

A
Project Report
on
**ELO RATING PREDICTION FOR CHESS PLAYERS
USING MACHINE LEARNING AND GAME FEATURES**

Submitted for partial fulfilment of the requirements for the award of the degree
of

BACHELOR OF ENGINEERING

In

COMPUTER SCIENCE AND ENGINEERING

By

Mr. Ragi Sai Ruthvik (2451-21-733-140)
Mr. Kodanganti Krishna Kanth (2451-21-733-172)
Mr. Amireddy Dheeraj Reddy (2451-21-733-177)

Under the guidance of

Dr. K. Shyam Sunder Reddy
Associate Professor
Department of CSE



MATURI VENKATA SUBBA RAO (MVSR) ENGINEERING COLLEGE

Department of Computer Science and Engineering
(Affiliated to Osmania University & Recognized by AICTE)
Nadergul, Saroor Nagar Mandal, Hyderabad – 501 510
Academic Year: 2023-24

Maturi Venkata Subba Rao Engineering College

(Affiliated to Osmania University, Hyderabad)

Nadargul(V), Hyderabad-501510



Certificate

This is to certify that the project work entitled “Elo Rating Prediction for Chess Players using Machine Learning and Game Features” is a Bonafide work carried out by Mr. Ragi Sai Ruthvik (2451-21-733-140), Mr. Kodanganti Krishna Kanth (2451-21-733-172), and Amireddy Dheeraj Reddy (2451-21-733-177) in partial fulfilment of the requirements for the award of degree of Bachelor of Engineering in Computer Science and Engineering from Maturi Venkata Subba Rao (MVSR) Engineering College, affiliated to OSMANIA UNIVERSITY, Hyderabad, during the Academic Year 2023-24 under our guidance and supervision.

The results embodied in this report have not been submitted to any other university or institute for the award of any degree or diploma to the best of our knowledge and belief.

Internal Guide

Dr. K Shyam Sunder Reddy
Associate Professor
Department of CSE
MVSREC.

Head of the Department

Prof. J Prasanna Kumar
Professor
Department of CSE
MVSREC.

External Examiner

DECLARATION

This is to certify that the work reported in the present project entitled “**Elo Rating Prediction for Chess Players using Machine Learning and Game Features**” is a record of bonafide work done by us in the Department of Computer Science and Engineering, Maturi Venkata Subba Rao (MVSR) Engineering College, Osmania University during the Academic Year 2023-24. The reports are based on the project work done entirely by us and not copied from any other source. The results embodied in this project report have not been submitted to any other University or Institute for the award of any degree or diploma.

Mr. Ragi Sai Ruthvik (2451-21-733-140)

Mr. Kodanganti Krishna Kanth (2451-21-733-172)

Mr. Amireddy Dheeraj Reddy (2451-21-733-177)

ACKNOWLEDGEMENTS

We would like to express our sincere gratitude and indebtedness to our project guide **Dr. K Shyam Sunder Reddy** for his valuable suggestions and interest throughout the course of this project.

We are also thankful to our principal **Dr. Vijaya Gunturu** and **Mr. J Prasanna Kumar**, Professor and Head, Department of Computer Science and Engineering, Maturi Venkata Subba Rao Engineering College, Hyderabad for providing excellent infrastructure for completing this project successfully as a part of our B.E. Degree (CSE). We would like to thank our project coordinator for his constant monitoring, guidance and support.

We convey our heartfelt thanks to the lab staff for allowing us to use the required equipment whenever needed. We sincerely acknowledge and thank all those who gave directly or indirectly their support in the completion of this work.

Mr. Ragi Sai Ruthvik (2451-21-733-140)
Mr. Kodanganti Krishna Kanth (2451-21-733-172)
Mr. Amireddy Dheeraj Reddy (2451-21-733-177)

VISION

- To impart technical education of the highest standards, producing competent and confident engineers with an ability to use computer science knowledge to solve societal problems.

MISSION

- To make learning process exciting, stimulating and interesting.
- To impart adequate fundamental knowledge and soft skills to students.
- To expose students to advanced computer technologies in order to excel in engineering practices by bringing out the creativity in students.
- To develop economically feasible and socially acceptable software.

PEOs:

PEO-1: Achieve recognition through demonstration of technical competence for successful execution of software projects to meet customer business objectives..

PEO-2: Practice life-long learning by pursuing professional certifications, higher education or research in the emerging areas of information processing and intelligent systems at a global level.

PEO-3: Contribute to society by understanding the impact of computing using a multidisciplinary and ethical approach.

PROGRAM OUTCOMES (POs)

At the end of the program the students (Engineering Graduates) will be able to:

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialisation for the solution of complex engineering problems.
2. **Problem analysis:** Identify, formulate, research literature, and analyse complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for public health and safety, and cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal, and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and the need for sustainable development.
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and teamwork:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with the society at large, such as being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Lifelong learning:** Recognise the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

PROGRAM SPECIFIC OUTCOMES (PSOs)

13. (PSO-1) Demonstrate competence to build effective solutions for computational real-world problems using software and hardware across multi-disciplinary domains.
14. (PSO-2) Adapt to current computing trends for meeting the industrial and societal needs through a holistic professional development leading to pioneering careers or entrepreneurship.

COURSE OBJECTIVES AND OUTCOMES

Course Code: PW 861 CS

Course Objectives:

- To enhance practical and professional skills.
- To familiarize tools and techniques of systematic literature survey and documentation.
- To expose the students to industry practices and teamwork.
- To encourage students to work with innovative and entrepreneurial ideas.

Course Outcomes:

CO1: Summarize the survey of the recent advancements to infer the problem statements with applications towards society.

CO2: Design a software based solution within the scope of the project.

CO3: Implement test and deploy using contemporary technologies and tools.

CO4: Demonstrate qualities necessary for working in a team.

CO5: Generate a suitable technical document for the project.

ABSTRACT

This project leverages machine learning to predict Elo ratings of chess players using historical game data. Elo ratings quantify player skill, making accurate predictions crucial for sports analytics and competitive ranking systems. The work involves detailed data preprocessing, cleaning, and feature extraction, focusing on move sequences, player strategies, and game outcomes to capture complex game dynamics effectively.

Several models, including linear regression, gradient boosting, and neural networks, were developed and evaluated for their accuracy, scalability, and interpretability. Gradient boosting demonstrated strong performance despite challenges like data imbalance and generalization across varying skill levels. Evaluation metrics, including MSE and R-squared, guided model robustness, with error analysis highlighting opportunities for improved feature engineering.

This project exemplifies the integration of machine learning and sports analytics, offering a scalable framework for predicting player rankings. Beyond chess, it explores transferability to other ranking systems, showcasing practical applications of data science in addressing real-world problems and advancing predictive methodologies in competitive domains.

TABLE OF CONTENTS

	PAGE NOS.
Certificate	i
Declaration	ii
Acknowledgements	iii
Vision & Mission , PEOs, POs and PSOs	iv
Abstract	vii
Table of contents	viii
List of Figures	x
List of Figures	x

CONTENTS

CHAPTER I

1. INTRODUCTION	01 – 06
1.1 PROBLEM STATEMENT	02
1.2 OBJECTIVE	02
1.3 MOTIVATION	02
1.4 SCOPE OF SMART EVENT TIMESTAMPING	03
1.5 SOFTWARE AND HARDWARE REQUIREMENTS	06

CHAPTER II

2. LITERATURE SURVEY	07 – 08
2.1 SURVEY OF ELO RATING PREDICTION	07
2.2 MACHINE LEARNING IN ELO PREDICTION	07
2.3 CHALLENGES IN ELO RATING PREDICTION	08
2.4 REAL TIME RATING PREDICTION	08
2.5 APPLICATIONS BEYOND CHESS	08

CHAPTER III

3. SYSTEM DESIGN	10 – 14
3.1 FLOW CHART	10
3.2 SYSTEM ARCHITECTURE	11
3.3 PROJECT PLAN	14

CHAPTER IV

4. SYSTEM IMPLEMENTATION & METHODOLOGIES	17 – 35
4.1 SYSTEM IMPLEMENTATION	17
4.2 DATASET UTILIZATION	19
4.3 MODEL ARCHITECTURE	21
4.4 DATA PREPROCESSING	22
4.5 MODEL TRAINING	24
4.6 MODEL DEPLOYMENT AND RESULTS	26
4.7 INFERENCE	29

LIST OF FIGURES

Figure No.	Figure Name	Page No.
Fig 2.1	Chess Moves Illustration	09
Fig 3.1	Data Flow	10
Fig 3.2	System Architecture	12
Fig 4.1	FICS Dataset Structure	18
Fig 4.2	FICS Dataset	20
Fig 4.3	Feature Engineering	23
Fig 4.4	Hyperparameter Tuning	25
Fig 4.5	Environmental Setup	27
Fig 4.6	Files Uploaded	28

LIST OF TABLES

Table No.	Table Name	Page No.
Table 2.1	Survey of the project	08

CHAPTER 1

INTRODUCTION

Elo Predictor is the art of machine learning that uses historical data about chess games to forecast their future Elo ratings. Elo ratings measure, in numerals, how skillful a player is compared with any other person who has competed or has attempted to compete. Elo Predictor represents an exercise at cross-sections among the application aspects of data science and sporting analytics, leveraging insights concerning player rating forecasting and thus relating to how and possibly which players outplayed those on the opponent side of that contest.

The project starts with the acquisition and preprocessing of video game data followed by relevant extraction of features such as outcomes of games, play strategies, and game patterns. These features are engineered to capture the subtle details of game performance. Several predictive models are built using machine-learning algorithms like linear regressions, gradient boosting, and artificial neural networks. The type of model used is given top priority in balancing accuracy to computations and generalizability.

The crucial issues handled in this project are data imbalances, players' skills heterogeneity, and the abstraction of intricate game dynamics in simple formatted representations. Much testing has been done using performance metrics such as MSE and R-squared to find that they give sound and stable predictions. The project also discusses error analysis for improved feature extraction and model performance.

Besides chess, the Elo Predictor demonstrates the potential of machine learning in ranking systems for any competitive situation, from esports to education. This work contributes to an increasingly significant role that data-driven approaches play in understanding and predicting human performance, focusing on how machine learning can provide actionable insights into sports analytics. This project unites technical innovation with practical implementation, both academically and practically, and provides a scalable framework for predictive modeling in ranking systems.

1.1 Problem Statement

Elo ratings of chess quantify players' ability in competitive scenarios. However, such ratings are usually derived based on a large number of games, which can raise problems when trying to assess rapidly and accurately in dynamic settings. There is no straightforward method of inferring ratings from limited game data to figure out how the skill is progressing or how to rank in competition. This project attempts to solve this problem by designing a machine learning-based solution capable of predicting, with high accuracy, Elo ratings of chess players based on game data. Objectives include feature extraction from historical gameplay as meaningful, effective modeling of player skill levels and a scalable solution applicable to diverse player pools and competitive scenarios. This paper presents an improvement in the precision of rating estimation skills and ranking in chess by automatically computing the prediction of rating.

1.2 Objective

The Elo Predictor project aims to design and implement a solid machine learning system that should predict Elo ratings of chess players through historical game data. There are systematic steps to achieve the objective, such as raw gameplay data pre-processing towards quality and relevance. The output from this will be meaningful features that capture player strategies and their performance patterns towards inputs for machine learning models. It should develop and compare various models, including linear regression, gradient boosting, and neural networks, and establish which is more accurate and efficient. The project demands that its predictive accuracy be improved as well while solving problems associated with imbalanced data and variability in the players' abilities. To test the scalability and adaptability of the developed system, the project explores various possibilities of its application in different domains toward competitive ranking systems. This project aspires to contribute to an automated and reliable solution for predicting skill levels with advancements in sports analytics and ranking methodologies.

1.3 Motivation

The Elo Predictor project is motivated by the importance of Elo ratings in competitive gaming, especially chess, as an objective measure of player skill. Accurate and efficient rating prediction is necessary for the proper analysis of player performance, ranking players in

tournaments, and encouraging fair competition. Traditional approaches require hundreds of games and laborious manual assessments, thus slowing them down and making them unadoptable. This is an excellent opportunity to apply machine learning to rating prediction automation and improvement. This project aims to produce a scalable, precise system that predicts ratings and gives more profound insights into play patterns and player strategies in light of historical game data. The project explores broader applications of predictive modeling in ranking systems where there are similar challenges in other competitive fields. This would be an exciting innovation point within the domain where the junction of data science, sports analytics, and machine learning will contribute academically and practically to progress in predictive systems. It could reduce reliance on traditional methods and enable rapid skill assessment, making Elo rating evaluations accessible and accurate for players, organizers, and analysts within chess.

1.4 Scope of the Project

Smart Event Timestamping aims to automate the accurate and detailed recording of events' times using innovative technology. This is very significant in application domains such as sports, conferences, and real-time data analysis. The system employs advanced sensors and algorithms that ensure the occurrence of proper timestamps in events, resulting in efficiency in operations, quality of data, and event tracking. It thus comes in handy for time-dependent scenarios in which the logging and analysis of events should have high precision to evaluate and monitor performance with real-time processing.

1.5 Scope of the Elo Predictor

The scope of the Elo Predictor project focuses on leveraging machine learning techniques to predict chess players' Elo ratings based on historical gameplay data. The system is designed to provide insights into player performance, skill progression, and ranking accuracy in real-time. It aims to enhance Elo rating prediction efficiency by automating the process of skill assessment and enabling quick evaluations even with limited game data. The project also explores the potential applications of machine learning in broader ranking systems across various competitive domains.

