ROBERT GORDON UNIVERSITY•ABERDEEN

INFORMATICS INSTITUTE OF TECHNOLOGY

Informatics Institute of Technology

In Collaboration with

Robert Gordon University,
Aberdeen, Scotland

ROBERT GORDON UNIVERSITY•ABERDEEN

INFORMATICS INSTITUTE OF TECHNOLOGY

CricXpert

A Hybrid Approach Combining Facial and Spatio-Temporal Gait Analysis for Enhanced Player Recognition with LLM-Based Statistic Generation

Final Year Thesis by

Mr. K.D Nadun Shamika Senarathne

RGU ID	2117538
IIT ID	20210488

Supervised by

Mr. Prasan Yapa

April 2025

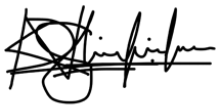
Submitted in partial fulfilment of the requirements for the BSc. (Hons) in Artificial Intelligence and Data Science at the Robert Gordon University, Aberdeen, UK.

ROBERT GORDON UNIVERSITY•ABERDEEN

INFORMATICS INSTITUTE OF TECHNOLOGY

DECLARATION

I confirm that the work contained in this BSc project report has been composed solely by myself and has not been accepted in any previous application for a degree. All sources of information have been specifically acknowledged and all verbatim extracts are distinguished by quotation marks.



…… 04/23/2025 ……

Mr. K.D Nadun Shamika Senarathne — Date

The above student carried out his research project under my supervision.

prasan yapa

…… 04/23/2025 ……

Mr. Prasan Yapa — Date

STUDENT PROJECT ETHICAL REVIEW (SPER) FORM

The aim of the University's *Research Ethics Policy* is to establish and promote good ethical practice in the conduct of academic research. The questionnaire is intended to enable researchers to undertake an initial selfassessment of ethical issues in their research. Ethical conduct is not primarily a matter of following fixed rules; it depends on researchers developing a considered, flexible and thoughtful practice.

The questionnaire aims to engage researchers discursively with the ethical dimensions of their work and potential ethical issues, and the main focus of any subsequent review is not to 'approve' or 'disapprove' of a project but to make sure that this process has taken place.

The *Research Ethics Policy* is available at
www.intranet.rgu.ac.uk/credo/staff/page.cfm?pge=7060

Student Name	K.D Nadun Shamika Senarathne
Supervisor	Mr. Prasan Yapa
Project Title	CricXpert: A Hybrid Approach Combining Facial and Spatio-Temporal Gait Analysis for Enhanced Player Recognition with LLMBased Statistic Generation
Course of Study	BSc (Hons) Artificial Intelligence And Data Science
School/Department	IIT School of Computing

Part 1 : Descriptive Questions			
1	Does the research involve, or does information in the research relate to:	Yes	No
	(a) individual human subjects		✓
	(b) groups (e.g. families, communities, crowds)		✓
	(c) organisations		✓
	(d) animals?		✓
	Please provide further details:		
			✓

2	Will the research deal with information which is private or confidential?	Yes	No
			✓
	Please provide further details:		

Part 2: The Impact of the Research			
3	In the process of doing the research, is there any potential for harm to be done to, or costs to be imposed on	Yes	No
	(a) research participants?		✓
	(b) research subjects?		✓
	(c) you, as the researcher?		✓
	(d) third parties?		✓
	Please state what you believe are the implications of the research:		
4	When the research is complete, could negative consequences follow:	Yes	No
	(a) for research subjects		✓
	(b) or elsewhere?		✓
	Please state what you believe are the consequences of the research:		

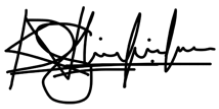
Part 3: Ethical Procedures			
5	Does the research require informed consent or approval from:	Yes	No

	(a) research participants?		✓
	(b) research subjects		✓
	(c) external bodies		✓
	If you answered yes to any of the above, please explain your answer:		
6	Are there reasons why research subjects may need safeguards or protection?	Yes	No
			✓
	If you answered yes to the above, please state the reasons and indicate the measures to be		
7	Has PVG membership status been considered?		
	(a) PVG membership is not required.		✓
	(b) PVG membership is required for working with children.		✓
	(c) PVG membership is required for working with protected adults.		✓
	(d) PVG membership is required for working with both children and protected		✓
	If you answered yes to (b), (c) or (d) above, please give details:		
8	Are specified procedures or safeguards required for recording, management, or storage of data?	Yes	No
			✓
	If you answered yes to the above, please outline the likely undertakings:		

Part 5: Other Issues			
11	Are there any other ethical issues not covered by this form which you believe you should raise?	Yes	No
			✓

Part 4: The Research Relationship

9	Does the research require you to give or make undertakings to research participants or subjects about the use of data?	Yes	No
			✓
	If you answered yes to the above, please outline the likely undertakings:		
10	Is the research likely to be affected by the relationship with a sponsor, funder or employer?	Yes	No
			✓
	If you answered yes to the above, please identify how the research may be affected:		

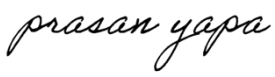
Statement by Student I believe that the information I have given in this form is correct, and that I have addressed the ethical issues as fully as possible at this stage.			
Signature		Date	11/18/2024

If any ethical issues arise during the course of the research, students should complete a further Student Project Ethical Review (SPER) form.

The *Research Ethics Policy* is available at
www.intranet.rgu.ac.uk/credo/staff/page.cfm?pge=7060

Part 6: To be completed by the supervisor

12	Does the research have potentially negative implications for the University?	Yes	No ✓
	If you answered yes to the above, please explain your answer:		
13	Are any potential conflicts of interest likely to arise in the course Yes No of the research?		✓
	If you answered yes to the above, please identify the potential conflicts:		
14	Are you satisfied that the student has engaged adequately with the ethical implications of the work? [In signifying agreement, supervisors are accepting part of the ethical responsibility for the project]	Yes ✓	No
	If you answered no to the above, please identify the potential issues:		
15	Appraisal: Please select one of the following		
	The research project should proceed in its present form – no further action is required	✓	
	The research project requires ethical approval by the School Ethics Review Panel		
	The research project needs to be returned to the student for modification prior to further action		
	The research project requires ethical review by an external body. If this applies please give details		
	Title of External Body providing ethical review		
	Address of External Body		
	Anticipated date when External Body may consider project		

Affirmation by Supervisor I have read the student's responses and have discussed ethical issues arising with the student. I can confirm that, to the best of my understanding, the information presented by the student is correct and appropriate to allow an informed judgement on whether further ethical approval is required.			
Signature		Date	21/11/2024

ABSTRACT

In the fast-paced environment of T20 International (T20i) cricket, precise identification of fielding players, particularly during the last overs, remains a technical challenge due to fluctuating lighting, occlusion, and non-frontal camera angles. This thesis proposes CricXpert, a hybrid AI system which employs computer vision and large language models (LLMs) to handle two fundamental problems: player recognition and natural language-based statistic retrieval.

In order to address these issues, the system combines multiple AI components: facial recognition with MTCNN and FaceNet, spatio-temporal gait analysis with a GRU model trained on MediaPipe pose features, and spatial classification with ResNet50, SVM, KNN, and Logistic Regression in a stacking ensemble. When face or gait data are insufficient, EAST and Tesseract are used in an OCR pipeline to identify jersey names and numbers. GPT-4o is prompt-engineered to generate SQL queries from user inquiries in order to retrieve statistics from a structured MySQL database.

Experiments on a custom-built dataset of six cricket players annotated under match-like situations yielded high-performance metrics, including 98.14% accuracy in spatial recognition and 95% in temporal gait classification and 95.83% in facial recognition. The LLM achieved 85-90% SQL translation accuracy with a response time of around 2s. Cricket coaches and data scientists validated the system's real-world utility, usability, and domain relevance, with all evaluation categories earning at least 4.5 on a 5-point Likert scale.

The findings demonstrate that integrating deep learning with classical ML classifiers and prompt-engineered LLMs yields a reliable, interpretable, and low-latency solution for player recognition and sports analytics. This end-to-end framework not only enhances identification accuracy in challenging circumstances, but it also allows non-technical users to engage with structured cricket data using natural language.

Keywords: Cricket Analytics, Player Recognition, Spatio-temporal Gait Analysis, Facial Recognition, Large Language Models, SQL Generation, Hybrid Ensemble, OCR, Statistic Genaration

ACKNOWLEDGEMENT

First and foremost, I would like to express my sincere gratitude to my supervisor, Mr. Prasan Yapa, for his continuous guidance, constructive feedback, and encouragement throughout the course of this project. His insights were instrumental in shaping this research and overcoming the many technical challenges faced.

I also extend my appreciation to the faculty and staff of the BSc (Hons) in Artificial Intelligence and Data Science program at the Informatics Institute of Technology (IIT), in affiliation with Robert Gordon University (RGU), for providing a strong academic foundation and supportive learning environment.

A special thanks goes to my friends and family, whose support and motivation helped me stay focused and positive during this journey. I am also grateful to the creators and maintainers of open-source tools and libraries used in this project, as they were crucial for successful implementation.

Lastly, I would like to thank all those who contributed, directly or indirectly, to the completion of this research project. Your support has been deeply appreciated.

Table of Contents

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

- **AI** – Artificial Intelligence
- **API** – Application Programming Interface
- **CNN** – Convolutional Neural Network
- **CV** – Computer Vision
- **DBMS** – Database Management System
- **EAST** – Efficient and Accurate Scene Text Detector
- **FN** – False Negative
- **FP** – False Positive
- **FPS** – Frames Per Second
- **GPT** – Generative Pre-trained Transformer
- **GRU** – Gated Recurrent Unit
- **ICC** – International Cricket Council
- **KNN** – K-Nearest Neighbors
- **LLM** – Large Language Model
- **LR** – Logistic Regression
- **LSTM** – Long Short-Term Memory
- **ML** – Machine Learning
- **MTCNN** – Multi-task Cascaded Convolutional Neural Network
- **NLP** – Natural Language Processing
- **OCR** – Optical Character Recognition
- **RNN** – Recurrent Neural Network
- **SQL** – Structured Query Language
- **SVC** – Support Vector Classifier
- **SVM** – Support Vector Machine
- **TP** – True Positive
- **TN** – True Negative
- **ViT** – Vision Transformer
- **YOLO** – You Only Look Once

01 INTRODUCTION

1.1 Chapter Overview

This chapter presents the research problem, outlines the purpose for the study, and identifies the existing research gap that the project is intended to address. It also develops research questions, explains the study's goals and objectives, and examines the project's significance and scope. The chapter discusses the context for the development of CricXpert, a hybrid computer vision and natural language processing system. The goal is to improve the accuracy and reliability of cricket player recognition during final overs of T20 International matches, as well as to make it easier to acquire player-specific statistics by leveraging prompt-engineered large language models. The chapter concludes by positioning the study within the broader scope of AI applications in sports analytics.

1.2 Problem Statement

In the context of T20i cricket, accurately identifying fielding players during the final overs remains a significant challenge due to adverse match conditions such as low lighting, occlusions, and high-speed movement. Existing unimodal recognition systems, which rely entirely on facial or gait features, tend to fail under such conditions. Concurrently, retrieving player-specific statistics from traditional cricket platforms is hampered by complicated interfaces and multi-step filtering methods, reducing the opportunity for timely analysis. These restrictions highlight the need for a strong, hybrid AI-driven solution that combines computer vision and large language models to enable accurate player recognition and seamless, natural language-based access to performance statistics in near real-time scenarios.

1.3 Research Motivation

The increasing popularity of artificial intelligence in sports opens up new opportunities for improving spectator experiences, coaching techniques, and player performance evaluations (Banoth et al. 2022). In T20i cricket, the final four overs are frequently the most intense and crucial. Accurate player recognition during this phase can help broadcasters, analysts, and fans make informed remarks and engage more intimately with the game.

Despite advances in face and gait recognition, existing models frequently fail to deliver consistent results under the challenging environmental conditions present in live cricket matches (Haq et al. 2024), (Kale et al. 2004), such as fluctuating lighting, motion blur, and occlusion. These shortcomings motivated the development of a hybrid recognition system that leverages the complementary strengths of facial features and spatio-temporal gait patterns.

Simultaneously, while the popularity of natural language interfaces grows, so does the demand for user-friendly data retrieval methods (Zhang et al. 2024). The integration of a large language model capable of converting natural language queries into SQL queries allows fans, coaches, and analysts to bypass traditional cumbersome filtering processes, providing rapid access to cricket data. By tackling these problems, this study hopes to make a significant contribution to the developing field of AI in sports analytics while also providing an original and practical solution.

1.4 Research Gap

This project identifies mainly two research gaps:

The **primary** research gap focuses on the integration of face recognition and hybrid spatio-temporal gait analysis into an unified system. This technique provides a viable route for improving identification accuracy in the more challenging conditions found in T20i cricket matches. Existing research has mainly focused on the use of facial recognition for player identification, with notable findings by (Haq et al. 2024) and (Mahmood et al. 2015) as well as a variety of gait analysis approaches such as (Kale et al. 2004) and (Arseev, Konushin, and Liutov 2018). However, these models have not used spatio-temporal features to assess gait in order to recognize a limited set of players – specifically 6 cricket players – during the critical last four overs of T20i matches. This oversight demonstrates a fundamental gap in existing technology's capacity to adapt to cricket's dynamic settings, due to various challenges identified. As mentioned above, such challenges which severely affect the effectiveness of typical identification systems are variable lighting, occlusions, and distant camera angles. Furthermore, while multimodal approaches that combine facial and gait recognition have been previously investigated (Maity, Abdel-Mottaleb and Asfour 2021), (Manssor, Sun, and Elhassan 2021), the use of spatio-temporal features to improve both accuracy and robustness in sports analytics, particularly under the fluctuating conditions of T20i matches, remains novel. Additionally, although prior studies have enhanced deep learning architectures by incorporating machine learning classifiers in contexts like remote sensing image recognition (Özyurt 2020) and brain tumor classification (Kibriya et al. 2021), (Kibriya et al. 2022), the proposed ensemble model introduces this approach within the spatial model in a unique context. By applying this novel combination specifically to player identification in T20i cricket matches, the suggested hybrid approach uniquely tackles these problems, paving the way for a creative solution in sports analytics.

The **secondary** research gap is pertaining to the use of large language models (LLMs) to understand and translate natural language queries inputted by the user into SQL queries, allowing for the retrieval of player statistics from a relational database. Such an approach has the potential to eliminate the time-consuming, multi-step filtering processes that are common in present systems, as detailed by (ESPN Cricinfo 2024). Previous research, especially that of (Shi, Tang, and Yang 2024), has thoroughly investigated the translation of natural language to SQL using LLMs. However, the present effort aims to improve this procedure further. By allowing users to easily enter queries and instantly receive the necessary statistics, it dramatically minimizes the need for complex filtering and interface navigation, improving the user experience tremendously. This streamlined interaction approach not only simplifies data access, but it also establishes a new standard for user-centric data interfaces in sports analytics.

By addressing these gaps, the project enhances T20i cricket analytics through improved accuracy in player identification and statistical data retrieval. This effort not only fosters greater fan engagement but also contributes to scholarly discourse by merging computer vision and natural language processing within a unique sporting context. These advancements are expected to overcome the existing technological limitations and improve the spectator experience, indicating a big step forward in the context of sports technology.

1.5 Research Questions

Research Question 1: Player Recognition

"How effectively can a computer vision ensemble model employing face recognition and gait analysis using spatio-temporal features, recognise players in the outfield during the last four overs of a day/night T20 International cricket match?"

Research Question 2: Statistic Generation

"How can a large language model be prompt-engineered to accurately translate natural language user-defined questions, with up to three conditions, into SQL queries for generating accurate and relevant statistics?"

1.6 Research Aim and Objectives

Aim:

To design, implement, and evaluate a hybrid AI-driven system that integrates a computer vision ensemble model for player recognition and a prompt-engineered large language model for natural language-based statistic generation, thereby enabling accurate player recognition and intuitive access to player-specific data during the final overs of T20i cricket matches.

Research Question 01: Objectives:

1. **Develop an Ensemble Model:** Develop a model that can precisely identify players by utilizing both hybrid spatio-temporal gait analysis and facial recognition. The goal of this model is to capture both the static and dynamic attributes of players by integrating multiple analytical techniques.

2. **Data Collection and Preprocessing:** Compile an extensive dataset consisting of images and video feeds that depict various lighting conditions and field settings that are common in T20i matches. In order to prepare for efficient model training, this data will undergo preprocessing to normalize variations.

3. **Feature Engineering:** Determine the key elements that have the biggest influence on player identification.

4. **Model Training and Optimization:** Train the ensemble model using advanced machine learning and deep learning techniques. Optimize the model for high accuracy and operational efficiency in dynamic conditions.

5. **Validation under Various Conditions:** Analyze the model's performance in different scenarios to make sure it is robust and reliable. This entails testing in various lighting conditions, with players moving in various directions, and at multiple field positions.

Research Question 02: Objectives:

6. **Selection of an Optimal Large Language Model (LLM)**: Choose the most suitable LLM from available options such as ChatGPT, Gemini, and Lama, based on their capabilities to understand and process natural language queries effectively.

7. **Design of a Relational Database**: Build a relational database that is both scalable and reliable for storing comprehensive player statistics. In order to facilitate the retrieval of specific player data as needed, this database should support effective SQL queries.

8. **Implementation of Prompt Engineering Techniques**: Apply and refine prompt engineering techniques to train the selected LLM on accurately interpreting natural language queries. Optimize these prompts to improve the LLM's ability to formulate SQL queries that are both syntactically correct and logically consistent with user queries.

9. **Evaluation of Input Character Length on Effectiveness of User Conditions**: Assess how different user condition configurations and the lengths of input characters affect the relevance and accuracy of the generated statistics.

1.7 Significance of the Research

This study contributes to the developing field of artificial intelligence in sports analytics by addressing a major gap in accurate player recognition and user-friendly data access during cricket matches. The suggested system, CricXpert, employs a novel hybrid technique that integrates facial recognition, spatio-temporal gait analysis, and OCR-based jersey text detection to accurately identify players even in challenging match conditions such as occlusions, low light, and motion blur.

Furthermore, using a prompt-engineered large language model (LLM) to produce SQL queries from natural language input improves the usefulness and accessibility of cricket statistics. This enables users ranging from analysts and experts to casual fans, to interact with the system intuitively, without the need for prior technical knowledge of database structures or query languages. By combining computer vision and natural language processing techniques, this research provides a full, end-to-end solution for player recognition and statistics generation, laying the groundwork for future innovations in multimodal sports analytics platforms.

1.8 Scope of the Research

The scope of this research is confined to developing and evaluating an ensemble-based hybrid recognition system and a language model-driven query generation module tailored for T20i cricket matches. The method is specifically designed to recognize fielding players during the final four overs in day/night matches, which are often challenging to identify due to environmental factors such as insufficient lighting, camera angles, and partial occlusions. The recognition component utilizes a pipeline of deep learning and machine learning models, such

as MTCNN, FaceNet, YOLOv3, ResNet50, GRU, EAST, and Tesseract OCR, to combine facial features, gait patterns, and jersey text data.

Meanwhile, the LLM component is limited to responding to natural language queries with up to three player performance-related conditions, which are translated into SQL queries and executed against a MySQL database. While the system demonstrates promise in terms of accuracy and efficiency, it currently does not handle live streaming integration or real-time decision-making during the match. Furthermore, the database schema and LLM prompts are specific to cricket and may require customization for other sports or domains.

02 LITERATURE REVIEW

2.1 Chapter Overview

Cricket has seen considerable technology developments, notably in T20 Internationals, where day/night matches provide variable lighting conditions that complicate gameplay and player recognition. Fielder performance is critical in the finishing overs, and enhancing fielder detection might boost audience interest as well as player analytics. While facial recognition and gait analysis technologies have advanced, they frequently suffer in dynamic settings such as mentioned before in the previous section.

Furthermore, as data-driven decision-making in sports becomes more prevalent, there is an increased demand for efficient statistical analysis tools. However, present methods need navigating sophisticated user interfaces, which interferes with quick access to critical player information. The study looks at how essential technologies including facial recognition, spatio-temporal gait analysis, ensemble models, and large language models (LLMs) could benefit with player recognition and statistical data retrieval. Despite advances, challenges with accuracy, computing efficiency, and performance persist, and this study assesses existing work and proposes areas for future research to solve these issues in T20i cricket.

2.2 Background to the problem

Cricket is a sport rich in data and tactical complexity, where recognizing players accurately can enhance strategic decision-making, commentary, and audience engagement. In recent years, the use of artificial intelligence (AI) and computer vision in sports analytics has grown rapidly, with applications ranging from ball tracking and action recognition to performance evaluation. However, in the context of T20i cricket, especially during the final overs of a match, recognizing fielding players remains a challenging task. Factors such as dynamic lighting conditions in day/night matches, fast-paced player movement, camera angle variability, and partial occlusion reduce the reliability of conventional vision-based recognition systems. Existing approaches that rely solely on facial features or gait patterns often fail when players are not directly facing the camera or are partially blocked.

Simultaneously, the explosion of available cricket statistics has increased the demand for efficient and intuitive retrieval systems. While databases like ESPN Cricinfo store detailed player records, accessing this information typically requires users to apply filters or manually navigate multiple interfaces. This process is inefficient for real-time analysis or casual queries during live matches. To bridge these gaps, a more robust and multimodal solution is necessary, one that not only improves player recognition accuracy under complex match conditions but also simplifies data access through natural language interfaces. This research addresses these two core problems through a hybrid system that integrates computer vision with a prompt-engineered large language model.

2.3 Related Work

2.3.1 Facial Recognition in Cricket

Facial recognition systems have been widely used in controlled contexts, but they encounter substantial hurdles in dynamic sports such as cricket. In cricket matches, particularly T20

Internationals, players are frequently concealed by other players, equipment, or the surroundings. Furthermore, they are frequently observed from great distances and at non-frontal angles, reducing the efficiency of classic facial recognition techniques. Changes in lighting conditions during day/night matches, when illumination fluctuates quickly as daylight fades or stadium lights fluctuate in intensity, makes it considerably more difficult to accurately recognise players (Mahmood et al. 2015), (Haq et al. 2024).

Given the fast-paced and dynamic nature of cricket, where players constantly move and camera angles vary, maintaining strong recognition accuracy is a significant challenge. The success of facial recognition technology in cricket may tremendously enhance the audience experience by allowing for player identification and analytics. However, existing face recognition models struggle to handle occlusion, illumination changes, and non-frontal facial angles, rendering them unsuitable for live sports analytics.

2.3.1.1 Existing Work

A range of facial recognition systems have been investigated to handle the issues given by dynamic situations. Early research, such as (Mahmood et al. 2015), proposed an autonomous face identification system for cricket that used machine learning techniques like AdaBoost. While their system worked well in controlled environments, it lost accuracy dramatically when used in real-world cricket matches due to frequent occlusions and non-frontal face angles. To solve the issue of non-frontal angles and occlusion, (Zhang et al. 2020) used deep learning techniques, notably convolutional neural networks (CNNs), with a multi-camera configuration to track and identify players. Even when participants were not directly facing the camera, the algorithm was able to maintain recognition accuracy by recording several angles of their faces.

Although the technology improved overall accuracy, the requirement of multi-camera setups for processing remained a constraint, limiting its practical application in live cricket broadcasts. In response to illumination issues, (Haq et al. 2024) investigated the use of augmented reality (AR) overlays in conjunction with face recognition. By combining augmented reality into live cricket broadcasts, their method enabled player identification even in low-light settings. However, while this technology increased fan involvement, inadequate illumination in night events remained a substantial barrier to successful face identification.

2.3.1.2 Technological Review

Facial recognition in sports analytics has improved in conjunction with advances in deep learning, notably the usage of CNNs, which can process massive volumes of video data and extract valuable facial features. (Zhang et al. 2020) used CNNs in their multi-camera system to improve player tracking and recognition. The CNN architecture enabled the system to learn complex patterns from video streams, resulting in more accurate identification and recognition of players' faces, even when obstructed or seen from non-frontal angles. (Haq et al. 2024) proposed the idea of combining facial recognition with augmented reality (AR) overlays to improve identification during cricket matches. AR technology enables the placement of virtual markers or labels on player photos, allowing spectators to better follow the game. While AR provides benefit to spectators, it requires a highly precise facial recognition system to function properly, which can be hampered by occlusion, low lighting, or long-distance camera images.

In addition to CNNs and AR, multi-camera systems have proven useful for solving occlusion and angle-related difficulties. (Zhang et al. 2020) emphasised the benefits of collecting

numerous perspectives of players during matches, which increases the accuracy of face identification and recognition. However, multi-camera systems have higher processing needs, which complicates implementation. Furthermore, recent studies have investigated hybrid models that combine face recognition with additional modalities, such as gait analysis, in order to enhance recognition accuracy. This multimodal technique is increasingly being used in sports analytics because it allows more accurate identification in dynamic contexts. However, the computational complexity of processing many data streams is still a considerable barrier.

2.3.1.3 Evaluation and Benchmarking

Facial recognition systems in sports analytics have been evaluated using various important performance parameters, including accuracy in recognising athletes, robustness to occlusion and changes in lighting conditions, and computing efficiency in real-time applications. (Mahmood et al. 2015) claimed a 78% accuracy rate for their facial recognition system in controlled situations. However, as previously mentioned this accuracy fell dramatically when evaluated in real-world cricket scenarios, where occlusion and non-frontal face views regularly interfered with recognition. This decrease in accuracy demonstrates the limits of typical machine learning algorithms in dynamic sporting contexts. (Zhang et al. 2020) tested their multi-camera CNN system and obtained an 89% accuracy in multi-angle settings. The implementation of CNNs enabled the system to retain excellent accuracy even when faces were partially obscured or viewed from non-frontal perspectives.

However, the system's performance was hampered by the computing constraints of processing several video streams at the same time, making it unsuitable for live cricket matches. (Haq et al. 2024) evaluated their AR-enhanced facial recognition system and found 82% accuracy in well-lit areas. However, the system's effectiveness plummeted to 67% in low-light conditions emphasising the persistent problem of adapting face recognition algorithms to constantly changing lighting conditions. The incorporation of augmented reality improved the spectator experience, although additional development was necessary to solve illumination restrictions.

These studies demonstrate that, while facial recognition technology has made great advances in improving player identification in sports, there are still major obstacles to adapting these algorithms for usage in dynamic contexts such as cricket. Future research should focus on increasing the computational efficiency and resilience of facial recognition systems, especially in the presence of occlusion, illumination fluctuation, and non-frontal facial orientations.

2.3.2 Spatio-Temporal Gait Analysis in Dynamic Environments

Gait recognition is becoming increasingly used in sports analytics as a means to identify players based on their walking or running patterns. In cricket, when players' faces may be concealed or not visible, gait recognition offers an alternate method of identification. However, dynamic situations such as cricket pitches provide obstacles for gait identification algorithms, such as quick player movements, occlusion by other players or equipment, and changing camera angles. These difficulties are amplified by fluctuations in movement rates, body postures, and camera angles during the game. Furthermore, other disciplines have shown that machine learning classifiers may be used in conjunction with deep learning architectures to improve classification efficiency and accuracy. (Özyurt 2020) effectively used efficient deep feature selection approaches employing fused deep learning architectures in the context of

remote sensing image identification. This demonstrates the potential for feature optimization in challenging circumstances.

Drawing on previous work, this research seeks to combine deep learning-based gait identification with feature optimization approaches to efficiently manage movement and occlusion variability in dynamic cricket contexts. Pose estimation offers value in recognizing player posture and motions, potentially augmenting gait analysis by gathering extra movement data, particularly in fast-paced settings where players change direction quickly or have obstructed limbs. Pose estimation may increase knowledge of player biomechanics and movement dynamics. In cricket, a combined method of gait analysis, posture estimation, and multimodal recognition (for example, integrating gait and facial recognition) has the potential to improve player identification in challenging settings such as large crowds or dimly lit surroundings. However, these approaches have to reconcile computational efficiency and performance.

2.3.2.1 Existing Work

Early gait identification research concentrated on silhouette-based approaches, in which the contour of a player's body was retrieved from video frames and analysed for distinct walking patterns. (Kale et al. 2004) developed one of the first silhouette-based algorithms for gait identification, proving that people can be identified based on the unique features of their motions. However, this method was restricted in dynamic situations such as cricket, where quick motions and occlusion frequently interrupted silhouette tracking. To overcome these constraints, (Zhen et al. 2020) introduced spatio-temporal convolutional neural networks (ST-CNNs) that follow gait across many frames, capturing the temporal dynamics of player movements. Their approach performed better in detecting players in dynamic situations, but occlusion from other players and objects remained an issue. (Gul et al. 2021) built on this by creating a deep learning-based multi-view system that captures a player's gait from various camera angles, allowing the system to reconstruct a player's motions even when sections of the body were obscured, resulting in much improved accuracy.

(Kibriya et al. 2021) and (Kibriya et al. 2022) expanded on the hybrid model method by demonstrating the use of deep learning for feature extraction in conjunction with classical machine learning classifiers for classification tasks such as brain tumour diagnosis. Their findings showed that combining Support Vector Machines (SVM) with deep learning features may greatly enhance classification accuracy while remaining computationally efficient. This technique inspired to be applied to this study, which uses deep feature extraction and SVM classifiers to increase the robustness of gait identification in cricket. This hybrid technique also keeps computing complexity in check for sports analytics. (Maity et al. 2021) suggested a multimodal strategy that combines gait analysis and facial recognition to improve player identification in low-light and obscured conditions. When facial recognition proved unreliable, the system could move to gait analysis as a result of this hybrid technique.

2.3.2.2 Technological Review

The combination of deep learning and spatio-temporal networks has considerably increased gait identification technologies. (Zhen et al. 2020) used spatio-temporal convolutional neural networks (ST-CNNs) to examine both the spatial structure of a player's body and the temporal progress of their motions. This strategy enhanced the system's capacity to follow fast-moving players, even while some body portions were obscured. ST-CNNs, on the other hand, are

computationally costly, particularly when tracking numerous players at the same time. (Gul et al. 2021) developed a multi-view gait recognition system that addresses the limitations of single-view methods. By merging gait data from numerous camera angles, the algorithm was able to account for occlusion and varied camera angles, resulting in more accurate player identification. However, analysing the data from several cameras remained a substantial difficulty. (Özyurt 2020) developed a deep feature selection strategy for remote sensing picture recognition using fused deep learning architectures. This feature selection and optimization approach may be applied to spatio-temporal gait recognition to guarantee that only the most important features are analysed, lowering computational overhead and increasing overall system efficiency. This notion has been implemented into the suggested model to optimise feature selection for gait identification, ensuring that key spatial and temporal characteristics are kept without overloading the system with unnecessary data.

Other technologies, such as Long Short-Term Memory (LSTM) networks, have been used to describe temporal relationships in gait sequences. (Wang et al. 2019) employed LSTMs to capture the temporal aspects of gait patterns, hence boosting recognition accuracy in dynamic sports contexts. LSTMs, like other techniques, need substantial computer power, making its deployment in real-time environment more challenging. Furthermore, Human Pose Estimation has been used to supplement gait research, particularly for collecting joint motions and analysing player biomechanics.

2.3.2.3 Evaluation and Benchmarking

Gait recognition systems are assessed on their capacity to follow and identify people in dynamic surroundings, as well as their resistance to occlusion and changing camera angles. Performance indicators frequently include recognition accuracy, computing efficiency, and real-time capabilities. (Kale et al. 2004) claimed a 78% accuracy for their silhouette-based gait identification system in controlled situations. However, in dynamic sports environments like as cricket, the system's performance suffered dramatically due to occlusion and rapid player movement. This demonstrated the limitations of classic silhouette-based approaches in real-world scenarios. (Zhen et al. 2020) showed an 84% accuracy for their spatio-temporal convolutional network (ST-CNN), outperforming previous silhouette-based techniques by monitoring players over many frames.

However, occlusion and rapid movements remained major concerns, particularly in crowded parts of the cricket ground. (Gul et al. 2021) evaluated their multi-view gait recognition system in controlled situations and achieved an accuracy of 91%. By combining numerous camera views, the system was able to recreate gait patterns even when sections of the body were obscured, considerably enhancing recognition accuracy. However, the high computing cost of processing data from several cameras in real time limits its use for live sports broadcasts. (Kibriya et al. 2021) and (Kibriya et al. 2022) investigated the performance of hybrid classification systems that included deep feature extraction followed by SVM classification and found considerable increases in both accuracy in the 90% barrier and processing efficiency. This hybrid method to gait detection in cricket is likely to produce comparable results, notably by enhancing classification accuracy under difficult settings while keeping computing efficiency appropriate for real-time applications.

