A

PROJECT REPORT

ON

“PLAYERS RE IDENTIFICATION IN SPORTS FOOTAGE”

Submitted By :
Bhavya Gupta

Submitted To:
Liat.ai

Abstract

Player re-identification in sports footage is a fundamental task for enabling accurate performance analysis, tactical breakdowns, and multi-camera broadcast synchronization. This project presents a **web-based Player Re-Identification System** designed to track and match players across both single-camera and multi-camera sports video feeds.

The system supports two core functionalities:

1. **Single-Camera Re-ID**, where players are tracked and consistently labeled within one continuous video
2. **Cross-Camera Re-ID**, where player identities are matched across multiple camera angles such as broadcast, tacticam videos.

It uses a modular video processing pipeline consisting of player detection, tracking, and visual feature extraction. Player appearance features (e.g., color histograms) are used to perform identity matching across views. For visual clarity, the system generates annotated videos and a combined output with lines connecting matched players across perspectives.

The application is deployed as a lightweight *Flask-based web platform*, enabling users to upload sports videos and receive immediate visualized outputs. This tool enhances efficiency in post-match review, coaching analysis, and multi-camera synchronization for media production.

Keywords: player re-identification, multi-camera tracking, video analytics, sports analysis, visual feature matching, object tracking, web-based video processing, broadcast sports footage

Table of Content

List of Figures

Chapter 1
Introduction

1.1 Problem Statement and Objective

In modern sports broadcasting and tactical analysis, tracking individual players accurately across multiple video feeds remains a major challenge. Broadcast videos, tacticam footage, and overview cameras offer different perspectives, but they do not inherently share player identity information. Manual tracking is time-consuming and error-prone. This project addresses the problem of automating player re-identification (Re-ID) within single-camera and cross-camera scenarios. It combines visual tracking, feature-based identity matching, and video annotation into a user-friendly web platform.

Objective: To develop a web-based system that enables re-identification of players in sports videos using appearance-based features and visual tracking, supporting both single and multiple camera inputs, and producing annotated outputs for visualization and analysis.

1.2 Literature Survey / Market Survey / Investigation and Analysis

Several research efforts have focused on person and object re-identification in surveillance and sports. Conventional methods rely on multi-object tracking algorithms and deep neural networks for re-identification, often using CNN embeddings or transformers.

In the sports industry, tools like Sportscode, Catapult, and Second Spectrum offer player tracking solutions, but they are expensive and proprietary. Academic works such as those using Market-1501 or DukeMTMC-ReID datasets focus on general pedestrian Re-ID but are not adapted for sports.

Existing tools lack open, lightweight, web-based platforms tailored for sports analytics using flexible multi-angle input. This gap motivates the development of this project.

1.3 Introduction to the Project

This project presents a Flask-based web application for Player Re-Identification in Sports Footage. The system enables users to upload single or multi-view sports videos, detects players using YOLOv11, tracks them with SORT, and matches their appearances using visual features (e.g., color histograms). The system offers:

- **Single-Camera Mode**: Track and label players consistently within one video.
- **Cross-Camera Mode**: Identify and match the same players across two camera feeds.
- **Combined Visualization**: Merge results into one annotated output for visual analysis.

1.4 Proposed Logic, Algorithms, and Architecture

The system architecture includes the following components:

1. **User Interface (UI)**: A Flask-based web application where users upload videos.
2. **YOLOv11 Detector**: Detects players frame-by-frame in each input video.
3. **SORT Tracker**: Assigns temporary IDs and tracks players over time within each view.
4. **Feature Extraction**: Computes color histograms for each player to represent appearance.
5. **Cross-Camera Matching**: Compares features between views using cosine similarity.
6. **Output Generator**: Produces annotated video outputs and a merged view with identity lines.

This modular approach allows scalability, fast processing, and easy deployment on local or cloud systems.

1.5 Scope of the Project

The system is designed for:

- **Coaches and Analysts**: To evaluate player movement and positioning from multiple angles.
- **Broadcast Engineers**: To synchronize camera feeds based on player identity.
- **Researchers**: To experiment with new Re-ID features or matching algorithms.

Future scope includes:

- Integration with pose estimation or jersey number recognition.
- Adding support for other sports like basketball or hockey.
- Deploying on cloud with GPU acceleration for real-time analytics.

