

Assignment cover sheet

Please complete this cover sheet for each assignment submission. For group assignments, please ensure that each group member completes and submits an individual cover sheet.

Module:	SOC10101 Honours Project
Assessment Title:	Dissertation
Module Leader:	Khristin Fabian
Student registration number:	40679914
Student name	Ruben Lazell

Declaration

I declare, in accordance with [Edinburgh Napier University's Academic Integrity Regulations](#) that: except where explicit reference is made to the contribution of others*, this assignment is the result of my own work, and has not been submitted for any module, programme or degree at Edinburgh Napier University or any other institution.

*IMPORTANT: Contribution of includes use of generative Artificial Intelligence (AI) tools. Ensure you have read the University [Guidelines for Students on AI & Writing Assistant Tools](#)). Please declare here whether you have used such tools, and to what extent:

☒ NO I have not used such tools

☐ YES I have used such tools and I have provided details and included sample prompts and responses below <or: in an appendix>.

If you have used AI tools in completing this submission, please briefly describe in approximately 100 words in the box below how you have used these tools:

--

Detecting Cheating in Online Chess: A Data-Driven Approach
to Identifying Engine Assistance in Titled Tuesday
Tournaments

Ruben Lazell

Submitted in partial fulfilment of
the requirements of Edinburgh Napier University
for the Degree of
Data Science (BSc)

School of Computing, Engineering & The Built Environment

April 2025

Authorship Declaration

I, Ruben Pither Lazell, confirm that this dissertation and the work presented in it are my own achievement.

Where I have consulted the published work of others this is always clearly attributed;

Where I have quoted from the work of others the source is always given. With the exception of such quotations this dissertation is entirely my own work;

I have acknowledged all main sources of help;

If my research follows on from previous work or is part of a larger collaborative research project I have made clear exactly what was done by others and what I have contributed myself;

I have read and understand the penalties associated with Academic Misconduct.

I also confirm that I have obtained **informed consent** from all people I have involved in the work in this dissertation following the School's ethical guidelines

Signed: Ruben Lazell

Date: 08/04/2025

Matriculation no: 40679914

General Data Protection Regulation Declaration

Under the General Data Protection Regulation (GDPR) (EU) 2016/679, the University cannot disclose your grade to an unauthorised person. However, other students benefit from studying dissertations that have their grades attached.

Please sign your name below *one* of the options below to state your preference.

The University may make this dissertation, with indicative grade, available to others.

Ruben Lazell

The University may make this dissertation available to others, but the grade may not be disclosed.

The University may not make this dissertation available to others.

Abstract

With over-the-board cheating scandals being mostly circumstantial and online cheating incidents being commonly dismissed because of anonymity protection or lack of impact, this project identifies a gap in the current literature surrounding cheating in chess. Specifically, it aims to identify players who have used engine-assistance in Titled Tuesday games played between October 28, 2014, and September 24, 2024 – a weekly high-stakes event only available to titled players, which occupies a unique middle ground between serious over-the-board tournaments and informal online scrimmages. Although Chess.com have certainly banned players for unlawful conduct in Titled Tuesday, the details of these bans have not been made public. This study seeks to shed light on what is in the dark and provide a unique, data-oriented analysis of cheating detection in Titled Tuesday.

Contents

Introduction.....	1
Early Forms of Cheating.....	1
The Emergence of Engine Assistance.....	1
The 2010 FIDE Olympiad Scandal.....	2
FIDE's Response: From Tolerance to Prevention.....	2
The Rise of Online Cheating.....	3
How Online Cheating Works.....	3
Detection Efforts by Chess Platforms.....	4
A Gap in the Literature: Titled Tuesday.....	4
Research Questions.....	5
Structure.....	5
Conclusion of Introduction.....	5
Literature Review.....	6
Overview of Existing Literature.....	6
Gaps in the Literature.....	6
Addressing Gaps: The Role of Titled Tuesday and Time-Based Metrics.....	6
Statistical Models: The Foundation of Cheat Detection.....	7
Regan & Haworth's Intrinsic Chess Ratings.....	7
Comparative Insights: Ferreira (2012) and Guid & Bratko (2006).....	8
Neural Networks for Cheat Detection.....	9
Patria et al. (2021): CNN and DNN Approaches.....	10
Limitations of the Patria et al. Model.....	11
Belhoucine & Feng (2025): Behavioural Features and ANN Classification.....	12
Critique of Belhoucine and Feng's Dataset and Methodology.....	13
Hybrid Approaches: Iavich & Kevanishvili (2025).....	13
Centaur and Complex Detection.....	14
Comparative Summary of Key Studies.....	15
Datasets: Strengths and Weaknesses.....	16
Engine Usage and Evaluation Depth.....	16
Conclusion and Rationale for This Study.....	17
Methodology.....	17
Research Design and Rationale.....	17

Dataset Collection and Preprocessing.....	18
Research Procedure.....	18
Engine Analysis Script.....	18
Custom Modifications.....	23
Machine Learning Model for Anomaly Detection.....	25
Temporal Analysis.....	27
Ethical Considerations	29
Data Collection and Preprocessing	29
Selection and Acquisition of the Dataset	29
Combining and Preparing Game Files	29
Local Runtime Estimation and University Cluster	30
Final Dataset	31
Data Analysis, Results and Discussion	32
Initial Accuracy-Based Analysis.....	32
Identifying Top 100 High-Accuracy Games.....	32
Case Studies of Edge Cases	32
Refinement of Filtering Criteria.....	33
Adjusting Move Count Range.....	33
Excluding Drawn Games	34
Suspicion Case Study: Frogo47.....	34
Game-Level Evaluation.....	35
Tournament-Level Accuracy Evaluation	35
Temporal Analysis of Player Behaviour	36
Time-Based Suspicion Modelling.....	36
Frogo47's Temporal Trends	37
Comparison Across Players	38
Selected Player Time Profiles	38
Comparative Analysis	43
Value of Visual Anomaly Detection.....	44
Machine Learning Approach.....	44
Generating the Suspicious Dataset.....	44
Suspicion Rate Summary Table	45
Comparison to Existing Research.....	46

Improvements on Engine-Based Models	46
Specialisation in Temporal Features	46
Strengths of the Unsupervised Approach.....	46
Research Questions	46
RQ1 – Can Move Time Indicate Cheating?	47
RQ2 – How Prevalent is Cheating in Titled Tuesday?.....	47
RQ3 – Which Features Are Most Predictive?	47
Research Limitations.....	47
Stockfish Engine Depth	47
Isolation Forest Calibration.....	47
Chess Understanding Requirements	48
Conclusion	48
Appendix 1 – Self Review	49
Bibliography	51

Introduction

When Deep Blue defeated Garry Kasparov in May of 1997, the silicon ceiling was shattered – the seemingly safe stronghold surrounding the intellectual dominance humans had over computers, evaporated, leaving in its place a ruptured gas canister, once silent and contained, now screaming with the untamed potential of the next generation of virtual intelligence. Fast forward nearly thirty years, and while artificial intelligence still feels toddler-aged in the scope of its lifespan, many of the theorised uses of technology have been realised. In 1997, GPS would have got you lost, electric cars meant Scalextrics, and ‘the cloud’ was what made the sky grey. Nowadays, apps like Google Maps and Waze are cornerstones of every sensible citizen’s home screen, electric vehicles represent around one in four new car sales in Europe (International Energy Agency, 2024), and the total amount of data consumed globally reached 149 zettabytes in 2024 (Taylor, 2024) – a 5.73 million percent increase from 1993 (Hilbert & López, 2011). These drastic advancements have seeped into every crevasse of every industry on the planet, eroding old limitations and creating new standards far beyond human capability. With its emphasis on intellect and intricacy, chess is no exception.

Early Forms of Cheating

However, as technology has advanced, so too have the ways it can be effectively exploited, and there are not many fields where this is more evident than chess. Before the age of computers, cheating in chess was a strenuous task. In fact, the earliest recognised method of cheating utilised an ingenious device named “The Turk” (Chess.com, 2023). Created by Wolfgang von Kempelen in 1770, it consisted of a mystical-looking life-size figurine sitting by a wooden box. He had a smoking pipe in his left hand, and nothing in his right, as this was what he would play the chess moves with. Inside the box were only wires to calculate the best chess move in any position and signal it back to the figure, who would then play the move. The Turk toured the world, and managed to beat Napoleon Bonaparte and Ben Franklin, but narrowly lost to then-world champion - Francois-Andre Danican Philidor. It seems inconceivable that a device so advanced existed in the same century that doctors were using leeches to balance the body's humours, and rightly so, because it was a façade. The way The Turk operated was a strong chess player would sit in the box and force the figure’s hand to play good moves. The Turk fooled the world, with the secret only being revealed over 80 years later. This makes it the earliest known form of cheating, and a very successful one at that.

The Emergence of Engine Assistance

Before the Deep Blue match in 1997, the idea of a computer being used to aid a competitive chess player would have been ridiculous, as humans were still seen as superior. Kasparov was the strongest player in the world at the time and is regarded as the best player of all time by some grandmasters. Given that Deep Blue was a computer with limited chess understanding but an instantaneous brute-force calculation ability far greater than any human, it was obvious that all players could gain an advantage with a cutting-edge chess engine on their side.

The first use of an engine during a tournament post-Deep Blue came at the 1998 Böblingen Open when Clemens Allwermann was found guilty of using Fritz (ChessBase, 1997) – an early downloadable engine for Windows (Goos et al., 2009, p. 107). This resulted in the Bavarian Chess Federation issuing Allwermann a lifetime ban from future tournaments. At this point in time, although the power of using unlawful engine assistance was well-known, it was not an easy feat to sneak a device into a tournament, as said devices were mostly large and not portable, increasing the risk of being caught. In fact, it was not until 2005 at the HB Global Chess Challenge in Minneapolis that the next major incident occurred. This saw Alexandre Mirtchouk – a player competing in the under 2000 rating section caught using a cell phone to communicate with someone else in the building. It was hypothesised by directors that he was receiving moves, which led to an ethics complaint, and a disqualification from the tournament (Greengard, 2005). Only six weeks later, Mirtchouk played in the World Open and came in joint first place in the 2200 and under section, which earned him \$5,833. The federation attempted to disqualify him but lacked hard evidence that he cheated in the previous event (Greengard, 2005).

Despite a few more scandals between 2005 and 2010, like the two aforementioned examples, cheating in chess seemed to be an issue that was mostly being contained by governing bodies across the world. Cheaters seemed to consistently display suspicious behaviour, which made the task of detecting a suspect relatively easy. However, with the dawn of the cell phone age, suddenly cheating “over the board” – which refers to an in-person game – became far more manageable, and consequently, far more problematic for the chess world.

The 2010 FIDE Olympiad Scandal

This shift was dramatically illustrated at the 2010 FIDE Olympiad in Khanty-Mansiysk, Russia – widely regarded as the “World Cup of chess” – where the French team got caught in one of the first major cheating controversies at a high-stakes event. This time, the plan seemed flawless. The backup player, Cyril Marzolo, would follow his teammates’ games from home and run them through a chess engine. He would then text the best moves to the team’s coach, Arnaud Hauchard, who would discreetly signal these moves to Sébastien Feller – one of the French players – by moving around the playing hall and sitting or standing at predetermined tables. The plan was working until Marzolo had to borrow a phone from a French Chess Federation (FFE) employee. That employee noticed a suspicious message sent from Hauchard that read: “Hurry up, send moves.” This led to the exposure of the cheating scheme. The trio, including two grandmasters, received suspensions of nearly three years from both FIDE (the World Chess Federation) and the FFE (ChessBase, 2012). This lit the chess world on fire, sparking a campaign to urge FIDE to guarantee a fifteen-minute delay between moves being played at an event and moves being displayed online (Barden, 2011).

FIDE's Response: From Tolerance to Prevention

As cheating incidents became increasingly prevalent over the next few years, FIDE realised that something had to be done. In 2013, the ACC (Anti-Cheating Commission) was launched, demonstrating FIDE’s acknowledgement that cheating was a significant issue in the chess

world. Before this, cheating cases were handled inconsistently by individual tournament organisers or national federations. With a centralised board responsible for investigating allegations and standardising penalties, FIDE hoped they would be able to put a clamp down on cheating and reinstate integrity in chess. After a meeting in Paris in October 2013 and Buffalo in April 2014, the ACC released their official legislation in November 2014, which featured both preventative and reactive measures against cheating (FIDE, 2014).

In this new legislation, it was stated that at every “Maximum Protection Event”, at least one of metal detectors, electro-magnetic screening for metal detection, electronic jamming devices, or closed-circuit cameras, is a mandatory requirement for entry (FIDE, 2014, p. 22). This was put in place to stop hopeful cheaters from interacting with anyone outside of the playing hall, and to prevent cheaters from bringing devices with portable engine applications on them. FIDE also announced the creation of a new internet-based game screening tool “for pre-scanning games and identifying potential instances of cheating, together with the adoption of a full-testing procedure in cases of complaints” (FIDE, 2014). Before this, there were no FIDE-approved statistical screening tools available to tournament organisers. These efforts marked a significant shift in FIDE’s attitude toward chess cheaters, representing a turning point from passive tolerance to proactive prevention.

The Rise of Online Cheating

Despite FIDE’s best efforts, the war on cheating was far from over. Throughout the following years, it became clear that the threat of cheating had not been eliminated – it had only changed its habitat. Over-the-board cheating, while not completely eradicated, was being contained. Instead, a new, more slippery, more elusive enemy had emerged: online cheating.

In 2020, Chess.com – the world’s largest website facilitating online chess – reported the closure of over 400,000 accounts for cheating, with more than 500 of which being titled players. To understand what a “titled player” is, in chess, there are four levels of achievement one can earn through high performance in FIDE-certified events. Candidate Master (CM) is the lowest, followed by FIDE Master (FM), International Master (IM), and finally Grandmaster (GM). There are also women-only titles which work by adding a “W” in front of CM for WCM, a “W” in front of FM for WFM, etc. Considering that over 500 of the banned accounts were titled players, and there are just under 23,000 titled players worldwide (FIDE, 2025), more than 2% of all titled players had already been caught cheating online by 2020, highlighting just how ubiquitous the issue has become.

How Online Cheating Works

There are two main reasons that cheating online is so widespread. Firstly, engines have never been more accessible than they are now. All one must do to cheat is load into a chess.com game, open an online engine in a separate tab, and feed their opponent’s moves into the computer. Within two or three seconds, the best possible move is detected in any possible chess position. With a few simple clicks, anybody with a stable internet connection has the potential to play at a level higher than any grandmaster in the history of chess.

Secondly, if done cleverly, it is almost impossible to pick up on. Unlike over-the-board cheating, there are no suspicious bathroom breaks, worried looks, or unusual glances that can be picked up on in a regular online chess game. The only thing that must be done to avoid being caught is to deviate from the computer's best move every now and again. If the best moves are saved for crucial positions, a cheater should manage to stay off the chess.com wanted list with ease.

Detection Efforts by Chess Platforms

However, despite the difficulty of catching cheaters online, platforms like Chess.com have developed sophisticated systems to detect suspicious behaviour. To prevent reverse engineering of their algorithm, Chess.com have not released the exact method they use to detect cheaters, stopping people from hiding in plain sight. What they have said is this:

“We load these games into a tool that provides the probability that a given player is playing cleanly or with the assistance of a computer engine. Before any accounts are closed, all reports are thoroughly reviewed by a team of specialists who have reviewed and closed thousands of accounts in their roles as Chess.com statisticians” (Chess.com, 2022).

This suggests that Chess.com have an algorithm at their disposal that uses statistical modelling/methods to calculate the likelihood of a player's moves aligning with moves selected by an engine, with respect to the player's rating and expected strength. It also demonstrates that banning an account is a decision that is only reached after heavy consideration from the team of experts.

Statistical detection models like those developed by Professor Kenneth Regan offer insight into how these systems might work. Regan is one of the most prolific figures in chess cheating detection methods and has been published countless times on the topic. One of his more famous papers – *Intrinsic Chess Ratings* (Regan & Haworth, 2011) – documents the creation of a model that evaluates the likelihood of a player's moves aligning with the moves of a state-of-the-art chess engine. This paper exists as a foundational work in the field of cheating detection and was crucial for the development of modern algorithms. Because of its prominence, it will be discussed in detail later in this study.

A Gap in the Literature: Titled Tuesday

With over-the-board cheating scandals being mostly circumstantial, and online cheating incidents being commonly dismissed because of anonymity protection or lack of impact, this project identifies a gap in the current literature surrounding cheating in chess. Specifically, it aims to identify players who have used engine-assistance in Titled Tuesday games played between October 28, 2014, and September 24, 2024 – a weekly high-stakes event only available to titled players, which occupies a unique middle ground between serious over-the-board tournaments and informal online scrimmages. Although Chess.com have certainly banned players for unlawful conduct in Titled Tuesday, the details of these bans have not been made public. This study seeks to shed light on what is in the dark and provide a unique, data-oriented analysis of cheating detection in Titled Tuesday.

Research Questions

The study has three main research questions. Each question will be answered through the interpretation of data produced by the developed algorithm, along with insights from existing academic and professional work in the field. The questions are as follows:

- 1.) “To what extent can move time be used as an indicator of engine assistance in competitive online chess games?”
- 2.) “How significant is the issue of cheating in Titled Tuesday events?”
- 3.) “Which in-game factors are most predictive of engine usage in online chess?”

Structure

The dissertation will consist of four main sections. First, the literature review will delve into fundamental concepts in online chess, critique the works of researchers who have already developed models to detect cheating, and identify the research gaps that this study addresses. This is followed by the methodology section, which outlines the design of the algorithm, introduces pre-processing steps, explores the ethical considerations necessary to meet Edinburgh Napier’s guidelines, and describes the way in which the research was carried out.

The third section will focus on the data collection and preprocessing detailing the challenges of sourcing a reliable dataset and the steps taken to convert raw PGN files into a usable format. There was a great deal of preprocessing required to make the dataset valuable, so this will be described in detail, showing the journey of the data from messy and unusable to clean and useful. The final section of the paper is the data analysis, results, and discussion section. Here, the results of the data analysis will be presented and discussed. Visualisations and statistical summaries will also be displayed, with key insights being made to help address the research questions. The importance of the findings with respect to the existing body of research will be discussed, with inspirations being credited, and novel findings being recognised.

Conclusion of Introduction

As the platform for which competitive chess is held is migrated to the online world, it is vital that the rules, regulations, and tools used to preserve its integrity evolve *with* the game, not behind it. By building upon the existing works of trailblazers in cheat detection, this study aims to explore new ways in which foul play can be identified, particularly through time-based metrics. The following chapter reviews existing work on chess cheating, providing the foundation upon which this dissertation is built, and identifies where it can be improved for a comprehensive, data-rich approach to catching cheaters in online chess.

Literature Review

Overview of Existing Literature

To understand how this study fits into the broader conversation surrounding chess cheating, it is essential to first explore the existing literature in the field. Specifically, the rationales, methodologies, and key findings of each study must be analysed to appreciate the nuances, areas of agreement, and gaps in research within the field of cheating in online chess. Over the last thirty years, several experts have developed algorithms with the goal of identifying potential breaches of fair use. From Regan and Haworth's probabilistic approach which uses a Bayesian model to estimate player skill from move quality (Regan & Haworth, 2011), to Patria et al.'s later work which tests convolutional and dense neural networks on board position and engine-analysed data (Patria et al., 2021), each effort has followed a distinct method for detecting foul play.

Gaps in the Literature

While the body of work relating to cheating detection is extensive, there are still notable gaps to consider. For example, the vast majority of academic studies rely on datasets consisting of either high-stakes over-the-board games or low-stakes online games. While the former are certainly significant, and dominate chess headlines when scandals arise (the Magnus Carlsen vs Hans Niemann saga being a recent example (Lutz, 2023)) - there is limited room for cheating due to the strict protocols put in place by FIDE, and the lack of anonymity inherent to an in-person environment. Resultantly, detecting cheating in expansive OTB datasets is akin to finding a needle in a haystack.

On the other hand, while engine-assistance is more common in low-stakes online games, the consequences of catching and punishing a cheater are minimal in terms of broader impact on the community. Without financial incentives or reputation at stake, no prize money is unfairly won, and little harm is done – such cases can reasonably be considered as relatively victimless crimes.

Another significant gap in the research is the lack of studies that consider the time taken to play each move as a potential indicator of engine-assistance. When using a tool like Stockfish (2024) – the most widely used chess engine and the one used in this study – cheaters will often have the program open in a parallel window to their chess game, inputting their opponent's moves and waiting for the engine to feed them the optimal response. A typical cheater can perform this process in roughly five seconds, and significantly, will perform this process repeatedly, playing each move in consistent increments regardless of the complexity of their position. A fair player would naturally spend a lot more time on a difficult move than a simple move, making this discrepancy a promising metric for detecting suspicious play.

Addressing Gaps: The Role of Titled Tuesday and Time-Based Metrics

This dissertation aims to investigate these gaps in the research. Firstly, to address the issue of inadequate datasets, a dataset comprised of Titled Tuesday games – a high-stakes, online

weekly tournament hosted by Chess.com (Chess.com, 2025a) – is used as the basis of analysis. This occupies a middle ground between casual online play and professional OTB events. Secondly, to address the absence of time-based metrics in existing studies, this research conducts not only traditional move-by-move analysis using Stockfish, but also a parallel analysis which considers move time as a potential factor for engine assistance.

By examining the vast body of work surrounding chess cheating, this literature review will evaluate the strengths and limitations of existing detection methods, with particular attention given to their practical applications and methodologies. In doing so, it will establish the foundation of the study and demonstrate why a systematic inspection of the last ten years of Titled Tuesday games – incorporating temporal and engine-based analysis - is a valuable, and novel contribution to the chess world.

Statistical Models: The Foundation of Cheat Detection

One of the main ways academics have attempted to identify engine use is through probabilistic and statistical analysis. In general, this involves comparing a player's moves to those suggested by a strong chess engine and calculating the probability or likelihood that such moves could occur naturally given some prior knowledge. For instance, a model might factor in the player's rating, historical performance, and the difficulty of the position.

Regan & Haworth's Intrinsic Chess Ratings

One of the most respected works in this field is Ken Regan and Guy Haworth's 2011 paper – "Intrinsic Chess Ratings (Regan & Haworth, 2011). Regan and Haworth build on previous work regarding "fallible agents" (Reibman & Bailard, 1983) – a term that defines humans as decision-makers who are subject to errors and uncertainties. By modelling such agents with two parameters - sensitivity (s), which is defined as how well a player can distinguish a good move from a bad move, and consistency (c), which is how often a player avoids bad moves, Regan and Haworth create a statistical profile of each player's ability to make decisions.

The authors found that higher Elo players demonstrate lower s scores (meaning high sensitivity), and higher c scores (indicative of strong consistency). A combination of both shows that they can both identify and play good moves. This can be seen in [Figure 1](#), which shows the trend in s and c scores across varying Elos.

Elo	s	c	cfit	sfit	mmp/mma	adp/ada	Qfit
2700	0.078	0.503	0.513	0.08	56.2/56.3	0.056/0.056	0.009
2600	0.092	0.523	0.506	0.089	55.0/54.2	0.063/0.064	0.041
2500	0.092	0.491	0.499	0.093	53.7/53.1	0.067/0.071	0.028
2400	0.098	0.483	0.492	0.1	52.3/51.8	0.072/0.074	0.016
2300	0.108	0.475	0.485	0.111	51.1/50.3	0.084/0.088	0.044
2200	0.123	0.49	0.478	0.12	49.4/48.3	0.089/0.092	0.084
2100	0.134	0.486	0.471	0.13	48.2/47.7	0.099/0.102	0.034
2000	0.139	0.454	0.464	0.143	46.9/46.1	0.110/0.115	0.065
1900	0.159	0.474	0.457	0.153	46.5/45.0	0.119/0.125	0.166
1800	0.146	0.442	0.45	0.149	46.4/45.4	0.117/0.122	0.084
1700	0.153	0.439	0.443	0.155	45.5/44.5	0.123/0.131	0.065
1600	0.165	0.431	0.436	0.168	44.0/42.9	0.133/0.137	0.129

Figure 1 - Adapted table of mean fitted and actual values of sensitivity (*s*), consistency (*c*), based on OTB games from 2006-2009 in (Regan & Haworth, 2011)

The study involved a dataset of over 150,000 OTB tournament games and applied maximum likelihood estimation to fit *s* and *c* parameters to each player's behaviour, allowing the model to identify how plausible each player's move choices were with respect to their supposed ability.

To define *s* and *c* scores for each player, they utilised the Rybka 3 chess engine (Vasik & Kaufmann, 2007) as it was "rated the strongest program on the CCRL rating list (Banks et al., 2010) between its release in August 2007 and the release of RYBKA 4 in May 2010" (Regan & Haworth, 2011, p. 834). In every position of every game, each legal move is scored by the engine, and the model calculates the probability that each move is played using the formula:

$$y_i = e^{-(\delta_i/s)^c}$$

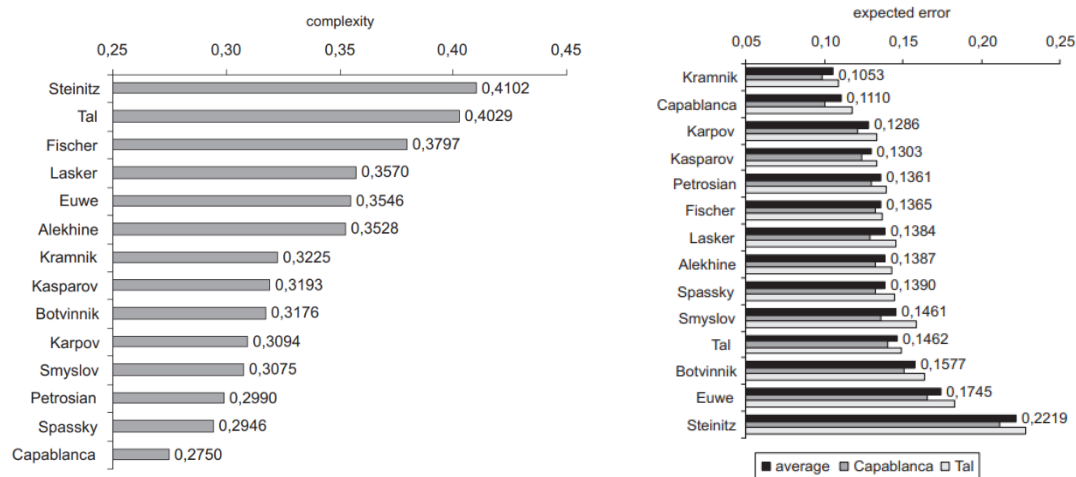
Where y_i is the unnormalised likelihood of move *i* being played, and δ_i is the difference in engine score between the optimal move and move *i*. To determine *s* and *c*, they then perform a fitting process where the curve of all y_i s that best matches the expected distribution at the player's Elo is produced. *S* and *c* scores that are too low or high, respectively, suggest the player is playing stronger than expected, thus suggesting potential engine-usage.