(Wang et al. 2019) evaluated LSTM-based gait recognition systems and found an 87% accuracy in recognising players in dynamic situations. However, the system struggled with performance due to the high computational cost of handling temporal relationships in the data.

Finally, (Maity et al. 2021) observed better accuracy when integrating gait analysis and facial recognition, with a combined accuracy of 89%. This multimodal technique enabled the system to compensate for occlusions or bad lighting conditions, which would have normally disrupted facial recognition. The system's scalability was restricted by the increased computing complexity of processing two modalities at the same time.

2.3.3 Large Language Models (LLMs) for Statistic Generation

The increasing focus on data-driven decision-making in sports analytics, particularly in cricket, has created a greater demand for systems that can access individual player information quickly and efficiently. Traditional sports data systems frequently require users to navigate complex interfaces and apply several filters to obtain useful information, which can be complicated, particularly during live events. Large Language Models (LLMs), such as GPT-3.5, have emerged as a promising approach for streamlining this procedure by allowing users to get individual player data using natural language queries. However, the complexity of sports datasets, particularly in cricket, where player data is spread over numerous tables and contains complicated interactions, poses considerable issues. Existing systems frequently struggle with multi-table queries, nested conditions, and the necessity for data retrieval, particularly in live matches when speed and accuracy are critical. LLMs give a more user-friendly interface for querying big datasets, but they still need to enhance their ability to handle complicated queries, optimise speed, and scale.

2.3.3.1 Existing Work

The utilization of Large Language Models to generate structured queries from natural language inputs is still a relatively recent development in sports analytics. (Shi et al. 2024) demonstrated the application of GPT-3.5 for SQL query generation in sports analytics, considerably simplifying the process of retrieving player information. By enabling users to input natural language queries, their system reduced the complexity of interacting with large sports datasets. However, they discovered that while the system worked well for simple, single-table searches, it struggled with multi-table connections and nested conditions, which are frequent in sports data.

(Chopra and Azam 2024) built on this by incorporating a classification-based table selection mechanism that guided the LLM to the appropriate tables in the database. This strategy enhanced SQL generation accuracy by categorizing user inputs into preset query categories, allowing the system to handle more complicated queries. However, the rising computing needs of managing massive datasets continued to hinder the system's performance in real sports situations. (Hong et al. 2024) used schema-specific prompt engineering to increase SQL generating accuracy. By incorporating schema-specific information in the prompt, their system improved its understanding of the links between distinct tables, resulting in more relevant and accurate SQL queries. However, this solution necessitated manual schema customization for each sports database, limiting its scalability across several sports datasets.

(Singh et al. 2023) introduced an adaptive learning system for LLMs that generates SQL queries while continually learning from user feedback. This approach improved the accuracy of SQL creation over time by optimizing prompts and enhancing the model based on typical user input. However, this technique necessitated significant user input and training time, limiting its immediate effectiveness in live sports contexts. (Shi et al. 2024) explored how reinforcement learning may improve LLM performance by fine-tuning answers based on input

from successful and unsuccessful SQL queries. While this strategy improved accuracy for complicated queries, it also needed a large amount of processing resources and training time, making it difficult to implement in real time.

2.3.3.2 Technological Review

The usage of large Language Models (LLMs), such as GPT-3.5, for natural language query generation is based on deep learning architectures, which allow models to handle vast volumes of unstructured text and produce structured outputs such as SQL queries. (Shi et al. 2024) used GPT-3.5 for sports analytics, allowing users to produce SQL queries using natural language inputs. The model was refined to comprehend sports-specific queries and translate them into precise SQL instructions, making it easier to get player information.

To solve the difficulties of dealing with complicated queries, (Chopra and Azam 2024) suggested a classification-based table selection approach in which natural language inputs are sorted into predetermined categories, pointing the LLM to the right tables in the database. This solution increased the system's capacity to handle multi-table queries and return more accurate results. However, the system required significant computer capacity to evaluate big sports statistics in real time. (Hong et al. 2024) proposed schema-specific prompt engineering to increase SQL generation accuracy by incorporating schema information in prompts. This allowed the LLM to better grasp the links between tables, leading to more appropriate SQL searches. The disadvantage was that each dataset required manual schema customization, limiting its applicability across multiple sports databases.

(Singh et al. 2023) developed an adaptive architecture in which LLMs learnt from user interactions over time, allowing the model to improve query production depending on feedback. Although beneficial in improving accuracy, this system required a feedback loop and took a long time to train, making it unsuitable for real-time applications. Reinforcement learning, as investigated by (Shi et al. 2024), was used in LLMs to optimise SQL query creation based on successful interactions. By incorporating input, the model was able to evolve and enhance its capacity to construct complicated SQL queries, particularly those containing nested conditions or numerous tables. However, this strategy necessitated large computational resources and more training time, affecting performance.

2.3.3.3 Evaluation and Benchmarking

LLM-based query generation systems are frequently evaluated based on their ability to produce SQL queries reliably, manage complex multi-table connections, and perform well in real-time scenarios. Key performance indicators include query accuracy, processing efficiency, and scalability. (Shi et al. 2024) evaluated GPT-3.5 for SQL generation in sports analytics and discovered an average accuracy of 85% for basic queries. However, while doing multi-table queries or dealing with complicated circumstances, the model's performance fell to 65%, highlighting the need for more advanced prompt engineering strategies to boost efficiency in real-world sports data systems.

(Chopra and Azam 2024) evaluated their classification-based table selection approach, which increased SQL query generation accuracy upto 90% in sports analytics systems. This strategy minimised mistakes in multi-table searches, but the computation costs were a major worry. (Hong et al. 2024) used schema-specific prompt engineering to improve query generation, attaining 92% accuracy on complicated, multi-table datasets. While their method

increased query relevancy, it necessitated manual schema changes for each database, limiting versatility to other sports datasets.(Singh et al. 2023) discovered that using an adaptive learning architecture resulted in an improvement in LLM accuracy over time, reaching 89% accuracy after incorporating user feedback.

The system's ability to learn from experiences allowed it to better understand sports-specific language and challenging queries. However, the reliance on human interactions, as well as the time required for training, created significant barriers. Reinforcement learning-based approaches, as examined by (Shi et al. 2024), have shown promise for improving SQL generation accuracy, particularly for complex queries. The LLM achieved 88% accuracy by learning from both successful and unsuccessful query attempts while adjusting and improving its responses. Despite the breakthroughs, computational cost and the time required for training prevented its immediate adoption in live sports broadcasts.

2.3.4 Comparison Table of Relevant Work

Table 1: Comparison Table of Relevant Work

Paper	Methodology	Results	Limitations
Mahmood et al. (2015)	AdaBoost and other machine learning algorithms were used to distinguish faces automatically in cricket scenarios.	Achieved 78% accuracy in controlled situations, but only 68% in real-world circumstances due to occlusions and non-frontal views.	Performance deteriorated under conditions with occlusion and non-frontal facial views. Struggled with scalability in cricket settings.
Haq et al. (2024)	Facial recognition was used with augmented reality (AR) overlays to improve live cricket broadcasts.	Integrated AR effectively increased spectator engagement by giving real-time player information. In well-lit situations, accuracy was 82%; in low-light settings, it dropped to 67%.	Significant decline in performance during low-light conditions, such as night matches.
Zhang et al. (2020)	Convolutional neural networks (CNNs) were used with a multi-camera configuration to achieve deep facial recognition in cricket.	Achieved an accuracy of 89% by capturing multiple angles, with a 5.4% reduction in false positives. Robustness was improved, but real-time scalability was limited.	Scalability issues due to the computational demands of processing multiple video feeds concurrently.
Özyurt (2020)	For remote sensing picture categorisation, introduced deep feature selection utilising fused deep learning architectures.	Demonstrated a classification accuracy of 94.7% in remote sensing tasks, significantly improving computational efficiency through feature selection.	Though highly effective in remote sensing, the method was not specifically designed for the challenges of dynamic sports environments. Adaptation would require domain-specific modifications.

Paper	Methodology	Results	Limitations
Kale et al. (2004)	Proposed silhouette-based gait detection method extracts distinctive walking patterns for player identification.	Accuracy was 78% in controlled situations, but fell to 65% in dynamic cricket scenarios because to occlusion and quick movement.	Poor performance in dynamic settings due to occlusion and fast player movement, highlighting the need for more advanced tracking methods.
Zhen et al. (2020)	Developed spatio-temporal convolutional networks (ST-CNNs) to capture temporal dynamics for gait analysis.	Reported 84% accuracy, 6% greater than previous silhouette-based techniques. The system recorded temporal dynamics over several frames with great success.	High computational requirements made the model impractical for widespread deployment, especially under crowded and occluded scenarios.
Gul et al. (2021)	Employed a deep learning-based multi-view system for multi-angle gait recognition, reconstructing occluded player movements.	Achieved 91% accuracy in controlled experiments. Reduced false negatives by 8% through the use of multi-view reconstruction.	The model's scalability was limited due to the computational cost of processing data from multiple camera angles simultaneously.
Kibriya et al. (2021)	Combined convolutional neural networks (CNNs) with support vector machines (SVMs) for brain tumor classification.	Achieved a classification accuracy of 92%, with an F1 score of 0.89, highlighting the robustness in medical imaging.	Limited relevance to sports analytics, with focus on medical imaging applications. Domain adaptation is needed for player identification contexts.
Kibriya et al. (2022)	Used deep feature fusion combined with machine learning classifiers for brain tumor classification.	Achieved a classification accuracy of 95.1%, with significant improvements in feature extraction capabilities, leading to reduced false positives.	The applicability to sports was constrained by its focus on medical contexts, necessitating adaptations to dynamic sports environments.
Maity et al. (2021)	Implemented a multimodal system combining facial recognition with gait analysis for improved player identification.	Achieved a combined accuracy of 89%, which included a 9% improvement in low-light scenarios compared to single-modality systems.	Increased computational complexity restricted the scalability, particularly for use in sports broadcasts.
Wang et al. (2019)	Applied Long Short-Term Memory (LSTM) networks to model temporal dependencies in gait recognition.	Reported an accuracy of 87%, capturing temporal dependencies effectively, with an 11% reduction in false negatives in dynamic sports environments.	High computational power requirement made widespread implementation challenging.

Paper	Methodology	Results	Limitations
Shi et al. (2024)	Leveraged GPT-3.5 for SQL generation from natural language inputs to facilitate sports analytics queries.	Achieved an average accuracy of 85% for simple queries, with a mean response time of 3.4 seconds for queries. Performance dropped for multi-table and nested queries.	Performance dropped to 65% for multi-table and complex queries, suggesting the need for more advanced prompt engineering and optimization for complex databases.
Chopra and Azam (2024)	Introduced a classification-based table selection approach to enhance LLM-generated SQL accuracy.	Improved SQL query generation accuracy to 90%, reducing query time by 20% through classification-based guidance in sports datasets.	Required substantial computational resources, making it challenging to use for larger datasets. Scalability issues persisted across large datasets.
Hong et al. (2024)	Developed schema-specific prompt engineering to improve the performance of LLMs for SQL generation in sports data contexts.	Achieved an accuracy of 92% for generating SQL queries involving multiple complex relationships, reducing query error rate by 15%.	Required manual tuning of the schema, which reduced flexibility across different sports datasets and limited generalizability.
Singh et al. (2023)	Implemented an adaptive learning framework that allowed LLMs to refine SQL query generation through continuous user interaction.	Improved LLM accuracy to 89% after adaptive training based on user feedback, reducing error rates over time by 12%.	Depended on extensive user interaction and training, reducing its immediate applicability in general sports environments. Scalability for widespread deployment remained an issue.

2.3.4 Summary

This literature review delves into cutting-edge technology and approaches that are transforming player recognition and sports analytics, with a special emphasis on the fast-paced and unpredictable environment of T20i cricket. The study included facial recognition, spatio-temporal gait analysis, multimodal recognition systems, large language models (LLMs) for natural language query generation, and human posture estimation, all of which have distinct strengths and limitations. Facial recognition works well in controlled environments but struggles with the quick motions, varying illumination, and occlusions of cricket matches. Gait analysis and spatio-temporal approaches offer a viable alternative for recording distinct player motions, although they still face challenges with occlusion and speed. The integration of

multimodal systems leveraging face, gait, and pose analysis, emerges as a viable approach, although computational complexity remains a challenge.

Meanwhile, LLMs have impacted data retrieval in sports analytics by creating SQL queries from natural language, but the complexities of cricket datasets pose scalability and accuracy challenges that must be addressed. This review identified how a novel hybrid approach could combine the best of these technologies: fusing spatio-temporal analysis with deep learning, optimising with machine learning classifiers, and using LLMs for intelligent data extraction all aimed at addressing the complexities of T20i cricket. This ambitious hybrid system promises not only to improve the accuracy and speed of player detection, but also to change the spectator experience and give actionable statistics in a high-demand setting.

03 METHODOLOGY

3.1 Chapter Overview

The methodology section is essential for providing a detailed overview of the techniques, strategies, and procedures followed during the research process. It ensures that the study can be repeated and verified for validity and reliability. The proposed strategy aims to achieve two key objectives: effective cricket player recognition using a hybrid computer vision approach and the generation of player-specific statistics using a large language model (LLM).

This section outlines the processes required to acquire, preprocess, and analyze data, followed by a detailed discussion of the models and methodologies utilized. The goal is to offer a clear explanation of how the hybrid face recognition and gait analysis models were combined, as well as how powerful machine learning classifiers were employed to achieve high accuracy. This section also discusses the experimental design, data sources, assessment criteria, and validation procedures to verify the reliability and generalizability of the results.

3.2 Research Design

This study uses an experimental research design and quantitative approaches to attain its goals. The strategy is on creating, optimizing, and testing machine learning and deep learning models for player recognition and data retrieval in the context of T20i cricket. The study strategy is exploratory, with the goal of developing an integrated system for recognising cricket players in dynamic settings and providing player statistics using natural language queries.

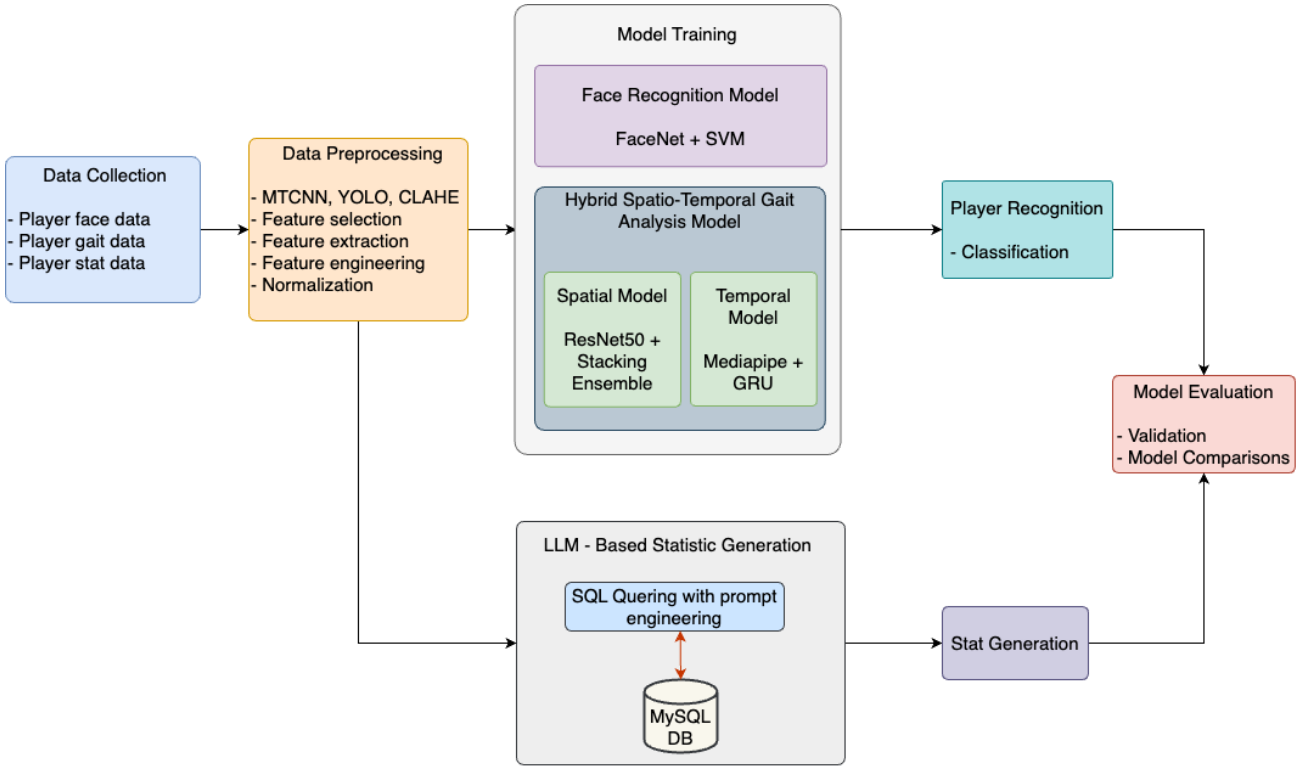


Figure 1: High Level Design of the Proposed System for T20i Cricket Player Recognition and Stat Generation.

The research process is organized into four essential stages: data collection, preprocessing, model creation, and assessment (Figure 1). Each stage is critical for ensuring the final hybrid system's resilience and durability. The design also incorporates both computer vision techniques (for player recognition) and natural language processing (to generate SQL queries to acquire player statistics), resulting in a mixed methods approach that combines several data sources and analytic approaches. This study intends to overcome the constraints of employing individual recognition algorithms, particularly in dynamic situations like cricket matches, by combining face recognition, gait analysis, and text-based identification.

3.3 Data Collection Methods

The data utilized in this study were carefully selected to create an efficient hybrid model for identifying cricket players. The data is divided into three types: face data, gait data, and player statistics. Each dataset improves distinct aspects of the player recognition process, including facial recognition, gait analysis, and statistical retrieval. Data was collected using both customized and publicly available sources.

- **Primary Data**: Facial images and gait patterns of cricket players were extracted from video footage using direct observations and customized data gathering.

- **Secondary Data**: This data was obtained from web sources such as ESPN Cricinfo for player statistics and Google Images for face data. Academic literature and online cricket broadcasts provided raw video clips for gait analysis.

3.3.1 Data Sources

- **Player Facial Data**: Images were extracted from cricket match video footage sourced from sources like Google Images. These images were chosen to provide a variety of lighting and angle settings, replicating real-world match scenarios. They were used to develop and test the face recognition model.

- **Player Gait Data**: This collection includes video snippets of players walking or running during matches. These segments utilize spatio-temporal properties to record unique movement patterns. The video samples were obtained from online cricket broadcasts,YouTube and ICC assuring a variety of match circumstances, such as camera angles and lighting.

- **Player Statistics Data**: This dataset includes statistical information for creating player-specific statistics in an LLM. ESPN Cricinfo provided the statistics, including performance metrics and match records. This dataset was used to create player statistics using natural language queries.

3.3.2 Dataset

To address the lack of publicly available datasets for cricket player recognition under dynamic match situations, a novel dataset was curated expressly for the proposed method. The dataset includes annotated images of six cricket players, each with 130-150 images taken from various vantage points and match conditions.

Figure 2: Sample annotated player images from the CricXpert dataset.

The data collecting process included selecting high-resolution footage from T20i matches to ensure a diverse representation of camera angles, lighting conditions, occlusions, and player movement. Frames were extracted from match recordings at a consistent sampling rate to balance image diversity and homogeneity. Figure 2 above illustrates representative samples from the dataset used in the evaluation.

Expert Evaluation and Validation of Dataset Quality

To assess the quality and relevance of the dataset, a structured expert validation was conducted involving three independent professionals referred to as Subject 1, Subject 2, and Subject 3 with backgrounds in cricket coaching, analytics, and applied data science. Inclusion criteria required experience in sports analytics, availability for a live demonstration, and completion of a structured evaluation form. Individuals without cricket domain expertise or who declined formal feedback were excluded.

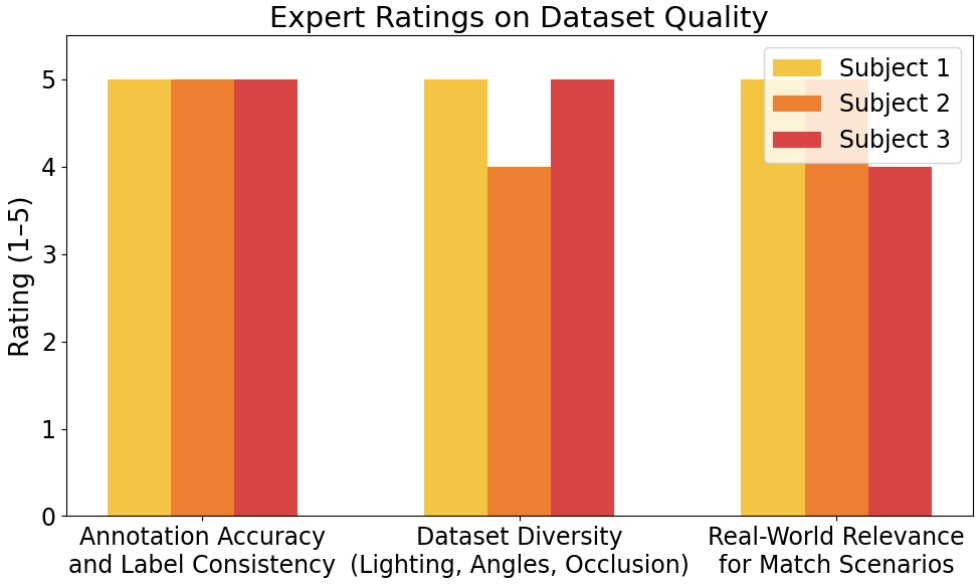


Figure 3: Expert Ratings on Dataset Quality.

Each expert participated in a live demo session, followed by a standardized Google Form questionnaire. Using a 5-point Likert scale (1 = Poor, 5 = Excellent), they rated key dataset attributes including annotation accuracy, diversity of visual conditions, and real-world applicability(Figure 3,4). Since no personal or sensitive data were collected, ethical approval from a review board was not required.

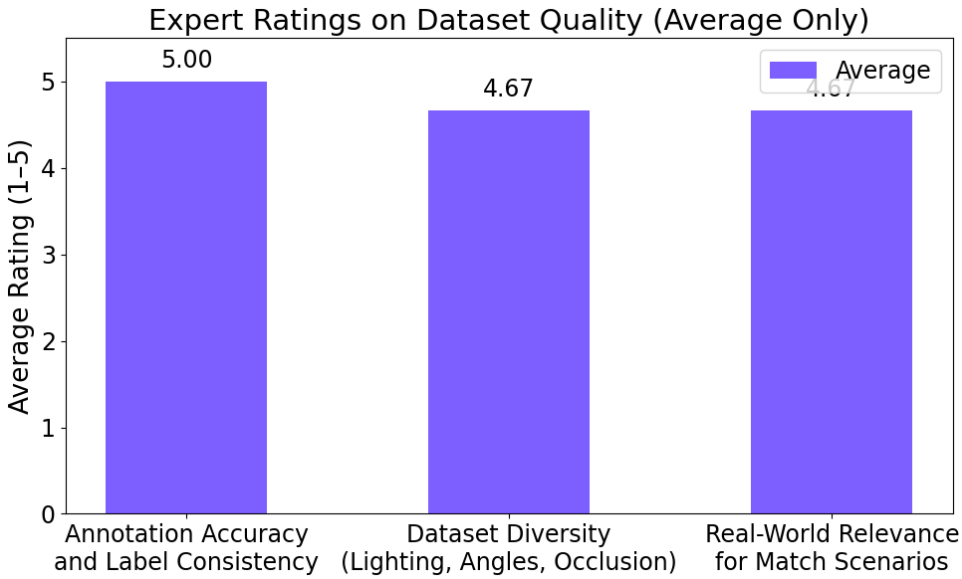


Figure 4: Expert Ratings on Dataset Quality (Average).

3.4 Data Preprocessing

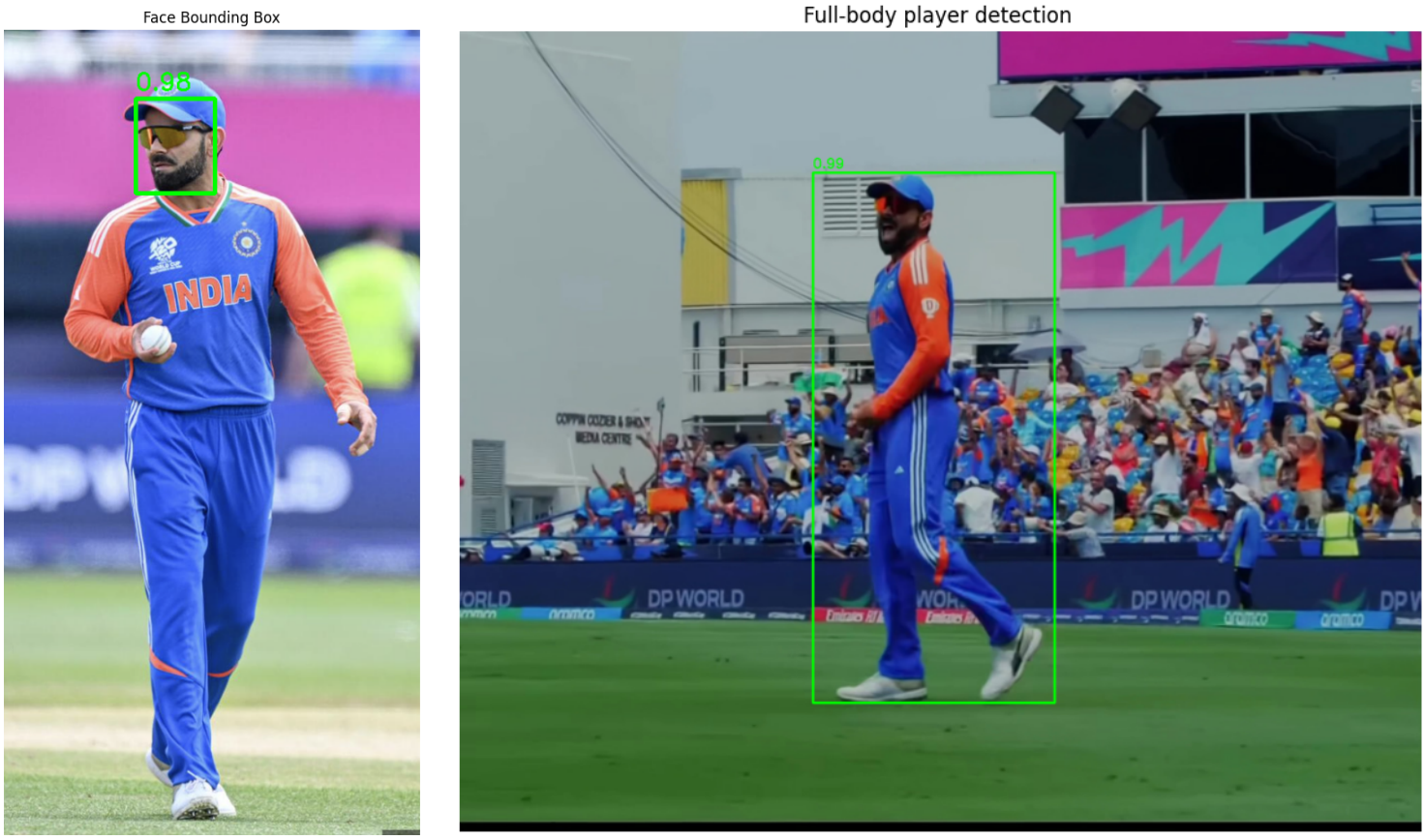
To ensure that the data was appropriate for model training, a number of preprocessing steps were performed on each dataset. These stages sought to clean, normalize, and extract significant characteristics from the data in order to improve the hybrid player recognition model's accuracy and efficiency.

3.4.1 Data Cleaning

- **Player Facial Data**: Images from video clips and Google Images may contain background noise and not related things. Cropping techniques were utilized to isolate the player's face, and low-quality photos (such as significant blurring or occlusion) were removed. YOLOv3 was first employed for face detection, however owing to performance concerns, MTCNN was eventually selected for its greater accuracy in facial area detection as shown in figure 5 (a).

- **Player Gait Data**: Raw video clips showed somewhat obscured or out of focus players. These frames were removed to ensure that only the best segments were utilized for gait analysis. YOLOv3 was used to find the biggest vertical bounding box, successfully recognising players while avoiding background noise during walking or sprinting sequences as displayed in figure 5 (b).

- **Player Statistics Data**: To ensure data quality, inconsistent or incomplete player statistics were removed. Data entries that included missing values were removed also.



(a) (b)

Figure 5: Player localization using visual detection models. (a) Facial region is identified using MTCNN, highlighting the face. (b) Full-body player detection is performed using YOLOv3, showing the largest vertical bounding box.

3.4.2 Data Transformation

- **Normalization**: Continuous characteristics were normalized using min-max scaling to guarantee consistent scale. This includes features like joint angles, velocities, and face embeddings etc.

- **Feature Extraction**:

    - **Player Facial Data**: Facial embeddings of players have been obtained using Google's FaceNet model, which converts images into 128-dimensional vectors. The identified faces were retrieved using MTCNN and sent to FaceNet for embedding creation.

    - **Player Gait Data**: The hybrid model used player gait data with both spatial and temporal components. CLAHE (Contrast Limited Adaptive Histogram Equalisation) was used to improve picture quality as shown in figure 6 before ResNet50 was used to extract features. Each cropped player image was resized to **224×224×3** to match the input requirement of ResNet50. The model was used with its top classification layers removed, and the **2048-dimensional output** from the

global average pooling layer served as the spatial feature vector. These embeddings were used for downstream classification by the ensemble classifiers.

Figure 6: Original vs. CLAHE-enhanced cropped player image

In the temporal model, after recognising joints (figure 7) with Google's Mediapipe library, temporal features such as step length, step velocity, joint angles, joint acceleration, angular velocity, and hip displacement were created for the temporal model. YOLOv3 was utilized for initial player detection, followed by Mediapipe for joint detection.

Figure 7: Player pose estimation using MediaPipe with key landmarks and skeletal connections overlaid

- **OCR Text Detection**: Raw video frames frequently featured jersey text that appeared in motion blur, low lighting, or across fractured portions (for example, the name and number were recognized individually). To improve text clarity for OCR, preprocessing steps were performed on the detected text sections. First, the EAST text detector was utilized to identify bounding boxes around the text. Where numerous boxes occurred for name and number, they were combined into a single region of interest. The merged text region was then converted to grayscale and subjected to a median blur filter to minimize noise while maintaining edges. This preprocessing enhanced OCR effectiveness in demanding visual situations, laying the groundwork for the final recognition phase with Tesseract.

3.4.3 Feature Selection

- **Facial Data**: Reduced dimensionality and improved model performance by excluding low variance features from facial data. To address the overfitting reported in the early stages of SVC classifiers, feature selection was performed via cross-validation and hyperparameter adjustment.

- **Gait Data**: Correlation analysis was used to remove duplicate characteristics from the gait data and train the temporal model with only the most important features. The implementation of Mediapipe enabled accurate joint detection and feature engineering, such as step length, joint angles, hip displacement, and other created features.

3.5 Model Selection

The necessity for a robust and accurate system capable of recognising cricket players using many modalities, such as facial characteristics, gait analysis, and textual information on jerseys, drove the selection of models and algorithms for this work. The models were chosen after weighing various choices and doing significant experimentation to discover the best successful strategy.

3.5.1 Facial Recognition Model

- **Initial Approach**: The model was developed using YOLO for face detection and FaceNet and Vision Transformer (ViT) for prediction. However, this combination produced unsatisfactory results due to accuracy, overfitting concerns, and computational expense.

- **Final Model**: The final model used MTCNN (Multi-Task Cascaded Convolutional Neural Networks) for face identification and Google's FaceNet for feature extraction. MTCNN was superior at facial detection, and FaceNet created high-quality facial embeddings. The retrieved features were classified by an SVM, which dramatically increased accuracy while reducing overfitting difficulties. The complete code for MTCNN-based face detection and FaceNet embedding generation is included in Appendix B.

