Introduction

The goal of this project is to capture the move characteristics of human players at different skill levels and train a model that identifies player strength from pgn files. The model should be able to infer a rating of either player or both players from the moves of the game itself, with no additional information on the player's actual rating.

Chess players who engage in online play on platforms like chess.com and lichess.org or participate in real-life tournaments receive an Elo rating specific to each platform. This rating, ranging from less than 1000 for beginners to over 2400 for masters, fluctuates based on performance against other players. The goal is to develop a model capable of understanding differences across various player rating bands, such as 1000-1200, 1200-1400, and so forth. Uncovering systematic differences in moves and errors within these rating ranges holds the potential to enhance chess coaching strategies or refine human-centric aspects in engine evaluations. One existing project that is similar to what we are trying to achieve is Maia Chess. Maia Chess is a bot created using neural networks that also tries to predict the moves that human players at various skill levels are most likely to play. We'll be using a similar dataset (lichess database), but will be trying different approaches to representing chess data for the purpose of this project. Ultimately, the goal is to create a LSTM model that takes games in pgn format as input, and reliably returns the estimated level of the player.

Data Source

The dataset used in this study is sourced from the Lichess database, a comprehensive archive of online chess games freely available at (https://database.lichess.org/). For this specific analysis, the dataset of interest is a zipped folder of 33.5 GB from January 2023, containing a massive collection of 100 million games, of which 7% has engine analysis data that provides an objective computer assessment of the position after each move. The games are stored in Portable Game Notation (PGN), a standard text format that records the moves of chess games in a structured manner, along with metadata such as player ratings, time controls, and outcomes.

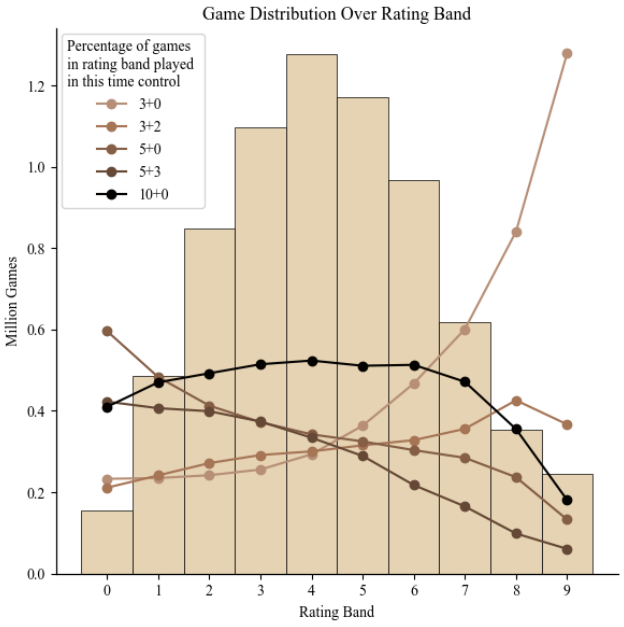

Figure 1

Reading directly from the downloaded zipped folder, initial preprocessing involved extracting games with engine analysis from the comprehensive dataset, a process that required parsing the large PGN file to identify and segregate games that contained the necessary engine annotations. The extracted games were then separately written to files accounting for allocated time per side (time control) and player strength (rating band). This is done to make sure our training data is as balanced as possible, and that biases in global data distribution affect our model prediction as little as possible. The balanced dataset that helped develop our model had an equal number of games for each rating band which consisted of a fixed proportion of the various time controls.