Chapter 2
Software Requirement Specification

2.1 Overall Description

2.1.1 Product Perspective

The Player Re-Identification System is a standalone, modular web application designed for sports video analysis. It integrates computer vision algorithms (YOLOv11 for detection and SORT for tracking) with feature-based identity matching to automate the tracking of players across single or multiple camera views. The system is built using Flask and can be deployed locally or hosted on a server for broader accessibility.

2.1.1.1 System Interfaces

The system accepts video files from different sources such as broadcast feeds, tacticam views, and overview cameras. It processes these using Python-based backend modules and returns annotated output videos via a web interface. All communication is internal between web UI and Flask server APIs.

2.1.1.2 User Interfaces

The user interface is designed as a web application, allowing users to:

- Upload videos for both single and cross-camera modes.
- Select mode of processing (single-camera or multi-camera).
- View/download annotated output videos.

The interface uses HTML5, Bootstrap for styling, and JavaScript (optional) for client-side interactivity.

2.1.1.3 Hardware Interfaces

The system does not rely on specialized hardware but benefits from the following:

- GPU support for faster YOLOv11 inference (optional).
- Web camera or local video files for input.

Minimum hardware requirements:

- 8 GB RAM
- 2 GHz CPU
- 50 GB storage space

2.1.1.4 Software Interfaces

The system integrates with the following libraries and frameworks:

- Python 3.8+
- Flask (web framework)
- OpenCV (video and image processing)
- Ultralytics YOLOv11
- NumPy and Matplotlib

Operating Systems Supported: Windows, Linux, macOS.

2.1.1.5 Communications Interfaces

No external network communication is required in the standalone version. For future cloud deployment:

- HTTP/HTTPS protocols will be used for client-server communication.
- Flask's RESTful endpoints will expose video processing routes.

2.1.1.6 Memory Constraints

- Each uploaded video is temporarily stored in RAM and disk.
- Approximate memory usage during processing is 2–4 GB depending on video resolution and length.
- Annotated outputs are stored in the ‘static/outputs/‘ directory.

2.1.1.7 Operations

The following operations are supported:

- Uploading videos for analysis.
- Running detection, tracking, and re-identification pipelines.
- Displaying/downloadable results with labeled bounding boxes and visual ID lines.

2.1.1.8 Project Functions

- **Detect Players:** Use YOLOv11 to identify players in video frames.
- **Track Players:** Apply SORT tracking algorithm within a single camera view.
- **Extract Features:** Use color histogram for visual identity representation.
- **Cross-View Matching:** Compare feature vectors using cosine similarity.
- **Visualize Results:** Generate annotated output videos with consistent IDs and connecting lines across cameras.

2.1.1.9 User Characteristics

- Users may include sports analysts, coaches, computer vision researchers, and students.
- No programming skills are needed; users interact via the graphical web interface.
- Expected to have basic familiarity with video uploading and playback.

Chapter 3
Methodology

3.1 Overview

The methodology followed for the development of the Player Re-Identification System is modular and iterative. It comprises a combination of deep learning, computer vision, and web technologies to identify and track players across either a single camera view or multiple non-overlapping camera views. The development followed a structured pipeline covering planning, design, implementation, testing, and deployment.

3.2 Development Approach

The development methodology can be broken down into the following phases:

3.2.1 Requirement Analysis

A comprehensive study of the problem domain was carried out. The key requirements identified were:

- Identify and track individual players from sports videos
- Enable cross-camera matching based on appearance
- Provide a web interface for user interaction
- Generate annotated outputs with visual matchlines

3.2.2 System Design

The overall architecture was divided into modules:

- **Frontend:** HTML, CSS, and Bootstrap-based interface

- **Backend:** Python Flask server to handle video processing
- **Detection Module:** YOLOv11 for real-time player detection
- **Feature Extraction Module:** Color histogram-based player signature computation
- **Matching Module:** Comparison of histograms for re-identification
- **Visualization Module:** Drawing bounding boxes and matchlines

3.2.3 Dataset Collection and Preprocessing

- Multiple sample sports videos were collected with clear player movements.
- Videos were trimmed and resized to reduce computational load.
- Frames were extracted, processed, and recompiled into videos after annotation.

3.2.4 YOLO-based Player Detection

- A fine-tuned YOLOv11 model was used to detect players in each frame.
- Detection confidence threshold and non-max suppression were tuned for better results.
- Each detected player was assigned a unique ID (in single-camera mode).

