A PROJECT REPORT

on

"CHESS BOT DETECTOR"

Submitted to

KIIT Deemed to be University

In Partial Fulfilment of the Requirement for the Award of

BACHELOR'S DEGREE IN
INFORMATION TECHNOLOGY

BY

ASHUTOSH DASH	2206248
AAYUSHI DUTTA	22052436
KANISHK JAKHMOLA	2206265
ANKIT RANJAN	2206241

UNDER THE GUIDANCE OF
DR. AJIT KUMAR PASAYAT

SCHOOL OF COMPUTER ENGINEERING

KALINGA INSTITUTE OF INDUSTRIAL TECHNOLOGY

BHUBANESWAR, ODISHA - 751024

April 2025

A PROJECT REPORT
on

“CHESS BOT DETECT”

Submitted to
KIIT Deemed to be University

In Partial Fulfilment of the Requirement for the Award of

BACHELOR’S DEGREE IN INFORMATION TECHNOLOGY

BY

ASHUTOSH DASH	2206248
AAYUSHI DUTTA	22052436
KANISHK JAKHMOLA	2206265
ANKIT RANJAN	2206241

UNDER THE GUIDANCE OF
DR. AJIT KUMAR PASAYAT

SCHOOL OF COMPUTER ENGINEERING
KALINGA INSTITUTE OF INDUSTRIAL TECHNOLOGY
BHUBANESWAE, ODISHA -751024

April 2025

KIIT Deemed to be University

School of Computer Engineering
Bhubaneswar, ODISHA 751024

CERTIFICATE

This is certify that the project entitled

"CHESS BOT DETECT "

submitted by

ASHUTOSH DASH	2206248
AAYUSHI DUTTA	22052436
KANISHK JAKHMOLA	2206265
ANKIT RANJAN	2206241

is a record of bonafide work carried out by them, in the partial fulfilment of the requirement for the award of Degree of Bachelor of Engineering (Computer Science & Engineering OR Information Technology) at KIIT Deemed to be university, Bhubaneswar. This work is done during year 2024-2025, under our guidance.

Date: 08 / 04 / 2025

Dr. Ajit Kumar Pasayat
Project Guide

Acknowledgements

We are profoundly grateful to **DR. AJIT KUMAR PASAYAT** of **Affiliation** for his expert guidance and continuous encouragement throughout to see that this project rights its target since its commencement to its completion. His insights and feedback played a crucial role in shaping our work and pushing us toward excellence.

Right from the inception of this project to its successful completion, his unwavering motivation and deep knowledge have inspired us to stay focused and committed to achieving our objectives. His mentorship has not only enhanced the quality of my project but has also significantly contributed to our learning experience.

We truly appreciate the time and effort he has invested in us, and er are indebted to him for making this project a valuable and enriching endeavour.

ASHUTOSH DASH
AAYUSHI DUTTA
KANISHK JAKHMOLA
ANKIT RANJAN

ABSTRACT

With the rapid rise of online chess platforms like Chess.com and Lichess, millions of players now engage in competitive gameplay from around the world. While this digital transformation has made chess more accessible than ever, it has also given rise to a significant issue — the misuse of chess engines and bots. These automated programs can play with superhuman precision, allowing players to gain unfair advantages and undermining the spirit of fair competition. Existing detection methods typically rely on long-term behavior analysis and user reports, which are often not suitable for real-time or single-match evaluations. To address this challenge, we developed a machine learning-based system capable of analyzing match-level statistics to determine the likelihood of bot involvement. The methodology involved collecting real-game data from online platforms, preprocessing it through cleaning, normalization, and feature selection, and evaluating various classification models including Logistic Regression, Decision Tree, and Random Forest. The system was designed to prioritize speed, accessibility, and ease of use, making it ideal for quick and reliable detection outside of platform-level infrastructure.

To address this problem, our project introduces a lightweight, match-level bot detection system powered by machine learning. The model analyzes specific gameplay features such as player accuracy, the number of blunders and brilliant moves, consistency in move timing, total number of moves, and the final result to predict whether a bot was involved. Achieving an accuracy of approximately 87%, the model has been integrated into a user-friendly web application developed using Flask, which allows users to enter game statistics and instantly receive bot/human predictions. The platform is deployed on Render, making it freely accessible and efficient. In addition to solving a practical issue in digital chess, this project demonstrates the fusion of AI, data science, and web technologies to promote integrity, fairness, and ethical standards in online gaming environments.

Keywords - Chess Bot Detection, Machine Learning, Online Gaming Fairness, Game-Level Analysis, Web Application, Feature Engineering

Contents

List of Figures

Chapter 1

Introduction

The game of chess, often regarded as the ultimate test of human intellect, has witnessed a dramatic shift in recent years due to the digital revolution. Online chess platforms like Chess.com and Lichess have made it easier than ever for players across the globe to engage in competitive matches. With millions of active players, the digital chess ecosystem has evolved into a vibrant and competitive community.

