

- Flipping : Flipping an image horizontally or vertically can create a new image that is different from the original but still represents the same object or scene.
- Rotation : Rotating an image by a certain angle can create a new image that is different from the original but still represents the same object or scene.
- Scaling : Scaling an image by increasing or decreasing its size can create a new image that is different from the original but still represents the same object or scene.
- Cropping : Cropping an image by removing parts of it can create a new image that focuses on a specific region of the original image.
- Color jittering: Adjusting the brightness, contrast, saturation, and hue of an image can create a new image that has different colour characteristics from the original.

These are just some of the many data augmentation techniques that can be used to increase the size and diversity of a dataset. By applying these techniques to our data, we can create a larger and more diverse dataset that better represents the challenging scenarios we intend to model.

Roboflow

Roboflow is a platform that empowers developers to build their own computer vision applications, no matter their skillset or experience. It provides all of the tools needed to convert raw images into a custom trained computer vision model and deploy it for use in applications.

With Roboflow, developers can upload training data directly from the source, annotate images super fast right within their browser, conduct and manage

experiments all in one centralised place, and host a trained model with a single click. Roboflow also provides a streamlined workflow for identifying edge cases and deploying fixes. With each iteration, models become smarter and more accurate(Overview - Roboflow, 2023).

Roboflow offers various pricing plans to suit different needs. The Public plan is free forever and is best for personal, open source, and research projects. The Starter plan is \$249 per month and is best for startups and small businesses. The Enterprise plan offers custom deployment options, dedicated Machine Learning Engineers, and SLAs.

Features of Roboflow

Roboflow provides a comprehensive set of features to help developers build their own computer vision applications. Some of the key features of Roboflow include:

- Upload training data directly from the source: Upload files manually or via API including images, annotations, and videos. Roboflow supports dozens of annotation formats and makes it easy to continuously add new training data as you collect it.
- Annotate images super fast, right within someone's browser: Label using any operating system without downloading any software. Use the most popular annotation formats including JSON, XML, CSV, and TEXT. Our team can annotate hundreds of images in mere minutes.
- Conduct and manage experiments all in one centralised place: Assess the quality of chosen datasets and prepare them for training. Experiment with transformation tools to generate new training data and see what configurations lead to improved model performance.
- Host a trained model with a single click: Deploy a model to the cloud, the edge, or the browser. Get predictions where we need them and in half the time.

These are just some of the many features that Roboflow offers to help developers build their own computer vision applications. In addition to all the features mentioned above, Roboflow hosts over 90,000 public computer vision datasets with more than 66 million images(Computer Vision Datasets, 2017). These datasets are available in many popular formats including CreateML JSON, COCO JSON, Pascal VOC XML, YOLO v3, and Tensorflow TFRecords. Downsized and augmented versions are also available for users' convenience.

Software Requirements Specification

Functional and non-functional requirements

The ability to track the location and movements of autonomous vehicles in real-time is crucial for their safe and efficient operation. This requires a system that can accurately track vehicles even in challenging scenarios like harsh weather or low visibility. The system must also detect and avoid potential collisions with other vehicles, pedestrians, and obstacles. To achieve these requirements, the system must log and analyse data from the real-time cameras and other sensors to continuously improve their performance and safety. Additionally, the system must meet nonfunctional requirements such as reliability, scalability, low latency, security, usability, and maintainability to ensure its effectiveness and efficiency over time.

Functional Requirements

- The system must be capable of detecting and tracking various types of vehicles in real-time, even in challenging scenarios such as heavy rain or snow, fog, ice and snow covered roads, extreme temperatures, high speed merging, construction zones, emergency vehicles, pedestrian crossings, parking lots, and navigating through busy city centres.
- The system must be able to provide accurate and timely information to User.
- The system should be able to provide alerts and warnings to the autonomous car's control system to ensure safe and efficient operation.

Nonfunctional Requirements

- The system shall have a high degree of accuracy in tracking, even in challenging scenarios.
- The system shall be able to operate in a variety of challenging scenarios, including low-light conditions, adverse weather conditions, and complex urban environments, while maintaining minimal latency.
- The system shall be scalable and able to handle large volumes of data, while maintaining real-time operation.

Use case scenarios

General Use Case scenario for “Vehicle Tracking system using YOLOv8 algorithm”

Use Case Scenario: In a challenging scenario such as occlusion and low light conditions, the YOLO object detection algorithm is used to detect and recognize objects on the road. The system accurately tracks the location and movement of vehicles, providing crucial information for autonomous driving. The hyperparameters of the model are optimised to ensure accurate object detection and vehicle tracking, and the system's performance is evaluated using precision, recall, and F1-score metrics. The results of the evaluation demonstrate the system's effectiveness in challenging scenarios, providing robust and scalable object detection and vehicle tracking for autonomous vehicles. Vehicle tracking plays a critical role in autonomous driving, enabling self-driving vehicles to understand the movement of other vehicles on the road and make decisions accordingly. Here are some use case scenarios for vehicle tracking in challenging scenarios for autonomous driving as per our functional requirements:

Harsh weather conditions

1. Heavy rain or snow:

In heavy rain or snow, visibility can be severely limited, making it difficult for self-driving cars to navigate safely. Vehicle tracking technology can help the self-driving car identify other vehicles on the road, even when visibility is low, and adjust its path accordingly (see Figure 2). Pre- and Post- conditions described in Table 1.

Table 1.

Heavy rain or snow driving condition Use Case

Use Case	Pre-condition	Post-condition
Heavy rain or snow.	Self-driving cars are operating in heavy rain or snow.	Self-driving car adjusts its path to ensure safe navigation in heavy rain or snow.

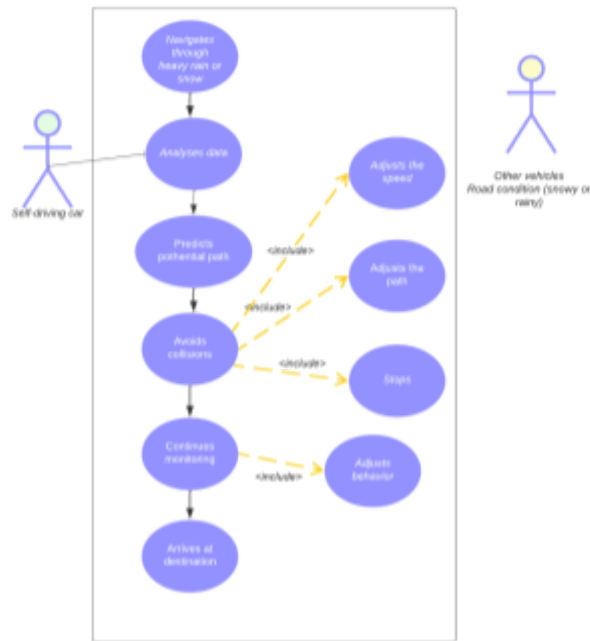


Figure 2

Heavy rain or snow driving condition

2. Fog:

Fog is another weather condition that can reduce visibility and make it difficult for self-driving cars to detect obstacles on the road. Vehicle tracking technology can help the self-driving car identify other vehicles and objects in its path, even when visibility is limited. (see Figure 3). Pre- and Post- conditions described in Table 2.

Table 2.

Driving in foggy condition Use Case

Use Case	Pre-condition	Post-condition
Fog	Self-driving cars are operating in foggy weather conditions.	Self-driving car successfully navigates through the foggy conditions and reaches its destination or continues its journey safely.



Figure 3.

Driving in foggy condition

3. Ice and snow-covered roads:

Ice and snow can create slippery conditions on the road, making it challenging for self-driving cars to maintain traction and avoid accidents. Vehicle tracking technology can monitor the movements of nearby vehicles and adjust the self-driving car's speed and path to ensure that it travels safely on the road. (see Figure 4). Pre- and Post- conditions described in Table 3.

Table 3.

Ice & snow-covered roads Use Case

Use Case	Pre-condition	Post-condition
Ice and snow-covered roads	Self-driving cars encounter ice or snow-covered roads.	Self-driving cars safely navigate through ice or snow-covered roads.



Figure 4.