Comparative Insights: Ferreira (2012) and Guid & Bratko (2006)

With over 60 citations (Google Scholar, 2025), *Intrinsic Chess Ratings* is one of the most respected papers in its field. Regan and Haworth's model has been praised for incorporating positional complexity – something Ferreira (2012) acknowledges is lacking in his study - "Determining the Strength of Chess Players Based on Actual Play" - released a year after Regan and Haworth's. He states, "There is no consideration whatsoever of the complexity of the positions. This is perhaps one of the few issues in which the present approach is lacking in comparison with both Guid and Bratko (2006) and Regan and Haworth (2011)" (p. 17).

Top chess players all have varying styles of play and typically try to steer the game toward positions that complement their strengths or attack their opponents' weaknesses. Guid and Bratko (2006) viewed this as an important insight in their research titled *Computer Analysis*

of *World Chess Champions*, where they introduced a custom metric for position complexity. They understood that players who favour complicated positions (known as sharp positions) – such as the Latvian former world champion Mikhail Tal - are more likely to accrue higher error rates, as they opt for complex positions more often. This is supported by [Figures 2 and 3](#), which show Tal registering the second-highest average position complexity and the fourth-highest expected error coefficient among all historical world chess champions.



Figures 2 & 3 - Expected error rates of historical World Chess Champions, including comparisons with Capablanca and Tal, based on Guid and Bratko's (Guid & Bratko, 2006) analysis

Ferreira (2012) recognises this as a positive aspect of Guid and Bratko's (2006) work, stating that it was "one of the few issues" (p. 17) in his approach. Instead of fitting a statistical model like Regan and Haworth (2011), Ferreira (2012) directly measures the quality of each move using an engine and creates a distribution for each player representing how often they make moves of different strengths. He then compares distributions between players to estimate their difference in strength or assumed Elo. This is a more straightforward comparison based on actual gameplay, without requiring complex statistics.

However, one way in which the authors agree in methodology is that both works can be considered statistical by nature. On the one hand, Regan and Haworth (2011) use a model based on fitting two parameters – sensitivity and consistency – which are estimated with maximum likelihood estimation. On the other hand, Ferreira (2012) takes a more simple, non-parametric approach, building distributions of move strength and then comparing them. While Ferreira (2012) does not adopt involved statistical inference, his method of finding the average difference between the move played and the optimal move still relies on statistical principles, using the mean instead of a probabilistic model.

Neural Networks for Cheat Detection

Another commonly used method to detect players performing above their rating is by using neural networks. Generally, this means training a model on a large dataset of chess games, allowing it to learn patterns of play and discern which players are playing accurately enough to raise suspicion. It is important to first understand the difference between a chess engine and a neural network before assessing popular studies.

A chess engine, for example, Stockfish (2024), can evaluate chess positions and suggest the strongest moves using brute-force calculations with a simple position evaluation function. To perform these calculations, it looks several moves into the future and gives a numerical evaluation for the position. Positive scores indicate white has a better position, while negative scores indicate black has a better position. For example, +1.1 means white has a slightly better position, and -6.2 means black has a far better position. Whichever move leads to the greatest evaluation is elected as its top move in the position.

On the other hand, a chess neural network, for example, AlphaZero, (Silver et al., 2018) uses deep reinforcement learning to evaluate positions. AlphaZero learned chess from scratch, playing 44 million games against itself in training (Silver et al., 2017) with only the rules of the game to its knowledge. To evaluate a position, it transforms the board into a 3D tensor, inputs it into the neural network, and outputs two things: a probability distribution of the best legal moves, and the expected outcome of the game. When used like this, neural networks like AlphaZero (Silver et al., 2018) can find better gameplans than any engines, as demonstrated by its dominant match vs Stockfish in 2017 where they played 100 games. AlphaZero won 28, they drew 72 times, and Stockfish won 0 (Silver et al., 2017). While AlphaZero was designed to play chess at the highest level, researchers have since explored the potential of using similar neural networks not to play chess, but to detect instances of cheating within it.

Patria et al. (2021): CNN and DNN Approaches

In Patria et al.'s (2021) study - *Cheat Detection on Online Chess Games using Convolutional and Dense Neural Network* - the researchers built two types of neural networks: A convolutional Neural Network (CNN), and a Dense Neural Network (DNN), with the goals of classifying each game in their database into three labels: "No cheating", "White cheated", or "Black cheated". Each model was tested on two types of data. The first of which was unanalysed data, consisting of raw board positions. Secondly, they tested on analysed data, which was made up of raw board positions along with Stockfish evaluations and clock times after every move.

The games used came from the Lichess API (Lichess.org, 2024), which provides access to a database containing over 6.4 billion standard rated online games (Lichess.org, 2025). Patria et al. (2021) labelled "cheater" games using a public tool called Cheat-Net – "a python application dedicated to analysing reported cheaters on Lichess" (Clarkerubber, 2017, p. 2). They analysed 2,500 games by users flagged as cheaters by Cheat-Net (2017), and 2,500 by predicted clean players, comprising a dataset of 5,000 games with 32 million board positions. This was split into 80% training data and 20% testing data, with 10% of the training data used for validation.

For the DNN, for both the analysed and raw data, each game was flattened into 71,680 features. Within the analysed dataset, a second array was included, which contained the Stockfish evaluation after every move, and the clock time after every move adding an additional 160 features (Patria et al., 2021, p. 393). All datasets were fed through three fully connected layers (512, 256, and 64 neurons, respectively), with each layer using ReLU

(Rectified Linear Unit) activation to help the model focus on important features and encourage sparse learning. For the output layer, SoftMax activation (Patria et al., 2021a) classified each game into the three categories described previously.

The CNN followed a similar structure, with the main difference being how the input was processed. Instead of flattening board positions into features, the CNN reshaped the games into five-dimensional tensors, encouraging the model to focus on the spatial arrangement of the board (e.g. a strong rook on an open file). CNNs are particularly good at recognising patterns, regardless of where they are on the board. For instance, a CNN would recognise that a knight fork on g4, is the same idea as a knight fork on b7, whereas a DNN would not have the intuition to understand this since the data is flattened. After the data was processed, the SoftMax output layer worked identically to the DNN, leading to results indicating cheaters, non-cheaters, and clean games. The models produced the following accuracies in [Figure 4](#):

Neural Network	Input Data	Validation Accuracy	Test Accuracy
CNN	Analysed	57.50%	57.2%
CNN	Unanalysed	53.75%	57.2%
DNN	Analysed	56.25%	56.25%
DNN	Unanalysed	48.75%	51.6%

Figure 4 - Validation and test accuracy of four neural network models (Patria et al., 2021, p. 393 - 394)

As seen above, the CNNs outperformed the DNNs irrespective of whether the data was analysed or unanalysed, likely due to their spatial awareness. Analysed data was significantly more accurate for both models however, with the DNN increasing 8.5% from unanalysed to analysed. This is probably because of the inclusion of Stockfish evaluations, as cheaters will play closely to the engine's top choice often. Across all the models, the training accuracy reached around 97%. Patria et al. (2021, p. 394) hypothesised that this was because the data was not varied enough. They stated that “due to the nature of the game of chess itself, in which billions upon billions of positions could be generated, the training data of ~32 million positions are not enough to cover every oddity every player could make” (p. 394).

Limitations of the Patria et al. Model

While this study is a strong example of how neural networks can be used to detect cheating, there are several limitations that perhaps contributed to the suboptimal results. For one, the use of Cheat-Net (Clarkerubber, 2017) to label cheaters and non-cheaters is problematic. Cheat-Net is not peer-reviewed, nor is it commonly utilised, so using it as the foundation upon which the entire dataset is built is a questionable decision and lacks scientific validation. Secondly, the setup of the model assumes that all games played by a confirmed

cheater, are examples of cheating. This is an unrealistic simplification, as many cheaters toggle their engine usage depending on whether they are in a critical moment in the game, or they may only cheat very occasionally. Therefore, by labelling entire games by cheaters as “cheated”, there is a high risk of a large number of false positives, potentially causing the model to associate normal behaviour with cheating.

Belhoucine & Feng (2025): Behavioural Features and ANN Classification

Another paper that explores the use of neural networks for cheating detection is Belhoucine and Feng’s (2025) work titled *Detecting Chess Cheating Thanks To ML & ANN Technology*. While Patria et al. (2021) use two deep learning models: a DNN and a CNN, Belhoucine and Feng (2025) utilise a custom artificial neural network (ANN) in their methodology. However, the core of their study was a hybrid, combining statistical analysis with machine learning. Instead of using raw board positions as input, they collected behavioural metrics including average time per move, position complexity, Shannon entropy, move accuracy, and distribution of move quality as inputs. These factors were statistically evaluated independently of the ANN by the authors. Another key difference was that while Patria et al. (2021) used Cheat-Net (Clarkerubber, 2017), to create a labelled dataset of suspicious games, Belhoucine and Feng (2025) created their own labels based on a combination of behavioural patterns and statistical indicators (ie: unusually high accuracy and low entropy). While this seems dubious, it could perhaps be considered a more optimal strategy, as cheating can be very obvious to a chess player, even with little experience, especially when engine evaluations are compared with Elo ratings.

After data was collected and statistically processed, Belhoucine and Feng (2025) fed the behavioural features into their shallow ANN. The features were selected as they were hypothesised to be strong indicators of suspicious patterns, such as consistently accurate move choices, and low variance in complexity. The ANN then mapped these inputs to a binary decision: “fair play” or “cheating” (Belhoucine & Feng, 2025, p. 36) for each game, enabling the model to flag games with engine-like behaviour.

To evaluate the performance of the model, the authors tested on 63 games (Belhoucine & Feng, 2025) played by players ranging from 900 Elo (in the 11th percentile of all players) and 2500 Elo (in the 99th percentile of all players) (Lichess, 2025). 36 of the 63 games were pre-labelled as “cheated games”. Results can be seen in [Figure 5](#):

Table 1

Player	Elo Rating	Total games	Games Cheated (Y)	Games Detected (Y)	Game Cheated (N)	Game Detected (N)	Reason for Detection (if any)
Player 1	900	7	6	6	1	0	Correct detection in all games where cheating occurred.
Player 2	1200	7	5	5	2	0	Always detected. Correct detection in all games where cheating occurred.
Player 3	1500	7	3	3	4	0	Correct detection in all games where cheating occurred.
Player 4	1800	7	6	6	1	0	Always detected. Correct detection in all games where cheating occurred.
Player 5	1900	7	4	4	3	0	Correct detection in all games where cheating occurred.
Player 6	2100	7	3	3	4	0	Correct detection in all games where cheating occurred.
Player 7	2300	7	4	4	3	2	Detected for high accuracy and timing in Game 1; subtle cheating in Game 2 went unnoticed
Player 8	2400	7	2	2	5	2	2 games undetected due to complex timing and behavior patterns.
Player 9	2500	7	3	3	4	4	4 games undetected due to high precision moves and engine like play.

Games Cheated (Y): the games in which the player has cheated

Games detected (Y): the games in which the player was detected as cheater

Games Cheated (N): the games in which the player didn't cheated

Figure 5 – Results table from Belhoucine and Feng (2025)

Critique of Belhoucine and Feng's Dataset and Methodology

As seen in the results, 36 of the 36 cheated games were correctly identified by the model, indicating 100% sensitivity, while 8 out of 27 of the fair games were falsely identified as cheated. This indicates that the model was overly sensitive, particularly at higher Elo levels, where legitimate play can closely resemble engine-assisted behaviour. The dataset tested on likely complemented the model's over-sensitivity, containing more cheated games than fair games. In reality, detecting a cheated game consists of dealing with very large datasets that are dominated by fair games. The dataset makes the model look stronger than it is, as the 29% false positive rate would be highly contentious if used in the real world, flagging nearly a third of fair players as cheaters.

Another key issue lies in the way the cheating dataset is comprised. Although the method of manually labelling is potentially more reasonable than the method used by Patria et al., (2021) where Cheat-Net (Clarkerubber, 2017) was employed, it still presents limitations. Manually defining what constitutes cheating introduces subjectivity and skews the model in favour of the authors' opinion. Given these challenges, a stronger alternative may be to apply unsupervised learning techniques, which can detect anomalies in player behaviour without needing predefined cheating labels.

It should be noted that the 36/36 cheated games detected is still impressive and evokes the idea of using this model on a dataset of suspicious games. While the false positive would still likely be apparent, being able to consistently and correctly label cheated games is a useful tool and would certainly be appropriate in such an environment. In response to these limitations, other approaches have tried to combine statistical indicators with behavioural in more comprehensive ways.

Hybrid Approaches: Iavich & Kevanishvili (2025)

One example is the work by Iavich and Kevanishvili (2025) and their hybrid CNN-RNN (Recurrent Neural Network) model which utilised behavioural factors and engine analysis to detect cheating. As input to their model, the authors collected centipawn loss data from both Stockfish – a chess engine, and Leela Chess Zero – an open-source neural network based on AlphaZero (Pascutto & Linscott, 2019). They also input move likelihood data from Maia (McIlroy-Young et al., 2020) – a state-of-the-art human-move prediction engine, as well as timing patterns for each game.

To create their dataset, Iavich & Kevanishvili (2025) collected games from Lichess (2024), Chessbase (2025), and The Free Internet Chess Server (2025) to form a 26,000-game dataset encompassing games played by Grandmasters, identified cheaters, and personal games (Iavich & Kevanishvili, 2025, p. 136). Games were then run through Stockfish (2024) and Leela Chess Zero (Pascutto & Linscott, 2019) to output centipawn losses. This is a widely adopted measure of how close a player's move is to the engine-recommended move, with large loss suggesting inaccuracy, and loss close to zero suggesting high accuracy. Maia (McIlroy-Young et al., 2020) has the ability to replicate decisions made by players at varying skill levels, so by creating a distribution that represented how likely a human at a certain Elo would be to play a certain move, the authors were able to identify suspicious moves as strong moves that were not predicted by Maia. This level of pre-processing is more rigorous than the works by Patria et al. (2021) and Belhoucine and Feng (2025), and led to a stronger foundation for their model to predict anomalies.

In addition to a more thorough preprocessing pipeline, Iavich & Kevanishvili (2025) also adopt a more complex neural network than those used in the other studies. While Patria et al. (2021) utilised CNN and DNN models trained on position tensors, and Belhoucine and Feng (2025) created a shallow network built on user-created features, Iavich & Kevanishvili (2025) introduced a hybrid CNN-RNN network, able to recognise both spatial patterns on the board, and temporal patterns of player behaviour as the game played out.

The authors agree with this premise, stating that “ReLU (Rectified Linear Unit) is employed for the convolutional layers, which excels at handling image-like data and extracting meaningful features” (Iavich & Kevanishvili, 2025, p. 140). They also demonstrate their reasoning for the RNN component, positing that they “excel at recognising temporal patterns and sequences, which are crucial for detecting anomalies and irregularities in players' moves that may indicate cheating” (p. 135). They go on to affirm the reasons for the hybrid approach, suggesting that it “allows us to differentiate between authentic human gameplay and instances of external assistance, thereby ensuring the integrity of online chess competitions” (p. 135).

Centaurs and Complex Detection

The standout element of Iavich and Kevanishvili's (2025) research however, is their attention to “centaurs” – defined by the authors as “players who combine their human ingenuity with the computational power of chess engines” (Iavich & Kevanishvili, 2025). This could be as simple as delaying playing engine moves by a few seconds in sharp positions, or as complex as consistently selecting the engine's second or third-best move to avoid detection. By

looking for moves that are accurate but not human-like (low centipawn loss, low Maia probability), considering centipawn loss data from multiple engines, and cross-referencing with time data, their model attempts to catch centaur-like behaviour.

After training their hybrid CNN-RNN model on 26,000 games, over 150 epochs, the researchers reported a validation accuracy of 98.4% in distinguishing between human and engine-assisted play. When compared with Patria et al.'s (2021) and Belhoucine and Feng's (2025) best reported test accuracies of 57.2% and 87.3% respectively, Iavich and Kevanishvili's (2025) model is considerably the strongest at successfully detecting chess cheating.

Despite their extremely high validation accuracy, Iavich and Kevanishvili (2025) make one caveat; suggesting that their biggest limitation was "the omission of "centaur" games, where players utilize chess engines for assistance while still making their own decisions" (p. 143). Although detecting centaur games was perhaps the author's biggest challenge, "the high success rate of this study underscores the immense potential of integrating various analytical tools to combat digital cheating in chess" (p. 143).

Comparative Summary of Key Studies

Now that the most notable studies in the field of chess cheat detection have been reviewed, it is useful to compare some of their key components and evaluate which approaches seem to be optimal. [Figure 6](#) outlines two key characteristics of each piece of research that have not been explored in detail yet: the dataset used, and the engine(s) used.

Study	Dataset	Engine(s) Used	Result / Accuracy
Regan & Haworth (2011)	150,000+ OTB tournament games	Rybka 3 (depth 13)	No classification; s & c scores fitted to player ability
Guid & Bratko (2006)	World Championship matches (historical OTB)	Crafty (depth 12, adapted for positional complexity)	Expected error rates and complexity scores; no classification
Ferreira (2012)	Assorted online and historical games	Houdini 1.5a (depth 20)	Elo estimated using move accuracy distributions; no classification
Patria et al. (2021)	5,000 Lichess games (2,500 flagged cheaters via Cheat-Net)	Stockfish (evaluations for analysed data)	CNN (Analysed): 57.2% test accuracy; DNN (Analysed): 56.25% test accuracy
Belhoucine & Feng (2025)	450,000 in training, 63 manually-labelled games in test data (36 cheated)	Not specified (used engine evaluations/statistics)	36/36 cheated games detected, 8/27 false positives (87.3% accuracy)
Iavich & Kevanishvili (2025)	26,000 games from Lichess, Chessbase, FICS	Stockfish, Leela Chess Zero, Maia	98.4% validation accuracy (hybrid CNN-RNN)

Figure 6 – Research Feature Comparison

Early studies (Regan & Haworth, 2011), (Guid & Bratko, 2006), (Ferreira, 2012) focus on statistical modelling, engine evaluation, and complexity, without using machine learning. They have a slightly different aim, which is to understand what constitutes human play or estimate Elo based on move accuracy, but they do not attempt binary cheat detection. However, they are still fundamental to the later studies; Regan and Haworth's (2011) *Intrinsic Chess Ratings* is cited heavily in all three of the neural network researches discussed. The more recent studies (Patria et al., 2021), (Belhoucine & Feng, 2025), (Iavich & Kevanishvili, 2025) do build classifiers, either with deep learning or neural networks, and try to directly label games as cheated or not cheated. Due to the nature of this paper where detecting cheating is paramount, these three are more significant, and form the basis for comparison when evaluating reliability, accuracy, and practicality of the algorithm.

Datasets: Strengths and Weaknesses

Patria et al. (2021) and Belhoucine and Feng (2025) use small datasets, which raises reliability concerns, and could be one of the causes of the comparatively weaker results. While Patria et al.'s (2021) dataset was small with only 5,000 games, the main issue lies in its use of Cheat-Net (Clarkerubber, 2017) which flags all games played by cheaters as cheated, and is not a trustworthy source. Although Belhoucine and Feng's (2025) training dataset was large with over 450,000 games (p. 29), their test dataset only featured 63 games, with more than half of which being labelled as cheated. This is too substantial of a proportion, and is not representative of the real world.

In contrast, Regan and Haworth (2011) and Iavich & Kevanishvili, (2025) both use more reliable datasets. Regan and Haworth (2011) analysed over 150,000 over-the-board tournament games, which guarantees a supreme level of integrity because of their size and legitimacy. Although Iavich and Kevanishvili's (2025) dataset was of medium size, the way it was sourced from various reputable sources, and then pre-processed rigorously using several different engines, making it more robust than the others.

Engine Usage and Evaluation Depth

Something that has not been discussed as of yet is depth. When a chess engine is run, the user must decide how far down the decision tree they want it to search - this is called the engine depth. A depth of 10 means the engine analyses 10 moves into the future, and a depth of 30 means the engine analyses 30 moves, etc. This is crucial when comparing studies, as models trained using deeper engine evaluations are likely to generate more reliable assessments.

Regan and Haworth (2011) used Rybka 3 at a depth of 13, which in 2011, was a state-of-the-art engine, and a reasonable depth for analysis. Comparatively, Guid and Bratko (2006) used the slightly weaker engine – Crafty, confirmed by the 2010 World Computer Rapid Chess Championships, where Rybka came in first with an Elo of 2,813, and Crafty came in third with an Elo of 2,665 (ACCA, 2010). Chess engines develop every year, so the engine used in 2006 being weaker than the engine used in 2011 is likely a symptom of the evolution of engines over time. Being the most recent of the three statistical studies, Ferreira (2012) used

the lesser-known engine: Houdini 1.5a at a depth of 20, thus allowing for a lot deeper engine analysis than Regan and Haworth (2011).

Patria et al. (2021) used Stockfish to evaluate, but they did not give information on the version or depth used. Belhoucine and Feng (2025) did not specifically say which engine they used at all, which makes a comparison difficult between the two. Iavich & Kevanishvili (2025) on the other hand, gave the most detailed recital of their engine usage. They explain their reliance on Stockfish (2024) and Leela Chess Zero (Pascutto & Linscott, 2019); the two highest-rated engines in the world, having played each other in the last four TCEC Cup finals (TCEC, 2025). Without question, this paper stands out in its transparency and sophistication of engine use.

Conclusion and Rationale for This Study

In summary, although all six studies contribute something meaningful to the field, there is a clear distinction between the older statistical approaches and the more recent machine learning-based classifiers. The three neural network studies vary in dataset strength and engine utilisation, with Iavich & Kevanishvili’s (2025) study standing out for its comprehensive dataset and advanced engine usage. These insights give an idea of the evolution of cheat detection methods over time, as well as the importance of consistent data sourcing and engine usage in building effective models.

In conclusion, the existing literature surrounding the field of cheating detection in chess, reveals a clear movement from early statistical approaches to more recent machine learning methods. As cheating in chess has become more accessible over the years, this progression is natural and necessary. While foundational studies like Regan & Haworth (Regan & Haworth, 2011) and Guid & Bratko (2006) offer valuable insights into human move-quality, they fall short when it comes to classification. More recent works, such as those by Patria et al. (2021) and Iavich & Kevanishvili (2025), attempt to classify games as one of cheated or not cheated using neural networks. However, gaps still remain: datasets are often unrealistic and poorly labelled, time behaviour is underutilised, and engine specifications are rarely reported. This dissertation aims to address these issues by analysing a large set of high-stakes online games from weekly Titled Tuesday events. By incorporating both Stockfish-based evaluations and timing data and offering a nuanced and reliable model of engine assistance. In doing so, it will build upon existing research, while responding to its most pressing limitations.

Methodology

Research Design and Rationale

This section outlines the methodology adopted to investigate engine-assisted cheating in online chess, with a focus on analysing Titled Tuesday games using both engine evaluations and temporal features. This study adopts an unsupervised approach to detect anomalies in online chess games, largely because of the difficulty in obtaining access to a trustworthy labelled dataset where the “cheater” games are reliably cheated in. As mentioned, supervised

models often struggle to generalise and consistently learn faulty patterns from the misinformed data.

Dataset Collection and Preprocessing

Titled Tuesday was selected as the focus of analysis due to its stance in the middle ground between high-stakes OTB chess, and low-stakes online chess. As seen in the literature review, all existing studies feature a wide variety of games, with some analysing World Championship games (Guid & Bratko, 2006), others analysing casual games from players at different points on the Elo spectrum (Iavich & Kevanishvili, 2025), and a few dealing with OTB tournament games (Regan & Haworth, 2011). All things considered, it is a novel idea to perform research on Titled Tuesday, which is surprising because of how open Chess.com (2025) are with allowing users to access games. In fact, on their website, Chess.com (2025) allow anyone to download every game from every tournament directly to their device.

Additionally, the study amalgamates a traditional engine analysis with temporal features such as time-to-move patterns, and consistency across different phases of the game. Although this has become more popular in 2025 (with Iavich and Kevanishvili (2025) being a recent example), treating time management as a fundamental indicator of fair play breaches is a relatively new, breakthrough concept, and for good reason. Iavich and Kevanishvili (2025) proved its worth, gaining over 98% accuracy in their results, which they put partly down to their temporal analysis. This study sets out to delve deeper into this by running detailed statistical testing on time management as a lone factor. In doing so, nuanced signals could be detected that may signify intermittent or centaur-style cheating – a detail often missed by accuracy-based models.

The dataset analysed in this study consisted of approximately 880,000 games from Chess.com's (2025) weekly Titled Tuesday tournaments, spanning from October 2014 to September 2024. These games feature high-level online competition between titled players, providing a consistent and rich source of data for analysis.

The next step was to source the engine that would be used to run analysis on the dataset. Stockfish (2024) was chosen due to its presence at the top of the Computer Chess Rating Lists (CCRL) ranking list as of March 2025 (CCRL, 2025). Furthermore, Stockfish is one of the only chess engines with built-in Python integration, making it easily accessible and simple to use for analysis locally.