However, this rise in online engagement has also brought a major challenge to the forefront — the use of bots and chess engines during gameplay. Bots are automated programs or scripts capable of playing chess with superhuman accuracy. Players using such tools can gain an unfair advantage, undermining the very foundation of fair competition and sportsmanship. Detecting such foul play manually is both time-consuming and inefficient. Traditional detection systems on popular platforms rely on long-term behavior tracking, statistical anomalies, and user reports. These methods often fail to provide real-time or one-match-based detection, which is essential for casual users or small tournament organizers who want immediate analysis.

Our project addresses this issue by proposing a **lightweight, match-level bot detection system**. We have developed a **machine learning model** that analyzes specific game features — including player accuracy, number of brilliant or blunder moves, consistency in move time, number of moves, and game result — to detect potential bot involvement.

To make this system widely accessible, we have also developed a **user-friendly website** where users can enter their match statistics and get instant predictions on whether the gameplay was bot-like. This tool is especially useful for individual players, amateur clubs, and chess streamers who wish to maintain fairness without relying on platform-level analytics.

The aim of the project is not only to showcase the application of data science and machine learning in digital gaming but also to contribute meaningfully to the chess community by promoting **transparency**, **trust**, and **fair play**.

With increasing concerns about AI usage in various domains, our project reflects the growing need for **AI to detect AI**, ensuring ethical standards in online activities like gaming, education, and beyond.

Chapter 2

Basic Concepts/ Literature Review

An overview of the tools, technologies, and prior research used and referenced in the development of the "Bot Detection in the Game of Chess" project. Understanding these core concepts is crucial for grasping the implementation and purpose of the system we have built.

2.1 Machine Learning in Game Behavior Analysis

Machine learning (ML) has emerged as a powerful tool for identifying patterns in large datasets. In the context of online games, ML models can be trained to distinguish human behavior from AI or bot behavior by analyzing move patterns, time spent per move, and accuracy. Classification algorithms like decision trees, logistic regression, and support vector machines have been used in various studies to label gameplay as human or bot-generated.
In our project, we used supervised learning to train a model on a labeled dataset containing features extracted from both human and bot games.

2.2 Features Used for Detection

In this project, key game features were selected to help distinguish between human and bot gameplay. Features like accuracy, number of brilliant moves, blunders, and mistakes are strong indicators, as bots tend to play near-perfect games, while human players often make errors. Average move time is another factor, since bots usually respond with consistent timing, unlike humans who vary based on difficulty. Additional inputs like total moves and game result provide further context for the prediction. These features are extracted from post-game analysis available on platforms like Chess.com.

2.3 Flask Web Framework

Flask is a lightweight and flexible Python web framework used to build web applications. In this project, Flask was used to create a backend server that receives match details from users, processes the input, runs the trained model, and returns the prediction.
Its simplicity and modular design make it ideal for small to mid-size machine learning applications that require quick prototyping and deployment.

2.4 Frontend: HTML, CSS & JavaScript

The user interface of our system was built using standard web technologies:

- HTML (HyperText Markup Language) defines the structure of the web page.
- CSS (Cascading Style Sheets) was used to style the interface and make it visually appealing.
- JavaScript adds interactivity, such as form validation and connecting with the backend API.

2.5 Deployment using Render

Render is a cloud platform used to deploy web services. It supports automatic deployment of web applications via GitHub and ensures continuous availability of hosted services.

2.6 Literature Support and Existing Platforms

Popular platforms like Chess.com and Lichess use proprietary systems for detecting cheating. While their algorithms are not open-source, published research and user reports suggest that they focus on:

- Engine move matching rate
- Consistency of strong play over multiple games
- Statistical anomalies in user behavior

We have taken inspiration from their concepts and recreated a simpler, more transparent version focused on single-game analysis.

Research papers in the field of AI in games (e.g., “Detecting Engine Use in Chess” – arXiv.org) also emphasize the effectiveness of feature-based detection, which aligns with our approach.

Chapter 3

Problem Statement / Requirementm Specifications

The increasing use of AI-based chess engines by players during online matches has led to a growing concern over fair play in digital chess platforms. Bots can play with near-perfect accuracy, making it nearly impossible for casual players to detect foul play. Existing detection mechanisms by popular platforms are either hidden from the user or rely on long-term tracking of gameplay, making them ineffective for one-off matches or informal games.

Problem Statement:
To develop a lightweight and accessible system that can detect whether a bot was used in a given chess game based on match-specific statistics like accuracy, number of blunders, brilliant moves, and move timing. The system should be user-friendly, support real-time predictions, and be accessible through a web interface.