Ice & snow-covered roads Use Case

4. Extreme temperatures:

In extreme temperatures, such as extreme heat or cold, self-driving cars may need to adjust their behaviour to ensure that their systems are functioning correctly. Vehicle tracking technology can monitor the temperature of the vehicle and its surroundings, helping the self-driving car adjust its behaviour to ensure that it remains safe and efficient. (see Figure 5). Pre- and Post- conditions described in Table 4.

Table 4.

Extreme temperature conditions Use Case

Use Case	Pre-condition	Post-condition
Extreme temperatures	Self-driving cars are operating in extreme temperatures.	Self-driving car adjusts its behaviour to ensure safe and efficient operation in extreme temperatures.



Figure 5.

Extreme temperature conditions Use Case

Other challenging situations

1. High-speed merging:

A self-driving car is merging onto a highway with fast-moving traffic. Vehicle tracking technology can monitor the speed and distance of nearby vehicles to determine the best time to merge, while also ensuring that the self-driving car maintains a safe following distance. (see Figure 6). Pre- and Post- conditions described in Table 5.

Table 5.

High speed merging Use Case

Use Case	Pre-condition	Post-condition
High-speed merging	1. The self-driving car is equipped with vehicle tracking technology.	The self-driving car has successfully

	<ol style="list-style-type: none"> The self-driving car has identified an on-ramp to merge onto the highway. There is fast-moving traffic on the highway 	merged onto the highway and is travelling at a safe speed and following distance from other vehicles.
--	--	---



Figure 6.

High speed merging Use Case

2. Construction zones:

Construction zones can be challenging for self-driving cars due to changes in road conditions and lane closures. Vehicle tracking technology can monitor the movements of construction vehicles and workers, helping the self-driving car navigate through the area safely. (see Figure 7). Pre- and Post- conditions described in Table 6.

Table 6.

Construction zone Use Case

Use Case	Pre-condition	Post-condition
Construction zones	<ol style="list-style-type: none"> 1. Self-driving cars are in operation. 2. The self-driving car is approaching a construction zone. 	<ol style="list-style-type: none"> 1. The self-driving car has safely navigated through the construction zone. 2. Self-driving cars resumes its previous course.



Figure 7.

Construction zone Use Case

3. Emergency vehicles:

Self-driving cars need to be able to respond to emergency vehicles, such as ambulances and fire trucks, which require priority access to the road. Vehicle tracking technology can detect the approach of emergency vehicles and adjust the self-driving car's path to give them clear passage. (see Figure8).Pre- and Post-conditions described in Table 7.

Table 7.

Emergency Use Case

Use Case	Pre-condition	Post-condition
Emergency vehicles	<ol style="list-style-type: none"> 1. Self-driving cars are in operation. 2. Emergency vehicle is approaching. 	<ol style="list-style-type: none"> 1. Self-driving cars have safely given priority passage to the emergency vehicle. 2. Self-driving cars resumes its previous course.

**Figure 8.***Emergency Use Case*

4. Pedestrian crossings:

Pedestrians are unpredictable and can be difficult for self-driving cars to detect. Vehicle tracking technology can monitor the movements of nearby vehicles and pedestrians to determine the safest path forward, while also ensuring that the self-driving car gives pedestrians the right of way. (see Figure 9). Pre- and Post-conditions described in Table 8.

Table 8.*Pedestrian crossing Use Case*

Use Case	Pre-condition	Post-condition
Pedestrian crossings	<ol style="list-style-type: none"> 1. The self-driving car is in operation and 	<ol style="list-style-type: none"> 1. The self-driving car has safely navigated

	<p>has access to vehicle tracking technology.</p> <p>2. The pedestrian crossing is within the range of the self-driving car's sensors and is properly marked.</p>	<p>through the pedestrian crossing, giving pedestrians the right of way.</p> <p>2. No accidents or incidents have occurred.</p>
--	---	---

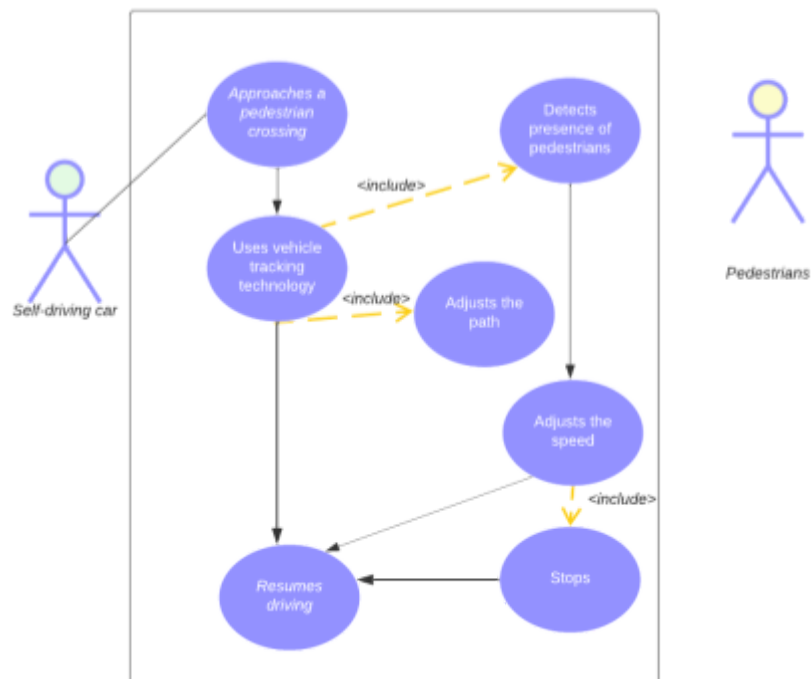


Figure 9.

Pedestrian crossing Use Case

5. Parking lots:

Parking lots can be challenging for self-driving cars due to tight spaces and limited visibility. Vehicle tracking technology can monitor the movements of nearby vehicles and obstacles to help the self-driving car park safely and efficiently. (see Figure 10). Pre- and Post- conditions described in Table 9.

Table 9.

Parking lots Use Case

Use Case	Pre-condition	Post-condition
Parking lots	<ol style="list-style-type: none"> 1. The self-driving car needs to park in a designated parking spot. 2. The parking lot needs to be equipped with vehicle tracking technology. 3. The parking lot needs to have clear markings and signage indicating parking spots. 	The self-driving car is parked safely and efficiently in the designated spot.

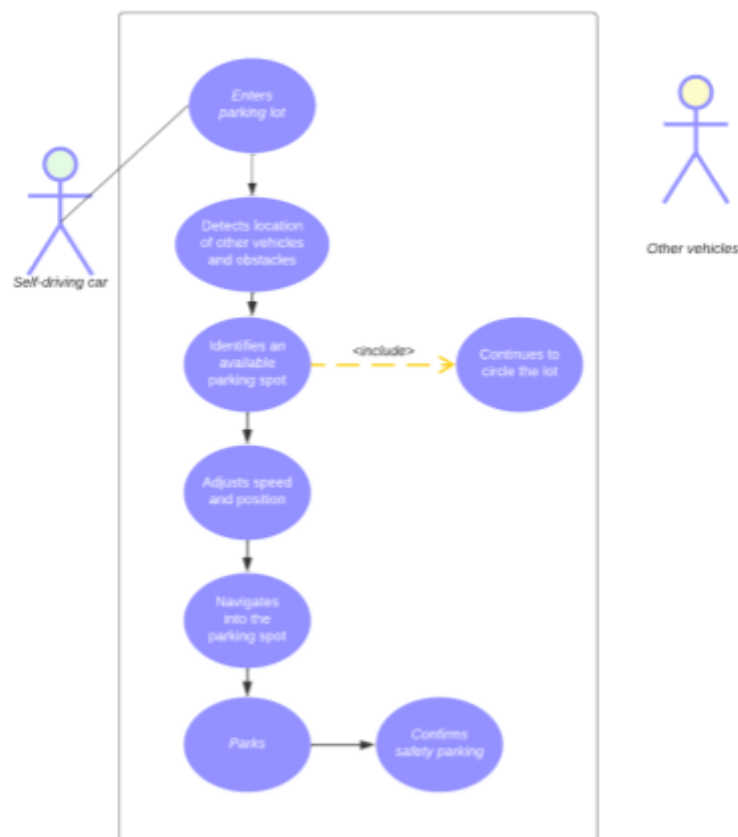


Figure 10.