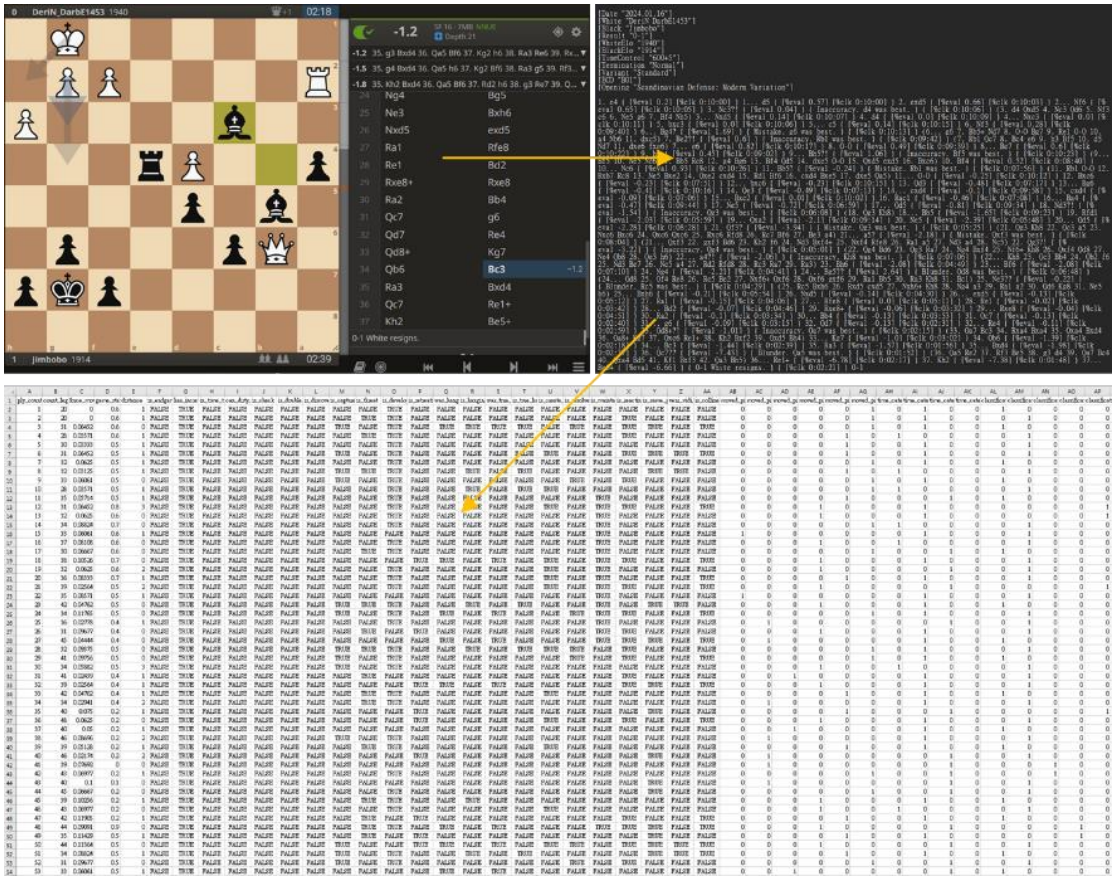

Figure 2

Feature Engineering

Each game in the 100,000-game balanced dataset is represented as an instance of the ChessGame class, capturing essential metadata such as player ratings, game result, and time control. The game's moves are processed sequentially, with each move represented as an instance of the ChessMove class. This class structure enables a detailed examination of each move within the context of the game, including the board state before and after the move, move characteristics (e.g., check, capture, piece development), and the strategic implications of the move (e.g., creating tension, resolving tension).

For every move (ChessMove instance), a comprehensive set of features is extracted to describe both the move itself and its context within the game. Key features can be further divided into external and intrinsic factors. Some external factors are time related, such as time spent on move or whether the player had low time, while other external features has to do with the computer assessment of the position, such as the quality of the move and the game state or winning probability for either side in a position. As for intrinsic factors, some require a bit of strategic understanding and domain knowledge while others are more straightforward. “Domain features” include whether a move is creating, maintaining, or resolving tension, whether it allows the opponent to capture the piece on the next turn, or whether it threatens to capture a higher value piece on the next turn. Meanwhile, the simpler “Piece features” encodes whether the move resulted in a check or capture, and which piece was moved and how many squares it traversed.

Once a ChessGame instance is populated by ChessMove objects, the relevant features are extracted from the moves, and prepared for input into the machine learning model.
Normalization: Certain features, like the number of legal moves and the distance moved, are normalized to ensure consistency across the dataset.
Encoding: Categorical data, such as the moved piece, time spent (e.g., "instant", "fast", "slow") and move quality (e.g., "Great", "Good", "Mistake"), are split into columns.
Aggregation: Features that have cross-move implications are added. These include whether a move is a reaction to the opponent’s previous move, whether the piece moved is the same piece as last turn, or whether a capture is premeditated by one’s previous move, which initiated an attack on the later captured piece.
Transformation: The final step involves converting the aggregated game data into a numpy array format that is compatible with our LSTM model.

Supervised Learning

I. Methods Description

How might chess players at different levels play the game differently? The collection of all legal chess moves is quite limited, with each move placing one of six types of pieces onto one of 64 squares, the underlying purpose behind each move heavily depends on the context and not the annotation, or spelling, of the move itself. To extend this analogy, imagine each chess game a document, and each move a word. Dialogue between the white and black king is encased within the confines of a lexicon already limited, yet for each turn further constrained to a narrower set of its former breadth. Behind every move played is a thousand thoughts unsaid, that which distinguishes the novice from the veteran. In these subtleties lie the essence of chess mastery, harnessing which to make rating predictions is the objective of our model.