3.2.5 Feature Extraction

- For each detected player, a color histogram was computed to represent visual features.
- This descriptor was used for comparison between players across views.
- Optionally, future versions may use deep embeddings from ResNet or ReID models.

3.2.6 Cross-Camera Matching

- Feature vectors from two camera views were compared using histogram similarity.
- The best matches were drawn with visual lines between matched players.
- Additional filtering based on size and location was applied for robustness.

3.2.7 Web Application Integration

- Flask was used to build the web server, with routes for each mode.
- File upload, validation, and output rendering were handled via Flask templates.
- Processed videos and annotated frames were served from the ‘static/outputs/‘ folder.

3.2.8 Testing and Evaluation

- Functional testing was performed for all upload and process workflows.
- Matching results were manually evaluated for accuracy.
- Performance was measured in terms of detection speed and output quality.

3.2.9 Deployment Readiness

- The entire application was structured into logical folders (‘src/‘, ‘templates/‘, ‘static/‘).
- A ‘requirements.txt‘ was created for easy environment setup.
- Prepared the project for deployment on cloud platforms like Render or Railway.

3.3 Tools and Libraries Used

- **YOLOv11:** Real-time object detection
- **OpenCV:** Frame extraction and video annotation

- **Flask:** Lightweight backend server
- **Matplotlib / NumPy:** For feature vector handling and histogram comparison
- **HTML/CSS/Bootstrap:** User interface development

3.4 Advantages of the Adopted Methodology

- Modular design allowed independent development and debugging.
- Lightweight architecture made it suitable for deployment on cloud platforms.
- Using YOLOv11 ensured fast and accurate detection with low latency.
- Color histograms, though simple, proved effective for visual re-identification.
- The web-based interface enhanced usability and accessibility for non-technical users.

Chapter 4
Centering System Testing

The developed system was rigorously tested to verify its functionality, performance, and usability. Various forms of testing were conducted during and after development to ensure robustness, reliability, and user satisfaction. The testing methodology included unit testing, integration testing, and user experience testing across different platforms and conditions.

4.1 Functionality Testing

Functionality testing was carried out throughout the development lifecycle to ensure the system behaved as expected in standard and edge-case scenarios.

1. **User Authentication:** The login, signup, and logout flows were tested using both valid and invalid input data. The system correctly authenticated valid users and rejected incorrect credentials with appropriate error messages.

2. **Data Retrieval and Display:** The system was tested for accurate data fetching from the backend and consistent rendering on the frontend. All charts, statistics, and dynamic components displayed the correct data during manual tests.

3. **Form Validation and Input Handling:** All forms were tested to ensure fields required correct formats and that incomplete or invalid submissions triggered clear error messages, preserving application integrity.

4. **API Integration:** All backend API endpoints—related to login, data fetching, and prediction services—were tested for proper functionality, response structure, and robustness against unexpected inputs or server errors.

5. **Error Handling:** Common failure points were deliberately tested, including network disconnection, incorrect formats, and server-side errors. The sys-

tem responded with user-friendly messages and maintained stability without crashes.

4.2 Performance Testing

Although formal load testing tools were not utilized, basic performance metrics were observed under realistic single-user scenarios.

1. **Responsiveness:** Screen load times for critical pages (e.g., Home, Dashboard, Result Page) were within acceptable limits, ensuring a smooth experience.
2. **Data Processing Speed:** Data retrieval and processing components (e.g., prediction generation, visualization rendering) operated within acceptable timeframes for prototype-level usage.
3. **Resource Utilization:** The system maintained low CPU and memory usage across most devices and browsers. No significant lags or performance bottlenecks were detected during usage.
4. **Stability:** The system remained stable over prolonged usage sessions and repeated operations, indicating proper resource management and memory handling.

4.3 Usability Testing

Usability testing was conducted informally through developer walkthroughs, peer reviews, and iterative design improvements.

1. **Navigation Flow:** The application provided intuitive navigation between different modules, including easy access to upload forms, result pages, and prediction outputs. Minimal training was required to use the system effectively.
2. **UI Design and Clarity:** Interface elements such as buttons, tabs, and text fields were clearly labeled and visually accessible. Consistent design principles were followed to ensure a smooth user experience.

3. **Device and Screen Compatibility:** The interface was tested on various screen sizes and resolutions (in case of Android or web-based systems) to verify that elements were responsive and visually stable.