Parking lots Use Case

6. Navigating through busy city centre:

Driving in rush hour can be challenging due to close proximity of other vehicles. By analysing data from sensors and cameras, the system predicts the movements of other vehicles and adjusts the self-driving car's path accordingly to avoid potential collisions. (see Figure 11). Pre- and Post- conditions described in Table 10.

Table 10.

Navigation through busy city centre Use Case

Use Case	Pre-condition	Post-condition
Navigating through busy city centre	The self-driving car is driving in a busy city centre during rush hour.	The self-driving car successfully navigates through the city centre without colliding with other vehicles.

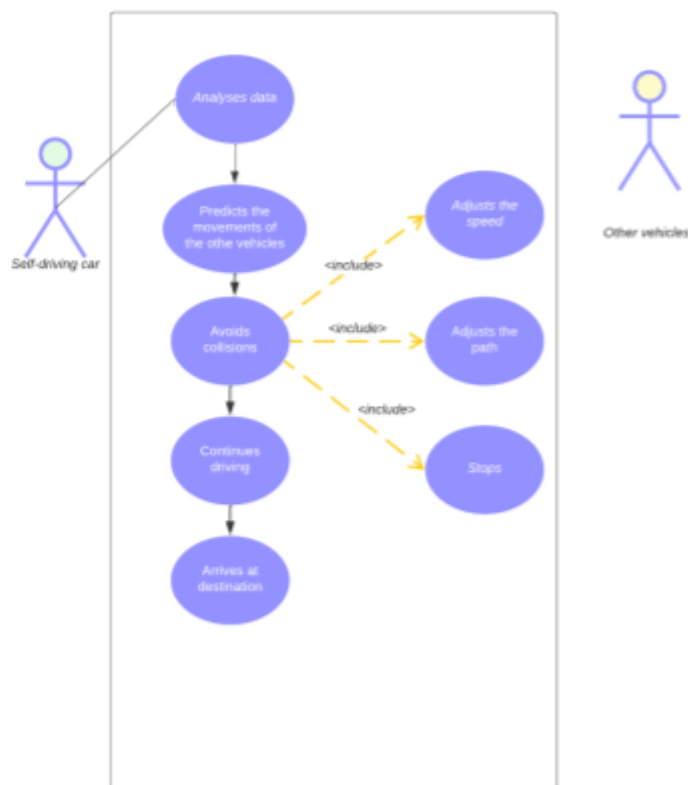


Figure 11.

Navigation through busy city centre Use Case

UML diagrams

UML (Unified Modeling Language) is a standardised visual modelling language used in software development to represent system requirements, design, and architecture. There are several types of UML diagrams, each with its own specific purpose and use case.

Class diagram

Class Diagram is a type of UML diagram that visually represents the static structure of a system or application, showing the classes, their attributes, methods, and relationships (see Figure 12). It provides a high-level overview of the object-oriented structure of the system, depicting the classes and their interactions. Class diagrams are commonly used for modelling the data and behaviour of a system, supporting software design, analysis, and documentation. Class diagrams can help to visualise the relationships between classes, such as associations, aggregations, inheritances, and dependencies, and can also depict constraints, multiplicity, and visibility of class members. Class diagrams are widely used in object-oriented modelling and design, supporting the communication and understanding of the structure and behaviour of complex systems.

Class Vehicle Detector: in the attributes section we have one private variable `detection_model`. In our instance it is a class type.

The parameters used for our `track_vehicles()` method in the Vehicle Detector are:

1. `input_video`: This parameter would specify the path to the input video file or a live camera stream to be processed by the detector.
2. `output_video`: This parameter would specify the path to the output video file that contains the frames with bounding boxes drawn around the detected vehicles. This parameter could also be a live video stream.
3. `conf_threshold`: This parameter would specify the confidence threshold for the detector to consider a detection as valid. The threshold is usually a value between 0 and 1, where a higher value would result in fewer but more confident detections.
4. `nms_threshold`: This parameter would specify the non-maximum suppression (NMS) threshold for the detector to eliminate redundant detections. The threshold is usually a value between 0 and 1, where a higher value would result in fewer but more accurate detections.
5. `draw_boxes`: This parameter would specify whether the detector should draw bounding boxes around the detected vehicles in the output video or not. This could be a boolean value, where `True` would indicate that bounding boxes should be drawn, and `False` would indicate that no boxes should be drawn.

6. `save_output`: This parameter would specify whether the output video should be saved to a file or not. This could be a boolean value, where `True` would indicate that the output should be saved, and `False` would indicate that the output should only be displayed but not saved.

A YOLOv8 detector class is a class used for object detection, which utilises the YOLOv8 object detection algorithm. The class could have the following attributes:

1. `model`: The YOLOv8 object detection model, class type.
2. `conf_threshold`: is a float value between 0 and 1 that specifies the minimum confidence score required for an object to be detected.
3. `nms_threshold`: The non-maximum suppression threshold, which is a float value between 0 and 1 that specifies the maximum overlap allowed between bounding boxes for different objects.
4. `input_size`: The input size for the YOLOv8 model, which is a tuple of (width, height) values specifying the size of the input images to be processed.
5. `class_names`: The list of class names, which is a string or list of strings that specify the names of the classes to be detected.

The class could have the following methods:

1. `__init__(self, model, conf_threshold, nms_threshold, input_size, class_names)`: The constructor method that initialises the YOLOv8 detector class with the specified attributes.
2. `train(self)`: A method that trains the YOLOv8 model using a dataset of labelled images.
3. `detect(self, image, conf_threshold, nms_threshold)`: A method that performs object detection on an input image using the YOLOv8 model. This method should return a list of objects detected in the image, where each object is represented by an instance of the `Object` class.
4. `load_model(self, weights_file_path)`: A method that loads a pre-trained YOLOv8 model from a specified file path.
5. `set_conf_threshold(conf_threshold)`: A method that sets the confidence threshold for object detection to a specified value.

The `Object` class represents an object detected in an image or video stream, such as a person, car, or traffic sign. Here is a description of its attributes and methods:

Attributes:

1. `ID` (integer): A unique identifier for the object.
2. `class_names` (string): The label of the object's class, in our case, "car" and "truck".
3. `confidence` (float): The confidence score for the detection, which represents the probability that the detected object is of the specified class.

4. `bbox` (tuple): The bounding box coordinates for the detected object, represented as a tuple of four values: (`xmin`, `ymin`, `xmax`, `ymax`). These values specify the coordinates of the top-left and bottom-right corners of the bounding box, respectively.

Methods:

1. `get_ID()`: This method returns the ID of the object.
2. `get_class_name()`: This method returns the name of the object's class.
3. `get_conf_score()`: This method returns the confidence score for the object detection.

In our case, the vehicle class inherits all of the attributes and methods of the object class. In a class diagram, this relationship is represented by an inheritance arrow from the vehicle class to the object class.

The bounding box class has the following attributes:

1. `xmin` (integer): The x-coordinate of the top-left corner of the bounding box.
2. `ymin` (integer): The y-coordinate of the top-left corner of the bounding box.
3. `xmax` (integer): The x-coordinate of the bottom-right corner of the bounding box.
4. `ymax` (integer): The y-coordinate of the bottom-right corner of the bounding box.

Methods:

1. `get_xmin()`: This method returns the x-coordinate of the top-left corner of the bounding box.
2. `get_ymin()`: This method returns the y-coordinate of the top-left corner of the bounding box.
3. `get_xmax()`: This method returns the x-coordinate of the bottom-right corner of the bounding box.
4. `get_ymax()`: This method returns the y-coordinate of the bottom-right corner of the bounding box.
5. `set_xmin(xmin)`: This method sets the x-coordinate of the top-left corner of the bounding box to the specified value.
6. `set_ymin(ymin)`: This method sets the y-coordinate of the top-left corner of the bounding box to the specified value.
7. `set_xmax(xmax)`: This method sets the x-coordinate of the bottom-right corner of the bounding box to the specified value.
8. `set_ymax(ymax)`: This method sets the y-coordinate of the bottom-right corner of the bounding box to the specified value.

