WIP 302

Performance Analysis of YOLOv6, YOLOv7, YOLOv8, and NAS for Object Detection

Irina Getman

Bachelor of Software Engineering, Yoobee Colleges

Supervisory team: Dr. Mohammad Norouzifard

September 22, 2023

Contents

0.1 Introduction

The field of computer vision, strengthened by the rapid advancements in deep learning, has transformed our ability to detect and track objects in diverse scenarios. Object detection, especially in challenging environments, has become a focal point due to its wide range of applications and the critical nature of accurate detection. Among the leading-edge models in this domain is the YOLO (You Only Look Once) series, which has consistently demonstrated superior performance in real-time object detection tasks.

This report commences with a presentation of a pipeline diagram, offering a bird's-eye view of the integral steps and components of our object detection system, which leverages various YOLO models, including YOLOv6, YOLOv7, YOLOv8, and NAS. This diagram clearly shows how data moves and is processed, from start to finish. By using this visual, we want to easily explain the system's complex structure and how it works, adding to the detailed written explanation that comes next.

A comprehensive enumeration of the development tools, programming languages, and technologies used for this project is also provided. This is followed by an in-depth discussion on software prerequisites, both functional and non-functional, supplemented by illustrative use case diagrams, class diagrams, and activity charts.

The UX/UI modeling segment of the report offers a holistic overview of the user experience (UX) and user interface (UI) design for our object detection application, developed using Streamlit. Streamlit, renowned for its simplicity and efficiency, facilitates the swift creation of interactive web applications, making it an invaluable asset for our endeavor.

Feedback from usability testing forms the cornerstone of our iterative design process, guiding the evolution of our initial designs to better cater to user needs. This iterative approach underscores our commitment to delivering a product that resonates with the end-users, both in terms of functionality and user-friendliness.

In short, this report gives a detailed look at object detection using different YOLO models. It highlights the importance of UX/UI design, including initial sketches, user feedback, design changes, and final prototypes. The Software Requirements Specification (SRS) section provides essential guidelines for building and launching the system. The report ends by discussing potential improvements, challenges faced, and the progress made in object detection.

0.2 Software Process Model Selection and Tools

A pipeline diagram of the "Comparative Evaluation of YOLO Models for Advanced Object Detection" deep learning-based project (refer to Figure 1), visually represents the different stages or steps involved in our project's workflow.

1. **Data Collection:** This stage involves gathering the necessary data for vehicle tracking tasks. This task includes video or image datasets contain-

ing 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 for training and evaluation.

- Tools/frameworks: 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, Roboflow.

3. **Model Training:** This stage involves training various YOLO models, including YOLOv6, YOLOv7, YOLOv8, and NAS, using the preprocessed data. This stage may include tasks such as configuring the YOLO model architecture, setting hyperparameters, and training the model using GPU-accelerated hardware.

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

4. **Model Evaluation:** Once the YOLO models are trained, they need to be evaluated to measure their 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 analyzing the results to assess the model's accuracy and robustness in challenging scenarios.

   - Tools/frameworks: Darknet, 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 visualizing the tracked vehicles on the output frames or images.

   - Tools/frameworks: Darknet, 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 visualizations or outputs to visualize 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 visualizations 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 analyzed. 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: Streamlit, SQLite, Snowflake.

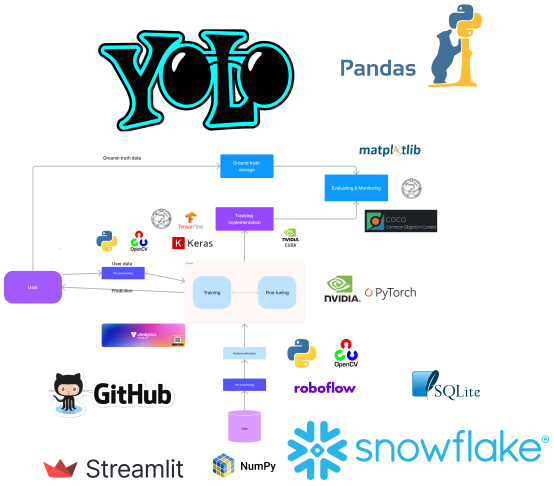


Figure 1: Pipeline Diagram of Vehicle Tracking in Challenging Scenarios Using Various YOLO Models

[1]

[1]The detailed diagram can be accessed at https://www.figma.com/file/Ww9SKdfCnHvCT1dX68qxuz/302-Capstone?type=design&node-id=0%3A1&mode=design&t=lcC9mVee0OsqgNQi-1.

0.2.1 Tools and Technologies

The following is a list of tools, IDEs, programming languages, technologies, and frameworks selected for the project, along with justifications for their selection:

- **IDE - Visual Studio Code**: Versatile, supports multiple languages, and has a rich ecosystem of extensions.
- **Programming Language - Python**: Chosen for its simplicity, extensive libraries, and suitability for web and AI development.
- **Framework - Streamlit**: Enables rapid development of interactive web applications, especially for data applications.
- **Backend API - FastAPI**: Modern, fast web framework for building APIs with Python.
- **Collaborative Tool - GitHub**: For version control, collaboration, and continuous integration.
- **Database - SQLite**: Lightweight, serverless, and suitable for the project's scale.

0.2.2 Advanced Features in Streamlit

Secrets Management

Streamlit's secrets management system allows developers to securely store sensitive information, such as API keys, database URIs, and passwords. This feature ensures that sensitive data remains confidential and is not exposed in the application's source code.

- **Storing Secrets:** Secrets can be added to the Streamlit application through the 'secrets.toml' file. This file should never be committed to version control to ensure the confidentiality of the data.
- **Accessing Secrets:** Within the Streamlit app, secrets can be accessed using the 'st.secrets' command. For instance, to access an API key, one would use 'st.secrets["api_key"]'.
- **Environment-specific Secrets:** Streamlit supports environment-specific secrets, allowing developers to have different sets of secrets for development, staging, and production environments.

Connecting to Data

Streamlit offers a variety of methods to connect to data sources, ensuring flexibility and efficiency in data-driven applications (Streamlit, 2023).

- **Databases:** Streamlit can connect to various databases, including PostgreSQL, MySQL, and SQLite. The connection often requires a database URI, which can be securely stored using Streamlit's secrets management.
- **Cloud Storage:** Streamlit supports connections to cloud storage solutions such as AWS S3, Google Cloud Storage, and Azure Blob Storage. This allows for the retrieval and display of data stored in the cloud.
- **Caching:** To improve performance, Streamlit provides a caching mechanism. By using the '@st.cache' decorator, developers can ensure that data loading and processing operations are cached, reducing redundant operations and speeding up the app.

0.3 System Requirements

In developing our YOLO model evaluation application, understanding both functional and non-functional requirements is essential. These requirements outline the system's capabilities and features, ensuring it meets user needs and maintains consistent performance and reliability. This section covers these requirements, highlighting user interactions, system performance standards, and security aspects. We then move on to discuss use case scenarios based on these requirements, followed by class and activity diagrams.

0.3.1 Functional Requirements

1. **Evaluate Model:**
    - User uploads a video/image.
    - User selects a YOLO model for evaluation.
    - System processes the uploaded content using the selected model.
    - System returns the evaluation results.
2. **View Results:**
    - User views the results of the evaluation.
    - Results include detected objects, their coordinates, and other relevant metrics.
3. **View Graphs:**
    - User accesses graphical representations of the evaluation data.
    - Graphs include performance metrics, object counts, and other relevant visualizations.
4. **User Registration and Authentication:**
    - New users can create an account.

- Existing users can log into the application.

5. **Access Dashboard:**
   - Authenticated users can access their personalized dashboard.
   - Dashboard displays user-specific evaluations, saved queries, and other personalized content.
6. **Manipulate Data:**
   - Users can filter, sort, or export data from their dashboard.
7. **View Model Info:**
   - Users can access detailed information about a specific YOLO model.
   - Information includes model architecture, training data, performance metrics, and other technical details.
8. **Provide Feedback:**
   - Users can submit feedback on a model evaluation.
9. **Receive Notification:**
   - Users receive a notification when an evaluation is complete or when there's relevant information to share.
10. **Custom Queries (For Data Analysts):**
    - Data analysts can run custom queries on the data for specific insights.
    - System provides a safe and secure interface for inputting and processing these queries.

0.3.2 Non-Functional Requirements

1. **Usability:**
   - The system should be user-friendly, with intuitive navigation and clear instructions.
   - Annotations & Tooltips: Offer explanations for each metric, especially for complex ones.
2. **Performance:**
   - The system should process evaluations in a reasonable time frame.
   - Custom queries should be optimized for performance, with feedback provided for long-running queries.
3. **Security:**

- User data, including uploaded content and personal information, should be stored securely.
- Custom queries should be sanitized and validated rigorously to prevent SQL injection or other malicious attacks.

4. **Scalability:**
   - The system should be able to handle a large number of users and evaluations.
   - It should be designed to scale up as the user base grows.
5. **Reliability:**
   - The system should have a high uptime, with minimal outages or disruptions.
   - There should be mechanisms in place for data backup and recovery.
6. **Concerns about Custom Queries:**
   - Allowing custom queries can introduce security and performance risks.
   - There should be mechanisms in place to ensure that queries are safe and do not negatively impact system performance.