Research Procedure

Engine Analysis Script

Once Stockfish (Stockfish, 2024) was elected and set up, a Python script was created to run every game one by one through the engine, and output key elements for each game, including centipawn-loss, accuracy, and Elo for each player. This was then run on the Edinburgh Napier cutting-edge computer cluster (Edinburgh Napier University, 2023), which outputted the file in around three days. This script was developed based on the open-source repository -

chess_accuracy - made by GitHub user: Asavis (2024) and was adapted to suit the aims of this dissertation. However, the core sections of the script were retained.

Centipawn Evaluation Formatting

For example, the `get_eval_str` function, which formats engine scores into a more readable output, was maintained. It does this by dividing the centipawn score from white's perspective by 100, to return a regular evaluation (e.g. +0.23 or -1.88). It also handles the case where the engine sees a forced checkmating sequence and returns a string like "Mate in 6 for White". This function can be seen in [Figure 7](#).

```
def get_eval_str(score, board):
    if score.is_mate():
        if score.relative.mate() > 0:
            mating_side = "White" if board.turn else "Black"
        else:
            mating_side = "Black" if board.turn else "White"
        return "Mate in " + str(abs(score.relative.mate())) + " for " + mating_side
    else:
        return str(score.white().score() / 100.0)
```

Figure 7 – `get_eval_str` function from code

Move Accuracy Calculation

The next significant function retained from Asavis' (2024) code is the `move_accuracy_percent` function. This function converts changes in winning chances into a percentage accuracy. This is vital, as it reflects the severity of each move error relative to the player's winning chances and therefore handles position significance. If a player were cheating, and there was only one move that led to an advantage, they would always opt for the best move, and this function keeps track of this. The way it works is if the elected move improves or retains winning chances:

$$\text{Accuracy} = 100\%$$

Otherwise:

$$\text{Accuracy} = \min(\max(103.167 \cdot e^{-0.04354 \cdot \Delta W} - 3.167 + 1, 0), 100)$$

where $\Delta W = \text{win probability before} - \text{win probability after}$

If the move improves or retains the evaluation ΔW is negative, and if the move worsens the position ΔW is positive. In the latter case, because chess accuracy is nonlinear, and missing mate in 1 is much worse than playing a move that slightly worsens a pawn structure, a linear model like $\text{Accuracy} = 100 - k \cdot \Delta W$ would not reflect how humans and engines perceive error severity. Exponential decay better models this, which is why the constants are there to shape the curve appropriately. Considering the exponential function:

$$e^{-k \cdot \Delta W}, \quad \text{where } k = -0.04354$$

If ΔW is small, then the exponent is close to 0 ($e^0 = 1$), meaning high accuracy. If ΔW is large, then the exponent becomes very negative ($e^{-large\ number} \approx 0$). In other words, the closer $e^{-k \cdot \Delta W}$ is to 0, the lower the accuracy of the move. As for why $k = -0.04354$, a larger magnitude of k makes accuracy fall off at a faster rate as ΔW increases, and a smaller k makes accuracy more forgiving for poor move choices. $k = -0.04354$ was likely found through curve fitting. Asavis' (2024) may have tested on some real data and adjusted the curve until it matched a distribution of accuracy with respect to Elo. Admittedly, this could have been tested at different values for a more thorough analysis, so it is noted as a potential downside of the project as a whole. Exponential decay was retained as it drops off quickly, meaning that even a minute drop in move quality results in a drop in accuracy, which fits the purpose of attempting to catch engine users.

This leads to the choice of the initial amplitude of exponential decay of 103.167, and a correction offset of $-3.167 + 1$. The amplitude value of 103.167 ensures the output starts slightly above 100%, which allows some room for the offset correction. The subtraction of -3.167 pulls the curve downward so that strong moves align more closely with a score of 100% accuracy, and the addition of 1 is a final tweak for calibration. These constants were all retained from the original script by Asavis (2024), who likely decided on them through curve fitting. Finally, the use of the min and max functions makes sure the resulting accuracy value is always bounded between 0% and 100%. The function can be seen in [Figure 8](#).

```
def move_accuracy_percent(before, after):
    if after >= before:
        return 100.0
    else:
        win_diff = before - after
        raw = 103.1668100711649 * math.exp(-0.04354415386753951 * win_diff) + -3.166924740191411
        return max(min(raw + 1, 100), 0)
```

Figure 8 – move_accuracy_percent function from code

Win Probability Conversion from Centipawn Scores

The winning_chances_percent function was also retained from Asavis (2024) and is responsible for translating centipawn evaluations into a win probability percentage using a logistic transformation. It uses a sigmoid function to output values between 0% and 100% as follows:

$$\text{Winning Chance} = 50 + 50 \cdot \max\left(\min\left(\frac{2}{1 + e^{-0.00368208 \cdot \text{cp}}} - 1, 1\right), -1\right)$$

This causes positions at +0 centipawns to indicate a 50% chance of winning, whereas large positive (white winning) or negative (black winning) values will asymptotically approach 100% or 0%, as seen in the [graphical representation of the function in Figure 9](#) (Desmos, 2025). This transformation is essential, as it allows the system to gain information about the significance of a move error in probabilistic, human terms. The output is then used for the calculation of ΔW in the move_accuracy_percent function. The function can be seen in [Figure 10](#).



Figure 9 – Graphical Representation of winning_chances_percent function

```
def winning_chances_percent(cp):
    multiplier = -0.00368208
    chances = 2 / (1 + math.exp(multiplier * cp)) - 1
    return 50 + 50 * max(min(chances, 1), -1)
```

Figure 10 – winning_chances_percent function from code

Accuracy Averaging Using the Harmonic Mean

The harmonic_mean function was also retained from Asavis' (2024) original script. It calculates the harmonic mean of a set of move accuracies for each player, settling on a

conservative average that punishes low-accuracy moves more heavily than high-accuracy. This is significant in the context of cheat detection, because it emphasises consistency in accuracy over occasional brilliance – something anyone is capable of.

In the code, the harmonic mean is applied separately to each player's accuracy scores (`accuracies_white` and `accuracies_black`). The output is then used to determine the final accuracy score for each player in each game, alongside the volatility-weighted mean. This ensures that a player's final score is not skewed by a small number of accurate moves, if the rest of their play is poor. The function can be seen in [Figure 11](#).

```
def harmonic_mean(values):  
    n = len(values)  
    if n == 0:  
        return 0  
    reciprocal_sum = sum(1 / x for x in values if x)  
    return n / reciprocal_sum if reciprocal_sum else 0
```

Figure 11 – harmonic_mean function from code

Volatility-Weighted Accuracy Averaging

Volatility-weighted mean calculation of all accuracies is also retained from Asavis' (2024) code. Rather than giving each move equal importance, it weights each accuracy based on how volatile the position was at the time of the move. A volatile position is one where the evaluation could shift rapidly depending on the move played, especially when there are only one or two moves that maintain the evaluation score. For each move, the function extracts a two-move window of win chances from the branching positions. It then calculates the standard deviation of these win chances to measure volatility. The more sparsely populated the sub-sequence, the more weight that move gets. Weights have an upper bound of 12 and a lower bound of 0.5 to avoid extreme values. Finally, it calculates the weighted average.

In non-volatile positions, it is far easier to play well without assistance. However, in volatile positions, consistently finding the best move is much harder. This method offers something different to the harmonic mean by considering a volatility-sensitive measure of accuracy. The function can be seen in [Figure 12](#).

```

def volatility_weighted_mean(accuracies, win_chances, is_white):
    weights = []
    for i in range(len(accuracies)):
        base_index = i * 2 + 1 if is_white else i * 2 + 2
        start_idx = max(base_index - 2, 0)
        end_idx = min(base_index + 2, len(win_chances) - 1)

        sub_seq = win_chances[start_idx:end_idx]
        weight = max(min(std_dev(sub_seq), 12), 0.5)
        weights.append(weight)

    weighted_sum = sum(accuracies[i] * weights[i] for i in range(len(accuracies)))
    total_weight = sum(weights)
    weighted_mean = weighted_sum / total_weight if total_weight else 0

    return weighted_mean

```

Figure 12 – volatility_weighted_mean function from code

Final Player Accuracy Metric

The final accuracy metric is calculated by taking the mean of the volatility-weighted mean and the harmonic mean, as seen in [Figure 13](#):

```

accuracy_white = (harmonic_mean_accuracy_white + weighted_mean_accuracy_white) / 2
accuracy_black = (harmonic_mean_accuracy_black + weighted_mean_accuracy_black) / 2

```

Figure 13 – Final accuracy calculation from code

Custom Modifications

There were also parts of the final code that were not taken or inspired by Asavis' (2024) code. Some of these fall into pre-processing and will be explored there, but some of them are appropriate for the methodology section.

Game Metadata Extraction

For example, full metadata extraction was added to the code as seen in [Figure 14](#):

```
# Get game metadata from PGN headers
event = game.headers.get("Event", "Unknown Event")
date = game.headers.get("Date", "Unknown")
round = game.headers.get("Round", "Unknown")
white_player = game.headers.get("White", "Unknown")
black_player = game.headers.get("Black", "Unknown")
white_elo = game.headers.get("WhiteElo", "Unrated")
black_elo = game.headers.get("BlackElo", "Unrated")
game_result = game.headers.get("Result", "Unknown Result")
termination = game.headers.get("Termination", "Unknown")
```

Figure 14 – Metadata extraction from code

This was inserted to allow analysis of all attributes and later add them to a CSV – another novel aspect of the code - as displayed in [Figure 15](#).

```
# Write the results to the CSV file, including move count
csv_writer.writerow([event, date, round, white_player, white_elo, black_player, black_elo, game_result, termination,
f'{avg_cp_loss_white:.0f}', f'{accuracy_white:.0f}',
f'{avg_cp_loss_black:.0f}', f'{accuracy_black:.0f}', move_count])
```

Figure 15 – CSV autowriter from code

Automated CSV Output of Game Features

Instead of printing results to the terminal like in Asavis' (2024) code, a CSV is automatically created and written with important features. The resulting output file is titled `combined_stockfish_results.csv`. A snippet of the file is seen in [Figure 16](#):

A1	Game ID													
	A	B	C	D	E	F	G	H	I	J	K	L	M	N
	Game ID	Date	Round	White Player	White Rating	Black Player	Black Rating	Result	Termination	ACPL (White)	Accuracy (White)	ACPL (Black)	Accuracy (Black)	Move Count
1	Early-Titled-Tuesday-Blitz-June-20-2023	2023.06.20	3	Itt04	2692	gurelediz	2952	0-1	gurelediz won by resignation	32	86	21	88	48
2	Early-Titled-Tuesday-Blitz-June-20-2023	2023.06.20	3	ninguno46	2610	Veleamater	2340	0-1	Veleamater won by resignation	29	89	22	93	66
3	Early-Titled-Tuesday-Blitz-June-20-2023	2023.06.20	3	ArtemDyachuk	2615	DvaHrasta	2450	1-0	ArtemDyachuk won on time	20	91	30	83	43
4	Early-Titled-Tuesday-Blitz-June-20-2023	2023.06.20	3	GoltsevDmitry200C	2795	goofypenguin96	2573	0-1	goofypenguin96 won by checkmate	38	85	18	91	47
5	Early-Titled-Tuesday-Blitz-June-20-2023	2023.06.20	3	OparinGrigoriy	2995	Kev4444444444	2193	1-0	OparinGrigoriy won by resignation	26	89	40	80	53
6	Early-Titled-Tuesday-Blitz-June-20-2023	2023.06.20	3	mr_gustavo	2626	AdrianaNikolovi	2479	1/2-1/2	Game drawn by repetition	32	85	33	86	60
7	Early-Titled-Tuesday-Blitz-June-20-2023	2023.06.20	3	Alenzer	2142	emeli_chess	2375	0-1	emeli_chess won by resignation	43	87	24	91	57
8	Early-Titled-Tuesday-Blitz-June-20-2023	2023.06.20	3	adricalderon8	2423	AndreilTrifan	2544	1-0	adricalderon8 won by checkmate	68	72	83	66	47
9	Early-Titled-Tuesday-Blitz-June-20-2023	2023.06.20	3	LUISMIGUELOV	2127	Jojo_898	2340	0-1	Jojo_898 won on time	53	77	40	86	46
10	Early-Titled-Tuesday-Blitz-June-20-2023	2023.06.20	3	Doraemon7	2735	manmanla1	2498	1/2-1/2	Game drawn by repetition	26	89	26	89	56
11	Early-Titled-Tuesday-Blitz-June-20-2023	2023.06.20	3	eagleclaw07	2674	Durarbayli	2933	0-1	Durarbayli won by resignation	57	80	45	83	53
12	Early-Titled-Tuesday-Blitz-June-20-2023	2023.06.20	3	ELECTRODYNAMIC	2651	madmax011	2333	1-0	ELECTRODYNAMIC_DRACULA won on time	33	84	37	82	50
13	Early-Titled-Tuesday-Blitz-June-20-2023	2023.06.20	3	Goughfather	2004	ChessFanNM	2313	0-1	ChessFanNM won on time	56	78	59	77	48
14	Early-Titled-Tuesday-Blitz-June-20-2023	2023.06.20	3	Alt_rasibod	2700	amintabatabaei	2974	0-1	amintabatabaei won on time	36	82	24	96	51
15	Early-Titled-Tuesday-Blitz-June-20-2023	2023.06.20	3	SRDR290	2374	AttackSparrow	2559	0-1	AttackSparrow won by resignation	19	91	11	95	56
16	Early-Titled-Tuesday-Blitz-June-20-2023	2023.06.20	3	patzer-reloaded	2518	Elsa167	2825	1-0	patzer-reloaded won by checkmate	16	96	28	92	65
17	Early-Titled-Tuesday-Blitz-June-20-2023	2023.06.20	3	queen_jryna	2367	matagordo	2188	0-1	matagordo won by checkmate	51	85	35	90	50
18	Early-Titled-Tuesday-Blitz-June-20-2023	2023.06.20	3	Partytime1126	2624	treshan	2477	1/2-1/2	Game drawn by repetition	35	80	34	80	57
19	Early-Titled-Tuesday-Blitz-June-20-2023	2023.06.20	3	zano2023	2173	pirulillo	2366	0-1	pirulillo won on time	53	82	37	88	59
20	Early-Titled-Tuesday-Blitz-June-20-2023	2023.06.20	3	Nl3c5	2585	the_chess_child	2841	1-0	Nl3c5 won by checkmate	25	90	39	82	49
21	Early-Titled-Tuesday-Blitz-June-20-2023	2023.06.20	3	anon6121824	2863	Just_Lucky_1	2525	0-1	Just_Lucky_1 won by resignation	61	66	50	59	53
22	Early-Titled-Tuesday-Blitz-June-20-2023	2023.06.20	3	conformalfunctor	2302	AttilaTurco	2510	1-0	conformalfunctor won by resignation	29	86	39	81	55
23	Early-Titled-Tuesday-Blitz-June-20-2023	2023.06.20	3	MalyDominator	2408	campochess	2194	1-0	MalyDominator won by resignation	37	86	44	83	71
24	Early-Titled-Tuesday-Blitz-June-20-2023	2023.06.20	3	vi_pranav	3018	JakubPulpan	2715	1-0	vi_pranav won by resignation	15	92	25	87	52
25	Early-Titled-Tuesday-Blitz-June-20-2023	2023.06.20	3	Rikikits	2801	SCIM	2550	1/2-1/2	Game drawn - insufficient material	40	79	41	79	65
26	Early-Titled-Tuesday-Blitz-June-20-2023	2023.06.20	3	TomLiu777	2441	joksjoksi	2621	0-1	joksjoksi won by resignation	64	62	50	81	50
27	Early-Titled-Tuesday-Blitz-June-20-2023	2023.06.20	3	becemix	2607	baki83	2874	1/2-1/2	Game drawn by repetition	23	86	24	87	63
28	Early-Titled-Tuesday-Blitz-June-20-2023	2023.06.20	3	AlexandraSamagar	2307	Better_Chess_Fi	1998	1-0	AlexandraSamaganova won by checkmate	19	89	36	81	58
29	Early-Titled-Tuesday-Blitz-June-20-2023	2023.06.20	3	OriTaichman	2612	MathiasWomac	2449	1-0	OriTaichman won on time	57	77	49	81	51
30	Early-Titled-Tuesday-Blitz-June-20-2023	2023.06.20	3	viZaragatski	2503	killer_dreams	2356	0-1	killer_dreams won on time	53	72	35	83	57
31	Early-Titled-Tuesday-Blitz-June-20-2023	2023.06.20	3	FGHSMN	2876	RandomNoob04	2655	0-1	RandomNoob04 won by resignation	37	78	22	90	55
32	Early-Titled-Tuesday-Blitz-June-20-2023	2023.06.20	3	Philos_85	2314	malakismayil	2521	1/2-1/2	Game drawn by repetition	19	87	17	90	61
33	Early-Titled-Tuesday-Blitz-June-20-2023	2023.06.20	3	Enriquack	2549	Herzog2012	2423	0-1	Herzog2012 won on time	52	76	44	79	58
34	Early-Titled-Tuesday-Blitz-June-20-2023	2023.06.20	3	Rodalguar	2714	GOGIEFF	3013	0-1	GOGIEFF won on time	38	88	25	90	68

Figure 16 – combined_stockfish_results.csv

Multi-Game Batch Processing

In order to achieve this, multi-game looping had to be added so that the script could run through the entire two-gigabyte PGN (Portable Game Notation) file at once, instead of just one game like the original code.

Machine Learning Model for Anomaly Detection

Because of the success found in the field within the literature review, the project also incorporates a machine learning focus in its analysis. An Isolation Forest (Liu et al., 2008) model is adopted to detect potentially suspicious chess games within the combined_stockfish_results.csv file. Using standardised performance features, including accuracy and centipawn loss relative to player Elo, as well as move count, the model detects anomalies which can be analysed.

Initial Game Filtering Criteria

The first thing the Isolation Forest script does is filter the move count of each game, so that only games with between 25 and 100 moves are considered. This is because games with extremely low or high move counts tend to have issues with accuracy calculation, as detailed later in the results section. It also filters out games where the combined accuracy between the two players is less than 170. This is also detailed later, but it ensures that games where one player plays badly are not featured. Furthermore, a filter was put in place to require a minimum Elo of 1900 for both players. This was inserted after the initial model was trained, as in 2017, there were often untitled players that managed to find their way into Titled Tuesday who were dominating the results. The code that performs this is seen in [Figure 17](#):

```
# Filter games
df_filtered = df[
    (df["Move Count"] >= 25) &
    (df["Move Count"] <= 100) &
    ((df["Accuracy (White)"] + df["Accuracy (Black)"]) > 170) &
    (df["White Rating"] >= 1900) &
    (df["Black Rating"] >= 1900)
].copy()
```

Figure 17 – Isolation Forest Filtering

Calculating Performance Relative to Elo

The script also introduces a new constraint: performance relative to Elo. The calculation for this is straightforward. Simply, the accuracy and centipawn loss for each player is divided by their respective ratings to form a balanced evaluation metric. This is put in place to avoid punishing extremely high-rated players for playing with high accuracy, something they are much more likely to do than lower-rated players naturally. For example, a 99% accuracy game from a 2000-rated player is far more suspicious than if it were from a 3000-rated player. By normalising relative to rating, outliers within rating groups are identified, instead of just players capable of playing with a high accuracy in the upper echelon. These are later

chosen as features of the model along with move count. The code for this is seen in [Figure 18](#).

```
# Create Elo-normalised features
df_filtered["White Accuracy vs Elo"] = df_filtered["Accuracy (White)"] / df_filtered["White Rating"]
df_filtered["Black Accuracy vs Elo"] = df_filtered["Accuracy (Black)"] / df_filtered["Black Rating"]
df_filtered["White ACPL vs Elo"] = df_filtered["ACPL (White)"] / df_filtered["White Rating"]
df_filtered["Black ACPL vs Elo"] = df_filtered["ACPL (Black)"] / df_filtered["Black Rating"]
```

Figure 18 – Relative performance calculation

Feature Scaling with StandardScaler

The next section of the Isolation Forest script was feature scaling. This ensures all features are contributing to the anomaly detection equally. By using StandardScaler from scikit-learn (Pedregosa et al., 2011) each feature is transformed to have a mean of 0 and a standard deviation of 1. This is crucial as the raw accuracy vs Elo might be around 0.03, white ACPL vs Elo might be around 0.001, and the move count ranges from 25 to 100. Without this implemented, move count would dominate the model and render the results unusable. The code for this is seen in [Figure 19](#).

```
# Scale features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(df_filtered[features])
```

Figure 19 – Scale Features

Training the Model and Anomaly Detection Setup

Finally, the Isolation Forest model is set up and trained on the data. Isolation Forest is great for detecting anomalies as it isolates rare instances directly. This makes it excellent for detecting subtle performances. This is supported by Liu et al., (2008) who stated that “This paper proposes a fundamentally different model-based method that explicitly isolates anomalies instead of profiles normal points” (p. 413) in their original Isolation Forest research. Since the dataset features nearly 880,000 games, Isolation Forest’s fast and scalable nature makes it perfect for this project. Liu et al., (2008) go on to substantiate this idea, saying that “iForest has a linear time complexity with a low constant and a low memory requirement. To our best knowledge, the best-performing existing method achieves only approximate linear time complexity with high memory usage” (p. 414). Since the project does not have access to a known cheater dataset, the machine learning is unsupervised. Isolation Forest supports this and does not require known anomalies in order to function well (Liu et al., 2008, p. 422).

A contamination value of 0.01 is selected for this model, meaning that the most extreme 1% of the data points will be labelled as anomalies. This is likely too many but is elected purposely so that analysis of common players within the resulting dataset can be identified. A binary system where cheaters are anomalies labelled with -1, and normal players are common and labelled with 1 is incorporated. This section of code is seen in [Figure 20](#):

```

# Train Isolation Forest
model = IsolationForest(contamination=0.01, random_state=42)
df_filtered["Anomaly Score"] = model.fit_predict(X_scaled)

# Extract flagged games
suspicious = df_filtered[df_filtered["Anomaly Score"] == -1]

# Print results
print("Suspicious Games:")
print(suspicious[[
    "Game ID", "White Player", "Black Player", "White Rating", "Black Rating",
    "Accuracy (White)", "Accuracy (Black)", "Move Count", "Anomaly Score"
]])

# Save to CSV
suspicious.to_csv("suspicious_games_from_isolation_forest.csv", index=False)

```

Figure 20 – Isolation Forest Setup and Training

Temporal Analysis

A second script was also coded to extract the time taken to play every move from every game by a certain player. The user can type in the name of a player, and time graphs will be generated. More information on this is found in the analysis section. This was not as large as the main Python script and was run locally. The script is called `time_analysis.py` and can be seen in [Figure 21](#) (without the graph creation (detailed later)):

```

import re
from statistics import stdev
from collections import defaultdict
from datetime import datetime
import matplotlib.pyplot as plt
import numpy as np

player_name = "Fandorine" # change this to analyse any other player
pgn_path = "all_Fandorine_games.pgn"

def extract_time_seconds(clock_str):
    m = re.match(r"(\d+):(\d+):([\d.]+)", clock_str)
    if not m:
        return None
    return int(m.group(1)) * 60 + int(m.group(2)) + float(m.group(3))

with open(pgn_path, encoding='utf-8') as f:
    content = f.read()

games = content.strip().split('\n\n\n')
date_to_coeffs = defaultdict(list)

for game in games:
    if player_name not in game:
        continue

    date_match = re.search(r'\[Date\s+"(\d{4}\.\d{2}\.\d{2})"\]', game)
    if not date_match:
        continue

    date_str = date_match.group(1).replace('.', '-')
    try:
        date_obj = datetime.strptime(date_str, "%Y-%m-%d").date()
    except ValueError:
        continue

    moves = re.findall(r'\{\[%clk ([^}]+)\}', game)

    if f'[White "{player_name}"]' in game:
        times = moves[1::2]
    elif f'[Black "{player_name}"]' in game:
        times = moves[::2]
    else:
        continue

    time_seconds = [extract_time_seconds(t) for t in times if extract_time_seconds(t) is not None]
    if len(time_seconds) < 2:
        continue

    raw_diffs = [abs(a - b) for a, b in zip(time_seconds, time_seconds[1:])]
    diffs = [max(0, d - 1) for d in raw_diffs] # account for 1-second increment

    if len(diffs) >= 2:
        coeff = stdev(diffs)
        date_to_coeffs[date_obj].append(coeff)

```

Figure 21 – *time_analysis.py*

Ethical Considerations

This study adhered strictly to the ethical principles outlined by the School of Computing and the wider School of Computing, Engineering, and the Built Environment (SoC/SCEBE) at Edinburgh Napier University. Due to the absence of research with human participants, the following ethical compliance was straightforward.

The dataset used in the project was wholly comprised of public chess data, available for anyone to download on the Chess.com website (Chess.com, 2025). The games are freely available online and feature usernames instead of actual birth names. There was also no contact made with any of the players featured in the data. In line with the principles of the SoC/SCEBE ethics guide (Edinburgh Napier University, 2025) no names of players are outed as cheaters, even if they are suspected as such during the study. As no personal data was processed, no Data Protection Impact Assessment (DPIA) or formal ethical approval was required.