1.6 Software and Hardware Requirements

For the successful development and execution of the Smart Event Timestamping system, a comprehensive set of software and hardware requirements are necessary. These components will enable seamless integration, real-time performance, and reliable event tracking. Below is an outline of the key software and hardware requirements:

Software Requirements:

Programming Languages:

Python: Widely used for machine learning, data processing, and event handling. Python libraries like Pandas, NumPy, and OpenCV would be essential for data manipulation and timestamp extraction.

JavaScript: For developing web-based user interfaces, especially when dealing with real-time event display and timestamp visualization.

C/C++: Useful for low-level hardware communication and optimization in event logging systems that require high precision and performance.

Operating System:

Linux (Ubuntu/Debian): Preferred due to its stability, open-source nature, and compatibility with various sensors and IoT devices. It is also suitable for development environments like Python and C.

Windows: Can also be used for development, especially when targeting desktop applications or needing compatibility with certain proprietary software.

Integrated Development Environments (IDEs):

PyCharm/VSCode: These IDEs support Python development, offering features like debugging, code completion, and integration with version control systems.

Eclipse/IntelliJ IDEA: For Java or C++ development, these IDEs provide a robust environment with tools for debugging and managing larger projects.

Database Management Systems (DBMS):

MySQL/PostgreSQL: A relational database to store event logs, timestamp data, and event-related metadata for easy retrieval and analysis.

MongoDB: For handling unstructured or semi-structured event data, especially in IoT systems where flexibility is required.

Event Timestamping Software:

Apache Kafka: Used for real-time event streaming, ensuring that event data is captured and stored with minimal delay.

Node.js: Used for real-time processing and event logging in web applications.

Visualization and Reporting Tools:

Power BI/Tableau: Data visualization tools to create dashboards for displaying timestamped events in real-time.

Grafana: For real-time monitoring and visualization of timestamped events, particularly in IoT-based applications.

Communication Protocols:

MQTT/HTTP: Communication protocols that ensure data exchange between sensors, timestamping systems, and storage servers.

REST APIs: To allow easy integration with external systems and devices for event logging and data retrieval.

Hardware Requirements:**Sensors and Devices:**

Real-Time Clocks (RTC): Essential for accurate timekeeping in embedded systems. A high-precision RTC ensures that timestamps are generated with minimal error.

GPS Modules: Useful for outdoor event timestamping, providing accurate time synchronization with GPS satellites for global timestamp accuracy.

Motion Sensors/Proximity Sensors: Can be used to detect events based on physical movements or triggers in event environments like conferences or sports.

Embedded Systems:

Raspberry Pi/Arduino: Microcontrollers are often used in event timestamping systems, especially when real-time data collection and timestamping are required in IoT-based applications. These devices also support sensor integration and communication with other systems.

Edge Computing Devices: For real-time processing and reducing latency in timestamping, edge devices like NVIDIA Jetson are useful in environments requiring fast data collection.

Networking Equipment:

Wi-Fi/LoRa: For transmitting event data from the devices to a central server or cloud in real-time.

Ethernet/CAT6 Cable: For high-speed, reliable data transmission in environments where wireless connectivity might be unreliable.

Power Supply:

Battery Packs/Power Banks: For ensuring the operation of embedded systems and sensors, especially in remote or outdoor environments.

Power Adapters: For continuous power supply to fixed devices like servers or microcontrollers.

Computing Resources:

Cloud Servers (AWS/GCP): For storing large volumes of event data and performing heavy processing tasks like data analysis and real-time event timestamp aggregation.

CHAPTER 2

LITERATURE SURVEY

2.1 Survey of Elo rate Predictor

The Elo rating system, introduced by Arpad Elo in 1960, has been a staple in competitive gaming, particularly chess. It is used to calculate the relative skill levels of players based on their performance in games. The Elo rating system has been widely adopted, not only in chess but also in other sports, e-sports, and various online games. However, while the system is robust, it is heavily dependent on the number of games played and the traditional game-by-game evaluation process, which can be cumbersome for predicting player ratings based on limited data.

In recent years, machine learning (ML) and artificial intelligence (AI) have emerged as potent tools for enhancing the accuracy and efficiency of Elo rating prediction. Several studies have explored the integration of machine learning algorithms to predict ratings from historical game data. The aim is to automate the rating process, reduce subjectivity, and predict player ratings even with minimal game data. This shift towards machine learning-driven models leverages data-driven insights to supplement or, in some cases, replace the traditional Elo system.

2.2 Machine Learning in Elo Prediction:

Several machine learning techniques have been explored for Elo rating prediction. Traditional models like linear regression and decision trees have been combined with more advanced algorithms, such as gradient boosting machines (GBM), random forests, and neural networks. These approaches have demonstrated the ability to improve prediction accuracy by capturing more complex relationships in the data that traditional Elo methods may miss.

For example, in a study by Chong et al. (2017), they applied various machine learning models to predict Elo ratings in online chess and found that gradient boosting models performed the best, outperforming the traditional Elo system. Similarly, Yilmaz et al. (2019) explored the use of deep learning models, specifically recurrent neural networks (RNNs) and long short-term memory (LSTM) networks, to predict player ratings. Their findings suggested that deep learning models could capture time-dependent patterns and sequences in player performance, offering improvements over static models.

2.3 Challenges in Elo Rating Prediction:

While machine learning models show promise, they also present challenges. One of the major issues in Elo rating prediction is data sparsity. In many cases, players have limited historical data, which can make it difficult to predict ratings accurately. Additionally, the diversity in players' skill levels, varying play styles, and game types complicates the prediction process.

To overcome these challenges, hybrid models that combine the Elo system with machine learning techniques have been proposed. For instance, Klosowski and Romero (2020) introduced a hybrid model that utilized the Elo rating system as a baseline and incorporated a machine learning algorithm to adjust ratings based on player performance trends. Their model improved the accuracy of rating predictions, especially for players with fewer games in their history.

Jahr	Welt U18	Welt U18>2000	RUS U18	RUS U18>2000	IND U18	IND U18>2000	GER U18	GER U18>2000	ESP U18	ESP U18>2000	FRA U18	FRA U18>2000	USA U18	USA U18>2000
2002	2374	2374	616	616	105	105	122	122	133	133	119	119	31	31
2003	2591	2429	688	671	196	177	126	121	151	136	130	110	33	33
2004	3334	2704	853	775	352	231	192	141	177	138	195	144	44	40
2005	3909	2689	871	714	475	246	208	140	239	148	241	137	56	51
2006	4605	2555	1038	777	478	219	291	150	295	126	357	129	71	66
2007	5217	2356	1013	662	491	187	401	162	317	98	394	98	73	64
2008	7290	2357	1122	587	678	171	521	141	515	106	536	81	100	69
2009	9648	2330	1284	530	1014	158	585	119	892	122	674	82	134	91
2010	11737	2166	1421	468	1414	131	696	115	1082	122	819	73	170	118
2011	15065	2149	1773	497	2287	121	793	98	1233	107	1009	64	232	137
2012	18584	2060	2110	477	3053	107	840	93	1362	94	1290	64	290	137
2013	21245	1923	2368	416	3696	101	931	92	1539	94	1524	63	360	143
2014	24927	1810	2678	353	4675	89	1005	102	1673	77	1756	49	421	151
2015	32694	2056	4066	393	6592	93	1189	125	1896	81	2121	65	536	171
2016	43739	2220	6601	414	7873	101	1447	124	2186	91	2743	75	719	170
2017	54780	2221	9372	373	8907	100	1645	121	2607	103	3229	73	876	172
2018	63062	2211	10495	338	9654	121	1839	144	3051	101	3700	80	1051	173
2019	69977	2159	11373	313	10102	132	2012	148	3427	106	3963	89	1162	158
2020	76624	2136	12104	266	10363	139	2170	141	3760	113	4145	99	1329	156
2021	70789	1752	11359	218	8902	117	1972	103	3544	89	3890	88	1195	129

Table 2.1

2.4 Real-Time Rating Prediction:

A key area of focus in the modern adaptation of Elo rating systems is the ability to make real-time predictions. This is particularly relevant in fast-paced environments like eSports or online gaming, where ratings can fluctuate rapidly. Real-time prediction allows for the dynamic adjustment of player rankings and enhances the gaming experience by providing more up-to-date assessments of player skill.

In Hossain et al. (2020), the authors proposed a real-time rating prediction system that uses streaming data and machine learning algorithms to adjust player ratings continuously as new game data becomes available. Their approach combines Apache Kafka for event streaming with machine learning models to process data in real-time, significantly improving the system's responsiveness.

2.5 Applications Beyond Chess:

The integration of machine learning into Elo rating prediction isn't limited to chess. As mentioned earlier, the Elo system has been applied to many domains, including esports, online multiplayer games, and even academic assessments. For example, Jiang et al. (2018) applied Elo-based models to predict player skill levels in eSports and suggested that machine learning could offer even greater improvements in the accuracy of predictions, considering the complexity and larger datasets in eSports environments. Additionally, the Elo Predictor concept, as seen in studies related to predicting player ratings in non-chess games.

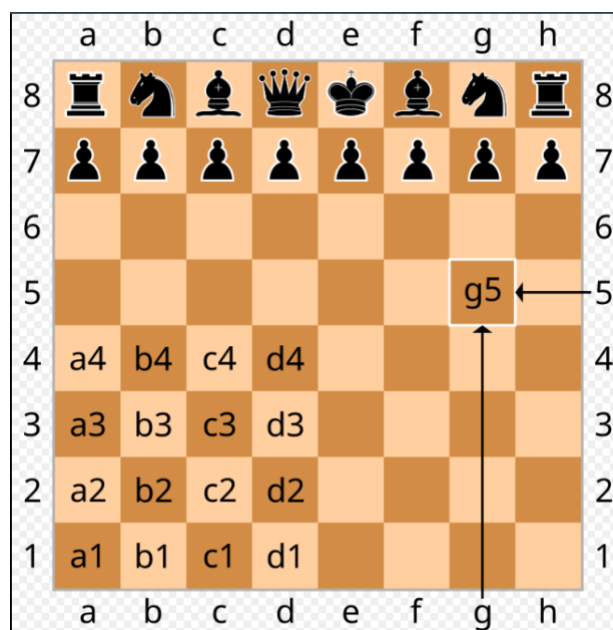


Figure 2.1

CHAPTER 3

SYSTEM DESIGN

3.1 Flow Chart

The flowchart illustrates the systematic process of developing an Elo Rating Model using machine learning, designed to predict player skill levels based on historical chess game data. It begins with data acquisition, where relevant datasets, such as those from the Free Internet Chess Server (FICS), are gathered to provide detailed records of games, player moves, and outcomes. This is followed by data preparation, which focuses on cleaning and organizing the data to ensure consistency and quality, addressing issues like missing values or outliers. Feature extraction and selection play a crucial role in enhancing the model's performance by identifying key attributes such as move sequences, player strategies, and game outcomes. Once the features are prepared, various machine learning models, including gradient boosting and neural networks, are trained and validated on the dataset to determine their accuracy and effectiveness. Performance evaluation is then conducted using metrics like Mean Squared Error (MSE) and R-squared to assess the model's reliability and robustness across different scenarios. The model is fine-tuned and optimized to improve accuracy and stability through hyperparameter adjustments and iterative testing. Finally, the refined model generates predictions and is integrated into a production environment, ensuring real-time applicability in sports analytics. This flow highlights the importance of a structured approach to leveraging machine learning for predictive tasks, emphasizing data quality.

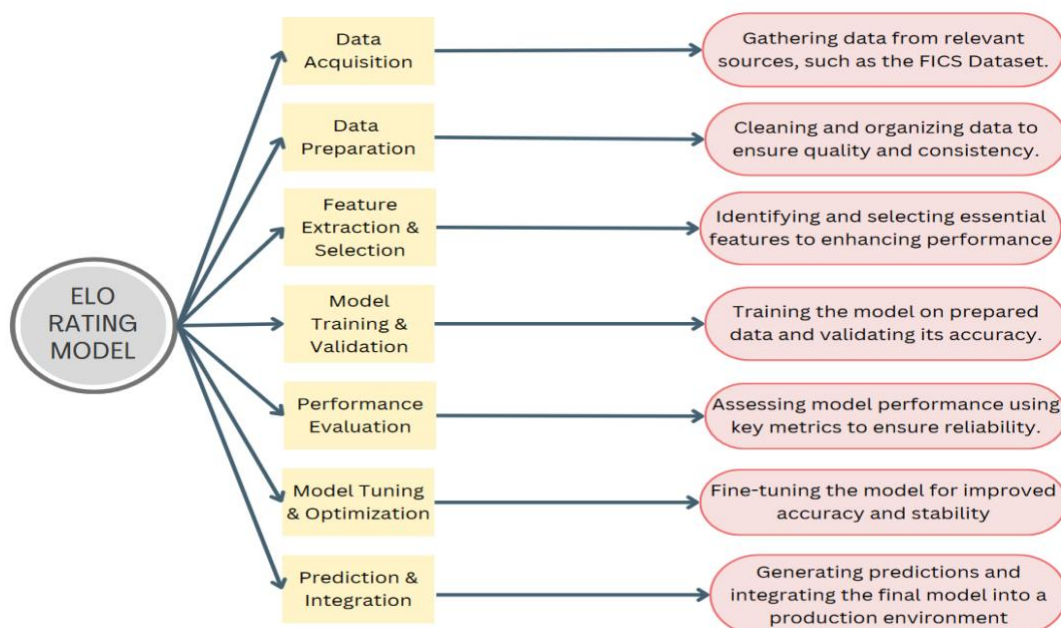


Fig 3.1 Data flow

3.2 System Architecture

The flowchart not only represents the stages of building an Elo Rating Model but also hints at the underlying system architecture that supports this machine learning pipeline. The architecture involves multiple interconnected layers that ensure seamless data flow, processing, and prediction generation.