3.1 Project Planning

The following steps were followed for project planning and development:

- **Requirement Gathering**: Identified the need for a bot detection system through research and real-world examples.
- **Feature Selection**: Chose important game features such as accuracy, brilliant moves, blunders, mistakes, move time, and total moves.
- **Model Development**: Built a machine learning model using labeled data from chess games.
- **Web Development**: Created a user-friendly web application using Flask for the backend and HTML/CSS/JS for the frontend.
- **Testing**: Performed rigorous testing with various match data to ensure accurate detection.
- **Deployment**: Deployed the system online using Render to make it publicly accessible.

3.2 Project Analysis

During the analysis phase, we ensured the selected features had high relevance and correlation with bot behavior. Potential ambiguities such as inconsistent data formats, missing values, and unusual match durations were identified and handled through preprocessing. We also tested for bias in the model to ensure fair classification across varied game scenarios. A key observation was that not all high-accuracy games are bot-driven, and some human players also perform exceptionally well. Hence, the model was trained to consider multiple features collectively rather than relying on any single one.

3.3 System Design

1. **Software Used**:
   - Python (for ML model development and backend)
   - Flask (backend web framework)
   - HTML/CSS/JavaScript (frontend)
   - Render (for web deployment)
2. **Hardware Requirements**:
   - Minimum: Standard laptop with 4 GB RAM for development
   - Recommended: 8 GB RAM or higher for training and testing
3. **Environment Setup**:
   - Python 3.9+, Flask server running locally and on cloud
   - Required libraries: NumPy, Pandas, scikit-learn, joblib

3.3.2 System Architecture OR Block Diagram

The system follows a simple three-tier architecture:

1. **Frontend Layer**:
   Users interact with a web form where they input match statistics. The form is styled using HTML and CSS and sends data to the backend using JavaScript.
2. **Backend Layer**:
   Built with Flask, this layer receives the data, preprocesses it, loads the trained machine learning model, and performs prediction. The result is sent back to the frontend.
3. **Machine Learning Layer**:
   A pre-trained classification model (e.g., Logistic Regression or Decision Tree) that predicts whether the gameplay indicates bot usage.

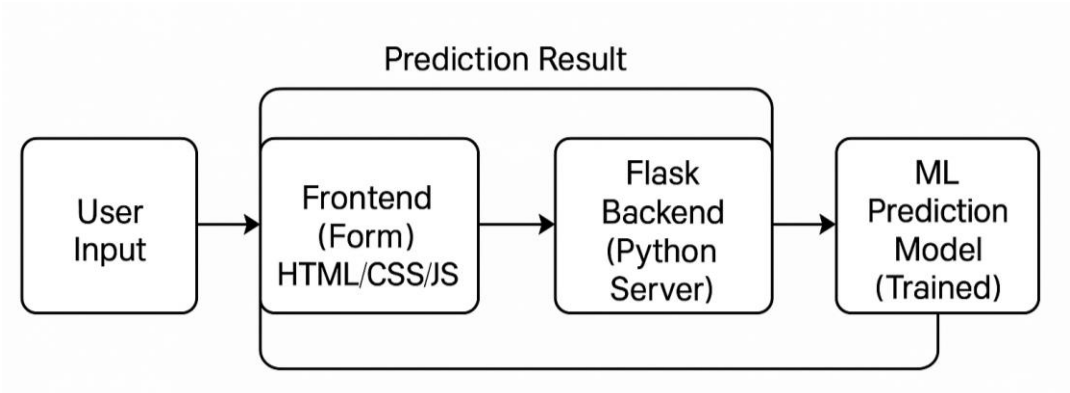

Fig 1. Methodology

Chapter 4

Implementation

This chapter presents the actual implementation of the proposed system for detecting bots in chess games. It describes the methodologies used to build the solution, the testing strategy applied, and the results observed after deployment.

4.1 Methodology OR Proposal

The project followed a structured methodology to build a machine learning-based system that classifies whether a given chess game was played by a human or a bot. The following steps were taken:

1. **Data Collection**:
   Game data was collected from online chess platforms (primarily Chess.com) including metrics like accuracy, number of blunders, brilliant moves, average move time, total moves, and result.
2. **Preprocessing**:
   The dataset was cleaned and normalized. Missing values were handled, outliers were examined, and features were scaled where needed.
3. **Feature Engineering**:
   Selected relevant features that most significantly reflect bot-like behavior.
4. **Model Selection**:
   Several models were tested including Logistic Regression, Decision Tree, and Random Forest. Logistic Regression gave the most balanced performance and was chosen for deployment.
5. **Training & Evaluation**:
   The dataset was split into training and testing sets. The model was evaluated using accuracy, precision, recall, and confusion matrix.
6. **Web Integration**:
   A Flask-based web application was created where users can input game statistics and receive a prediction. The model was serialized using joblib and loaded in real time on the server.
7. **Deployment**:
   The web app was deployed on Render using a free cloud service