Data Collection and Preprocessing

Selection and Acquisition of the Dataset

After Titled Tuesday had been decided as the target for the dataset, the process of locally downloading every tournament that took place between October 28, 2014, and September 24, 2024 began. This process was arduous and took around two months to complete. Each file comes in PGN (Portable Game Notation) format – a plain-text file format widely used in chess to store game data in a readable form. Every game has a unique PGN, which details every move played, the players playing, their Elos, and other optional keys. An example PGN can be seen in [Figure 22](#):

```
[Event "Rated blitz game"]
[Site "https://lichess.org/0VK1L37j"]
[Date "2025.04.05"]
[White "tater_magnus"]
[Black "Faceboom"]
[Result "1-0"]
[GameId "0VK1L37j"]
[UTCDate "2025.04.05"]
[UTCTime "12:12:57"]
[WhiteElo "657"]
[BlackElo "673"]
[WhiteRatingDiff "+6"]
[BlackRatingDiff "-6"]
[Variant "Standard"]
[TimeControl "300+3"]
[ECO "D00"]
[Opening "Queen's Pawn Game"]
[Termination "Normal"]

1. d4 d5 2. e3 Nc6 3. Nf3 Nf6 4. Nc3 a6 5. Bd2 Bg4 6. Be2 e6 7. O-O Be7 8. h3 Bh5 9. Bd3 Bxf3 10. Qxf3 O-O 11. e4 e5 12. Be3 Nb4 13. Rad1 Qd7 14. dxe5 Ne8 15. exd5 c6 16. Bf5 Qd8 17. d6 Bxd6 18. exd6 Nxd6 19. Bd3 Qd7 20. Bc5 Rae8 21. Bxd6 Qxd6 22. a3 Nd5 23. Nxd5 cxd5 24. c4 Re5 25. b4 b5 26. c5 Qc7 27. g3 a5 28. bxa5 Qxc5 29. Rfe1 Rfe8 30. Bxb5 Qxb5 31. Qxd5 Rxd5 32. Rxd5 Qxd5 33. Rxe8# 1-0
```

Figure 22 – example PGN file architecture

Combining and Preparing Game Files

Each tournament file has every game played in every round of every tournament (with the average number of games per tournament being 1440.5) – all back-to-back. Once downloaded, all tournament data was combined into one file using a Python script. The

resulting file contained every game from every tournament, and finished with a size of just over 2GB, and around 880,000 games.

At this point, the dataset was nearly ready and looked like the following ([Figure 23](#)):

```
[Event "Titled-Tuesday-Blitz-September-28-2021"]
[Site "Chess.com"]
[Date "2021.09.28"]
[Round "1"]
[White "swimmerchess"]
[Black "volkanyirik"]
[Result "1-0"]
[WhiteElo "2812"]
[BlackElo "2416"]
[TimeControl "180+1"]
[EndTime "10:01:25 PDT"]
[Termination "swimmerchess won by resignation"]

1. e4 {[%clk 0:03:00.9]} 1... c5 {[%clk 0:02:58.5]} 2. Nf3 {[%clk 0:03:00.2]}
2... d6 {[%clk 0:02:58.9]} 3. d4 {[%clk 0:03:00.6]} 3... cxd4 {[%clk 0:02:59.8]}
4. Nxd4 {[%clk 0:03:01.5]} 4... Nf6 {[%clk 0:03:00]} 5. Nc3 {[%clk 0:03:02.1]}
5... e6 {[%clk 0:03:00.4]} 6. Be2 {[%clk 0:03:01.3]} 6... a6 {[%clk 0:02:59.7]}
7. O-O {[%clk 0:03:01]} 7... Qc7 {[%clk 0:03:00.1]} 8. Be3 {[%clk 0:03:00.8]}
8... b5 {[%clk 0:03:00.3]} 9. a4 {[%clk 0:02:53.4]} 9... b4 {[%clk 0:02:59.6]}
10. Na2 {[%clk 0:02:52.4]} 10... Nxe4 {[%clk 0:02:56.2]} 11. Nxb4 {[%clk
0:02:51.4]} 11... d5 {[%clk 0:02:56.2]} 12. Nd3 {[%clk 0:02:49.3]} 12... Bd6
{[%clk 0:02:56.1]} 13. f4 {[%clk 0:02:48.5]} 13... O-O {[%clk 0:02:52.3]} 14.
Bf3 {[%clk 0:02:45.4]} 14... Bb7 {[%clk 0:02:52.4]} 15. Qe1 {[%clk 0:02:43.7]}
15... Nd7 {[%clk 0:02:51.8]} 16. Qh4 {[%clk 0:02:42.9]} 16... Be7 {[%clk
0:02:50.1]} 17. Qxe7 {[%clk 0:02:41.9]} 1-0

[Event "Titled-Tuesday-Blitz-September-28-2021"]
[Site "Chess.com"]
[Date "2021.09.28"]
```

Figure 23 – Finished dataset with clock times

Clock times were removed for the main script using a separate piece of code, but were kept for the secondary script that was for analysing time usage.

Local Runtime Estimation and University Cluster

The code was written in such a way that all a user needs to do to run it is have a PGN file saved in the same directory as the Python file, and then type a line like the following ([figure 24](#)) into the terminal.

```
# python chess_accuracy.py 16 2 ./stockfish/stockfish-windows-x86-64-avx2 -file="combined_titled_tuesday.pgn" -verbose
```

Figure 24 – Example Input

When this was run locally, the script was timed at taking 30.76 seconds to run six games. This meant that to run all games, it would take 4,197,340 seconds, or 48.6 days. This was not going to be feasible, so the Edinburgh Napier cluster (2025) was accessed to perform batch processing to speed up the process. After the batch processing was set up using an “sbatch” command in Linux, it was still going to take slightly too long using a Stockfish (2024) depth of 16 or 17, so the depth was lowered to 13.

With an estimated Elo of 2459 (estimated in the paper - *The Impact of the Search Depth on Chess Playing Strength*) (Ferreira, 2013), depth 13 is sufficient to approximate the play of strong titled players. The depth Elo table can be seen in [Figure 25](#).

Depth d_1	$d_1 - 20$	$(d_1 - 20) \times \eta_{(20)}$	Strength (Elo)	95% conf. int.
20	0	0	2894	[2859, 2929]
19	-1	-66	2828	[2786, 2868]
18	-2	-133	2761	[2714, 2807]
17	-3	-199	2695	[2642, 2745]
16	-4	-265	2629	[2570, 2684]
15	-5	-331	2563	[2498, 2623]
14	-6	-398	2496	[2426, 2562]
13	-7	-464	2430	[2354, 2500]
12	-8	-530	2364	[2282, 2439]
11	-9	-596	2298	[2209, 2378]
10	-10	-663	2231	[2137, 2317]
9	-11	-729	2165	[2065, 2255]
8	-12	-795	2099	[1993, 2194]
7	-13	-861	2033	[1921, 2133]
6	-14	-928	1966	[1849, 2071]

Figure 25 – Elo versus depth table from Ferreira (2013)

Furthermore, Titled Tuesday is a blitz tournament – with games of length three minutes for each player, with an additional second every time each player plays a move – which does not elicit the strongest play compared to longer time formats. This claim is supported by Van Harreveld et al., (2007) who state that “once players are forced to play faster, their ability during regular play under normal time controls becomes less predictive of their performance. They go on to posit, “this effect is particularly apparent when the players engage in a game of *blitz* or even faster playing tempos.” Because of this, very high depth levels are not necessary to find perfect, and hence suspicious, play.

Final Dataset

After being run through the cluster (Edinburgh Napier University, 2023) over three days, the data was extracted to a local device. Here, the processed data was run on several scripts to generate suspicious games.

Data Analysis, Results and Discussion

Initial Accuracy-Based Analysis

The first script run on the data had the purpose of identifying the 100 games where one of the players exhibited one of the 100 highest accuracies across the entire dataset ([Appendix A](#)). The output of this featured a large number of games similar to the following ([figure 26](#)):

Early-Titled-Tuesday-Blitz-August-15-2023,2023.08.15,5.0,Piotrek1979,2511.0,toniv1,2370.0,0-1,toniv1 won on time,4,99,2,100,2,100

Figure 26 – Low move count game identified

Identifying Top 100 High-Accuracy Games

Due to the extremely low move count of two, this game is not useful, and certainly not an example of cheating. Games with very few moves are fairly common in high-level chess. They usually occur for one of two reasons. Firstly, because either one of the players is not present for the game and their time runs out before they can play, and secondly, because the players agree to a draw in the first few moves of the game. These situations lead to very high accuracy, as the engine is only analysing opening moves (known as book moves), which are pre-prepared at a high level. Because of this, the code was appended so that it filtered out games of under 21 moves. The choice of 21 was somewhat arbitrary but has some logic to it. The mean length of games in the dataset was 43.9, so choosing a value below this, which is above the predicted average length of the opening stage of a chess game of five (Walczak, 1996) was strategic. This ensures strange outlier games like [Figure 26](#) are left out. The updated code can be seen in [Appendix B](#).

This achieved much more fruitful results and displayed 100 games where at least one of the players managed 100% accuracy over games of varying move counts. The lowest move count in the top 100 was 24 moves ([Figure 27](#)), which justifies the cut-off of 21 moves. The highest was one game with 280 moves ([Figure 28](#)).

Late-Titled-Tuesday-Blitz-June-20-2023,2023.06.20,4.0,santosalberto1987,2490.0,IngOscarArdila,2272.0,1-0,santosalberto1987 won by checkmate,2,100,38,87,24,100

Figure 27 – Lowest move count game in top 100

Late-Titled-Tuesday-Blitz-May-02-2023,2023.05.02,10.0,SCIM,2551.0,Championnn-9,2398.0,1-0,SCIM won by checkmate,2,100,4,97,280,100

Figure 28 – Highest move count game in top 100

Case Studies of Edge Cases

Short Game: Santosalberto1987 vs IngOscarArdila

By looking at the actual game played by [Santosalberto1987 \(2490 Elo\)](#) and [IngOscarArdila \(2272 Elo\)](#) ([Figure 27](#)) ([Appendix C](#)), one of the most common types of games that is featured in the dataset can be observed. Santosalberto1987 plays what is known as a miniature. This refers to a short game that sees one of the players attacking well in the early stages of the game, leading to a forced checkmate. In this game, Santosalberto1987 plays into Petrov's Defence: Classical Attack, Mason Showalter Variation – a rare opening which white was prepared for but black was not. Apart from the slight inaccuracy on move 13, where

white played nf1 instead of the engine's preferred move of ne5 (drops 0.24 centipawns but retains a 100 centipawn advantage), Santosalberto1987 plays perfectly in line with the engine's recommendations for the entire game. However, it could easily be something the player has studied before and is not necessarily a sign that they cheated. After black's mistakes of g5 on move 14, and nd7 on move 17, white's optimal moves are not overly difficult to find for a player of their level, so it is not unimaginable that they found them fairly. While Santosalberto1987 could have been cheating, it would be very unfair to say they were, which encourages further alteration of the script.

Long Game: Scim vs Championnn-9

The 280-move game played between [Scim \(2551 Elo\)](#) and [Championnn-9 \(2398 Elo\)](#) ([Figure 28](#)) ([Appendix D](#)) is also very interesting and reveals a flaw in the script. When analysed using Lichess's free-to-use engine tool (Lichess, 2025), which uses Stockfish at a depth of up to 99, Scim's moves match the engine's moves exactly for the first 15 moves of the game, by which point they have built up a considerable advantage of +5.7 by winning black's bishop on move 15. The rest of the game sees Championnn-9 not resigning despite being in a considerably worse position. Scim takes advantage of this and trolls his opponent by dragging the game out to 280 moves, choosing not to checkmate for most of the game, even though they undoubtedly knew they could. Because every move Scim played maintained or boosted their advantage, even if it was not playing checkmate where it was available, the 250+ moves where they were playing moves to prolong the game dominated the engine's evaluation and led to the accuracy score of 100%. It may seem suspicious that the first 15 moves were engine choices, but this is common at top-level chess, as players have access to engines while practicing and can learn the best openings in certain positions to get an advantage. There is inconclusive evidence to suggest Scim was cheating during the opening and middlegame, but it is very unlikely that engine-assistance was used in the endgame where Championnn-9 was waiting to be checkmated. This warrants further tweaking of the code to not include extremely long games in the top 100.

Refinement of Filtering Criteria

Adjusting Move Count Range

The code was updated so that it would change the move count range from 21+ moves to between 25 and 100 moves. This was put in place to filter out a few miniatures at the lower bound, and games dominated by easy moves at the upper bound, as seen in [Appendix E](#). A snippet of the output can be seen in [Figure 29](#):

```

1 Game ID,Date,Round,White Player,White Rating,Black Player,Black Rating,Result,Termination,ACPL (White),Accuracy (White),ACPL (Black),Accuracy (Black),Move Count,Total Accuracy
2 *** Titled Tuesday Blitz,2020.10.20,10.0,MokshanovAlexey,2582.0,ChessWarrior7197,2978.0,1/2-1/2,Game drawn - insufficient material,3,99,2,100,100,199,100
3 Early-Titled-Tuesday-Blitz-November-14-2023,2023.11.14,9.0,santosalberto1987,2565.0,Zohid6,2779.0,1/2-1/2,Game drawn by repetition,2,99,2,100,91,199,100
4 Early-Titled-Tuesday-Blitz-January-30-2024,2024.01.30,10.0,MagnusCarlsen,3299.0,HansOnTwitch,3214.0,1/2-1/2,Game drawn by repetition,1,100,1,100,87,200,100
5 Early-Titled-Tuesday-Blitz-March-21-2023,2023.03.21,11.0,Staudenhocker43,2548.0,vinniethepooh,2826.0,1/2-1/2,Game drawn - insufficient material,2,99,2,100,80,199,100
6 Late-Titled-Tuesday-Blitz-November-08-2022,2022.11.08,4.0,Msb2,2914.0,attack2mateU,2736.0,1/2-1/2,Game drawn by repetition,2,99,1,100,68,199,100
7 Early-Titled-Tuesday-Blitz-January-09-2024,2024.01.09,7.0,Rud_Makarian,3093.0,rasmussvane,3030.0,1/2-1/2,Game drawn - insufficient material,3,99,2,100,68,199,100
8 Early-Titled-Tuesday-Blitz-January-30-2024,2024.01.30,7.0,FREDERICKtheMATE,2588.0,Hovik_Hayrapetyan,2775.0,1/2-1/2,Game drawn by repetition,1,100,1,100,68,200,100
9 Early-Titled-Tuesday-Blitz-March-21-2023,2023.03.21,4.0,Msb2,3047.0,FormerProdigy,2926.0,1/2-1/2,Game drawn by repetition,2,99,1,100,62,199,100
10 *** Titled Tuesday Blitz,2020.04.14,6.0,Spalir,2602.0,theDJoker98,2461.0,1/2-1/2,Game drawn by repetition,2,99,2,100,60,199,100
11 Early-Titled-Tuesday-Blitz-January-17-2023,2023.01.17,5.0,Hanging_pointer,2509.0,politePlayer,2233.0,1/2-1/2,Game drawn by repetition,2,100,2,99,59,199,100
12 Late-Titled-Tuesday-Blitz-August-22-2023,2023.08.22,3.0,GaiozNigalidze,2697.0,dropstoneDP,2973.0,1/2-1/2,Game drawn by repetition,2,99,2,100,58,199,100
13 Early-Titled-Tuesday-Blitz-October-04-2022,2022.10.04,10.0,SaiHanThiha,2584.0,chessllemmo,2446.0,1/2-1/2,Game drawn - insufficient material,3,99,2,100,58,199,100
14 Early-Titled-Tuesday-Blitz-June-21-2022,2022.06.21,11.0,CSB7,2748.0,Fandorine,2895.0,1/2-1/2,Game drawn - insufficient material,3,99,2,100,55,199,100
15 *** MasterClass Titled Tuesday Blitz,2017.12.12,9.0,GMakobianSTL,2686.0,daro94,2671.0,1/2-1/2,Game drawn by repetition,2,100,1,100,55,200,100
16 Early-Titled-Tuesday-Blitz-August-08-2023,2023.08.08,1.0,Bullet_Jachym08,2508.0,dropstoneDP,3025.0,0-1,dropstoneDP won by resignation,12,88,2,100,54,188,100
17 Early-Titled-Tuesday-Blitz-March-19-2024,2024.03.19,6.0,Jospem,3068.0,Hikaru,3279.0,1/2-1/2,Game drawn by repetition,3,99,2,100,49,199,100
18 Titled-Tuesday-Blitz-July-27-2021,2021.07.27,4.0,Russian_berezka,2624.0,OK97,2738.0,1/2-1/2,Game drawn by repetition,3,99,2,100,48,199,100
19 Late-Titled-Tuesday-Blitz-January-09-2024,2024.01.09,5.0,dropstoneDP,3011.0,GMKrikor,2745.0,1/2-1/2,Game drawn by repetition,2,99,2,100,47,199,100
20 Early-Titled-Tuesday-Blitz-July-26-2022,2022.07.26,7.0,DenLaz,3021.0,Hikaru,3178.0,1/2-1/2,Game drawn by repetition,2,100,2,100,46,200,100

```

Figure 29 – Updated top 100 output

Excluding Drawn Games

The unusual thing about these results is that there appear to be many drawn games in the extracted data. In fact, it was found that out of these 100 games, 65 ended in a draw. To understand why this is, one of the games is used as a case study. The game in question was played between [MagnusCarlsen \(3299 Elo\)](#) and [HansOnTwitch \(3214 Elo\)](#) ([figure 30](#)) ([Appendix F](#))

```

Early-Titled-Tuesday-Blitz-January-30-2024,2024.01.30,10.0,MagnusCarlsen,3299.0,HansOnTwitch,3214.0,1/2-1/2,Game drawn by repetition,1,100,1,100,87,200,100

```

Figure 30 – Example of drawn game with high accuracy

This game sees the two players exchanging a few pieces early on and not trying to go for an attack, which is a defensive strategy usually and leads to drawish (positions likely leading to a draw) positions. After a huge exchange of pieces on move 25, the players are left in a king and pawns endgame, where it is impossible for either side to forge an advantage, granted that neither player makes a crazy mistake. The rest of the game features the players dancing around their territories with their kings, until move 87, where the game ends because of the position being repeated three times – a common way to draw in high-level chess. Because there is nothing better to do than make aimless king moves between moves 25 and 87, and both players made it impossible for the other to break through in the opening and middlegame, every move played was optimal with a low level of difficulty, leading to accuracy scores of 100% for each player.

Fixing this issue in the script is simple. Since engines are much stronger than humans nowadays, if one player is cheating, then the game should lead to a decisive result in most cases. Because of this, drawn games can be filtered out, as seen in [Appendix G](#).

Suspicion Case Study: Frogo47

After the code in [Appendix G](#) was applied, the resulting dataset was promising, and displayed 100 games, which all (on the surface) look like they could have been cheated in.

One game in this dataset was played between [Frogo47 \(2492 Elo\)](#) and [Vohiraa \(2218 Elo\)](#) ([Figure 31](#)) ([Appendix H](#)).

```

*** Titled Tuesday Blitz,2020.04.28,8.0,Frogo47,2492.0,vohiraa,2218.0,1-0,Frogo47 won by checkmate,2,100,37,88,30,188,100

```

Figure 31 – Example of potential cheating game

Game-Level Evaluation

Using Chess.com’s game review (2025) feature In [Figure 32](#), it can be observed that Frogo47 played 11 book moves (theoretical opening moves), 16 best moves (the best move in a position), and 4 brilliant moves (difficult moves to see that are also the best move).



	White		Black
Players			
Accuracy	100.0		85.7
Brilliant	4	!!	0
Great	0	!	0
Best	16	★	8
Excellent	0	👍	5
Good	0	✓	2
Book	11	📖	10
Inaccuracy	0	?!	1
Mistake	0	?	1
Miss	0	✗	0
Blunder	0	??	0

Figure 32 – Chess.com Game review of Frogo47 vs Vohiraa

This indicates perfect play. When looking at the game itself, while white found themselves at an advantage of roughly +3 after black’s mistake of hxcg5 on move five, the game was still complex and not routine to win after this point. The probability of seeing four brilliant moves in a single game is very low. It is an outlier behaviour worth flagging, even if not conclusive evidence on its own. It should be said that it is still unlikely that white cheated in this game, as they are a very high-rated player and are more than capable of playing perfect games occasionally, but it is certainly an example of a game that could have been cheated in.

This suspicion warrants the investigation of all Frogo47’s games in the dataset, and is executed in the code in [Appendix I](#). This creates a sub-dataset of games only played by Frogo47 and sorts by accuracy.

Tournament-Level Accuracy Evaluation

To interpret their performance, a scatter plot was created that demonstrates Frogo47’s accuracy by tournament over time, as seen in [Figure 33](#):

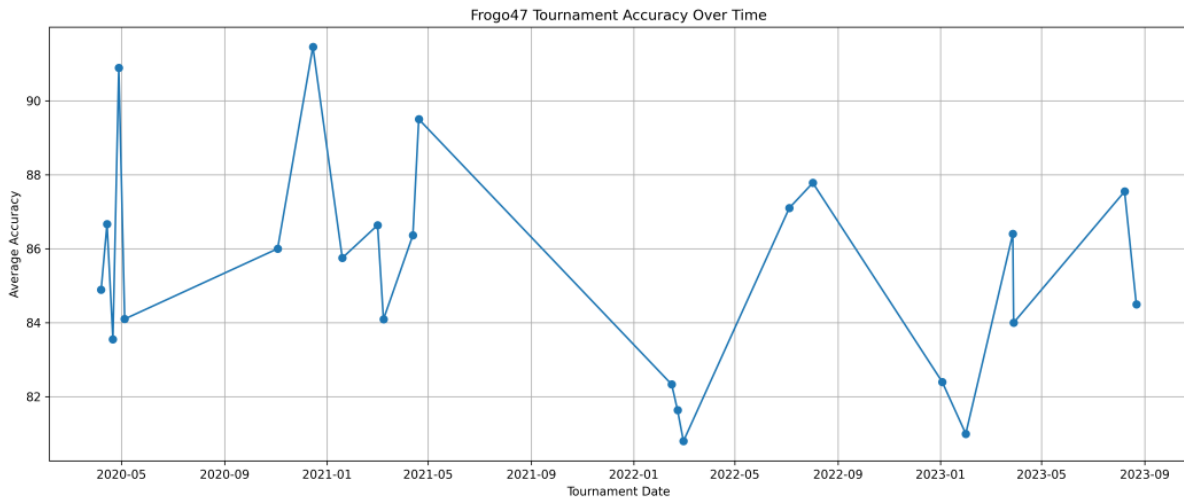


Figure 33 – Frogo47 Tournament Accuracy Over Time

As seen in the plot, Frogo47 displayed an average accuracy of over 90% in one tournament in May 2020, and noticeably in a tournament in December 2020 where their accuracy was roughly 98%. These spikes are clear outliers when compared to the player’s average tournament accuracy, which typically hovers around 85–89%. This suggests that in those high-accuracy tournaments, Frogo47 either performed exceptionally well or the tournament format/opposition may have allowed for easier victories. They also had an average tournament accuracy between 80% and 82% three times – twice in 2022, and once in 2023. The tournament with an average accuracy of 98% in December 2020 stands out. While this level of precision is possible at the top level, it also warrants further review for signs of engine assistance.

Temporal Analysis of Player Behaviour

Taking a suspicious player like Frogo47 and analysing games one by one would be a valid method of moving forward, but temporal analysis can also be performed on all their games as an alternate method. As discussed in the literature review, with the assistance of works by Iavich and Kevanishvili (2025) specifically, using analysis of the time taken to play each move is an appropriate way to identify breaches in fair use.

Time-Based Suspicion Modelling

Standard Deviation of Move Times

In this project, a script was written to calculate the standard deviation of the time taken to play each move for all given tournaments. This is useful as normal players will take much longer to assess which move is best in critical positions, but will play easier moves much quicker. Therefore, a high standard deviation of move times, i.e. taking different amounts of time to play moves across a game, is indicative of fair play. Conversely, a low standard deviation is typical of engine play, as players take the same amount of time every move to read from their other tab or device, and play the optimal move.

ANTSC: Average Normalised Time Suspicion Coefficient

In addition to the standard deviation of move times, an average normalised time suspicion coefficient (ANTSC) was computed for each tournament. This coefficient was derived by normalising the standard deviation of move times and inverting them, for a more accurate comparison of their performance across different games. By comparing the deviation in move times to their own averages, this coefficient accounts for a player's natural playing style and reduces the influence of personal variability.

Frogo47's Temporal Trends

To use in comparison with the plot in [Figure 33](#), plots were created for Frogo47's average standard deviation per tournament and ANTSC per tournament. This can be seen in [Figures 34 and 35](#).