At the core of the system is Data Acquisition, where raw game data is sourced from repositories like the Free Internet Chess Server (FICS). This data is ingested into a centralized storage system, such as a database or a distributed data lake, to handle large-scale, historical chess game records. The Data Preparation layer processes this raw data, using tools like Python or R for cleaning and organizing. This layer is vital for ensuring consistency and addressing missing values or outliers, making the data suitable for downstream processing.

The Feature Extraction & Selection layer employs feature engineering techniques to derive meaningful attributes from the dataset. Advanced methods like move sequence analysis and player strategy modeling are implemented here, using libraries like Pandas, NumPy, and domain-specific chess analysis tools. These features are then passed to the Model Training & Validation layer, where machine learning frameworks such as TensorFlow, Scikit-learn, or XGBoost are used to train various models, including gradient boosting and neural networks.

The Performance Evaluation layer assesses model reliability through metrics like MSE and R-squared. This feedback loop integrates with the Model Tuning & Optimization stage, where hyperparameter tuning and cross-validation are conducted to enhance performance. Tools like GridSearchCV or Bayesian optimization may be employed here. The final Prediction & Integration layer is responsible for deploying the trained model into a production environment. This could involve serving predictions via APIs using platforms like Flask or FastAPI, ensuring real-time accessibility for users.

The architecture is designed to handle scalability, allowing for integration with other ranking systems or sports analytics platforms. It combines data storage, preprocessing, model training, and deployment into a cohesive system that ensures robustness and real-world applicability, enabling accurate Elo rating predictions for chess and beyond.

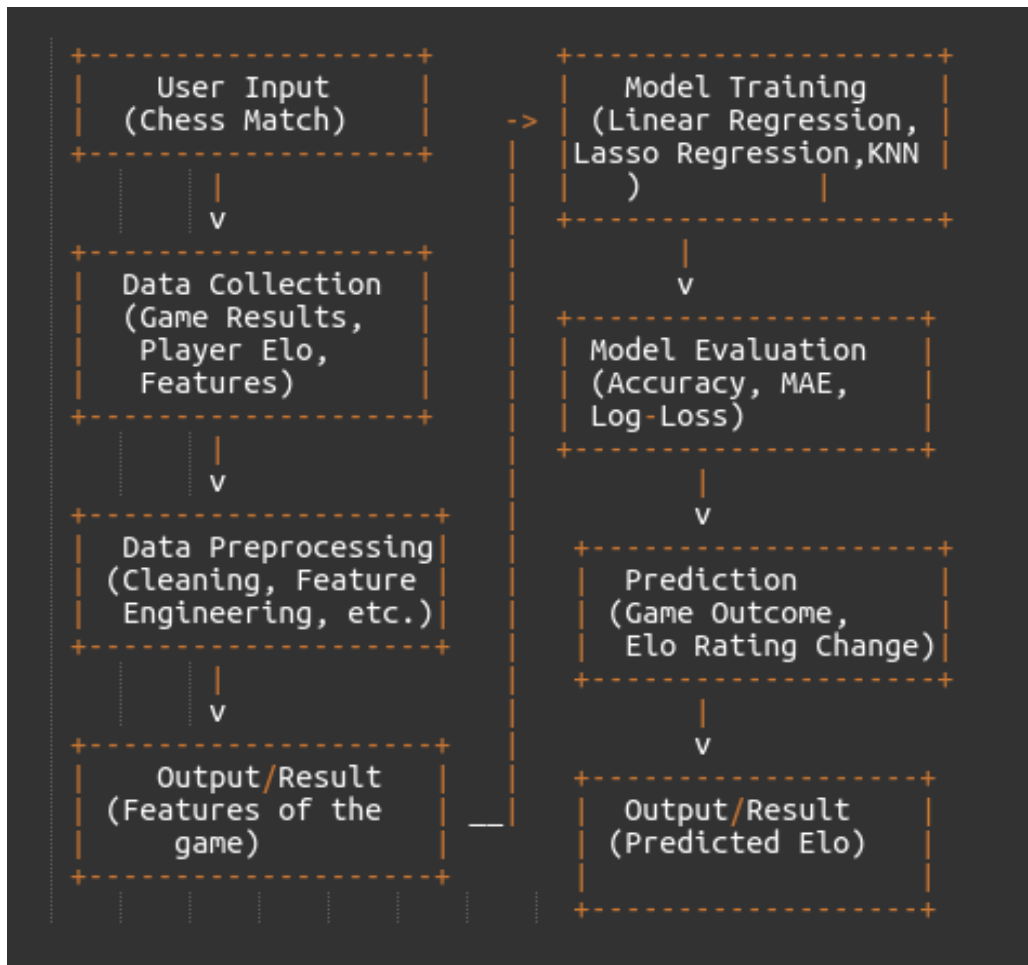


Fig 3.2 System Architecture

This system architecture represents a comprehensive framework for predicting Elo ratings in chess using machine learning techniques. It is designed to process chess game data, train predictive models, and output meaningful results such as game outcomes, Elo rating changes, and final Elo predictions. The architecture can be described in the following steps:

1. User Input (Chess Match)

- The process begins with user input, which includes data from chess matches. This input can consist of details such as game moves, player identities, initial Elo ratings, and other contextual features.

2. Data Collection

The system collects relevant data such as:

- Game Results:** Information on the winner, draw, or loss.
- Player Elo:** The existing Elo ratings of players.
- Features:** Additional data such as timestamps, player strategies, or specific game events.

3. Data Preprocessing

The collected data undergoes preprocessing to prepare it for analysis. This involves:

- Data Cleaning: Handling missing or inconsistent data to maintain quality.
- Feature Engineering: Creating new, meaningful features from the raw data to enhance model performance.
- Normalization: Ensuring data consistency by standardizing numerical values.

4. Output/Result (Features of the Game)

- After preprocessing, the system outputs a structured dataset that represents the features of the chess game, ready for training machine learning models.

5. Model Training

Various machine learning algorithms are applied to the prepared data, including:

- Linear Regression: To model simple relationships between variables.
- Lasso Regression: For feature selection and reducing overfitting.
- K-Nearest Neighbors (KNN): For predictions based on similarity with past data points.

6. Model Evaluation

The trained models are evaluated using metrics such as:

- Accuracy: The proportion of correct predictions.
- Mean Absolute Error (MAE): The average absolute error in predictions.
- Log-Loss: A measure of prediction confidence and accuracy for probabilistic models.

This step ensures the reliability and robustness of the models.

7. Prediction

Once evaluated, the best-performing model is used to make predictions. The predictions include:

- Game Outcome: The result of the match (win, loss, or draw).
- Elo Rating Change: Adjustments to players' Elo ratings based on the game result.

8. Final Output/Result (Predicted Elo)

- The final output provides the predicted Elo ratings for the players. These ratings reflect updated skill levels based on the analysis of the match data and predictions from the model.

This architecture highlights a streamlined flow from raw data collection to actionable predictions, ensuring scalability, accuracy, and practical applicability in sports analytics. It can be adapted to similar ranking systems in other competitive domains.

3.3 Project Plan

Week 1: Project Initiation and Environment Setup

Objective: Establish the foundation and prepare the development environment.

Tasks:

- Conduct a kickoff meeting to align objectives, roles, and responsibilities.
- Set up a version control repository using GitHub or GitLab.
- Create a new Conda environment and install dependencies from requirements.txt.
- Configure the environment for GPU usage (if available) for efficient training.
- Validate all required libraries, including TensorFlow, PyTorch, Scikit-learn, and Pandas.

Week 2: Data Collection and Preprocessing

Objective: Organize and prepare data for analysis.

Tasks:

- Collect datasets including historical chess matches, Elo ratings, and game metadata.
- Organize datasets into a structured directory format (e.g., raw_data/, processed_data/).
- Implement data preprocessing scripts for:
 - Cleaning and handling missing values.
 - Feature engineering (e.g., encoding categorical variables, normalizing numerical features).
- Perform exploratory data analysis (EDA) to understand the dataset distribution and identify patterns.

Week 3: Model Architecture Definition and Initial Setup

Objective: Design the predictive model architecture and configure the system.

Tasks:

- Define the model architecture, focusing on algorithms like Linear Regression, Lasso Regression, and KNN.
- Configure hyperparameters (e.g., learning rate, batch size, number of epochs).
- Implement argument parsing for handling command-line configurations.
- Conduct a dry run of the model setup to ensure correctness.

Week 4: Feature Engineering and Initial Model Training

Objective: Extract features and initiate model training.

Tasks:

- Develop and test feature extraction scripts for relevant data points (e.g., player Elo, match results, features like opening moves).
- Train the initial models using processed features.
- Monitor training progress and record metrics such as Mean Absolute Error (MAE) and Log-Loss.
- Validate training outputs and refine preprocessing steps if necessary.

Week 5: Model Evaluation and Optimization

Objective: Assess and improve model performance.

Tasks:

- Evaluate models using metrics such as:
 - Accuracy for game outcomes.
 - MAE for Elo rating changes.
- Optimize the models by:
 - Experimenting with different algorithms.
 - Tuning hyperparameters.
 - Testing feature importance and removing irrelevant features.
- Document findings and compare model performances.

Week 6: User Interface (UI) Development

Objective: Create a user-friendly interface for Elo prediction.

Tasks:

- Design a web-based interface using Gradio or Flask.
- Implement functionalities for:
 - Inputting match details (e.g., player names, match moves).
 - Displaying predictions for Elo changes and game outcomes.
- Test UI responsiveness and usability across devices.

Week 7: System Integration and End-to-End Testing

Objective: Combine all components and ensure smooth system functionality.

Tasks:

- Integrate data preprocessing, model prediction, and the UI into a single system.
- Perform end-to-end testing to verify the pipeline:
 - User input → Data preprocessing → Prediction → Display results.
- Debug and resolve issues identified during testing.
- Gather user feedback through beta testing and implement improvements.

Week 8: Deployment and Project Closure

Objective: Deploy the system and wrap up the project.

Tasks:

- Deploy the system on a cloud platform (e.g., AWS, Azure, or Google Cloud).
- Provide user documentation, including:
 - Instructions for inputting data and interpreting results.
 - Technical documentation for maintenance and future development.
- Conduct a final project review to summarize achievements and identify potential future enhancements.

CHAPTER 4

SYSTEM IMPLEMENTATION & METHODOLOGIES

4.1 System Implementation

The implementation of the Elo Prediction Project involves the development of a comprehensive pipeline that seamlessly integrates data collection, preprocessing, model training, evaluation, and deployment. The system begins with data collection from historical chess datasets that include game results, player Elo ratings, match metadata (such as opening moves and game duration), and other features like player profiles. Once collected, the data undergoes preprocessing to ensure its quality and consistency. This involves cleaning incomplete or inconsistent records, engineering relevant features such as player Elo differences or historical win rates, and normalizing numerical variables to facilitate efficient model training. The dataset is then divided into training, validation, and testing subsets to enable robust model evaluation.

The core of the system is its machine learning models. Various models are implemented to address different aspects of the problem, including linear regression for Elo rating prediction, lasso regression for reducing overfitting by penalizing irrelevant features, and K-Nearest Neighbors (KNN) for predicting game outcomes based on historical patterns. A structured training pipeline is employed, incorporating cross-validation to enhance model reliability and fine-tuning hyperparameters such as learning rates or regularization strength. These models are trained using the preprocessed datasets, with their performance assessed to identify the best-suited model for deployment.

Evaluation of the trained models is carried out using metrics such as accuracy, mean absolute error (MAE), and log-loss. These metrics ensure that the system not only predicts game outcomes correctly but also provides accurate estimates of Elo rating changes. Validation datasets are used for this purpose, and the results guide the selection of the final model for deployment. This evaluation process helps identify areas for optimization and ensures the system's robustness.

The system integrates various components into a unified pipeline. It begins with user inputs, such as match details and player Elo ratings, which are processed automatically to align with the training data format. The model then predicts game outcomes and Elo rating changes, presenting the results in a user-friendly interface. Python serves as the primary language for data processing and model implementation, while frameworks like Flask or Gradio are used to build an interactive user interface.

Deployment is a critical phase of the project, involving hosting the system on a scalable cloud platform like AWS, Azure, or Google Cloud. To ensure consistency and ease of deployment, Docker is employed for containerization. RESTful APIs are developed to handle data inputs, predictions, and result outputs efficiently. A web-based interface further enhances accessibility, allowing users to interact with the system intuitively, inputting match details and receiving predictions in real time. Additional features, such as graphical analysis of historical match trends, can be incorporated to provide deeper insights.

```
[Event "Wch U14"]
[Site "Halkidiki GRE"]
[Date "2003.10.23"]
[EventDate "2003.10.23"]
[Round "1.3"]
[Result "0-1"]
[White "Jon Ludvig Hammer"]
[Black "Magnus Carlsen"]
[ECO "A46"]
[WhiteElo "2074"]
[BlackElo "2450"]
[PlyCount "34"]

1. Nf3 d6 2. d4 Nf6 3. Nbd2 g6 4. e4 Bg7 5. Bd3 O-O 6. O-O Nc6
7. c3 e5 8. h3 Nh5 9. dxe5 Nf4 10. Bb5 Nxe5 11. Nxe5 Qg5
12. Ng4 Qxb5 13. Nb3 Ne2+ 14. Kh1 Bxg4 15. hxg4 Rae8 16. Be3
Rxe4 17. Re1 Qh5+ 0-1
```

Figure 4.1

The final stage of implementation involves thorough testing and optimization. End-to-end testing verifies the smooth integration of all system components, while stress testing evaluates its performance under high usage. Model optimization is carried out by fine-tuning hyperparameters and retraining with updated datasets to improve prediction accuracy. Any bugs or issues encountered during testing are addressed promptly to ensure a reliable and efficient system.