0.3.3 Use Case Scenarios

In software development, use case scenarios are fundamental in illustrating how users interact with the system. They provide a clear overview of the system's functions from the user's perspective. The following tables present the main use cases for our application, detailing interactions for both general and registered users. Each use case includes a brief description and a series of user-system interactions. This structured presentation helps readers grasp the core functions of the application and how users engage with it.

General User

The general user's interactions with the system encompass a range of functionalities, from model evaluation and data retrieval to the graphical representation of results, as illustrated in Table 1.

Registered User

For registered users, the system offers advanced functionalities, including user registration, data manipulation, and the ability to run custom queries. These interactions are detailed in Table 2.

Use Case	**Description**	**Interactions**
Model Evaluation	Allows the user to select a YOLO model and upload a video or image for evaluation (as shown in Figure 2).	1. User selects a model. 2. User uploads video/image. 3. System evaluates and displays results.
Data Retrieval	User can retrieve the results of the evaluation, including coordinates and number of detected objects (as shown in Figure 3).	1. User requests data. 2. System displays results.
Graphical Representation	Provides a visual representation of the evaluation results (as shown in Figure 4).	1. User requests graphs. 2. System generates and displays graphs.
Model Information Retrieval	Users can access detailed information about a specific YOLO model (as shown in Figure 5).	1. User selects a model. 2. System displays model details.
Feedback Submission	Users can provide feedback on the application (as shown in Figure 6).	1. User submits feedback. 2. System acknowledges receipt.

Table 1: Use Case Scenarios for General User

Use Case	**Description**	**Interactions**
User Registration	Allows a new user to create an account (as shown in Figure 7).	1. User provides registration details. 2. System creates account.
User Login	User can log into their account (as shown in Figure 8).	1. User provides login credentials. 2. System authenticates user.
Data Manipulation	Registered users can manipulate their past evaluations (as shown in Figure 9).	1. User accesses dashboard. 2. User filters/sorts data.
Access Dashboard	Authenticated user accesses their personalized dashboard (as shown in Figure 10).	1. User logs in. 2. System displays dashboard.
Custom Queries	User can run custom queries on the data (as shown in Figure 11).	1. User inputs query. 2. System displays query results.

Table 2: Use Case Scenarios for Registered User

0.3.4 Use Case Diagrams

In software design, use case diagrams are essential tools that give an overview of the system's functions and how users interact with it. These diagrams visually represent the system's operations and the roles users play. The upcoming diagrams offer a clear understanding of how users engage with the system, highlighting its main features and design.