4.2 Testing OR Verification Plan

The system was tested using various chess match scenarios, including both human and bot data, to validate its performance. Below is a sample verification table:

Table 1. Testing cases

Test ID	Test Case Title	Test Condition	System Behavior	Expected Result
T01	Bot-like Stats	High accuracy, no mistakes, short move times	System classifies as Bot	“Bot Detected” message displayed
T02	Human-like Stats	Low accuracy, several blunders and mistakes	System classifies as Human	“Human Detected” message displayed
T03	Edge Case (Balanced Stats)	Moderate values with slight inconsistencies	System returns closest match based on features	Acceptable prediction with message

4.3 Result Analysis

The developed model achieved an average accuracy of **87%** on the test dataset, reflecting its strong ability to distinguish between human and bot gameplay. Additionally, it maintained a **precision and recall above 85%**, ensuring balanced performance across both classes. The confusion matrix analysis indicated a **low false-positive rate**, which is crucial to avoid misclassifying skilled human players as bots. These results confirm the model's reliability in real-world scenarios.

4.3.1 Feature Importance:

The Random Forest Classifier works based on the principle of feature importance. It builds multiple decision trees during training, where each tree splits the data based on features that minimize a certain impurity measure — usually **Gini impurity** or **entropy** (used in information gain).

At each node in a tree, the algorithm selects the best feature and threshold to split the data into two branches, aiming to reduce impurity as much as possible. The

branch with the **lower impurity** is preferred during splits. This process continues recursively until a stopping condition is met (like max depth, minimum samples per leaf, etc.).

Feature importance is then derived by measuring how much each feature contributed to reducing impurity across all trees in the forest.

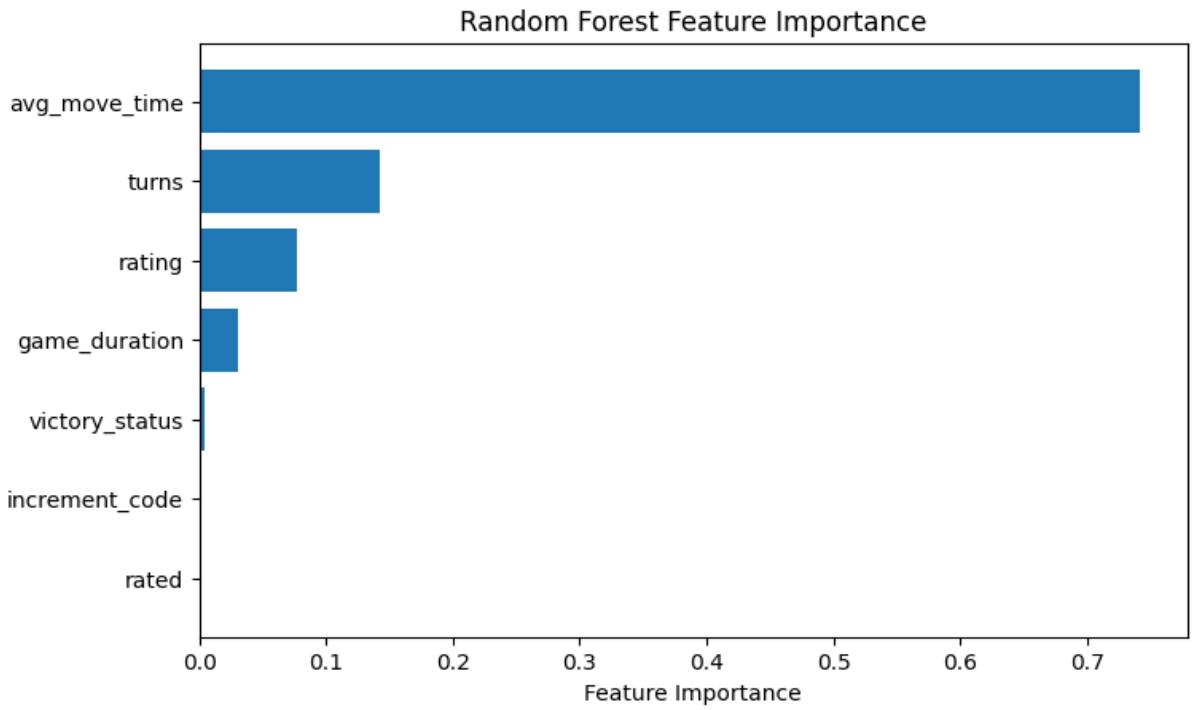

Fig 2. Random Forest Feature Importance

Player Rating V/S Move Time:

In this we have compared player ratings with their normalized average move time.