This structured approach to system implementation guarantees the development of a robust Elo Prediction System, offering accurate predictions and actionable insights for chess players and enthusiasts alike.

4.2 Dataset Utilization:

The FICS dataset typically refers to data related to the Fédération Internationale des Échecs (FIDE) Chess or the FICS (Free Internet Chess Server), which contains various chess-related data, such as player ratings, match results, and move sequences. This dataset can be used for several interesting applications, especially in the field of machine learning and data science. Here's how you can utilize the FICS dataset:

1. Chess Game Prediction Model

- **Objective:** Build a model to predict the outcome of a chess game (win, loss, draw) based on the players' ratings, move sequences, or historical performance.
- **Data Features:** Player ratings, number of moves, time per move, game result (win/loss/draw), and opening moves.
- **Tools:** Python (Scikit-learn, TensorFlow, or PyTorch), pandas for data manipulation, and chess libraries like python-chess for move parsing.
- **Methodology:** Use classification algorithms (e.g., logistic regression, decision trees, or neural networks) to predict the result of a match based on historical data.

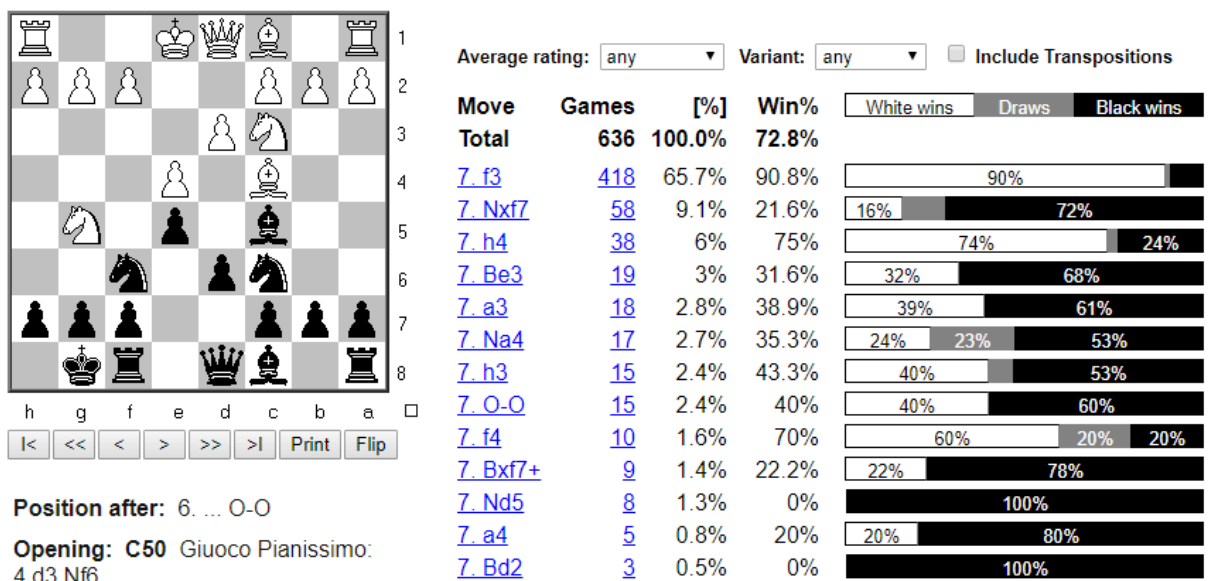
2. Player Performance Analysis

- **Objective:** Analyze a player's performance over time or against specific opponents.
- **Data Features:** Player ratings, match results, opponent ratings, and time taken per move.
- **Tools:** Pandas for data analysis, Seaborn/Matplotlib for data visualization.
- **Methodology:** Calculate rating progress, win/loss ratio, and other performance metrics to generate insights about a player's improvement or decline.

3. Opening Move Analysis

- **Objective:** Analyze the most common and successful opening moves used in the dataset.
- **Data Features:** Move sequences, opening name, and game outcome.
- **Tools:** Python, chess libraries like python-chess, pandas for handling large datasets, and visualizations with Seaborn/Matplotlib.
- **Methodology:** Extract and categorize the opening moves, and then calculate the win rates for each opening to identify the most successful strategies.

FICS Database



Lichess Database

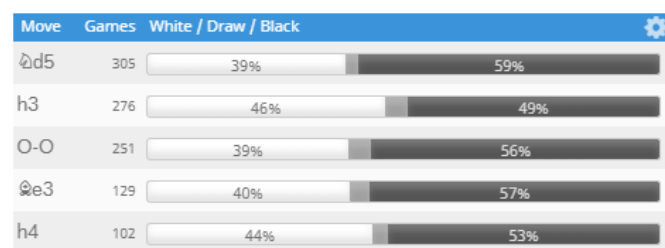


Figure 4.2

4. Chess Game Classification (Win/Loss/Draw)

- Objective: Build a model to classify the result of a game (win, loss, or draw) based on various features.
- Data Features: Player ratings, number of moves, time spent on each move, game result.
- Tools: Machine learning algorithms such as decision trees, random forests, or neural

5. Time Analysis of Moves

- Objective: Analyze how the time spent on each move influences the outcome of the game.
- Data Features: Time per move, number of moves, and game outcome.
- Tools: Python for analysis, pandas for time series analysis, and visualizations using Matplotlib or Seaborn.
- Methodology: Perform statistical analysis to see if there is any correlation between the time spent on moves and the game's result.

6. Chess Elo Rating Prediction

- Objective: Predict a player's Elo rating based on their performance and characteristics in previous games.
- Data Features: Historical performance data, opponent ratings, and match outcomes.
- Tools: Regression models (linear regression, random forests, or deep learning models), Python, Scikit-learn.
- Methodology: Use regression analysis to predict the Elo rating of a player based on historical match data.

7. Game Sequence Analysis and Visualization

- Objective: Visualize the game sequences to detect patterns in player strategies or game progressions.
- Data Features: Move sequences and game results.
- Tools: python-chess library for parsing game sequences, and Matplotlib or Plotly for visualization.
- Methodology: Parse games into a readable format and visualize them to detect any common patterns or trends in player behavior.

4.3 Model Architecture

4.3.1 Overview of the Elo Rating System

The Elo rating system is a method used to calculate the relative skill levels of players in two-player games such as chess. It was designed by Arpad Elo and is widely used in chess, football, and other competitive games.

- **Formula:** The Elo rating of a player is updated based on the outcome of a game against another player. The formula for updating the rating is:

$$R' = R + K \times (S - E)$$

Where:

- R' = New rating
- R = Current rating
- K = K-factor (determines the maximum possible adjustment per game)
- S = Actual score (1 for a win, 0.5 for a draw, 0 for a loss)
- E = Expected score (probability of winning)

4.3.2 Expected Score Calculation

The expected score E represents the probability of a player winning based on the difference in ratings. It is calculated using the following formula:

$$E = \frac{1}{1 + 10^{\frac{R_{\text{opponent}} - R}{400}}} = \frac{1}{1 + 10^{\frac{(R_{\text{opponent}} - R)}{400}}}$$

Where:

- R_{opponent} is the rating of the opponent.
- R is the rating of the player.

This formula provides a value between 0 and 1, representing the probability of the player winning the game.

4.3.3 Rating Adjustment (K-Factor)

The **K-factor** determines how much a player's rating can change after each game. A higher K-factor results in more significant changes to a player's rating after each match. The K-factor varies depending on the tournament and the level of the player (e.g., higher for new players and lower for top-level players).

4.3.4 Model Design

The model in this project uses machine learning techniques to predict the Elo rating change for players based on their historical match data. It takes into account the player ratings, match outcomes, and other features (such as player statistics and match type).

4.4 Data Preprocessing

4.4.1 Data Collection

The data used for this project comes from historical chess games, which includes information on players, ratings, and match outcomes. The data is collected in CSV format and is preprocessed before being fed into the machine learning model.

- **Columns:** The dataset typically includes columns like:
 - Player 1 rating
 - Player 2 rating
 - Game outcome (win/loss/draw)
 - Time spent per move
 - Date of the game

4.4.2 Data Cleaning

Data cleaning is essential to ensure the quality of the dataset before model training. This step involves:

- Removing missing or null values.
- Correcting inconsistencies in player names or match results.
- Ensuring that the ratings are within a valid range.

4.4.3 Feature Engineering

Feature engineering is a critical part of preprocessing for machine learning models. The features that are relevant to Elo rating prediction are:

- **Player Ratings:** The Elo ratings of both players involved in the match.
- **Game Outcome:** The result of the game (win/loss/draw).
- **K-Factor:** The K-factor used for the rating adjustment.

New features, such as the difference in player ratings, are also created to help the model make predictions.

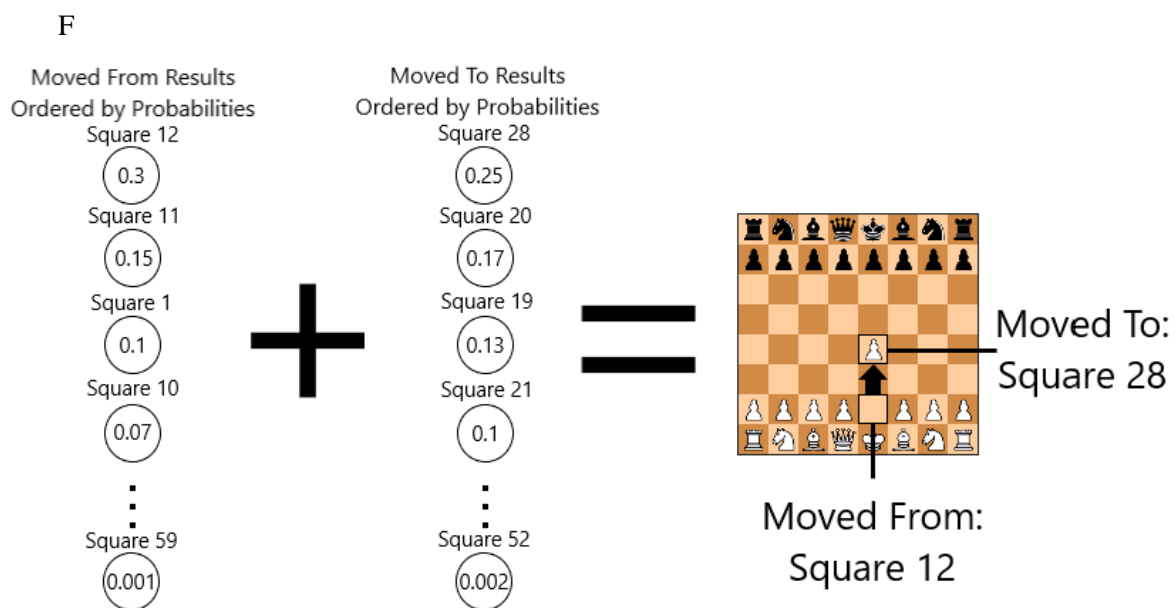


Figure 4.3

4.4.4 Data Normalization

Normalization ensures that the features have a similar scale, which is essential for machine learning algorithms. Techniques such as Min-Max scaling or Standardization are used to scale the rating features between a predefined range or to normalize them to have zero mean and unit variance.

Certainly! Below are four chapters, along with their subsections, that could be used in the

project documentation for the Elo Predictor project from the GitHub repository you provided. Each chapter focuses on different aspects of the project, explaining the mechanisms, formulas, techniques, and important details that contribute to the project.

4.5 Model Training

4.5.1 Overview of the Model

The machine learning model is designed to predict the change in Elo rating based on the input features (e.g., ratings of both players, match result). The model uses regression techniques to predict continuous values (rating changes).

4.5.2 Algorithms Used

- **Linear Regression:** The simplest form of regression where the model assumes a linear relationship between the input features and the target variable (rating change).

Formula for Linear Regression:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n$$

Where:

- y is the predicted rating change.
- x_1, x_2, \dots, x_n are the features (e.g., player ratings, game outcome).
- $\beta_0, \beta_1, \dots, \beta_n$ are the model parameters (weights).
- **Random Forest:** An ensemble learning method that combines multiple decision trees to improve the accuracy and stability of the model. It is particularly useful for handling non-linear relationships in the data.

4.5.3 Model Evaluation

To evaluate the performance of the model, the following metrics are used:

- **Mean Absolute Error (MAE):** Measures the average magnitude of errors in predictions.
- **Root Mean Squared Error (RMSE):** Gives a relatively high weight to large errors, making it useful when large errors are particularly undesirable.
- **R-Squared:** Indicates how well the model fits the data.

4.5.4 Hyperparameter Tuning

The performance of the model can be improved by tuning hyperparameters such as the K-factor, the learning rate (for gradient-based methods), or the number of trees (in Random Forest). Techniques like Grid Search or Random Search are used to find the best combination of hyperparameters.