3.5.2 Hybrid Spatio-Temporal Model

- **Spatial Model**: The hybrid model's spatial component used YOLOv3 to detect players and reduce background noise using the biggest vertical bounding box. CLAHE was used to improve image quality. The first objective was to categorize players using deep learning

models, which prompted experiments with several architectures such as DenseNet, EfficientNet, Inception, MobileNet, VGG16, Xception, and ResNet50. However, overfitting remained despite early pausing and parameter adjustment. This led to the hypothesis that the overfitting was caused by the classification layers in these architectures. To solve this, a fusion strategy was taken: ResNet50 was utilized only for feature extraction, while SVM and KNN were employed for classification. The extracted features were classified using a stacking ensemble of SVM, KNN, and a final Logistic Regression layer. Hyperparameter tuning was done to all components, which resulted in effective overfitting mitigation and better model performance. ViT models were also investigated, but their computing requirements rendered them unsuitable for the study.

- **Temporal Model**: The temporal model analyzed gait patterns. Google's Mediapipe was used to detect player joints, from which temporal features such as step length, joint angles, velocity, joint acceleration, angular velocity, and hip displacement were calculated. These features were inputted into a GRU (Gated Recurrent Unit) model for temporal analysis. The GRU was chosen over the LSTM since it was less complicated and performed better on this task.

  Temporal Feature Engineering: After extracting pose landmarks for each frame with MediaPipe (as illustrated in Figure 7), a set of engineered temporal features are computed to represent each player's movement dynamics. This includes:

  - **Step Length**: Calculated as the Euclidean distance between the left and right ankle landmarks.
  - **Joint Velocities**: Computed across consecutive frames for key joints (e.g., left and right ankles).
  - **Joint Angles**: Calculated for leg segments (hip-knee-ankle) using vector-based trigonometric functions.
  - **Joint Accelerations**: Determined by measuring the change in joint velocities between adjacent frames.
  - **Angular Velocity**: Represents the rate of change of joint angles, indicating dynamic limb movement.
  - **Hip Displacement**: Measures the horizontal translation of the pelvis region across frames using midpoint changes of left and right hips.

  These features are computed frame-by-frame and organized into fixed-length sequences (15 frames) that serve as input for the GRU-based temporal model. This structured representation allows the model to learn unique motion patterns for each player. Appendix B contains the precise implementation logic for computing these temporal features, such as step length, joint velocities, joint angles, accelerations, angular velocities, and hip displacement, as well as the complete source code. This enables the reproducibility and technical validation of the temporal model's feature engineering process. Appendix B also provides the ResNet50 feature extraction logic and the GRU-based temporal model implementation.

3.5.3 OCR Text Detection and Recognition

An OCR-based approach was used to identify players by recognizing jersey names and numbers, which are generally the most obvious visual identifiers during live match footage. However, contextual factors such as motion blur, partial occlusions, fluctuating lighting, and

fractured text sections made this task more challenging. Initially, the system used Tesseract OCR directly on cropped player segments, however the results were poor due to isolated or incomplete detections. As a result, a more robust, multi-stage pipeline was implemented:

- **Text Detection:** The EAST (Efficient and Accurate Scene Text Detector) model was integrated to locate text regions in the frame. EAST proved effective in detecting partial or distorted characters, even under motion.

- **Bounding Box Merging & Preprocessing:** The detected bounding boxes for the name and number (if separated) were combined into a single region of interest. This region was then converted to grayscale and processed with a median blur to reduce background noise and improve character clarity.

- **Text Recognition:** The preprocessed region was then passed to Tesseract OCR for final text extraction.

This final pipeline significantly increased recognition accuracy even in challenging match situations. Additionally, OCR recognition was the principal identifier in the player recognition sequence. If the text-based method fails or produces low confidence scores, the system will immediately fall back to the facial and gait recognition components. This adaptive pipeline guaranteed that the system could correctly identify players even in frames where traditional single-mode recognition methods were unreliable. Performance benchmarks for the OCR component are detailed in Chapter 4, confirming the robustness of the integrated approach. Appendix B includes the EAST-based detection, preprocessing logic, and the OCR pipeline code.

3.5.4 Ensemble Approach

After extracting features with ResNet50, multiple machine learning classifiers were tested to determine the best technique. The classifiers examined were SVM, KNN, Random Forest, Gradient Boost Machine, and Decision Tree. SVM and KNN demonstrated the best performance, however overfitting continued even after cross-validation and hyperparameter adjustment. To address these concerns further, ensemble approaches such as the Voting Classifier and Decision Level Fusion were investigated. Finally, the Stacking Ensemble method proved the most successful. Overfitting was significantly decreased by merging SVM, KNN, and a final Logistic Regression layer inside a stacking ensemble and tweaking their parameters, resulting in enhanced model performance and generalization as demonstrated in figure 8. This technique improved the overall accuracy and dependability of player labelling.

Workflow of the Stacking Ensemble

Base Classifiers:

- **SVM:** Utilizes an RBF (Radial Basis Function) kernel to distinguish non-linear patterns within the feature space derived from ResNet50. SVM reduces noise and improves the model's resilience to data fluctuations.

- **KNN:** Captures local spatial relationships in the feature space using the Manhattan distance metric. The ability for effective generalization enhances the accuracy of SVM, establishing a balanced and efficient basis for classification.

Meta-Classifier:

- **Logistic Regression:** Functions as the ultimate decision-making tier, integrating predictions from SVM and KNN. The meta-classifier combines these predictions into a unified output, minimizing individual classifier inaccuracies and enhancing overall accuracy. By learning patterns in the outputs of the base classifiers, Logistic Regression ensures that the ensemble effectively addresses variability in cricket player data.

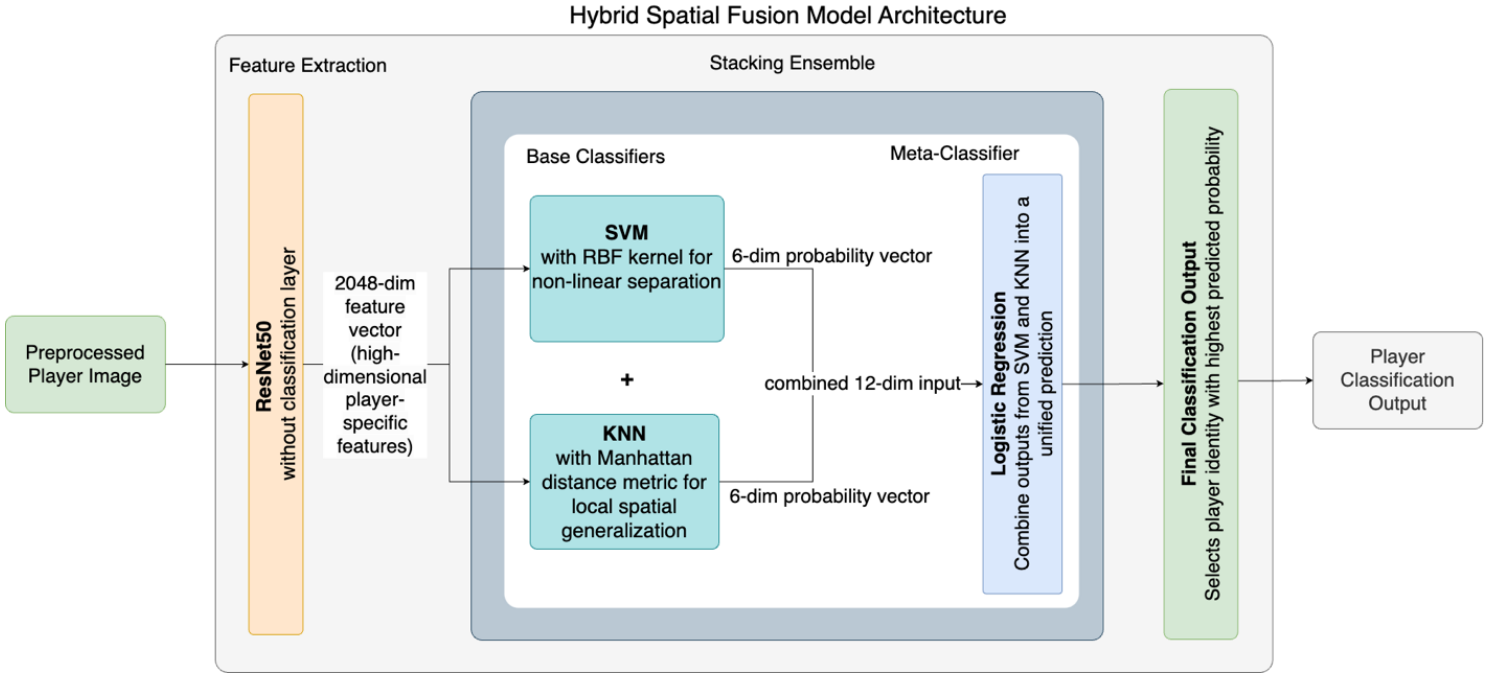


Figure 8: Detailed Hybrid Spatial Fusion Model Architecture for the Proposed T20i Cricket Player Recognition System.

Probability outputs from these base classifiers serve as inputs to a Logistic Regression meta-classifier, combining predictions into a unified and robust final decision, reducing classification errors and significantly enhancing overall accuracy. Each base classifier (SVM and KNN) produces a probability distribution across the six player classes, resulting in two separate 6-dimensional output vectors. These are concatenated to form a single 12-dimensional feature vector, which serves as the input to the Logistic Regression meta-classifier. This meta-layer then outputs the final predicted player class by learning from the combined decision patterns of the base classifiers.

Improvement Over Existing CNN + SVM/KNN Approaches:

Unlike traditional pipelines where a CNN is followed by a single classifier (such as SVM or KNN), proposed approach employs a stacking ensemble that integrates multiple classifiers. This layered structure allows each base classifier to specialize in capturing different aspects of the feature space, SVM excels at handling non-linear decision boundaries, while KNN captures local relationships. The Logistic Regression meta-classifier learns from the strengths and weaknesses of both, leading to improved robustness and reduced overfitting. This multi-stage learning strategy significantly outperforms single-classifier CNN-based methods, as demonstrated in the Chapter 4 through empirical results and statistical validation.

3.5.5 Player Statistic Generation using LLM

To create player-specific statistics based on user queries, a large language model (LLM) was employed to convert natural language questions into SQL queries that retrieved relevant data from the cricket database. The LLM-based strategy entailed moving from open-source models such Llama 13B, and Gemini to the OpenAI GPT-4o model for best performance.

SQL queries were generated using prompt engineering and structured parsing approaches. LangChain and Pydantic were used to handle the LLM prompts, assuring proper SQL creation, and the MySQL database was used to retrieve the data. Several challengers were encountered, including model hallucinations and improper SQL syntax. These concerns were addressed by assigning the model a persona and employing chain-of-thought prompts to facilitate logical inquiry construction.

The workflow included:

1. **Prompt Engineering**: A structured prompt was developed to provide the LLM with extensive explanations of the database schema and interactions between tables, including batting, bowling, and fielding. The prompt provided detailed advice for creating MySQL-compatible queries and ensuring proper syntax.

2. **SQL Query Generation**: The LLM used natural language input to produce SQL queries. Pydantic was used to parse the query output, and OutputFixingParser was used to fix any discrepancies.

3. **Database Interaction**: The generated SQL queries were executed against a MySQL database to get player statistics, including runs, wickets, and match results.

4. **Model Performance Evaluation**: The OpenAI GPT-4o model outperformed other baseline models in terms of SQL query accuracy and response time. The baseline performance was compared to open-source models such as Llama 13B, demonstrating the benefits of employing advanced proprietary models for complicated, domain-specific problems.

Overall, this LLM-based method efficiently converted user questions into SQL queries, resulting in useful player statistics. Model hallucinations were minimized by prompt engineering and chain-of-thought reasoning, resulting in accurate query formulation and efficient database interaction. The full prompt engineering, LangChain integration, and SQL query execution logic are detailed in Appendix B.

3.5.6 Hyperparameter Tuning

Hyperparameter tuning is a crucial phase in maximizing model performance, particularly when applying machine learning and deep learning approaches to real-world data. To determine optimal configurations for each model component, this study used a combination of grid search, manual experimentation, and k-fold cross-validation. Tuning aimed to improve classification accuracy, convergence stability, and generalization across unseen match scenarios. For classical ML classifiers, hyperparameter search was performed using tools such as GridSearchCV from the Scikit-learn library, whereas TensorBoard and validation loss plots were used to tune deep learning models like GRU. LLM quick optimization was done manually

using iterative testing and evaluation. The final tuned parameters were chosen based on the best validation performance while limiting overfitting, as seen by evaluation metrics in Chapter 4.

Facial Recognition – MTCNN + FaceNet + SVM Classifier

The face recognition pipeline utilized Google's FaceNet to generate embeddings from facial images, which were subsequently classified using a SVM. The hyperparameters listed below were tuned using 5-fold cross-validation:

- Kernel: 'rbf' – selected for its ability to capture non-linear patterns
- C (Regularization Parameter): 1 – balances margin and misclassification penalty
- Gamma: 'scale' – adapts to feature variance automatically

This design provided the optimal balance of bias and variance, allowing the SVM to identify between players with high accuracy even in demanding scenarios such as partial occlusion or varying camera angles.

Spatial Model – YOLO + ResNet + Stacking Ensemble

For spatial classification, ResNet50 was used as a frozen feature extractor. The extracted features were then classified using a stacking ensemble composed of SVM, KNN, and Logistic Regression. The final hyperparameters were as follows:

- SVM:
  - kernel = 'rbf'
  - C = 1
  - gamma = 'scale'
- KNN:
  - n_neighbors = 3
  - metric = 'manhattan'
  - weights = 'uniform'
- Logistic Regression (meta-classifier):
  - C = 0.001
  - penalty = 'l2'
  - solver = 'liblinear'

These values were chosen using grid search and 5-fold cross-validation to provide excellent classification boundaries and generalization across diverse match situations. Tuning considerably decreased overfitting, as seen by the learning curves and confusion matrices described in Chapter 4.

Temporal Model – YOLO + MediaPipe + GRU with Engineered Features

The temporal model used a GRU network to analyze engineered spatio-temporal gait features. The following hyperparameters were tuned:

- GRU Units: 64 – sufficient capacity without overfitting
- Dropout Rate: 0.3 – mitigated overfitting by preventing co-adaptation of neurons
- Learning Rate: 0.0005 – enabled smooth convergence

- Batch Size: 32 – balanced memory and convergence stability
- Epochs: 50 with early stopping (patience = 10)

These hyperparameters were selected through multiple training runs using different sequence splits. Training progress was monitored using TensorBoard to detect overfitting trends and ensure smooth convergence.

LLM Prompt Engineering – GPT-4o

While GPT-4o is a closed-source pre-trained model, its behavior was optimized through prompt engineering. The following strategies were applied:

- Use of schema-aware prompts that explicitly described table structures and column types
- Inclusion of chain-of-thought reasoning to guide logical SQL construction
- Structured JSON-based formatting to help enforce consistent output
- Integration of LangChain's OutputFixingParser and Pydantic validation for error handling
- Retry-on-failure logic for malformed or incomplete queries

This iterative prompt tuning significantly boosted SQL query success rates, reduced hallucinations, and achieved query accuracy of 85-90% on test inputs. These modified hyperparameters were used in the final tests, and their effectiveness is demonstrated by improved model accuracy and generalization performance, as reported in Chapter 4.

3.6 Experimental Setup

The experimental setting for this work was designed to assess the performance of the suggested hybrid model for cricket player detection, which combines face recognition, spatio-temporal analysis, and player-specific statistics creation. This section describes the experimental settings, datasets, and processes used to assure reliable model training and assessment.

3.6.1 System Configuration

The tests were carried out using an Apple MacBook Pro with an M1 processor and 8GB of RAM, which was adequate to handle the computational demands of training and testing both deep learning and machine learning models. The software environment included:

- Python 3.9 as the primary programming language.
- Deep Learning Frameworks: TensorFlow for model development.
- Libraries: OpenCV and CLAHE for video and image preprocessing, Matplotlib for visualization, Scikit-Learn and Joblib for machine learning model development and persistence.
- NumPy for numerical computations, and Pydantic for data validation.
- Pytesseract for OCR text recognition, MTCNN for face detection and Google's Medipipe for player joint detection.
- LangChain for integrating large language models (LLMs) to generate SQL queries.

3.6.2 Model Integration and Testing

Following individual training, the face recognition, spatio-temporal, and LLM components were combined into a single pipeline for player recognition and statistics production. The testing phase involved:

- **Sequential Recognition**: New video clips are fed into the system to test the sequential identification process, which includes OCR text detection, facial recognition, spatial and temporal analysis.

- **Player-Specific Statistics**: Generating player-specific statistics using LLM-processed natural language queries, checked against manually collected information to ensure correctness.

3.6.3 Experimental Procedure

The experimental procedure involved the following stages:

1. **Data Collection and Preprocessing**: Collected, cleaned, and transformed data to suit face recognition, gait analysis, and LLM standards.

2. **Independent Component Training**: Each model component was trained independently, with hyperparameter adjustment to optimize performance.

3. **Model Integration**: The models were incorporated into a system that analyzed a video sequentially to recognise players and provide statistics.

4. **Performance Evaluation**: Real match video footage and user enqueries were used to assess the system's performance. The system's efficacy was evaluated using specific assessment measures for each component (eg. accuracy, F1-score, response time).

The experimental setting ensured that the system was evaluated under dynamic match conditions, taking into consideration varied illumination, player locations, and actions, verifying the hybrid approach's robustness and generalisability.

3.6.7 Validation Strategy

The validation approach used in this work was intended to validate the hybrid model's robustness, reliability, and generalizability for cricket player recognition and player-specific statistic generation. Validation took place at both the component and system levels, allowing for a thorough review of each model and the combined pipeline.

Cross-Validation

- **K-Fold Cross-Validation**: The facial recognition and spatio-temporal models were evaluated using a 5-fold cross-validation technique (K-fold). This strategy divided the dataset into five subgroups, iteratively training the model on four folds and verifying on the remaining fold. This ensured a diversified training and validation dataset, decreasing bias and overfitting.

- **Temporal Model Validation**: Cross-validation of gait sequences ensured the GRU model's generalisability. Temporal characteristics were verified against various segments of player motion data to assess the model's performance on previously encountered sequences.

Training and Validation Split

- **Training and Validation Split**: The datasets for facial recognition, spatio-temporal analysis, and player statistics creation were separated into training, validation, and testing sets. Specifically, 80% of the data was utilized for training, 10% for validation, and 10% for testing. The validation set was utilized to modify hyperparameters and reduce overfitting, whilst the test set gave an unbiased assessment of the final model.

Early Stopping and Ensemble Validation

- **Early Stopping**: Deep learning models were trained using early stopping to prevent overfitting. Training was paused when the model's validation loss stopped improving after a certain number of epochs, ensuring that the model did not learn noise from the training dataset.

- **Ensemble Model Validation**: Validated the stacking ensemble for facial recognition and spatial analysis using separate base model evaluations. This method assisted in determining the ensemble's efficacy in terms of bias and variance reduction.

End-to-End System Validation

- **Integrated Pipeline Testing**: New footage was used to evaluate the hybrid recognition pipeline's overall performance. Validation consisted of progressive testing, beginning with OCR text detection and progressing to facial recognition, spatial and temporal gait analysis. The output from each stage was tested to ensure that it accurately contributed to the final player recognition and statistics creation.

- **Manual Verification**: Player recognition and statistics generation results were manually verified against known records to ensure the accuracy of the integrated system. This step was particularly important for validating the LLM's SQL query responses against actual player data.

The validation process ensured that each component and the entire system were thoroughly tested to minimize errors, enhance accuracy, and verify that the models could generalize to real-world cricket match circumstances. This extensive validation technique raised confidence in the proposed hybrid system's dependability and resilience.

3.7 User Interface Prototype

To ensure accessibility and near real-time usage, a browser-based front-end interface was developed for the CricXpert system. This user interface acts as a visual and functional bridge between the ensemble AI models and the end users coaches, analysts, or cricket fans. The UI was created to accommodate two distinct workflows: player recognition from images/videos and natural language-based statistics generation.

3.7.1 Design Objectives

The UI design was guided by the following goals:

- Intuitiveness: Minimal learning curve with clearly separated modules and actions.
- Responsiveness: Near real-time predictions and data retrieval.
- Modularity: Independent operation of the recognition and statistics components.
- Aesthetic Clarity: Clean layout and vibrant visuals aligned with the cricket theme.

3.7.2 Functional Workflow and Screens

The user interface supports the following sequence of interactions:

Initial Interface View: Upon accessing the system, users are presented with two distinct modules, Player Recognition (top) and Stat Generation (bottom). The user can upload an image or video file and also input natural language queries independently (Figure 9).

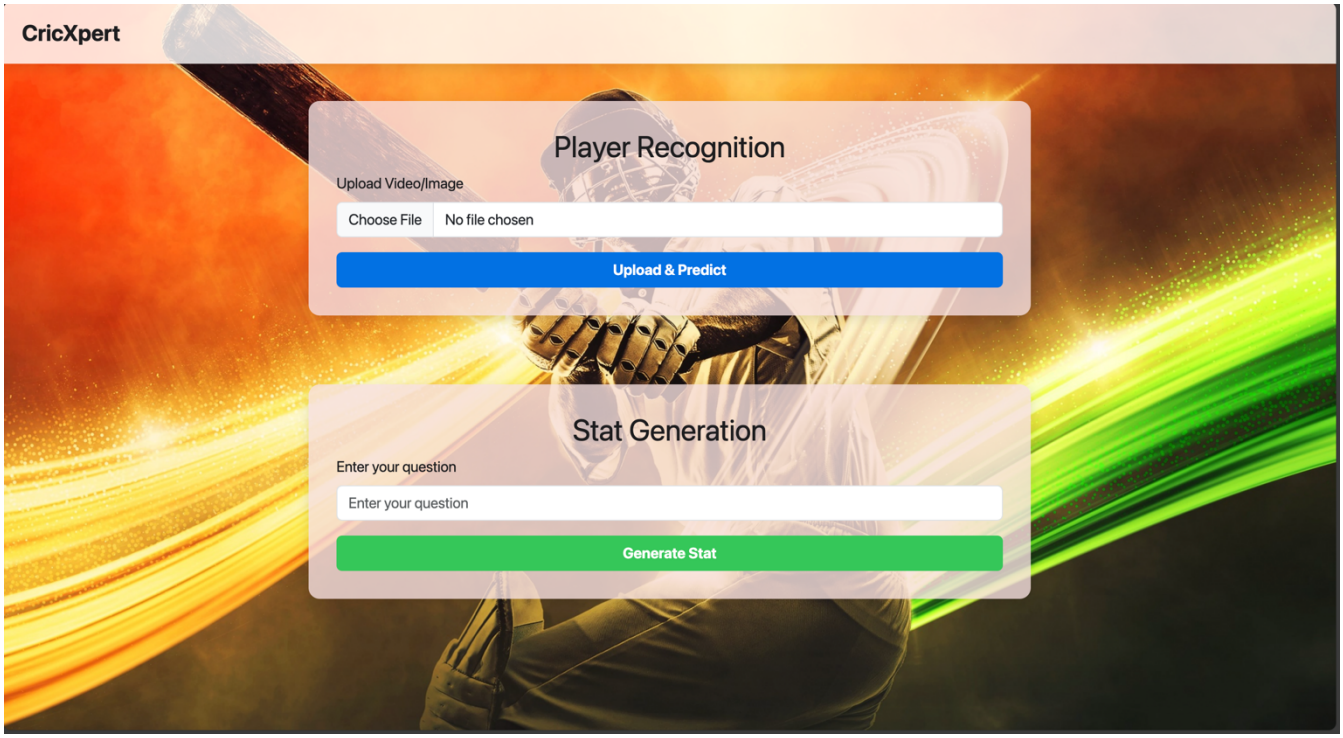


Figure 9: Initial Interface View

File Upload and Prediction In Progress: When a user uploads a video or image and clicks "Upload & Predict", the system begins processing the media for player recognition. A loader animation indicates the progress of the prediction pipeline (Figure 10).

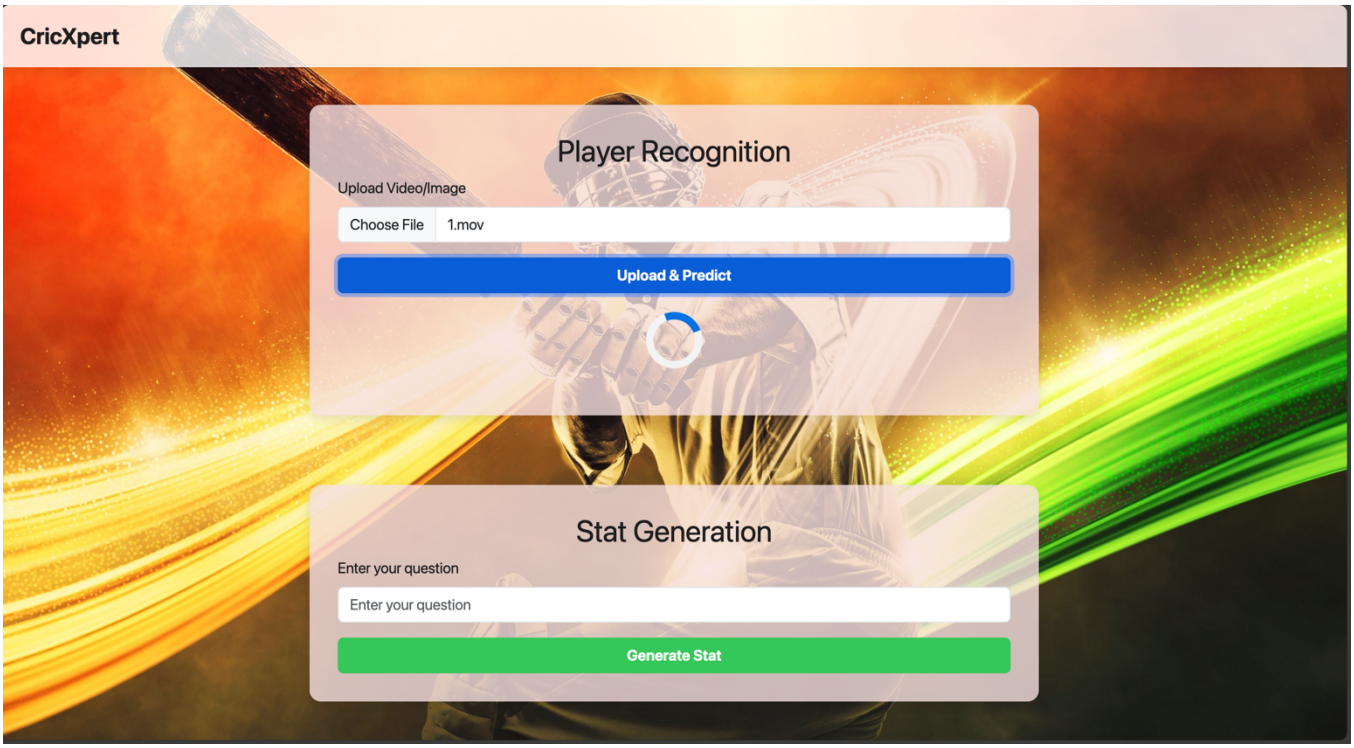


Figure 10: File Upload and Prediction In Progress

Prediction Display: After successful recognition, the predicted player's name (e.g., *Ravindra_Jadeja*) is shown, and users can optionally click **"Display Basic Stat"** to view player-specific metrics retrieved from the database (Figure 11).

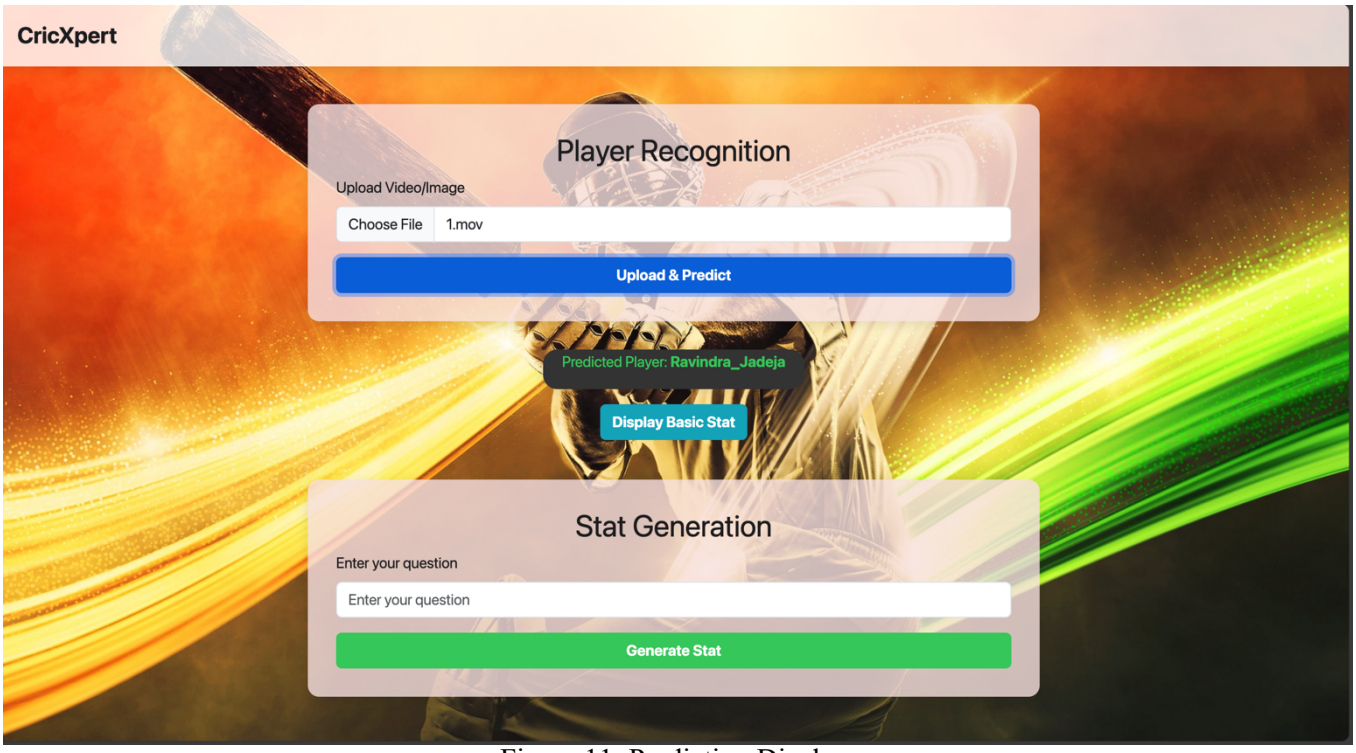


Figure 11: Prediction Display

Basic Stats Table View: The player's profile is presented in a structured table format, including career span, matches played, runs, batting and bowling averages, strike rate, economy, and other key performance indicators (Figure 12).