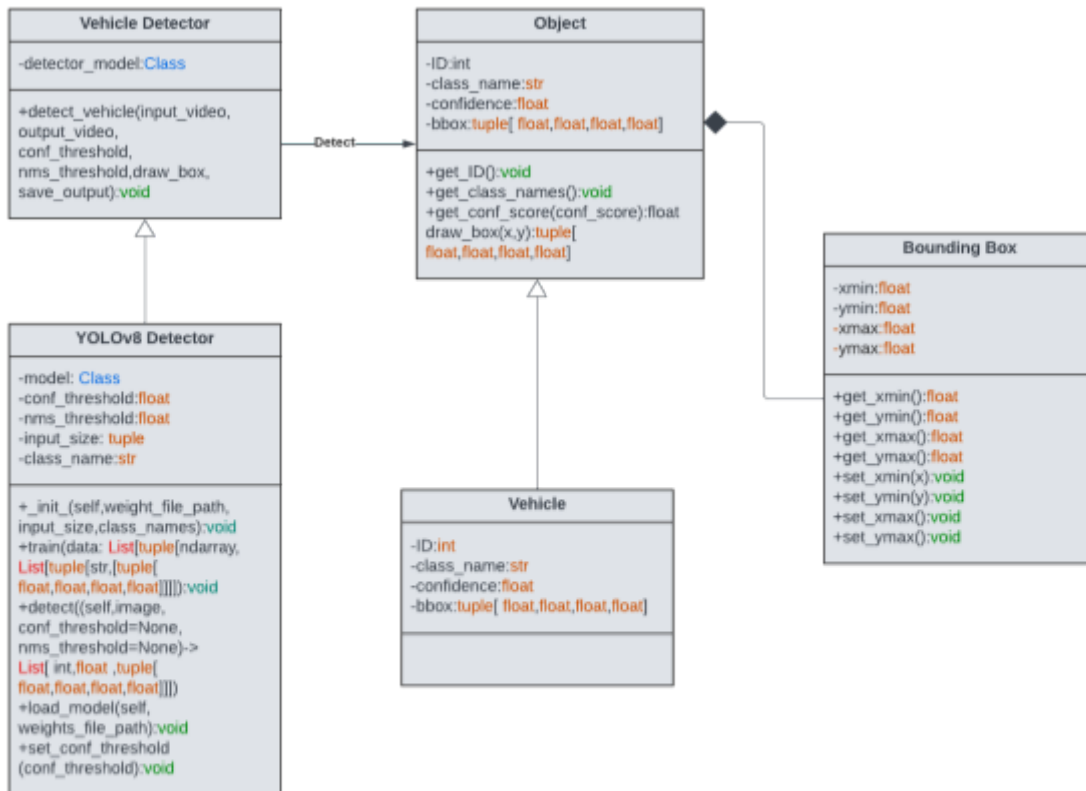


Figure 12.

Class diagram of Vehicle Tracking System using YOLOv8 model

3

Activity diagram

An activity diagram is a type of Unified Modeling Language (UML) diagram that is used to model the behaviour or flow of activities or actions within a system, process, or workflow. It represents the dynamic aspects of a system, focusing on the flow of actions, decisions, and transitions between activities. Activity diagrams are commonly used in business process modelling, system analysis, and software development to visualise and analyse the flow of activities and their interactions. Key features: activities, control flow, decision nodes, fork and join nodes, initial and final nodes, swimlanes.

The activity diagram shows the different activities involved in the vehicle tracking system, including the video preprocessing system, the YOLOv8 tracker, and the two classes that the system can identify.

³By clicking on the image, you can access the link to LucidChart

The activity diagram begins with the Video Input action, which is then followed by the Preprocessing System Swimlane, which includes a set of video preprocessing activities such as video stabilisation, frame resizing and scaling, contrast enhancement, noise reduction, and object ROI extraction (see Figure 13). These activities are grouped together using a structured activity node to make the diagram more organised.

After the video preprocessing system, the YOLOv8 Tracker Swimlane returns, which includes the activities of vehicle detection, multi-object tracking, occlusion handling, followed by Vehicle Identification and confidence score calculation. The Confidence Score activity is then followed by a decision activity, which determines whether the object detected belongs to Class 'car' or Class 'truck'.

If the object belongs to Class 'car', the diagram continues with the Class 'car' Swimlane, which includes the activities of display ID, display confidence score, and display bounding box. If the object belongs to Class 'truck', the diagram continues with the according Class Swimlane, which includes the activities of display ID, display confidence score, and display bounding box.

To summarise, the activity diagram provides a clear visualisation of the different activities involved in the vehicle tracking system and how they are related to each other.

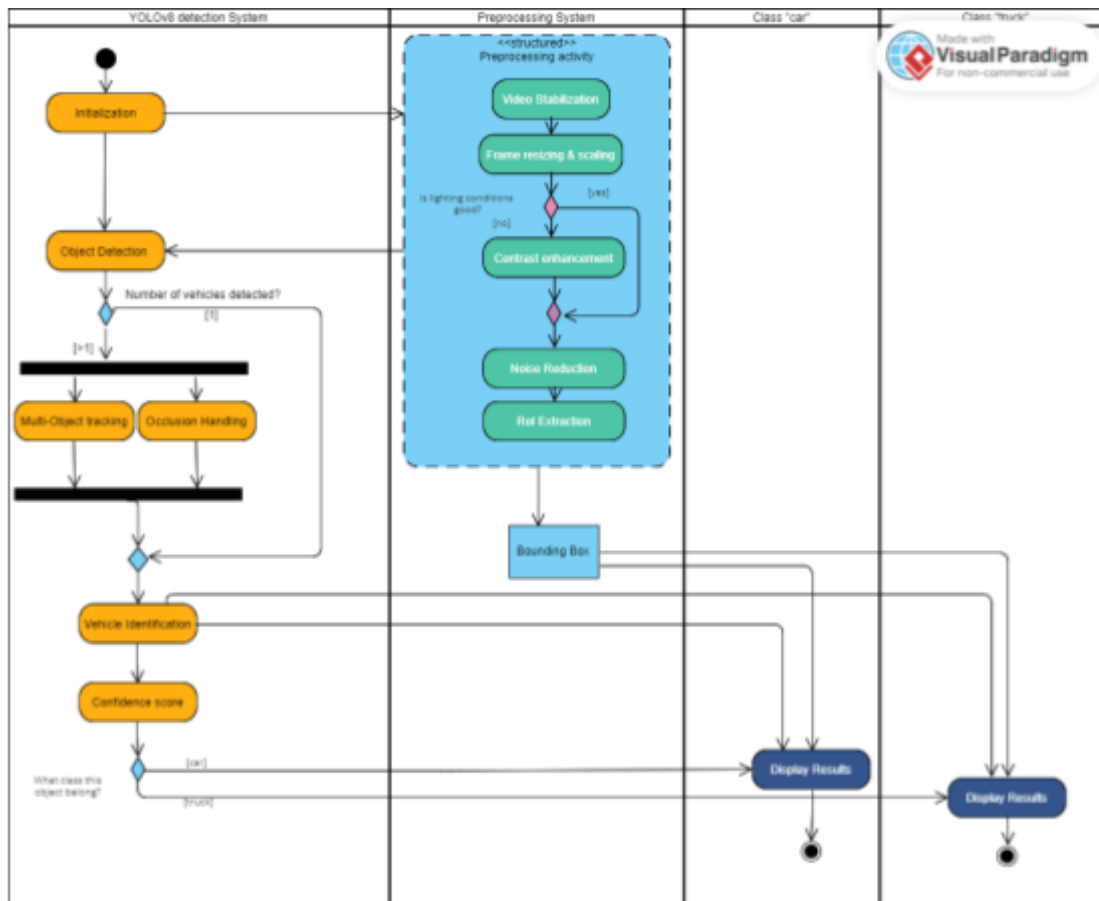


Figure 13.