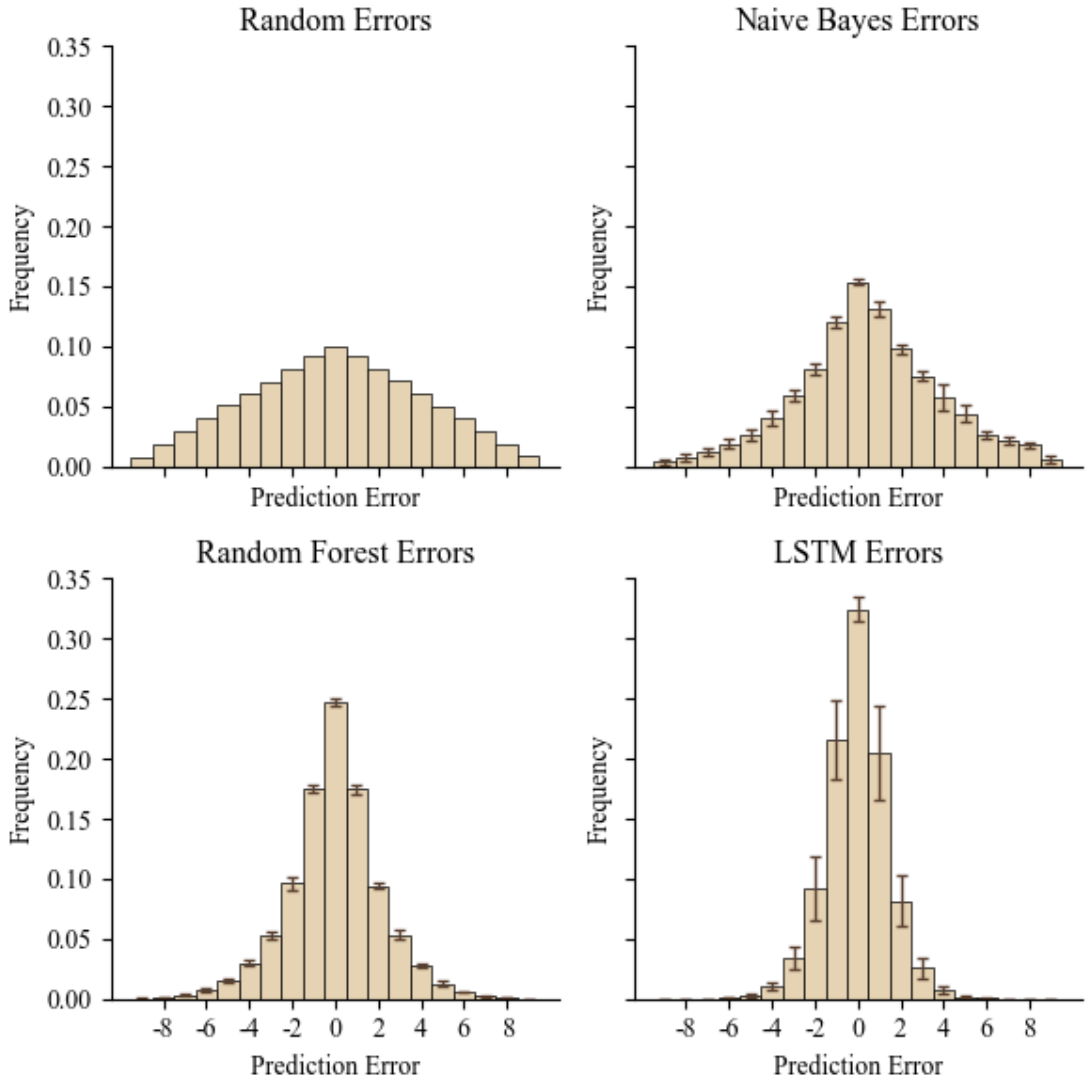

Figure 3

How can we evaluate the performance of a model that predicts player strength based on the game they played? The poetry of chess aside, we must establish baselines to compare our progress against. Derived from their ratings, we divide chess players into 10 classes in increasing order of mastery. [FIG] shows the distribution of errors of model predictions versus the ground truth. Predicting class 4 while the true class is 2 yields a prediction error of 2, and 0 while the ground truth is 9 yields -9. All four error distributions are unimodal, but with decreasing standard deviations as model complexity increase. The input dimensions to these models are 0-D for Random Error, 1-D for Naïve Bayes and Random Forests, and 2-D for LSTM, with the 1-D input being the vector of feature-wise averages of all moves in a game. On the top left, we have truly random predictions, which in theory predicts all classes with uniform probability regardless of input. This is our minimum baseline. Next, we have a naïve bayes probabilistic classifier that infers Bayesian probabilities from training data and assumes

independence among predictors. This is our baseline to judge whether our final model is adequately learning from feature interactions. The Random Forest model is a tree-based approach that improves prediction accuracy from feature importance and interaction, which will be the third baseline for our model. The final model of choice we settled on is an LSTM model, a great choice for modeling entire sequences of data, and we'll compare its results to Random Forest to see how the finer granularities of sequential vocabulary matrix data instead of document averages vectors contributes to the prediction task.