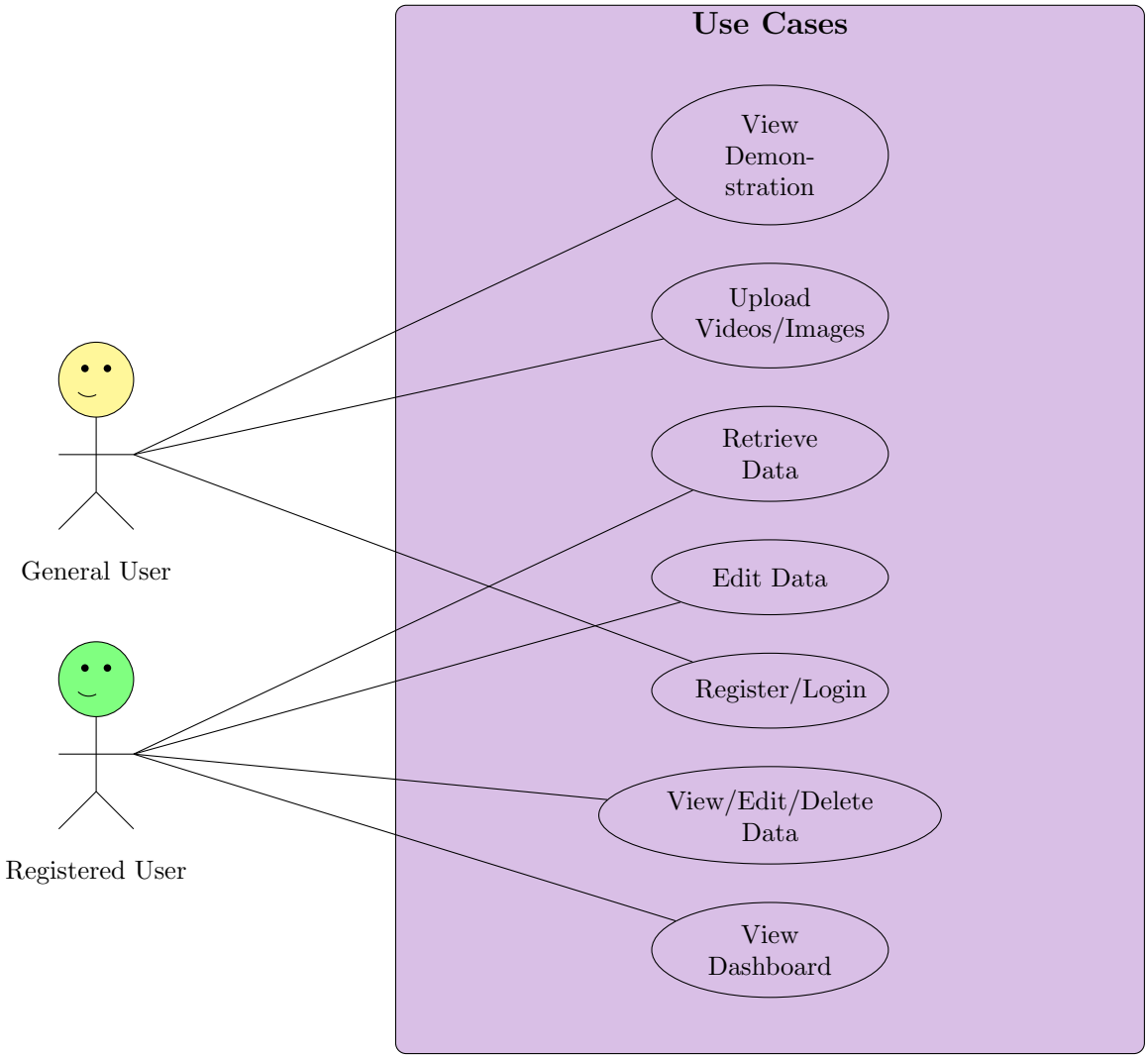


Figure 2: User Interaction with YOLO models evaluation system

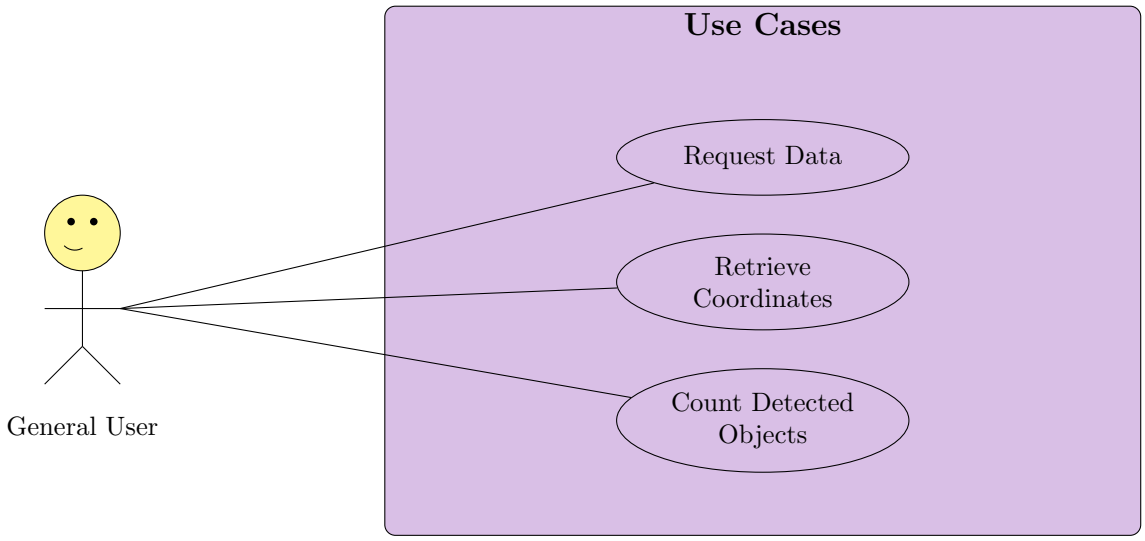


Figure 3: Data Retrieval Use Case

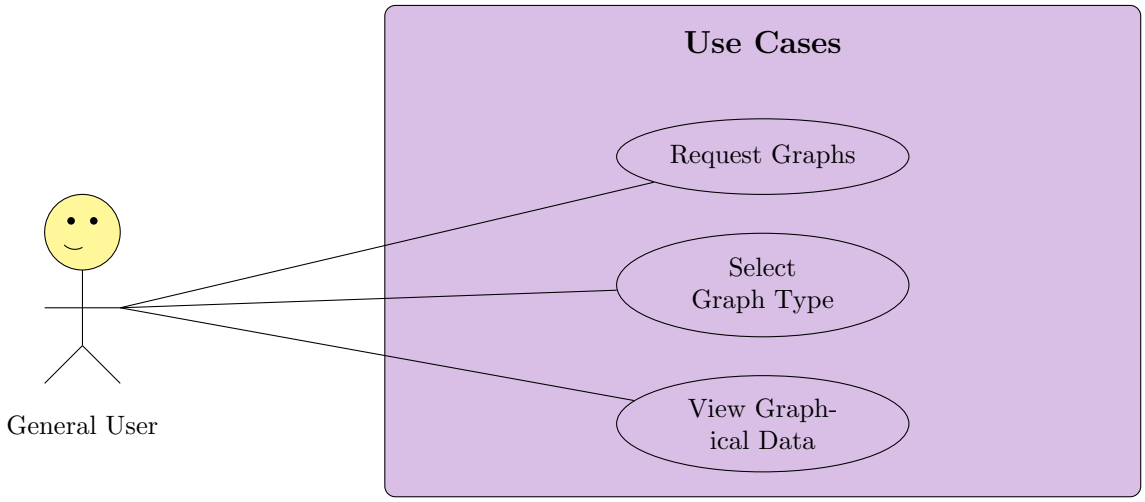


Figure 4: Graphical Representation Use Case

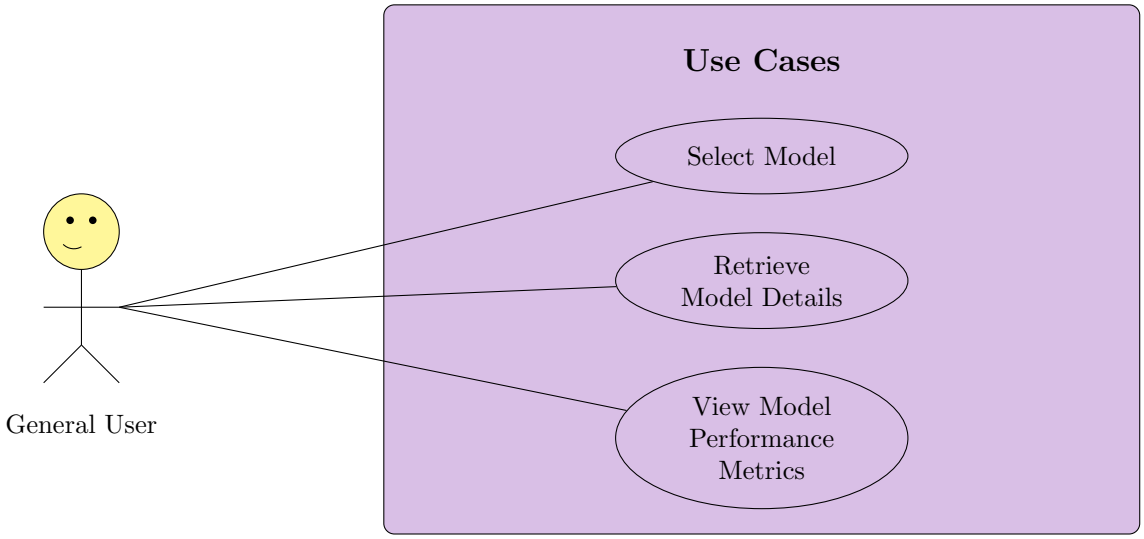


Figure 5: Model Information Retrieval Use Case

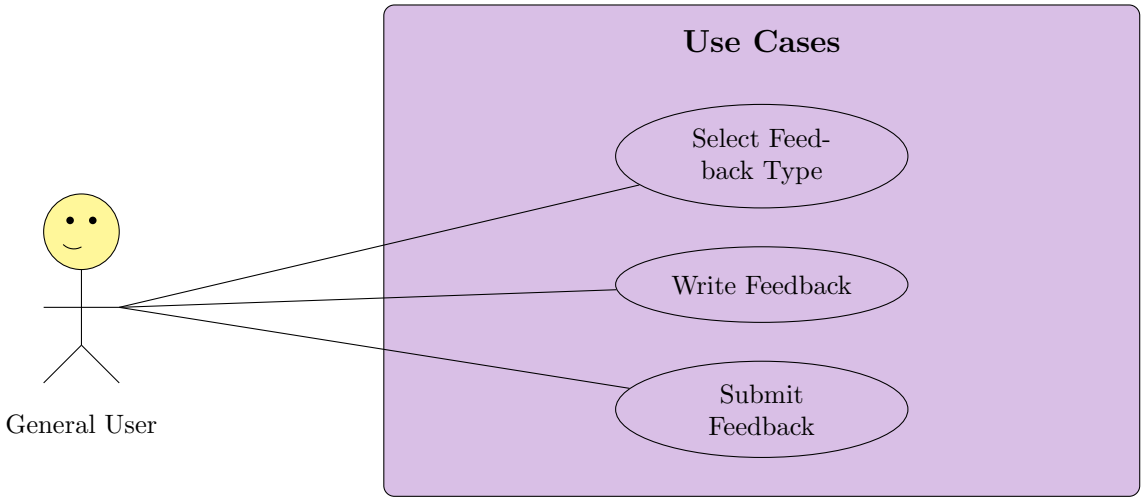


Figure 6: Feedback Submission Use Case

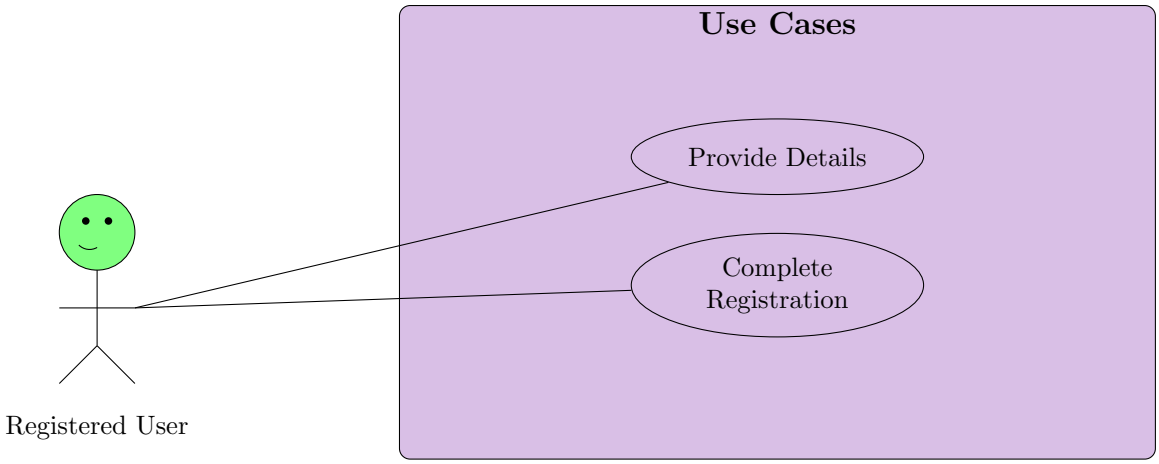


Figure 7: User Registration Use Case

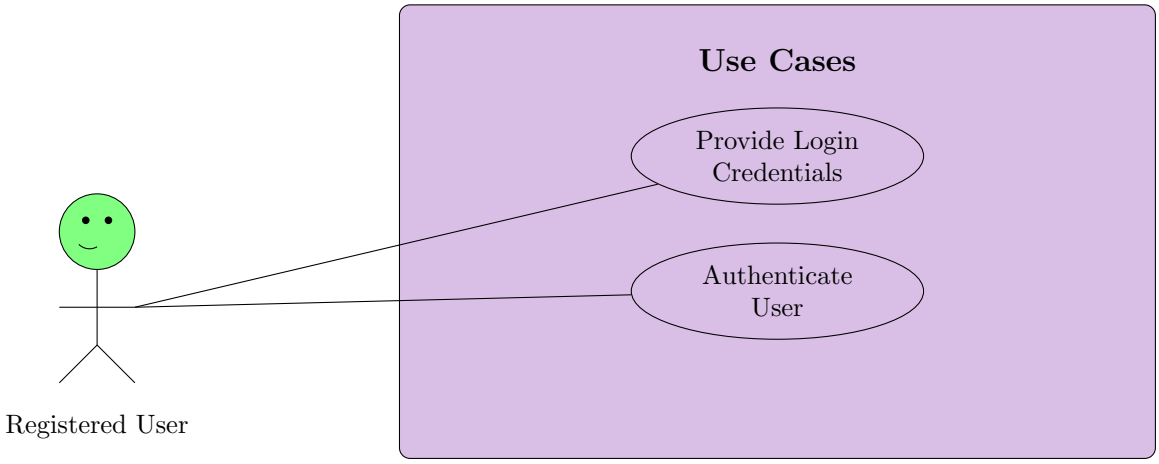


Figure 8: User Login Use Case

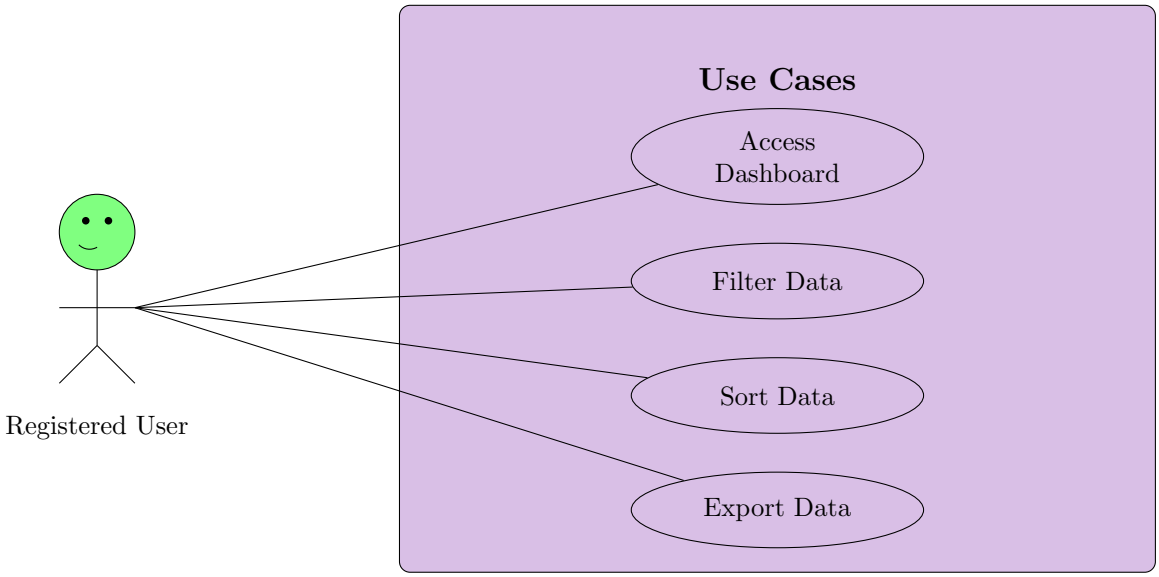


Figure 9: Data Manipulation Use Case

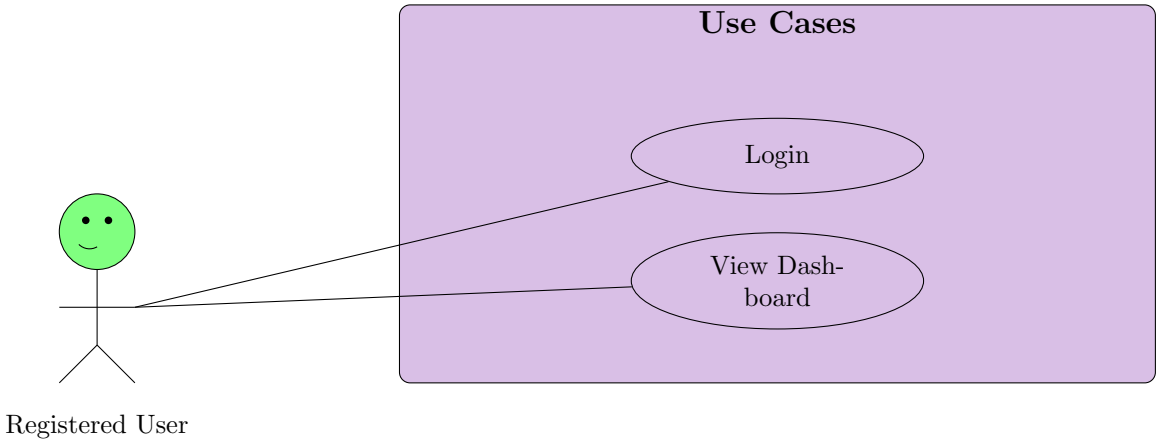


Figure 10: Access Dashboard Use Case

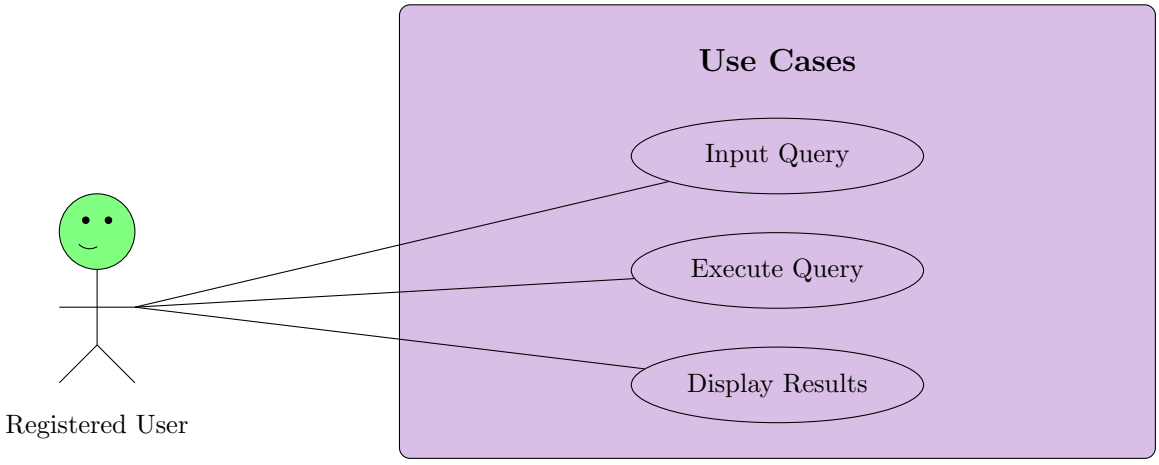


Figure 11: Custom Queries Use Case

0.3.5 Class Diagram

The class diagram provides a structured visualization of the YOLO Model Evaluation App, capturing its primary entities and the relationships between them. This diagram serves as a blueprint for understanding the system's architecture, the data it manages, and the interactions that occur within it. The classes, their attributes, and the relationships between them are detailed as follows:

1. **User Class**:
   - *Attributes*:
     - `userID`: A unique identifier for each user.
     - `username`: The name the user uses to log into the system.
     - `password`: The user's password for authentication.
   - *Relationships*:
     - The `User` class has an association relationship with the `Evaluation` class, indicating that a user can use multiple evaluations. This is represented by the "uses" association.
     - The `User` class also has an association with the `Feedback` class, suggesting that a user can submit multiple feedbacks. This is denoted by the "submits" association.
     - Additionally, the `User` class is associated with the `Dashboard` class, signifying that a user views one dashboard. This is depicted by the "views" association.