Activity diagram of Vehicle Tracking System using YOLOv8 model

UX/UI Modeling

User experience (UX) and user interface (UI) modelling are critical components of modern design processes. Our goal is to create an exceptional user experience and visually appealing interface. In this section, we will dive into the world of UX and UI modelling, exploring techniques and approaches that will help us craft a one-of-a-kind digital product. UX modelling involves understanding and shaping the overall experience of users as they interact with a product, while UI modelling focuses on the visual and interactive elements that users interact with on the screen. We will cover key concepts including information architecture, interaction design, visual design, and usability testing, to ensure that our digital product meets the needs and expectations of our users.

User flow diagram

User flow diagrams, also known as flowcharts or process maps, are visual representations of the steps a user takes to complete a specific task or achieve a

particular goal within a digital product or system. They provide a visual overview of the user's journey, depicting the sequence of actions, decisions, and interactions that a user goes through to accomplish their objective. User flow diagrams are widely used in user experience (UX) and user interface (UI) design to understand, analyse, and optimise the user experience and interaction design of a digital product. Use flow diagram for our vehicle tracking app can be seen in Figure 14. The diagram starts with 'open app' button then followed by decision fork whether a user being asked to sign in or sign up. After a successful verification process, user is prompted to choose a file from a local machine whether it is a video file or an image. By clicking the 'Download' button user will be shown a processed image with the objects being classified. The diagram finishes with the 'End' button.



Figure 14.

Flow diagram of Vehicle Tracking App

4

Lo-Fi Sketches

Lo-Fi sketches, also known as low-fidelity sketches or wireframes, are a visual representation of the basic layout and functionality of a user interface or system. They are typically simple and quick hand-drawn or digital sketches that focus on the overall structure, content, and flow of a design, without going into details of the visual design or aesthetics. Lo-Fi sketches are a valuable tool in the early stages of the design process.

In our iterative design process, we have made several alterations to the initial design idea, starting with basic sketches or wireframes. After each alteration, we have conducted brief usability tests to evaluate the effectiveness of the design changes and gather feedback from users. These usability tests have helped us identify and address potential usability issues, gather insights on user preferences and behaviour, and make data-driven decisions to further improve the design. Additionally, these tests have allowed us to validate the design

⁴By clicking on the image, you can access the link to Figma

changes against the intended goals and requirements, ensuring that the design aligns with the needs and expectations of the target users. By incorporating usability testing at each iteration, we have been able to continuously refine and optimise the design, resulting in a more user-friendly and effective solution.(See Figure 15)

User testing 1

Here are the questions we asked our users:

1. Can you easily locate and identify buttons on the screen?
2. How easy or difficult was it for you to click on the buttons?
3. Do you have any suggestions for further improving the design?
4. Were there any buttons that were particularly challenging for you to locate or identify on the screen?
5. How confident do you feel in your ability to accurately click on the buttons?
6. Did you encounter any issues or difficulties when trying to click on the buttons?
7. Do you have any suggestions for improving the visibility or accessibility of the buttons?
8. Were there any aspects of the design that you found confusing or unclear in relation to the buttons?
9. Did you encounter any unexpected behaviours or errors while interacting with the buttons?
10. How satisfied are you with the overall usability of the buttons in the current design?

And here are the answers from 5 users:

User 1:

1. Yes, I was able to easily locate and identify the buttons on the screen.
2. It was relatively easy for me to click on the buttons.
3. One suggestion for improvement could be to increase the size of the buttons slightly to make them even more accessible.
4. 4. No, I was able to easily locate and identify all the buttons on the screen.
5. It was fairly easy for me to click on the buttons.
6. No, I did not encounter any difficulties in clicking on the buttons.
7. One suggestion for improvement could be to increase the contrast between the buttons and the background to make them stand out more.
8. I found the design to be clear and intuitive in relation to the buttons.
9. No, I did not encounter any unexpected behaviours or errors.
10. I am satisfied with the usability of the buttons in the current design.



Figure 15.

Initial design Lo-Fi sketches

5

User 2:

1. Yes, I found it easy to locate and identify the buttons on the screen.
2. It was fairly easy for me to click on the buttons.
3. I think the design is already user-friendly, but adding some visual cues or feedback when clicking on the buttons could further enhance the user experience.
4. I had some difficulty locating one of the buttons on the screen as it was placed in a less prominent position.
5. It was moderately easy for me to click on the buttons.
6. I encountered some issues in clicking on the less prominent button due to its placement.
7. One suggestion for improvement could be to make all buttons equally prominent to avoid confusion.

⁵ By clicking on the image, you can access the link to Figma

8. I found the design to be generally clear, but the placement of one button could be improved. Also, there is no navigation back to the landing page from the image-screening page.
9. No, I did not encounter any unexpected behaviours or errors.
10. I am moderately satisfied with the usability of the buttons in the current design.

User 3:

1. Yes, I could easily locate and identify the buttons on the screen.
2. It was not difficult for me to click on the buttons.
3. One suggestion for improvement could be to add some colour contrast to the buttons to make them stand out more.
4. Yes, I could easily locate and identify all the buttons on the screen.
5. It was very easy for me to click on the buttons.
6. I did not encounter any difficulties in clicking on the buttons.
7. I think the design is already user-friendly and does not require any major improvements.
8. I found the design to be clear and easy to understand in relation to the buttons.
9. No, I did not encounter any unexpected behaviours or errors.
10. I am highly satisfied with the usability of the buttons in the current design.

User 4:

1. Yes, I was able to easily locate and identify the buttons on the screen.
2. It was relatively easy for me to click on the buttons.
3. I think the design is already intuitive, but what if instead of repeating buttons "Choose File" and "Send" on a second screen, just add a button "Download".
4. Yes, I was able to easily locate and identify all the buttons on the screen.
5. It was relatively easy for me to click on the buttons.
6. I did not encounter any difficulties in clicking on the buttons.
7. One suggestion for improvement could be to increase the font size of the button labels for better readability.
8. I found the design to be clear, but the button labels could be slightly larger.
9. No, I did not encounter any unexpected behaviours or errors.
10. I am satisfied with the usability of the buttons in the current design.

User 5:

1. Yes, I found it easy to locate and identify the buttons on the screen.
2. It was not difficult for me to click on the buttons.
3. One suggestion for improvement could be to group related buttons together or provide a visual hierarchy to indicate their importance.
4. No, I did not have any difficulties in locating or identifying the buttons on the screen.

5. It was very easy for me to click on the buttons.
6. I did not encounter any difficulties in clicking on the buttons.
7. I think the design is already effective and does not require any major improvements.
8. I found the design to be clear and intuitive in relation to the buttons.
9. No, I did not encounter any unexpected behaviours or errors.
10. I am highly satisfied with the usability of the buttons in the current design.

Based on the feedback from the users, here are some proposed changes to the initial design:

1. Increase the contrast between the buttons and the background to make them stand out more.
2. Ensure all buttons are equally prominent to avoid confusion in locating them.
3. Consider increasing the font size of the button labels for better readability.
4. Make the button labels slightly larger to improve clarity.
5. On the second screen with the option "Download" the chosen image, add a button "Download".
6. Add a button "Back" on the image-displaying page. This button will navigate back to the landing page. (see Figure 16)