The Long Short-Term Memory (LSTM) network model architecture comprises of three LSTM layers stacked on top of each other, designed to capture temporal dependencies within the sequential data of chess moves with higher levels of abstraction, leveraging the LSTM's ability to remember long-term patterns. It is followed by seven Fully Connected (FC) layers that can identify complex relationships between the features extracted by the LSTM output to make predictions or classifications. There are two output layers—one for classification and one for regression tasks. Due to the ordinal nature of our ground truth labels, we're interested in maximizing prediction accuracy, but also minimizing prediction error values. The loss function combines the classification and regression losses to punish more significant errors in predictions that are far off the mark. The training process involves iterating over epochs, calculating the combined loss, and adjusting the model weights accordingly through backpropagation.

II. Supervised Evaluation

Our models are judged against the following metrics of performance measurement, raw accuracy and mean cumulative accuracy. Raw accuracy is the percentage of time the model correctly predicts the ground truth label, while the mean cumulative accuracy metric takes into consideration the average model accuracy for every unique margin of error.

	err = 0	err ≤ 1	err ≤ 2	err ≤ 3	err ≤ 4
Uniform Random	10.00% (0.00%)	28.38%	44.63%	58.80%	70.88%
Naïve Bayes	15.34% (0.13%)	40.42%	58.34%	71.80%	81.56%
Linear Reg.	20.19% (0.33%)	55.46%	81.09%	94.01%	98.60%
Random Forest	24.70% (0.17%)	59.75%	78.79%	89.41%	95.06%
LSTM	33.22% (0.15%)	75.06%	92.28%	97.90%	99.47%

Figure 4

All models, otherwise specified, are trained with 5-fold cross validation over 100,000 games. We utilized the sklearn implementation of the three intermediary models, Naïve Bayes, Linear Regression, and Random Forest, and collapsed each game matrix into a vector by calculating the mean of each column across rows. We used pyTorch to build our LSTM model, which took the entire game matrix as input.

By both metrics, the LSTM model achieves the best performance. It correctly predicts the ground truth label 1 in 3 games, and its prediction falls within ± 1 of the correct label 3 out of 4 games. Its prediction is off by more than 2 every 1 in 12 games, and off by more than 3 once every 40 games. As model complexity increases, the distribution of errors rapidly converges to the center (*Figure 5*), and the confusion matrix below highlights a diagonal strip of probable predictions centered around the actual labels.

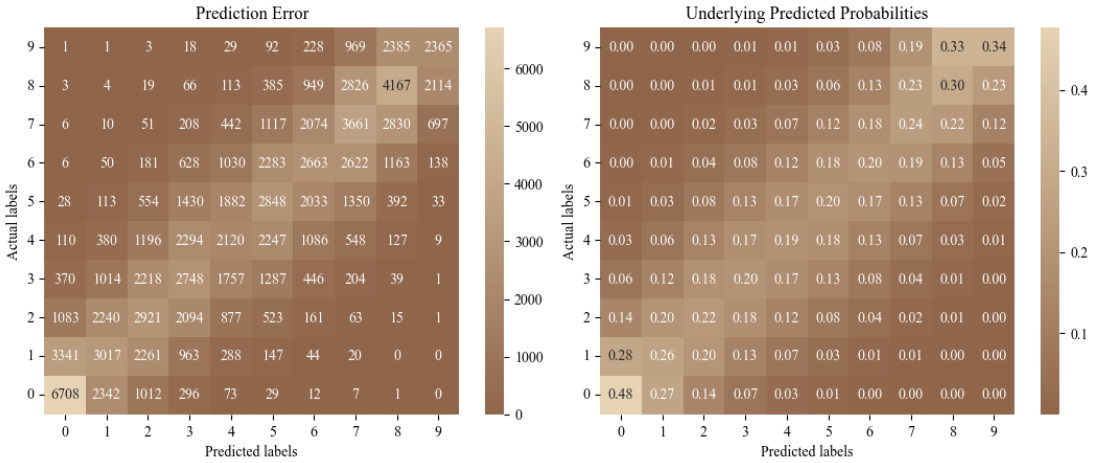

Figure 5

Feature Analysis

We implemented the IntegratedGradients attribute of the captum library, which is based on attributing the contribution of each input feature to the output. This feature analysis method compares against a baseline, a perfect fit for our model that pads shorter games. *Figure 6* shows the average feature analysis over a sample of 1000 games and 3 specific game instances (two of which were padded). Below, we will compare the effects of the 42 features on our LSTM model, Random Forest model, and Linear Regression model.

The distribution of feature significance of our Random Forest model is quite uniform, with the most influential feature at 3.56% and the least influential of the top 20% at 2.95%, and each of the top 20% most significant RF features can also be found in the top and bottom 10 features of the LSTM model. As for the Linear Regression model, which had its output rounded and clipped to match the class labels, it most significant coefficients shares 6 out of both the top and bottom 10 features with the LSTM model.