2. **Evaluation Class**:
   - *Attributes*:
     - `evalID`: A unique identifier for each evaluation.
     - `video/image`: The file (either video or image) that the user uploads for evaluation.
     - `results`: The results produced after the evaluation.
   - *Relationships*:
     - The `Evaluation` class has an association with the `YOLOModel` class, indicating that an evaluation uses a specific YOLO model. This is represented by the "uses" association.
     - The `Evaluation` class has a composition relationship with the `Results` class, suggesting that an evaluation produces results, and those results cannot exist without the evaluation. This is denoted by the "produces" association.
3. **YOLOModel Class**:
   - *Attributes*:
     - `modelID`: A unique identifier for each YOLO model.

- `modelName`: The name of the YOLO model.
- `version`: The version of the YOLO model.

4. **Results Class**:
   - *Attributes*:
     - `coordinates`: A list containing the coordinates of detected objects.
     - `objectCount`: The total number of objects detected.
5. **Feedback Class**:
   - *Attributes*:
     - `feedbackID`: A unique identifier for each feedback.
     - `content`: The content of the feedback submitted by the user.
6. **Dashboard Class**:
   - *Attributes*:
     - `dashboardID`: A unique identifier for each dashboard.
     - `evaluations`: A list containing evaluations viewed on the dashboard.
   - *Relationships*:
     - The `Dashboard` class has an aggregation relationship with the `Evaluation` class, suggesting that a dashboard contains multiple evaluations, but evaluations can exist without the dashboard. This is depicted by the "contains" association.

In summary, the class diagram (as shown in Figure 12), offers a clear representation of the system's classes and their interrelationships, providing insights into the app's design and functionality.