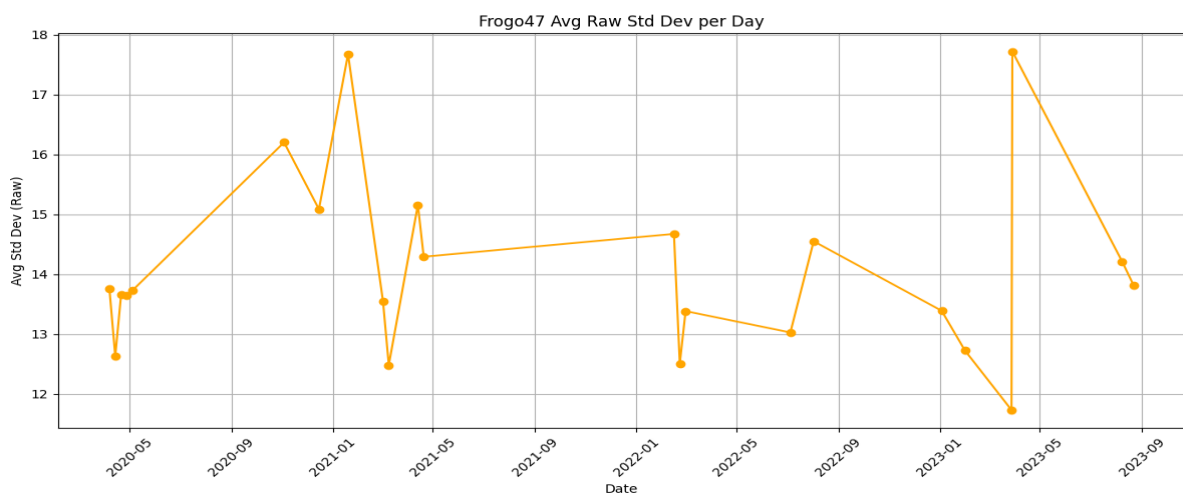


Figure 34 – Frogo47 Average Standard Deviation across tournaments

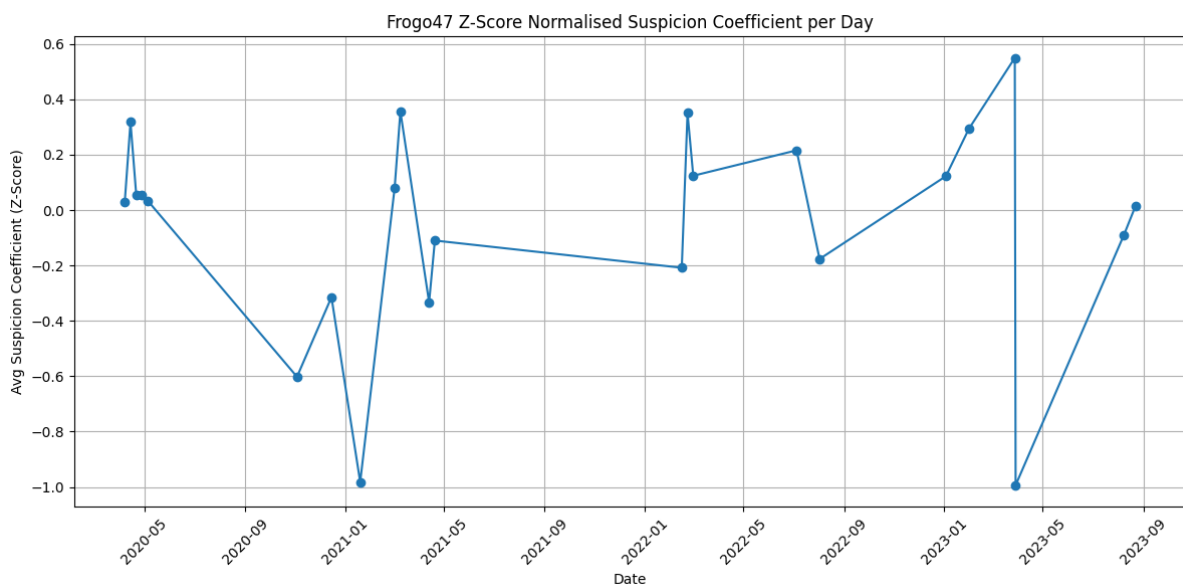


Figure 35 – Frogo47 ANTSC across tournaments

The two graphs provide insights into Frogo47's playstyle over time. With noticeable peaks of standard deviations of nearly 18 seconds in a tournament in January 2021, and one in April 2023, from [Figure 34](#), they probably were playing more erratically during this time, potentially taking a long time to play certain moves and quickly playing others. There are also troughs in April 2020, March 2021, March 2022, and April 2023 of around 12 seconds, indicating lower standard deviations. However, 12 seconds still seems relatively high and does not support the argument of engine usage.

The graphs can also be compared with the accuracy chart in [Figure 33](#). When Frogo47 played at a particularly high average accuracy in December 2020, their ANTSC was very normal at around -0.3. As this is close to 0, it suggests they were not playing in a different temporal style than usual, and perhaps just had a great day. To get confirmation of this, a random sample of 5 players from the top 100 dataset were selected and used in comparison with Frogo47.

Comparison Across Players

The temporal analysis graphs of five randomly selected players – Magila31, TC2304, eljanov, Fandorine, and bascheyaro from the top 100 dataset are seen below:

Selected Player Time Profiles

Magila31

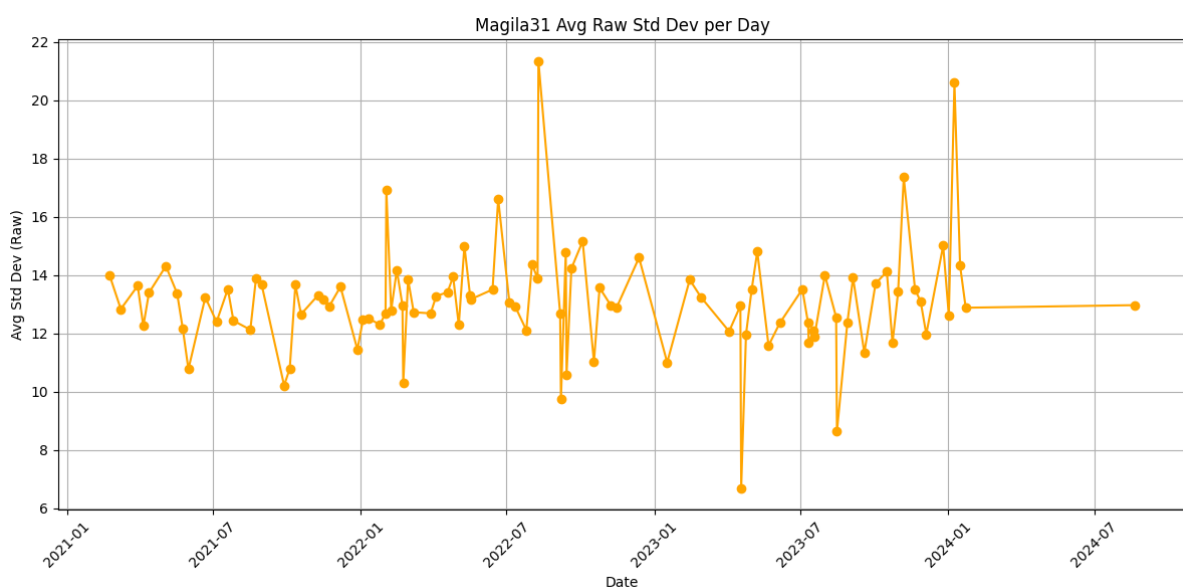


Figure 36 - Magila31 Average Standard Deviation across tournaments

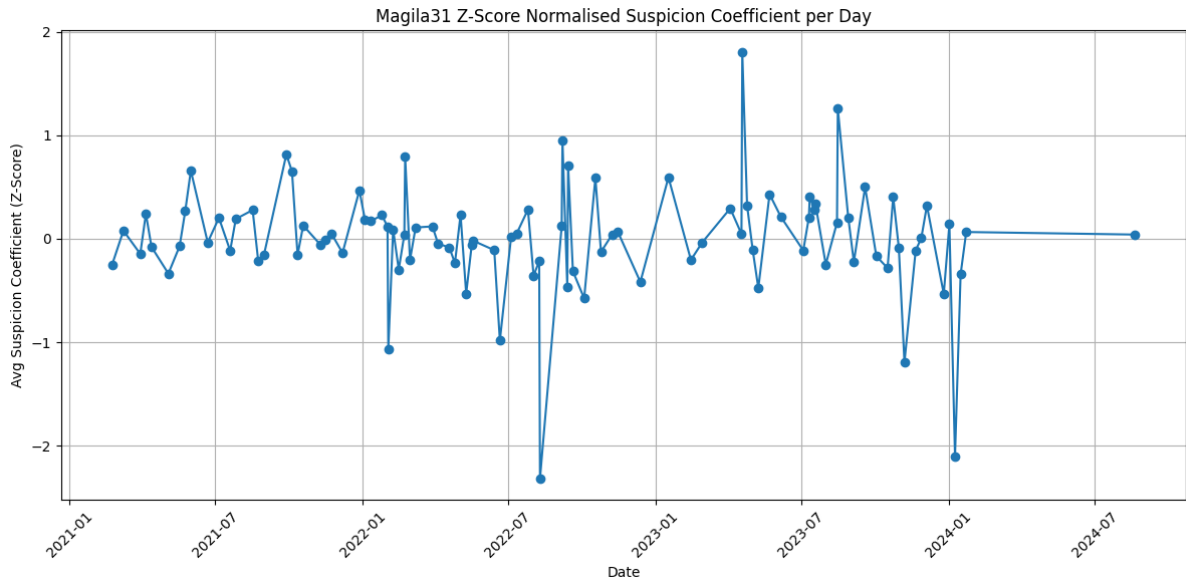


Figure 37 – Magila31 ANTSC across tournaments

TC2304

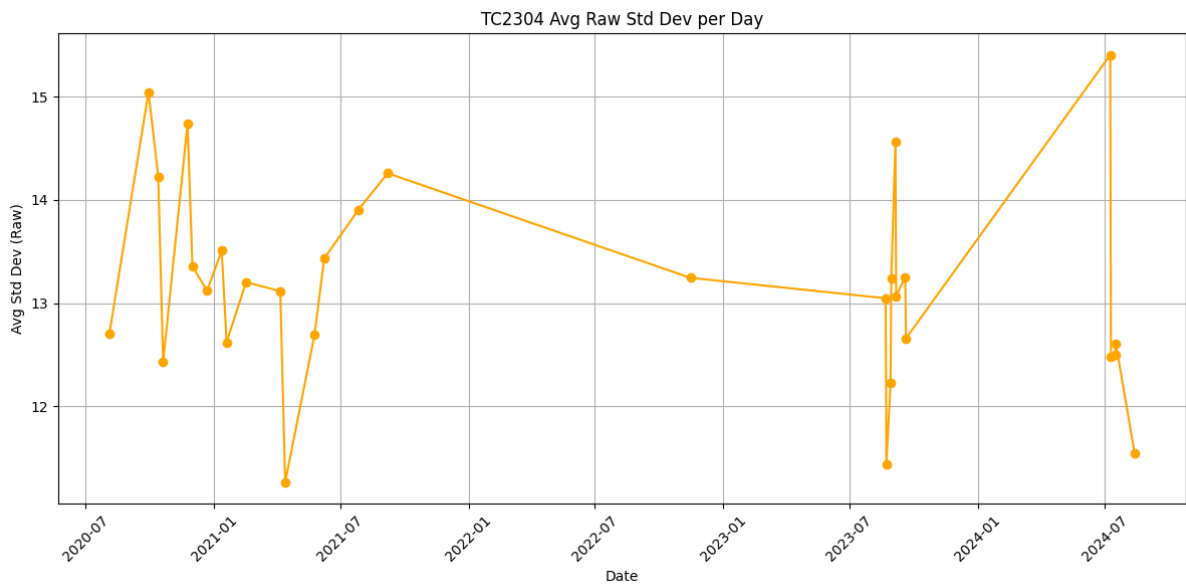


Figure 38 -TC2304 Average Standard Deviation across tournaments

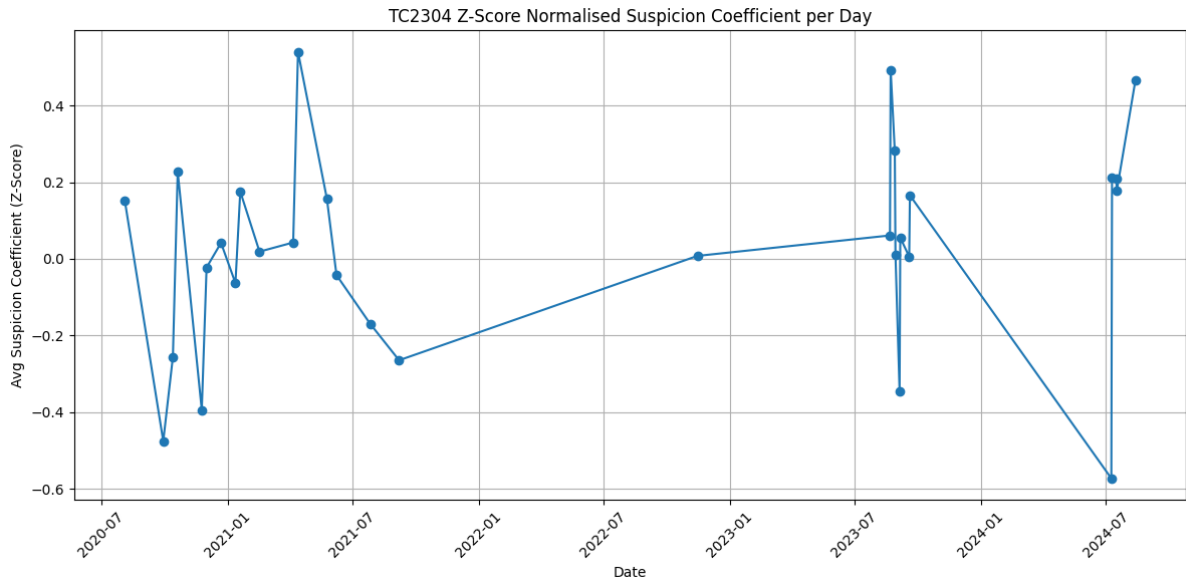


Figure 39 – TC2304 ANTSC across tournaments

eljanov

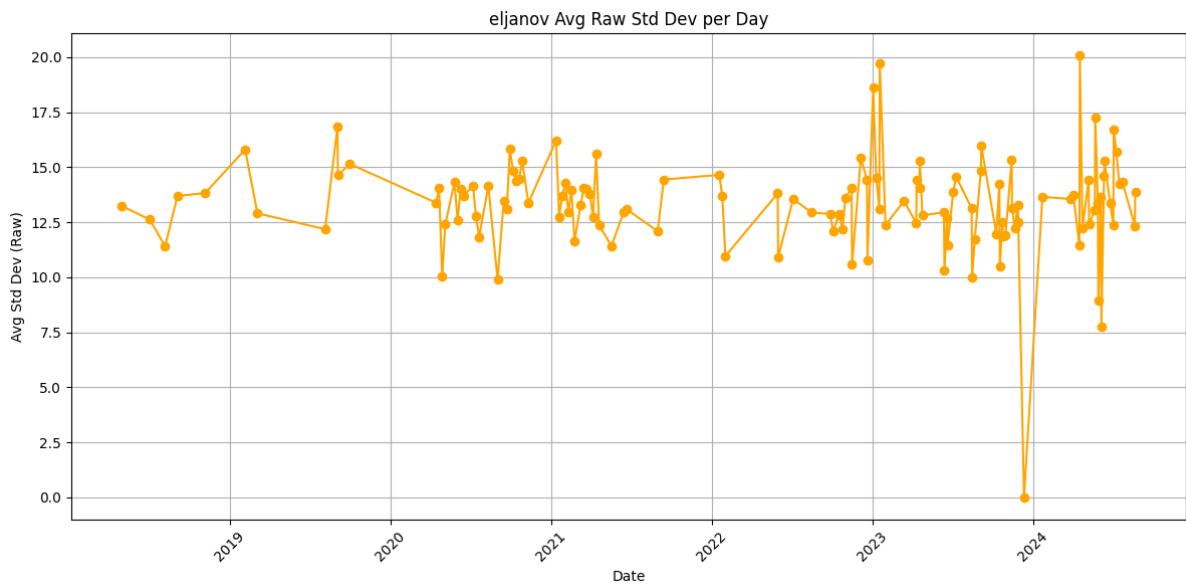


Figure 40 - eljanov Average Standard Deviation across tournaments

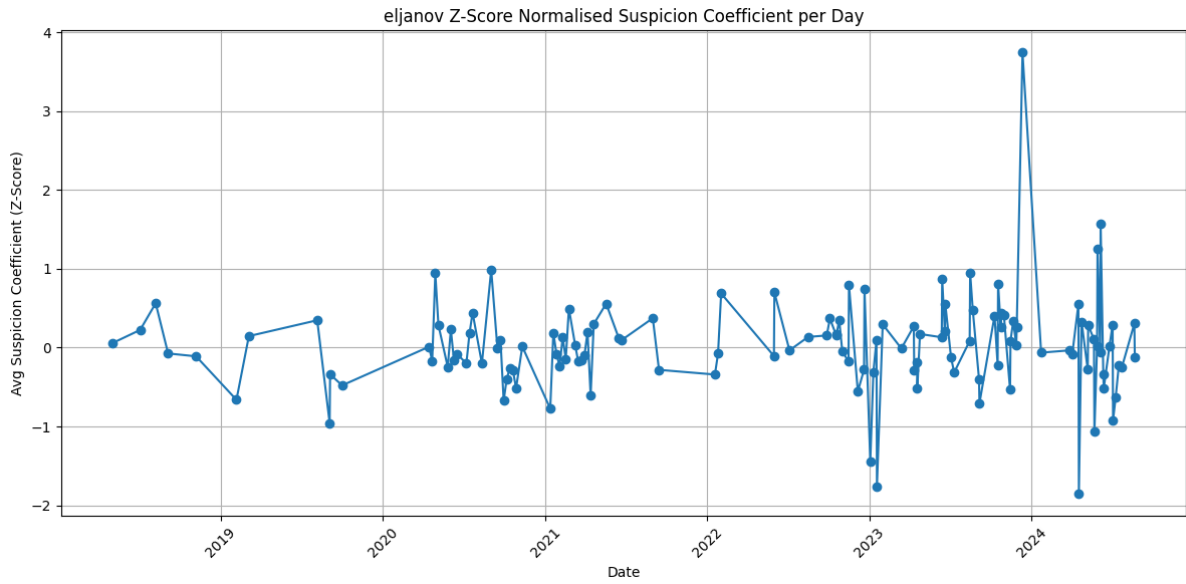


Figure 41 - eljanov ANTSC across tournaments

Fandorine

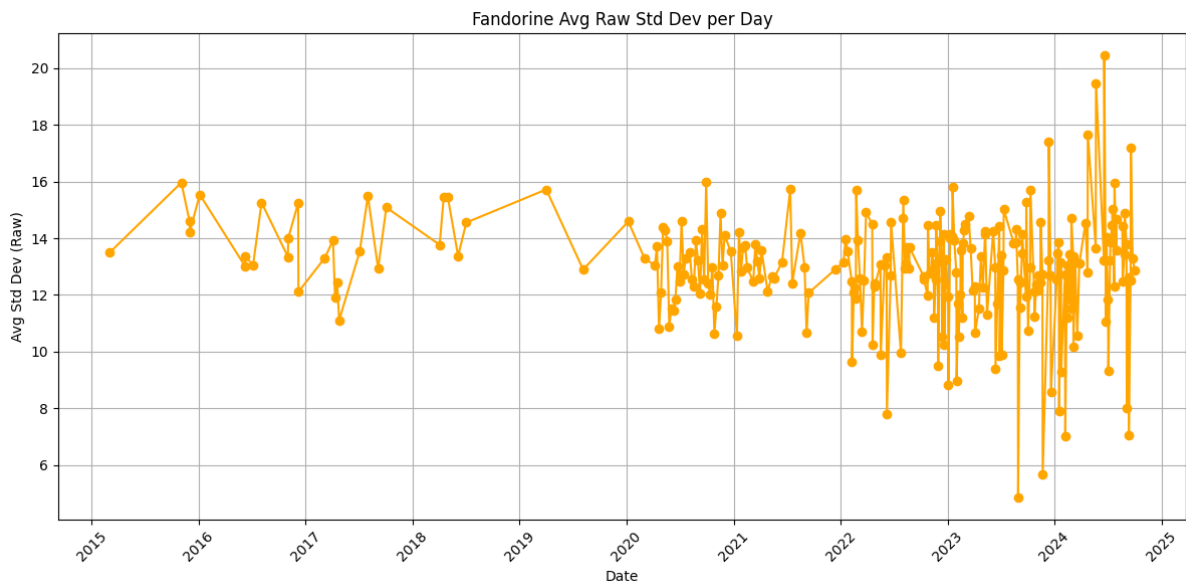


Figure 42 - Fandorine Average Standard Deviation across tournaments

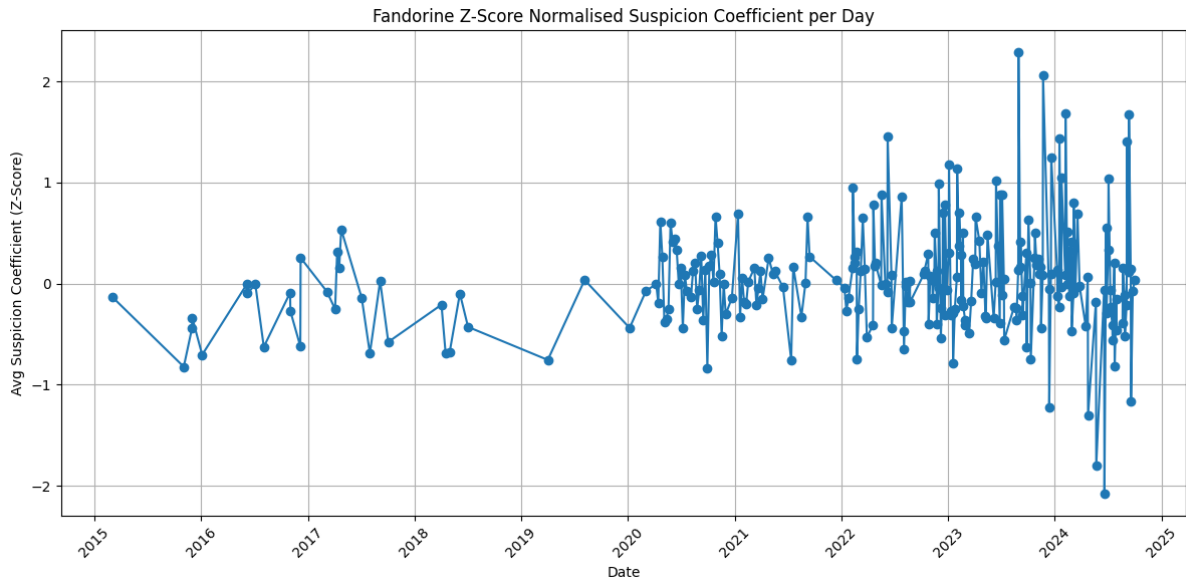


Figure 43 - Fandorine ANTSC across tournaments

bascheyaro

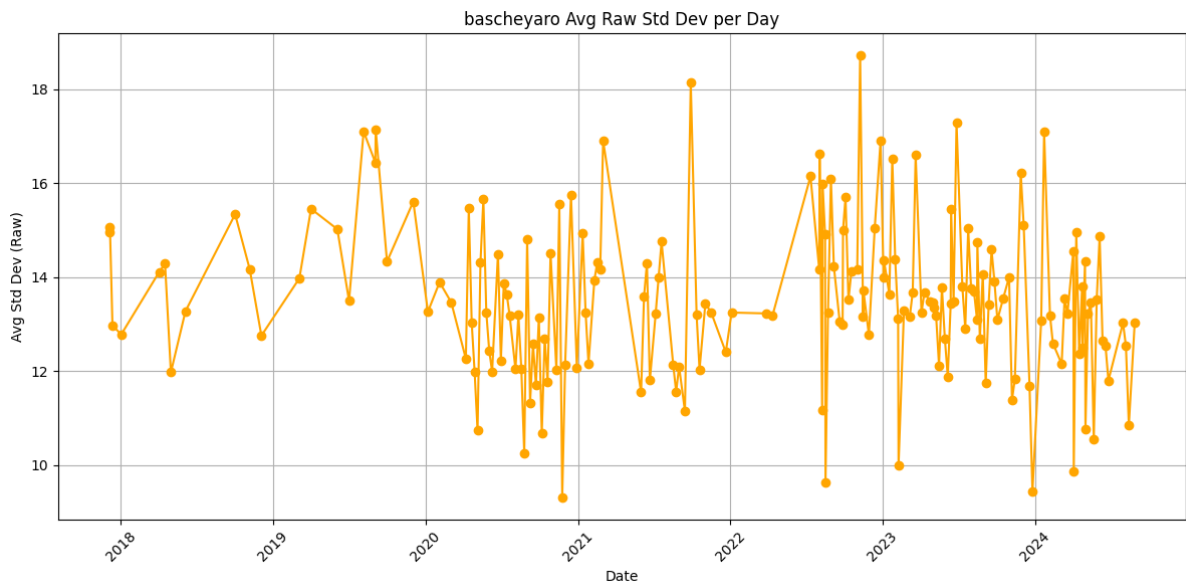


Figure 44 - bascheyaro Average Standard Deviation across tournaments

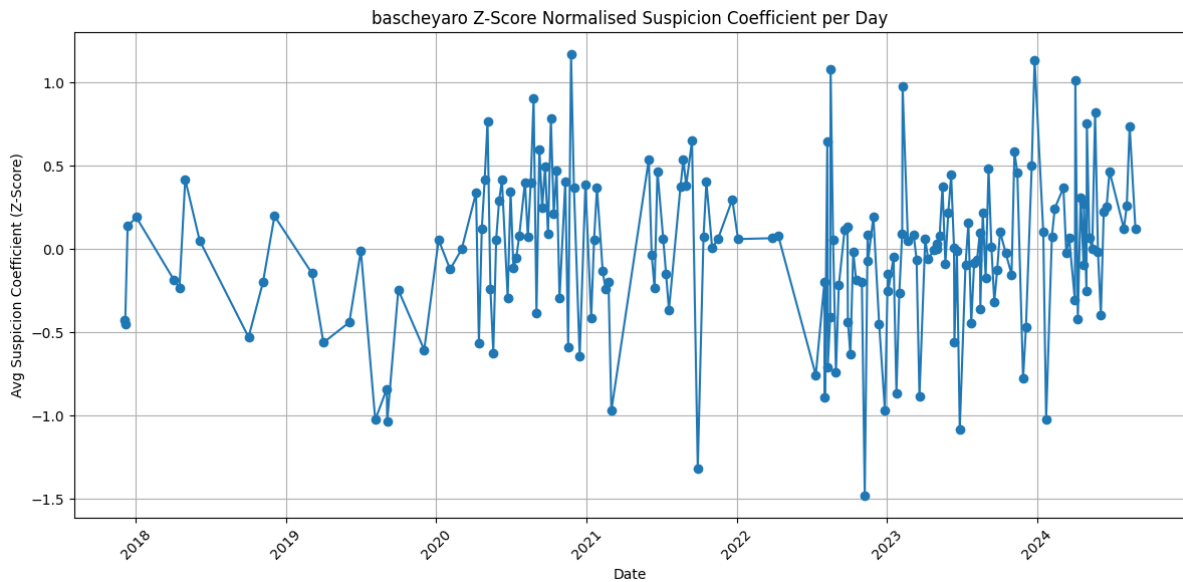


Figure 45 - bascheyaro ANTSC across tournaments

Comparative Analysis

As seen in [figures 36-45](#), the average standard deviation is around 13 seconds, with some players like bascheyaro ([figure 44](#)) edging a little higher at around 14 seconds. Since this is also true for Frogo47, it is a suggestion that their data is not anomalous, and in fact points toward fair play when assessing temporal style. However, there are some interesting results in the other analysed players. For example, Fandorine ([Figure 42](#)) achieved a tournament average standard deviation of under six seconds on two occasions in 2023. This corresponded to ANTSC scores ([figure 43](#)) of over 2, which at a 95% confidence level, is statistically significant because they are both over 1.96. This is far higher than any other player reached (apart from one occasion from eljanov, which will be analysed), and should be explored further.