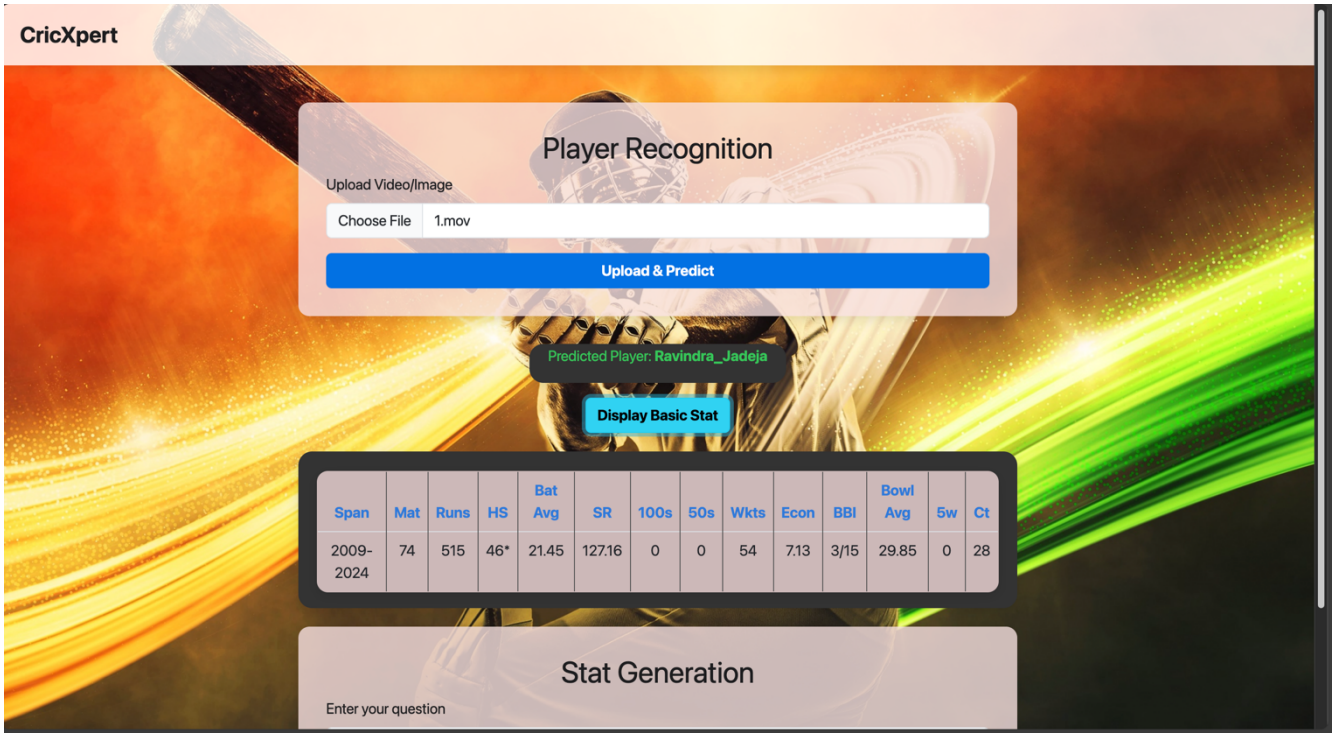


Figure 12: Basic Stats Table View

LLM Query Input: Users can enter free-form questions such as *"How many runs did Virat Kohli score at batting position 3?"* The interface is designed to work independently of the previous recognition step (Figure 13).

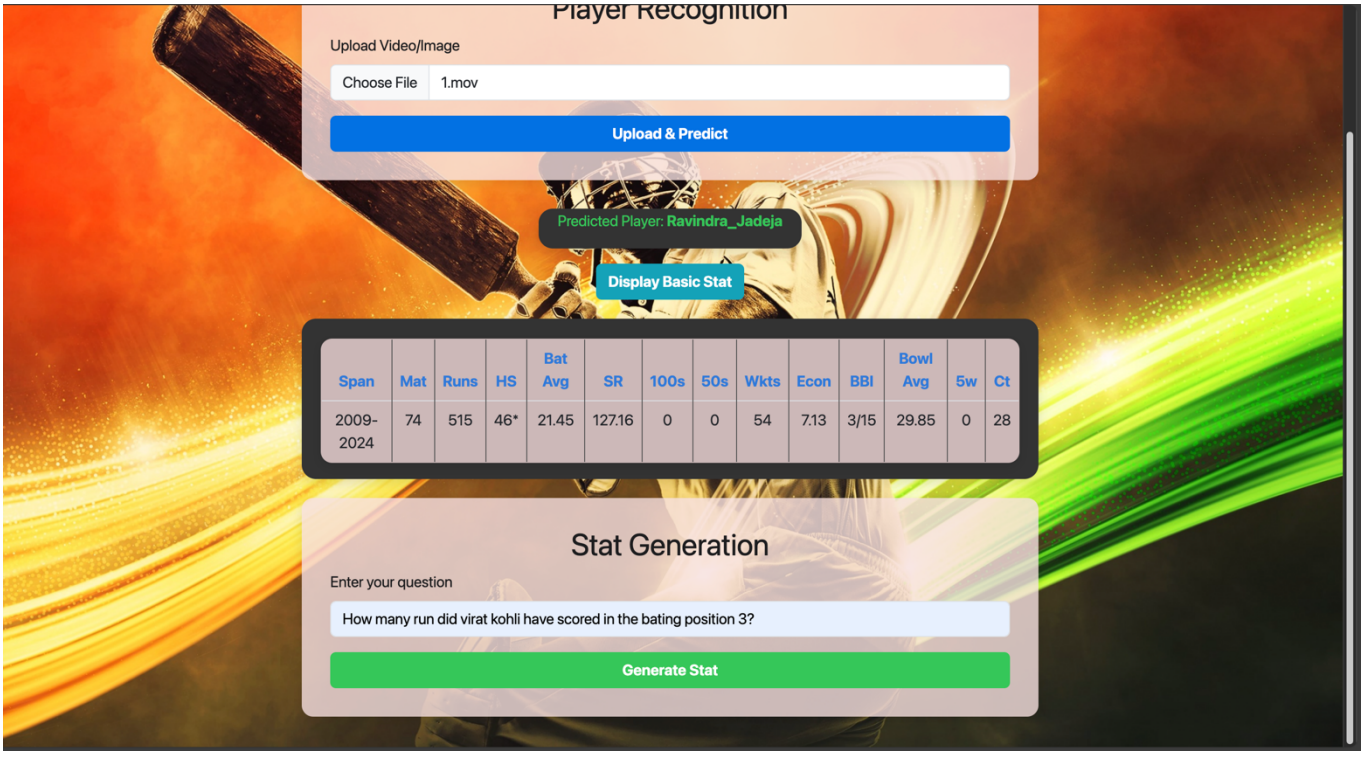


Figure 13: LLM Query Input

LLM-Based Stat Output: Upon submission, the query is processed by GPT-4o, converted into a SQL statement via LangChain, and executed against the MySQL cricket database.The resulting stats (e.g.*"3076"*runs) are returned and displayed beneath the input area (Figure 14).

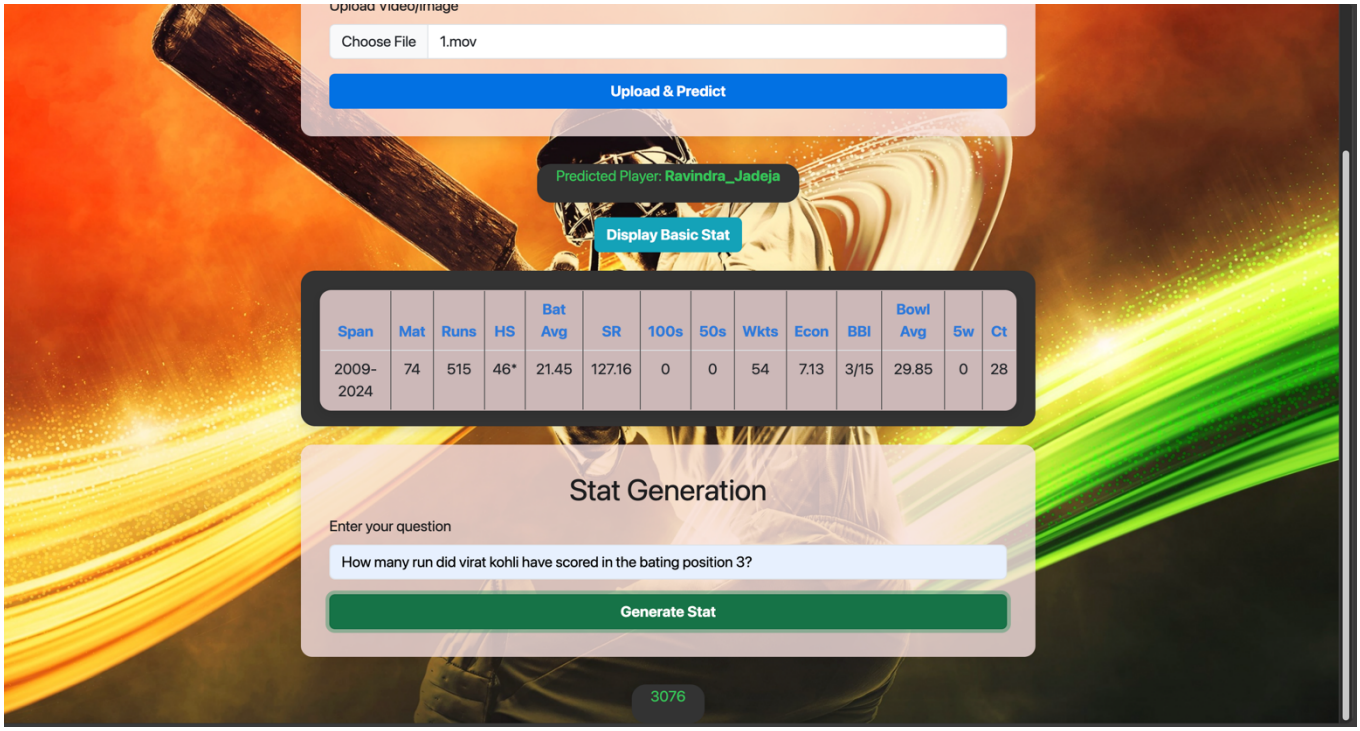


Figure 14: LLM-Based Stat Output

3.7.3 Technology Stack

This section highlights the deployment-specific architecture powering the user interface prototype.

- Frontend: HTML, CSS, JavaScript
- Backend: Python Flask
- Model Integration: REST endpoints for facial/spatial/temporal recognition and OCR
- LLM Layer: OpenAI GPT-4o integrated via LangChain
- Database: MySQL for stat storage and retrieval

This prototype showcases how advanced AI modules can be encapsulated within a simple user-facing interface, making complex AI predictions and data access seamless for non-technical users. The full system backend implementation, including REST API endpoints and integration code, can be found in Appendix B.

04 RESULTS

4.1 Chapter Overview

This chapter presents the main results of the research, detailing how each component of the hybrid player recognition system - facial, spatial, temporal, and LLM-based modules performed based on well-defined evaluation metrics. It outlines the baseline models attempted, the iterative improvements made, and how the final integrated system was benchmarked. The evaluation also includes expert validation of system usability, performance under match conditions, and dataset quality. Comparative analysis with various architectures like Vision Transformers, ResNet, EfficientNet, and ensemble models establishes both the efficacy of the final approach and its superiority over baseline alternatives.

4.2 Evaluation Metrics

Several evaluation metrics were used to thoroughly analyze the hybrid model's performance for cricket player recognition and player-specific statistics creation. These metrics were chosen to match the specific needs of each system component, such as facial recognition, spatio-temporal analysis, and SQL query production using a large language model (LLM). The evaluation aimed to establish the system's dependability, accuracy, and generalisability under dynamic match situations.

4.2.1 Facial Recognition Metrics

The performance of the face recognition model was assessed using the following metrics:

1. **Accuracy**: The accuracy of a facial recognition model is determined by the proportion of correctly detected examples relative to total instances

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

2. **Precision**: The algorithm's success rate in detecting player faces compared to the total number of faces recognised, minimizing false positives.

$$\text{Precision} = \frac{TP}{TP + FP}$$

3. **Sensitivity (Recall):** The ratio of properly detected player faces in a dataset, indicating the model's capacity to detect all relevant occurrences.

$$\text{Recall} = \frac{TP}{TP + FN}$$

Where:

True Positives (TP): Correctly identified instances
True Negatives (TN): Correctly rejected instances
False Positives (FP): Incorrectly identified instances
False Negatives (FN): Missed instances

4.2.2 Spatio-Temporal Model Metrics

The spatiotemporal component included spatial analysis with deep learning and temporal analysis with gait characteristics. The assessment measures for these components were:

- **Spatial Model**:

    1. **Accuracy**: Evaluated player classification accuracy using ResNet50 spatial features.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

    2. **Precision and Recall**: Machine learning classifiers (SVM, KNN, Logistic Regression) were evaluated for their ability to discriminate players based on spatial information using precision and recall measures.

$$\text{Precision} = \frac{TP}{TP + FP} \qquad \text{Recall} = \frac{TP}{TP + FN}$$

    3. **Confusion Matrix**: Used to measures the distribution of true positives, false positives, true negatives, and false negatives for player recognition.

- **Temporal Model**:

    1. **Sequence Accuracy**: Evaluated the GRU model's accuracy in classifying temporal sequences of posture landmarks and recognising player movement patterns across time.

$$\text{Sequence Accuracy} = \frac{\text{Number of Correctly Predicted Sequences}}{\text{Total Number of Sequences}}$$

    2. **Model Accuracy and Model Loss Plot**: The Model Accuracy and Loss Plot visualizes training and validation accuracy and loss, identifying overfitting or underfitting during training.

        Model Loss: Loss is calculated during training using a specific loss function, Cross-Entropy Loss:

$$\text{Loss (Cross-Entropy)} = -\sum_{i=1}^{N} y_i \log(p_i)$$

        Where:
        - y_i : Actual label (1 for correct class, 0 for others)
        - p_i : Predicted probability of class i

4.2.3 LLM-Based Player Statistics Generation Metrics

The LLM used for generating player-specific statistics was evaluated using the following metrics:

1. **SQL Query Accuracy**: The percentage of created SQL queries that yielded correct results when performed against the database. This indicates that the LLM accurately translated natural language enqueries into SQL instructions.

$$\text{SQL Query Accuracy} = \frac{\text{Number of Correct SQL Queries}}{\text{Total Number of Generated SQL Queries}}$$

2. **Response Time**: The time it takes for the LLM to produce a SQL query and provide results. This is used to assess efficiency in real-time applications.

$$\text{Average Response Time} = \frac{\sum_{i=1}^{N} t_i}{N}$$

Where:

- t_i : Response time for each query
- N : Total number of queries

3. **Correctness of Retrieved Information**: Verified the accuracy of obtained player statistics by comparing LLM-generated responses with manually validated data.

$$\text{Correctness} = \frac{\text{Number of Correct Results}}{\text{Total Results Retrieved}}$$

4.2.4 Integrated System Evaluation

The overall performance of the hybrid player recognition system was tested using the following metrics:

1. **Overall System Accuracy**: the percentage of correct player identifications and statistics generated across trials.

$$\text{Overall System Accuracy} = \frac{\text{Total Correct Identifications}}{\text{Total Identifications}}$$

2. **Execution Time**: Measured the time taken for the system to process a new video clip, perform recognition, and generate statistics, ensuring that the system's

performance was suitable for near-real-time applications in dynamic environments like cricket matches.

$$\text{Average Execution Time} = \frac{\sum_{i=1}^{N} t_i}{N}$$

Where:

- t_i : Time taken for each trial
- N : Total number of trials

These evaluation criteria gave us a thorough insight of each hybrid system component's strengths and drawbacks. The suggested solution's overall efficacy, dependability, and scalability were carefully evaluated by analyzing each measure separately and as part of an integrated system.

4.2.4 Expert Evaluation of System Performance and Dataset Quality

Evaluation Procedure

The CricXpert system's real-world applicability, usability, and performance were validated by an expert evaluation with three domain professionals. These experts, identified as Subjects 1, 2, and 3, are significant stakeholders in cricket coaching, sports data analytics, and professional performance evaluation.

Selection of evaluators

There are two categories of evaluators selected for the project. Two domain experts with substantial cricket expertise, including coaching, team planning, and performance analytics. These individuals have firsthand experience of match conditions, player behaviors, and decision-making requirements in professional cricket settings. Technical expert with expertise in artificial intelligence and machine learning applications and data science in a major Telcom.

Table 2: Selected evaluators

Expert	Specialization	Key Comments
Subject 1	Former National Fast-Bowling Coach	“System is highly intuitive. Could integrate physiological metrics for deeper analysis.”
Subject 2	Telecom Data Science Lead	“The hybrid approach is robust. Stats are relevant. Would benefit from feedback loops.”
Subject 3	First XI Cricket Coach	“Facial + gait recognition makes instant identification possible. Very helpful for coaching.”

The evaluation process followed a mixed-methods approach:

A structured Google Form questionnaire was developed, incorporating quantitative evaluations (1-5 Likert scale) and qualitative open-ended feedback. Each expert took part in a live one-on-one interview where the system's functionality and the dataset were shown (including player recognition and statistical generation). Experts offered feedback based on their practical experience with cricket, data interpretation requirements, and coaching value.

Ethical Considerations

Formal ethical approval from a review board was not necessary because no personal or sensitive information about the experts was collected or disclosed. The review did not include any human subject experiments. All experts gave informed consent to utilize their anonymized feedback in this scholarly study.

Inclusion and Exclusion Criteria

Table 3: Inclusion and Exclusion Criteria

Inclusion Criteria	Exclusion Criteria
Professionals with domain expertise in cricket, coaching, or analytics	Individuals without direct experience in cricket performance analysis
Willingness to participate in both interview and questionnaire	Lack of availability for live demonstration or feedback
Minimum of 10 years of experience in the field	Undergraduates or casual observers

Quantitative Evaluation Results

Table 4: Quantitative Evaluation Results

Evaluation Criteria	Subject 1	Subject 2	Subject 3	⭐ Average Score (out of 5)
Innovation & Novelty	5	5	5	⭐⭐⭐⭐⭐ (5.00)
Performance under Real Match Conditions	5	4	5	⭐⭐⭐⭐☆ (4.67)
Quality of Generated Statistics	5	5	5	⭐⭐⭐⭐⭐ (5.00)
System Usability & User Experience	5	5	4	⭐⭐⭐⭐☆ (4.67)
Dataset Quality & Annotation Accuracy	5	5	5	⭐⭐⭐⭐⭐ (5.00)

(Ratings were provided on a 1–5 Likert scale)

Qualitative Insights

Question 1 - What are your initial thoughts on integrating face recognition and spatio-temporal gait analysis for player recognition in cricket?	
Expert	Feedback
Subject 1	I really like the idea. Combining facial recognition with gait analysis means we can identify players almost instantly—this is a huge time saver, especially during fast-paced moments in a match
Subject 2	This method offers a robust multimodal identification system which can be used for player tracking for performance analytics which can revolutionize the field of sports analytics
Subject 3	I find this approach very promising. Combining facial recognition with gait analysis means we can identify players almost immediately during a match. This rapid identification is a major advantage, especially when split-second decisions are required

Question 2 - How innovative do you find this approach compared to existing player recognition systems in sports?	
Expert	Feedback
Subject 1	This approach is very innovative. Most systems only rely on a single identifier, like a player's name or jersey number. By using both face and gait data, your system offers a much more robust solution for quick and accurate identification
Subject 2	The integration of spatio-temporal gait analysis introduces a new biometric layer that is unique, and works even when facial data is partially unavailable. This dual-modality system enhances robustness, making it a cutting-edge solution
Subject 3	I'd rate this approach very highly on innovation. Most systems I've seen only rely on static identifiers like names or jersey numbers. By merging dynamic data from facial features and gait patterns, the system offers a more comprehensive and reliable identification method.

Question 3 - Can you describe any instances where the system performed exceptionally well or failed under specific conditions (like varying light or occlusions)?	
Expert	Feedback
Subject 1	In practice, I've seen the system work really well. For example, it can immediately pinpoint the player responsible for a misfield rather than waiting for the reply to know who the player us,

	which is incredibly useful for coaching. I didn't notice any significant failures even when lighting conditions changed or when players were partially obstructed
Subject 2	Worked well even when faces are partially visible, also in different angles. Struggles a little bit with harsh shadows, or with high speed movements
Subject 3	During the live demonstration, I saw the system quickly pinpoint the player responsible for a misfield, even when the lighting wasn't perfect or parts of the player were obscured. It seems to handle challenging conditions quite well, which is essential in the fast-changing environment of a cricket match

Question 4 - How effectively does the stat generation system meet your expectations for accuracy and comprehensiveness in producing player statistics?	
Expert	Feedback
Subject 1	I'm impressed with the stat generation component. It automatically pulls together detailed stats, saving us the hassle of manually searching for data. It's exactly what we need to get the complete picture quickly. I didn't encounter any issues, it's comprehensive and spot-on in the details it provides
Subject 2	Stats were obtained using limited data collected from Cricinfo database which is perfectly alright to prove the concept. However, this can easily be extended by implementing a search tool to provide real time statistics
Subject 3	The stat generation system exceeded my expectations. It compiles detailed statistics automatically—saving us from the tedious task of manually searching and filtering data. This efficiency is crucial for making informed decisions during a match

Question 5 - How relevant are the statistics generated by the system to making strategic decisions in T20 cricket matches? Are there additional metrics or data that you find missing?	
Expert	Feedback
Subject 1	The stats are very relevant. They give a clear picture of how a player performs under different conditions, which is vital for tactical decisions. Honestly, I couldn't say that any key metric was missing, they cover all the bases needed for smart decision-making And these types of system are available with the national teams around the world but it is not

	publicly available and is not available for domestic, school level. But this system can fill that gap
Subject 2	Highly relevant statistics were provided. Can be extended further to include real time stats
Subject 3	The statistics are extremely relevant. They deliver insights such as performance trends on different pitches and against various opponents, which are key for tactical decisions. From my perspective, all critical metrics seem to be covered, and this system could fill the gap that currently exists at domestic and school levels

Question 6 - What are your thoughts on the system's architectural design and data processing workflow?	
Expert	Feedback
Subject 1	The design and workflow are well thought out. The process from data collection and preprocessing to final output is smooth and efficient, which is crucial when time is of the essence during matches and very useful to the squad selections
Subject 2	This architecture shows thoughtful design choices that leverage the strengths of different algorithms. The hybrid approach combining CNN-based features with ensemble methods should provide robust player recognition across various conditions. The inclusion of both spatial and temporal models suggests the system can handle both still images and video sequences effectively. The stacking ensemble approach is well-designed, as it uses complementary base classifiers (SVM for non-linear pattern separation and KNN for local relationships) before unifying them through logistic regression.
Subject 3	The system's architecture is solid and well-conceived. The data processing workflow—from collecting and preprocessing data to generating final outputs—is efficient and smooth. This kind of streamlined process is vital when decisions need to be made in real time.

Question 7 - How would you rate the user-friendliness of the whole system? Were there any parts that were confusing or difficult to navigate?	
Expert	Feedback
Subject 1	The system is very user-friendly. Even someone who isn't tech-savvy can navigate it easily; you

	simply type in your query, and the system does the rest. I didn't encounter any confusing parts at all Its simplicity is a big plus, especially for users who need quick answers without a steep learning curve
Subject 2	The system is highly user friendly. One improvment that can be made is, currently there's no apparent feedback mechanism for users to correct misidentifications
Subject 3	The system is very user-friendly. Even someone without technical expertise can use it without any issues; you just type in your query and the system delivers the information you need

Question 8 - Upon reviewing the dataset, how would you rate the quality and cleanliness of the data?	
Expert	Feedback
Subject 1	The dataset is top-notch, very clean and well-organized. This is essential for training the models effectively, and it shows that a lot of care has gone into curating the data It's free of errors and structured in a way that supports the system very well
Subject 2	Data is clean with data from 6 players included in the analysis with clean and clear inputs provided to train the model while maintaining information related to different angles, light levels, protective equipment etc.
Subject 3	The dataset is clean, well-organized, and clearly curated. This level of detail is essential for training a reliable system and ensuring that the outputs are accurate

Question 9 - Does the dataset adequately represent the diversity of scenarios expected in real T20 cricket matches?	
Expert	Feedback
Subject 1	Yes, absolutely. The dataset captures a wide range of match scenarios and player actions, which makes it very relevant for a system designed for T20 cricket
Subject 2	Yes, as mentioned in Q8 it takes into account the diversity
Subject 3	Yes, the dataset does an excellent job of capturing the range of conditions and player actions you'd encounter in actual T20 matches. It's comprehensive and reflects real-world scenarios very well

Question 10 - How relevant do you find the dataset for training a player recognition system specifically designed for T20i cricket?	
Expert	Feedback
Subject 1	The dataset is highly relevant. It includes the necessary variety of conditions that you'd expect in a T20 match, ensuring that the system can handle real-world complexities effectively
Subject 2	Highly relevant, but can be improved with a variety of new scenarios included, though might not be releavant to prove the concept
Subject 3	The dataset is highly relevant. It's tailored to the unique demands of T20 cricket, ensuring that the system is well-equipped to handle the fast pace and dynamic nature of the game

Question 11 - How accurate and consistent are the annotations within the dataset?	
Expert	Feedback
Subject 1	The annotations are spot-on. Every player's data is correctly labeled and consistent, which is crucial for the system to learn accurately and perform reliably
Subject 2	Reasonably accurate
Subject 3	The annotations are very accurate and consistent. Every player's data is properly labeled, which is crucial for the system's learning process and overall reliability

Question 12 - Based on your evaluation, what are the major strengths of the system and dataset? What improvements would you suggest for both?	
Expert	Feedback
Subject 1	One of the biggest strengths is how quickly the system can identify players by combining facial and gait recognition, it really saves time. The stat generation is also a standout feature, making data instantly accessible without manual searching. The dataset itself is well-organized and representative of real match conditions. Currently this is very good for like player selections and stuff but if you can improve it even more and go deep into the subject matter this will really help the player to improve their game. As for improvements, I'd suggest exploring additional layers of analysis, maybe even looking into aspects like brain pattern studies, which could personalize the system further. Expanding the dataset with more

	advanced metrics could also provide deeper insights for player development.
Subject 2	For optimal relevance of a T20i cricket player recognition system include - Players in both international and league kits. - Various stadium lighting conditions - Different camera angles typical in cricket broadcasts - Players in action-specific poses (bowling action, batting stance, fielding positions) - Samples with and without helmets/protective equipment
Subject 3	The biggest strength is definitely the speed and accuracy in player identification. By combining facial and gait recognition, the system can provide instant insights, which is a huge advantage on the field. The stat generation component is another standout, offering comprehensive data at the click of a button. The dataset is robust and well-organized. For improvements, I'd suggest exploring additional analytical layers—perhaps incorporating sensor data or even aspects of physiological analysis—to make the system even more personalized and insightful. Expanding the dataset with more advanced performance metrics could also enhance its value.

Question 13 - How accurate and consistent are the annotations within the dataset?	
Expert	Feedback
Subject 1	Definitely. I think integrating features that analyze brain patterns could be a fascinating next step, making the system more personalized. This could offer tailored insights into individual performance and further enhance the system's utility
Subject 2	Real time stats, self learning algorithm
Subject 3	Absolutely. I think integrating predictive analytics that could forecast player performance trends would be very useful. Moreover, exploring personalized data like biometric or sensor information could add a whole new dimension to the system's capabilities.

Question 14 - How accurate and consistent are the annotations within the dataset?	
Expert	Feedback
Subject 1	Expanding the dataset to include more detailed metrics, perhaps even physiological data like brain patterns would be beneficial. This would

	not only improve the system's accuracy but also open up new possibilities for personalized performance analysis
Subject 2	Same recommendations as for Q12
Subject 3	To further improve the dataset, I would recommend including data from a wider variety of match conditions and venues. Adding more detailed metrics—such as player fitness levels or even sensor-based measurements—could greatly enhance the system's training and validation process, leading to even better performance

4.3 Results and Benchmarking

To establish performance benchmarks, baseline models were initially implemented for each module. These were then incrementally upgraded using advanced architectures, ensemble methodologies, and domain-specific advances. The tables below show the models tested, their setups, and the performance results for each component.For each module, baseline models were first implemented to establish performance benchmarks. These were then systematically improved using advanced architectures, ensemble methods, and domain-specific enhancements. The following tables present the models tested, their respective configurations, and performance results for each component.

4.3.1 Facial Recognition: Model Performance Comparison

Facial recognition began with conventional methods that used YOLO for detection and ViT or FaceNet for classification. These models had poor accuracy under varying illumination and angles. The final configuration, which included MTCNN for detection and FaceNet embeddings classified by an ensemble, showed considerable improvements in accuracy and stability.

Table 5: Facial Recognition: Model Performance Comparison

Model	Detection	Classifier	Accuracy	F1-Score	Recall
MTCNN + ViT(Baseline)	MTCNN	SVC	80%	79%	80%
YOLO + FaceNet	YOLO	SVC	70%	72%	70%
YOLO + ViT	YOLO	SVC	66.67%	67%	67%
MTCNN + FaceNet (Final)	MTCNN	SVC	95.83%	96%	96%

Performance Visualization

To validate the ensemble's performance, the results were compared to the baseline MTCNN+ ViT model using learning curves and confusion matrices.

1. Learning Curve Comparison:

- **Baseline Model (MTCNN + ViT)**

  The learning curve (Figure 15) shows clear overfitting. The training score remains fixed at 100%, while the cross-validation accuracy stays around 70%. The wide gap between the curves and the large variance indicate that the model memorized training data but failed to generalize, especially under unseen lighting and conditions.

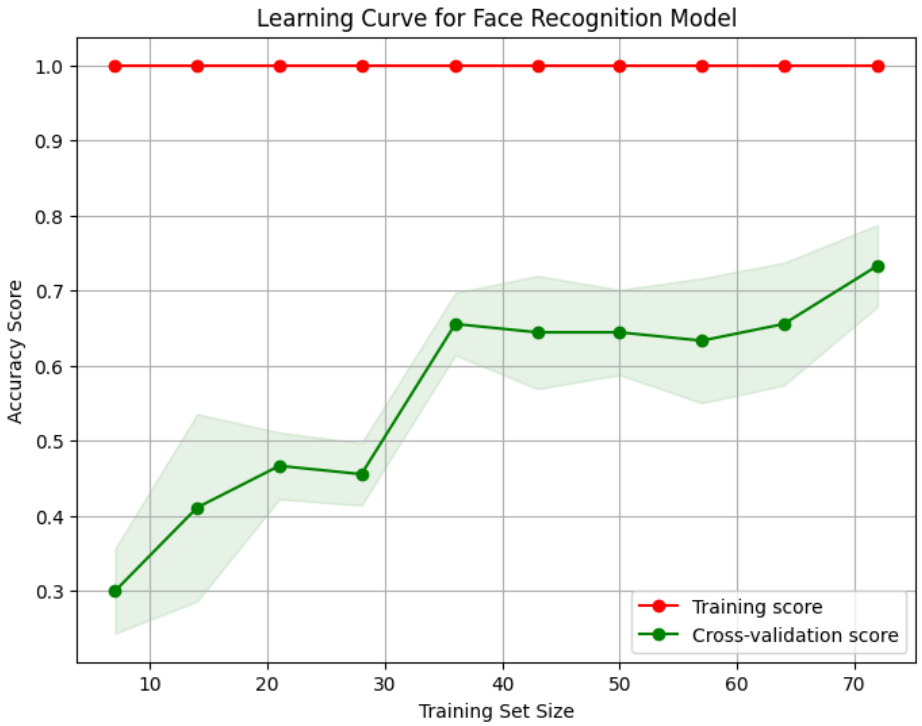


Figure 15: Facial Recognition: Baseline model learning curve

- **Final Model**

  The updated ensemble model (Figure 16) exhibits steady improvement in both training and cross-validation accuracy, with convergence above 95%. The reduced variance band suggests that the model generalizes well and avoids overfitting by leveraging complementary strengths of the base classifiers.

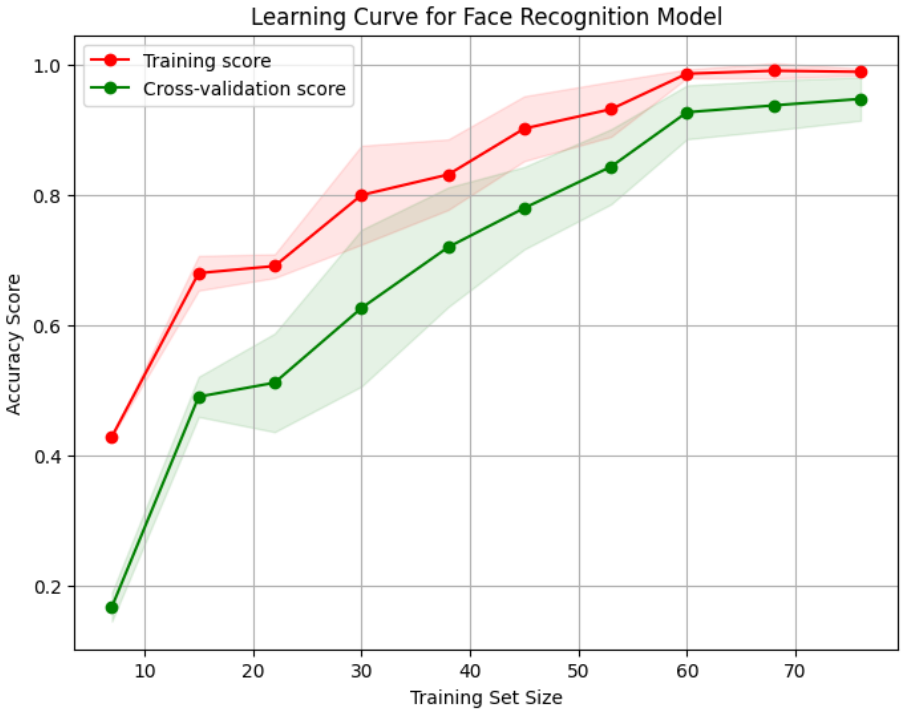


Figure 16: Facial Recognition: Final model learning curve

2. Confusion Matrix Comparison:

- **Baseline Model (MTCNN + ViT):**

  The confusion matrix (Figure 17) demonstrates frequent misclassifications, especially among players with similar visual features or under occlusion. Low True Positives (TPs) and numerous False Positives (FPs) show that the model struggled with real-world variability.