While the IntegratedGradients outputs the feature impact on each move, the Random Forest and Linear Regression models take vector means as input. Let's average the LSTM feature impact across moves to make a comparison. On average, "Blunders", "Captured a piece", "White is winning", and "Played same piece as previous move" are negatively correlated with player strength, while "Great move", "Instant move", and "Number of legal moves in position" are positively correlated with player strength.

Other features that are great predictors of strong players include distance of moves, frequency a move creates or resolves tension, whether the moved piece seem to be able to be captured by the opponent, and whether it is a developing move. On the other hand, the opponent having mere seconds on their clock with no increment, the frequency of moves being checks, whether a piece moved to a square that can legally be captured, and whether the same piece that moved on the previous move now captures an enemy piece are, on average, solid indicators of weak play.

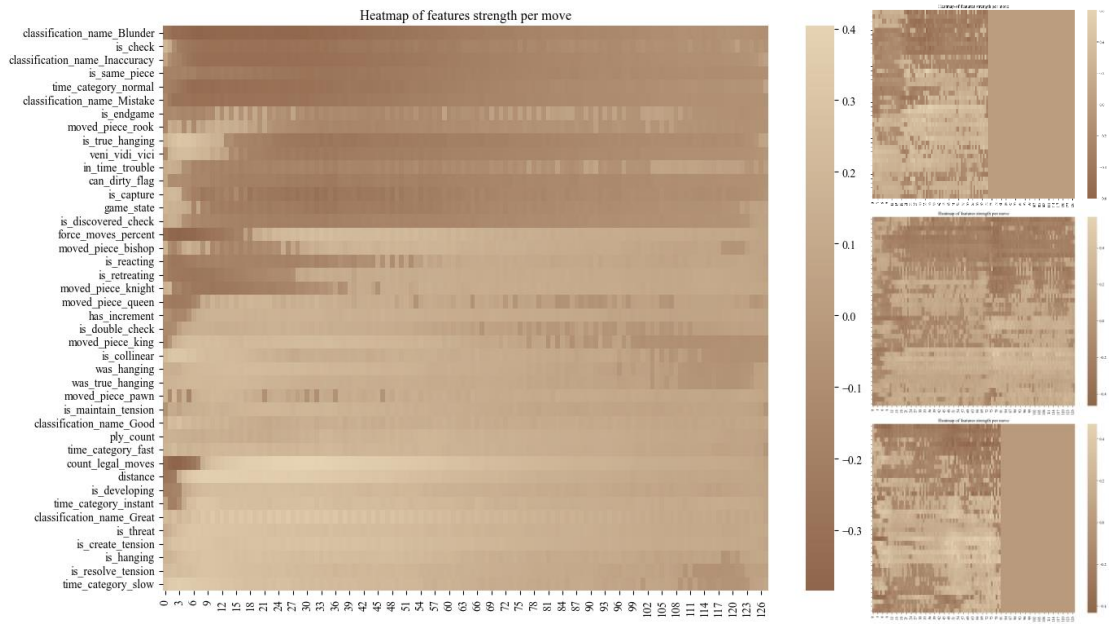

Figure 6

The benefit of analyzing chess games with a recurrent neural network is preservation of the granularities in move details and trends over time. The three smaller heatmaps on the right column in *Figure 6* are specific single instances of games from lower, medium, and high rating bands. The first and third heatmaps has a “padding” rectangle, which is the baseline for the IntegratedGradients computation. Looking at how each feature changes color move by move for specific games may provide more insight and credibility to the model’s justifications and explainability, which lays out how combinations of different features on different moves may convey different meanings in terms of move strength. In terms of general trends, let’s shift our focus to the larger figure on the left. This is extensively insightful for each feature, but we’ll constrain ourselves to summarize three generalizable points. First, longer games are lightly correlated with player strength; second, some features are indicative of weak play in the opening, but are otherwise good traits that are suggestive of player strength as the game progresses to the late opening or middlegame, or even the endgame; third, some features that, on average, correlates with inaccurate play, may tell of strength in the right conditions, such as sacrificing material for activity (gambits) in the opening.

Ablation Analysis

We separated our 42 features into 2 categories, with 2 groups in each. Inherent Features are features that are characterized by the move itself, such as the piece moved (moved-piece feature) or whether it created or released tension (domain-knowledge feature). External Features are either related to the time or engine, such as whether a move was made in time trouble (time), or whether a move is a Blunder (engine).