Fandorine reached ANTSC scores of 2.29 and 2.06 on 30/08/2023 and 22/11/2023, respectively ([Figure 43](#)). The most significant aspect of these dates is that they were Wednesdays, not Tuesdays. Chess.com records game dates in GMT, and Fandorine played one game in each tournament that ran over midnight, and finished on a new day, causing the PGN to record the date played as a Wednesday, not a Tuesday. This also means the parser treats this day as a new tournament, which is a mistake. While this error is comical, both games still could indicate cheating. The more likely explanation, however, is that they were coincidences. This is the reason why averages over entire tournaments are taken for standard deviation and ANTSC, and not just individual games. Because of this debacle, it is not worth analysing these games in detail.

The game mentioned earlier, where [eljanov achieved a ANTSC score of 3.75](#) also turns out to be disappointing. The player only made three moves before resigning, likely because they had to go, or decided they did not want to play chess that day. Because each move was played instantly, the standard deviation was extremely low. This is not evidence of cheating and perfectly explains why their score was so significant.

Value of Visual Anomaly Detection

Despite not finding significant results, these graphs are a powerful tool for narrowing down which tournaments or players merit further investigation. Instead of trawling through thousands of games manually, the visualisation of temporal data highlights crucial statistical anomalies. By flagging tournaments with unusually low standard deviation and high ANTSC scores, researchers can then selectively review individual games in context. By comparing with engine likeness charts, it provides a balanced method for analysing fair play.

Machine Learning Approach

In addition to the previously discussed methods for identifying anomalies, an Isolation Forest (2008) model was trained to identify the most anomalous 1% of games in the combined_stockfish_results dataset.

Generating the Suspicious Dataset

This model, and a new filtered dataset of suspicious games were created. A snippet of the dataset is seen in [Figure 46](#):

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1	Game ID	Date	Round	White Play	White Ratir	Black Play	Black Ratir	Result	Terminatio	ACPL (Whi	Accuracy (ACPL (Blac	Accuracy (Move Cour	White Acci	Black Acci	White ACP	Black ACP	Anomaly Score
2	Early-Titlet 2023.06.21		4	NMChess1	2239	tortorches	1999	1-0	NMChess1	9	97	40	86	26	0.043323	0.043022	0.00402	0.02001	-1
3	Early-Titlet 2023.06.21		8	Wilver92	2153	Better_Che	2010	1-0	Wilver92 w	20	96	44	89	34	0.044589	0.044279	0.009289	0.021891	-1
4	Early-Titlet 2023.06.21		8	KVAIDAN	2350	Zmlyevik22	2502	0-1	Zmlyevik22	68	85	53	90	74	0.03617	0.035971	0.028936	0.021183	-1
5	Early-Titlet 2022.11.11		7	MatthewG	2857	Azatic	2571	1-0	MatthewG	67	91	84	87	53	0.031852	0.033839	0.023451	0.032672	-1
6	Early-Titlet 2022.11.11		7	psychokille	2616	mirceapar	2736	1-0	psychokille	59	90	67	87	78	0.034404	0.031798	0.022554	0.024488	-1
7	Early-Titlet 2022.11.11		10	banane_ve	2376	Oleg_Papa	2484	1-0	banane_ve	50	87	66	86	95	0.036616	0.034622	0.021044	0.02657	-1
8	Early-Titlet 2022.11.11		10	Hikaru	3198	FabianoCa	3077	0-1	FabianoCa	17	90	13	95	90	0.028143	0.030874	0.005316	0.004225	-1
9	Early-Titlet 2022.11.11		10	VerdeNoti	2924	Ginzburg_	2593	1-0	VerdeNoti	44	88	48	86	95	0.030096	0.033166	0.015048	0.018511	-1
10	Late-Titlet 2022.11.11		1	Nick12772	2357	Maitreia	2715	0-1	Maitreia w	53	85	44	88	93	0.036063	0.032413	0.022486	0.016206	-1
11	Late-Titlet 2022.11.11		2	Sam_Ches	2852	Alexei_Gul	2561	1-0	Sam_Ches	72	92	92	87	39	0.032258	0.033971	0.025245	0.035923	-1
12	Late-Titlet 2022.11.11		2	nOcHeaTil	2608	DrVelja	2877	1-0	nOcHeaTil	53	87	62	84	63	0.033359	0.029197	0.020322	0.02155	-1
13	*** SCC G 2020.07.1		7	rugbyclub	2114	gagtemp	1935	1-0	rugbyclub v	9	96	25	88	26	0.045412	0.045478	0.004257	0.01292	-1
14	*** SCC G 2020.07.1		7	Euge_ches	2000	isainz	2146	0-1	isainz won	40	89	39	94	38	0.0445	0.043802	0.02	0.018173	-1
15	*** SCC G 2020.07.1		8	MVM2008	2220	AurumO	1994	1/2-1/2	Game draw	11	95	9	96	25	0.042793	0.048144	0.004955	0.004514	-1
16	*** SCC G 2020.07.1		8	XxPetrofsi	1973	Shushik	2204	1-0	XxPetrofsi	23	92	57	83	27	0.046629	0.037659	0.011657	0.025862	-1
17	*** Titled 1 2020.01.0		8	ShahMatk	2542	VeraNebol	2384	0-1	VeraNebol	90	82	141	92	63	0.032258	0.038591	0.035405	0.059144	-1
18	*** Titled 1 2020.01.0		8	otrov-11	2422	shanemelz	2282	1-0	otrov-11 w	73	89	77	87	99	0.036746	0.038124	0.03014	0.033742	-1
19	Titled-Tues 2021.06.01		8	HackWithH	2292	Sanyura	2510	0-1	Sanyura w	65	84	56	87	93	0.036649	0.034661	0.02836	0.022311	-1
20	Titled-Tues 2021.06.01		9	vikber1961	1950	Zarikovs19	2120	0-1	Zarikovs19	53	83	15	94	26	0.042564	0.04434	0.027179	0.007075	-1
21	Titled-Tues 2021.06.01		10	chessmac	2273	akanga001	1989	1-0	chessmac	23	94	59	82	26	0.041355	0.041227	0.010119	0.029663	-1
22	Titled-Tues 2021.06.01		10	Pantata00	2284	zurzan	1992	1-0	Pantata00	35	94	52	86	57	0.041156	0.043173	0.015324	0.026104	-1
23	Titled-Tues 2021.06.1		1	scarabee4	2621	tteshan	2337	0-1	tteshan wc	67	83	52	90	64	0.031667	0.038511	0.025563	0.022251	-1
24	Titled-Tues 2021.06.1		1	mr65	2179	BigAl	2555	0-1	BigAl won c	5	97	5	98	75	0.044516	0.038356	0.002295	0.001957	-1
25	Late-Titlet 2024.02.1		7	Canal_Sho	2061	SmirnovTir	1964	1-0	Canal_Sho	22	96	48	86	33	0.046579	0.043788	0.010674	0.02444	-1
26	Late-Titlet 2024.02.1		7	ritanovikov	2204	pfalcon194	1953	0-1	pfalcon194	57	84	34	94	33	0.038113	0.048131	0.025862	0.017409	-1
27	Late-Titlet 2024.02.1		8	norwaytoa	2211	boguszp	2024	1-0	norwaytoa	27	94	62	83	27	0.042515	0.041008	0.012212	0.030632	-1
28	Late-Titlet 2024.02.1		9	gxbg2000	2054	Ready4abs	2227	1/2-1/2	Game draw	10	96	9	96	37	0.046738	0.043107	0.004869	0.004041	-1
29	Late-Titlet 2024.02.1		9	tanax17	2119	SmirnovTir	1950	1-0	tanax17 w	25	89	41	86	45	0.043417	0.0439	0.011768	0.009698	1

Figure 46 – Isolation Forest Resulting Suspicious Games Dataset

This is not overly useful in isolation, but it provides a basis for interesting analysis. Using the code found in [Appendix J](#), a table was created that features five columns: player name, number of suspicious games identified, total games in the main dataset, mean Elo across all flagged games, and suspicion rate. These are all self-explanatory, aside from suspicion rate, which is determined by the number of suspicious games divided by the total games in the main dataset for each player with at least one suspicious game. A snippet of this table is seen in [Figure 47](#).

	A	B	C	D	E
1	Player	Suspicious_Games	Mean_Elo_Flagged	Total_Games	Suspicion_Rate
2	Stup_achello123	1	2324	1	1
3	ochukov	1	2068	2	0.5
4	Gougoune25	1	2223	2	0.5
5	WendyTheTing	1	1940	2	0.5
6	P4AD	1	2648	3	0.333333333
7	ChessCoachRachel	1	1956	3	0.333333333
8	gauranga	5	2019.6	17	0.294117647
9	shepi13	2	1993.5	7	0.285714286
10	KikaLaurincova	2	1975.5	7	0.285714286
11	MrHenriksson	5	2017.8	18	0.277777778
12	BlackSwan13	3	2176.666667	11	0.272727273
13	punkmonkZ	2	1952.5	8	0.25
14	monika84	2	2038	8	0.25
15	demondom	1	2075	4	0.25
16	Just-Live-Simply	2	2050	8	0.25
17	LunaTheTuna2013	4	1989.25	16	0.25
18	APotatoInthekitchen	1	2161	4	0.25
19	Miamichess	1	2050	4	0.25
20	WGMCarlaHeredia	2	2057	8	0.25
21	ZaidejasChEgis	1	2207	4	0.25
22	LoggiaFelixdaMaria	1	2092	4	0.25
23	ZalupiNputiN	2	2041	9	0.222222222
24	Suraj240704	1	2371	5	0.2
25	Lich	1	1910	5	0.2
26	sachiniranasinghe	3	1959.333333	15	0.2
27	I_ll_Be_Back_Soon	1	1960	5	0.2
28	joshuacao	1	2012	5	0.2
29	mrsmarple1968	2	1939.5	10	0.2
30	zuzanak	2	1985	10	0.2
31	hsskoo	4	1936.25	22	0.181818182
32	supermaster15	2	2679	11	0.181818182

Figure 47 – Suspicion Table

Suspicion Rate Summary Table

As seen in the table, the first 30 entries are entirely made up of players with a maximum of 18 games in Titled Tuesday. This can be interpreted in a few ways. For one, it could be because of the small sample size for these players. If they had played more games, maybe their suspicion rate would not be as high because there is less chance of a coincidence. It could also be because these players have been banned by Chess.com and cannot play on their website anymore, which is unlikely, but it is nice to dream big! Nevertheless, this table is an extremely useful tool for cheating detection and could be used for further analysis by checking individual games based on a player's suspicion rate and cross-referencing with the other mechanisms in the project.

Comparison to Existing Research

This study builds upon the body of literature on chess cheating detection by introducing a novel methodology that works with engine accuracy, temporal behaviour analysis, and unsupervised anomaly detection using Isolation Forests.

Improvements on Engine-Based Models

Unlike traditional methods such as Regan and Haworth's (2011) probabilistic modelling or Ferreira's (2012) distribution comparison, the Stockfish-based accuracy tool in this project involves position complexity, move volatility, and harmonic mean calculation, which is far better suited for more nuanced environments like chess. This was inspired by Guid & Bratko's (2006) emphasis on position complexity as a key factor in evaluating move quality, and provides a more specific method of evaluating engine-like play.

Specialisation in Temporal Features

Secondly, by integrating temporal behaviour analysis directly, a gap in the research was located and addressed. While Iavich & Kevanishvili (2025) did explore this, they simply used it as a feature in their neural network. Although this was certainly fruitful, it is impossible to know if that was what led them to such significant results. Belhoucine and Feng (2025) delved into behavioural methods also, but did not specialise in time management habits. The method used in this project, however, goes further by applying z-score normalised suspicion coefficients over entire tournaments, hence highlighting the statistically significant behaviour that could suggest engine usage.

Strengths of the Unsupervised Approach

Finally, this project uses unsupervised learning via Isolation Forests to flag games that deviate from Elo-adjusted norms. By normalising with respect to relative Elo, the model filters out a lot of noise, and in consequence selects games that are indicative of suspicious behaviour. The use of unsupervised learning was in response to Patria et al.'s (2021) heavy reliance on unreliable training data through Cheat-Net (Clarkerubber, 2017). The performance of this project proves that outlier detection is possible even without a targeted labelled dataset.

Research Questions

In the Literature Review, three research questions were posed with the intention of being able to answer them upon completion of the project. The questions were:

- 1.) "To what extent can move time be used as an indicator of engine assistance in competitive online chess games?"
- 2.) "How significant is the issue of cheating in Titled Tuesday events?"
- 3.) "Which in-game factors are most predictive of engine usage in online chess?"

RQ1 – Can Move Time Indicate Cheating?

To answer the first question, move times can indeed be used as an indicator of engine assistance. The creation of the average normalised time suspicion coefficient (ANTSC) gives a well-informed statistical basis for how engine-like a player's temporal management patterns are. While this study did not identify players as cheaters directly through this, it worked excellently in acting as an adjudicator for players who had been flagged through the Stockfish accuracy review. It is evident that with further investigation and more time to analyse the entire dataset using this tool, suspicious players could be located and tested using the other two methods developed in this study.

RQ2 – How Prevalent is Cheating in Titled Tuesday?

The second question is more difficult to answer using the resources in this study. Because there were no players who were identified as outright cheaters, lots of players with suspicious repertoires were found. It would be unethical to suggest that the methods in this research are foolproof enough to have sensed all the cheaters in Titled Tuesday, but it is almost certain that there are some that have been flagged correctly. Detecting cheating in online chess is extremely difficult – the inevitability of centaur players, a lack of vision into the physical behaviour of players, and the subtlety of engine-assisted play are some of the key challenges. However, enough players are raising serious suspicion, particularly in the Isolation Forest table, that cheating in Titled Tuesday tournaments is undeniably an issue.

RQ3 – Which Features Are Most Predictive?

As for the third question, the study indicates that engine-assisted play is most accurately detected not by one single factor. For assuredness regarding whether a player is cheating, a combination of high engine accuracy, low centipawn loss, suspicious timing patterns, and performance higher than one's Elo should be present among other factors not analysed in this research. Because of how difficult it is to catch cheaters, as displayed both by this project and existing research, the perfect algorithm is likely unattainable.

Research Limitations

Stockfish Engine Depth

The Stockfish engine used in this study was only run at depth 13. A higher depth would have been more suitable, as the best performing players in Titled Tuesday often possess Elos of over 3000, which is less than the predicted Elo of depth 13. Even while using a high-performance cluster (Edinburgh Napier University, 2023), analysis at a depth of 20 would have taken months to run and was therefore infeasible given the time limitations of the study.

Isolation Forest Calibration

Since the training of the Isolation Forest model was unsupervised, there are no confidence scores, nor are there probability estimates of cheating. In an ideal world, a reliable cheating dataset would be available for use, in which case the player's performances could have been compared against it.

Chess Understanding Requirements

To completely understand everything discussed in this research, a minimum of a basic understanding of chess is helpful. This makes it slightly inaccessible to most people and is a flaw in the research. However, a lot of work went into making it as digestible as possible, with explanations of chess terminology where necessary, and a focus on making the ideas in the study applicable to other subjects.

Conclusion

Ultimately, this study provides a significant step in the right direction in the detection of suspicious behaviour in online chess. The most novel idea in the research was creating a dataset of every game ever played in the most ubiquitous online chess tournament in the world: Titled Tuesday. The research was all-encompassing, adapting methods from existing studies to create three different methods of detecting cheating, and fundamentally, it was a prodigious success. From the Stockfish-based accuracy tool incorporating position complexity and harmonic mean scoring, to the temporal analysis of move times, and the Isolation Forest model detecting Elo-based outliers, the scripts developed in this project show strong potential for uncovering suspicious behaviour. Admittedly, no players were found to be likely cheaters, but this study should not be considered a prosecuting tool, where identified players are accused of fair play misuse. Instead, it should be deemed a tool for lowering the workload of organisations tasked with banning these players. It took a vast dataset and filtered it to one-hundredth of the size, with players demonstrating at least one of the three barometers for cheating in all cases.

From the Stockfish-based accuracy tool incorporating position complexity and harmonic mean scoring, to the temporal analysis of move times, and the Isolation Forest model detecting Elo-based outliers, the scripts developed in this project show enormous potential for uncovering dubious behaviour. By narrowing down hundreds of thousands of games to a carefully selected shortlist, this project exhibits how data science can complement judgment in the fight against engine-assisted cheating. While the findings are not definitive, they pave the way for higher-budget and professional analysis tools in the future. With further refinement and wider scope, these techniques could one day shape the environment of anti-cheating detection in online chess.

Appendix 1 – Self Review

At the beginning of this project, my aim was to explore the extent of how cheating in online chess can be detected. Specifically, I wanted to use statistical analysis, behavioural analysis, and unsupervised machine learning techniques to identify potential engine assistance in online chess tournaments. My objective was to build a system that could combine Stockfish-based evaluations, timing patterns, and an Isolation Forest model to flag suspicious behaviour in Titled Tuesday games. Reflecting on the project as a whole, I believe the project fulfilled these aims to a highly respectable degree, though it was not without its challenges.

One of the most valuable aspects of this dissertation was the technical growth it demanded. I had to significantly deepen my knowledge of Python to fully understand and adapt the code that the Stockfish script was inspired by, and to write the script to initialise the dataset, my grasp of data manipulation techniques was improved. I also became much more confident working with large datasets due to the nearly 900,000 chess games that encompassed it. By reading documentation on parsing text and PGN files, I learned how to process and extract meaningful insights from them. Running intensive code on the Edinburgh Napier computing cluster was another skill I had never used before this project, and it was an essential part of the project. I have little experience in Linux, so working with the Linux-based cluster was a big obstacle and took a lot of research to process my code in batches.

On the soft skills side, this project tested my self-discipline and problem-solving ability more than anything I've accomplished in my degree. I had to work independently over a long period, maintain clear documentation of my code, and develop a substantial plan for the direction of the research, all while running into various hurdles that changed my plan along the way. For example, although I initially planned to run Stockfish at depth 17 or 20 for precision, I learned to make pragmatic choices and settled on depth 13 after breaking down time constraints. This was difficult to do, and I was fearful that it would invalidate my findings. But after researching, I understood that this depth was still good enough for master-level play.

I also encountered difficulties in filtering and preparing the data. For example, clock annotations were not always consistently formatted. Combining them across nearly a decade of tournaments introduced a lot of unreliability. Overcoming this required unexpected research on the issues that were arising which was time spent where I planned to be doing other things. Likewise, designing an appropriate time-oriented suspicion coefficient took several iterations to get right. I struggled initially with how to create an intuitive metric where peaks referred to suspicion and troughs referred to human play. Eventually, I solved this problem by normalising and then inverting the standard deviation chart.

The unsupervised nature of the project was another challenge. Initially, it felt like it was going to be impossible to be able to even identify players who looked like they were cheating without a labelled dataset. I knew that organisations who have tasked themselves with detecting cheating in chess (like Chess.com) have clear labelled data to use for supervised models. Because of this, it felt like I was stabbing in the dark. To confirm which games were

genuinely cheated in, it was difficult to evaluate the effectiveness of my model, as there was nothing to refer to. In response to this, I focused on statistical outliers and interpreted them with accuracy scores, move quality, timing behaviour, and tournament data. While this did not provide hard proof, it did offer a strong framework for further investigation, and a basis for cross-referencing with the other evaluation metrics in this project.

Overall, this dissertation has strengthened my technical skills and improved my ability to work independently massively. It has shown me how complex cheating detection is, and the importance of balancing data science with knowledge. The next time I am tasked with solving an open-ended problem, I will not be doubtful that I can produce impactful mechanisms and results. If given more time, I would spend it on performing temporal analysis on the entire dataset, instead of just using it as a confirmation tool. I would also explore supervised methods using publicly known cheater datasets, even if they are not inherently reliably. Nonetheless, I am immensely proud of what I have built and am genuinely thrilled that I have planned and executed a meaningful and novel contribution to the growing body of research in the field of detecting cheating in chess.

Bibliography

- ACCA. (2010). 2010 Fourth Annual ACCA World Computer Chess Championships -- Results. *ACCA*.
<https://aigames.net/ACCA/ACCAWCRCC/2010ACCAWCRCC/WCRCCResults.html>
- Asavis. (2024). *Chess_Accuracy*. https://github.com/asavis/chess_accuracy
- Banks, G., Banks, R., Bird, S., Kryukov, K., & Smith, C. (2010).
computerchess.org.uk_ccrl_. <https://www.computerchess.org.uk/ccrl/>
- Belhoucine, A., & Feng, Z. (2025). Detecting Chess Cheating Thanks To ML & ANN Technology. *International Journal of Computational Engineering Research (IJCER)*, 15(2), 28–37. https://www.ijceronline.com/papers/Vol15_issue2/15022837.pdf
- CCRL. (2025). *CCRL 40/15 Downloads and Statistics*.
<https://computerchess.org.uk/ccrl/4040/>
- ChessBase. (1997). chessbase1997fritz5. *ChessBase GmbH*.
<https://books.google.co.uk/books?id=9UXeygAACAAJ>
- ChessBase. (2012). *FIDE confirms sanctions in French cheating case _ ChessBase*.
- ChessBase GmbH. (2025). *ChessBase Database*. <https://en.chessbase.com/>
- Chess.com. (2022). *Articles Articles Chess.com Fair Play And Cheat-Detection*.
<https://www.chess.com/article/view/chess-com-fair-play-and-cheat-detection>
- Chess.com. (2023). *The Turk - Chess Terms - Chess.com*. <https://www.chess.com/terms/turk-chess-automaton>
- Chess.com. (2025a). *Articles Articles Titled Tuesday: All The Information*.
<https://www.chess.com/article/view/titled-tuesday>
- Chess.com. (2025b). *Live Chess Tournaments Live Chess Tournaments Arena Swiss Titled Tuesday*. <https://www.chess.com/tournament/live/titled-tuesdays>
- Clarkerubber. (2017). cheat-net. *Github*. <https://github.com/clarkerubber/Cheat-Net>
- Desmos. (2025). *Custom Graph on Winning Chances from Centipawn Loss*.
<https://www.desmos.com/calculator/ql81rwrk2>
- Edinburgh Napier University. (2023). *New High Performance Computing capacity to enhance research possibilities*. <https://www.napier.ac.uk/about-us/news/enu-compute-cluster-high-performance-computing>
- Edinburgh Napier University. (2025). *SoC _SEBE Research Integrity*.
<https://livenapierac.sharepoint.com/sites/schools/SOC/research/ri/Ethics/SitePages/School-of-Computing-Research-Integrity.aspx?CT=1713174535369&OR=OWA-NT-Mail&CID=16a5d00c-90c6-b793-7a37-d4f45271b1f2>

- Ferreira, D. R. (2012). *Determining the Strength of Chess Players Based on Actual Play 3*. *DETERMINING THE STRENGTH OF CHESS PLAYERS BASED ON ACTUAL PLAY*. <http://ssdf.bosjo.net/>
- Ferreira, D. R. (2013). The Impact of the Search Depth on Chess Playing Strength. *ICGA Journal*, 36(2), 67–80. <https://doi.org/10.3233/ICG-2013-36202>
- FIDE. (2014). *Anti-Cheating Guidelines*. <https://www.fide.com/FIDE/handbook/Anti%20Cheating%20Guidelines.pdf>
- FIDE. (2025, March 22). *FIDE Ratings - Titled Players*. https://ratings.fide.com/advanced_search.phtml
- Free Internet Chess Server. (2025). *FICS Games Database and API*. <https://www.ficsgames.org/>
- Google Scholar. (2025). *Intrinsic Chess Ratings - Google Scholar*. https://scholar.google.co.uk/scholar?hl=en&as_sdt=0%2C5&q=intrinsic+chess+ratings&btnG=&oq=intrinsic+chess+
- Greengard, M. (2005). *U2000 Intrigue at HB*. www.chessninja.com/dailydirt/2005/07/u2000-intrigue-at-hb.htm
- Guid, M., & Bratko, I. (2006). Computer Analysis of World Chess Champions. *ICGA Journal*, 29, 65–73. https://www.researchgate.net/publication/220174548_Computer_Analysis_of_World_Chess_Champions
- Hilbert, M., & López, P. (2011). *The World's Technological Capacity to Store, Communicate, and Compute Information*. [https://doi.org/https://doi.org/10.1126/science.1200970](https://doi.org/10.1126/science.1200970)
- Iavich, M., & Kevanishvili, Z. (2025). A Neural Network Approach to Chess Cheat Detection. In D. and B. R. and Č. J. Lopata Audrius and Gudonienė (Ed.), *Information and Software Technologies* (pp. 131–145). Springer Nature Switzerland. https://link.springer.com/chapter/10.1007/978-3-031-84263-4_12
- International Energy Agency. (2024). *Global EV Outlook 2024 Moving towards increased affordability*. www.iea.org
- Lichess. (2025). Weekly Blitz rating distribution • lichess.org. *Lichess*. <https://lichess.org/stat/rating/distribution/blitz>
- Lichess contributors. (2025). *Lichess Game Analysis*. <https://lichess.org/analysis>
- Lichess.org. (2024). *Lichess API*. Lichess. <https://lichess.org/api>
- Lichess.org. (2025). lichess.org open database. *Lichess*. <https://database.lichess.org/>
- Liu, F. T., Ting, K. M., & Zhou, Z.-H. (2008). Isolation Forest. *2008 Eighth IEEE International Conference on Data Mining*, 413–422. <https://doi.org/10.1109/ICDM.2008.17>