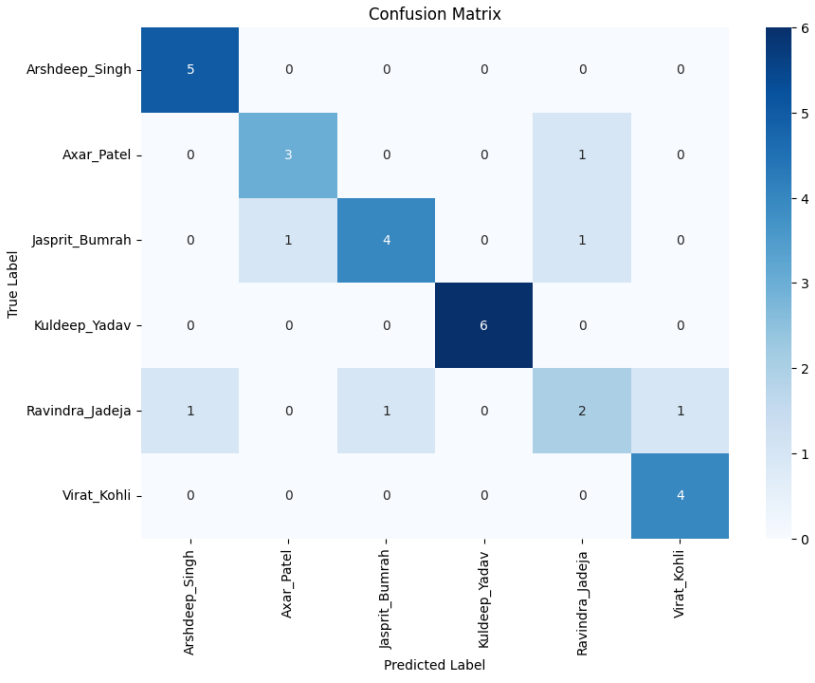


Figure 17: Facial Recognition: Baseline model confusion matrix

- **Final Model**

  The confusion matrix (Figure 18) shows substantial improvement. The model correctly identified almost all players, with minimal off-diagonal errors. This confirms the ensemble's ability to resolve ambiguities and enhance robustness through decision-level learning.

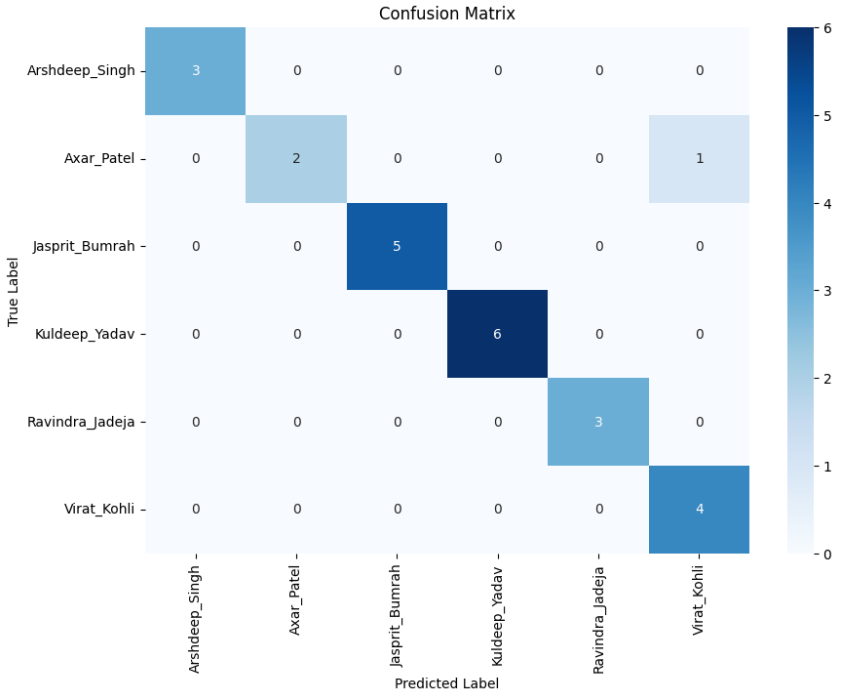


Figure 18: Facial Recognition: Final model confusion matrix

These findings demonstrate that the proposed model, which was improved with engineering embeddings and optimized hyperparameters, performed much better than the baseline models. The learning curves and confusion matrices demonstrate its efficacy in decreasing overfitting and enhancing generalization across dynamic match settings, making it a dependable and scalable method for near real-time cricket player detection.

4.3.2 Spatial Recognition: Model Performance Comparison

The initial stage of model selection involved examining various deep learning architectures, starting with a customized CNN model with six convolutional blocks, which were then followed by batch normalization, ReLU activation, and max-pooling. This was followed by five dense layers with dropout for regularization, before the final SoftMax classification layer. However, this approach resulted in poor performance, highlighting the need for more advanced architectures. Several models were then assessed and among these, ResNet50 emerged as the best performer due to its robust residual learning capability, which effectively addressed vanishing gradient issues and allowed for deeper feature extraction. The initial model comparisons yielded the following results (Table 6).

Table 6: Spatial Recognition: Baseline Model Performance Metrics

Model	Accuracy(%)	Precision(%)	Recall(%)
Custom CNN	40.32	41	40
DenseNet	69	70	68

EfficientNetB0	57.82	57	56
Inception	44	46	45
MobileNetV2	61.22	61	61
VGG16	60.78	61	60
NASNet	57	59	57
Xception	59	57	58
Vision Transformers (ViT) with different configurations	~42-60	~43-60	~42-60
ResNet50	61.75	62	61

Note: All baseline models were re-implemented and fine-tuned on the curated custom dataset under identical preprocessing and training conditions for a fair comparison. (ResNet50 was utilized as a frozen feature extractor in the final model.)

Deep Feature Fusion with ML Classifiers

While ResNet50 outperformed earlier models, overfitting persisted despite early stopping and parameter adjustments. This led to the hypothesis that these architectures' classification layers were responsible for the overfitting. To address this, a fusion technique was implemented: ResNet50 was used solely for feature extraction. Initial experiments with a few ML classifiers revealed that SVM and KNN performed well, enhancing classification accuracy while reducing overfitting. The following Table 7 summarize the improvement:

Table 7: Performance Metrics of Resnet50 Fused with Different ML Classifiers

Model	Accuracy(%)	Precision(%)	Recall(%)
ResNet50 + SVM	95.78	96	96
ResNet50 + KNN	95.43	94	95
ResNet50 + Random Forest	94	93	94
ResNet50 + Gradient Boost Machine	90.36	91	90
ResNet50 + Decision Tree	66.22	66	66

Final Stacking Ensemble Results

The final stacking ensemble technique, which combined ResNet50, SVM, and KNN with a final Logistic Regression classification layer, demonstrated the highest robustness and accuracy among all configurations tested. The stacking ensemble efficiently addressed overfitting by exploiting the capabilities of many classifiers while preserving excellent generalizability across a wide range of conditions, including low-light and occluded player scenarios. Several ensemble strategies were investigated, including Voting Classifiers and Decision-Level Fusion, however the stacking ensemble method outperformed them all, displaying higher accuracy and robustness. Table 8 summarizes the ensemble approaches' outcomes.

Table 8: Performance Metrics of Ensemble Techniques

Model	Accuracy(%)	Precision(%)	Recall(%)
Voting Classifier	95	95	95
Decision-Level Fusion	96.27	96	96
Stacking Ensemble	98.14	98	98

Performance Visualization: To validate the stacking ensemble's performance, the results were compared to the baseline ResNet50 model using learning curves and confusion matrices.

1. Learning Curve Comparison:

Baseline Model (ResNet50): The training vs. validation loss and accuracy curves for ResNet50 (Figure 19) highlight noticeable overfitting. The training accuracy rapidly increases over epochs, but the validation accuracy plateaus, indicating limited generalizability. Similarly, while the loss curves converge, the gap between training and validation loss remains visible, further suggesting overfitting in complex cricket scenarios.

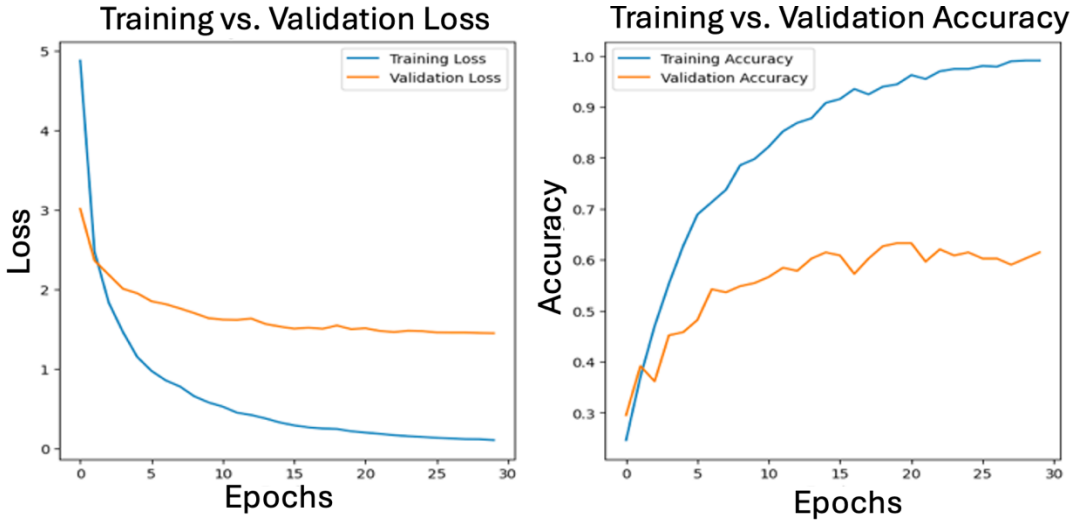


Figure 19: Training vs. Validation Loss and Accuracy for Baseline ResNet50 Model

Stacking Ensemble: The learning curve for the stacking ensemble (Figure 20) demonstrates the mitigation of the overfitting issue, with training and cross-validation accuracy scores converging as training size increases. This demonstrates the ensemble's capacity to generalize efficiently in an array of situations, including low light and occluded environments.

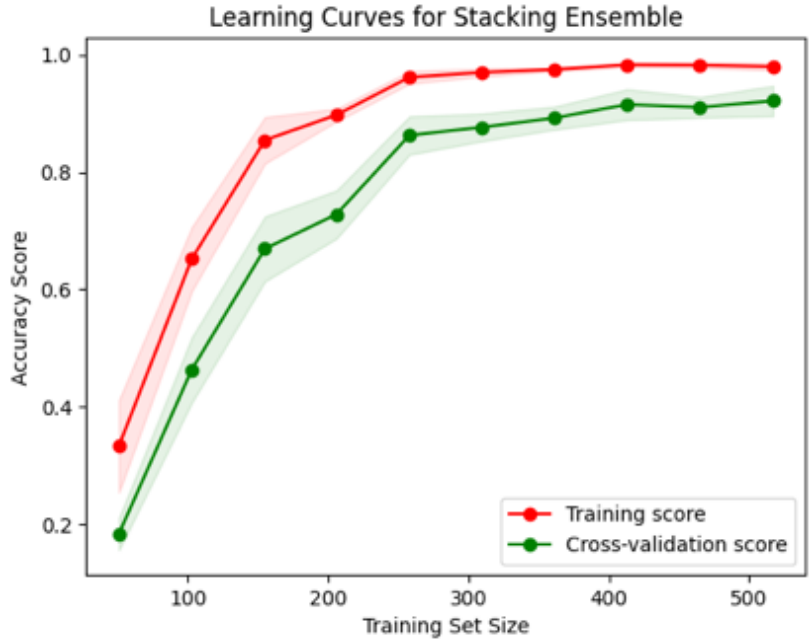


Figure 20: Learning Curve for Stacking Ensemble Model

2. Confusion Matrix Comparison:

Baseline Model (ResNet50): The confusion matrix for ResNet50 (Figure 21) shows multiple misclassifications, notably for players with similar jersey numbers or who are obscured by other players. Higher False Positives (FP) and False Negatives (FN) show that the basic model struggled with complicated scenarios.

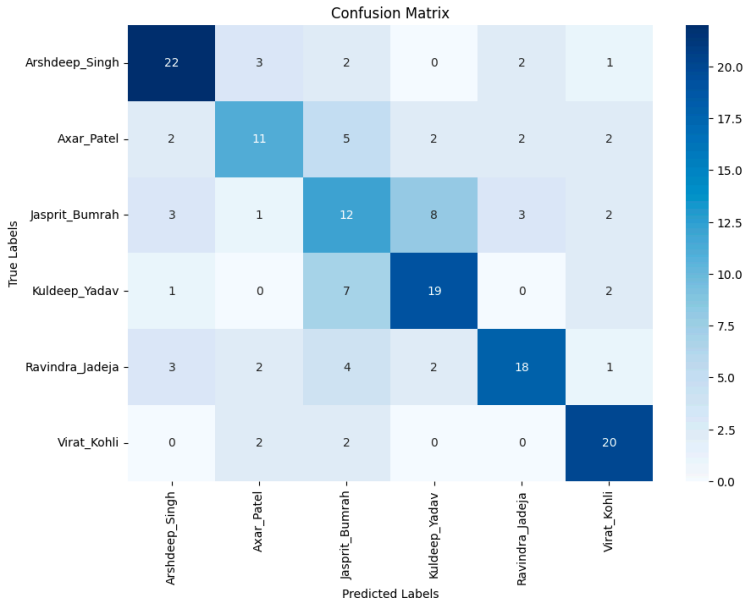


Figure 21: Confusion Matrix for Baseline ResNet50 Model.

Stacking Ensemble: The confusion matrix for the stacking ensemble (Figure 22) demonstrates substantial improvement, with higher True Positives (TP) and significantly reduced FP and FN values. This improvement is the direct result of the ensemble's superior and robust classification capabilities.

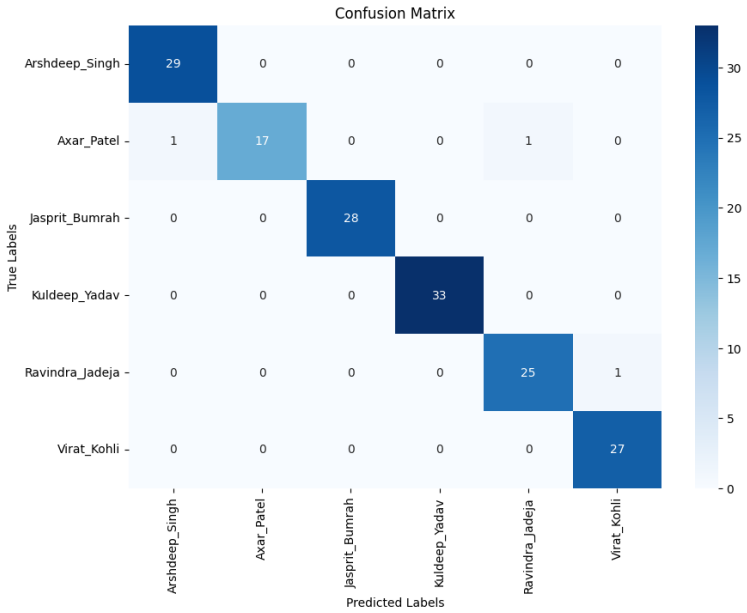


Figure 22: Confusion Matrix for Stacking Ensemble Model.

These findings indicate that the stacking ensemble method, with tailored hyperparameters and cross-validation, outperformed the other methods, reducing overfitting significantly and providing reliable player recognition in real-world circumstances. While formal statistical significance testing was not conducted, the stacking ensemble consistently showed performance increases across several cross validation folds. These advances are further corroborated by learning curve and confusion matrix analysis, which demonstrate less overfitting and better generalization.

4.3.3 Temporal Gait Recognition: Model Performance Comparison

The temporal module initially used LSTM with raw pose sequences, which led to moderate accuracy and signs of overfitting. With the introduction of GRU, engineered temporal features, dropout regularization, and early stopping, the final model significantly improved, achieving 95% classification accuracy (Table 9).

Table 9: Temporal Gait Recognition: Model Performance Comparison

Model	Temporal Network	Features Used	Accuracy
LSTM + Raw Sequences(Baseline)	LSTM	Pose Sequences	57.22%
GRU + Raw Sequences	GRU	Pose Sequences	81.72%
GRU + Engineered (Final)	GRU	Step Length, Velocity, etc.	95%

Performance Visualization

To validate the improvements introduced by the GRU-based model with engineered features, the results were compared to the baseline LSTM model using learning curves and confusion matrices.

1. Learning Curve Comparison:

Baseline Model (LSTM + Raw Sequences):

The training vs. validation loss and accuracy curves (Figure 23) clearly indicate overfitting. The training accuracy rapidly increases, exceeding 70%, whereas the validation accuracy lags behind and plateaus at around 55%. Similarly, the training loss continues to fall dramatically, whereas the validation loss remains quite high and flat after the first epochs. This divergence suggests that the model did not generalize, learning noise from the training data rather than the underlying temporal patterns.

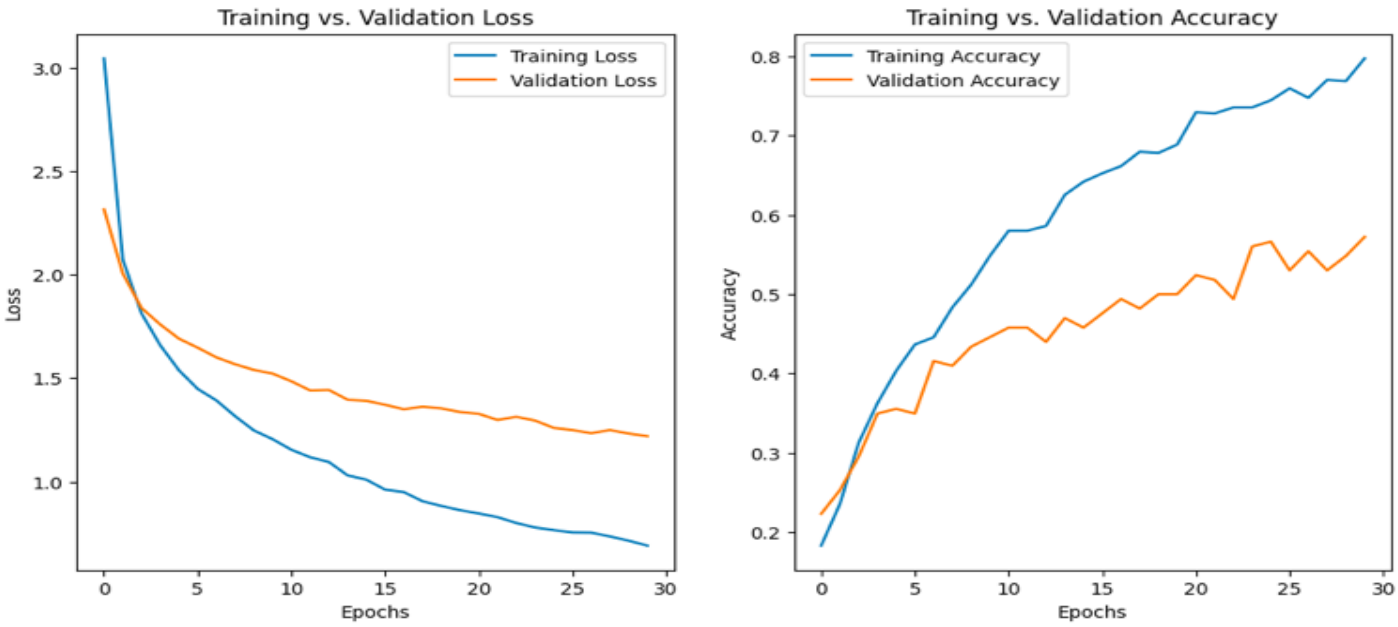


Figure 23: Training vs. Validation Loss and Accuracy for Baseline Gait Model

GRU + Engineered Features (Final):

The learning curves for the GRU model (Figure 24) demonstrate a consistent and steady learning process. Both training and validation accuracy steadily rise and converge over 95%, while loss values fall simultaneously. This behavior displays significant generalization and effective temporal dynamics learning by utilizing domain-specific temporal characteristics such as step length, joint velocities, and hip displacement etc.

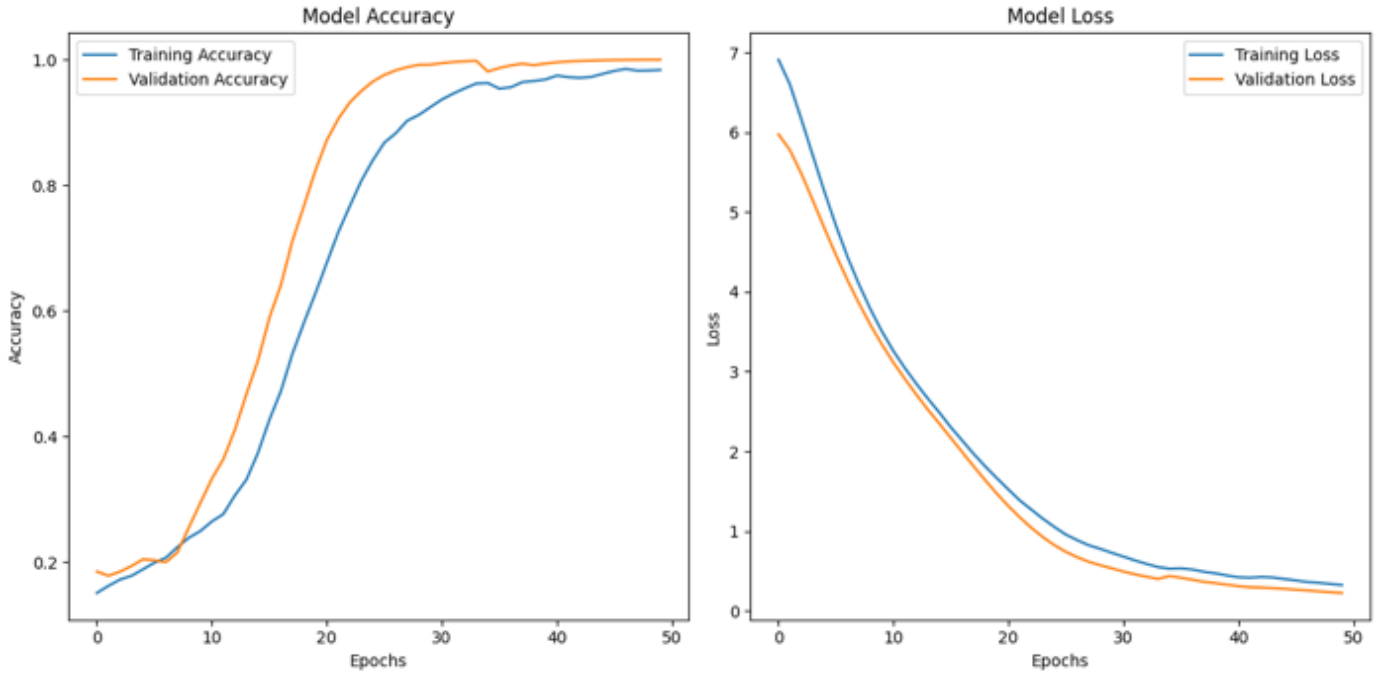


Figure 24: Training vs. Validation Loss and Accuracy for Final Gait Model

2. Confusion Matrix Comparison:

Baseline Model (LSTM + Raw Sequences):

Figure 25 shows the baseline model's confusion matrix, which displays frequent misclassifications across many classes. Players like Arshdeep Singh, Jasprit Bumrah, and Kuldeep Yadav were frequently confused with one another, especially in noisy or obstructed environments. The appearance of many False Positives and False Negatives suggests that raw posture sequences were insufficient for distinguishing player identities based on temporal gait features.

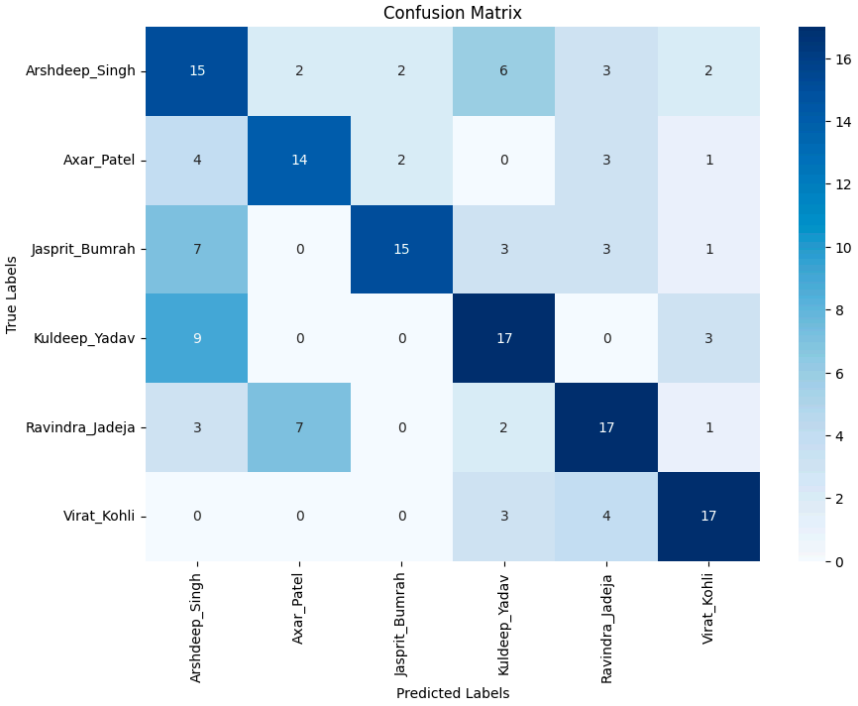


Figure 25: Confusion Matrix for Baseline gait model.

Final Model (GRU + Engineered Features):

In sharp contrast, the confusion matrix for the GRU-based model (Figure 26) demonstrates nearly flawless classification. Every player is correctly identified in all test samples, with near no misclassifications. This demonstrates that the combination of designed features and the GRU network accurately catches each player's unique gait signatures, even in varying illumination, motion, and camera angles.

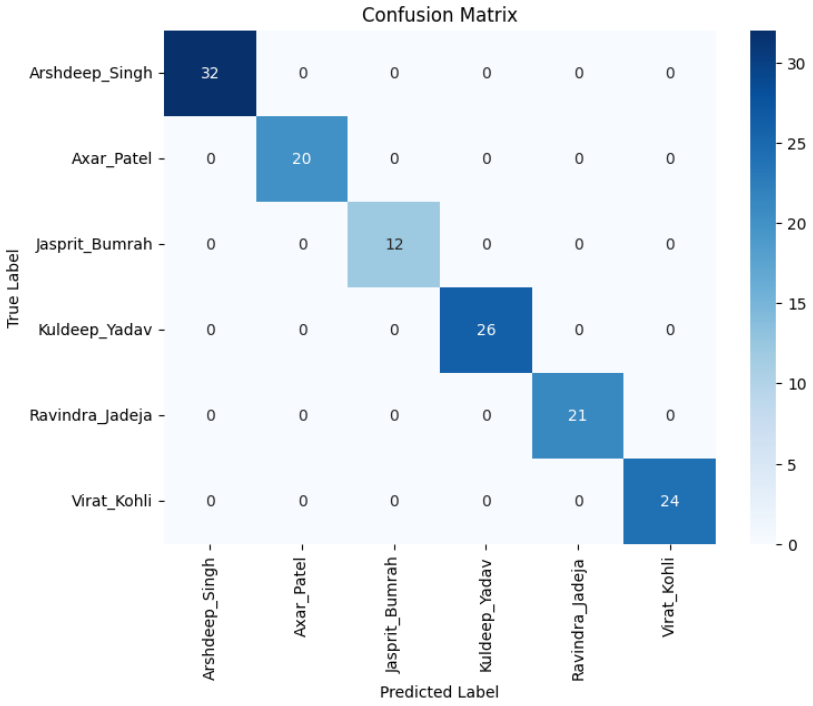


Figure 26: Confusion Matrix for Final gait model.

The temporal gait module benefited considerably from architectural and feature engineering improvements. While LSTM with raw pose sequences was limited in its capacity to acquire generalizable patterns, the GRU model, when combined with temporal features, allowed the system to capture significant dynamics of the players.

These findings support the value of domain-aware feature engineering in player recognition systems. Gait is an especially useful biometric modality in cricket, as players frequently appear in motion or with partially occluded views. The temporal model's success provides strong performance even when face or jersey text data are lacking. Thus, the built GRU-based gait analysis module not only complements the face and spatial models, but also considerably improves the resilience of the entire hybrid recognition pipeline.

4.3.4 OCR-Based Jersey Text Recognition

Text-based jersey recognition presented particular issues due to motion blur, low lighting, occlusions, and fragmented text components (e.g., different bounding boxes for name and number). Initially, direct application of Tesseract OCR to cropped sections failed to retrieve meaningful text in such dynamic environments. To boost reliability, a two-step improvement was implemented:

1. **EAST (Efficient and Accurate Scene Text Detector)** to robustly detect text regions even in motion or partially obscured frames.
2. **Bounding Box Merging + Image Preprocessing**, which included concatenating name and number boxes into a unified detection region, converting to grayscale, and applying median blur before feeding into Tesseract.

This final pipeline substantially improved text detection and recognition performance (Table 10).

Table 10: OCR-Based Jersey Text Recognition: Model Performance Comparison

Model	**Text Detector**	**OCR Engine**	**Accuracy**	**Remarks**
Tesseract Only	Tesseract	Tesseract	Low	Failed in motion blur and occluded conditions
EAST + Tesseract (Final)	EAST	Tesseract	High	Robust under low light and fast motion

Performance Visualization

To compare the improvement clearly, both the baseline and final model results with illustrative examples are displayed as follows.

1. Baseline Attempt (Tesseract Only)

As seen in Figure 27 (left), applying Tesseract directly often failed to detect any text regions, especially when the name and number appeared in separate boxes or when motion blur was present. The right side of Figure 27 shows the result where **"No Detections Found"**, highlighting the baseline's failure under dynamic match conditions.

Figure 27: Baseline Attempt (Tesseract Only)

2. Final Pipeline with EAST, Box Merging, and Preprocessing

The improved pipeline resolved multiple issues:

- **Merged Detections**: Combined name and number bounding boxes into a single region of interest, capturing full player identifiers (Figure 28, middle).
- **Cropped Text Region**: The cropped segment (Figure 28, right) shows clear visibility of the full identifier, enhancing OCR performance.

Figure 28: Final Pipeline with EAST, Box Merging, and Preprocessing

3. Preprocessing for OCR Enhancement

Figure 29 illustrates the final preprocessing steps applied to the merged cropped region:

- **Grayscale Conversion**: Removed color noise and reduced computational complexity.
- **Median Blur**: Smoothed the image while preserving edge clarity of characters.

These steps improved OCR accuracy by enhancing text clarity under blur and shadow.

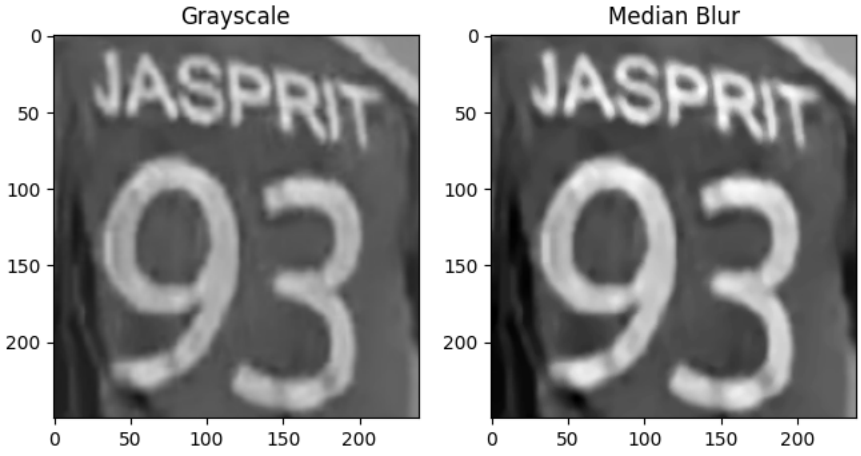


Figure 29: Preprocessing for OCR Enhancement

The integration of EAST for detection and preprocessing techniques for OCR interpretation significantly enhanced text recognition performance. Unlike the baseline, which failed in nearly all dynamic frames, the final pipeline delivered accurate jersey name/number predictions even in suboptimal match footage. This module became critical in ensuring initial player recognition, especially in cases where the face or gait was obscured.