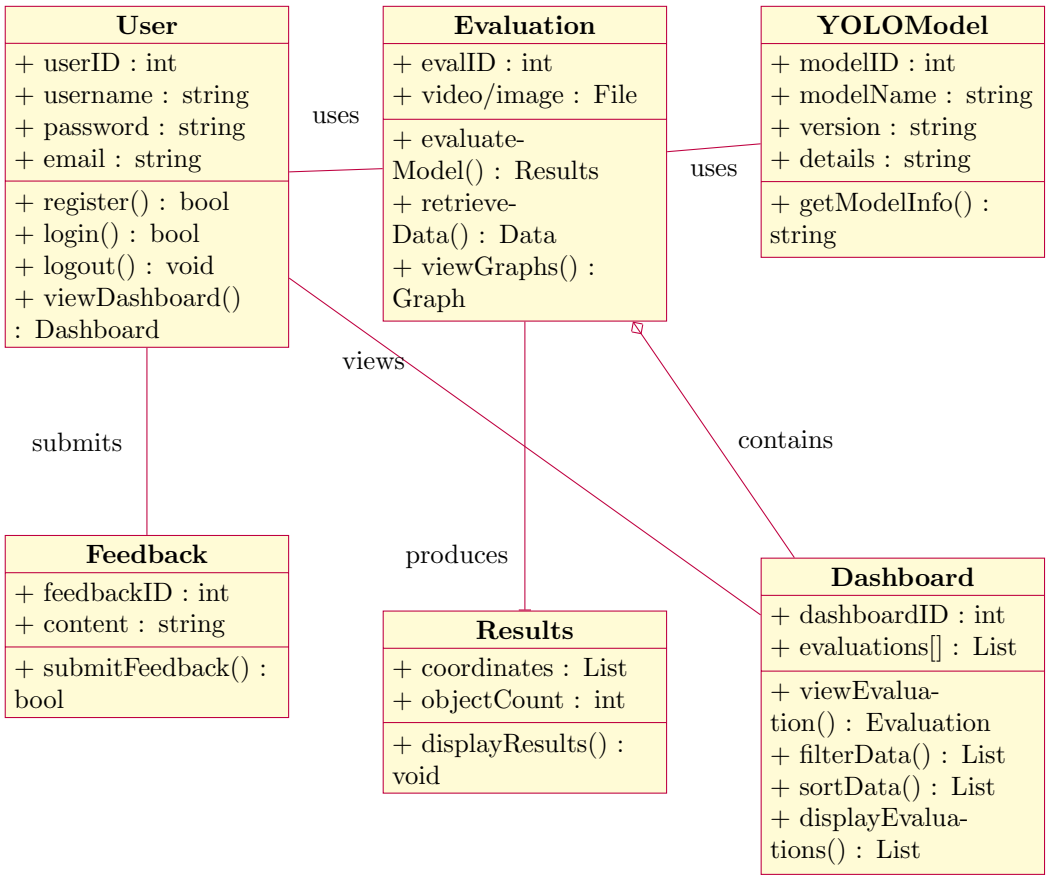


Figure 12: Class Diagram for YOLO Model Evaluation App

0.3.6 Activity Diagrams

The activity diagram (refer to Figure 13) showcases the system's journey. It starts with **Initializations** in a main flow **Object Detection**. The system initiates the list of models for users and sets up the interface. Users can then pick a model and upload their data.

In the **User Interaction** section, users choose a model from a list and upload their data. Advanced users can tweak detection settings for a custom experience.

The **Authentication** part ensures only authorized users access certain features. Users log in, and if there's an issue, they get an error. New users can sign up, their data will be validated and an error is displayed when the validation process fails. The new user's data will be stored, and a verification email will be sent to the new user.

Once data is uploaded, it's *reprocessed*. Videos are stabilized, and images

are resized. The system checks lighting and improves it if needed. Then, it identifies key areas for detection by producing bounding boxes for detection and the processed data sent for object detection.

The main action happens in **object detection**. The system identifies and classifies objects. Each detection gets a confidence score,

Finally, in **post-processing**, the system refines and displays results. Users see clear visuals like bounding boxes. Results are stored for future use, and users can give feedback.

This flow ensures a smooth vehicle detection experience.

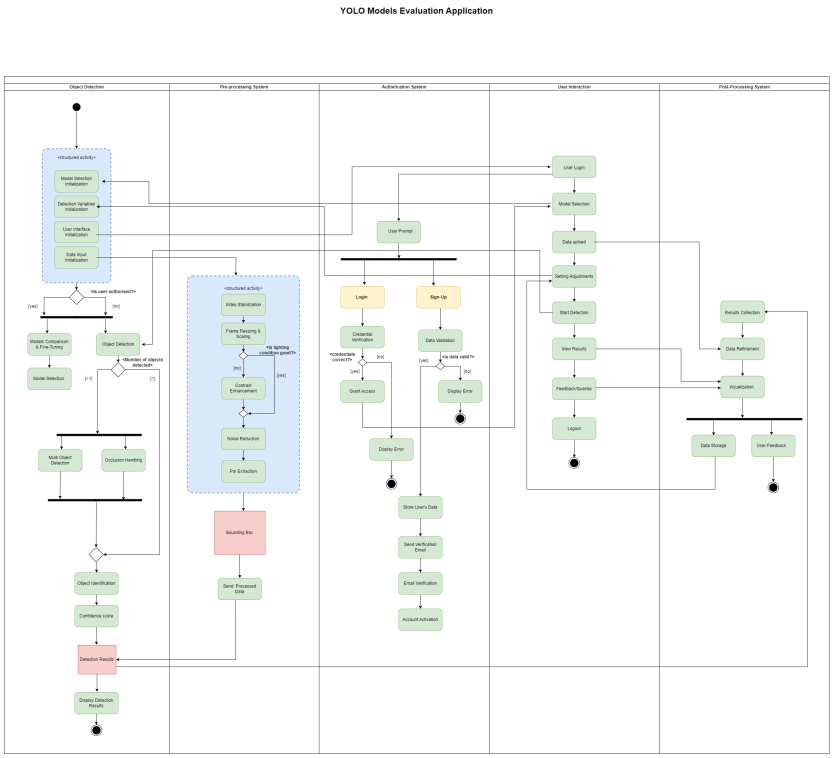


Figure 13: Activity Diagram of Evaluation App of Yolo models

[2]

0.4 UX/UI Modeling

As we transition from understanding the system's requirements, it becomes crucial to visualize how users will interact with the application. The *UX/UI*

[2]The detailed diagram can be accessed at https://www.figma.com/file/Ww9SKdfCnHvCT1dX68qxuz/302-Capstone?type=design&node-id=0%3A1&mode=design&t=lcC9mVee0OsqgNQi-1.

Modeling section goes into the design considerations, layout, and aesthetics of the application, ensuring that the user interface is intuitive, engaging, and user-friendly. This section will shed light on wireframes, mockups, and interactive prototypes, painting a vivid picture of the user's journey within the application. In essence, the system requirements underscored the importance of a seamless user experience and robust performance. It set the foundation for the application's development, ensuring that all stakeholders have a clear understanding of the system's objectives and constraints.

0.4.1 Streamlit Design System

Streamlit offers a range of components that make it a powerful tool for rapid application development, especially for data and Machine Learning applications. Here are some of the primary components that we carefully selected to be implemented in our application:

- **Status Elements**: Streamlit provides a set of elements that allow developers to display the status of operations or tasks. These elements can be particularly useful when you're running computations that take some time, and we want to give feedback to the user about the progress or status of that computation: `st.progress`, `st.warning`, `st. error`, `st.info`, `st.success`, and `st.spinner` (As shown in Figure 14).
- **Container Elements**: they allow developers to structure their apps in a more organized manner.
  - `st.expander`: This creates an expandable container that users can click on to show or hide its content. It's useful for hiding advanced configurations or detailed explanations.
  - `st.sidebar`: This is a special container for placing widgets that should appear in the app's sidebar instead of the main page.
  - `st.form` container in Streamlit is a special container designed to hold input widgets and a submit button. When the submit button is pressed, all the input widgets inside the form are processed at once. This is particularly useful when you want to collect multiple inputs from a user and then perform an action (like a computation or a database query) only once after the user has finished providing all the inputs. Refer to Figure 15.
- **Chart Elements**.

  Streamlit offers a variety of chart elements that allow developers to visualize data in an intuitive and interactive manner. These chart elements are designed to integrate seamlessly with popular Python data visualization libraries, ensuring flexibility and ease of use.

- `st.bar_chart`: This element provides a simple way to create bar charts directly from data frames or series. It's especially useful for categorical data comparison.
- `st.line_chart`: Ideal for visualizing time series or any sequential data, this element offers a straightforward method to generate line charts.
- `st.plotly_chart`: Streamlit supports Plotly, a popular interactive graphing library. With this element, we can embed Plotly figures and charts directly into the apps.
- `st.pyplot`:Streamlit provides an element to display figures created using the `pyplot` module, ensuring compatibility with a wide range of custom visualizations.

These chart elements, combined with Streamlit's interactive widgets, enable developers to create dynamic data-driven applications with minimal effort. Refer to Figure 16

- **Text Elements**:

  Streamlit provides a rich set of text elements to allow developers to present information in a structured and readable manner. These elements cater to different levels of content hierarchy, ensuring that the information is presented clearly.

  - `st.write`: A versatile function that accepts a wide range of input types, including strings, data frames, and even Markdown. It's the go-to function for general-purpose content display.
  - `st.title`: Used to display a prominent title on the app page. It's the largest text element and is typically used for the main heading of the application.
  - `st.header`: A sub-level to the title, this function is used to display section headings, making content organization clearer.
  - `st.subheader`: As the name suggests, this is a sub-level to the header. It's useful for subsection headings or to highlight smaller content divisions.
  - `st.text`: A straightforward function to display plain text. It doesn't support Markdown or other special formatting, ensuring the content remains unstyled.
  - `st.caption`: Ideal for adding small notes or captions, typically under images or charts. The text is smaller and subtler compared to other elements.
  - `st.code`: Designed to display code snippets. It accepts a string of code and an optional language parameter, ensuring syntax highlighting for better readability.

- **Input Widgets**:

  Streamlit offers a variety of input widgets that enable user interaction and data input in a user-friendly manner. These widgets are designed to be intuitive, ensuring that users can easily provide inputs without any hassle.