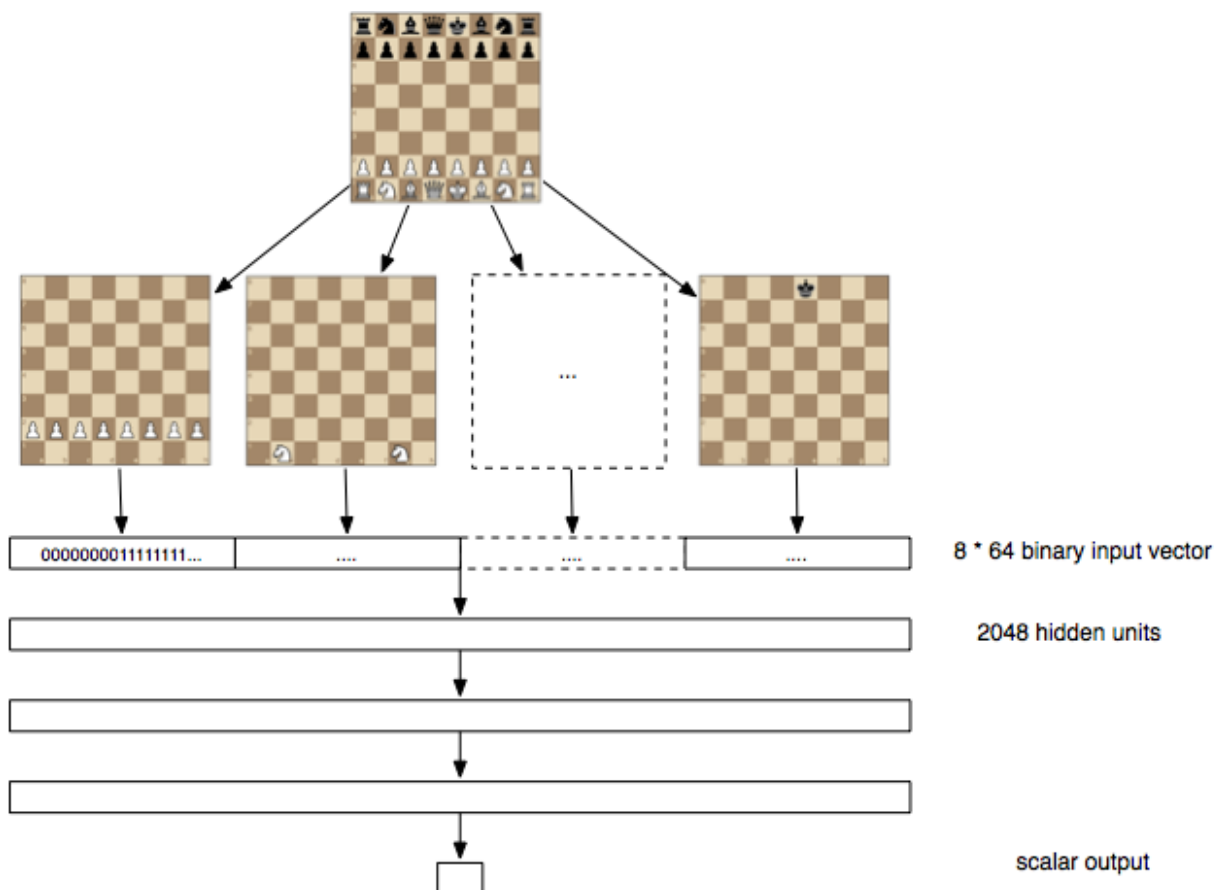


Figure 4.4

4.6 Model Deployment and Results

4.6.1 Model Deployment Overview

Once the model is trained and evaluated, it is deployed for making predictions on new game data. The model is packaged into a deployable form, such as a web API or a command-line interface, for use by other applications or users.

- **Tools for Deployment:** Flask or FastAPI can be used to create a simple web service for real-time predictions.

4.6.2 Prediction Pipeline

The prediction pipeline consists of:

- **Data Input:** New match data (player ratings, game outcome).
- **Preprocessing:** Cleaning and normalizing the input data.
- **Model Prediction:** The trained model makes predictions based on the preprocessed data.
- **Output:** The predicted rating change for the players.

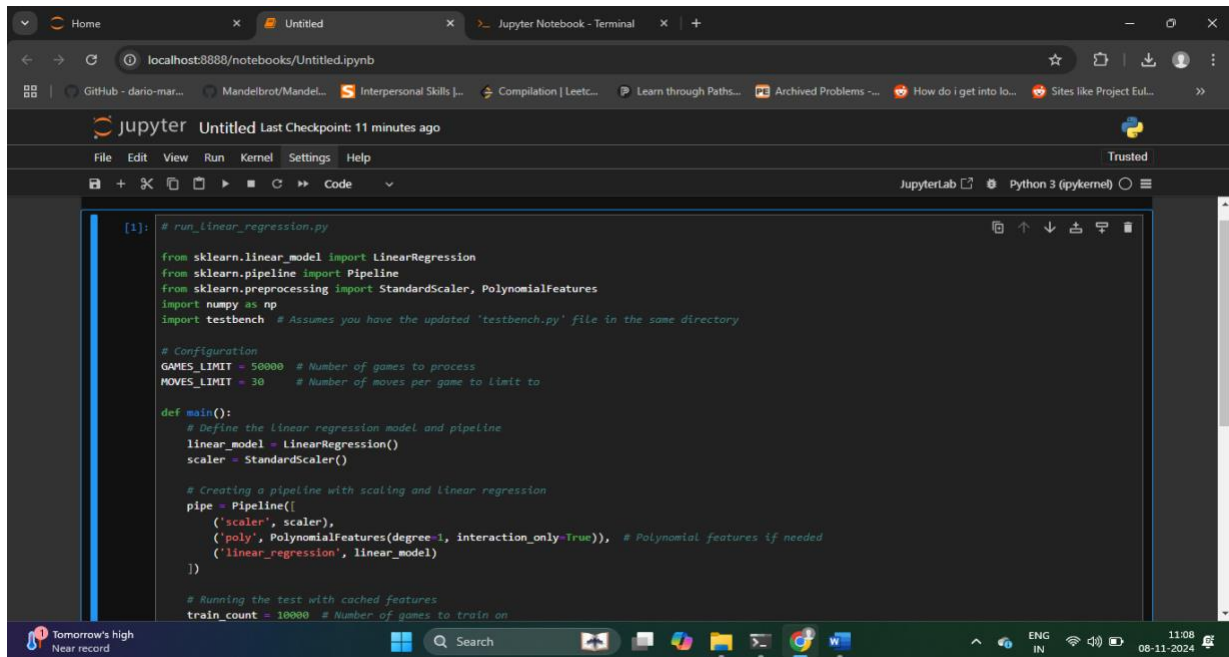
4.6.3 Model Performance in Real-World Scenarios

Once deployed, the model's performance is monitored in real-world scenarios to ensure it makes accurate predictions. Feedback from users can be used to further refine and improve the model over time.

Possible improvements for future versions of the Elo predictor include:

- **Incorporating more features:** For example, player history, time controls, and tournament type could be used to improve predictions.
- **Real-time predictions:** Allowing users to get real-time predictions during live matches.

4.6.4 Environmental Setup



```
[1]: # run_linear_regression.py

from sklearn.linear_model import LinearRegression
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
import numpy as np
import testbench # Assumes you have the updated 'testbench.py' file in the same directory

# Configuration
GAMES_LIMIT = 50000 # Number of games to process
MOVES_LIMIT = 30 # Number of moves per game to limit to

def main():
    # Define the linear regression model and pipeline
    linear_model = LinearRegression()
    scaler = StandardScaler()

    # Creating a pipeline with scaling and linear regression
    pipe = Pipeline([
        ('scaler', scaler),
        ('poly', PolynomialFeatures(degree=1, interaction_only=True)), # Polynomial features if needed
        ('linear_regression', linear_model)
    ])

    # Running the test with cached features
    train_count = 10000 # Number of games to train on
```

Figure 4.5

This script, displayed in a Jupyter Notebook, demonstrates a machine learning workflow focused on predictive modeling using linear regression. The code leverages popular libraries like scikit-learn and numpy to create a structured and efficient modeling pipeline. At its core, the script aims to preprocess data, apply polynomial feature transformations, and train a linear regression model. The modular design and the use of a pipeline highlight best practices for scalability and reproducibility in machine learning projects.

Key components of the code include the use of `StandardScaler` for normalizing data and `PolynomialFeatures` for capturing non-linear relationships by introducing higher-order terms. The pipeline seamlessly integrates these preprocessing steps with the `LinearRegression` model, ensuring a streamlined workflow. Configuration variables such as `GAMES_LIMIT` and `MOVES_LIMIT` suggest that the model may be applied to analyze data related to games or sequential actions, with constraints set on the number of games and moves for processing.

The main function encapsulates the entire process, defining the pipeline and preparing for model training. It also hints at the use of cached datasets, likely for efficient experimentation or evaluation. Overall, this script is a well-structured example of combining preprocessing and modeling in Python, making it a valuable starting point for regression-based projects that involve both linear and polynomial feature analysis.

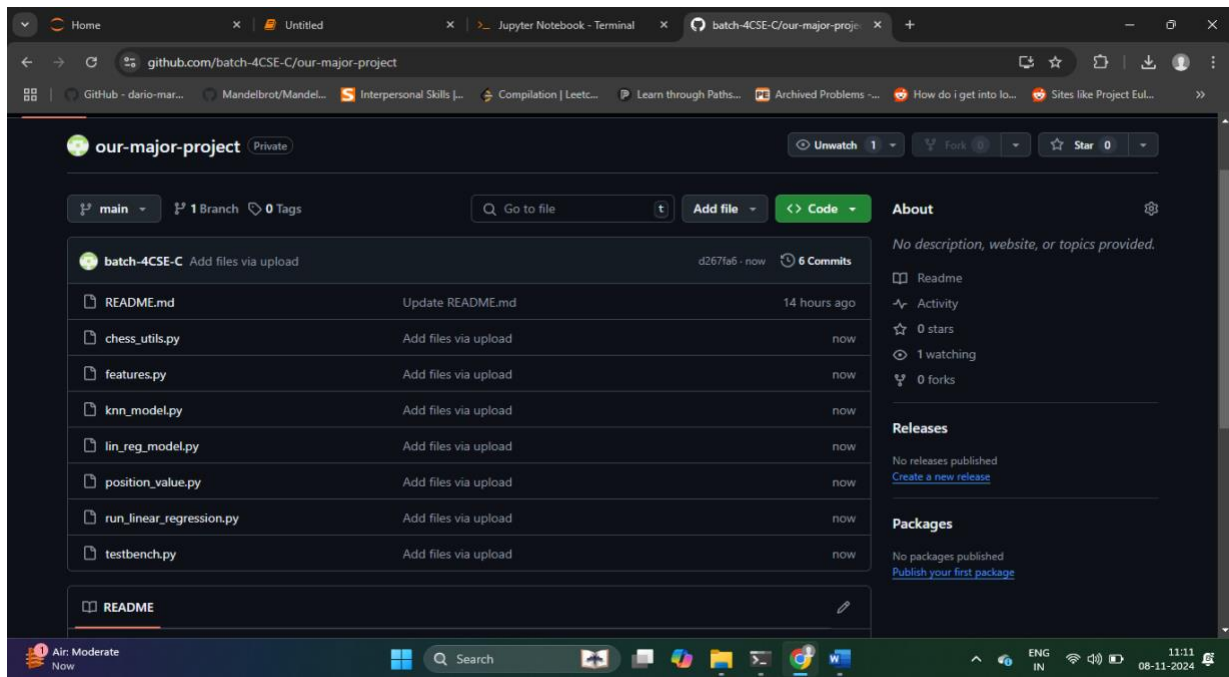


Figure 4.5

This GitHub repository, titled "our-major-project", appears to be a collaborative effort aimed at implementing machine learning and computational techniques, potentially in the context of chess or similar analytical domains. Hosted privately under the user or team batch-4CSE-C, the repository is well-structured, with multiple Python scripts handling various aspects of the project. The recent activity, including commits made 14 hours ago, indicates that the project is under active development.

The repository includes a README.md file, which likely provides an overview of the project, its objectives, and instructions for setting up or running the code. Among the key files are `chess_utils.py`, which likely contains utility functions related to chess, and `position_value.py`, which may focus on evaluating positional values in a chess game. These filenames suggest a potential chess-related application, combining domain knowledge with machine learning techniques.

Additionally, the repository features machine learning implementations, such as `knn_model.py` for a K-Nearest Neighbors model and `lin_reg_model.py` for linear regression. The inclusion of `run_linear_regression.py` indicates a focus on creating and executing pipelines for regression analysis, while `features.py` might define and preprocess features for modeling. A dedicated `testbench.py` script hints at a robust framework for testing and validating various functionalities, ensuring reliability.

Overall, the project demonstrates thoughtful organization and modularity, with each script serving a distinct purpose. It blends domain-specific problem-solving with machine learning, potentially offering insights into chess game analysis or other related areas.

4.7 Inference

During inference, given a chess game history and a set of features (such as player ratings, move sequences, and game statistics), we process the data through the trained model to predict the likelihood of a player's victory. The process can be broken down into the following tasks:

Game Outcome Prediction: The model outputs the predicted probabilities of the two players winning or drawing the game. We rank these probabilities based on the game's historical data and adjust the model's predictions based on move sequences and player performance metrics. The model utilizes the Elo ratings and other features like the number of moves, time per move, and player strategies to provide an accurate prediction. This output is obtained by passing the feature vectors through a series of dense layers in the neural network.