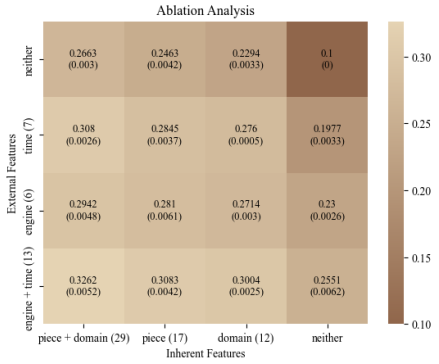

Figure 7

Our ablation analysis shows that the predictive strength of our model is robust. Were any group of feature to be dropped, the model performance would only drop by 2%. The strongest set of predictive features are the 17 piece features, themselves able to correctly predict the ground truth label 24.6% of the time, closely followed by the 6 engine features at 23%. However, the marginal benefit of engine isn't as pronounced as one would otherwise assume, meaning future analysis could be done on games without computer analysis, which not only increases the potential dataset size from the same source by 14 times, but also eliminates the necessity of a great deal of preprocessing effort. Overall, all four groups contribute to the model's predictive performance, and our model judges each game with a holistic view. Moreover, such robustness echoes the relatively uniform feature importance of observed in Random Forest classifier.

Learning Curve Analysis

We did stratified sampling on our 100,000 game dataset to synthesize batches of game samples starting from 100 and scaling up approximately 1.6x each step. Up until the sample size reached 1600 records, the LSTM model was unable to escape the local minimum defined in the loss function, and always predicted a single label for all inputs. However, the model finally started learning from the input when the sample size reached 2500 games, and started to predict a few different labels for the next three steps. By the time our sample size reached 10,000 games, the model predicted labels from each rating band most of the time, and its performance has steadily and nearly linearly improved up until 100,000 games. As expected, the 95% confidence interval shrinks as sample size increases, and intervals tend to widen for larger margin of errors across the table.

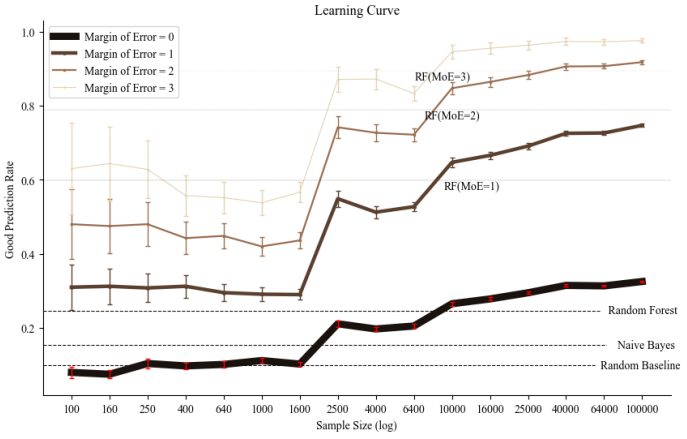

Figure 8

Sensitivity Analysis

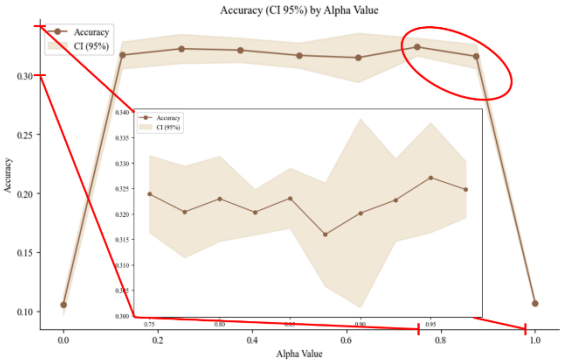

Figure 9

The flexibility inherent in our model's design presented significant challenges when attempting to evaluate the myriad combinations of hyperparameters, a task that grows exponentially with each additional variable. In response to these challenges, we opted for a strategy of hyperparameter tuning that focused on eight key variables, recognizing that a comprehensive iteration through merely five potential values for each would necessitate an impractical two million training and testing cycles. Constrained by computational resources and time, we employed an iterative grid search methodology, applied in lower-dimensional spaces, to methodically tune an optimal subset of hyperparameters. This approach began with broad intervals between values under examination, progressively narrowing the focus to those intervals demonstrating the most potential, thereby pruning less promising hyperparameter combinations. Such a strategy aimed to identify a local optimum that exhibited both stability and robustness.