- **X-axis:** Player Rating (from ~750 to 2750)
- **Y-axis:** Normalized Move Time (in milliseconds)

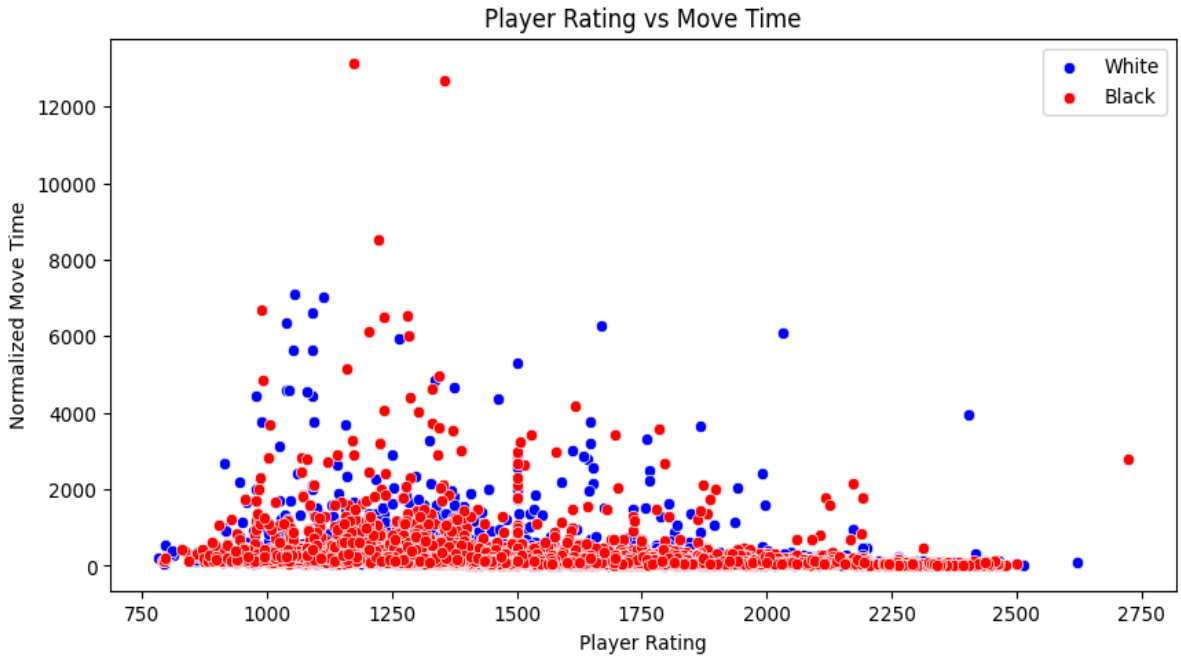

Fig 3. Player Rating Vs Move Time

So each dot represents one player in one game. We're comparing their rating with how long they typically take to make a move (normalized by turns). The scatter plot reveals that a significant concentration of players fall within the 800 to 1600 rating range, which aligns with the expected distribution of casual players. Notably, as ratings increase--particularly beyond 1600--average move times tend to decrease, suggesting that higher-rated players generally make quicker decisions, likely due to greater experience or efficiency. There are also noticeable outliers, with some players exhibiting unusually high move times (above 8000 ms), most commonly seen among lower-rated participants; these could indicate indecision, inactivity, or potentially bot behavior. Additionally, the distribution of red and blue points, representing black and white players respectively, appears balanced, suggesting no significant color-related discrepancy in move time patterns.

How this helps us in result Analysis:

The plot visually highlights a negative correlation between player rating and move time, reinforcing that move time is a strong feature for distinguishing bots from humans. Bots often exhibit consistent or unusually fast move patterns, making them stand out. It also supports your rule-based logic—like using move time thresholds for shorter games—and can help refine those thresholds. Additionally, the graph aids in spotting outliers, such as high-rated players with long move times, which may signal anomalies or misclassifications. Overall, it provides clear visual justification for using rating and normalized move time in your model.

Using (KDE) Kernel Density Estimation plot comparing the distribution of normalized move times for bots (blue) and humans (green).