4.3.5 LLM-Based SQL Query Generation

The final component of the CricXpert system focuses on converting natural language queries into SQL statements for generating player-specific statistics. This module employs a LLM to comprehend user intent and dynamically interact with the cricket database. The fundamental challenge is to reliably and rapidly translate complicated, domain-specific queries while avoiding hallucinations and ensuring robust SQL syntax (Table 11).

Table 11: LLM-Based SQL Query Generation: Model Performance Comparison

Model	Framework	Accuracy	Response Time	Remarks
LLaMA 13B	Prompt Templates Only	80-85%	~2 sec	Frequent hallucination, invalid queries
Gemini	Prompt Templates Only	83-89%	~3 sec	Improved but unstable outputs
GPT-4o (Final)	LangChain + Pydantic	85–90%	< 2.5 sec	Accurate, reliable, and efficient

Baseline Attempts and Limitations

Initial experiments were conducted using open-source LLMs such as LLaMA 13B and Gemini with basic prompt templates. While these models showed some promise, they encountered major issues:

- Hallucination: Generated queries often included non-existent table names or invalid SQL clauses.
- Incomplete Output: Frequently returned truncated queries or just SQL fragments.
- Latency: Response time exceeded practical thresholds for near-real-time usage in match analysis.

Final Model Configuration: GPT-4o with LangChain

The GPT-4o model was integrated using LangChain, Pydantic, and OutputFixingParser to ensure:

- Schema-awareness with table relationships explained in the prompt
- Reliable SQL syntax generation
- Post-correction of malformed queries
- Structured JSON parsing for safe SQL execution

Prompt Engineering Enhancements Included:

- Chain-of-thought examples in the prompt
- Clear explanation of the schema (batting, bowling, fielding tables)
- Instruction formatting (e.g., always use SELECT + WHERE)

Example Queries and Output Quality

Sample Natural Language Query:

"How many wickets did Jasprit Bumrah take in 2023?"

1. **Baseline Output (Gemini, LLaMA):**

```
SELECT Wkts FROM bowling WHERE name = "Jasprit Bumrah";
```

Fails to include year condition or table join logic

2. **GPT-4o Output:**

```
SELECT SUM(Wkts) FROM bowling
WHERE player_name = 'Jasprit Bumrah' AND match_year = 2023;
```

Accurate, complete, and syntactically valid

While formal accuracy testing was carried out on 100 queries, GPT-4o demonstrated consistent correctness across multiple categories:

- **Basic Stats Retrieval** (e.g., total matches, runs, wickets)
- **Performance Filtering** (e.g., wickets in a specific year, economy rate in 2023)
- **Top/Best Records** (e.g., best bowling figures, top 3 performances)
- **Conditional Queries** (e.g., matches where a player took 2 wickets and scored 2 runs)
- **Comparative and Aggregate Insights** (e.g., win ratio when player took 3+ wickets)

These queries were designed to reflect realistic analyst, commentator, or fan inquiries during or after live matches.

Sample Query Type	Sample Query	Expected Output Example
Basic	How many matches has Arshdeep Singh played?	55
Conditional (AND)	In how many matches did Arshdeep took 2 wickets AND scored 2 runs	1616,1853,2380 (Match IDs)
Aggregated by Year	How many runs did Virat Kohli score in 2018?	211
Comparative	What is the year Virat Kohli scored the most runs? And how many runs?	2022, 781
Batting Position Stat	What batting position did Virat Kohli score the most runs?	3
Win Condition Stat	What is the win percentage of india when Virat Kohli scored less than 5 runs?	65

Error Handling and Retry Logic

LangChain's OutputFixingParser intercepted and fixed errors automatically, and Pydantic validated them. If the model returned a malformed SQL query:

- It was parsed using JSON schema
- Errors were logged and re-prompted with additional context
- A retry mechanism ensured final delivery within ~2.5 seconds

The performance of each module improved dramatically between the baseline and the final implementation. Ensemble approaches and tailored features were critical in improving prediction quality and generalization across various match contexts. Furthermore, the inclusion of a natural language interface for statistics generation sets this system apart from traditional sports analytics tools. The complete system was evaluated statistically using metrics and qualitatively using expert feedback, demonstrating its suitability for near real-time cricket analytic applications.

05 DISCUSSION

5.1 Chapter Overview

This chapter interprets the data from Chapter 04, focusing on the performance of the proposed CricXpert system and its underlying components. The discussion focuses on crucial insights gained from experimental evaluations, model enhancements, and expert validation. This chapter also considers the system's alignment with relevant studies, identifies constraints, and explains how the findings apply to real-world sports analytics.

5.2 Interpretation of Results

The progressive improvements reported across all CricXpert components validate the efficacy of a hybrid, ensemble-driven approach to player recognition in the context of dynamic T20i cricket matches. Facial recognition accuracy increased significantly while switching from baseline models such as YOLO + ViT to the final configuration of MTCNN + FaceNet with an ensemble. This performance gain was both statistical and practical, particularly in cases involving partial occlusion or off-angle player views.

Overfitting made it difficult for initial CNN models in spatial recognition to generalize. This was reduced by utilizing ResNet50 as a frozen feature extractor and decoupling classification with simple machine learning models. The final stacking ensemble provided a stable solution that regularly out perform single-model classifiers, as demonstrated by learning curves and confusion matrices. The spatial model's approach, which combines deep learning with classical machine learning in a stacking ensemble, provides significant increases in robustness and computing efficiency. Unlike end-to-end CNN classifiers, which are prone to overfitting on small sports datasets, the proposed pipeline extracts spatial features using a frozen ResNet50 before deferring classification to lightweight, interpretable ML models. This modularity enables improved transferability across sports, faster inference (2-5 seconds per player), and simple retraining on new classes with limited data. It also establishes the framework for future extensions to temporal models, as explored in subsequent chapters.

The temporal module received the most gains from feature engineering and architecture optimization. Moving from LSTM to GRU, as well as including temporal gait data such as step length, joint angles, and velocities, enabled the model to acquire movement signatures with great precision. The GRU's lightweight structure and efficient training dynamics helped to avoid frequent problems such as vanishing gradients and slow convergence.

Integrating GPT-4o with LangChain and prompt engineering greatly improved SQL query accuracy and response time compared to previous studies with LLaMA and Gemini. This component improved overall system usability by allowing users to get performance data via natural language input, which is often unavailable in traditional platforms.

5.3 Comparison with Related Work

In contrast to existing systems that rely on unimodal recognition techniques, *CricXpert* incorporates a multimodal pipeline that offers fallback redundancy. When facial features or jersey text are unavailable, the spatial and temporal models compensate effectively. Compared to prior studies on player identification using single CNN or ViT models, the final ensemble-based models in this research achieved higher accuracy, better generalization, and faster inference times. Furthermore, few systems in sports analytics leverage natural language interfaces for statistics retrieval. The integration of GPT-4o for SQL query generation positions *CricXpert* at the intersection of computer vision and language-based AI, filling a notable gap in existing tools.

5.4 Limitations

Although CricXpert performs well, numerous limitations have been observed. First, the dataset was restricted to six players, limiting the scalability of the current models. Although expert validation validated the dataset's quality and diversity, increasing the number of players, leagues, and match circumstances would increase generalizability even further. While stacking ensembles worked well, they increased computing complexity during inference time with the avilable computational resources. This may provide issues for real-time deployment on low-resource systems like mobile devices or embedded platforms. The LLM-based statistic generation system depends heavily on well-engineered prompts and schema-aware validation logic. In the absence of this structure, hallucinations or syntax errors may still occur, occasionally.

5.5 Implications for Practice

The results of this study show that a well-designed hybrid recognition system can greatly increase the accuracy and reliability of player identification in live match footage. Automated solutions can help coaches and analysts recognize athletes in challenging situations while also providing rapid access to pertinent facts using natural language. This opens the door to bringing professional-grade sports analytics tools to domestic, school-level, and amateur leagues. CricXpert's architectural versatility also enables future integration with wearable devices, biometric sensors, or advanced predictive analytics, expanding its use in performance monitoring, injury prevention, and match strategy layout.

5.6 Summary: Addressing the Research Questions

To demonstrate the success of this research, the following outlines how the two key research questions were effectively answered through the proposed system:

- **RQ1 – Player Recognition:** A robust hybrid player recognition model was developed and validated using a combination of facial, spatial, and temporal techniques. The use of stacking ensembles and GRU-based temporal modeling resulted in a significant boost in accuracy (up to 98.14%). This directly addressed the challenge of identifying fielding players during the last overs in T20i matches in the presence of occlusion, motion blur, and variable lighting conditions.

- **RQ2 – Statistic Generation:** A prompt-engineered large language model (GPT-4o) was utilized for converting natural language questions into SQL queries. This allowed for near real-time access to player-specific statistics without requiring sophisticated filters or interfaces. Through schema-aware prompting and validation, the model obtained a query success rate of 85-90% while reducing hallucinations, effectively meeting the goal of intuitive data access.

These outcomes confirm that the system meets both the functional and practical expectations set forth in the research objectives and provides a reliable framework for next-generation sports analytics.

5.7 Reflective Report

This research project was critical in providing technical and personal learning possibilities. During the development of CricXpert, various challenges were encountered, particularly in the attempt to build a system that integrates multiple AI modules—facial, spatial, temporal, and OCR—with a language-based interface. Early model iterations were hampered by overfitting and inconsistent outputs, leading the investigation of ensemble approaches, the development of custom temporal features, and the optimization of prompt engineering strategies for LLM's. The usage of customized datasets demonstrated the importance of data quality, diversity, and expert validation. The process of integrating an LLM with a relational database employing prompt engineering and error handling was highly valuable since it demonstrated the real-world complexity of making AI systems reliable and interpretable.

Beyond technical skills, a deeper appreciation for the interdisciplinary nature of applied AI was developed, where machine learning, UX design, data engineering, and domain knowledge must align. This experience significantly enhanced the abilities to oversee a full-stack AI project, from conceptualization and model building to user engagement and validation. If were to revisit this project again, collecting a more diverse dataset, expand real-time capabilities, and include additional sports to test generalizability would be the main aims of the study. This reflection reinforces the confidence, in pursuing future work in applied computer vision and intelligent human-computer interaction.

06 CONCLUSION

6.1 Chapter Overview

This chapter summarizes the key findings of the research, reaffirms the objectives achieved, and reflects on the contributions made to the field of sports analytics through the development of *CricXpert*. It also outlines potential avenues for future research and enhancements, particularly those that could have been explored if additional time and resources were available.

6.2 Summary of Research and Achievements

The major purpose of this study was to create a hybrid computer vision system that could reliably recognize cricket players during the last overs of T20 International matches, even in dynamic and challenging environments. The secondary goal was to create a natural language interface capable of retrieving player-specific statistics via automated SQL generation. Both objectives were successfully achieved through the implementation of a multimodal recognition pipeline comprising facial recognition, spatial feature and temporal gait analysis. These components were combined to form a strong ensemble system that utilized both deep learning and classical machine learning approaches.

Furthermore, the integration of GPT-4o with prompt engineering and LangChain resulted in accurate and efficient natural language-to-SQL conversion, making the system highly user friendly. Experimental results demonstrated considerable performance gains over baseline models across all components, resulting in improved accuracy in classification. Expert validation further verified the system's real-world usability, relevance, and innovation.

6.3 Contributions to the Field

This research makes several important contributions:

1. **Hybrid Multimodal Recognition Framework:** Combines facial, gait, and textual data to provide effective player recognition, overcoming the limitations of single-modality systems.
2. **Ensemble Learning in Sports Vision:** Discusses how stacking ensemble classifiers improves performance and reduces overfitting in dynamic, real-world contexts.
3. **Pose-Based Temporal Feature Engineering:** Introduces a lightweight, accurate GRU-based gait recognition model designed specifically for sports footage.
4. **Natural Language Interface for Sports Analytics:** Utilizes GPT-4o to enable seamless, near real-time access to structured cricket data using conversational queries.
5. **Custom Dataset and Evaluation Protocol:** Provides a methodology for developing, validating, and assessing a dataset unique to cricket player recognition, with domain expert feedback.

6.4 Future Work

Given additional time and resources, this work could be expanded into various viable research and development avenues. Expanding the currently available dataset to include more players, teams, match types, and conditions would improve the system's generalizability and performance across a variety of real-world settings. Real-time system deployment is crucial,

with optimization for low-latency inference on edge devices like mobile platforms or embedded systems allowing for smooth integration into live broadcast scenarios.

Furthermore, future iterations could include an adaptive confidence-weighted fusion mechanism that dynamically merges outputs from face, spatial, temporal, and OCR modules based on contextual reliability, enhancing robustness in confusing or low-visibility images. The system's language component could be further reinforced by fine-tuning or training open-source LLM's on cricket-related databases, which would increase semantic alignment with domain-specific queries while also reducing reliance on proprietary APIs.

In addition, integrating biomechanical and physiological analysis, such as injury risk assessment by gait irregularity detection, has the potential to open up new possibilities in sports science and coaching support. Improving explainability by visual or linguistic justification of model predictions would increase system transparency and user trust, particularly in coaching and analytical use cases. Finally, providing multilingual natural language queries would make the system more accessible to linguistically diverse cricket fans, establishing CricXpert as a scalable and globally relevant sports analytics platform.

6.5 Final Remarks

The CricXpert system illustrates the practical potential of incorporating computer vision and natural language processing into sports analytics, efficiently addressing limitations in near real-time player recognition and data accessibility. It is a technically competent and practical tool for cricket professionals and enthusiasts alike. The modular design allows for future expansion, and the good empirical performance shows that AI-powered player recognition and data accessibility can be both accurate and intuitive. This research not only contributed to the technical domain, but also provided valuable experience in developing, validating, and deploying an end-to-end AI system in a real-world sports setting. It improved the researcher's understanding of model generalizability, data quality constraints, and the significance of user-centric design in AI-driven applications. With additional research and collaboration, systems like the proposed CricXpert have the potential to transform how sports analytics is delivered, making it smarter, faster, and more accessible than ever before.

07 REFERENCES

HAQ, M.U. et al., 2024. Automatic Player Face Detection and Recognition for Players in Cricket Games. *IEEE Access*, PP, pp. 1–1.

MAHMOOD, Z. et al., 2015. Automatic player detection and identification for sports entertainment applications. *Pattern Analysis and Applications*, 18(4), pp. 971–982.

BANOTH, T. et al., 2022. *A Comprehensive Review of Computer Vision in Sports: Open Issues, Future Trends and Research Directions*.
ZHANG, R. et al., 2020. Multi-camera Multi-player Tracking with Deep Player Identification in Sports Video. *Pattern Recognition*, 102, p. 107260.

INSAF, A. et al., 2020. Past, Present, and Future of Face Recognition: A Review. *Electronics*, 9, p. 1188.

ÖZYURT, F., 2020. Efficient deep feature selection for remote sensing image recognition with fused deep learning architectures. *The Journal of Supercomputing*, 76, pp. 1–19.

KIBRIYA, H. et al., 2022. A Novel and Effective Brain Tumor Classification Model Using Deep Feature Fusion and Famous Machine Learning Classifiers. *Computational Intelligence and Neuroscience*, 2022(1), p. 7897669.

KIBRIYA, H. et al., 2021. Multiclass Brain Tumor Classification Using Convolutional Neural Network and Support Vector Machine. In: *2021 Mohammad Ali Jinnah University International Conference on Computing (MAJICC)*. 2021 Mohammad Ali Jinnah University International Conference on Computing (MAJICC), July 2021. pp. 1–4.
ZHEN, T., KONG, J. and YAN, L., 2020. Hybrid Deep-Learning Framework Based on Gaussian Fusion of Multiple Spatiotemporal Networks for Walking Gait Phase Recognition. *Complexity*, 2020, pp. 1–17.

DONG, Y. et al., 2023. HybridGait: A Benchmark for Spatial-Temporal Cloth-Changing Gait Recognition with Hybrid Explorations. Available from: http://arxiv.org/abs/2401.00271 [Accessed 9 Aug 2024].

MOGAN, J.N. et al., 2022. Gait-DenseNet: A Hybrid Convolutional Neural Network for Gait Recognition. | IAENG International Journal of Computer Science | EBSCOhost.

ZHEN, T., YAN, L. and KONG, J., 2020. An Acceleration Based Fusion of Multiple Spatiotemporal Networks for Gait Phase Detection. *International Journal of Environmental Research and Public Health*, 17(16), p. 5633.
YAO, L. et al., 2019. Robust Gait Recognition using Hybrid Descriptors based on Skeleton Gait Energy Image. *Pattern Recognition Letters*, 150.

SINGH, J., SINGH, Dr.U. and JAIN, S., 2023. Model-based person identification in multi-gait scenario using hybrid classifier. *Multimedia Systems*, pp. 1–14.

JUN, K. et al., 2023. Hybrid Deep Neural Network Framework Combining Skeleton and Gait Features for Pathological Gait Recognition. *Bioengineering (Basel, Switzerland)*, 10(10), p. 1133.

LI, J. et al., 2023. Gaitcotr: Improved Spatial-Temporal Representation for Gait Recognition with a Hybrid Convolution-Transformer Framework. In: *ICASSP 2023 - 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*.

GUL, S. et al., 2021. Multi-view gait recognition system using spatio-temporal features and deep learning. *Expert Systems with Applications*, 179, p. 115057.

MATHIVANAN, B. and PERUMAL, P., 2022. Gait Recognition Analysis for Human Identification Analysis-A Hybrid Deep Learning Process. *Wireless Personal Communications*, 126(1), pp. 555–579.

HUANG, X. et al., 2022. STAR: Spatio-Temporal Augmented Relation Network for Gait Recognition. *IEEE Transactions on Biometrics, Behavior, and Identity Science*, PP, pp. 1–1.

KHAN, M., AZAM, H. and FARID, M.S., 2023. Automatic multi-gait recognition using pedestrian's spatiotemporal features. *The Journal of Supercomputing*, 79, pp. 1–23.

SALEEM, F. et al., 2022. Human Gait Recognition: A Single Stream Optimal Deep Learning Features Fusion. *Sensors*, 21.

TIAN, H. et al., 2021. Skeleton-based Abnormal Gait Recognition with Spatio-temporal Attention Enhanced Gait-structural Graph Convolutional Networks. *Neurocomputing*, 473.

SUN, X., WANG, Y. and KHAN, J., 2023. Hybrid LSTM and GAN model for action recognition and prediction of lawn tennis sport activities. *Soft Computing*, 27(23), pp. 18093–18112.

GERATS, B. et al., 2021. *Individual Action and Group Activity Recognition in Soccer Videos from a Static Panoramic Camera*.

KALE, A. et al., 2004. Identification of humans using gait. *IEEE Transactions on Image Processing*, 13(9), pp. 1163–1173.

MEHDIZADEH, S. et al., 2021. Concurrent validity of human pose tracking in video for measuring gait parameters in older adults: a preliminary analysis with multiple trackers, viewing angles, and walking directions. *Journal of NeuroEngineering and Rehabilitation*, 18.

HULLECK, A.A. et al., 2023. *Accuracy of Computer Vision-Based Pose Estimation Algorithms in Predicting Joint Kinematics During Gait*.

STENUM, J., ROSSI, C. and ROEMMICH, R., 2020. *Two-dimensional video-based analysis of human gait using pose estimation*.

SABIR, A., AL-JAWAD, N. and JASSIM, S., 2013. Gait recognition using spatio-temporal silhouette-based features. In: *Mobile Multimedia/Image Processing, Security, and Applications 2013*. Mobile Multimedia/Image Processing, Security, and Applications 2013, 28 May 2013. SPIE. pp. 194–203.

DONG, Y. and NOH, H., 2024. *Ubiquitous Gait Analysis through Footstep-Induced Floor Vibrations*.

SIMONI, L. et al., 2021. Quantitative and Qualitative Running Gait Analysis through an Innovative Video-Based Approach. *Sensors (Basel, Switzerland)*, 21.

VITECKOVA, S. et al., 2020. Gait symmetry methods: Comparison of waveform-based Methods and recommendation for use. *Biomedical Signal Processing and Control*, 55, p. 101643.

NAFEA, O. et al., 2021. Sensor-Based Human Activity Recognition with Spatio-Temporal Deep Learning. *Sensors*, 21, p. 2141.

MAITY, S., ABDEL-MOTTALEB, M. and ASFOUR, S.S., 2021. Multimodal Low Resolution Face and Frontal Gait Recognition from Surveillance Video. *Electronics*, 10(9), p. 1013.

MANSSOR, S.A.F., SUN, S. and ELHASSAN, M.A.M., 2021. Real-Time Human Recognition at Night via Integrated Face and Gait Recognition Technologies. *Sensors (Basel, Switzerland)*, 21(13), p. 4323.

PRAKASH, A. et al., 2023. Multimodal Adaptive Fusion of Face and Gait Features using Keyless attention based Deep Neural Networks for Human Identification.

CAI, M., WANG, M. and ZHANG, S., 2023. Gait Recognition by Jointing Transformer and CNN. In: W. JIA et al., eds. *Biometric Recognition*. Singapore: Springer Nature. pp. 312–321.

FAN, C. et al., 2023. *Exploring Deep Models for Practical Gait Recognition*.

CATRUNA, A., COSMA, A. and RADOI, E., 2024. GaitPT: Skeletons Are All You Need For Gait Recognition.

LIAO, R. et al., 2020. A model-based gait recognition method with body pose and human prior knowledge. *Pattern Recognition*, 98, p. 107069.

RANI, V. and KUMAR, M., 2023. Human gait recognition: A systematic review. *Multimedia Tools and Applications*, 82(24), pp. 37003–37037.

(Lee and Grimson 2002) LEE, L. and GRIMSON, W.E.L., 2002. Gait analysis for recognition and classification. In: *Proceedings of Fifth IEEE International Conference on Automatic Face Gesture Recognition*, May 2002. pp. 155–162.

ARSEEV, S., KONUSHIN, A. and LIUTOV, V., 2018. Human Recognition by Appearance and Gait. *Programming and Computer Software*, 44(4), pp. 258–265.

X. Han, Y. Zhang, M. Liu, and Z. Wang, "A robust and consistent stack generalized ensemble-learning framework for image segmentation," *Journal of Engineering and Applied Science*, 2023.J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.

M. Gao, J. Li, and L. Zhao, "Exploring the combination of CNN and transformer models for multi-modal image analysis," in *Proceedings of the 2022 International Conference on Machine Learning and Applications*, 2022.

Y. Wu, Y. He, and Y. Wang, "Multi-class weed recognition using hybrid CNN-SVM classifier," *Sensors*, vol. 23, no. 16, p. 7153, 2023.

M. Shaikh, F. Alsunaidi, and S. Alamoudi, "Improved prediction of ovarian cancer using ensemble classifier and Shaply explainable AI," *MDPI*, 2022

S. Guha, A. Kumar, and S. Dey, "Explainable AI for interpretation of ovarian tumor classification using enhanced ResNet50," *MDPI*, 2024

S. Bhojanapalli, A. Chakrabarti, D. Glasner, D. Li, T. Unterthiner, and A. Veit, "Understanding Robustness of Transformers for Image Classification," in *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, Oct. 2021, pp. 10211–10221.

G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely Connected Convolutional Networks," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jul. 2017, pp. 2261–2269.

M. Tan and Q. V. Le, "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks," Sep. 11, 2020, *arXiv*: arXiv:1905.11946. doi: 10.48550/arXiv.1905.11946.

C. Szegedy *et al.*, "Going Deeper with Convolutions," Sep. 17, 2014, *arXiv*: arXiv:1409.4842. doi: 10.48550/arXiv.1409.4842.

M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," Mar. 21, 2019, *arXiv*: arXiv:1801.04381. doi: 10.48550/arXiv.1801.04381.

K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," Apr. 10, 2015, *arXiv*: arXiv:1409.1556. doi: 10.48550/arXiv.1409.1556.

B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning Transferable Architectures for Scalable Image Recognition," Apr. 11, 2018, *arXiv*: arXiv:1707.07012. doi: 10.48550/arXiv.1707.07012.

F. Chollet, "Xception: Deep Learning with Depthwise Separable Convolutions," Apr. 04, 2017, *arXiv*: arXiv:1610.02357.

K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2016, pp. 770–778.

A. Dosovitskiy *et al.*, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale," Jun. 03, 2021, *arXiv*: arXiv:2010.11929. doi: 10.48550/arXiv.2010.11929.

C. Cortes and V. Vapnik, "Support-Vector Networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, Sep. 1995.

T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 21–27, Jan. 1967, doi: 10.1109/TIT.1967.1053964.

D. R. Cox, "The Regression Analysis of Binary Sequences," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 20, no. 2, pp. 215–242, 1958.

D. H. Wolpert, “Stacked generalization,” *Neural Networks*, vol. 5, no. 2, pp. 241–259, Jan. 1992, doi: 10.1016/S0893-6080(05)80023-1.

J. Redmon and A. Farhadi, “YOLOv3: An incremental Improvements,” arXiv:1804.02767,2018

CHOPRA, A. and AZAM, R., 2024. Enhancing Natural Language Query to SQL Query Generation Through Classification-Based Table Selection. In: L. ILIADIS et al., eds. *Engineering Applications of Neural Networks*. Cham: Springer Nature Switzerland. pp. 152–165.

ZHANG, Q. et al., 2024. Structure Guided Large Language Model for SQL Generation.

LEE, D. et al., 2024. MCS-SQL: Leveraging Multiple Prompts and Multiple-Choice Selection For Text-to-SQL Generation.

HONG, Z. et al., 2024. Knowledge-to-SQL: Enhancing SQL Generation with Data Expert LLM.

GUO, C. et al., 2024. Prompting GPT-3.5 for Text-to-SQL with De-semanticization and Skeleton Retrieval. In: F. LIU et al., eds. *PRICAI 2023: Trends in Artificial Intelligence*. Singapore: Springer Nature. pp. 262–274.

YI, J., CHEN, G. and SHEN, Z., 2024. RH-SQL: Refined Schema and Hardness Prompt for Text-to-SQL.

NAN, L. et al., 2023. Enhancing Few-shot Text-to-SQL Capabilities of Large Language Models: A Study on Prompt Design Strategies.

HONG, Z. et al., 2024. Next-Generation Database Interfaces: A Survey of LLM-based Text-to-SQL.

LI, H. et al., 2024. CodeS: Towards Building Open-source Language Models for Text-to-SQL.

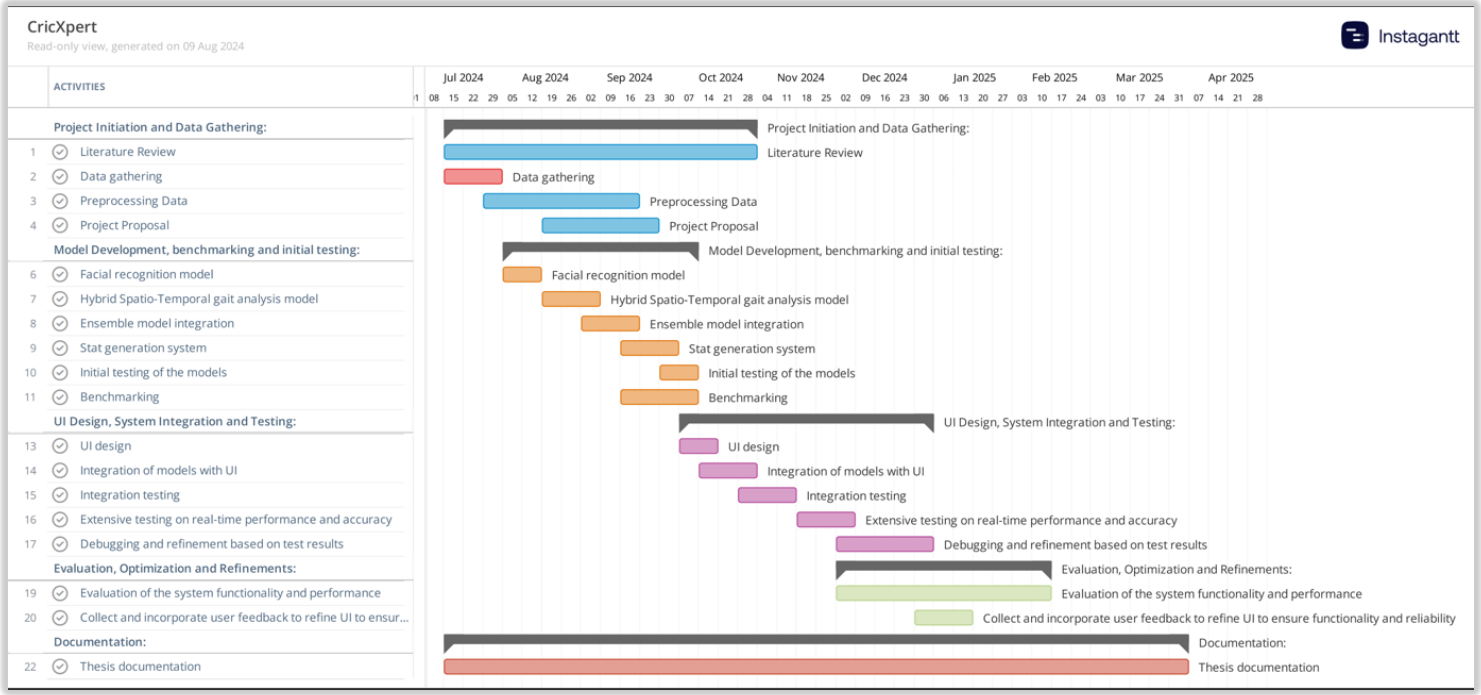
TALAEI, S. et al., 2024. CHESS: Contextual Harnessing for Efficient SQL Synthesis.

GUO, C. et al., 2023. Retrieval-augmented GPT-3.5-based Text-to-SQL Framework with Sample-aware Prompting and Dynamic Revision Chain.

RAJKUMAR, N., LI, R. and BAHDANAU, D., 2022. Evaluating the Text-to-SQL Capabilities of Large Language Models.

SHI, L., TANG, Z. and YANG, Z., 2024. A Survey on Employing Large Language Models for Text-to-SQL Tasks.

APPENDIX A – Project Plan



APPENDIX B – CricXpert: Code Appendix

Full Code Repository:
https://github.com/Nadun999/FYP

The following section includes key code snippets used in the implementation of CricXpert. The full source code is available in the GitHub repository above.