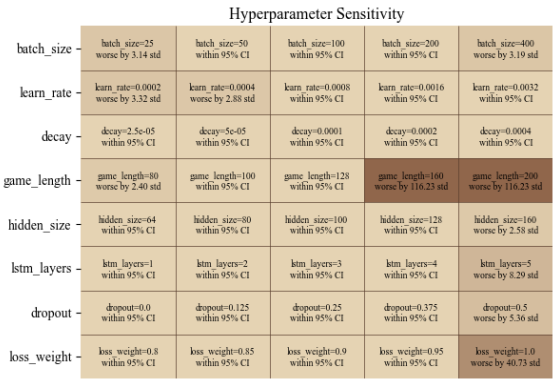

Figure 10

Our sensitivity analysis scrutinized the impact of each parameter on the model's prediction accuracy, employing varying scales of incrementation (2x, 1.25x, constant) across the parameters. Among the hyperparameters, there were a few usual suspects. Large hidden size and a high number of layers, or conversely, a low batch size, were found to contribute to overfitting and introduce excessive noise into the model. Similarly, a high dropout rate, an overly large batch size, or a low learning rate could impede the model's ability to detect crucial signals. Despite the general stability of most parameters within or slightly outside the 95% confidence interval for prediction accuracy, we identified two critical hyperparameters—game length and loss weight (alpha)—whose improper calibration could detrimentally impact model performance. The former, game length, pertains to the fixed number of moves extracted from each game; set it too high and the model becomes confounded by superfluous padding, leading to uniform predictions. The latter, loss weight, balances the classification and regression losses within the loss function, a crucial factor for guiding the model away from the "cold-start problem"—a scenario in which the model is trapped in a local minimum, making uniform label predictions across inputs. The sensitivity to game length is a major flaw of our model.

III. Failure analysis

Recall the darker shaded triangles in the top left and bottom right in *Figure 5*, these are the inaccurate classifications our model is currently making. In general, there are three kinds of mistakes, and we will now be diving into the cause and recommendations for each. Although misclassification come in many forms, one overarching theme persists, and that is the importance of the quality of opening moves, especially in short games.

Instance CL7kj0OG: Prediction – High rating band (7); Truth – Low rating band (1)

A chain is only as strong as its weakest link, and the strength of a player is constrained by the level of mistakes he makes. Since the input to our LSTM model does not include the board state or location of pieces, and we're using a database of relatively fast paced games, it does not necessarily follow that stronger players will play more accurately in the opening stages, nor do we have access to databases that document the specific styles in which players at different levels leave the well-trodden pavements of chess theory. As the adage goes, o*ne bad move nullifies 40 good ones.* Yet, for the aforementioned reasons, there are certain situations where a novice may assume the guise of a veteran, and reviewing the catalog of misclassifications, the inevitability of the "one bad move" can be circumvented in two ways. It always starts with playing a solid, compromising, and reserved opening. The moves may be inaccurate, but avoiding major blunders in the first 10 to 20 moves is the objective here. Well begun is half done, and the player is rewarded with a host of options at this juncture. Either blunder immediately and resign, spend a lot of time and head straight from the opening into time trouble, or, in a tacit coordinated effort with your opponent, lock up the position and fight at a later date. The only proxy for the complexity of position our model has as input is "num_legal_moves", but it fails to convey the relative ease to navigate closed positions and accumulate moves. To cater to a wide range of games, the model truncates moves from long games and pad moves to short games, and as long as the fight commences after the threshold move, mistakes that reveal the player's true strength will not be seen by our model, and our model would lack the evidence or confidence to consign the game to a low rating.

Instance nwNVI4SO: Prediction – Low rating band (1); Truth – High rating band (7)

Chess is as objective a game as it is fair. The name of a master itself does not guarantee victories over the board; it requires good moves and only good moves to breach the marble walls and granite forts. In stark contrast to the straight path to mastery, the ocean of opportunities and the endless possibilities for mistakes mean that misclassifications of this sort come in all shapes and sizes. To meticulously establish stringent definitions would be a ceaseless errand, yet the simplicity of this cumbersome task can be summed up no better than by the juxtaposition of the previous example in the opening moves.

Aggressively pushing pawns, attacking without mobilizing, repeatedly moving the same piece, or even speculative or downright refuted sacrifices, these are signs of weak moves, signs that a player, whatever his justification, is adhering to a subpar strategy at his own demise. Often, the master will vindicate his mischiefs by unrelentingly pressing with vigor and creativity after the opening stages, revealing his true strength in the process and allowing our model to correct its earlier assumptions, but were the game to abruptly terminate, be it disconnection, early blunders and resignation, or mouseslips, the nuances in how the mistakes of a self-sabotaging master differs from unsuspecting novices is not picked up by the model. It does not have the depth to understand the danger, or compensation in a position, for emotionless silicon calculators aren't known for their empathy towards a human, especially under the pressure of a ticking clock. In this particular example, White played a provocative opening. Pushing pawns, launching attacks, repeatedly moving his developed pieces, and neglecting development, all this to realize he is on the brink of defeat not even a dozen moves into the game, and then promptly engineering an ingenious series of exchanges that, ceteris paribus, should avert him of any troubles in an equal position. Such astute awareness, clean calculation, and creative commands were not lost upon keen observers of the game, yet in a twisted turn of fate, it is revealed that all that brilliance was but a valiant last stand, and White fell short by just one critical move. This intricacy did not reach the model. Instead, the model saw white cruise to defeat after playing without discipline in the opening and exchanging off all the pieces into a dead lost endgame and fighting till the bitter end, as is typical of players at lower levels, though black's conversion is typically rougher.