  - `st.file_uploader`: Allows users to upload a file to the Streamlit app. It supports various file types and can be customized to accept specific formats, making data input seamless.
  - `st.number_input`: A widget that lets users input a number. It can be tailored to accept integers, floats, and can even have defined min-max ranges.
  - `st.selectbox`: Presents a dropdown menu for users to choose from a list of options. It's ideal for scenarios where there's a predefined set of choices, such as YOLO models in our case.
  - `st.multiselect`: An extension of the selectbox, this widget allows users to select multiple options from a dropdown list, providing flexibility in data selection. We will be using this widget for model comparison.
  - `st.text_area`: Enables users to input multiline text. It's useful for scenarios where detailed input, like feedback or descriptions, is required.
  - `st.slider`: A draggable slider widget. It can be used to select a value or a range from a continuous set of options, making it perfect for settings like confidence scores.

- **Data Display Elements**:

  Streamlit provides a range of elements to visually represent and display data in a clear and effective manner. One of the notable elements for data representation is:

  - `st.metrics`: This widget is designed to showcase key metrics or statistics in a visually appealing manner. It presents the metric's name, value, and an optional delta that indicates the change or difference from a previous value. The delta can be color-coded to signify positive or negative changes, making it easier for users to quickly grasp the significance of the metric.

  The `st.metrics` widget is particularly useful for dashboards or applications where it's essential to highlight specific data points or KPIs. By using this widget, we can ensure that crucial information stands out and is easily noticeable by the users.

Streamlit's design system is a harmonious blend of functionality and aesthetics. The platform employs a mix of serif and sans-serif fonts, striking a

(a) Progress (b) Warning (c) Error (d) Info

(e) Success

Figure 14: Streamlit's Status Elements: Progress, Warning, Error, Info, and Success.

balance between modernity and readability. This typographical choice ensures that information is not only presented with clarity but also with a structured hierarchy. One of the standout features of Streamlit is its unwavering commitment to design consistency. Regardless of whether a user interacts with a widget, views a chart, or engages with media, the design remains uniform and intuitive. Moreover, this consistency extends across devices, with Streamlit apps being responsive and ensuring an optimal viewing experience on both desktop and mobile platforms. In essence, Streamlit's design approach showcases its dedication to user-centric design, prioritizing both form and function.

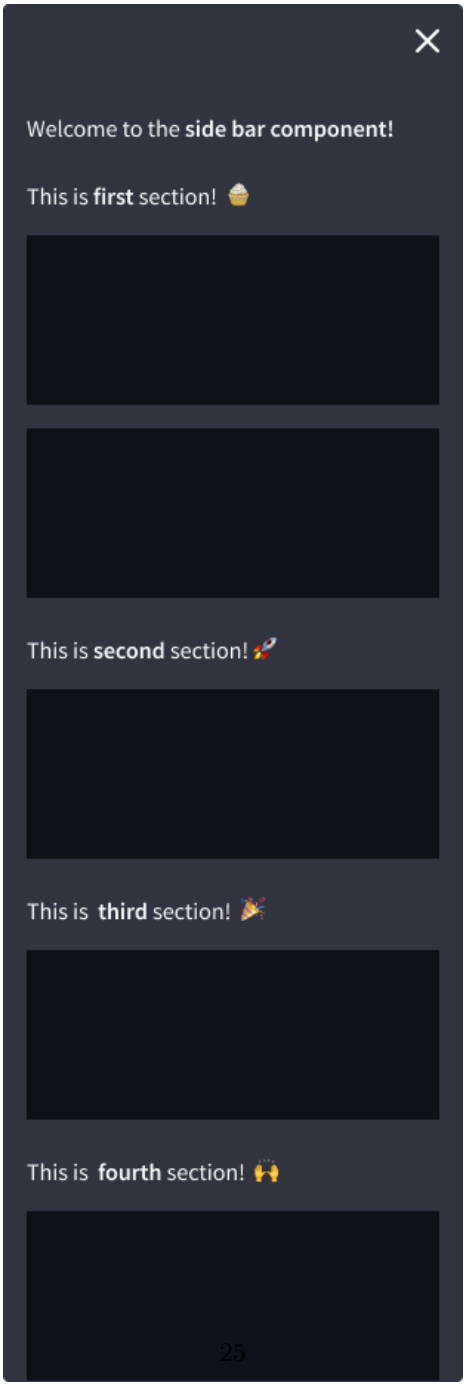
...
3

[3]The complete Design system can be accessed at https://www.figma.com/file/Ww9SKdfCnHvCT1dX68qxuz/302-Capstone?type=design&node-id=0%3A1&mode=design&t=lcC9mVee0OsqgNQi-1.

(a) Form

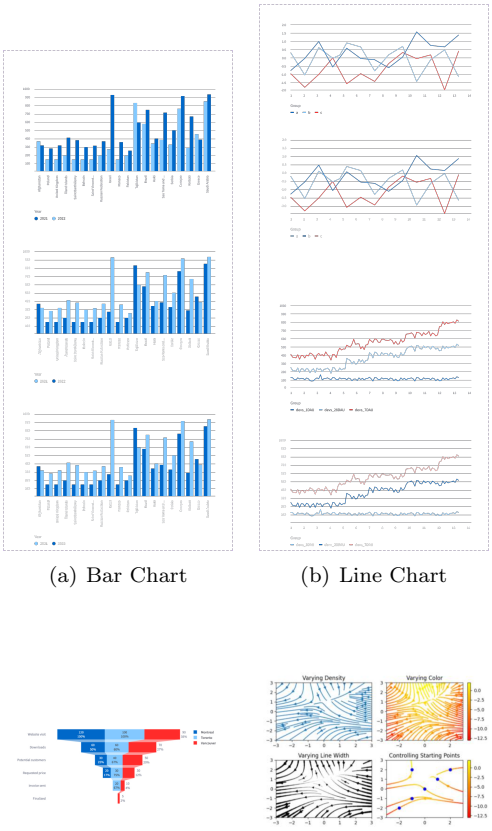
(b) Expander

(c) Expander



(d) Sidebar

Figure 15: Streamlit's Container Elements: Expanders, Form and Sidebar.



(a) Bar Chart (b) Line Chart

(c) Plotly Chart (d) Pyplot Chart

Figure 16: Streamlit's Chart Elements: Bar Chart, Line Chart, Plotly Chart, and Pyplot.

0.4.2 Lo-Fi Design

Initial Design Proposal

In the early design phase, we envisioned a system that seamlessly integrates with Streamlit's design elements:

- **Main Interface:** Centered around object detection, this was the primary interaction point for users.
- **Sidebar:** An auxiliary space for additional settings and options.
- **Model Selection:** Using 'st.selectbox', users could choose from a list of available detection models.
- **Data Input:** The 'st.file_uploader' allowed users to easily upload images or videos for processing.
- **Visualization:** Basic visualization tools like 'st.bar_chart' were integrated to present detection results.
- **Textual Displays:** Information and results were presented using 'st.write' and other textual elements.

User Testing 1

To ensure the efficacy and user-friendliness of our preliminary design, we embarked on a user testing session. The participants were presented with the following inquiries:

1. How intuitive did you find the main interface for object detection?
2. Were you able to easily locate and use the sidebar for additional settings?
3. How straightforward was the model selection process using the dropdown menu?
4. Did you encounter any challenges while uploading data using the file uploader?
5. Were the visualizations clear and informative for understanding the detection results?
6. How did you find the textual displays? Were they clear and easy to understand?
7. Was there any feature you felt was missing or could be improved upon?
8. How would you rate the overall user experience on a scale of 1 to 10?
9. Were there any elements that you found redundant or unnecessary?
10. Would you prefer more interactive visualizations, or are the current ones sufficient?

User Testing 1: Responses

Here are some valuable responses we received from our users.

User 1:

1. The main interface was intuitive, but the model selection dropdown was initially overlooked.
2. The data input system was straightforward to use.
3. The interface was clear, but additional visual feedback after making a selection would be beneficial.
4. The design is effective, but could benefit from enhanced visual cues.
5. The visualizations were clear and provided a good understanding of the detection results.
6. Textual displays were clear and easy to understand.
7. A tutorial or walkthrough might be a good addition for first-time users.
8. 8
9. No, all elements seemed necessary.
10. The current visualizations are sufficient, but interactivity might be a plus for advanced users.

User 2:

1. The sidebar was a highlight, keeping the main interface clutter-free.
2. Uploading data was simple, but a confirmation of successful upload would be helpful.
3. The main interface was user-centric, but tooltips or help icons might be a valuable addition.
4. Overall, the design is efficient, but more feedback mechanisms could elevate the user experience.
5. The visualizations were straightforward and informative.
6. Textual displays were concise and to the point.
7. Maybe a "help" or "FAQ" section could be added.
8. 7
9. No, all components were essential.
10. Interactive visualizations might be a good addition for a more in-depth analysis.