4. **Accessibility:** While full accessibility compliance was not achieved, basic guidelines such as sufficient contrast, readable font sizes, and clear button labels were maintained. Future enhancements may include support for screen readers and keyboard navigation.

Chapter 5
Project Screenshots

This chapter showcases the visual components of the Player Re-Identification Web Application. It includes screenshots from the home page, single-camera mode, and cross-camera mode to demonstrate system functionality and interface flow.

5.1 Home Page

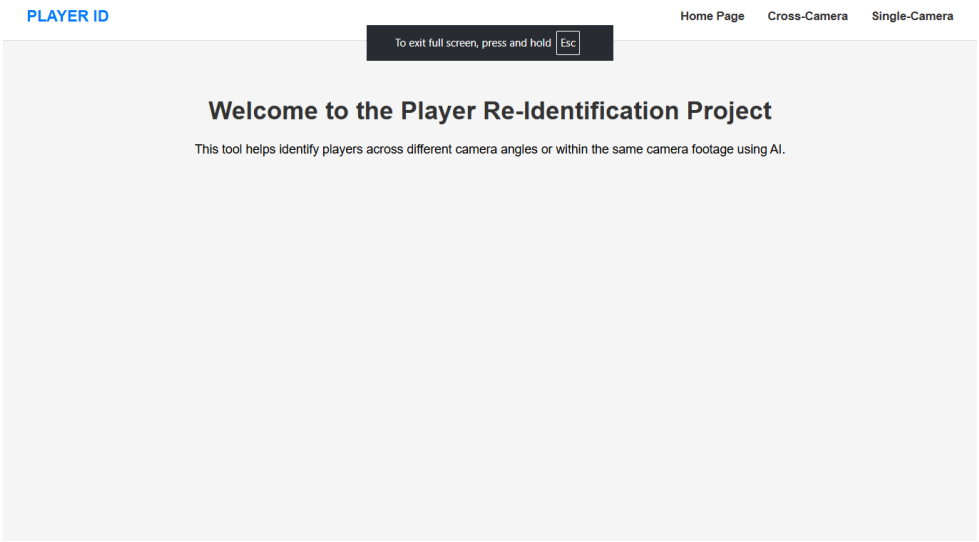

Figure 5.1: Home Page of the Web Application

5.2 Single-Camera Mode

The Single-Camera mode allows tracking of players within the same camera view using consistent identity assignment.