- Lutz, T. (2023). Carlsen and Niemann settle dispute over cheating claims that rocked chess _ Chess _ The Guardian. *Guardian*.
<https://www.theguardian.com/sport/2023/aug/28/magnus-carlsen-hans-niemann-chess-cheating-allegations-settlement>
- McIlroy-Young, R., Sen, S., Kleinberg, J., & Anderson, A. (2020). Aligning Superhuman AI with Human Behavior: Chess as a Model System. *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD '20)*, 1677–1687. <https://doi.org/10.1145/3394486.3403219>
- Pascutto, G.-C., & Linscott, G. (2019). *Leela Chess Zero*. <http://lczero.org/>
- Patria, R., Favian, S., Caturdewa, A., & Suhartono, D. (2021a). Cheat Detection on Online Chess Games using Convolutional and Dense Neural Network. *2021 4th International Seminar on Research of Information Technology and Intelligent Systems, ISRITI 2021*, 389–395. <https://doi.org/10.1109/ISRITI54043.2021.9702792>
- Patria, R., Favian, S., Caturdewa, A., & Suhartono, D. (2021b). Cheat Detection on Online Chess Games using Convolutional and Dense Neural Network. *2021 4th International Seminar on Research of Information Technology and Intelligent Systems, ISRITI 2021*, 389–395. <https://doi.org/10.1109/ISRITI54043.2021.9702792>
- Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, É. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12(85), 2825–2830.
<http://jmlr.org/papers/v12/pedregosa11a.html>
- Regan, K. W., & Haworth, G. M. (2011). *Intrinsic Chess Ratings*.
<https://doi.org/https://doi.org/10.1609/aaai.v25i1.7951>
- Reibman, A. I., & Bailard, B. W. (1983). *NON-MINTMAX SEARCH SI'RAMGIES FOR USE AGAINST FALLIBIX OPPOmS*. www.aaai.org
- Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., & Hassabis, D. (2017). *Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm*. <https://arxiv.org/abs/1712.01815>
- Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., & Hassabis, D. (2018). A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362(6419), 1140–1144. <https://doi.org/10.1126/science.aar6404>
- Stockfish. (2024). *Stockfish Chess Engine*. Github. <https://github.com/official-stockfish/Stockfish>

- Taylor, P. (2024). *Volume of data/information created*.
<https://www.statista.com/statistics/871513/worldwide-data-created/>
<https://www.statista.com/statistics/871513/worldwide-data-created/>
- TCEC. (2025). *TCEC Wiki*. <http://chessopeningsforengines.wikidot.com/opening-lines-in-tcec>
- van Harreveld, F., Wagenmakers, E.-J., & van der Maas, H. L. J. (2007). The effects of time pressure on chess skill: an investigation into fast and slow processes underlying expert performance. *Psychological Research*, 71(5), 591–597. <https://doi.org/10.1007/s00426-006-0076-0>
- Vasik, R., & Kaufmann, L. (2007). *Rybka - for the serious chess player*.
<http://www.rybkachess.com/>
- Walczak, S. (1996). Improving opening book performance through modeling of chess opponents. *Proceedings of the 1996 ACM 24th Annual Conference on Computer Science*, 53–57. <https://dl.acm.org/doi/pdf/10.1145/228329.228334>

Appendix A

```
1 import pandas as pd
2
3 df = pd.read_csv("combined_stockfish_results.csv")
4
5 # Convert accuracy
6 df["Accuracy (White)"] = pd.to_numeric(df["Accuracy (White)"], errors='coerce')
7 df["Accuracy (Black)"] = pd.to_numeric(df["Accuracy (Black)"], errors='coerce')
8
9 # max accuracy column
10 df["Max Accuracy"] = df[["Accuracy (White)", "Accuracy (Black)"]].max(axis=1)
11
12 # Drop bad rows
13 df = df.dropna(subset=["Max Accuracy"])
14
15 # Sort by Max Accuracy and select 100
16 top_100_games = df.sort_values(by="Max Accuracy", ascending=False).head(100)
17
18
19 print(top_100_games[["White Player", "Black Player", "Accuracy (White)", "Accuracy (Black)", "Max Accuracy"]])
20
21 # Save to CSV
22 top_100_games.to_csv("top_100_high_accuracy_games.csv", index=False)
23
```

Appendix B

```
1 import pandas as pd
2
3 df = pd.read_csv("combined_stockfish_results.csv")
4
5 # Convert accuracy
6 df["Accuracy (White)"] = pd.to_numeric(df["Accuracy (White)"], errors="coerce")
7 df["Accuracy (Black)"] = pd.to_numeric(df["Accuracy (Black)"], errors="coerce")
8 df["Move Count"] = pd.to_numeric(df["Move Count"], errors="coerce")
9
10 # Filter out games with fewer than 21 moves
11 df = df[df["Move Count"] > 20]
12
13 # Max accuracy column
14 df["Max Accuracy"] = df[["Accuracy (White)", "Accuracy (Black)"]].max(axis=1)
15
16 # Sort by max accuracy and then by move count
17 df_sorted = df.sort_values(by=["Max Accuracy", "Move Count"], ascending=[False, False])
18
19 # Select the top 100
20 top_100 = df_sorted.head(100)
21
22 # Save to CSV
23 top_100.to_csv("top_100_high_accuracy_games.csv", index=False)
24
25
```

Appendix C

[Event "Late-Titled-Tuesday-Blitz-June-20-2023"]

[Site "Chess.com"]

[Date "2023.06.20"]

[Round "4"]

[White "santosAlberto1987"]

[Black "IngOscarArdila"]

[Result "1-0"]

[WhiteElo "2490"]

[BlackElo "2272"]

[TimeControl "180+1"]

[EndTime "21:36:17 GMT+0000"]

[Termination "santosAlberto1987 won by checkmate"]

1. e4 {[%clk 0:03:00.9]} 1... e5 {[%clk 0:02:59.5]} 2. Nf3 {[%clk 0:03:00.9]}
2... Nf6 {[%clk 0:02:59.9]} 3. Nxe5 {[%clk 0:02:58.3]} 3... d6 {[%clk
0:02:59.7]} 4. Nf3 {[%clk 0:02:58.5]} 4... Nxe4 {[%clk 0:02:59.9]} 5. d4 {[%clk
0:02:59]} 5... Be7 {[%clk 0:02:59.6]} 6. Bd3 {[%clk 0:02:59.5]} 6... d5 {[%clk
0:03:00.1]} 7. O-O {[%clk 0:02:58.5]} 7... Nc6 {[%clk 0:03:00.1]} 8. Re1 {[%clk
0:02:58.4]} 8... Nf6 {[%clk 0:02:59.8]} 9. h3 {[%clk 0:02:52.9]} 9... O-O {[%clk
0:02:59.5]} 10. c3 {[%clk 0:02:51.4]} 10... h6 {[%clk 0:02:58.3]} 11. Bf4 {[%clk
0:02:51]} 11... Be6 {[%clk 0:02:52]} 12. Nbd2 {[%clk 0:02:50.4]} 12... a6 {[%clk
0:02:47.6]} 13. Nf1 {[%clk 0:02:50.1]} 13... Qd7 {[%clk 0:02:45.8]} 14. Ng3
{[%clk 0:02:50.2]} 14... g5 {[%clk 0:02:41.4]} 15. Ne5 {[%clk 0:02:43.1]} 15...
Qd8 {[%clk 0:02:27.1]} 16. Nxc6 {[%clk 0:02:42.7]} 16... bxc6 {[%clk 0:02:26.7]}
17. Be5 {[%clk 0:02:42.8]} 17... Nd7 {[%clk 0:02:23.4]} 18. Qh5 {[%clk
0:02:42.6]} 18... Nxe5 {[%clk 0:02:14.7]} 19. Rxe5 {[%clk 0:02:39.6]} 19... Kg7
{[%clk 0:02:14.3]} 20. Rxe6 {[%clk 0:02:24.4]} 20... Bf6 {[%clk 0:02:13]} 21.
Nf5+ {[%clk 0:02:20.8]} 21... Kg8 {[%clk 0:02:10.2]} 22. Nxb6+ {[%clk

0:02:18.7]} 22... Kg7 {[%clk 0:02:10.7]} 23. Nf5+ {[%clk 0:02:18.6]} 23... Kg8
 {[%clk 0:02:11.1]} 24. Ne7+ {[%clk 0:02:15.7]} 24... Kg7 {[%clk 0:02:09.1]} 25.
 Qh7# {[%clk 0:02:15.3]} 1-0

Appendix D

1. e4 {[%clk 0:03:00.9]} 1... c5 {[%clk 0:02:56.9]} 2. Nf3 {[%clk 0:02:59.9]} 2... Nc6
 {[%clk 0:02:55.7]} 3. d4 {[%clk 0:03:00.8]} 3... cxd4 {[%clk 0:02:55.8]} 4. Nxd4 {[%clk
 0:03:01]} 4... e5 {[%clk 0:02:55.8]} 5. Nb5 {[%clk 0:03:00.7]} 5... a6 {[%clk 0:02:54.6]} 6.
 Nd6+ {[%clk 0:02:59.4]} 6... Bxd6 {[%clk 0:02:54.5]} 7. Qxd6 {[%clk 0:03:00]} 7... Nf6
 {[%clk 0:02:52.5]} 8. Nc3 {[%clk 0:02:57.5]} 8... Qe7 {[%clk 0:02:51.9]} 9. Qxe7+ {[%clk
 0:02:56.1]} 9... Nxe7 {[%clk 0:02:52.8]} 10. Bg5 {[%clk 0:02:52.7]} 10... b5 {[%clk
 0:02:37.8]} 11. O-O-O {[%clk 0:02:43]} 11... Bb7 {[%clk 0:02:35.3]} 12. f3 {[%clk
 0:02:41.6]} 12... O-O {[%clk 0:02:25.6]} 13. Bxf6 {[%clk 0:02:39.5]} 13... gxf6 {[%clk
 0:02:26.1]} 14. Rxd7 {[%clk 0:02:39.4]} 14... Nc6 {[%clk 0:02:22.4]} 15. Rxb7 {[%clk
 0:02:35.5]} 15... Na5 {[%clk 0:02:20.4]} 16. Rd7 {[%clk 0:02:34.6]} 16... Rfc8 {[%clk
 0:02:18.9]} 17. Bd3 {[%clk 0:02:32.8]} 17... Kg7 {[%clk 0:02:18.8]} 18. Rd1 {[%clk
 0:02:30.9]} 18... Nc6 {[%clk 0:02:16.4]} 19. Nd5 {[%clk 0:02:26.2]} 19... Nd4 {[%clk
 0:02:15.6]} 20. Nb6 {[%clk 0:02:21.7]} 20... Rd8 {[%clk 0:02:15.8]} 21. Rxd4 {[%clk
 0:02:14.4]} 21... exd4 {[%clk 0:02:12.8]} 22. Nxa8 {[%clk 0:02:14.8]} 22... Rxa8 {[%clk
 0:02:12.2]} 23. Be2 {[%clk 0:02:13.4]} 23... Rd8 {[%clk 0:02:08.2]} 24. c3 {[%clk
 0:02:13.4]} 24... Rc8 {[%clk 0:02:07.8]} 25. Rxd4 {[%clk 0:02:13.2]} 25... Rc6 {[%clk
 0:02:06.3]} 26. Kd2 {[%clk 0:02:13.5]} 26... Kg6 {[%clk 0:02:06.1]} 27. f4 {[%clk
 0:02:11]} 27... f5 {[%clk 0:02:04.8]} 28. Bd3 {[%clk 0:02:06.3]} 28... Kh5 {[%clk 0:02:03]}
 29. exf5 {[%clk 0:01:59.2]} 29... Kg4 {[%clk 0:02:03.2]} 30. Ke3 {[%clk 0:01:59]} 30...
 Rh6 {[%clk 0:02:01.2]} 31. h3+ {[%clk 0:01:43]} 31... Kg3 {[%clk 0:02:01.5]} 32. Bf1
 {[%clk 0:01:42.3]} 32... Rc6 {[%clk 0:01:57.3]} 33. Rd8 {[%clk 0:01:38.9]} 33... Rc7
 {[%clk 0:01:54.3]} 34. Rg8+ {[%clk 0:01:38.9]} 34... Kh2 {[%clk 0:01:54.8]} 35. g4 {[%clk
 0:01:33.9]} 35... Kg3 {[%clk 0:01:54]} 36. g5 {[%clk 0:01:32.5]} 36... Re7+ {[%clk
 0:01:53.4]} 37. Kd4 {[%clk 0:01:29.8]} 37... Kxf4 {[%clk 0:01:53.7]} 38. g6 {[%clk
 0:01:24.6]} 38... hxg6 {[%clk 0:01:47.4]} 39. fxg6 {[%clk 0:01:25.5]} 39... fxg6 {[%clk
 0:01:47.3]} 40. Rxxg6 {[%clk 0:01:25.2]} 40... Re1 {[%clk 0:01:41.1]} 41. Bd3 {[%clk
 0:01:21.5]} 41... Re7 {[%clk 0:01:37.7]} 42. Rg4+ {[%clk 0:01:20.7]} 42... Kf3 {[%clk
 0:01:37.8]} 43. Be4+ {[%clk 0:01:18.3]} 43... Kf2 {[%clk 0:01:37.3]} 44. Rg2+ {[%clk
 0:01:17.6]} 44... Kf1 {[%clk 0:01:37.7]} 45. Rd2 {[%clk 0:01:14.2]} 45... Rd7+ {[%clk
 0:01:37.5]} 46. Ke3 {[%clk 0:01:13.7]} 46... Rxd2 {[%clk 0:01:31.5]} 47. Kxd2 {[%clk
 0:01:13]} 47... Kf2 {[%clk 0:01:32.3]} 48. h4 {[%clk 0:01:11.9]} 48... Kg3 {[%clk
 0:01:32.6]} 49. h5 {[%clk 0:01:12.8]} 49... Kf4 {[%clk 0:01:33.5]} 50. h6 {[%clk
 0:01:11.7]} 50... Kxe4 {[%clk 0:01:34.1]} 51. h7 {[%clk 0:01:12.6]} 51... Kd5 {[%clk
 0:01:34.6]} 52. h8=Q {[%clk 0:01:12.7]} 52... Kc5 {[%clk 0:01:35.5]} 53. Qd4+ {[%clk
 0:01:12.6]} 53... Kc6 {[%clk 0:01:35.9]} 54. b4 {[%clk 0:01:12.8]} 54... Kc7 {[%clk
 0:01:34.9]} 55. Qc5+ {[%clk 0:01:13.7]} 55... Kb7 {[%clk 0:01:35.7]} 56. Qd6 {[%clk

0:01:13.6}} 56... a5 {[%clk 0:01:34.8]} 57. bxa5 {[%clk 0:01:13.6]} 57... b4 {[%clk
 0:01:35]} 58. cxb4 {[%clk 0:01:13.4]} 58... Kc8 {[%clk 0:01:33.4]} 59. b5 {[%clk
 0:01:14.3]} 59... Kb7 {[%clk 0:01:33.1]} 60. Kc3 {[%clk 0:01:15.2]} 60... Kc8 {[%clk
 0:01:33]} 61. Kb4 {[%clk 0:01:16.1]} 61... Kb7 {[%clk 0:01:33.5]} 62. Kc5 {[%clk
 0:01:16.8]} 62... Kc8 {[%clk 0:01:33.8]} 63. Qe7 {[%clk 0:01:16.9]} 63... Kb8 {[%clk
 0:01:33.4]} 64. b6 {[%clk 0:01:17.1]} 64... Kc8 {[%clk 0:01:34.3]} 65. Kb5 {[%clk
 0:01:16.8]} 65... Kb8 {[%clk 0:01:34.4]} 66. Kc4 {[%clk 0:01:17.1]} 66... Kc8 {[%clk
 0:01:34.8]} 67. Kd3 {[%clk 0:01:17.8]} 67... Kb8 {[%clk 0:01:34.9]} 68. Ke2 {[%clk
 0:01:18.4]} 68... Ka8 {[%clk 0:01:33.9]} 69. Kf1 {[%clk 0:01:19]} 69... Kb8 {[%clk
 0:01:34.8]} 70. Kg1 {[%clk 0:01:19.2]} 70... Kc8 {[%clk 0:01:34.9]} 71. Kh2 {[%clk
 0:01:19.8]} 71... Kb8 {[%clk 0:01:35.4]} 72. Kg3 {[%clk 0:01:18.9]} 72... Kc8 {[%clk
 0:01:36.2]} 73. Kh4 {[%clk 0:01:19.4]} 73... Kb8 {[%clk 0:01:37.1]} 74. Kg5 {[%clk
 0:01:20]} 74... Kc8 {[%clk 0:01:38]} 75. Kf6 {[%clk 0:01:20.5]} 75... Kb8 {[%clk
 0:01:38.9]} 76. Ke5 {[%clk 0:01:20.8]} 76... Kc8 {[%clk 0:01:39.8]} 77. Kd6 {[%clk
 0:01:20.6]} 77... Kb8 {[%clk 0:01:40.6]} 78. Qh7 {[%clk 0:01:20.7]} 78... Kc8 {[%clk
 0:01:41.4]} 79. a3 {[%clk 0:01:18.6]} 79... Kb8 {[%clk 0:01:42]} 80. Ke5 {[%clk
 0:01:18.7]} 80... Kc8 {[%clk 0:01:42.8]} 81. Kf4 {[%clk 0:01:19.4]} 81... Kb8 {[%clk
 0:01:43.5]} 82. Kg3 {[%clk 0:01:20.2]} 82... Kc8 {[%clk 0:01:44.3]} 83. Kh2 {[%clk
 0:01:21]} 83... Kb8 {[%clk 0:01:45]} 84. Kg1 {[%clk 0:01:21.8]} 84... Kc8 {[%clk
 0:01:45.9]} 85. Kf2 {[%clk 0:01:22.5]} 85... Kb8 {[%clk 0:01:46.8]} 86. Ke1 {[%clk
 0:01:23.2]} 86... Kc8 {[%clk 0:01:47.6]} 87. Kd2 {[%clk 0:01:24.1]} 87... Kb8 {[%clk
 0:01:48.4]} 88. Kc1 {[%clk 0:01:24.9]} 88... Kc8 {[%clk 0:01:49.3]} 89. Kb2 {[%clk
 0:01:25.2]} 89... Kb8 {[%clk 0:01:49.2]} 90. Kc3 {[%clk 0:01:25.8]} 90... Kc8 {[%clk
 0:01:49.9]} 91. Kb4 {[%clk 0:01:26.4]} 91... Kb8 {[%clk 0:01:50.6]} 92. Kc5 {[%clk
 0:01:26.9]} 92... Kc8 {[%clk 0:01:51.3]} 93. Kd5 {[%clk 0:01:26.8]} 93... Kb8 {[%clk
 0:01:52.1]} 94. Ke6 {[%clk 0:01:26.8]} 94... Kc8 {[%clk 0:01:52.9]} 95. Kf5 {[%clk
 0:01:27.4]} 95... Kb8 {[%clk 0:01:53.6]} 96. Ke4 {[%clk 0:01:28.2]} 96... Kc8 {[%clk
 0:01:54.3]} 97. Kd3 {[%clk 0:01:29.1]} 97... Kb8 {[%clk 0:01:55.1]} 98. Kc2 {[%clk
 0:01:29.9]} 98... Kc8 {[%clk 0:01:55.9]} 99. Kb3 {[%clk 0:01:30.5]} 99... Kb8 {[%clk
 0:01:56.7]} 100. Ka2 {[%clk 0:01:31.1]} 100... Kc8 {[%clk 0:01:57.4]} 101. a4 {[%clk
 0:01:31.2]} 101... Kb8 {[%clk 0:01:58.1]} 102. Ka1 {[%clk 0:01:31.5]} 102... Kc8 {[%clk
 0:01:58.9]} 103. Kb2 {[%clk 0:01:32.1]} 103... Kb8 {[%clk 0:01:59.7]} 104. Kc1 {[%clk
 0:01:32.7]} 104... Kc8 {[%clk 0:02:00.5]} 105. Kd2 {[%clk 0:01:33.4]} 105... Kb8 {[%clk
 0:02:01.3]} 106. Ke1 {[%clk 0:01:34.2]} 106... Kc8 {[%clk 0:02:02.1]} 107. Kf2 {[%clk
 0:01:34.8]} 107... Kb8 {[%clk 0:02:02.9]} 108. Kg1 {[%clk 0:01:35.4]} 108... Kc8 {[%clk
 0:02:03.7]} 109. Kh2 {[%clk 0:01:35.8]} 109... Kb8 {[%clk 0:02:04.6]} 110. Kg3 {[%clk
 0:01:36]} 110... Kc8 {[%clk 0:02:05.5]} 111. Kh4 {[%clk 0:01:36.5]} 111... Kb8 {[%clk
 0:02:06.1]} 112. Kg5 {[%clk 0:01:37.1]} 112... Kc8 {[%clk 0:02:06.6]} 113. Kf4 {[%clk
 0:01:37.7]} 113... Kb8 {[%clk 0:02:07]} 114. Ke5 {[%clk 0:01:38.4]} 114... Kc8 {[%clk
 0:02:07.6]} 115. Kd4 {[%clk 0:01:39.1]} 115... Kb8 {[%clk 0:02:08.3]} 116. Kc5 {[%clk
 0:01:39.8]} 116... Kc8 {[%clk 0:02:09]} 117. Kb4 {[%clk 0:01:40.6]} 117... Kb8 {[%clk
 0:02:09.7]} 118. Kb5 {[%clk 0:01:40.5]} 118... Kc8 {[%clk 0:02:10.4]} 119. a6 {[%clk
 0:01:40.6]} 119... Kb8 {[%clk 0:02:11.2]} 120. Kc4 {[%clk 0:01:40.1]} 120... Kc8 {[%clk

0:02:12}} 121. Kd3 {[%clk 0:01:40.7]} 121... Kb8 {[%clk 0:02:12.6]} 122. Ke2 {[%clk
 0:01:41.3]} 122... Kc8 {[%clk 0:02:13.3]} 123. Kf1 {[%clk 0:01:42]} 123... Kb8 {[%clk
 0:02:14]} 124. Kg2 {[%clk 0:01:42.7]} 124... Kc8 {[%clk 0:02:14.7]} 125. Kh3 {[%clk
 0:01:43.3]} 125... Kb8 {[%clk 0:02:15.5]} 126. Kg4 {[%clk 0:01:43.8]} 126... Kc8 {[%clk
 0:02:16.2]} 127. Kf5 {[%clk 0:01:44.3]} 127... Kb8 {[%clk 0:02:17]} 128. Ke4 {[%clk
 0:01:44.8]} 128... Kc8 {[%clk 0:02:17.8]} 129. Kd5 {[%clk 0:01:45.2]} 129... Kb8 {[%clk
 0:02:18.6]} 130. Kc4 {[%clk 0:01:45.5]} 130... Kc8 {[%clk 0:02:19.4]} 131. Kb4 {[%clk
 0:01:45.6]} 131... Kb8 {[%clk 0:02:20.3]} 132. Kc5 {[%clk 0:01:45.7]} 132... Kc8 {[%clk
 0:02:21.2]} 133. a5 {[%clk 0:01:45.8]} 133... Kb8 {[%clk 0:02:22]} 134. Kd5 {[%clk
 0:01:45.5]} 134... Kc8 {[%clk 0:02:22.7]} 135. Ke4 {[%clk 0:01:46.2]} 135... Kb8 {[%clk
 0:02:23.4]} 136. Kf3 {[%clk 0:01:47.1]} 136... Kc8 {[%clk 0:02:24.1]} 137. Kg2 {[%clk
 0:01:47.7]} 137... Kb8 {[%clk 0:02:24.8]} 138. Kf1 {[%clk 0:01:48.3]} 138... Kc8 {[%clk
 0:02:25.6]} 139. Ke2 {[%clk 0:01:48.9]} 139... Kb8 {[%clk 0:02:26.4]} 140. Kd1 {[%clk
 0:01:49.3]} 140... Kc8 {[%clk 0:02:27.2]} 141. Kc2 {[%clk 0:01:49.8]} 141... Kb8 {[%clk
 0:02:28]} 142. Kb1 {[%clk 0:01:50.4]} 142... Kc8 {[%clk 0:02:28.7]} 143. Ka2 {[%clk
 0:01:51]} 143... Kb8 {[%clk 0:02:28.1]} 144. Kb3 {[%clk 0:01:51.9]} 144... Kc8 {[%clk
 0:02:28.9]} 145. Ka4 {[%clk 0:01:52.3]} 145... Kb8 {[%clk 0:02:29.7]} 146. Kb5 {[%clk
 0:01:52.3]} 146... Kc8 {[%clk 0:02:30.6]} 147. Qh8+ {[%clk 0:01:50.8]} 147... Kd7 {[%clk
 0:02:30.6]} 148. Qg7+ {[%clk 0:01:50.3]} 148... Kc8 {[%clk 0:02:31]} 149. Qh7 {[%clk
 0:01:48.8]} 149... Kd8 {[%clk 0:02:30.7]} 150. Kc4 {[%clk 0:01:48.2]} 150... Kc8 {[%clk
 0:02:31.2]} 151. Kd3 {[%clk 0:01:48.8]} 151... Kb8 {[%clk 0:02:31.7]} 152. Ke2 {[%clk
 0:01:49.3]} 152... Kc8 {[%clk 0:02:32.2]} 153. Kd1 {[%clk 0:01:49.9]} 153... Kb8 {[%clk
 0:02:32.9]} 154. Kc2 {[%clk 0:01:50.5]} 154... Kc8 {[%clk 0:02:33.6]} 155. Kb1 {[%clk
 0:01:51.1]} 155... Kb8 {[%clk 0:02:34.4]} 156. Ka2 {[%clk 0:01:51.6]} 156... Kc8 {[%clk
 0:02:35.2]} 157. Kb3 {[%clk 0:01:52.3]} 157... Kb8 {[%clk 0:02:36]} 158. Ka4 {[%clk
 0:01:52.8]} 158... Kc8 {[%clk 0:02:36.9]} 159. Kb5 {[%clk 0:01:53]} 159... Kb8 {[%clk
 0:02:37.8]} 160. Kc6 {[%clk 0:01:52.9]} 160... Kc8 {[%clk 0:02:38.7]} 161. Qb7+ {[%clk
 0:01:50.7]} 161... Kd8 {[%clk 0:02:38.7]} 162. a7 {[%clk 0:01:50.9]} 162... Ke8 {[%clk
 0:02:38.9]} 163. Kd5 {[%clk 0:01:50.7]} 163... Kd8 {[%clk 0:02:39.5]} 164. Ke4 {[%clk
 0:01:51.2]} 164... Ke8 {[%clk 0:02:40.2]} 165. Kf3 {[%clk 0:01:51.8]} 165... Kd8 {[%clk
 0:02:41]} 166. Ke2 {[%clk 0:01:52.2]} 166... Ke8 {[%clk 0:02:41.9]} 167. Kd1 {[%clk
 0:01:52.4]} 167... Kd8 {[%clk 0:02:42.8]} 168. Kc2 {[%clk 0:01:52.7]} 168... Ke8 {[%clk
 0:02:43.4]} 169. Kb3 {[%clk 0:01:53.3]} 169... Kd8 {[%clk 0:02:44.1]} 170. Ka4 {[%clk
 0:01:53.9]} 170... Ke8 {[%clk 0:02:44.8]} 171. Kb5 {[%clk 0:01:54.5]} 171... Kd8 {[%clk
 0:02:45.5]} 172. Kc5 {[%clk 0:01:55.1]} 172... Ke8 {[%clk 0:02:46.3]} 173. Kd4 {[%clk
 0:01:55.6]} 173... Kd8 {[%clk 0:02:47]} 174. Ke5 {[%clk 0:01:56.4]} 174... Ke8 {[%clk
 0:02:47.7]} 175. Kf4 {[%clk 0:01:57.2]} 175... Kd8 {[%clk 0:02:48.5]} 176. Ke3 {[%clk
 0:01:57.8]} 176... Ke8 {[%clk 0:02:49.2]} 177. Kf2 {[%clk 0:01:58]} 177... Kd8 {[%clk
 0:02:49.8]} 178. Ke1 {[%clk 0:01:57.9]} 178... Ke8 {[%clk 0:02:50.5]} 179. Kd2 {[%clk
 0:01:57.7]} 179... Kd8 {[%clk 0:02:51.4]} 180. a6 {[%clk 0:01:57.2]} 180... Ke8 {[%clk
 0:02:51.9]} 181. Kc3 {[%clk 0:01:57.5]} 181... Kd8 {[%clk 0:02:52.4]} 182. Kd4 {[%clk
 0:01:58.1]} 182... Ke8 {[%clk 0:02:52.9]} 183. Ke3 {[%clk 0:01:58.7]} 183... Kd8 {[%clk
 0:02:53.2]} 184. Kf4 {[%clk 0:01:59.6]} 184... Ke8 {[%clk 0:02:53.8]} 185. Kg5 {[%clk