User 3:

1. The main interface was intuitive and easy to navigate.
2. The sidebar was initially a bit overwhelming, but after a few interactions, it became clear.
3. The model selection dropdown was a nice touch, making the process straightforward.
4. Uploading data was clear, but a confirmation message would be reassuring.
5. The visualizations provided a clear understanding of the detection results.
6. Textual displays were clear and easy to understand.
7. A history or log of past uploads might be useful.
8. I'd rate the overall user experience as 8 out of 10.
9. No, all elements seemed necessary and added to the experience.
10. The current visualizations are good, but some interactivity might be beneficial for detailed analysis.

User 4:

1. The main interface was clean and well-organized.
2. The sidebar was well-organized, making secondary options easy to find.
3. The model selection process was seamless.
4. Uploading data was easy, especially with the progress bar.
5. The visualizations were clear and concise.
6. Textual displays were well-structured and informative.
7. Grouping related functions or features might be beneficial.
8. I'd rate the overall user experience as 9 out of 10.
9. No, all elements were essential.
10. While the current visualizations are sufficient, some interactivity might be a good addition.

User 5:

1. The main interface was clear and met expectations.
2. The sidebar was intuitive, but icons next to options might enhance visual appeal.

3. The model selection dropdown was easy to use.
4. Data input was straightforward, but a confirmation message post-upload would be helpful.
5. The visualizations were clear and added value to the results.
6. Textual displays were clear and easy to understand.
7. Perhaps a "recently viewed" or "history" feature might be a good addition.
8. I'd rate the overall user experience as 7.5 out of 10.
9. No, all elements seemed necessary.
10. The current visualizations are good, but adding some interactivity might be beneficial.

Feedback from this session was instrumental in guiding our subsequent design iterations.

Design Iterations

Based on the feedback from the initial user testing, we made several enhancements:

- **Enhanced Selection:** We introduced 'st.multiselect' to allow users to choose YOLO models for comparison according to User's case.
- **Feedback on User data upload:** Once the upload is complete, we will display a clear confirmation message like "Your data has been successfully uploaded!"

User Testing 2: Post Iterations

After implementing the design enhancements, we conducted a second round of usability testing to evaluate the improvements and gather further feedback. The primary focus was on the newly introduced features and the overall user experience.

Questions for the Second Usability Testing:

1. How intuitive did you find the multi-selection feature for choosing YOLO models?
2. Were you able to easily understand and use the 'st.multiselect' feature?
3. Did you receive a clear confirmation after uploading your data? Was it reassuring?
4. How would you rate the improvements made to the design based on your previous feedback?

5. Are there any additional features or changes you would suggest after these iterations?

6. How would you rate the overall user experience now, on a scale of 1 to 10?

7. Were there any elements that you found redundant or unnecessary after the changes?

8. Did the new features enhance your interaction and understanding of the app?

9. Were the textual displays and feedback messages clear and easy to understand?

10. Would you prefer more interactive visualizations, or are the current ones still sufficient?

Responses from User Testing 2:
User 1:

1. The main interface feels more refined, especially with the multi-selection feature. It's easier to compare models now.

2. The sidebar remains intuitive, and the addition of tooltips is a game-changer. It helped me understand certain features better.

3. The model selection process is more streamlined, and the dropdown menu is responsive.

4. Uploading data feels more secure with the enhanced data validation. The confirmation message is a nice touch, ensuring I know my data was uploaded successfully.

5. The visualizations are clearer than before. The interactive visualizations are particularly impressive, offering more insights into the data.

6. Textual displays are well-organized and easy to understand. The added tooltips provide valuable context where needed.

7. I appreciate the user customization feature. It's nice to adjust the app's visual aspects to my liking.

8. I'd rate the overall user experience a 9 out of 10 now. The improvements are evident.

9. No redundant elements noticed. The design feels cohesive.

10. The current visualizations are more than sufficient. The interactive ones add depth to the data representation.

User 2:

1. I found the multi-selection feature very useful, especially when comparing models.
2. The ‘st.multiselect‘ was easy to navigate.
3. The confirmation message was a great touch; it gave me confidence in the upload process.
4. The design feels more refined now, and the user experience is smoother.
5. The textual displays are clearer, but maybe tooltips could be added for first-time users.
6. I’d rate the experience an 8 out of 10 now.
7. No redundant elements noticed.
8. The new features have definitely made the app more user-friendly.
9. Feedback messages were concise and to the point.
10. The current visualizations are sufficient, but interactive ones might be a good future addition.

User 3:

1. The main interface was intuitive and easy to navigate.
2. The sidebar was initially a bit overwhelming, but after a few interactions, it became clear.
3. The model selection dropdown was a nice touch, making the process straightforward.
4. Uploading data was clear, but a confirmation message would be reassuring.
5. The visualizations provided a clear understanding of the detection results.
6. Textual displays were clear and easy to understand.
7. A history or log of past uploads might be useful.
8. I’d rate the overall user experience as 8 out of 10.
9. No, all elements seemed necessary and added to the experience.
10. The current visualizations are good, but some interactivity might be beneficial for detailed analysis.

User 4:

1. The main interface was clean and well-organized.
2. The sidebar was well-organized, making secondary options easy to find.

3. The model selection process was seamless.
4. Uploading data was easy, especially with the progress bar.
5. The visualizations were clear and concise.
6. Textual displays were well-structured and informative.
7. Grouping related functions or features might be beneficial.
8. I'd rate the overall user experience as 9 out of 10.
9. No, all elements were essential.
10. While the current visualizations are sufficient, some interactivity might be a good addition.

User 5:

1. The main interface was clear and met expectations.
2. The sidebar was intuitive, but icons next to options might enhance visual appeal.
3. The model selection dropdown was easy to use.
4. Data input was straightforward, but a confirmation message post-upload would be helpful.
5. The visualizations were clear and added value to the results.
6. Textual displays were clear and easy to understand.
7. Perhaps a "recently viewed" or "history" feature might be a good addition.
8. I'd rate the overall user experience as 7.5 out of 10.
9. No, all elements seemed necessary.
10. The current visualizations are good, but adding some interactivity might be beneficial.

Design Iterations 2

After the second round of user testing, we gathered the feedback and identified areas that still needed refinement. Based on the collective feedback, we made the following enhancements:

- **Tooltip Integration:** To assist first-time users, we integrated tooltips for key features, ensuring that users can get quick help without feeling lost.

- **Primary Colours Customization:** Recognizing that users might have different preferences, we changed some visual aspects of the app, such as theme colors.

These changes were aimed at further refining the user experience and ensuring that our design was not only functional but also intuitive and engaging.

Lo-Fi Sketches

To visually represent our design evolution, we created lo-fi sketches:

- **Initial Design Sketch:** This sketch showcased the basic layout, focusing on simplicity and leveraging Streamlit's default design system.
- **Iterated Design Sketch:** After incorporating user feedback, this sketch displayed enhanced features and a more intuitive layout.

It's crucial to note that these sketches served as a testament to our commitment to user-centric design, ensuring that their feedback was not just heard, but actively implemented. Refer to Figure 17

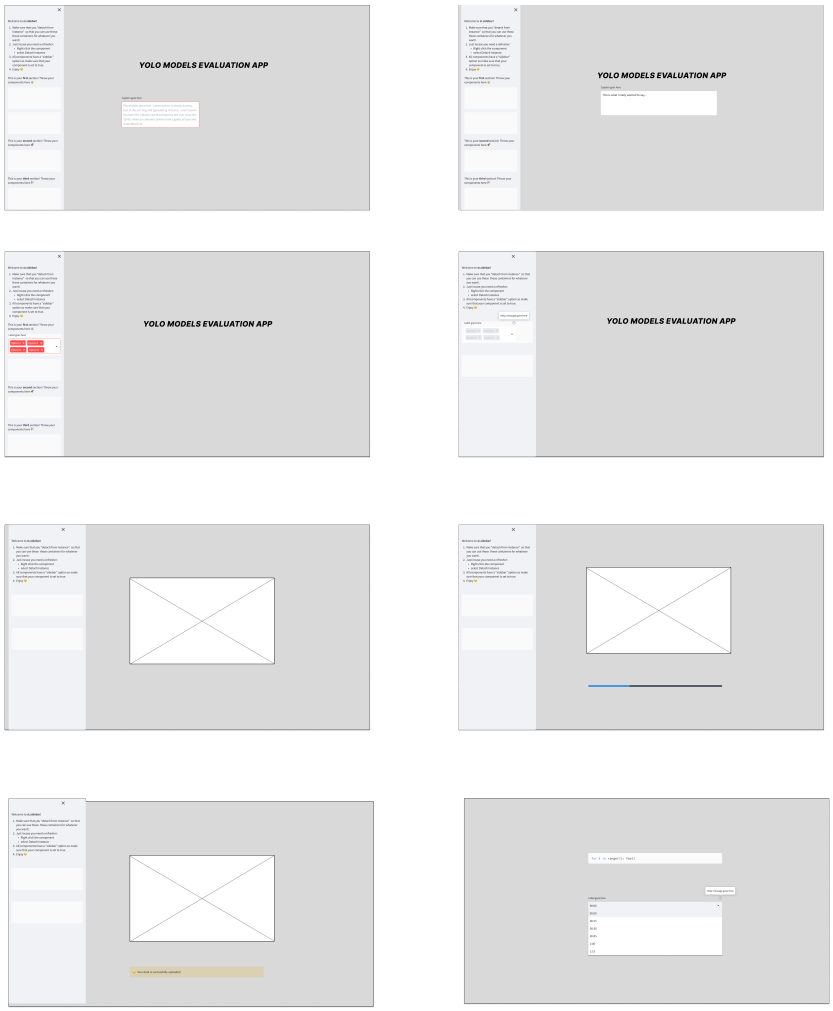


Figure 17: Comparison of initial and iterated lo-fi sketches. **Top row:** Input text box with default red borders (left) and iterated borders with a calmer color (right). **Second row:** Multi-select box with default colors (left) and iterated color (right). **Third row:** Data upload initial page (left) and data upload with progress bar (right).**Bottom row:** Successful upload confirmation message (left) and tooltip integration (right).