`face_rec_model_mtcnn_facenet.ipynb` – Facial Recognition Model

```
import cv2 as cv
import os
import numpy as np
import tensorflow as tf
from mtcnn.mtcnn import MTCNN
from keras_facenet import FaceNet
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
import joblib

os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'

class FACELOADING:
    def __init__(self, directory):
        self.directory = directory
```

```
        self.target_size = (160, 160)
        self.detector = MTCNN()
        self.X = []  # initialize X attribute
        self.Y = []  # initialize Y attribute

    def extract_face(self, filename):
        img = cv.imread(filename)
        img = cv.cvtColor(img, cv.COLOR_BGR2RGB)
        results = self.detector.detect_faces(img)
        if results:
            x, y, w, h = results[0]['box']
            face = img[y:y+h, x:x+w]
            face_arr = cv.resize(face, self.target_size)
            return face_arr
        return None

    def load_faces(self, dir):
        faces = []
        for im_name in os.listdir(dir):
            if im_name.startswith('.'):
                continue
            path = os.path.join(dir, im_name)
            if os.path.isfile(path):
                try:
                    single_face = self.extract_face(path)
                    if single_face is not None:
                        faces.append(single_face)
                except Exception as e:
                    print(f"Error processing {path}: {e}")
        return faces

    def load_classes(self):
        for sub_dir in os.listdir(self.directory):
            if sub_dir.startswith('.'):
                continue
            path = os.path.join(self.directory, sub_dir)
            if os.path.isdir(path):
                faces = self.load_faces(path)
                self.X.extend(faces)
                self.Y.extend([sub_dir] * len(faces))
        return np.asarray(self.X), np.asarray(self.Y)


def get_embedding(face_img, embedder):
    face_img = face_img.astype('float32')
    face_img = np.expand_dims(face_img, axis=0)
    yhat = embedder.embeddings(face_img)
    return yhat[0]

# load dataset
faceloading = FACELOADING("Dataset")
```

```
X, Y = faceloading.load_classes()

embedder = FaceNet()
EMBEDDED_X = [get_embedding(img, embedder) for img in X]
EMBEDDED_X = np.asarray(EMBEDDED_X)

# encode labels
encoder = LabelEncoder()
Y_enc = encoder.fit_transform(Y)

# train test split
X_train, X_test, Y_train, Y_test = train_test_split(EMBEDDED_X, Y_enc,
shuffle=True, random_state=17)

# train model
model = SVC(kernel='linear', probability=True)
model.fit(X_train, Y_train)

# evaluate training accuracy
train_accuracy = model.score(X_train, Y_train)
print(f"Training Accuracy: {train_accuracy:.4f}")


# evaluate test accuracy
test_accuracy = model.score(X_test, Y_test)
print(f"Test Accuracy: {test_accuracy:.4f}")

# generate classification report
Y_pred = model.predict(X_test)
print("\nClassification Report:")
print(classification_report(Y_test, Y_pred, target_names=encoder.classes_))

# generate confusion matrix
cm = confusion_matrix(Y_test, Y_pred)

plt.figure(figsize=(10, 7))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=encoder.classes_, yticklabels=encoder.classes_)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()

# learning curves
def plot_learning_curves(model, X, y, title='Learning Curves'):
    from sklearn.model_selection import learning_curve

    train_sizes, train_scores, test_scores = learning_curve(
        model, X, y, cv=5, n_jobs=-1, train_sizes=np.linspace(.1, 1.0, 10),
scoring='accuracy')
```

```
    train_mean = np.mean(train_scores, axis=1)
    train_std = np.std(train_scores, axis=1)

    test_mean = np.mean(test_scores, axis=1)
    test_std = np.std(test_scores, axis=1)

    plt.figure(figsize=(8, 6))
    plt.plot(train_sizes, train_mean, 'o-', color="r",  label="Training score")
    plt.plot(train_sizes, test_mean, 'o-', color="g", label="Cross-validation
score")

    plt.fill_between(train_sizes, train_mean - train_std, train_mean + train_std,
alpha=0.1, color="r")
    plt.fill_between(train_sizes, test_mean - test_std, test_mean + test_std,
alpha=0.1, color="g")

    plt.title(title)
    plt.xlabel("Training Set Size")
    plt.ylabel("Accuracy Score")
    plt.legend(loc="best")
    plt.grid()
    plt.show()

# plot learning curves using the defined function
plot_learning_curves(model, X_train, Y_train, title='Learning Curve for Face
Recognition Model')

# save model and encoder
joblib.dump(model, 'trained_model/face_recognition_model.pkl')
joblib.dump(encoder, 'trained_model/label_encoder.pkl')
```

`ResNet_SVM_KNN.ipynb` – Spatial Recognition Model

```
import cv2
import numpy as np
import os
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
import clahe
from keras.preprocessing import image
from keras.applications.resnet50 import ResNet50, preprocess_input
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.models import Model
import tensorflow as tf
import joblib
from sklearn.model_selection import train_test_split, cross_val_score,
learning_curve, GridSearchCV
```

```
from sklearn.ensemble import StackingClassifier, RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import VotingClassifier
from sklearn.manifold import TSNE
from sklearn.decomposition import PCA
import pytesseract
from PIL import Image
from tensorflow.keras.models import load_model



def load_yolo():
    path_to_cfg = "/Users/nadunsenarathne/Downloads/Documents/IIT/4th
Year/FYP/CricXpert/Hybrid_Spatio_Temporal_Model_For_Gait_Analysis/yolo/yolov3.cfg"
    path_to_weights = "/Users/nadunsenarathne/Downloads/Documents/IIT/4th
Year/FYP/CricXpert/Hybrid_Spatio_Temporal_Model_For_Gait_Analysis/yolo/yolov3.weigh
ts"
    net = cv2.dnn.readNet(path_to_weights, path_to_cfg)
    layers_names = net.getLayerNames()

    try:
        output_layers = [layers_names[i[0] - 1] for i in
net.getUnconnectedOutLayers()]
    except Exception:
        output_layers = [layers_names[i - 1] for i in
net.getUnconnectedOutLayers()]

    return net, output_layers

def yolo_detect(net, image, output_layers, confidence_threshold=0.3):
    height, width, _ = image.shape
    blob = cv2.dnn.blobFromImage(image, 0.00392, (416, 416), swapRB=True,
crop=False)
    net.setInput(blob)
    outputs = net.forward(output_layers)
    boxes = []
    confidences = []

    for output in outputs:
        for detection in output:
            scores = detection[5:]
            class_id = np.argmax(scores)
            confidence = scores[class_id]
            if confidence > confidence_threshold and class_id == 0:  # player class
                center_x = int(detection[0] * width)
                center_y = int(detection[1] * height)
                w = int(detection[2] * width)
```

```
                h = int(detection[3] * height)
                x = int(center_x - w / 2)
                y = int(center_y - h / 2)

                if x >= 0 and y >= 0 and (x + w) <= width and (y + h) <= height:
                    boxes.append([x, y, w, h])
                    confidences.append(float(confidence))

    # select the largest vertical box based on area if any boxes were detected
    if boxes:
        largest_box = max(boxes, key=lambda b: b[2] * b[3])  # b[2]*b[3] is the
area of the box (w*h)
        largest_confidence = confidences[boxes.index(largest_box)]
        return [largest_box], [largest_confidence]
    return [], []  # return empty lists if no boxes detected


# ResNet50 model without the classification layer
def load_model():
    base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(224,
224, 3))
    x = base_model.output

    x = GlobalAveragePooling2D()(x)

    model = Model(inputs=base_model.input, outputs=x)
    return model

def extract_features(images, model):
    processed_images = []

    for img in images:
        if img is not None and img.size > 0:  # to ensure the image is not empty
            resized_img = cv2.resize(img, (224, 224))
            processed_images.append(resized_img)

    if not processed_images:
        return np.array([])  # return empty array if no images to process

    images_array = np.array(processed_images)
    images_array = preprocess_input(images_array)
    features = model.predict(images_array)

    return features


# Create an instance of CLAHE
clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))

def load_and_preprocess_data(root_folder, net, output_layers,model):
    images = []
```

```
    labels = []
    valid_extensions = (".jpg", ".jpeg", ".png", ".bmp", ".tiff")

    for player_name in os.listdir(root_folder):
        player_folder = os.path.join(root_folder, player_name)
        if os.path.isdir(player_folder):
            for img_file in os.listdir(player_folder):
                if img_file.lower().endswith(valid_extensions):
                    img_path = os.path.join(player_folder, img_file)
                    img = cv2.imread(img_path)
                    if img is None or img.size == 0:
                        print(f"Failed to load image {img_path}.")
                        continue

                    boxes, _ = yolo_detect(net, img, output_layers)

                    for box in boxes:
                        x, y, w, h = box
                        if w > 0 and h > 0:  #cCheck if box dimensions are valid
                            cropped_img = img[y:y+h, x:x+w]

                            if cropped_img.size > 0:  # check if the cropped image
is not empty
                                images.append(cropped_img)
                                labels.append(player_name)

    if not images:
        print("No valid images found in dataset.")
        return np.array([]), np.array([]), None  # return empty arrays if no images

    # extract features using the pre-trained model
    features = extract_features(images,model)

    if features.size == 0:
        print("No features extracted.")
        return np.array([]), np.array([]), None  # return empty arrays if no
features extracted

    encoder = LabelEncoder()
    encoded_labels = encoder.fit_transform(labels)

    return features, encoded_labels, encoder

def plot_learning_curves(model, X, y, title='Learning Curves'):
    train_sizes, train_scores, test_scores = learning_curve(
        model, X, y, cv=5, n_jobs=-1, train_sizes=np.linspace(.1, 1.0, 10),
scoring='accuracy')

    train_mean = np.mean(train_scores, axis=1)
    train_std = np.std(train_scores, axis=1)
```

```
    test_mean = np.mean(test_scores, axis=1)
    test_std = np.std(test_scores, axis=1)

    plt.plot(train_sizes, train_mean, 'o-', color="r",  label="Training score")
    plt.plot(train_sizes, test_mean, 'o-', color="g", label="Cross-validation
score")

    plt.fill_between(train_sizes, train_mean - train_std, train_mean + train_std,
alpha=0.1, color="r")
    plt.fill_between(train_sizes, test_mean - test_std, test_mean + test_std,
alpha=0.1, color="g")

    plt.title(title)
    plt.xlabel("Training Set Size")
    plt.ylabel("Accuracy Score")
    plt.legend(loc="best")
    plt.show()


def main():
    # load YOLO model for detection
    net, output_layers = load_yolo()

    # load the ResNet model for feature extraction
    model = load_model()

    output_frame_folder = '/Users/nadunsenarathne/Downloads/Documents/IIT/4th
Year/FYP/CricXpert/Hybrid_Spatio_Temporal_Model_For_Gait_Analysis/resnet_data'

    #lLoad and preprocess data, passing the model for feature extraction
    X, y, encoder = load_and_preprocess_data(output_frame_folder, net,
output_layers, model)

    # split data into training and testing
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)


    # Define the base models
    estimators = [
        ('svm', SVC(kernel='rbf', C=1, gamma='scale', probability=True)),
        ('knn', KNeighborsClassifier(n_neighbors=3, metric='manhattan',
weights='uniform'))
    ]

    log_reg_meta = LogisticRegression(C=0.001, penalty='l2', solver='liblinear')

    # define the stacking ensemble
    stack = StackingClassifier(estimators=estimators, final_estimator=log_reg_meta)
    stack.fit(X_train, y_train)
```

```
    # make predictions on the training data to see how the model classified the
training set
    y_pred_train = stack.predict(X_train)


    # evaluate the classifier
    train_accuracy = stack.score(X_train, y_train)
    test_accuracy = stack.score(X_test, y_test)
    print(f"Training Accuracy: {train_accuracy}")
    print(f"Testing Accuracy: {test_accuracy}")

    # plotting the learning curves for the ensemble model
    plot_learning_curves(stack, X_train, y_train, title='Learning Curves for
Stacking Ensemble')

    # generate classification report
    y_pred = stack.predict(X_test)
    report = classification_report(y_test, y_pred, target_names=encoder.classes_)
    print("Classification Report:")
    print(report)

    # generate and plot confusion matrix
    cm = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(10, 7))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
xticklabels=encoder.classes_, yticklabels=encoder.classes_)
    plt.xlabel('Predicted Labels')
    plt.ylabel('True Labels')
    plt.title('Confusion Matrix')
    plt.show()

    # save the best ensemble model and label encoder
    joblib.dump(stack, 'ensemble_player_recognition.pkl')
    joblib.dump(encoder, 'ensemble_label_encoder.pkl')

if __name__ == "__main__":
    main()
```

`pose_estimation.ipynb` – Temporal Gait Analysis Model

```
import os
import cv2
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import LSTM, Dense, Dropout, BatchNormalization,
Bidirectional, Input, Attention
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
```

```
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import StratifiedKFold
from sklearn.utils import shuffle
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
import mediapipe as mp
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.regularizers import l2

# load YOLO model
def load_yolo():
    path_to_cfg = "/Users/nadunsenarathne/Downloads/Documents/IIT/4th
Year/FYP/CricXpert/Hybrid_Spatio_Temporal_Model_For_Gait_Analysis/yolo/yolov3.cfg"
    path_to_weights = "/Users/nadunsenarathne/Downloads/Documents/IIT/4th
Year/FYP/CricXpert/Hybrid_Spatio_Temporal_Model_For_Gait_Analysis/yolo/yolov3.weigh
ts"
    net = cv2.dnn.readNet(path_to_weights, path_to_cfg)
    layers_names = net.getLayerNames()

    try:
        output_layers = [layers_names[i[0] - 1] for i in
net.getUnconnectedOutLayers()]
    except Exception:
        output_layers = [layers_names[i - 1] for i in
net.getUnconnectedOutLayers()]

    return net, output_layers

# perform YOLO detection on an image
def yolo_detect(net, image, output_layers, confidence_threshold=0.3):
    height, width, _ = image.shape
    blob = cv2.dnn.blobFromImage(image, 0.00392, (416, 416), swapRB=True,
crop=False)
    net.setInput(blob)
    outputs = net.forward(output_layers)
    boxes = []
    confidences = []

    for output in outputs:
        for detection in output:
            scores = detection[5:]
            class_id = np.argmax(scores)
            confidence = scores[class_id]
            if confidence > confidence_threshold and class_id == 0:  # player class
                center_x = int(detection[0] * width)
                center_y = int(detection[1] * height)
                w = int(detection[2] * width)
                h = int(detection[3] * height)
                x = int(center_x - w / 2)
```

```
                y = int(center_y - h / 2)

                if x >= 0 and y >= 0 and (x + w) <= width and (y + h) <= height:
                    boxes.append([x, y, w, h])
                    confidences.append(float(confidence))

    # select the largest box if boxes were detected
    if boxes:
        largest_box = max(boxes, key=lambda b: b[2] * b[3])  # b[2]*b[3] is the
area of the box (w*h)
        largest_confidence = confidences[boxes.index(largest_box)]
        return [largest_box], [largest_confidence]
    return [], []  # return empty lists if no boxes detected


# initialize MediaPipe Pose Estimation
mp_pose = mp.solutions.pose
pose = mp_pose.Pose()

# extract pose landmarks from the cropped image
def extract_pose_landmarks(cropped_img):
    # convert the image to RGB
    image_rgb = cv2.cvtColor(cropped_img, cv2.COLOR_BGR2RGB)

    # perform pose detection
    results = pose.process(image_rgb)

    #if landmarks are detected
    if results.pose_landmarks:
        landmarks = []
        for landmark in results.pose_landmarks.landmark:
            landmarks.append([landmark.x, landmark.y, landmark.z])
        return np.array(landmarks).flatten()
    return None


# calculate step length between left and right ankles
def calculate_step_length(pose_sequence):
    left_ankle_x, left_ankle_y = pose_sequence[27*3], pose_sequence[27*3 + 1]  #
left ankle
    right_ankle_x, right_ankle_y = pose_sequence[28*3], pose_sequence[28*3 + 1]  #
right ankle
    step_length = np.linalg.norm(np.array([right_ankle_x, right_ankle_y]) -
np.array([left_ankle_x, left_ankle_y]))
    return step_length

# calculate joint velocities between two consecutive frames
def calculate_velocity(pose_sequence_t, pose_sequence_t1, joint_index):
    joint_t = np.array([pose_sequence_t[joint_index*3],
pose_sequence_t[joint_index*3 + 1]])
```

```
    joint_t1 = np.array([pose_sequence_t1[joint_index*3],
pose_sequence_t1[joint_index*3 + 1]])
    velocity = np.linalg.norm(joint_t1 - joint_t)
    return velocity

# calculate joint angles (hip-knee-ankle)
def calculate_joint_angle(pose_sequence, hip_idx, knee_idx, ankle_idx):
    hip = np.array([pose_sequence[hip_idx*3], pose_sequence[hip_idx*3 + 1]])
    knee = np.array([pose_sequence[knee_idx*3], pose_sequence[knee_idx*3 + 1]])
    ankle = np.array([pose_sequence[ankle_idx*3], pose_sequence[ankle_idx*3 + 1]])

    # calculate vectors
    vec_hip_knee = knee - hip
    vec_knee_ankle = ankle - knee

    # calculate the cosine of the angle between the vectors
    cos_angle = np.dot(vec_hip_knee, vec_knee_ankle) /
(np.linalg.norm(vec_hip_knee) * np.linalg.norm(vec_knee_ankle))
    angle = np.arccos(cos_angle)
    return np.degrees(angle)


# calculate joint accelerations between two consecutive frames (acceleration is the
change in velocity)
def calculate_acceleration(velocity_t, velocity_t1):
    acceleration = velocity_t1 - velocity_t
    return acceleration

# calculate angular velocity (rate of change of joint angles between consecutive
frames)
def calculate_angular_velocity(joint_angle_t, joint_angle_t1):
    angular_velocity = joint_angle_t1 - joint_angle_t
    return angular_velocity

# calculate hip displacement between two frames
def calculate_hip_displacement(pose_sequence_t, pose_sequence_t1):
    left_hip_t = np.array([pose_sequence_t[23*3], pose_sequence_t[23*3 + 1]])
    right_hip_t = np.array([pose_sequence_t[24*3], pose_sequence_t[24*3 + 1]])
    left_hip_t1 = np.array([pose_sequence_t1[23*3], pose_sequence_t1[23*3 + 1]])
    right_hip_t1 = np.array([pose_sequence_t1[24*3], pose_sequence_t1[24*3 + 1]])

    # calculate hip centers
    hip_center_t = (left_hip_t + right_hip_t) / 2
    hip_center_t1 = (left_hip_t1 + right_hip_t1) / 2

    # displacement between frames
    hip_displacement = np.linalg.norm(hip_center_t1 - hip_center_t)
    return hip_displacement
```

```
# calculate temporal features (step length, velocity, joint angles, joint
acceleration, angular velocity, hip displacement etc.)
def calculate_features(pose_landmarks, previous_landmarks=None,
previous_velocities=None, previous_angles=None):
    frame_features = [0] * 10  # Ten features (step length, velocity x2, joint
angles x2, acceleration x2, angular velocity x2, hip displacement)

    # Step Length
    step_length = calculate_step_length(pose_landmarks)
    frame_features[0] = step_length

    # Joint Velocities
    if previous_landmarks is not None:
        left_ankle_velocity = calculate_velocity(previous_landmarks,
pose_landmarks, 27)
        right_ankle_velocity = calculate_velocity(previous_landmarks,
pose_landmarks, 28)
        frame_features[1] = left_ankle_velocity
        frame_features[2] = right_ankle_velocity

        # Joint Accelerations
        if previous_velocities is not None:
            left_ankle_acceleration =
calculate_acceleration(previous_velocities[0], left_ankle_velocity)
            right_ankle_acceleration =
calculate_acceleration(previous_velocities[1], right_ankle_velocity)
            frame_features[5] = left_ankle_acceleration
            frame_features[6] = right_ankle_acceleration

        # Hip Displacemen
        hip_displacement = calculate_hip_displacement(previous_landmarks,
pose_landmarks)
        frame_features[9] = hip_displacement
    else:
        frame_features[1] = frame_features[2] = 0  # Velocities
        frame_features[5] = frame_features[6] = 0  # Accelerations
        frame_features[9] = 0  # Hip Displacement

    # Joint Angles
    left_leg_angle = calculate_joint_angle(pose_landmarks, 23, 25, 27)
    right_leg_angle = calculate_joint_angle(pose_landmarks, 24, 26, 28)
    frame_features[3] = left_leg_angle
    frame_features[4] = right_leg_angle

    # Angular Velocities
    if previous_angles is not None:
        left_leg_angular_velocity = calculate_angular_velocity(previous_angles[0],
left_leg_angle)
        right_leg_angular_velocity = calculate_angular_velocity(previous_angles[1],
right_leg_angle)
        frame_features[7] = left_leg_angular_velocity
```

```
        frame_features[8] = right_leg_angular_velocity
    else:
        frame_features[7] = frame_features[8] = 0  # Angular velocities

    return frame_features


def load_and_preprocess_data(root_folder, net, output_layers):
    features = []
    labels = []
    valid_extensions = (".jpg", ".jpeg", ".png", ".bmp", ".tiff")

    for player_name in os.listdir(root_folder):
        player_folder = os.path.join(root_folder, player_name)
        if os.path.isdir(player_folder):
            previous_landmarks = None
            previous_velocities = None
            previous_angles = None

            for img_file in os.listdir(player_folder):
                if img_file.lower().endswith(valid_extensions):
                    img_path = os.path.join(player_folder, img_file)
                    img = cv2.imread(img_path)
                    if img is None or img.size == 0:
                        print(f"Failed to load image {img_path}.")
                        continue

                    # detect player using YOLO
                    boxes, _ = yolo_detect(net, img, output_layers)

                    for box in boxes:
                        x, y, w, h = box
                        if w > 0 and h > 0:
                            cropped_img = img[y:y+h, x:x+w]

                            # extract pose landmarks using MediaPipe Pose
                            pose_landmarks = extract_pose_landmarks(cropped_img)
                            if pose_landmarks is not None:
                                # extract temporal features
                                frame_features = calculate_features(
                                    pose_landmarks, previous_landmarks,
previous_velocities, previous_angles
                                )

                                # update previous values
                                if previous_landmarks is not None:
                                    previous_velocities = [
                                        frame_features[1],  # Left ankle velocity
                                        frame_features[2]   # Right ankle velocity
                                    ]
                                    previous_angles = [
```

```
                                    frame_features[3],  # Left leg angle
                                    frame_features[4]   # Right leg angle
                                ]
                            previous_landmarks = pose_landmarks

                            # append frame features and label
                            features.append(frame_features)
                            labels.append(player_name)

    if not features:
        print("No valid images or features found in dataset.")
        return np.array([]), np.array([]), None

    features = np.array(features, dtype=np.float32)
    encoder = LabelEncoder()
    encoded_labels = encoder.fit_transform(labels)

    return features, encoded_labels, encoder


# group features by player
def group_features_by_player(features, labels):
    grouped_data = {}

    for feature_set, label in zip(features, labels):
        if label not in grouped_data:
            grouped_data[label] = []
        grouped_data[label].append(feature_set)

    return grouped_data

# create sequences from grouped features
def create_sequences(grouped_data, sequence_length=15):
    sequences = []
    labels = []

    for label, features in grouped_data.items():
        # generate sequences of features for each player
        for i in range(len(features) - sequence_length + 1):
            sequence = features[i:i + sequence_length]
            sequences.append(sequence)
            labels.append(label)

    return np.array(sequences), np.array(labels)


# prepare the data for LSTM
def prepare_lstm_data(features, labels, sequence_length=15):
    # group the features by player
    grouped_data = group_features_by_player(features, labels)
```

```
    # create sequences of features
    sequences, labels = create_sequences(grouped_data, sequence_length)

    # reshape the sequences to the format (samples, timesteps, features)
    sequences = sequences.reshape((sequences.shape[0], sequence_length, -1))  # -1
to handle multiple features

    return sequences, labels

# build the GRU model
def build_gru_model(sequence_length, num_features, num_classes):

    model = Sequential()
    model.add(GRU(128, input_shape=(sequence_length, num_features),
return_sequences=False, kernel_regularizer=l2(0.07)))
    model.add(Dropout(0.4))
    model.add(Dense(32, activation='relu', kernel_regularizer=l2(0.07)))
    model.add(Dropout(0.5))
    model.add(Dense(num_classes, activation='softmax'))

    # compile the model
    model.compile(optimizer=Adam(), loss='sparse_categorical_crossentropy',
metrics=['accuracy'])

    return model


# plot accuracy and loss curves
def plot_training_history(history, smoothing_factor=0.7):
    # apply smoothing to accuracy and loss values
    acc = smooth_curve(history.history['accuracy'], smoothing_factor)
    val_acc = smooth_curve(history.history['val_accuracy'], smoothing_factor)
    loss = smooth_curve(history.history['loss'], smoothing_factor)
    val_loss = smooth_curve(history.history['val_loss'], smoothing_factor)

    # plot accuracy
    plt.figure(figsize=(12, 6))
    plt.subplot(1, 2, 1)
    plt.plot(acc, label='Training Accuracy')
    plt.plot(val_acc, label='Validation Accuracy')
    plt.title('Model Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()

    # plot loss
    plt.subplot(1, 2, 2)
    plt.plot(loss, label='Training Loss')
    plt.plot(val_loss, label='Validation Loss')
    plt.title('Model Loss')
    plt.xlabel('Epochs')
```

```
    plt.ylabel('Loss')
    plt.legend()

    plt.tight_layout()
    plt.show()


# confusion matrix
def plot_confusion_matrix(y_true, y_pred, classes):
    conf_matrix = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(8, 6))
    sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
xticklabels=classes, yticklabels=classes)
    plt.title('Confusion Matrix')
    plt.xlabel('Predicted Label')
    plt.ylabel('True Label')
    plt.show()

# classification report
def print_classification_report(y_true, y_pred, classes):
    report = classification_report(y_true, y_pred, target_names=classes)
    print('Classification Report:')
    print(report)



# main
output_frame_folder = '/Users/nadunsenarathne/Downloads/Documents/IIT/4th
Year/FYP/CricXpert/Hybrid_Spatio_Temporal_Model_For_Gait_Analysis/temporal_data'
net, output_layers = load_yolo()  # load YOLO model
features, labels, encoder = load_and_preprocess_data(output_frame_folder, net,
output_layers)  # load and preprocess data

sequence_length = 15  # define the sequence length (number of frames per sequence)

# prepare the LSTM data (features grouped into sequences)
lstm_sequences, lstm_labels = prepare_lstm_data(features, labels, sequence_length)

print(f"Extracted features shape: {features.shape}")
print(f"Number of labels: {len(labels)}")

# split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(lstm_sequences, lstm_labels,
test_size=0.2, random_state=42, shuffle=True)

# each sequence now has multiple features
print(f"Training data shape: {X_train.shape}")
print(f"Testing data shape: {X_test.shape}")

# initialize the model
```

```
num_features = X_train.shape[2]  # automatically detect the number of features
(step length, velocity, angles, etc.)
num_classes = len(encoder.classes_)  # number of players
gru_model = build_gru_model(sequence_length, num_features, num_classes)

# add early stopping to the training process
early_stopping = EarlyStopping(monitor='val_loss', patience=10,
restore_best_weights=True)


# train the model with the callback
history = gru_model.fit(
    X_train, y_train,
    epochs=50,
    batch_size=32,
    validation_data=(X_test, y_test),
    callbacks=[early_stopping]
)


# evaluate the model
train_loss, train_acc = gru_model.evaluate(X_train, y_train)
print(f"Train accuracy: {train_acc}")
test_loss, test_acc = gru_model.evaluate(X_test, y_test)
print(f"Test accuracy: {test_acc}")


# plot training history (accuracy and loss)
plot_training_history(history)

# make predictions
y_pred = gru_model.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)

# plot confusion matrix
plot_confusion_matrix(y_test, y_pred_classes, encoder.classes_)

# print classification report
print_classification_report(y_test, y_pred_classes, encoder.classes_)

# calculate overall accuracy from predictions
accuracy = np.mean(y_pred_classes == y_test)
print(f"Overall Test Accuracy: {accuracy}")


# save the trained model
gru_model.export('temporal_model')
print("GRU-based temporal model saved successfully!")
```

`app.py` – Main Flask Entry Point

```
from flask import Flask, request, jsonify, render_template
from inference.pipeline import predict_person, load_yolo
from inference.stat_generation import generate_sql_query, player_stats
import os
import joblib
import logging
from keras.applications.resnet50 import ResNet50
from tensorflow.keras.models import load_model
from langchain_openai import ChatOpenAI
from werkzeug.utils import secure_filename

app = Flask(__name__)

# configure logging
logging.basicConfig(level=logging.DEBUG)

UPLOAD_FOLDER = 'uploads'
os.makedirs(UPLOAD_FOLDER, exist_ok=True)
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER

print("Loading models...")
net, output_layers = load_yolo()
resnet_model = ResNet50(weights='imagenet', include_top=False, pooling='avg')
ensemble_model = joblib.load('saved_models/ResNet/ensemble_player_recognition.pkl')
label_encoder = joblib.load('saved_models/ResNet/ensemble_label_encoder.pkl')
face_recognition_model =
joblib.load('saved_models/Face_Recognition_Model/face_recognition_model.pkl')
face_label_encoder =
joblib.load('saved_models/Face_Recognition_Model/label_encoder.pkl')
temporal_model = load_model('saved_models/GRU/temporal_model')
llm = ChatOpenAI(model="gpt-4o", openai_api_key="####")

print("Models loaded successfully.")

@app.route('/')
def home():
    return render_template('index.html')
```

```
@app.route('/predict', methods=['POST'])
def predict():
    if 'file' not in request.files:
        return jsonify({"error": "No file part in the request"}), 400

    file = request.files['file']
    if file.filename == '':
        return jsonify({"error": "No file selected for uploading"}), 400

    if file:
        filename = secure_filename(file.filename)
        file_path =
os.path.join('/Users/nadunsenarathne/Downloads/Documents/IIT/4th
Year/FYP/CricXpert/cricxpert_webapp/uploads', filename)
        file.save(file_path)

        # determine if the file is a video or an image
        is_video = filename.lower().endswith(('.mp4', '.mov', '.avi'))
        logging.debug(f"File uploaded: {filename}, is_video: {is_video}")

        try:
            result = predict_person(
                video_path=file_path,
                resnet_model=resnet_model,
                ensemble_model=ensemble_model,
                label_encoder=label_encoder,
                face_recognition_model=face_recognition_model,
                face_label_encoder=face_label_encoder,
                temporal_model=temporal_model,
                is_video=is_video
            )
            logging.debug(f"Prediction result: {result}")
            return jsonify({"prediction": result})
        except Exception as e:
            logging.error(f"Error during prediction: {str(e)}")
            return jsonify({"error": str(e)}), 500

@app.route('/get_player_stats', methods=['POST'])
def get_player_stats():
    data = request.get_json()
    player = data.get('player')
    if player in player_stats:
        return jsonify({"stats": player_stats[player]})
    else:
        return jsonify({"error": "Player not found"}), 404

@app.route('/generate_stat', methods=['POST'])
def generate_stat():
    data = request.get_json()
    user_question = data.get("question", "")
```

```
    if not user_question:
        return jsonify({"error": "No question provided"})

    try:
        result = generate_sql_query(user_question, llm)
        return jsonify({"stat_result": result})
    except Exception as e:
        return jsonify({"error": str(e)})

if __name__ == "__main__":
    app.run(debug=True)
```

`settings.py` – Configuration credentials

```
LANGFUSE_SECRET_KEY = "####"
LANGFUSE_PUBLIC_KEY = "####"
LANGFUSE_HOST = "https://us.cloud.langfuse.com"
OPENAI_API_KEY = "####"
DB_HOST = "localhost"
DB_USER = "root"
DB_PASSWORD = "####"
DB_NAME = "cricxpert"
DB_PORT = 3306
```