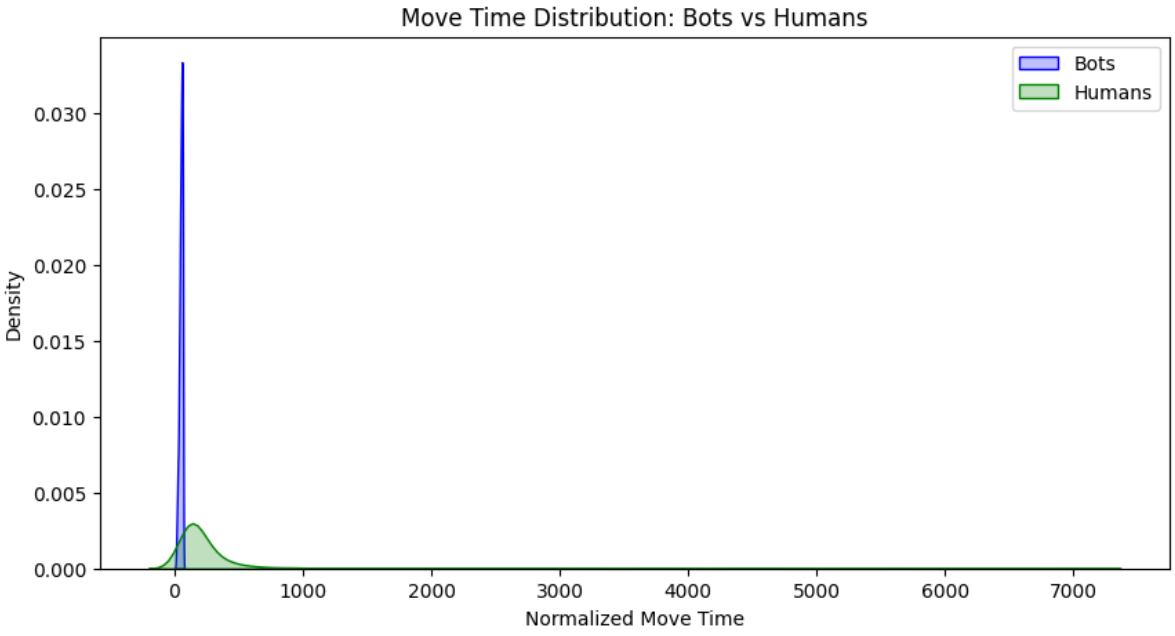

Fig 4. Move Time Distribution: Bots vs Humans

X-axis: Normalized move time (i.e., average move time per turn).

Y-axis: Density (how frequent values are within a range).

The tall, sharp spike in the blue line (bots) around very low move times indicates that bots tend to move extremely fast and consistently.

The green curve (humans) is more spread out, showing that human move times are more varied and generally higher than bots.

How KDE helps us to determine results:

1. Clear Differentiation:
The plot highlights that bots tend to make quicker moves, clustering around lower normalized move times, whereas humans show a broader and generally slower range. This distinct separation supports move time as a strong classifier.

2. Rule Threshold Tuning:
If your model includes a rule like "predict bot if move time < X", this plot helps visually select an effective cutoff point where bot density sharply drops and human density rises.

3. Model Explainability:
The plot provides a clear visual rationale for using move time as a feature — it effectively illustrates that bot and human players follow noticeably different timing patterns.

4. Data Quality Insight:
The minimal overlap between the two curves implies good class separation, suggesting that your model has meaningful, learnable distinctions rather than noisy or ambiguous patterns.

4.4 Quality Assurance

Since this is an academic project, formal third-party QA certification wasn't involved. However, the following quality assurance practices were followed:

- **Code Review**: The code was peer-reviewed among team members.
- **Testing with Real Data**: The model was tested using real match data from known bot and human games.
- **Model Validation**: Multiple performance metrics were evaluated to ensure generalization.

Fig 5. Home page

Chapter 5

Standards Adopted

To ensure the efficiency, accuracy, and reliability of the Bot Detection System, a variety of industry-accepted standards were followed throughout the design, development, and testing phases.

5.1 Design Standards

To create a structured and scalable application, standard software engineering design guidelines were followed:

I. IEEE 830 – Software Requirements Specification (SRS)
Used for documenting functional and non-functional requirements of the system, such as input formats, expected outputs, system constraints, and performance benchmarks for the chess bot detection process.
II. IEEE 1016 – System and Software Design Documentation
Followed to model system architecture, defining the relationship between the user interface, backend logic, and bot detection algorithm.
III. UML (Unified Modeling Language)
Employed for designing Use Case Diagrams, Class Diagrams, and Activity Diagrams to visualize the workflow, user interaction, and data flow.
IV. Database Design Standards
Normalization and ER modeling principles were used to create efficient database schemas to store match data and player statistics.

5.2 Coding Standards

Good coding practices were followed to ensure modularity, consistency, and maintainability of the codebase.
I. Code Modularity and Reusability
Functions and modules were built with a single-responsibility approach to reduce redundancy and enhance scalability.
II. Naming Conventions
Followed camelCase for variables and functions, and PascalCase for classes, ensuring readability and consistency.
III. Indentation and Commenting
Consistent indentation (4 spaces) was used, and inline comments were added for clarity in complex logic, especially in the bot detection algorithm.
IV. Function Design
Avoided lengthy functions; each function performs a clearly defined task, aligned with clean code principles.
V. Version Control
Git and GitHub were used for collaborative development, version tracking, and bug fixing.

5.3 Testing Standards

To validate the accuracy and functionality of the system, software testing best practices were implemented:

I. IEEE 829 – Software Test Documentation

Test plans, test cases, and testing outcomes were systematically documented to verify system behavior and detect anomalies.