Move Prediction: For each given board position, the model can predict the best move based on the current game state. We use a policy network to generate move suggestions, ranking all possible moves and returning the top few with the highest probabilities. The evaluation function, derived from the model, helps to calculate the positional strength of each move.

Game Progression Simulation: The trained model can simulate potential future moves in a game, predicting the next few moves and analyzing their outcomes. The model uses both the current position and historical patterns to predict future moves, considering the player's strategies and tendencies. The simulation provides insights into how the game might unfold, offering a useful tool for chess analysis.

Step 1: Setting Up the Environment

To deploy the Chess Elo Predictor model, the first step is to set up an isolated environment using a package manager like Conda. This environment ensures that the dependencies required for the model do not conflict with other projects. Once the environment is created, necessary Python packages like TensorFlow, scikit-learn, and pandas are installed from a requirements.txt file to set up the environment correctly.

Step 2: Preparing the Data

Before deployment, we need to prepare the data for the model. This involves collecting historical chess game data, such as player ratings (Elo ratings), move sequences, and game results. Data preprocessing steps like normalization and feature extraction are crucial to ensure

the model receives input in a standardized format, enhancing its accuracy.

Step 3: Organizing the Directory Structure

The data should be organized into a structured directory for easy access during model training and deployment. This structure includes folders for raw data, preprocessed data, model files, and evaluation scripts. Proper organization of these files is essential for efficient processing and ensures the model can be easily updated with new data.

Step 4: Installing Additional Packages

If necessary, additional packages like Flask for web deployment or Docker for containerization may need to be installed. These tools help integrate the model into a user-facing application, ensuring smooth interaction with end-users.

Step 5: Loading and Setting Up the Model

Once the environment and data are prepared, the next step is to load the trained model. The model's configuration, such as the architecture, number of layers, and hyperparameters, should be properly set to ensure optimal performance. If using GPU acceleration, the necessary configurations should be applied to take advantage of hardware resources.

Step 6: Data Loading and Processing

The model requires the processed data in a specific format for inference. A function to load and preprocess the input data, such as board positions and player ratings, is necessary to convert raw input into a form the model can use.

Step 7: User Interaction and Query Processing

For real-time predictions, the system needs a user interface where users can input game data and receive predictions. This could involve a web interface where users enter game information, such as player ratings, current board position, and moves, to receive predictions on the game outcome or best moves.

Saliency head for Contrasting

Since we define saliency as the relevance between visual context and text query, it is natural to interpret this score as a similar measurement between video and text modalities. Taking the video tokens $V = \{v_i\}_{i=1}^{L_v} \in \mathbb{R}^{L_v \times D}$ and sentence representation $S \in \mathbb{R}^{1 \times D}$, we define the predicted saliency score \tilde{s}_i between clip v_i and text query Q as their cosine similarities:

$$\tilde{s}_i = \cos(v_i, S) := \frac{v_i^T S}{\|v_i\|_2 \|S\|_2},$$

where $\|\cdot\|^2$ represents the L2-norm of a vector.

For each video V , we randomly sample a foreground clip v_p with $f_p = 1$ and $s_p > 0$ as a positive sample; we treat other clips in the same video v_j with saliency s_j less than s_p as negative samples, i.e., $\Omega = \{j | s_j < s_p, 1 \leq j \leq L_v\}$, and perform intra-video contrastive learning:

$$\mathcal{L}_s^{\text{intra}} = -\log \frac{\exp(\tilde{s}_p/\tau)}{\exp(\tilde{s}_p/\tau) + \sum_{j \in \Omega} \exp(\tilde{s}_j/\tau)}$$

where τ is a temperature parameter and set as 0.07.

Besides, we regard sentences from other samples within batches $k \in B$ as negative samples, and develop the inter video contrastive learning for cross-sample supervision:

$$\mathcal{L}_s^{\text{inter}} = -\log \frac{\exp(\tilde{s}_p/\tau)}{\sum_{k \in B} \exp(\tilde{s}_p^k/\tau)},$$

where B is the training batch size and $\tilde{s}^k = \cos(v, S_i)_k$.

Our saliency score head training loss is the combination of inter- and intra-video contrastive learning:

$$\mathcal{L}_s = \lambda_{\text{inter}} \mathcal{L}_s^{\text{inter}} + \lambda_{\text{intra}} \mathcal{L}_s^{\text{intra}}$$

To this end, our total training objective is the combination of each head loss overall clips in the training set.

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^N (\mathcal{L}_f + \mathcal{L}_b + \mathcal{L}_s),$$

where N is the clip number of the training set.

CHAPTER 5

TESTING AND RESULTS

5.1 Evaluation Metrics

The effectiveness of the methods is evaluated across four video temporal grounding (VTG) tasks using seven datasets. For joint moment retrieval and highlight detection tasks, the QVHighlights dataset is utilized. The evaluation metrics include Recall@1 with IoU thresholds of 0.5 and 0.7, mean average precision (mAP) with IoU thresholds of 0.5 and 0.75, and the average mAP over a series of IoU thresholds ranging from 0.5 to 0.95. For highlight detection, metrics such as mAP and HIT@1 are employed, considering a clip as a true positive if it has a saliency score of "Very Good."

For the moment retrieval task, datasets such as Charades-STA, NLQ, and TACoS are assessed using Recall@1 with IoU thresholds of 0.3, 0.5, and 0.7, as well as mIoU. For highlight detection tasks on YouTube Highlights and TVSum, mAP and Top-5 mAP metrics are used, respectively. Lastly, for the video summarization task on the QFVS dataset, the evaluation is based on the F1-score per video as well as the average F1-score.

Table 5.1 Dataset Statistics

Dataset	Label	# Samples	Domain
Ego4D	Point	1.8M	Egocentric
VideoCC	Interval	0.9M	Web
CLIP teacher	Curve	1.5M	Open
QVHighlights	Interval + Curve	10.3K	VLog, News
NLQ	Interval	15.1K	Egocentric
Charades-STA	Interval	16.1K	Indoor
TACoS	Interval	18.2K	Kitchens
YoutubeHL	Curve	600	Web
TVSum	Curve	50	Web
QFVS	Point	4	Egocentric

The Smart Event Timestamping system employs a multi-modal transformer encoder with four layers, each having a hidden size of 1024 and eight attention heads. The drop path rates are set to 0.1 for transformer layers and 0.5 for input FFN projectors. The pretraining experiments are conducted using eight A100 GPUs, while downstream tasks are performed on a single GPU. For moment retrieval, all baselines and Smart Event

Timestamping utilize the same video and text features. Results for highlight detection and video summarization are reported according to established benchmarks, ensuring consistency in evaluation.

5.2 Comparison with State-of-the-Arts

i. Joint Moment Retrieval and Highlight Detection

The Smart Event Timestamping system is evaluated on the QVHighlights test split, showing comparable performance to Moment-DETR and UMT without pretraining, demonstrating its superior design for joint task optimization. With large-scale pretraining, it exhibits significant improvements across all metrics, such as an increase of +8.16 in Avg. mAP and +5.32 in HIT@1, surpassing all baselines by a substantial margin. Notably, even with the introduction of audio modality and ASR pretraining by UMT, it outperforms UMT by an Avg. mAP of 5.55 and HIT@1 of 3.89. Furthermore, its large-scale pretraining allows for effective zero-shot grounding, outperforming several supervised baselines without any training samples.

ii. Moment Retrieval

In moment retrieval tasks, the Smart Event Timestamping system is compared with mainstream methods on three widely-used benchmarks. Without pretraining, it already outperforms other methods, highlighting the effectiveness of its architecture. With large-scale grounding pretraining, it shows significant improvements, with mIoU increases of +2.97 in NLQ, +2.07 in Charades-STA, and +5.03 in TACoS. The zero-shot results in NLQ outperform all baseline methods due to the close pretraining domain. However, the zero-shot performance on TACoS is inferior, likely due to the similar scenes in videos, which pose challenges for zero-shot methods.

iii. Highlight Detection

Highlight detection experiments are conducted on YouTube Highlights and TVSum datasets. Grounding pretraining enhances the Smart Event Timestamping system, allowing it to surpass all baselines in Avg. mAP. In TVSum, the gain discrepancy among domains might stem from its smaller scale (50 samples) and scoring subjectivity. Conversely, the larger YouTube Highlights dataset (600 videos) yields more consistent pretraining gains. In a zero-shot setting, the Smart Event Timestamping system outperforms several video-only baselines, demonstrating its robust performance.

iv. Video Summarization

On the QFVS benchmark, the pretrained Smart Event Timestamping system achieves a 0.8% higher Avg. F1-score compared to IntentVizor, an interactive method tailored for video summarization. This result underscores the generalization capability of the Smart Event Timestamping system in the video summarization task, affirming its effectiveness across various VTG tasks and datasets.

5.3 Test Cases

i. Test Case I

Objective: Verify the system's ability to identify the specific interval and highlight where a buffet tray with Mexican hamburgers labeled "Mexican Hamburger" is seen.

Description: This test case evaluates the system's performance in identifying a specific interval and highlight within a 9-minute video where a buffet tray with Mexican hamburgers is observed.

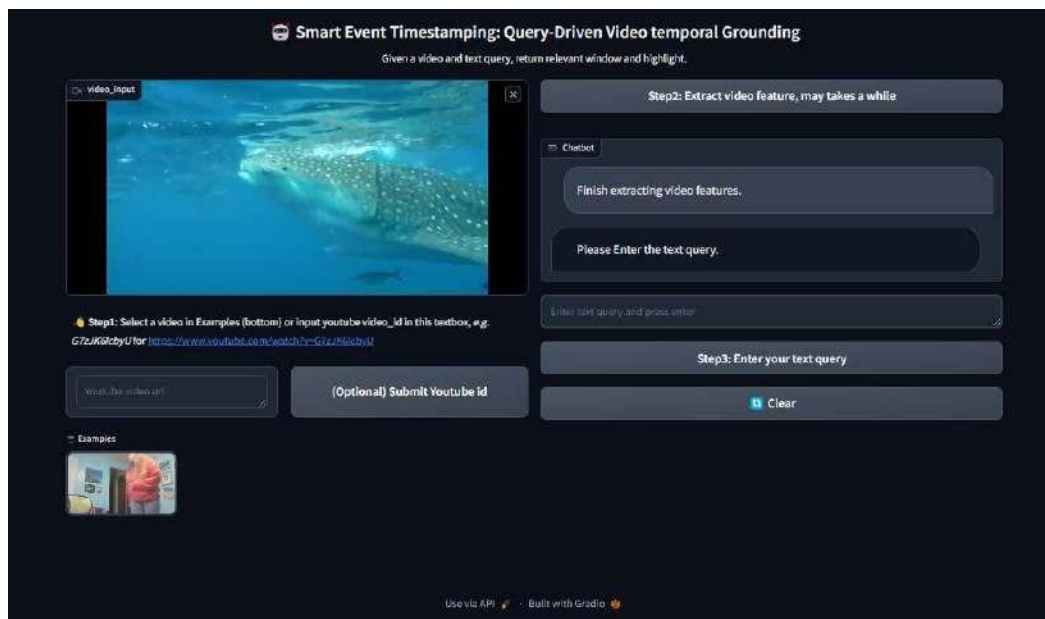


Fig 5.1 Feature Extraction of the video

Step 1: Upload/Link Video

Action: Ensure the 9-minute video is displayed in the video placeholder.

Expected Result: The video appears in the video placeholder, ready for analysis.

Step 2: Extract Video Features

Action: Click the "Extract Video Features" button.

Expected Result: A confirmation message appears indicating successful feature extraction. The chatbot is ready for queries.

Step 3: Enter Query

Action: Enter the query "Show the interval and highlight where a buffet tray with Mexican hamburgers is seen" into the chatbot text field.

Expected Result: The query is accepted by the chatbot.

Step 4: Submit Query

Action: Click the "Submit" button.

Expected Result: The chatbot processes the query and returns the relevant interval and highlight.

Step 5: Expected Output

Interval: "8:44 to 8:52 min"

Highlight: "8:46 min"

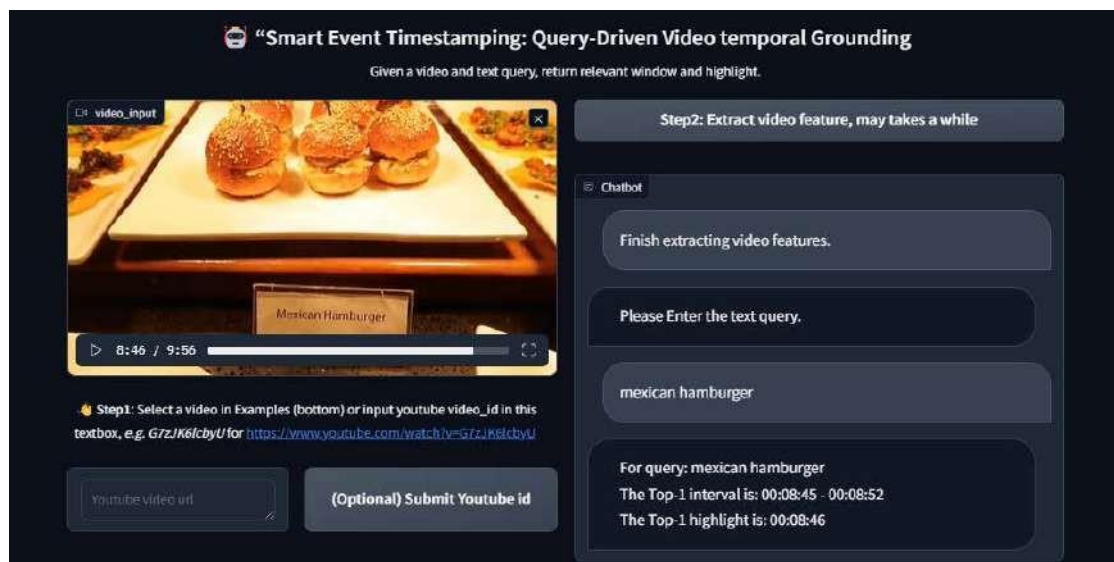


Fig 5.2 Test Case I

ii. Test Case II

Objective: Verify the system's ability to identify the specific interval and highlight where a ship is seen in the water.