.

0.4.3 High-Fidelity Designs

This section presents the high-fidelity designs of our YOLO models Evaluation application. These designs offer a detailed visual representation of the user interface, capturing the look and feel of the final product.

Authentication Process

The authentication process ensures that only authorized users can access the dashboard. This is crucial for maintaining the security and integrity of the application.

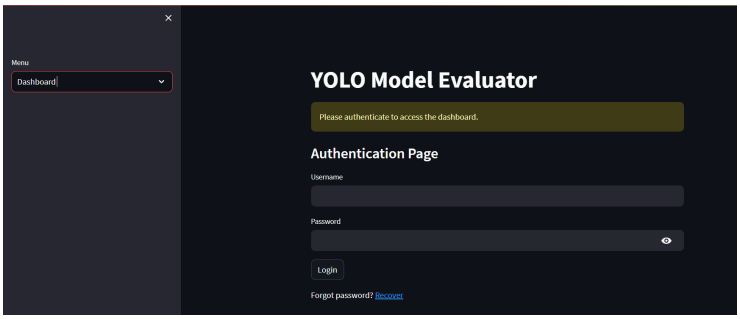


Figure 18: Attempt to access the dashboard without authentication.

As depicted in Figure 18, unauthorized access attempts are promptly identified and restricted.

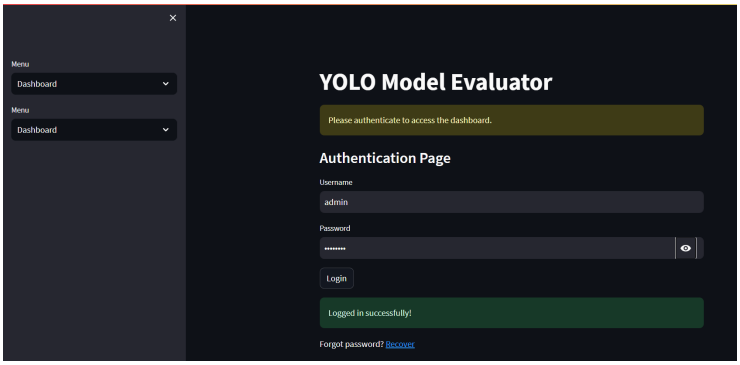


Figure 19: Successful authentication screen.

Upon successful authentication, as shown in Figure 19, users are granted access to the dashboard's main interface.

Dashboard Features

Once authenticated, users can navigate through various features and functionalities of the dashboard.

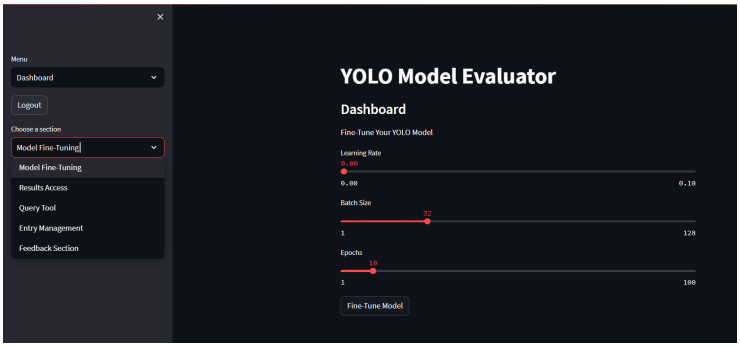


Figure 20: Dropdown box showcasing different dashboard functionalities and model fine-tuning option chosen from the dropdown box

Figure 20 displays the dropdown box, which allows users to:

- Access the model fine-tuning page.
- View detection results.
- Provide feedback.
- Execute professional SQL queries.

The dashboard's design ensures a seamless user experience, allowing users to efficiently navigate and utilize the application's features.

Feedback and Iterations: Upon presenting our hi-fi designs to users, we received feedback on visual elements like color contrasts, font sizes, and button placements. Based on this feedback, we made several iterations to improve clarity, readability, and overall user experience. We are planning to provide an option to do a professional query using SQL and also a multi-selection choice for comparison. Figure 21 presents the SQL query, while Figure 22 showcases the refinements made based on user feedback to have an ability to get some metrics without SQL query.

Testing Feedback: To ensure transparency and demonstrate our commitment to user-centric design, we documented feedback from our hi-fi testing sessions:

- **User A:** "The color contrast in the initial design was a bit harsh on the eyes. The revised design is much more comfortable to look at." (We added 'green' as a secondary color instead of 'red' by default)

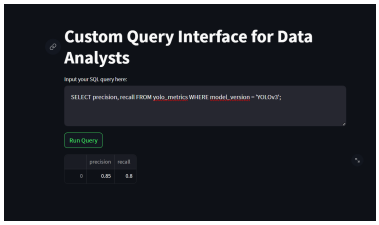


Figure 21: SQL Query

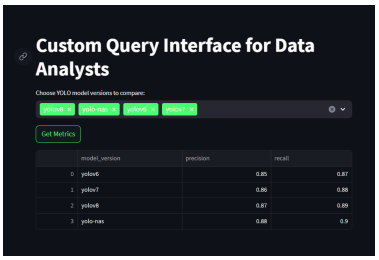


Figure 22: Multiselect box to get metrics on chosen models

- **User B:** "I appreciate the tooltips added in the revised design. They provide clarity on what each button does."
- **User C:** "The font size in the initial design was too small for me. The revised design is much more readable."
- **User D:** "I do not know how to perform a query using SQL but I still want to compare different models, adding the option using multiselect box was a delight."

For a comprehensive view, the full list of hi-fi and lo-fi screens can be accessed in the Figma file[4].

Transitioning from lo-fi sketches to hi-fi designs allowed us to refine our application's user interface, ensuring it's not only functional but also aesthetically pleasing and aligned with Streamlit's design principles.

0.5 Limitations & Conclusion

Limitations Embarking on this intricate journey of developing an object detection application using multiple YOLO models, I encountered several challenges. Being the sole developer on this multifaceted project, juggling between tasks like understanding the nuances of different YOLO models, integrating them into Streamlit, and ensuring a seamless user experience was demanding. My expertise in areas like deep learning, computer vision, and UX/UI design was continually tested. Hardware constraints, especially GPU capacity, affected the efficiency of model training and testing. Real-world testing scenarios were also limited due to time constraints, access to diverse datasets, and the challenge of obtaining accurate ground truth for app evaluation. However, these hurdles have been instrumental in my growth, teaching me resilience, enhancing my project management skills, and offering invaluable learning experiences.

[4]The detailed designs can be accessed at https://www.figma.com/file/Ww9SKdfCnHvCT1dX68qxuz/302-Capstone?type=design&node-id=0%3A1&mode=design&t=lcC9mVee0OsqgNQi-1.

Conclusion The Information Design and Development (IDD) phase has been pivotal in setting the groundwork for the object detection application using various YOLO models. The design methodologies employed, from UML diagrams to use case scenarios, have provided clarity and direction to the project.

This phase has elucidated the application's visual design, flow, and functionality, ensuring the creation of an intuitive user interface and a cohesive user experience. The UML diagrams, including class, sequence, and activity diagrams, have been indispensable for communicating and documenting the app's structure and design intricacies.

Furthermore, the use case scenarios have illuminated essential user interactions and requirements. The tools and frameworks chosen, such as different YOLO models, Streamlit, OpenCV, Roboflow, and others, are geared towards optimizing the development process.

As the project transitions into the development phase, the insights and groundwork from the IDD phase will be crucial. The meticulous design approach will undoubtedly play a significant role in realizing a robust, user-centric, and efficient object detection application, catering to both users and stakeholders.

Bibliography

Streamlit. (2023). *Connections and databases - streamlit docs* [Accessed: [19/09/2023]]. https://docs.streamlit.io/library/api-reference/connections

Date and Signature	Supervisor's Notes
	•
	•
	•

Table 3: Supervisor Notes and Signature

Appendix

Weekly Meetings

Date	Agenda	Questions
28/08	• Discuss the rubrics. • Review the project timeline and milestones.	• What are the primary deliverables expected at the end of this project? • Are there any specific methodologies or tools you recommend for this stage?
04/09	• Update on the progress made during the week. • UML Diagrams for IDD.	• How can I effectively address the challenges encountered during data post-processing and storage? • Are there any resources or references you suggest for the next phase?
11/09	• Document Writing & formating. • Submission package.	• Based on the results, what improvements do you suggest for the model training process? • How should I prioritize tasks for the next weeks?

Table 4: Summary of Weekly Meetings with Supervisor