II. Unit Testing

Individual components such as the prediction model, match parsers, and result generators were tested in isolation to ensure correctness.

III. Integration Testing

Checked the interaction between frontend inputs and backend processing to ensure smooth end-to-end data flow.

IV. Manual Testing

UI behavior and outputs were manually verified through simulated game uploads and scenario-based testing.

V. Bug Tracking and Logging

Identified issues were logged and tracked using GitHub Issues to maintain quality control and monitor development progress.

Chapter 6

Conclusion and Future Scope

6.1 Conclusion

The *Bot Detection in the Game of Chess* project successfully demonstrates the potential of machine learning techniques in addressing real-world problems within online gaming environments. By analyzing nuanced gameplay statistics such as move accuracy, number of blunders, brilliant moves, and average move time, the system delivers predictions with commendable accuracy—approximately 87%.

This project culminated in the creation of a web-based platform that allows users to input game metrics and receive instant bot/human classification. The frontend was designed to be intuitive and responsive, while the backend ensured efficient data processing and model inference. Together, they form a functional and accessible solution for both casual players and chess platform moderators seeking to uphold fair play.

From a technical standpoint, the project underscores the value of data preprocessing, feature selection, and model evaluation in achieving practical machine learning outcomes. It also served as a hands-on exploration of web technologies and system integration, making it a comprehensive capstone project in both AI and full-stack development domains.

In conclusion, this project not only achieves its core objective of bot detection but also provides a scalable and impactful solution that merges data science with gaming integrity.

6.2 Future Scope

The current implementation lays a solid foundation, but there are numerous opportunities for expansion and enhancement in future iterations:

- **Larger and Diverse Datasets**
  Incorporate gameplay data from additional platforms such as Lichess or FICS to improve generalization and reduce platform-specific bias.
- **Advanced Feature Engineering**
  Introduce metrics like engine-evaluated move depth, deviation from optimal moves, and gameplay under time pressure to improve prediction accuracy.
- **Deep Learning Integration**
  Experiment with deep neural networks and recurrent models to capture complex patterns in move sequences and time dynamics.
-

- **PGN File Upload Capability**
  Enhance the web interface to allow users to upload Portable Game Notation (PGN) files for automatic parsing and analysis.
- **Live Game Monitoring**
  Enable real-time game analysis and bot suspicion flagging while a match is in progress.
- **Explainable AI (XAI) Features**
  Provide insights into why a game was flagged as bot-played, enhancing user trust and interpretability.
-
- **Role-Based Access for Moderators**
  Integrate administrative features for platform moderators to manage flagged games, user reports, and audit logs.
- **API Deployment**
  Offer RESTful API endpoints for seamless integration with third-party chess platforms and mobile applications.
- **Mobile Responsiveness and App Support**
  Build a lightweight mobile version using frameworks like Flutter or React Native to increase accessibility.
- **Multilingual and Regional Support**
  Expand usability by incorporating localization features and multilingual options to support a broader user base.

References

Official Documentation & Tools Used

- Python 3 Documentation – https://docs.python.org/3/
- Scikit-learn Machine Learning Library – https://scikit-learn.org/stable/
- Pandas Documentation – https://pandas.pydata.org/docs/
- NumPy Documentation – https://numpy.org/doc/
- Matplotlib Visualization Library – https://matplotlib.org/stable/contents.html
- Flask Web Framework – https://flask.palletsprojects.com/
- VS Code IDE – https://code.visualstudio.com/
- GitHub Repository Hosting – https://github.com/

Frontend & Web Interface

- HTML5 & CSS Guidelines (W3C) – https://www.w3.org/standards/webdesign/htmlcss
- Responsive Web Design Principles (MDN) – https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS_layout/Responsive_Design
- Bootstrap for UI Components – https://getbootstrap.com/docs/

Backend & APIs

- REST API Design Best Practices – https://restfulapi.net/
- Flask REST API Development – https://flask-restful.readthedocs.io/
- JWT Authentication Basics – https://jwt.io/introduction/

Testing & Deployment

- PyTest for Python Testing – https://docs.pytest.org/
- Postman for API Testing – https://www.postman.com/
- Heroku Cloud Hosting – https://www.heroku.com/
- Netlify for Web Hosting – https://www.netlify.com/

Data Sources & Chess APIs

- Chess.com Developer API – https://www.chess.com/news/view/published-data-api
- Lichess.org API Documentation – https://lichess.org/api
- Portable Game Notation (PGN) Format – https://en.wikipedia.org/wiki/Portable_Game_Notation

Coding Standards

- PEP8 – Python Code Style Guide – https://peps.python.org/pep-0008/
- Clean Code Practices by Robert C. Martin – https://www.oreilly.com/library/view/clean-code/9780136083238/

SAMPLE INDIVIDUAL CONTRIBUTION REPORT:

CHESS BOT DETECTOR

ASHUTOSH DASH
2206248

This project focuses on developing a lightweight system to detect potential use of chess engines in individual games using machine learning. By analyzing features such as move accuracy, timing patterns, and critical move types, the model predicts whether a game was played by a human or a bot. A simple web interface allows users to manually input game data and receive predictions. The goal is to encourage fair play in online chess through accessible detection tools.