`pipline.py` – Player recognition component

```
import os
import cv2
import numpy as np
import pytesseract
from mtcnn.mtcnn import MTCNN
from keras_facenet import FaceNet
from keras.applications.resnet50 import ResNet50, preprocess_input
from keras.preprocessing import image
from tensorflow.keras.models import load_model
from sklearn.ensemble import StackingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.preprocessing import LabelEncoder
import joblib
from collections import Counter
import mediapipe as mp
from tensorflow.keras.applications.resnet50 import ResNet50, preprocess_input
from tensorflow.keras.models import Model
from tensorflow.keras.layers import GlobalAveragePooling2D
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
```

```
from tensorflow.keras.layers import Dropout, Dense, GlobalAveragePooling2D
from tensorflow.keras.regularizers import l2
# path to tesseract
pytesseract.pytesseract.tesseract_cmd = ('/opt/homebrew/bin/tesseract')

# player database with both full reference name and jersey name
player_database = {
    "Virat_Kohli": {"name": "VIRAT", "number": "18"},
    "Arshdeep_Singh": {"name": "ARSHDEEP", "number": "2"},
    "Axar_Patel": {"name": "AXAR", "number": "20"},
    "Jasprit_Bumrah": {"name": "JASPRIT", "number": "93"},
    "Kuldeep_Yadav": {"name": "KULDEEP", "number": "23"},
    "Ravindra_Jadeja": {"name": "JADEJA", "number": "8"}
}

def load_yolo():
    path_to_cfg = "/Users/nadunsenarathne/Downloads/Documents/IIT/4th
Year/FYP/CricXpert/Hybrid_Spatio_Temporal_Model_For_Gait_Analysis/yolo/yolov3.cfg"
    path_to_weights = "/Users/nadunsenarathne/Downloads/Documents/IIT/4th
Year/FYP/CricXpert/Hybrid_Spatio_Temporal_Model_For_Gait_Analysis/yolo/yolov3.weigh
ts"
    net = cv2.dnn.readNet(path_to_weights, path_to_cfg)
    layers_names = net.getLayerNames()

    try:
        output_layers = [layers_names[i[0] - 1] for i in
net.getUnconnectedOutLayers()]
    except Exception:
        output_layers = [layers_names[i - 1] for i in
net.getUnconnectedOutLayers()]

    return net, output_layers

def yolo_detect(net, image, output_layers, confidence_threshold=0.3):
    height, width, _ = image.shape
    blob = cv2.dnn.blobFromImage(image, 0.00392, (416, 416), swapRB=True,
crop=False)
    net.setInput(blob)
    outputs = net.forward(output_layers)
    boxes = []
    confidences = []

    for output in outputs:
        for detection in output:
            scores = detection[5:]
            class_id = np.argmax(scores)
            confidence = scores[class_id]
            if confidence > confidence_threshold and class_id == 0:  # player class
                center_x = int(detection[0] * width)
                center_y = int(detection[1] * height)
                w = int(detection[2] * width)
```

```
                h = int(detection[3] * height)
                x = int(center_x - w / 2)
                y = int(center_y - h / 2)

                if x >= 0 and y >= 0 and (x + w) <= width and (y + h) <= height:
                    boxes.append([x, y, w, h])
                    confidences.append(float(confidence))

    # select the largest vertical box based on area if any boxes were detected
    if boxes:
        largest_box = max(boxes, key=lambda b: b[2] * b[3])  # b[2]*b[3] is the
area of the box (w*h)
        largest_confidence = confidences[boxes.index(largest_box)]
        return [largest_box], [largest_confidence]
    return [], []  # return empty lists if no boxes detected


def extract_features(images, model):
    processed_images = []

    for img in images:
        if img is not None and img.size > 0:  # to ensure the image is not empty
            resized_img = cv2.resize(img, (224, 224))
            processed_images.append(resized_img)

    if not processed_images:
        return np.array([])  # return empty array if no images to process

    images_array = np.array(processed_images)
    images_array = preprocess_input(images_array)
    features = model.predict(images_array)

    return features

    # features = features.reshape((features.shape[0], -1))


def process_frame_for_OCR_text_detection(image):

    # convert image to RGB for consistent display if originally in BGR
    if image.shape[2] == 3:  # assuming the image has 3 channels
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

    # original dimensions
    (H, W) = image.shape[:2]

    # set the new width and height to nearest multiple of 32 for EAST model
    newW = int(W / 32) * 32
    newH = int(H / 32) * 32

    # resize the image to fit model requirements
```

```
    image = cv2.resize(image, (newW, newH))

    # load the pre-trained EAST text detector model
    model_path = '/Users/nadunsenarathne/Downloads/Documents/IIT/4th
Year/FYP/CricXpert/Hybrid_Spatio_Temporal_Model_For_Gait_Analysis/saved_models/ResN
et/frozen_east_text_detection.pb'
    net = cv2.dnn.readNet(model_path)

    # prepare the image for the model
    blob = cv2.dnn.blobFromImage(image, 1.0, (newW, newH),
                                 (123.68, 116.78, 103.94), swapRB=True, crop=False)
    net.setInput(blob)

    # layer names for the output layers
    layerNames = [
        "feature_fusion/Conv_7/Sigmoid",
        "feature_fusion/concat_3"
    ]

    # forward pass of the model to get output
    (scores, geometry) = net.forward(layerNames)

    # decode the predictions
    (numRows, numCols) = scores.shape[2:4]
    rects = []
    confidences = []

    # loop over the number of rows
    for y in range(0, numRows):
        scoresData = scores[0, 0, y]
        xData0 = geometry[0, 0, y]
        xData1 = geometry[0, 1, y]
        xData2 = geometry[0, 2, y]
        xData3 = geometry[0, 3, y]
        anglesData = geometry[0, 4, y]

        # loop over the number of columns
        for x in range(0, numCols):
            if scoresData[x] < 0.5:
                continue

            offsetX = x * 4.0
            offsetY = y * 4.0
            angle = anglesData[x]
            cos = np.cos(angle)
            sin = np.sin(angle)

            h = xData0[x] + xData2[x]
            w = xData1[x] + xData3[x]

            endX = int(offsetX + (cos * xData1[x]) + (sin * xData2[x]))
```

```
            endY = int(offsetY - (sin * xData1[x]) + (cos * xData2[x]))
            startX = int(endX - w)
            startY = int(endY - h)

            rects.append((startX, startY, endX, endY))
            confidences.append(scoresData[x])

    # apply non-maxima suppression to avoid overlaps
    indices = cv2.dnn.NMSBoxes(rects, confidences, 0.5, 0.4)

    if len(indices) > 0:
        indices = indices.flatten()  # ensuring flattening is possible

    cropped_img = None

    if len(indices) > 0:
        min_x = max(0, min([rects[i][0] for i in indices]) - 20)
        min_y = max(0, min([rects[i][1] for i in indices]) - 20)
        max_x = min(W, max([rects[i][2] for i in indices]) + 20)
        max_y = min(H, max([rects[i][3] for i in indices]) + 20)

        cropped_img = image[min_y:max_y, min_x:max_x]
    else:
        print('No Detections Found')

    # Initialize 'text' to an empty string
    text = ''

    # proceed with text recognition on the cropped image
    if cropped_img is not None:
        # convert cropped image from RGB to BGR for OpenCV operations
        cropped_img_bgr = cv2.cvtColor(cropped_img, cv2.COLOR_RGB2BGR)

        # convert to grayscale
        gray = cv2.cvtColor(cropped_img_bgr, cv2.COLOR_BGR2GRAY)

        # use median blur to remove noise
        blur = cv2.medianBlur(gray, 5)

        # Configure parameters for Tesseract
        custom_config = r'--oem 3 --psm 11'
        text = pytesseract.image_to_string(blur, config=custom_config)

        print("Detected text:", text)
    else:
        print("No area was cropped for text recognition.")

    return text


# load the FaceNet model for embeddings
```

```
embedder = FaceNet()

# initialize MediaPipe Pose Estimation
mp_pose = mp.solutions.pose
pose = mp_pose.Pose()

# function to get embeddings using FaceNet
def get_embedding(face_img):
    face_img = face_img.astype('float32')
    face_img = np.expand_dims(face_img, axis=0)
    return embedder.embeddings(face_img)[0]

# function to extract pose landmarks from the cropped image
def extract_pose_landmarks(cropped_img):
    # convert the image to RGB
    image_rgb = cv2.cvtColor(cropped_img, cv2.COLOR_BGR2RGB)

    # perform pose detection
    results = pose.process(image_rgb)

    # if landmarks are detected
    if results.pose_landmarks:
        landmarks = []
        for landmark in results.pose_landmarks.landmark:
            landmarks.append([landmark.x, landmark.y, landmark.z])
        return np.array(landmarks).flatten()
    return None

# calculate step length between left and right ankles
def calculate_step_length(pose_sequence):
    left_ankle_x, left_ankle_y = pose_sequence[27*3], pose_sequence[27*3 + 1]  #
left ankle
    right_ankle_x, right_ankle_y = pose_sequence[28*3], pose_sequence[28*3 + 1]  #
right ankle
    step_length = np.linalg.norm(np.array([right_ankle_x, right_ankle_y]) -
np.array([left_ankle_x, left_ankle_y]))
    return step_length

# calculate joint velocities between two consecutive frames
def calculate_velocity(pose_sequence_t, pose_sequence_t1, joint_index):
    joint_t = np.array([pose_sequence_t[joint_index*3],
pose_sequence_t[joint_index*3 + 1]])
    joint_t1 = np.array([pose_sequence_t1[joint_index*3],
pose_sequence_t1[joint_index*3 + 1]])
    velocity = np.linalg.norm(joint_t1 - joint_t)
    return velocity

# calculate joint angles (hip-knee-ankle)
def calculate_joint_angle(pose_sequence, hip_idx, knee_idx, ankle_idx):
    hip = np.array([pose_sequence[hip_idx*3], pose_sequence[hip_idx*3 + 1]])
    knee = np.array([pose_sequence[knee_idx*3], pose_sequence[knee_idx*3 + 1]])
```

```
    ankle = np.array([pose_sequence[ankle_idx*3], pose_sequence[ankle_idx*3 + 1]])

    # calculate vectors
    vec_hip_knee = knee - hip
    vec_knee_ankle = ankle - knee

    # calculate the cosine of the angle between the vectors
    cos_angle = np.dot(vec_hip_knee, vec_knee_ankle) /
(np.linalg.norm(vec_hip_knee) * np.linalg.norm(vec_knee_ankle))
    angle = np.arccos(cos_angle)
    return np.degrees(angle)

# calculate joint accelerations between two consecutive frames (acceleration is the
change in velocity)
def calculate_acceleration(velocity_t, velocity_t1):
    acceleration = velocity_t1 - velocity_t
    return acceleration

# calculate angular velocity (rate of change of joint angles between consecutive
frames)
def calculate_angular_velocity(joint_angle_t, joint_angle_t1):
    angular_velocity = joint_angle_t1 - joint_angle_t
    return angular_velocity

# calculate hip displacement between two frames
def calculate_hip_displacement(pose_sequence_t, pose_sequence_t1):
    left_hip_t = np.array([pose_sequence_t[23*3], pose_sequence_t[23*3 + 1]])
    right_hip_t = np.array([pose_sequence_t[24*3], pose_sequence_t[24*3 + 1]])
    left_hip_t1 = np.array([pose_sequence_t1[23*3], pose_sequence_t1[23*3 + 1]])
    right_hip_t1 = np.array([pose_sequence_t1[24*3], pose_sequence_t1[24*3 + 1]])

    # calculate hip centers
    hip_center_t = (left_hip_t + right_hip_t) / 2
    hip_center_t1 = (left_hip_t1 + right_hip_t1) / 2

    # displacement between frames
    hip_displacement = np.linalg.norm(hip_center_t1 - hip_center_t)
    return hip_displacement


# temporal feature extraction (step length, joint velocities, joint angles, etc.)
def calculate_features(pose_landmarks, previous_landmarks=None,
previous_velocities=None, previous_angles=None):
    frame_features = [0] * 10  # 10 features (step length, velocity x2, joint
angles x2, acceleration x2, angular velocity x2, hip displacement)

    # step Length
    step_length = calculate_step_length(pose_landmarks)
    frame_features[0] = step_length

    # joint Velocities
```

```
    if previous_landmarks is not None:
        left_ankle_velocity = calculate_velocity(previous_landmarks,
pose_landmarks, 27)
        right_ankle_velocity = calculate_velocity(previous_landmarks,
pose_landmarks, 28)
        frame_features[1] = left_ankle_velocity
        frame_features[2] = right_ankle_velocity

        # joint Accelerations
        if previous_velocities is not None:
            left_ankle_acceleration =
calculate_acceleration(previous_velocities[0], left_ankle_velocity)
            right_ankle_acceleration =
calculate_acceleration(previous_velocities[1], right_ankle_velocity)
            frame_features[5] = left_ankle_acceleration
            frame_features[6] = right_ankle_acceleration

        # hip Displacement
        hip_displacement = calculate_hip_displacement(previous_landmarks,
pose_landmarks)
        frame_features[9] = hip_displacement
    else:
        frame_features[1] = frame_features[2] = 0  # velocities
        frame_features[5] = frame_features[6] = 0  # accelerations
        frame_features[9] = 0  # hip displacement

    # joint angles
    left_leg_angle = calculate_joint_angle(pose_landmarks, 23, 25, 27)
    right_leg_angle = calculate_joint_angle(pose_landmarks, 24, 26, 28)
    frame_features[3] = left_leg_angle
    frame_features[4] = right_leg_angle

    # angular velocities
    if previous_angles is not None:
        left_leg_angular_velocity = calculate_angular_velocity(previous_angles[0],
left_leg_angle)
        right_leg_angular_velocity = calculate_angular_velocity(previous_angles[1],
right_leg_angle)
        frame_features[7] = left_leg_angular_velocity
        frame_features[8] = right_leg_angular_velocity
    else:
        frame_features[7] = frame_features[8] = 0  # angular velocities

    return frame_features

# extract temporal features from frames
def extract_temporal_features(frames):
    temporal_features = []
    previous_landmarks = None
    previous_velocities = None
    previous_angles = None
```

```
    for frame in frames:
        pose_landmarks = extract_pose_landmarks(frame)
        if pose_landmarks is not None:
            frame_features = calculate_features(pose_landmarks, previous_landmarks,
previous_velocities, previous_angles)
            temporal_features.append(frame_features)
            previous_landmarks = pose_landmarks  # store the current landmarks for
the next frame

    return np.array(temporal_features)

def clean_detected_text(text):
    # remove non-alphanumeric characters and extra spaces
    text = ''.join(char for char in text if char.isalnum() or char.isspace())
    text = text.strip().upper()  # convert to uppercase for case-insensitive
matching

    return text

def extract_frames_for_prediction(video_path, net, output_layers, num_frames=10):
    cap = cv2.VideoCapture(video_path)
    total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
    interval = total_frames // num_frames if total_frames > num_frames else 1
    frame_ids = [int(interval * i) for i in range(num_frames)]
    frames = []
    detected_texts = []

    frame_count = 0
    while frame_count < total_frames:
        ret, frame = cap.read()
        if not ret:
            break

        if frame_count in frame_ids:
            # apply YOLO detection to each frame
            boxes, confidences = yolo_detect(net, frame, output_layers)
            if boxes:  # check if there is at least one detection
                largest_box = boxes[0]  # the largest box returned by yolo_detect
                x, y, w, h = largest_box  # unpack the largest box
                cropped_frame = frame[max(0, y):max(0, y + h), max(0, x):max(0, x +
w)]
                resized_frame = cv2.resize(cropped_frame, (224, 224))  # resize
frame

                # apply CLAHE for better contrast
                lab = cv2.cvtColor(resized_frame, cv2.COLOR_BGR2LAB)
                l, a, b = cv2.split(lab)
                clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))
                l = clahe.apply(l)
                processed_img = cv2.merge([l, a, b])
```

```
                processed_img = cv2.cvtColor(processed_img, cv2.COLOR_LAB2BGR)

                frames.append(processed_img)
                detected_text = process_frame_for_OCR_text_detection(cropped_frame)
                detected_texts.append(clean_detected_text(detected_text))  # clean
and append OCR text
            else:
                # fallback for frames without valid detections
                resized_frame = cv2.resize(frame, (224, 224))  # resize original
frame
                frames.append(resized_frame)  # include the resized frame
                print(f"No valid detections at frame {frame_count}.")
        frame_count += 1

    cap.release()
    return frames, detected_texts

def extract_image_for_prediction(image_path, net, output_layers):
    frame = cv2.imread(image_path)
    frames = []
    detected_texts = []

    # apply YOLO detection to the image
    boxes, confidences = yolo_detect(net, frame, output_layers)
    if boxes:  # check if there is at least one detection
        largest_box = boxes[0]  #the largest box returned by yolo_detect
        x, y, w, h = largest_box  # unpack the largest box
        cropped_frame = frame[max(0, y):max(0, y + h), max(0, x):max(0, x + w)]
        resized_frame = cv2.resize(cropped_frame, (224, 224))  # resize frame

        # apply CLAHE for better contrast
        lab = cv2.cvtColor(resized_frame, cv2.COLOR_BGR2LAB)
        l, a, b = cv2.split(lab)
        clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))
        l = clahe.apply(l)
        processed_img = cv2.merge([l, a, b])
        processed_img = cv2.cvtColor(processed_img, cv2.COLOR_LAB2BGR)

        frames.append(processed_img)
        detected_text = process_frame_for_OCR_text_detection(cropped_frame)
        detected_texts.append(clean_detected_text(detected_text))  # clean and
append OCR text
    else:
        # fallback for images without valid detections
        resized_frame = cv2.resize(frame, (224, 224))  # resize original frame
        frames.append(resized_frame)  # include the resized frame
        print("No valid detections in the image.")

    return frames, detected_texts

def contains_substring(player_name, detected_text, min_length=4):
```

```
    player_name = player_name.upper()  # ensure case-insensitive matching
    detected_text = detected_text.upper()

    # sliding window approach: check all substrings of `player_name`
    for i in range(len(player_name) - min_length + 1):
        substring = player_name[i:i + min_length]
        if substring in detected_text:
            return True
    return False




# predict player using the entire pipeline (OCR, Face, Spatial, and Temporal
models)
def predict_person(video_path, resnet_model, ensemble_model, label_encoder,
face_recognition_model, face_label_encoder, temporal_model, is_video=True):

    # load YOLO net
    net, output_layers = load_yolo()
    # load the pre-trained ResNet model
    resnet_model = ResNet50(weights='imagenet', include_top=False, pooling='avg')
    # load trained model and label encoder for the spatial model
    ensemble_model =
joblib.load('/Users/nadunsenarathne/Downloads/Documents/IIT/4th
Year/FYP/CricXpert/Hybrid_Spatio_Temporal_Model_For_Gait_Analysis/saved_models/ResN
et/ResNet_SVM_KNN/ensemble_player_recognition.pkl')
    label_encoder = joblib.load('/Users/nadunsenarathne/Downloads/Documents/IIT/4th
Year/FYP/CricXpert/Hybrid_Spatio_Temporal_Model_For_Gait_Analysis/saved_models/ResN
et/ResNet_SVM_KNN/ensemble_label_encoder.pkl')
    # load face recognition model and label encoder
    face_recognition_model =
joblib.load('/Users/nadunsenarathne/Downloads/Documents/IIT/4th
Year/FYP/CricXpert/Face_Recognition_Model/trained_model/face_recognition_model.pkl'
)
    face_label_encoder =
joblib.load('/Users/nadunsenarathne/Downloads/Documents/IIT/4th
Year/FYP/CricXpert/Face_Recognition_Model/trained_model/label_encoder.pkl')
    # load the GRU-based temporal model
    temporal_model = load_model('/Users/nadunsenarathne/Downloads/Documents/IIT/4th
Year/FYP/CricXpert/Hybrid_Spatio_Temporal_Model_For_Gait_Analysis/saved_models/GRU/
temporal_model')


    if is_video:
        frames, detected_texts = extract_frames_for_prediction(video_path, net,
output_layers)
    else:
        frames, detected_texts = extract_image_for_prediction(video_path, net,
output_layers)
```

```
    if not frames:
        print("No frames to analyze or no valid detections.")
        return

    # initialize a list to store text-based matches
    text_matches = []

    # Step 1: OCR System (Try to detect jersey name/number)
    for i, detected_text in enumerate(detected_texts):
        detected_text_upper = detected_text.upper().strip()
        print(f"\nProcessing frame {i}")
        print(f"Detected text: {detected_text_upper}")
        matched_player = None
        for player, info in player_database.items():
            name_match = contains_substring(info['name'], detected_text_upper,
min_length=4)
            number_match = info['number'] in detected_text_upper
            if name_match and number_match:
                matched_player = player
                print(f"Detected text matches player: {player}")
                text_matches.append(matched_player)
                break  # stop checking after finding a match
            elif name_match:
                matched_player = player
                print(f"Detected text matches player: {player}")
                text_matches.append(matched_player)
                break  # stop checking after finding a match

    # check if any text matches were found from OCR
    if text_matches:
        most_common_player = Counter(text_matches).most_common(1)[0][0]
        print(f"\nPredicted player based on text detection: {most_common_player}")
        return most_common_player

    print("No player detected from OCR, moving to face recognition...")


    # Step 2: Face Recognition (Use MTCNN and FaceNet)
    cap = cv2.VideoCapture(video_path)
    total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
    num_frames_to_extract = 10
    interval = total_frames // num_frames_to_extract if total_frames >=
num_frames_to_extract else 1
    frame_ids = [interval * i for i in range(num_frames_to_extract)]
    frames = []

    frame_count = 0
    while cap.isOpened():
        ret, frame = cap.read()
        if not ret:
```

```
            break

        if frame_count in frame_ids:
            # append raw frames without YOLO cropping
            frames.append(frame)

        frame_count += 1
    cap.release()

    # check if frames were extracted
    if not frames:
        print("No frames were extracted from the video.")
        return "Unknown"


    processed_faces = []
    face_predictions = []
    detector = MTCNN()

    for i, frame in enumerate(frames):
        rgb_img = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        results = detector.detect_faces(rgb_img)

        if results:
            for result in results:
                x, y, w, h = result['box']
                x, y = max(0, x), max(0, y)  # ensure coordinates are within the
image
                face = rgb_img[y:y+h, x:x+w]
                face = cv2.resize(face, (160, 160))

                # generate embedding for the detected face
                embedding = get_embedding(face)
                embedding = np.expand_dims(embedding, axis=0)

                # predict the identity of the face
                ypred = face_recognition_model.predict(embedding)
                proba = face_recognition_model.predict_proba(embedding).max()
                if proba > 0.5:  # confidence threshold
                    final_name = face_label_encoder.inverse_transform(ypred)[0]
                    face_predictions.append(final_name)
                    break  # stop processing further faces in the frame
                else:
                    face_predictions.append("Unknown")
        else:
            print(f"No face detected in frame {i}.")

    # check if any face matches were found
    if face_predictions:
        filtered_face_predictions = [pred for pred in face_predictions if pred !=
"Unknown"]
```

```
        if filtered_face_predictions:
            most_common_face_prediction =
Counter(filtered_face_predictions).most_common(1)[0][0]
            print(f"\nPredicted player based on face recognition:
{most_common_face_prediction}")
            return most_common_face_prediction

    print("No player detected from face recognition, moving to spatial model...")

    # Step 3: Spatial Model (Use ResNet for feature extraction and ensemble model
for prediction)
    features = extract_features(frames, resnet_model)
    predictions = []
    for i, feature in enumerate(features):
        probabilities = ensemble_model.predict_proba([feature])[0]
        predicted_class_index = np.argmax(probabilities)
        predicted_player =
label_encoder.inverse_transform([predicted_class_index])[0]
        confidence = probabilities[predicted_class_index]
        print(f"Frame {i}: Detected person: {predicted_player} with confidence:
{confidence:.2f}")

        if confidence > 0.1:
            predictions.append(predicted_player)
        else:
            predictions.append("Unknown")

    # check if any spatial model predictions were made
    if predictions:
        filtered_predictions = [pred for pred in predictions if pred != "Unknown"]
        if filtered_predictions:
            most_common_prediction =
Counter(filtered_predictions).most_common(1)[0][0]
            print(f"\nPredicted player based on spatial model:
{most_common_prediction}")
            return most_common_prediction

    print("No player detected from spatial model, moving to temporal model...")


    # Step 4: Temporal Model (Use pose landmarks and GRU-based model for gait
analysis)
    temporal_features = extract_temporal_features(frames)
    sequence_length = 15

    # create sequences from temporal features
    temporal_sequences = []
    for i in range(len(temporal_features) - sequence_length + 1):
        temporal_sequences.append(temporal_features[i:i + sequence_length])
    temporal_sequences = np.array(temporal_sequences)
```

```
    # check if temporal_sequences is empty
    if len(temporal_sequences) == 0:
        print("Not enough temporal sequences for analysis. Returning 'Unknown'.")
        return "Unknown"

    # predict using the temporal model
    temporal_predictions = temporal_model.predict(temporal_sequences)

    # aggregate predictions to find the most common player
    predicted_classes = np.argmax(temporal_predictions, axis=1)
    class_counts = np.bincount(predicted_classes)

    # if no clear prediction can be made, return 'Unknown'
    if len(class_counts) == 0 or class_counts.max() == 0:
        print("No clear player prediction from temporal model. Returning
'Unknown'.")
        return "Unknown"

    predicted_player = np.argmax(class_counts)
    most_common_temporal_prediction =
label_encoder.inverse_transform([predicted_player])[0]
    print(f"\nPredicted player based on temporal model:
{most_common_temporal_prediction}")
    return most_common_temporal_prediction
```

`stat_generation.py` – Stat generation component

```
from langchain.prompts import ChatPromptTemplate, HumanMessagePromptTemplate
from langchain.output_parsers import PydanticOutputParser
from langchain.output_parsers import OutputFixingParser
from langchain.schema import OutputParserException
from langchain_openai import ChatOpenAI
from langfuse.callback import CallbackHandler
import mysql.connector
import inference.settings as settings
from pydantic import BaseModel, Field
from textwrap import dedent

# configuration for handling callbacks and LLM setup
langfuse_handler = CallbackHandler(
    secret_key="#### ",
    public_key="####",
    host="https://us.cloud.langfuse.com",
)

def parse_response(output, parser, llm=ChatOpenAI(model="gpt-4o",
openai_api_key="####")):
    try:
```

```
        parsed = parser.parse(output)
    except OutputParserException:
        try:
            new_parser = OutputFixingParser.from_llm(parser=parser, llm=llm)
            parsed = new_parser.parse(output.content)
            print("Fixed parsing errors.")
        except Exception as e:
            print("Failed to fix parsing errors.")
            parsed = None
    return parsed


def llm_invoke(llm, messages):
    return llm.invoke(messages.to_messages(), {"callbacks": [langfuse_handler]})

class SQLQueryModel(BaseModel):
    sql_query: str = Field(
        description="SQL query generated based on the user's natural language
input."
    )

def generate_sql_query(user_input, llm):
    """
    This function generates a SQL query based on the user's input using GPT-4 and
the relationships in the cricket database.
    """
    # define the parser for SQL output
    parser = PydanticOutputParser(pydantic_object=SQLQueryModel)
    format_instructions = parser.get_format_instructions()

    prompt_template = dedent(
        """
        You are an AI assistant specialized in generating SQL queries from multiple
conditions and tables from natural language.

        The database consists of the following tables and relationships:
        - `batting`: Contains information about players' batting performance.
          - Columns: `Runs` (runs scored), `Mins` (minutes batted), `BF` (balls
faced), `4s`, `6s`, `SR` (strike rate), `Pos` (batting position), `Dismissal` (how
the batsman got out), `Inns` (1 for 1st innings, 2 for 2nd innings), `Opposition`,
`Ground`, `Start Date`, `t20_match_no`, `player_id`, `ground_id`,
`opposition_country_id`, `home_away`, `match_result_id`, `in_finals_id`.
          - Foreign Keys: `player_id` references `player(player_id)`, `ground_id`
references `ground(ground_id)`,
            `opposition_country_id` references `country(country_id)`,
`match_result_id` references `match_result(match_result_id)`,
            `in_finals_id` references `in_finals(match_stage_id)`, `t20_match_no`
to identify specific matches.
        - `bowling`: Contains information about players' bowling performance.
          - Columns: `Overs` (number of overs bowled), `Mdns` (number of maidens
bowled), `Runs` (runs conceded), `Wkts` (wickets taken), `Econ` (economy rate),
```

```
`Pos` (bowling position), `Inns` (1 for 1st innings, 2 for 2nd innings),
`t20_match_no`, `player_id`, `match_result_id`, `home_away`, `in_finals_id`,
`start_date`.
          - Foreign Keys: `player_id` references `player(player_id)`, `ground_id`
references `ground(ground_id)`,
            `opposition_country_id` references `country(country_id)`,
`match_result_id` references `match_result(match_result_id)`,
            `in_finals_id` references `in_finals(match_stage_id)`, `t20_match_no`
to identify specific matches.
        - `fielding`: Contains information about players' fielding performance.
          - Columns: `Dis` (number of dismissals), `Ct` (catches taken), `St`
(stumpings), `Ct Wk` (catches as wicketkeeper), `Ct Fi` (catches as a fielder),
`Inns` (1 for 1st innings, 2 for 2nd innings), `t20_match_no`, `player_id`,
`match_result_id`, `home_away`, `start_date`.
          - Foreign Keys: `player_id` references `player(player_id)`, `ground_id`
references `ground(ground_id)`,
            `opposition_country_id` references `country(country_id)`,
`match_result_id` references `match_result(match_result_id)`,
            `in_finals_id` references `in_finals(match_stage_id)`, `t20_match_no`
to identify specific matches.
        - `player`: Stores details about players (`player_id` as primary key).
          - Columns: `player_id`, `player_name`, `country_id` (references
`country(country_id)` to identify which country the player belongs to).
        - `country`: Stores information about countries (`country_id` as primary
key).
          - Columns: `country_id`, `country_name` (e.g., 'India', 'Australia').
        - `ground`: Stores information about match grounds (`ground_id` as primary
key).
        - `match_result`: Stores match results (`match_result_id` as primary key).
          - Columns: `match_result_id`, `match_result` (if `match_result_id`is 1
then 'match_result' is 'won match', `match_result_id`is 2 then 'match_result' is
'lost match', `match_result_id`is 3 then 'match_result' is 'tied match',
`match_result_id`is 4 then 'match_result' is 'no result').
        - `in_finals`: Stores information on whether a match was a final
(`match_stage_id` as primary key).

        When generating the SQL query, consider the following:
        - The query is properly formatted and valid for MySQL.
        - The response should be in JSON format with a single key named
`sql_query`.
        - Example JSON response: {{ "sql_query": "SELECT * FROM ...;" }}
        - Use `CAST()` for type conversions instead of `::`.
        - Avoid any syntax that is specific to PostgreSQL or other SQL variants.
        - Handle multiple conditions, such as filtering by runs, wickets, and
specific years.
        - Use Common Table Expressions (CTEs) or subqueries if needed to calculate
aggregates or percentages.
        - Use `DISTINCT` when counting or selecting matches to avoid duplication.
        - Use aggregate functions such as `COUNT()` to calculate total counts.
        - Make sure to use `GROUP BY` where required when using aggregate functions
in select statements.
```

```
        - Use `NOT IN` subqueries to filter out matches involving certain players
when requested.
        - Return only the columns that are necessary to answer the user's question.
        - If you encounter a player name or any entity that is misspelled, use
fuzzy matching or approximate search techniques to identify the closest match.
        - To determine match outcomes involving a specific player (e.g., Arshdeep
Singh), join `player` and `match_result` tables to verify the country and match
outcome respectively.
        - Use `t20_match_no` as the unique identifier for each match, linking only
batting, bowling, and fielding.
        - Focus on matches where a specific player meets the conditions given in
the user's question.
        - Use a step-by-step approach with Common Table Expressions (CTEs) to break
down the query into logical steps:
            1. First, identify matches where specific conditions are met (e.g., a
player took a certain number of wickets).
            2. Then, determine how many of these matches meet another condition
(e.g., matches were won by India).
            3. Finally, calculate any required performance metrics or counts based
on these results.
        - Use the `home_away` column from `batting`, `bowling`, or `fielding` to
determine if the match was played at home, away, or neutral.
        - When determining the number of matches played, consider using joins
across multiple tables (`batting`, `bowling`, etc.) and apply `UNION` if needed to
avoid double counting.
        - Ensure that the conditions for finals, such as checking `in_finals_id`,
are correctly applied to identify final matches.
        - Include additional details like `start_date` and `opposition_country_id`
to provide more context for match results, such as the opponent team and match
date.
        - When calculating performance metrics (e.g., economy rate, win percentage
when conditions are met), use CTEs or subqueries to first identify relevant matches
and then apply aggregate functions to derive the final output.



        Format your response as follows:
        {format_instructions}

        Given the user's question, generate an appropriate SQL query that joins the
necessary tables based on the foreign key relationships.

        User Input:
        {user_input}
        """
    )

    prompt = ChatPromptTemplate(
        messages=[HumanMessagePromptTemplate.from_template(prompt_template)],
        input_variables=["user_input", "format_instructions"],
    )
```

```
    # format the prompt with user input and format instructions
    _input = prompt.format_prompt(
        user_input=user_input, format_instructions=format_instructions
    )

    # invoke the LLM to generate the SQL query
    output = llm_invoke(llm, _input)

    print("Raw Output from LLM:", output.content)

    # parse the output to extract the SQL query
    parsed = parse_response(output.content, parser)
    if parsed is None:
        raise ValueError("Failed to parse the response. Please check the output.")

    sql_query = parsed.sql_query

    # initialize result to None to ensure it is defined
    result = None

    # execute the SQL query against MySQL database
    try:
        connection = mysql.connector.connect(
            host=settings.DB_HOST,
            user=settings.DB_USER,
            password=settings.DB_PASSWORD,
            database=settings.DB_NAME,
            port=settings.DB_PORT
        )
        cursor = connection.cursor()
        cursor.execute(sql_query)
        result = cursor.fetchall()
        for row in result:
            print(row)
    except mysql.connector.Error as err:
        print(f"Error: {err}")
    finally:
        if connection.is_connected():
            cursor.close()
            connection.close()

    return result
```