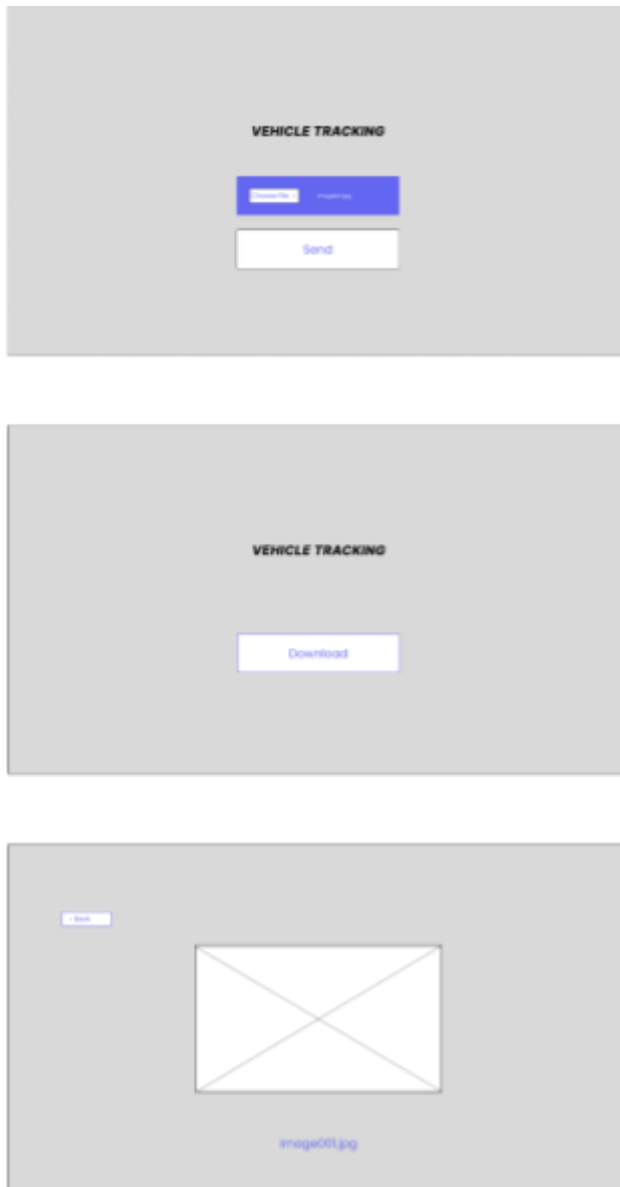


Figure 16.

Iterated design Lo-Fi sketches

User testing 2

1. Can you easily locate and identify the buttons on the screen with the increased contrast and improved prominence?
2. How easy or difficult was it for you to click on the buttons with the updated button size and label clarity?
3. Are the font size and readability of the button labels improved after the changes?
4. Is the addition of the "Download" button on the second screen with the option to download the chosen image clear and easy to understand?

5. Does the addition of the "Back" button on the image-displaying page help you navigate back to the landing page easily?

User 1:

1. Can you easily locate and identify the buttons on the screen with the increased contrast and improved prominence?

Answer: Yes, the buttons are now more noticeable and easy to locate.

2. How easy or difficult was it for you to click on the buttons with the updated button size and label clarity?

Answer: It was much easier to click on the buttons after the changes. The updated size and label clarity improved usability.

3. Are the font size and readability of the button labels improved after the changes?

Answer: Absolutely! The increased font size made the button labels more readable and clear.

4. Is the addition of the "Download" button on the second screen with the option to download the chosen image clear and easy to understand?

Answer: Yes, the "Download" button is now clearly visible and easy to understand.

5. Does the addition of the "Back" button on the image-displaying page help you navigate back to the landing page easily?

Answer: Yes, the "Back" button is a useful addition and makes it easy to navigate back to the landing page.

User 2:

1. Can you easily locate and identify the buttons on the screen with the increased contrast and improved prominence?

Answer: Yes, the buttons are now more prominent and easy to locate.

2. How easy or difficult was it for you to click on the buttons with the updated button size and label clarity?

Answer: It was much easier to click on the buttons after the changes. The updated size and label clarity improved usability.

3. Are the font size and readability of the button labels improved after the changes?

Answer: Yes, the font size and readability of the button labels have improved significantly after the changes.

4. Is the addition of the "Download" button on the second screen with the option to download the chosen image clear and easy to understand?

Answer: Yes, the "Download" button is now clearly visible and easy to understand.

5. Does the addition of the "Back" button on the image-displaying page help you navigate back to the landing page easily?

Answer: Yes, the "Back" button is a helpful addition and makes it easy to navigate back to the landing page.

User 3:

1. Can you easily locate and identify the buttons on the screen with the increased contrast and improved prominence?

Answer: Yes, the buttons are now more noticeable and easy to locate.

2. How easy or difficult was it for you to click on the buttons with the updated button size and label clarity?

Answer: It was much easier to click on the buttons after the changes. The updated size and label clarity improved usability.

3. Are the font size and readability of the button labels improved after the changes?

Answer: Absolutely! The increased font size made the button labels more readable and clear.

4. Is the addition of the "Download" button on the second screen with the option to download the chosen image clear and easy to understand?

Answer: Yes, the "Download" button is now clearly visible and easy to understand.

5. Does the addition of the "Back" button on the image-displaying page help you navigate back to the landing page easily?

Answer: Yes, the "Back" button is a useful addition and makes it easy to navigate back to the landing page.

User 4:

1. Can you easily locate and identify the buttons on the screen with the increased contrast and improved prominence?

Answer: Yes, the buttons are now more noticeable and easy to locate.

2. How easy or difficult was it for you to click on the buttons with the updated button size and label clarity?

Answer: It was much easier to click on the buttons after the changes. The updated size and label clarity improved usability.

3. Are the font size and readability of the button labels improved after the changes?

Answer: Yes, the font size and readability of the button labels have improved significantly after the changes.

4. Is the addition of the "Download" button on the second screen with the option to download the chosen image clear and easy to understand?

Answer: Yes, the "Download" button is now clearly visible and easy to understand.

5. Does the addition of the "Back" button on the image-displaying page help you navigate back to the landing page easily?

Answer: Yes, the "Back" button is a helpful addition and makes it easy to navigate back to the landing page.

User 5:

1. Can you easily locate and identify the buttons on the screen with the increased contrast and improved prominence?

Answer: Yes, the buttons are now more prominent and easy to locate.

2. How easy or difficult was it for you to click on the buttons with the updated button size and label clarity?

Answer: It was much easier to click on the buttons after the changes. The updated size and label clarity improved usability.

3. Are the font size and readability of the button labels improved after the changes?

Answer: Yes, the font size and readability of the button labels have improved significantly after the changes.

4. Is the addition of the "Download" button on the second screen with the option to download the chosen image clear and easy to understand?

Answer: Yes, the "Download" button is now clearly visible and easy to understand.

5. Does the addition of the "Back" button on the image-displaying page help you navigate back to the landing page easily?

Answer: Yes, the "Back" button is a useful addition and makes it easy to navigate back to the landing page.

Design System

A design system is a comprehensive set of guidelines, principles, and assets that provide a centralised source of truth for creating consistent and cohesive digital products. It is a structured approach to design and development that promotes efficiency, consistency, and usability across different interfaces, platforms, and devices.

A design system typically includes a set of components such as typography, colour palette, spacing, icons, form elements, and other UI elements, as well as guidelines on how these components should be used in the design and development process. The main purpose of a design system is to establish a consistent visual language and user experience.

When it comes to the stage of creating a design system, there are several important aspects to consider: (1) purpose and goals, (2) user needs, (3) components and guidelines, (4) testing and iteration.

And when it comes to choosing colours for a design system, opting for simple colours and keeping their numbers to a minimum can offer several benefits: (1) Consistency, (2) Clarity, (3) Flexibility, (4) Efficiency.

So our decision was to keep the design profile simple with a minimum number of colours yet provide full functionality and appealing look. (see Figures 17.1 and 17.2)

Design System for Web app

Colors

Primary	Secondary	Text	
			
			

Typography

Inter	H1	Inter ExtraBold 32
	H2	Inter Bold 24
	H3	Inter Semibold 18
	Medium	Inter Medium 16
	Regular	Inter Regular 16

Buttons



Checkbox

- ☒ Checked
- ☐ Unchecked

System icons

-  Settings
-  In-progress tasks

Figure 17.1.

Design System

Input Fields



File Explorer Icons



Drop Shadow



Avatars



Figure 17.2.

⁶ *Design System*

Hi-Fi prototypes

The hi-fi section of our project represents the pinnacle of our design process, where we have refined our initial concepts and transformed them into a highly polished and visually appealing prototype. In this section, we have leveraged advanced design tools and techniques to create a high-fidelity representation of our final product or user interface.