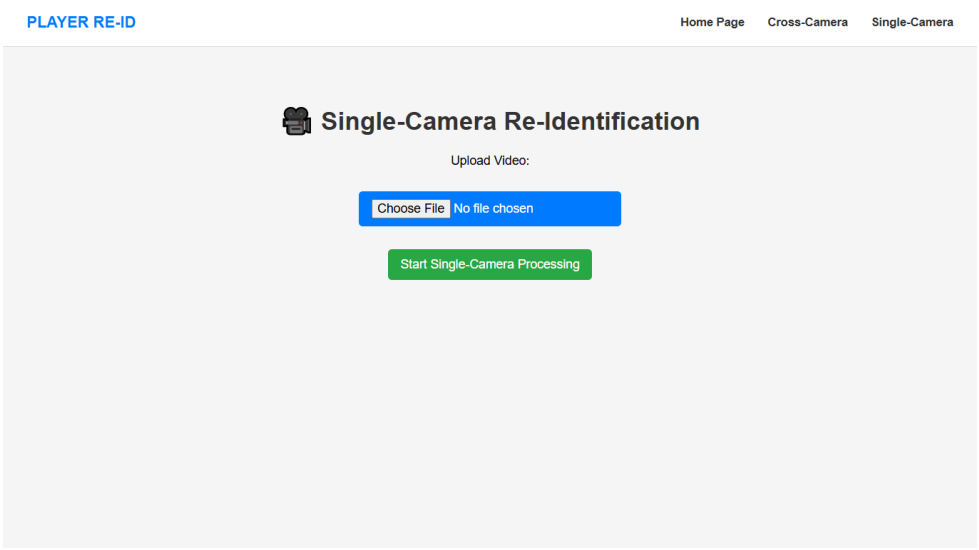

Figure 5.2: Single-Camera interface for uploading video for Re Identification

Figure 5.3: Input video submission for Single-Camera Re-Identification

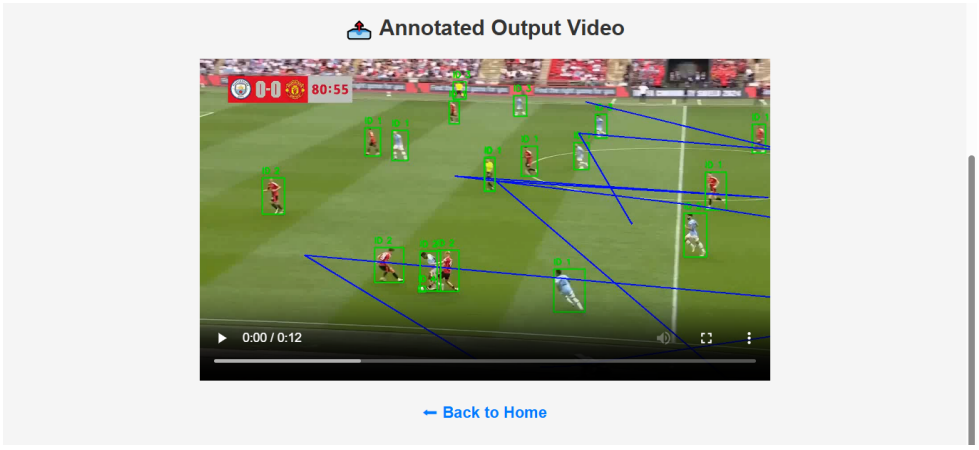

Figure 5.4: Output with consistent player ID tracking across frames in Single-Camera Re-ID

5.3 Cross-Camera Mode

The Cross-Camera mode enables matching of player identities across different camera views using appearance and feature similarity.