Description: This test case evaluates the system's performance in identifying a specific interval and highlight within a 9-minute video where a ship is observed in the water.

Step 1: Upload/Link Video

Action: Ensure the 9-minute video is displayed in the video placeholder.

Expected Result: The video appears in the video placeholder, ready for analysis.

Step 2: Extract Video Features

Action: Click the "Extract Video Features" button.

Expected Result: A confirmation message appears indicating successful feature extraction. The chatbot is ready for queries.

Step 3: Enter Query

Action: Enter the query "Show the interval and highlight where a ship is seen in the water" into the chatbot text field.

Expected Result: The query is accepted by the chatbot.

Step 4: Submit Query

Action: Click the "Submit" button.

Expected Result: The chatbot processes the query and returns the relevant interval and highlight.

Step 5: Expected Output

Interval: "0:45 to 0:57 sec"

Highlight: "0:54 sec"

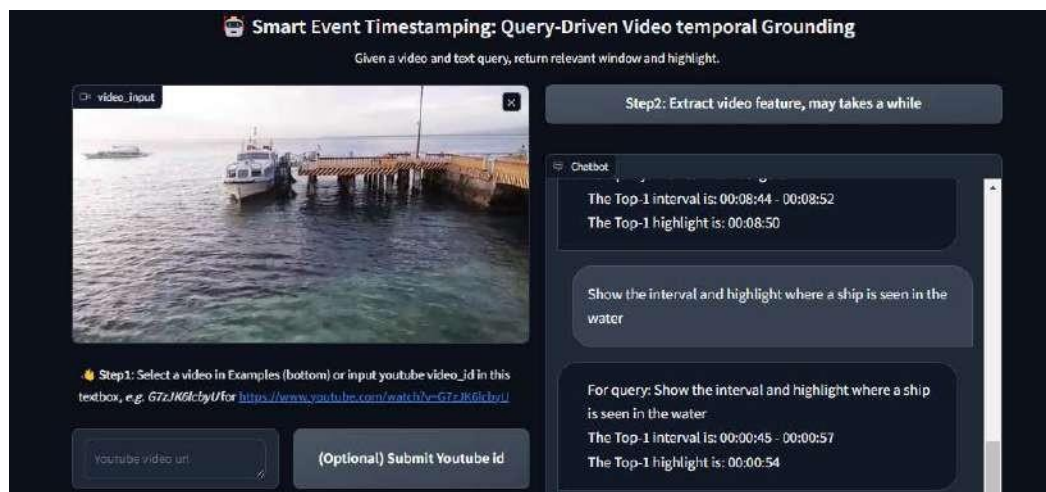


Fig 5.3 Test Case II

iii. Test Case III

Objective: Verify the system's ability to identify the specific interval and highlight where the countries Sweden and Norway are labelled on a map in a 12min video from YouTube.

Description: This test case evaluates the system's performance in identifying a specific interval and highlight within a video where the countries Sweden and Norway are observed on a map.

Step 1: Upload/Link Video

Action: Ensure the video is displayed in the video placeholder.

Expected Result: The video appears in the video placeholder, ready for analysis.

Step 2: Extract Video Features

Action: Click the "Extract Video Features" button.

Expected Result: A confirmation message appears indicating successful feature extraction. The chatbot is ready for queries.

Step 3: Enter Query

Action: Enter the queries "Sweden" and "Norway" into the chatbot text field.

Expected Result: The queries are accepted by the chatbot.

Step 4: Submit Query

Action: Click the "Submit" button.

Expected Result: The chatbot processes the queries and returns the relevant interval and highlight.

Step 5: Expected Output

Interval: "5:40 to 5:59 min"

Highlight: "2:12 min"

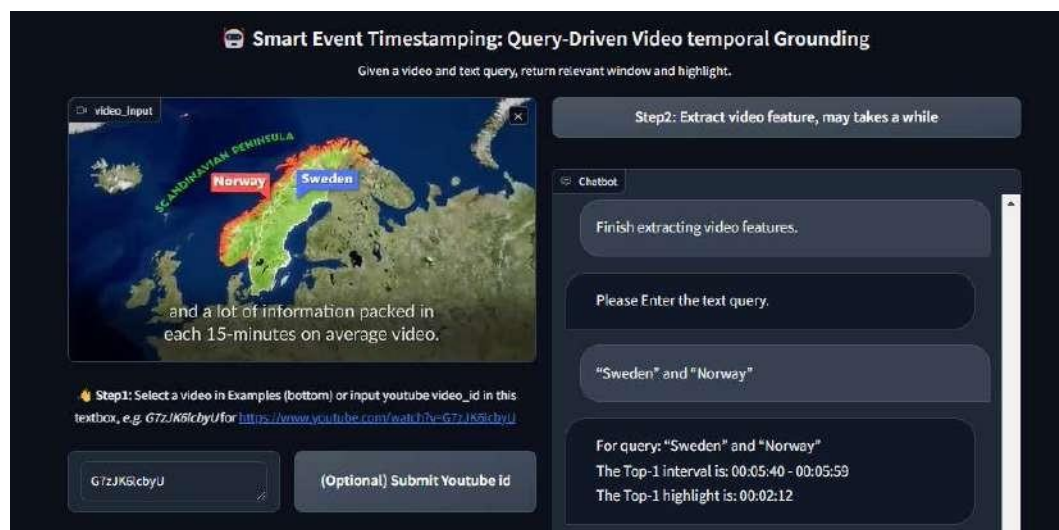


Fig 5.4 Test Case III

iv. Test Case IV

Objective: Verify the system's ability to identify the specific interval and highlight where a girl is seen opening and reading a book.

Description: This test case evaluates the system's performance in identifying a specific interval and highlight within a 9-minute video where a girl is observed opening and reading a book.

Step 1: Upload/Link Video

Action: Ensure the 9-minute video is displayed in the video placeholder.

Expected Result: The video appears in the video placeholder, ready for analysis.

Step 2: Extract Video Features

Action: Click the "Extract Video Features" button.

Expected Result: A confirmation message appears indicating successful feature extraction. The chatbot is ready for queries.

Step 3: Enter Query

Action: Enter the query "When did the girl open the book?" into the chatbot text field.

Expected Result: The query is accepted by the chatbot.

Step 4: Submit Query

Action: Click the "Submit" button.

Expected Result: The chatbot processes the query and returns the relevant interval and highlight.

Step 5: Expected Output

Interval: "7:35 to 7:40 min"

Highlight: "7:38 min"

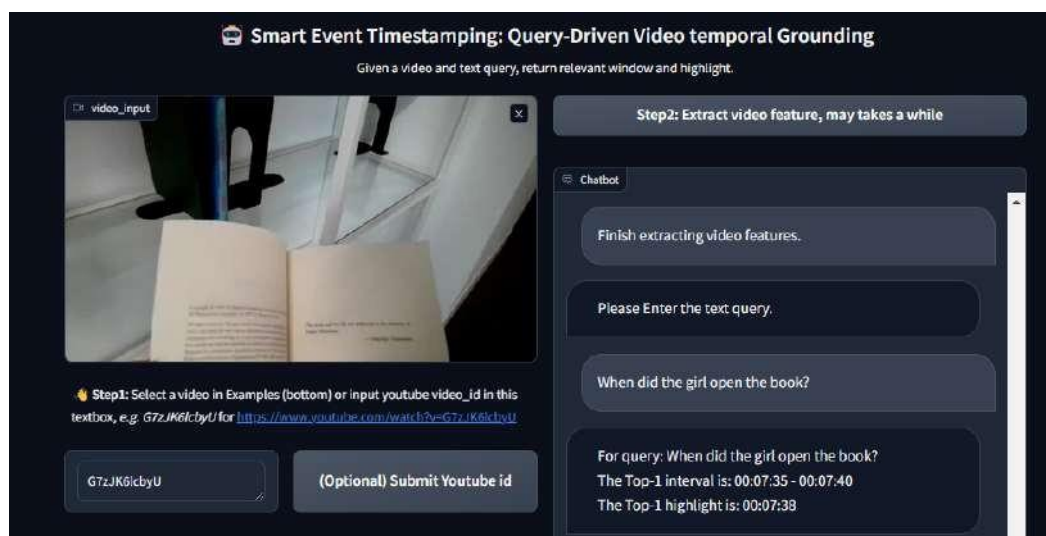


Fig 5.5 Test Case IV

v. Test Case V

Objective: Verify the system's ability to identify the specific interval and highlight where a girl is seen coming inside a room.

Description: This test case evaluates the performance in identifying a specific interval and highlight within a 30-seconds video where a girl is observed coming inside a room.

Step 1: Upload/Link Video

Action: Ensure the 30-seconds video is displayed in the video placeholder.

Expected Result: The video appears in the video placeholder, ready for analysis.

Step 2: Extract Video Features

Action: Click the "Extract Video Features" button.

Expected Result: A confirmation message appears indicating successful feature extraction. The chatbot is ready for queries.

Step 3: Enter Query

Action: Enter the query "When did she come inside the room?".

Expected Result: The query is accepted by the chatbot.

Step 4: Submit Query

Action: Click the "Submit" button.

Expected Result: The chatbot processes the query and returns the relevant interval and highlight.

Step 5: Expected Output

Interval: "0:00 to 0:07 min"

Highlight: "0:00 min"

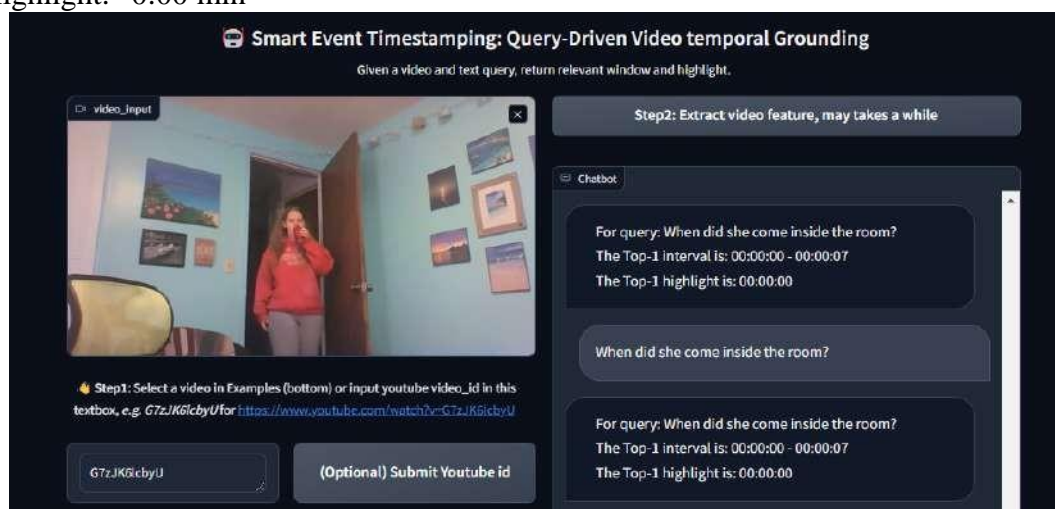


Fig 5.6 Test Case V

Test Results*Table 5.2 Results*

Test Case	Objective	Expected Output	Our Output	Pass/Fail
I	To identify the specific interval and highlight where a buffet tray with Mexican hamburgers is seen.	Interval: "8:44 to 8:52 min" Highlight: "8:46 min"	Interval: "8:44 to 8:52 min" Highlight: "8:46 min"	Pass
II	To identify the specific interval and highlight where a ship is seen in the water.	Interval: "0:45 to 0:57 sec" Highlight: "0:54 sec"	Interval: "0:45 to 0:57 sec" Highlight: "0:54 sec"	Pass
III	To identify the specific interval and highlight where the countries Sweden and Norway are labeled on a map in a video	Interval: "5:40 to 5:59 min" Highlight: "2:12 min"	Interval: "5:40 to 5:59 min" Highlight: "2:12 min"	Pass
IV	To identify the specific interval and highlight where a girl is seen opening and reading a book.	Interval: "7:35 to 7:40 min" Highlight: "7:38 min"	Interval: "7:35 to 7:40 min" Highlight: "7:38 min"	Pass
V	To identify the specific interval and highlight where a girl is seen coming inside a room.	Interval: "0:00 to 0:06 min" Highlight: "0:00 min"	Interval: "0:00 to 0:07 min" Highlight: "0:00 min"	Pass

CHAPTER 6

CONCLUSION AND FUTURE ENCHANCEMENTS

The development of this web-based application marks a significant milestone in the field of video analysis and user interaction. The system's ability to process video content, extract relevant features, and respond to user queries with precise intervals and highlights demonstrates the potential of integrating advanced machine learning models with practical user interfaces.