We have done our best to incorporate user feedback and enhance the design with a side panel that displays the user's name and entrance logs, providing a personalised experience. Our hi-fi prototype includes detailed visual elements, and functional features that closely mimic the intended user experience of the final product. The design has been refined based on user feedback, culminating

⁶ By clicking on the image, you can access the link to Figma

our design efforts and providing a comprehensive and visually engaging representation of our product's intended functionality and aesthetics. It serves as a critical milestone in our design process, helping us to refine and finalise our design before moving towards the actual implementation phase. (see Figure 18)

User testing 3

1. How would you rate the overall visual design and aesthetics of the app?
2. Were you able to easily navigate through the different features and options of the app?
3. Did you find the process of adding a new vehicle/image/video to your tracking list intuitive and user-friendly?
4. Did you encounter any issues or difficulties while using the app? If so, please provide details.
5. Were the entrance logs in the side panel helpful in tracking the history of vehicle entries? Why or why not?
6. How easy was it to access and view the details of past vehicle entries in the entrance logs?
7. Based on your experience with the app, would you recommend it to others? Why or why not?

User 1:

- I would rate the overall visual design and aesthetics of the app as 8 out of 10. The app has a clean and modern design with visually appealing colour schemes and icons.
- Yes, I was able to easily navigate through the different features and options of the app. The menu icons were clear, and the app's layout was intuitive and easy to follow.
- Adding a new vehicle/image to my tracking list was straightforward and user-friendly. The process was intuitive, and I could easily input the required information without any issues.
- Yes, the entrance logs in the side panel were helpful in tracking the history of vehicle entries. I could easily view the dates.
- It was very easy to access and view the details of past vehicle entries in the entrance logs. The logs were organised chronologically, making it convenient to find specific entries.
- Based on my experience with the app, I would definitely recommend it to others. The entrance logs feature was particularly useful in keeping track of my activity's history, and overall, the app was user-friendly and efficient.

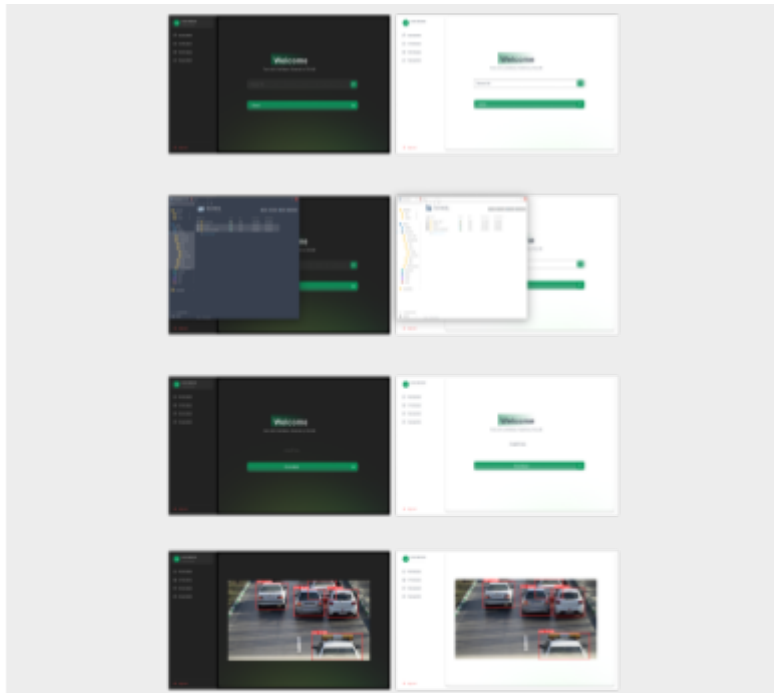


Figure 18.

Iterated design Hi-Fi prototypes

Note: displayed in two modes, dark and light.

7

2. User 2:

- I would rate the overall visual design and aesthetics of the app as 9 out of 10. The app has a sleek and modern design with visually pleasing colour palettes and well-designed icons.
- Yes, I found it easy to navigate through the different features and options of the app. The side panel and menu icons were clear and made it simple to access the functionalities I needed.
- Adding a new vehicle to my tracking list was straightforward and user-friendly. The app guided me through the process with clear instructions, and I could easily input the necessary information without any difficulties.
- Yes, the entrance logs in the side panel were helpful in tracking the history of vehicle entries. The logs provided a clear overview of past entries.
- It was easy to access and view the details of past vehicle entries in the entrance logs. The logs were neatly organised in a list view and were easily scrollable for quick review.
- Based on my experience with the app, I would recommend it to others. The entrance logs were a valuable feature for me, and I found the app overall user-friendly and effective in tracking my vehicle's location.

3. User 3:

⁷ By clicking on the image, you can access the link to Figma

- I would rate the overall visual design and aesthetics of the app as 7 out of 10. The app has a functional design with decent colour schemes and icons, but it could be more visually appealing.
- Yes, I was able to easily navigate through the different features and options of the app. The side panel and menu icons were clear and made it convenient to access the functionalities I needed.
- Adding a new vehicle to my tracking list was straightforward and user-friendly. The app provided clear instructions, and I could easily input the necessary information without any issues.
- Yes, the entrance logs in the side panel were helpful in tracking the history of vehicle entries. The logs provided a chronological list of past entries, which helped me keep track of my tracking activity.
- It was very easy to access and view the details of past vehicle entries in the entrance logs. The logs had a simple and intuitive design, and I could easily navigate through them to view the necessary details.
- Based on my experience with the app, I would recommend it to others. The entrance logs feature was helpful in tracking the history of my vehicle entries, and the app was overall user-friendly and efficient in its functionality.

4. User 4:

- I would rate the overall visual design and aesthetics of the app as 9 out of 10. The app has a modern and visually appealing design with clear icons and a well-organised layout.
- Yes, I found it easy to navigate through the different features and options of the app. The side panel and menu icons were clear and made it simple to access the functionalities I needed.
- Adding a new vehicle to my tracking list was intuitive and user-friendly. The app guided me through the process with clear instructions, and I could easily input the necessary information without any difficulties.
- Yes, the entrance logs in the side panel were helpful in tracking the history of vehicle entries. The logs provided a clear timeline of past entries.
- It was easy to access and view the details of past vehicle entries in the entrance logs. The logs were well-organised, making it convenient to find and review specific entries.
- Based on my experience with the app, I would recommend it to others. The entrance logs feature was useful in keeping track of my vehicle's history, and the app was overall user-friendly and effective in helping me track my vehicle's location.

5. User 5:

- I would rate the overall visual design and aesthetics of the app as 8 out of 10. The app has a clean and modern design with visually pleasing colour schemes and icons.

- Yes, I was able to easily navigate through the different features and options of the app. The side panel and menu icons were clear and made it convenient to access the functionalities I needed.
- Adding a new vehicle to my tracking list was straightforward and user-friendly. The app provided clear instructions, and I could easily input the necessary information without any issues.
- Yes, the entrance logs in the side panel were helpful in tracking the history of vehicle entries. The logs provided a comprehensive list of past entries by dates.
- It was very easy to access and view the details of past vehicle entries in the entrance logs. The logs had a simple and intuitive design, and I could easily scroll through them to review the necessary information.
- Based on my experience with the app, I would recommend it to others.

The UX/UI modelling section of the project involved creating low-fidelity wireframes to visualise the user interface design, conducting usability testing to gather feedback on the system's usability, incorporating the feedback through an alteration process, and finally developing high-fidelity prototypes. This section emphasised the importance of considering the end-users' perspective and incorporating user-centric design principles to create an intuitive and user-friendly interface. The usability testing results provided valuable insights into the strengths and weaknesses of the user interface, leading to iterative improvements and refinements. The high-fidelity prototypes showcased the final design of the user interface, providing a visual representation of the system's look and feel. Overall, the UX/UI modelling section played a crucial role in ensuring a user-centric approach and enhancing the overall user experience of the vehicle tracking system.

Limitations

As a sole individual working on such a complex task, I have found it quite challenging to manage multiple tasks simultaneously. Limitations in expertise or domain knowledge in areas such as computer vision, deep learning, and UX/UI design have posed challenges all the way through. Availability of hardware resources for training and testing the model, such as GPU capacity, was also limited, which impacted the efficiency and speed of the development process. Real-world testing scenarios were constrained due to factors such as time limitations, access to suitable technical resources, and obtaining accurate ground truth for evaluating our app's performance. Dealing with these constraints has provided valuable lessons for me and my personal growth as an inspiring software engineer. These challenges have served me well as learning opportunities, helped to refine project management skills and developed resilience in overcoming obstacles.