0:02:00.4}} 185... Kd8 {[%clk 0:02:54.3]} 186. Kh6 {[%clk 0:02:01]} 186... Ke8 {[%clk
 0:02:55.1]} 187. Kh5 {[%clk 0:02:01.6]} 187... Kd8 {[%clk 0:02:55.8]} 188. Kg4 {[%clk
 0:02:02.1]} 188... Ke8 {[%clk 0:02:56.5]} 189. Kf3 {[%clk 0:02:02.8]} 189... Kd8 {[%clk
 0:02:57.3]} 190. Ke4 {[%clk 0:02:03.6]} 190... Ke8 {[%clk 0:02:58]} 191. Kd3 {[%clk
 0:02:04.2]} 191... Kd8 {[%clk 0:02:58.9]} 192. Kc4 {[%clk 0:02:04.3]} 192... Ke8 {[%clk
 0:02:59.8]} 193. Kb5 {[%clk 0:02:04.5]} 193... Kd8 {[%clk 0:03:00.7]} 194. Kc6 {[%clk
 0:02:04.1]} 194... Ke8 {[%clk 0:03:01.6]} 195. Qd7+ {[%clk 0:02:03.7]} 195... Kf8 {[%clk
 0:03:01.7]} 196. b7 {[%clk 0:02:03.6]} 196... Kg8 {[%clk 0:03:01.7]} 197. Kb6 {[%clk
 0:02:03.7]} 197... Kf8 {[%clk 0:03:02.3]} 198. Kc5 {[%clk 0:02:04.3]} 198... Kg8 {[%clk
 0:03:03.1]} 199. Kd4 {[%clk 0:02:04.8]} 199... Kf8 {[%clk 0:03:03.9]} 200. Ke3 {[%clk
 0:02:05.4]} 200... Kg8 {[%clk 0:03:04.4]} 201. Kd2 {[%clk 0:02:06.1]} 201... Kf8 {[%clk
 0:03:05.1]} 202. Ke1 {[%clk 0:02:06.7]} 202... Kg8 {[%clk 0:03:05.8]} 203. Kf2 {[%clk
 0:02:07.3]} 203... Kf8 {[%clk 0:03:06.7]} 204. Kg3 {[%clk 0:02:07.7]} 204... Kg8 {[%clk
 0:03:07.2]} 205. Kh4 {[%clk 0:02:08.3]} 205... Kf8 {[%clk 0:03:08.1]} 206. Kg4 {[%clk
 0:02:08.6]} 206... Kg8 {[%clk 0:03:09]} 207. Kf3 {[%clk 0:02:08.7]} 207... Kf8 {[%clk
 0:03:09.9]} 208. Ke4 {[%clk 0:02:09]} 208... Kg8 {[%clk 0:03:10.8]} 209. Kd3 {[%clk
 0:02:09.1]} 209... Kf8 {[%clk 0:03:11.6]} 210. Kc4 {[%clk 0:02:08.9]} 210... Kg8 {[%clk
 0:03:12.5]} 211. Kb5 {[%clk 0:02:09.2]} 211... Kf8 {[%clk 0:03:13.3]} 212. Kc6 {[%clk
 0:02:09.7]} 212... Kg8 {[%clk 0:03:14]} 213. Kc7 {[%clk 0:02:08.1]} 213... Kf8 {[%clk
 0:03:14.9]} 214. Kc8 {[%clk 0:02:08.4]} 214... Kg8 {[%clk 0:03:15.8]} 215. b8=Q {[%clk
 0:02:07.7]} 215... Kf8 {[%clk 0:03:16.7]} 216. Qbc7 {[%clk 0:02:05.5]} 216... Kg8 {[%clk
 0:03:17.6]} 217. Kb7 {[%clk 0:02:05.5]} 217... Kf8 {[%clk 0:03:18.5]} 218. Kb6 {[%clk
 0:02:06.1]} 218... Kg8 {[%clk 0:03:19.4]} 219. Ka5 {[%clk 0:02:06.6]} 219... Kf8 {[%clk
 0:03:20.3]} 220. Ka4 {[%clk 0:02:07.2]} 220... Kg8 {[%clk 0:03:21.2]} 221. Ka3 {[%clk
 0:02:07.7]} 221... Kf8 {[%clk 0:03:22.1]} 222. Ka2 {[%clk 0:02:08]} 222... Kg8 {[%clk
 0:03:23]} 223. Ka1 {[%clk 0:02:08.4]} 223... Kf8 {[%clk 0:03:23.9]} 224. Kb1 {[%clk
 0:02:08.8]} 224... Kg8 {[%clk 0:03:24.8]} 225. Kb2 {[%clk 0:02:09.1]} 225... Kf8 {[%clk
 0:03:25.7]} 226. Kb3 {[%clk 0:02:09.6]} 226... Kg8 {[%clk 0:03:26.6]} 227. Kb4 {[%clk
 0:02:10]} 227... Kf8 {[%clk 0:03:27.5]} 228. Kb5 {[%clk 0:02:10.5]} 228... Kg8 {[%clk
 0:03:28.4]} 229. Kb6 {[%clk 0:02:11]} 229... Kf8 {[%clk 0:03:29.3]} 230. Kb7 {[%clk
 0:02:11.3]} 230... Kg8 {[%clk 0:03:30.2]} 231. Kb8 {[%clk 0:02:10.6]} 231... Kf8 {[%clk
 0:03:31.1]} 232. a8=Q {[%clk 0:02:07.4]} 232... Kg8 {[%clk 0:03:32]} 233. Kc8 {[%clk
 0:02:06.3]} 233... Kf8 {[%clk 0:03:32.9]} 234. Qab8 {[%clk 0:02:01.1]} 234... Kg8 {[%clk
 0:03:33.8]} 235. Qbb7 {[%clk 0:02:00.3]} 235... Kf8 {[%clk 0:03:34.7]} 236. Kb8 {[%clk
 0:01:59.8]} 236... Kg8 {[%clk 0:03:35.6]} 237. Ka7 {[%clk 0:01:59.4]} 237... Kf8 {[%clk
 0:03:36.5]} 238. Kb6 {[%clk 0:01:58.2]} 238... Kg8 {[%clk 0:03:37.4]} 239. Kb5 {[%clk
 0:01:58.4]} 239... Kf8 {[%clk 0:03:38.3]} 240. Kb4 {[%clk 0:01:58.9]} 240... Kg8 {[%clk
 0:03:39.2]} 241. Kb3 {[%clk 0:01:59.4]} 241... Kf8 {[%clk 0:03:40.1]} 242. Kb2 {[%clk
 0:02:00]} 242... Kg8 {[%clk 0:03:41]} 243. Kb1 {[%clk 0:02:00.6]} 243... Kf8 {[%clk
 0:03:41.9]} 244. Kc1 {[%clk 0:02:01.1]} 244... Kg8 {[%clk 0:03:42.8]} 245. Kc2 {[%clk
 0:02:01.7]} 245... Kf8 {[%clk 0:03:43.7]} 246. Kc3 {[%clk 0:02:02.2]} 246... Kg8 {[%clk
 0:03:44.6]} 247. Kc4 {[%clk 0:02:02.8]} 247... Kf8 {[%clk 0:03:45.5]} 248. Kc5 {[%clk
 0:02:03.3]} 248... Kg8 {[%clk 0:03:46.4]} 249. Kc6 {[%clk 0:02:03.7]} 249... Kf8 {[%clk

0:03:47.3}} 250. Kd6 {[%clk 0:02:04.2]} 250... Kg8 {[%clk 0:03:48.2]} 251. Kd5 {[%clk
 0:02:04.6]} 251... Kf8 {[%clk 0:03:49.1]} 252. Kd4 {[%clk 0:02:05.1]} 252... Kg8 {[%clk
 0:03:50]} 253. Kd3 {[%clk 0:02:05.6]} 253... Kf8 {[%clk 0:03:50.9]} 254. Kd2 {[%clk
 0:02:06.1]} 254... Kg8 {[%clk 0:03:51.8]} 255. Kd1 {[%clk 0:02:06.5]} 255... Kf8 {[%clk
 0:03:52.7]} 256. Ke1 {[%clk 0:02:06.2]} 256... Kg8 {[%clk 0:03:53.6]} 257. Ke2 {[%clk
 0:02:06.6]} 257... Kf8 {[%clk 0:03:54.5]} 258. Ke3 {[%clk 0:02:07.1]} 258... Kg8 {[%clk
 0:03:55.4]} 259. Ke4 {[%clk 0:02:07.6]} 259... Kf8 {[%clk 0:03:56.3]} 260. Ke5 {[%clk
 0:02:08]} 260... Kg8 {[%clk 0:03:57.2]} 261. Ke6 {[%clk 0:02:07.6]} 261... Kf8 {[%clk
 0:03:58.1]} 262. Kf6 {[%clk 0:02:07.6]} 262... Kg8 {[%clk 0:03:59]} 263. Kf5 {[%clk
 0:02:07.3]} 263... Kf8 {[%clk 0:03:59.9]} 264. Kf4 {[%clk 0:02:06.7]} 264... Kg8 {[%clk
 0:04:00.8]} 265. a7 {[%clk 0:02:06.5]} 265... Kf8 {[%clk 0:04:01.7]} 266. Kf3 {[%clk
 0:02:06.3]} 266... Kg8 {[%clk 0:04:02.6]} 267. Kf2 {[%clk 0:02:06.9]} 267... Kf8 {[%clk
 0:04:03.5]} 268. Kf1 {[%clk 0:02:07.2]} 268... Kg8 {[%clk 0:04:04.4]} 269. Kg1 {[%clk
 0:02:07.6]} 269... Kf8 {[%clk 0:04:05.3]} 270. Kg2 {[%clk 0:02:08.1]} 270... Kg8 {[%clk
 0:04:06.2]} 271. Kg3 {[%clk 0:02:08.3]} 271... Kf8 {[%clk 0:04:07.1]} 272. Kf4 {[%clk
 0:02:08.6]} 272... Kg8 {[%clk 0:04:08]} 273. Ke4 {[%clk 0:02:08.9]} 273... Kf8 {[%clk
 0:04:08.9]} 274. Kd4 {[%clk 0:02:09.1]} 274... Kg8 {[%clk 0:04:09.8]} 275. Kc4 {[%clk
 0:02:09.6]} 275... Kf8 {[%clk 0:04:10.7]} 276. Kb5 {[%clk 0:02:10.1]} 276... Kg8 {[%clk
 0:04:11.6]} 277. Ka6 {[%clk 0:02:10.7]} 277... Kf8 {[%clk 0:04:12.5]} 278. Qf3+ {[%clk
 0:02:09.1]} 278... Kg8 {[%clk 0:04:13.4]} 279. Qcg3+ {[%clk 0:02:07.7]} 279... Kh8
 {[%clk 0:04:14.3]} 280. Qe8+ {[%clk 0:02:02.8]} 280... Kh7 {[%clk 0:04:14.5]} 281.
 Qgg6# {[%clk 0:01:58.8]} 1-0

Appendix E

```
import pandas as pd

df = pd.read_csv("combined_stockfish_results.csv")

# Convert accuracy
df["Accuracy (White)"] = pd.to_numeric(df["Accuracy (White)"], errors="coerce")
df["Accuracy (Black)"] = pd.to_numeric(df["Accuracy (Black)"], errors="coerce")
df["Move Count"] = pd.to_numeric(df["Move Count"], errors="coerce")

# filter based on move count
df = df[(df["Move Count"] >= 25) & (df["Move Count"] <= 100)]

# only keep games where combined accuracy min 185
df["Total Accuracy"] = df["Accuracy (White)"] + df["Accuracy (Black)"]
df = df[df["Total Accuracy"] >= 185]

# max accuracy
df["Max Accuracy"] = df[["Accuracy (White)", "Accuracy (Black)"]].max(axis=1)

# sort by max accuracy and then move count
df_sorted = df.sort_values(by=["Max Accuracy", "Move Count"], ascending=[False, False])

# select top 100 and save
top_100 = df_sorted.head(100)
top_100.to_csv("top_100_high_accuracy_games.csv", index=False)
```

Appendix F

[Event "Live Chess"]

[Site "Chess.com"]

[Date "2024.01.30"]

[Round "?"]

[White "MagnusCarlsen"]

[Black "HansOnTwitch"]

[Result "1/2-1/2"]

[TimeControl "180+1"]

[WhiteElo "3299"]

[BlackElo "3214"]

[Termination "Game drawn by repetition"]

[ECO "C67"]

[EndTime "17:57:02 GMT+0000"]

[Link "<https://www.chess.com/game/live/100376111035?move=0>"]

1. e4 e5 2. Nf3 Nc6 3. Bb5 Nf6 4. O-O Nxe4 5. Re1 Nd6 6. Nxe5 Be7 7. Bf1 Nxe5 8.
Rxe5 O-O 9. d4 Bf6 10. Re1 Re8 11. c3 Rxe1 12. Qxe1 Ne8 13. Bf4 d5 14. Bd3 g6
15. Nd2 Ng7 16. Nf3 Bf5 17. Bxf5 Nxf5 18. Qe2 c6 19. Re1 Ng7 20. Be5 Ne6 21.
Bxf6 Qxf6 22. Ne5 Re8 23. Ng4 Qd8 24. Qe5 Ng7 25. Qxe8+ Nxe8 26. Rxe8+ Qxe8 27.
Nf6+ Kf8 28. Nxe8 Kxe8 29. f4 f5 30. Kf2 Kf7 31. b4 Ke6 32. Ke3 b5 33. Kf3 Kf6
34. Kg3 h6 35. h4 h5 36. Kf3 Ke6 37. Ke3 Kd6 38. Kf3 Kc7 39. Ke3 Kb6 40. a3 a5
41. Kd3 Ka6 42. Kc2 a4 43. Kd3 Kb6 44. Ke3 Kc7 45. Kf2 Kd7 46. g3 Ke6 47. Ke3
Kf7 48. Kd3 Kf6 49. Ke3 Ke6 50. Kd3 Kd7 51. Ke3 Kc8 52. Kd3 Kb8 53. Ke3 Ka8 54.
Kd3 Ka7 55. Ke3 Ka6 56. Kd3 Kb6 57. Ke3 Kb7 58. Kd3 Kc7 59. Ke3 Kc8 60. Kd3 Kd8
61. Ke3 Ke8 62. Kd3 Kf8 63. Ke3 Kg8 64. Kd3 Kh8 65. Ke3 Kh7 66. Kd3 Kh6 67. Ke3
Kg7 68. Kd3 Kf6 69. Ke3 Kf7 70. Kd3 Ke6 71. Ke3 Ke7 72. Kd3 Kd6 73. Ke3 Kd7 74.
Kd3 Kc7 75. Ke3 Kb6 76. Kd3 Ka6 77. Ke3 Ka7 78. Kd3 Kb7 79. Ke3 Ka8 80. Kd3 Kb8
81. Ke3 Kc7 82. Kd3 Kc8 83. Ke3 Kd7 84. Kd3 Kd8 85. Ke3 Ke7 86. Kd3 Ke8 87. Ke3
Kf7 1/2-1/2

Appendix G

```
import pandas as pd

df = pd.read_csv("combined_stockfish_results.csv")

# Convert accuracy
df["Accuracy (White)"] = pd.to_numeric(df["Accuracy (White)"], errors="coerce")
df["Accuracy (Black)"] = pd.to_numeric(df["Accuracy (Black)"], errors="coerce")
df["Move Count"] = pd.to_numeric(df["Move Count"], errors="coerce")

# Filter based on move count
df = df[(df["Move Count"] >= 25) & (df["Move Count"] <= 100)]

# Only keep games where combined accuracy is at least 185
df["Total Accuracy"] = df["Accuracy (White)"] + df["Accuracy (Black)"]
df = df[df["Total Accuracy"] >= 185]

# Filter out drawn games
df = df[df["Result"] != "1/2-1/2"]

# Max accuracy
df["Max Accuracy"] = df[["Accuracy (White)", "Accuracy (Black)"]].max(axis=1)

# Sort by max accuracy and then move count
df_sorted = df.sort_values(by=["Max Accuracy", "Move Count"], ascending=[False, False])

# Select top 100 and save
top_100 = df_sorted.head(100)
top_100.to_csv("top_100_high_accuracy_games.csv", index=False)
```


Appendix H

[Event "**** Titled Tuesday Blitz"]

[Site "Chess.com"]

[Date "2020.04.28"]

[Round "8"]

[White "Frogo47"]

[Black "vohiraa"]

[Result "1-0"]

[WhiteElo "2492"]

[BlackElo "2218"]

[TimeControl "180+1"]

[EndTime "18:35:28 GMT+0000"]

[Termination "Frogo47 won by checkmate"]

1. e4 {[%clk 0:03:00.9]} 1... c5 {[%clk 0:02:59.6]} 2. Nf3 {[%clk 0:03:00.4]}
2... d6 {[%clk 0:02:59.8]} 3. d4 {[%clk 0:02:57.6]} 3... cxd4 {[%clk 0:03:00.7]}
4. Nxd4 {[%clk 0:02:54.4]} 4... Nf6 {[%clk 0:03:00.2]} 5. Nc3 {[%clk 0:02:54.9]}
5... a6 {[%clk 0:03:00.1]} 6. Bg5 {[%clk 0:02:50.3]} 6... e6 {[%clk 0:02:58.8]}
7. f4 {[%clk 0:02:49.7]} 7... h6 {[%clk 0:02:58.6]} 8. Bh4 {[%clk 0:02:48.3]}
8... Qb6 {[%clk 0:02:58]} 9. a3 {[%clk 0:02:34.8]} 9... Be7 {[%clk 0:02:57.4]}
10. Bf2 {[%clk 0:02:31.6]} 10... Qc7 {[%clk 0:02:57.2]} 11. Qf3 {[%clk
0:02:23.2]} 11... Nbd7 {[%clk 0:02:48.4]} 12. O-O-O {[%clk 0:02:16.9]} 12... b5
{[%clk 0:02:46]} 13. g4 {[%clk 0:02:14]} 13... g5 {[%clk 0:02:45.4]} 14. h4
{[%clk 0:02:07.9]} 14... gxf4 {[%clk 0:02:43.7]} 15. g5 {[%clk 0:01:53.8]} 15...
hxc5 {[%clk 0:02:40.7]} 16. hxc5 {[%clk 0:01:46.6]} 16... Rxh1 {[%clk
0:02:37.7]} 17. Qxh1 {[%clk 0:01:46.5]} 17... Ng4 {[%clk 0:02:37]} 18. g6 {[%clk
0:01:27.4]} 18... fxg6 {[%clk 0:01:39.2]} 19. Nxe6 {[%clk 0:01:26]} 19... Qb7
{[%clk 0:01:31.8]} 20. Nd5 {[%clk 0:01:22.8]} 20... Nf8 {[%clk 0:01:04.8]} 21.
Nec7+ {[%clk 0:01:19.7]} 21... Kd8 {[%clk 0:00:58.6]} 22. Bb6 {[%clk 0:01:08]}

22... Rb8 {[%clk 0:00:46.2]} 23. Ba5 {[%clk 0:00:58.8]} 23... Ne3 {[%clk 0:00:33.2]} 24. Nxe7 {[%clk 0:00:43.2]} 24... Kxe7 {[%clk 0:00:26.6]} 25. Qh8 {[%clk 0:00:42]} 25... Be6 {[%clk 0:00:17]} 26. Qg7+ {[%clk 0:00:32.2]} 26... Bf7 {[%clk 0:00:17.3]} 27. e5 {[%clk 0:00:17.8]} 27... Qe4 {[%clk 0:00:10.5]} 28. Qf6+ {[%clk 0:00:09.1]} 28... Kd7 {[%clk 0:00:09.4]} 29. Rxd6+ {[%clk 0:00:07.9]} 29... Kc8 {[%clk 0:00:08.9]} 30. Rd8+ {[%clk 0:00:06.5]} 30... Kb7 {[%clk 0:00:08.7]} 31. Qb6# {[%clk 0:00:07]} 1-0

Appendix I

```
import pandas as pd

# Load data
df = pd.read_csv("combined_stockfish_results.csv")

# Convert accuracy
df["Accuracy (White)"] = pd.to_numeric(df["Accuracy (White)"], errors="coerce")
df["Accuracy (Black)"] = pd.to_numeric(df["Accuracy (Black)"], errors="coerce")

# Filter for games where Frogo47 is playing
frogo_games = df[(df["White Player"] == "Frogo47") | (df["Black Player"] == "Frogo47")].copy()

# Frogo47 Accuracy column
frogo_games["Frogo47 Accuracy"] = frogo_games.apply(
    lambda row: row["Accuracy (White)"] if row["White Player"] == "Frogo47" else row["Accuracy (Black)"],
    axis=1
)

# Sort by Accuracy (descending)
frogo_games = frogo_games.sort_values(by="Frogo47 Accuracy", ascending=False)

# Save to CSV
frogo_games.to_csv("frogo47_games.csv", index=False)

print(f"Top game accuracy: {frogo_games['Frogo47 Accuracy'].iloc[0]}")
print(f"Total games: {len(frogo_games)}")
```

Appendix J

```
1  import pandas as pd
2
3  df_all = pd.read_csv("combined_stockfish_results.csv")
4  df_suspicious = pd.read_csv("suspicious_games_from_isolation_forest.csv")
5
6  # Combine suspicious games with player and Elo
7  white_part = df_suspicious[["White Player", "White Rating"]].rename(
8  |     columns={"White Player": "Player", "White Rating": "Elo"})
9  black_part = df_suspicious[["Black Player", "Black Rating"]].rename(
10 |     columns={"Black Player": "Player", "Black Rating": "Elo"})
11 all_flagged = pd.concat([white_part, black_part])
12
13 # total suspicious games and mean Elo in flagged games
14 suspicious_summary = all_flagged.groupby("Player").agg(
15 |     Suspicious_Games=("Player", "count"),
16 |     Mean_Elo_Flagged=("Elo", "mean")
17 )
18
19 # Count total games per player from the full dataset
20 all_players = pd.concat([
21 |     df_all["White Player"],
22 |     df_all["Black Player"]
23 ])
24 total_games = all_players.value_counts().rename("Total_Games")
25
26 # Merge into final table
27 summary = suspicious_summary.join(total_games, how="left")
28 summary["Suspicion_Rate"] = summary["Suspicious_Games"] / summary["Total_Games"]
29
30 # Sort by suspicion rate descending
31 summary_sorted = summary.sort_values(by="Suspicion_Rate", ascending=False)
32
33 # Show top results
34 print(summary_sorted.head(20))
35
36 summary_sorted.to_csv("suspicion_rate_summary.csv")
```