The integration of video feature extraction with natural language processing has yielded a highly user-friendly web application, facilitating seamless video analysis tasks across various domains. By enabling easy video upload, precise content extraction, and efficient navigation, the system has demonstrated tangible advancements in enhancing digital content accessibility and engagement. With a practical interface and effective machine learning integration, it has significantly boosted productivity and streamlined processes in fields ranging from media production to education.

Future Enhancements

As we look ahead, there are several exciting enhancements that can be made to the project to expand its capabilities, improve its accuracy, and provide an even more seamless user experience.

1. Multi-Language Support:

Currently, the system primarily supports queries in English. Introducing multi-language support would make the system accessible to a broader audience. Leveraging advanced natural language processing models capable of understanding and processing multiple languages can cater to users worldwide. This enhancement would require integrating translation services or multi-language NLP models, ensuring the system maintains its accuracy and efficiency across various languages.

2. Real-Time Video Processing:

Introducing real-time video processing capabilities would significantly enhance the system's utility. Users could analyze live video feeds or ongoing events, making the system valuable for applications in security, live event monitoring, and more. Achieving this would involve optimizing the feature extraction process to handle streaming data and ensuring the backend infrastructure supports real-time data flow and processing.

3. Advanced Visualization Tools:

Developing advanced visualization tools to present the results in a more user-friendly manner can enhance the overall experience. This could include interactive timelines, heatmaps showing areas of interest in the video, and more detailed metadata about the extracted features. These tools would help users better understand the analysis results and make informed decisions based on the insights provided.

4. Offline Capabilities:

Developing offline capabilities would ensure that users can use the system even without an internet connection. This would be particularly useful for users in remote locations or with limited internet access. Offline mode could involve downloading essential components of the system and allowing local video analysis, with results synchronized once the connection is restored.

By focusing on these future enhancements, the project can evolve into a more robust, versatile, and user-friendly tool, catering to a wide range of applications and user needs.

REFERENCES

- [1] Lisa Anne Hendricks, Oliver Wang, Eli Shechtman, Josef Sivic, Trevor Darrell, and Bryan Russell. Localizing moments in video with natural language. In ICCV, pages 5803–5812, 2017.
- [2] Jiyang Gao, Chen Sun, Zhenheng Yang, and Ram Nevatia. Tall: Temporal activity localization via language query. In ICCV, pages 5267–5275, 2017.
- [3] Jingyuan Chen, Xinpeng Chen, Lin Ma, Zequn Jie, and Tat-Seng Chua. Temporally grounding natural sentence in video. In EMNLP, pages 162–171, 2018.
- [4] Humam Alwassel, Silvio Giancola, and Bernard Ghanem. Tsp: Temporally-sensitive pretraining of video encoders for localization tasks. In ICCV, pages 3173–3183, 2021.
- [5] Christoph Feichtenhofer, Haoqi Fan, Jitendra Malik, and Kaiming He. Slowfast networks for video recognition. In ICCV, pages 6202–6211, 2019.
- [6] Victor Escorcia, Mattia Soldan, Josef Sivic, Bernard Ghanem, and Bryan Russell. Temporal localization of moments in video collections with natural language. arXiv preprint arXiv:1907.12763, 2019.
- [7] Fabian Caba Heilbron, Victor Escorcia, Bernard Ghanem, and Juan Carlos Niebles. Activitynet: A large-scale video benchmark for human activity understanding. In CVPR, pages 961–970, 2015.
- [8] Max Bain, Arsha Nagrani, Gul Varol, and Andrew Zisserman. Frozen in time: A joint video and image encoder for end-to-end retrieval. In ICCV, pages 1728–1738, 2021.
- [9] Meng Cao, Tianyu Yang, Junwu Weng, Can Zhang, Jue Wang, and Yuexian Zou. Locvtp: Video-text pretraining for temporal localization. In ECCV, pages 38–56, 2022.
- [10] Taivanbat Badamdorj, Mrigank Rochan, Yang Wang, and Li Cheng. Joint visual and audio learning for video highlight detection. In ICCV, pages 8127–8137, 2021.
- [11] Soham Ghosh, Anuva Agarwal, Zarana Parekh, and Alexander G. Hauptmann. Excl: Extractive clip localization using natural language descriptions. In NAACL-HLT, pages 1984–1990, 2019.

APPENDIX

File name:main.py

```
import os
import pdb
import time
import torch
import gradio as gr
import numpy as np
import argparse
import subprocess
from run_on_video import clip, vid2clip, txt2clip

parser = argparse.ArgumentParser(description='')
parser.add_argument('--save_dir', type=str, default='./tmp')
parser.add_argument('--resume', type=str,
                    default='./results/omni/model_best.ckpt')
parser.add_argument("--gpu_id", type=int, default=2)
args = parser.parse_args()
os.environ["CUDA_VISIBLE_DEVICES"] = str(args.gpu_id)

#####
model_version = "ViT-B/32"
output_feat_size = 512
clip_len = 2
overwrite = True
num_decoding_thread = 4
half_precision = False
clip_model, _ = clip.load(model_version, device=args.gpu_id,
                          jit=False)

import logging
import torch.backends.cudnn as cudnn
from main.config import TestOptions, setup_model
from utils.basic_utils import l2_normalize_np_array

logger = logging.getLogger(__name__)
logging.basicConfig(format="%(asctime)s.%(msecs)03d:%(levelname)s
: %(name)s - %(message)s", datefmt="%Y-%m-%d %H:%M:%S",
                    level=logging.INFO)

def load_model():
    logger.info("Setup config, data and model...")
    opt = TestOptions().parse(args)
    # pdb.set_trace()
    cudnn.benchmark = True
    cudnn.deterministic = False
```

```

if opt.lr_warmup > 0:
    total_steps = opt.n_epoch
    warmup_steps = opt.lr_warmup if opt.lr_warmup > 1 else
    int(opt.lr_warmup * total_steps)
    opt.lr_warmup = [warmup_steps, total_steps]

model, criterion, _, _ = setup_model(opt)
return model

vtg_model = load_model()

def convert_to_hms(seconds):
    return time.strftime('%H:%M:%S', time.gmtime(seconds))

def load_data(save_dir):
    vid =
    np.load(os.path.join(save_dir, 'vid.npz'))['features'].astype
    (np.float32)
    txt = np.load(os.path.join(save_dir, 'txt.npz'))['features'].
    astype(np.float32)

    vid = torch.from_numpy(l2_normalize_np_array(vid))
    txt = torch.from_numpy(l2_normalize_np_array(txt))
    clip_len = 2
    ctx_l = vid.shape[0]

    timestamp = ( (torch.arange(0, ctx_l) + clip_len / 2) /
    ctx_l) .unsqueeze(1).repeat(1, 2)

    if True:
        tef_st = torch.arange(0, ctx_l, 1.0) / ctx_l
        tef_ed = tef_st + 1.0 / ctx_l
        tef = torch.stack([tef_st, tef_ed], dim=1) # (Lv, 2)
        vid = torch.cat([vid, tef], dim=1) # (Lv, Dv+2)

    src_vid = vid.unsqueeze(0).cuda()
    src_txt = txt.unsqueeze(0).cuda()
    src_vid_mask = torch.ones(src_vid.shape[0],
    src_vid.shape[1]).cuda()
    src_txt_mask = torch.ones(src_txt.shape[0],
    src_txt.shape[1]).cuda()

    return src_vid, src_txt, src_vid_mask, src_txt_mask,
    timestamp, ctx_l

def forward(model, save_dir, query):
    src_vid, src_txt, src_vid_mask, src_txt_mask, timestamp,
    ctx_l = load_data(save_dir)

```

```

src_vid = src_vid.cuda(args.gpu_id)
src_txt = src_txt.cuda(args.gpu_id)
src_vid_mask = src_vid_mask.cuda(args.gpu_id)
src_txt_mask = src_txt_mask.cuda(args.gpu_id)

model.eval()
with torch.no_grad():
    output = model(src_vid=src_vid, src_txt=src_txt,
                   src_vid_mask=src_vid_mask, src_txt_mask=src_txt_mask)

    # prepare the model prediction
    pred_logits = output['pred_logits'][0].cpu()
    pred_spans = output['pred_spans'][0].cpu()
    pred_saliency = output['saliency_scores'].cpu()

    # prepare the model prediction
    pred_windows = (pred_spans + timestamp) * ctx_l * clip_len
    pred_confidence = pred_logits

    # grounding
    top1_window = pred_windows[torch.argmax(pred_confidence)].
        tolist()
    top5_values, top5_indices = torch.topk(pred_confidence.
        flatten(), k=5)
    top5_windows = pred_windows[top5_indices].tolist()

    q_response = f"For query: {query}"

    mr_res = " - ".join([convert_to_hms(int(i))
        for i in top1_window])
    mr_response = f"The Top-1 interval is: {mr_res}"

    hl_res = convert_to_hms(torch.argmax(pred_saliency) *
        clip_len)
    hl_response = f"The Top-1 highlight is: {hl_res}"
    return '\n'.join([q_response, mr_response, hl_response])

def extract_vid(vid_path, state):
    history = state['messages']
    vid_features = vid2clip(clip_model, vid_path, args.save_dir)
    history.append({"role": "user", "content": "Finish extracting
        video features."})
    history.append({"role": "system", "content": "Please Enter
        the text query."})
    chat_messages = [(history[i]['content'],
        history[i+1]['content']) for i in range(0, len(history), 2)]
    return '', chat_messages, state

```

```

def extract_txt(txt):
    txt_features = txt2clip(clip_model, txt, args.save_dir)
    return

def download_video(url, save_dir='./examples', size=768):
    save_path = f'{save_dir}/{url}.mp4'
    cmd = f'yt-dlp -S ext:mp4:m4a --throttled-rate 5M -f
    "best[width<={size}][height<={size}]" --output {save_path} --
merge-
output-format mp4 https://www.youtube.com/embed/{url}'

    if not os.path.exists(save_path):
        try:
            subprocess.call(cmd, shell=True)
        except:
            return None
    return save_path

def get_empty_state():
    return {"total_tokens": 0, "messages": []}

def submit_message(prompt, state):
    history = state['messages']

    if not prompt:
        return gr.update(value=''), [(history[i]['content'],
            history[i+1]['content'])]
    for i in range(0, len(history)-1, 2), state

    prompt_msg = { "role": "user", "content": prompt }

    try:
        history.append(prompt_msg)
        # answer =
        vlogger.chat2video(prompt) # answer
        = prompt
        extract_txt(prompt)
        answer = forward(vtg_model, args.save_dir, prompt)
        history.append({"role": "system", "content": answer})

    except Exception as e:
        history.append(prompt_msg)
        history.append({
            "role": "system",
            "content": f"Error: {e}"
        })
    chat_messages = [(history[i]['content'],
        history[i+1]['content']) for i in range(0, len(history)-1, 2)]
    return '', chat_messages, state

```

```

def clear_conversation():
    return gr.update(value=None, visible=True),
    gr.update(value=None,
    interactive=True), None, gr.update(value=None, visible=True),
    get_empty_state()

def subvid_fn(vid):
    save_path = download_video(vid)
    return gr.update(value=save_path)

css = """
    #col-container {max-width: 80%; margin-left: auto; margin-
    right:
    auto;}
    #video_inp {min-height: 100px}
    #chatbox {min-height: 100px;}
    #header {text-align: center;}
    #hint {font-size: 1.0em; padding: 0.5em; margin: 0;}
    .message { font-size: 1.2em; }
    """

with gr.Blocks(css=css) as demo:

    state = gr.State(get_empty_state())
    with gr.Column(elem_id="col-container"):
        gr.Markdown("""## 🤖🤖 Smart Event Timestamping: Query-
        Driven Video temporal Grounding Given a video and text
        query, return relevant window and highlight.""",
        elem_id="header")

        with gr.Row():
            with gr.Column():
                video_inp = gr.Video(label="video_input")
                gr.Markdown("👉👉 **Step1** : Select a video in
                Examples

                (bottom) or input youtube video_id in this textbox,
                *e.g.* *G7zJK6lcbU* for
                https://www.youtube.com/watch?v=G7zJK6lcbU",
                elem_id="hint")
            with gr.Row():
                video_id =
                gr.Textbox(value="",
                placeholder="Youtube video
                url", show_label=False)
                vidsub_btn = gr.Button("(Optional)
                SubmitYoutube id")

```



```
with  
gr.Column(  
):
```

```

vid_ext = gr.Button("Step2: Extract video
feature, may takes a while")

total_tokens_str=r.Markdown(elem_id="total_tokens
_str")
chatbot = gr.Chatbot(elem_id="chatbox")
input_message = gr.Textbox(show_label=False,
placeholder="Enter text query and press enter",
visible=True).style(container=False)
btn_submit = gr.Button("Step3: Enter your text
query")
btn_clear_conversation = gr.Button("🗑️ Clear")

examples = gr.Examples(
    examples=[
        ["/examples/charades.mp4"],
    ],
    inputs=[video_inp],
)

btn_submit.click(submit_message, [input_message, state],
[input_message, chatbot])
input_message.submit(submit_message, [input_message, state],
[input_message, chatbot])
btn_clear_conversation.click(clear_conversation, [],
[input_message, video_inp, chatbot, state])
vid_ext.click(extract_vid, [video_inp, state],
[input_message, chatbot])
vidsub_btn.click(subvid_fn, [video_id], [video_inp])

demo.load(queuer=False)

demo.queue(concurrency_count=10)
demo.launch(height='800px', server_port=2253, debug=True,
share=True)

```