Conclusion

In conclusion, the Information Design and Development (IDD) phase of the project has been instrumental in laying the foundation for the successful implementation of the vehicle tracking app. Through the use of various design techniques, including UML diagrams, use case scenarios, and outlining collaborative development tools, I have made significant progress in shaping the direction and scope of the app.

The design phase has provided a clear understanding of the app's visual aesthetics, navigation flow, and functionality, enabling us to create intuitive user interfaces and seamless user experiences. The use of UML diagrams, such as class diagrams, sequence diagrams, and activity diagrams, has facilitated effective communication and documentation of the app's architecture and design patterns.

Additionally, the use case scenarios have helped to identify key user interactions and requirements. The outlined tools for development and collaborative development, such as YOLOv8, Flask, OpenCV, Roboflow, FiftyOne, Figma, and Lucidchart, have been carefully selected to streamline the development process. As we move forward with the development phase, the insights gained from the IDD phase will serve as a strong foundation for building a robust, user-friendly, and efficient vehicle tracking app. The comprehensive design approach, including the use of UML diagrams, use case scenarios, and collaborative development tools, will contribute to the successful delivery of the app, meeting the needs of the users and stakeholders.

References

AI. (2023). *Ultralytics | Revolutionizing the World of Vision AI*. Ultralytics.

<https://ultralytics.com/yolov8>

Blance, A. (2022, July 27). *Using AI to Generate 3D Models! | by Andrew Blance |*

Towards Data Science. Medium; Towards Data Science.

<https://towardsdatascience.com/using-ai-to-generate-3d-models-2634398c0799>

Computer Vision Datasets. (2017). *Computer Vision Datasets*. Roboflow.

<https://public.roboflow.com/>

Configuration - Ultralytics YOLOv8 Docs. (2023). Ultralytics.com.

<https://docs.ultralytics.com/usage/cfg/>

Dorfer, T. A. (2023, March 23). *Enhanced Object Detection: How To Effectively*

Implement YOLOv8. Medium; Towards Data Science.

<https://towardsdatascience.com/enhanced-object-detection-how-to-effectively-implement-yolov8-afd1bf6132ae>

Figma. (2023). Figma.

[https://www.figma.com/file/L0IIYHheLJONPPqJJoxQfk/Free-UX-Flow-3.0-\(FigmaJam\)-\(Community\)?node-id=13%3A28&t=xF9N83Std0z8M7ha-1](https://www.figma.com/file/L0IIYHheLJONPPqJJoxQfk/Free-UX-Flow-3.0-(FigmaJam)-(Community)?node-id=13%3A28&t=xF9N83Std0z8M7ha-1)

Jain, S. (2023, January 16). *YOLOv8 The new State Of The Art Detector? - The Modern*

Scientist - Medium. Medium; The Modern Scientist.

<https://medium.com/the-modern-scientist/yolov8-the-new-state-of-the-art-detector-89f627ad17aa>

LibGuides: APA Citation Style, Seventh Edition: Formatting Guidelines. (2022).

Ulethbridge.ca.

https://library.ulethbridge.ca/apa7style/formatting_guidelines#:~:text=Your%20essay%20should%20be%20typed,font%20consistently%20throughout%20the%20paper.

OpenMMLab. (2023a, January 17). *Dive into YOLOv8: How does this state-of-the-art model work?* Medium; Medium.

<https://openmmlab.medium.com/dive-into-yolov8-how-does-this-state-of-the-art-model-work-10f18f74bab1>

OpenMMLab. (2023b, January 17). *Dive into YOLOv8: How does this state-of-the-art model work?* Medium; Medium.

<https://openmmlab.medium.com/dive-into-yolov8-how-does-this-state-of-the-art-model-work-10f18f74bab1>

Overview - Roboflow. (2023). Roboflow.com. <https://docs.roboflow.com/>

OVIS | Occluded Video Instance Segmentation. (2022). Songbai.site.

<http://songbai.site/ovis/>

Photon. (2023, February 14). *"Exploring the Power of YOLOv8: Comprehensive Analysis of Object Detection, Classification, and Segmentation."* Medium; Medium.

<https://medium.com/@photon4273/revolutionizing-the-ability-of-yolov8-a-depth-insight-of-object-detection-classification-and-8c3801524176>

Qi, J., Gao, Y., Hu, Y., Wang, X., Liu, X., Bai, X., Belongie, S., Yuille, A., Torr, P. H. S., & Bai, S. (2022). Occluded Video Instance Segmentation: A Benchmark. *International Journal of Computer Vision*, 130(8), 2022–2039.

<https://doi.org/10.1007/s11263-022-01629-1>

YOLOv8 Detection 10x Faster With DeepSparse—Over 500 FPS on a CPU - Neural

Magic. (2023, January 18). Neural Magic - Software-Delivered AI.

<https://neuralmagic.com/blog/yolov8-detection-10x-faster-with-deepsparse-500-fps-on-a-cpu/>

Appendix

Data Collection

Data collection is an essential step in any machine learning project, including object detection and tracking using deep learning models. In this paper, the data used for training and testing the YOLO algorithm for vehicle tracking was obtained from the COCO dataset. The dataset contains over 330,000 images with more than 80 object categories, including cars and trucks, which were the selected classes for this project.

To download the COCO dataset with the selected classes of 'car' and 'truck,' I used the FiftyOne tool. FiftyOne is an open-source tool that provides an integrated interface in Jupyter Notebook for working with image and video datasets. It simplifies the process of downloading and managing large-scale datasets, making it an ideal tool for researchers working with deep learning models.

The process of collecting the data using FiftyOne involved the following steps:
Install FiftyOne tool: The first step was to install the FiftyOne tool on my computer. This involved downloading and installing the tool using the command line.

Download COCO dataset: The next step was to download the COCO dataset using the FiftyOne tool. To download the COCO dataset in FiftyOne I used the `fiftyone.zoo` module, which provides a collection of pre-built FiftyOne datasets. To download the COCO dataset, the `fiftyone.zoo.load_zoo_dataset()` method was used. Additionally, I used the filter to keep only cars and trucks in the dataset.

Preprocessing of the data: Once the COCO dataset was downloaded, I performed some preprocessing steps to prepare the data for training and testing the YOLOv8 model. The preprocessing steps involved resizing the images to a standard size of 416 x 416 pixels, converting them to the appropriate format (JPEG), and labelling the objects of interest using the FiftyOne labelling tool.

1. To resize the images, I used the `cv2.resize()` function from the OpenCV library. The function takes the input image and the desired output size as arguments and returns the resized image.
2. To convert the images to JPEG format, I used the PIL library. The library provides a `save()` method that allows you to save the image in various formats, including JPEG.
3. Finally, I used the FiftyOne labelling tool to label the objects of interest in the images.

Splitting the dataset: the COCO dataset is already split into training, validation, and testing sets. The dataset provides separate annotations for each set. The

training set is the largest and is typically used for training deep learning models, while the validation set is used for hyperparameter tuning and early stopping to prevent overfitting. The testing set is used to evaluate the performance of the trained model on unseen data.

In summary, data collection for this project involved using the FiftyOne tool to download the COCO dataset with the selected classes of 'car' and 'truck.' The tool simplified the process of downloading and managing large-scale datasets, allowing the researcher to focus on other aspects of the project, such as data preprocessing, model training, and evaluation.

YOLOv8 backbone diagram

To better understand the inner workings of the YOLOv8 model, we have included a diagram of the model below. The diagram was sourced from the original Ultralytics GitHub repository. (see Figure A1)



Figure A1.

YOLOv8 architecture diagram

Pretrained model prediction


Pretrained YOLO model prediction	Actual picture (COCO-17 dataset)
<pre>from ultralytics import YOLO model=YOLO("yolov8n-seg.pt") model.predict(source="images/000000000748.jpg")</pre> <p>Seg: 566660 1 person, 1 broccoli, 2 carrots, 1 chair, 1 couch, 1 potted p</p>	

Figure A2.

Pretrained YOLOv8n prediction on COCO dataset image