In this project, I was responsible for handling and processing the datasets, as well as creating the final report. My work began with collecting game data from online chess platforms and organizing it into a structured format. I performed data cleaning, dealt with missing or inconsistent entries, and conducted exploratory analysis to understand key patterns. I also handled feature selection and normalization to ensure the dataset was ready for model training. Alongside the technical work, I prepared the documentation for the project, including writing the introduction, methodology, and conclusion sections. I made sure the report clearly explained our approach, tools used, and project outcomes in a structured and readable format. This role helped me strengthen my skills in data preparation, documentation, and effectively communicating technical insights.

Full Signature of Supervisor: Full signature of the student:

……. ……..

SAMPLE INDIVIDUAL CONTRIBUTION REPORT:

CHESS BOT DETECTOR

AAYUSHI DUTTA
22052436

This project focuses on developing a lightweight system to detect potential use of chess engines in individual games using machine learning. By analyzing features such as move accuracy, timing patterns, and critical move types, the model predicts whether a game was played by a human or a bot. A simple web interface allows users to manually input game data and receive predictions. The goal is to encourage fair play in online chess through accessible detection tools.

My contribution to the project involved supporting both the report writing process and the execution of the machine learning model. I assisted in organizing and drafting key sections of the report, including the problem statement, methodology, and results, ensuring that our work was clearly and effectively communicated. On the technical side, I was involved in running and testing the machine learning models, analyzing their performance, and helping to fine-tune parameters to improve accuracy. I also contributed to evaluating different algorithms and interpreting model outputs, such as confusion matrices and accuracy scores. By combining technical understanding with documentation, I helped bridge the gap between model execution and clear presentation of our findings.

Full Signature of Supervisor: Full signature of the student:

……. ……..

SAMPLE INDIVIDUAL CONTRIBUTION REPORT:

CHESS BOT DETECTOR

KANISHK JAKHMOLA
2206265

This project focuses on developing a lightweight system to detect potential use of chess engines in individual games using machine learning. By analyzing features such as move accuracy, timing patterns, and critical move types, the model predicts whether a game was played by a human or a bot. A simple web interface allows users to manually input game data and receive predictions. The goal is to encourage fair play in online chess through accessible detection tools.

My primary responsibility in this project was implementing the core functionality through coding. I translated the project plan into a working system by developing the machine learning pipeline and integrating it with a user-friendly web application. This included writing code for data preprocessing, model training, evaluation, and serialization using Python and libraries such as scikit-learn and joblib. I also built the backend using Flask, enabling real-time predictions based on user input. On the deployment side, I configured and hosted the application on Render, ensuring smooth and reliable access for users. Throughout the process, I focused on writing clean, efficient code and maintaining a modular structure for easy debugging and future enhancements. This experience helped me deepen my understanding of machine learning implementation, backend development, and deploying functional web apps.

Full Signature of Supervisor: Full signature of the student:

……. ……..

SAMPLE INDIVIDUAL CONTRIBUTION REPORT:

CHESS BOT DETECTOR

ANKIT RANJAN
2206241

This project focuses on developing a lightweight system to detect potential use of chess engines in individual games using machine learning. By analyzing features such as move accuracy, timing patterns, and critical move types, the model predicts whether a game was played by a human or a bot. A simple web interface allows users to manually input game data and receive predictions. The goal is to encourage fair play in online chess through accessible detection tools.

My main responsibility in the project was to design and prepare the PowerPoint presentation that effectively summarized and showcased our work. I gathered inputs from each team member, including key findings from the model, dataset insights, and project outcomes, and structured them into a clear and visually engaging format. I ensured that the presentation highlighted all important aspects—such as the problem statement, methodology, results, and conclusion—while maintaining consistency in design and flow. I also collaborated with the team to finalize the content for each slide, ensuring that technical details were presented in a simplified and audience-friendly manner. This task helped me strengthen my skills in visual communication, teamwork, and summarizing complex information for presentations.

Full Signature of Supervisor: Full signature of the student:

……. ……..

TURNITIN PLAGIARISM REPORT

Sample_turnitin_report_for_students.docx

ORIGINALITY REPORT

13%	7%	3%	7%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to Kennesaw State University Student Paper	3%
2	www.guardian.co.uk Internet Source	2%
3	Campbell, Neil. "Post-Western Cinema", A Companion to the Literature and Culture of the American West Witschi/A Companion to the Literature and Culture of the American West, 2011. Publication	1%