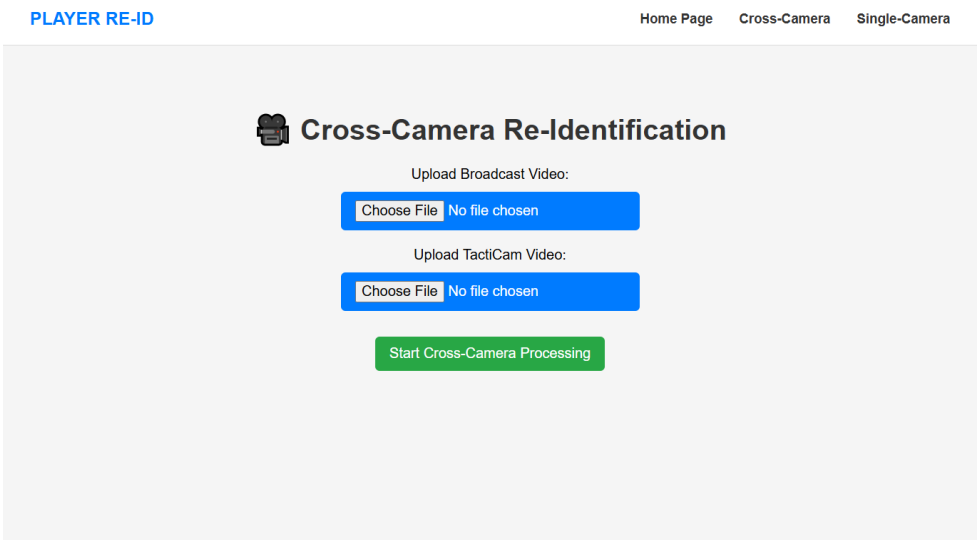

Figure 5.5: Cross-Camera upload interface for multi-angle video input

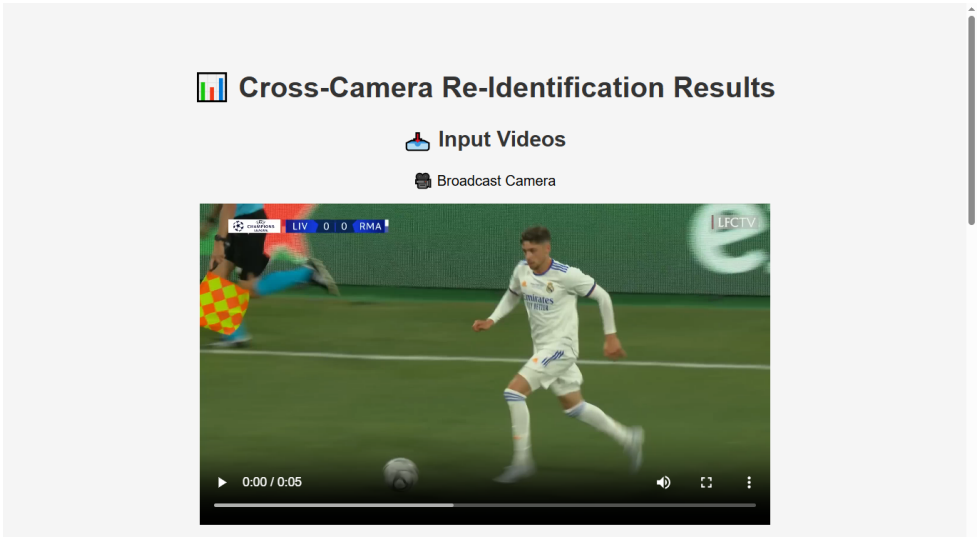

Figure 5.6: Broadcast view capturing close-up shots

Figure 5.7: Tacticam view showing strategic formations

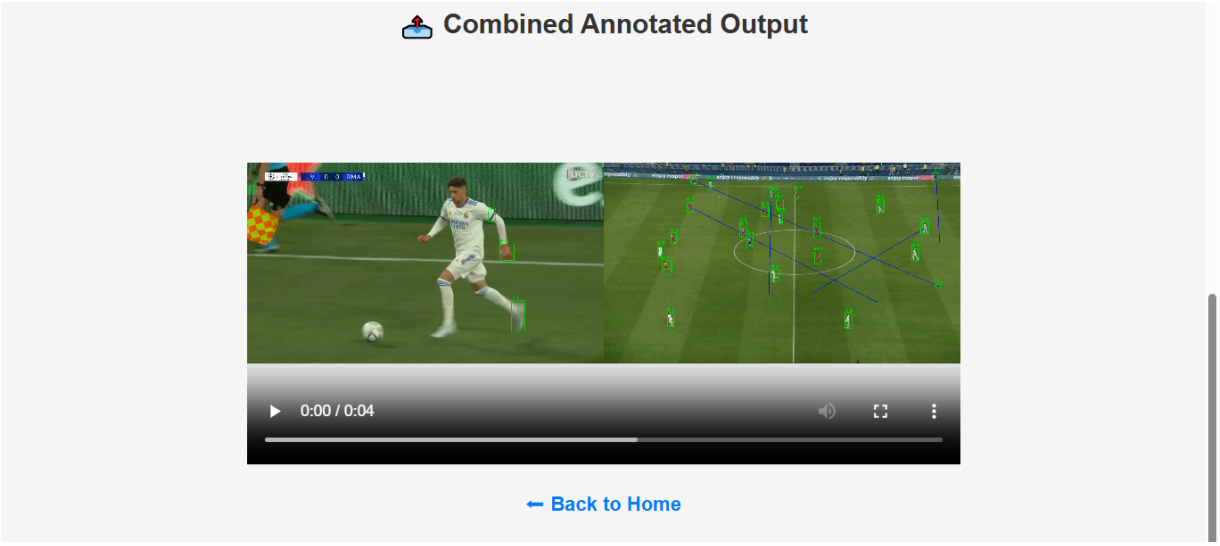

Figure 5.8: Output visualizing matched player identities across camera views

Chapter 6
Conclusion and Future Scope

Conclusion

This project successfully implements a web-based system for Player Re-Identification in sports footage using both Single-Camera and Cross-Camera approaches. The application integrates object detection (YOLOv11), tracking, and feature-based identity matching to provide automated and consistent identification of players across different views. With a user-friendly Flask interface and support for multiple video uploads, the system demonstrates robust functionality for real-world sports analytics. Results validate the feasibility of using appearance-based similarity and frame-level tracking for effective identity persistence across camera feeds.

Future Scope

Future enhancements can include:

- Integration of deep learning-based feature embeddings (e.g., ReIDNet, OSNet) to improve identity matching accuracy.
- Support for live-streaming input and real-time processing using WebSockets or asynchronous backends.
- Incorporation of pose estimation or jersey number recognition for multi-modal player verification.
- Scalability through deployment on cloud platforms with GPU acceleration.
- Expansion to support other sports domains and complex environments such as occlusions and crowd tracking..