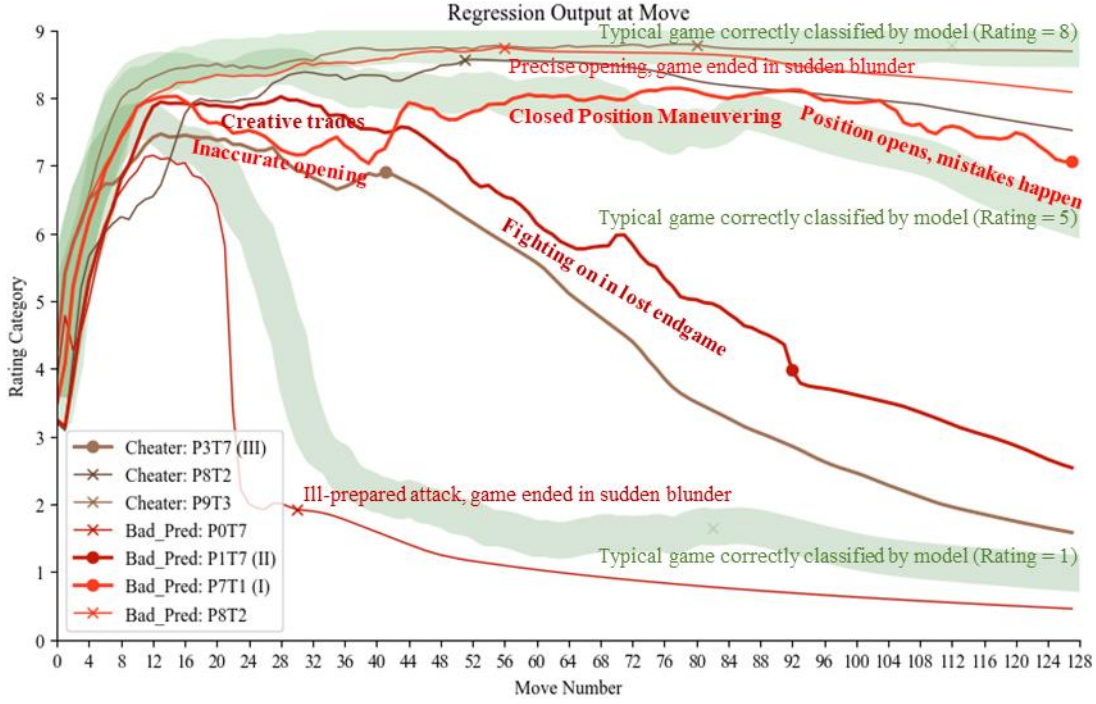

Figure 11

Instance 9agXlVUB: Cheating inconsistencies – Prediction (1) | Truth (7)

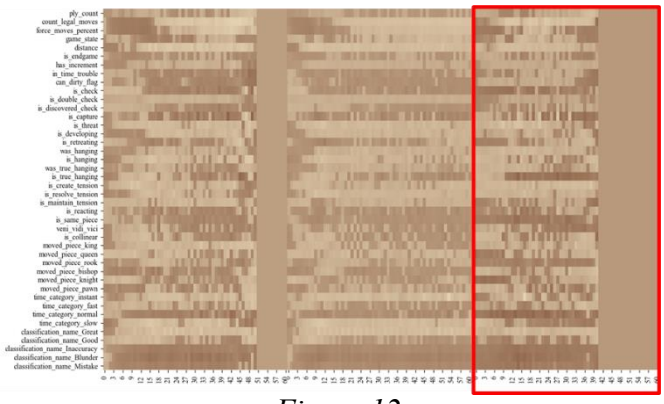

Figure 12

The relativity of chess extends from the rigidness of the game; abstractions and generalizations always work until they don't. Losing a queen is bad, but getting checkmated is worse. To endure a terrible fate so your opponent suffers one even worse is a staple strategy in chess. A discussion on computer chess and how it differs from natural human play is a fascinating topic that the following discussion can not do justice, but on a high level, computers are not indoctrinated by centuries of human chess wisdom, and may consider moves that, although possibly not profoundly difficult to calculate, simply does not emerge to human players as a candidate move in a position. *Figure 12* contains the feature breakdown by move for three games that likely employed engine assistance, and the third game, outlined in red, received a classification of rating band 1, despite being played between players in rating band 7 (and probably a computer). The third heatmap has distinguishably darker strips for features that generally suggest less mastery, and the suspected cheater did in fact repeatedly make moves that reaffirmed the model's waning confidence in his strength. Specifically, the suspected cheater made a lot of captures, kept shifting his knights around, hung material, and moved the same piece on multiple occasions. These moves were, of course, approved by the engine, but this playstyle, on the surface, resembles novice play rather than expert play, with the catch that all his moves received atypical blessings from the computer.