Estimating the Puzzlingness of Chess Puzzles

Sebastian Björkqvist
IPRally Technologies Oy
Helsinki, Finland
sebastian@iprally.com

***Abstract*—Solving chess puzzles, which require finding a certain sequence of moves to achieve a winning position (such as checkmate or a significant material advantage), is a commonly used method for improving chess skills, particularly tactical awareness. Estimating the puzzlingness—i.e., the difficulty—of a chess puzzle is challenging and typically involves showing the puzzle to multiple players and measuring their success rate. In this work, we present a method for predicting the difficulty of chess puzzles using only the initial position and the sequence of correct moves. We generate multiple features, including both hand-crafted features and those extracted from chess engines such as Maia, Leela Chess Zero, and Stockfish. We also train a residual neural network to directly predict the Glicko-2 puzzle rating. The output of this neural network, along with the other generated features, serves as input to a gradient boosting decision tree model to predict the final Glicko-2 rating. Our model was applied to the IEEE BigData 2024 Cup competition on Predicting Chess Puzzle Difficulty, where it achieved third place.**

***Index Terms*—chess, chess puzzle, machine learning**

I. Introduction

A chess puzzle is a particular state on a chessboard where the puzzle solver takes the place of one of the players in the game. The goal for the puzzle solver is to find a sequence of moves that lead to a winning position for the player, such as a checkmate or a large advantage in material. The solution to a chess puzzle often contains more than one move, and in this case, the opponent's moves are determined by the puzzle creator. Chess puzzles are a commonly used practice tool to improve chess skills, and they aid in spotting different tactical motifs (pins, skewers, forks, etc.) and improving calculation skills.

Traditionally, before the era of computers being used for playing and practicing chess, puzzles were created by hand and printed in books. Many chess instruction books employ puzzles as a part of the teaching material [1], [2], while others are compilations of a large number of chess puzzles [3], [4]. With the advent of online chess and chess engines, it became possible to create a large number of chess puzzles automatically by mining the positions of online games to find inflection points in games where a certain sequence of moves can result in a winning position for one of the players [5]. There are also online services that let the user solve puzzles using the web browser [6]–[8].

The difficulty of a chess puzzle is commonly determined by presenting the same puzzle to many different people and measuring the success rate. This information is used to calculate the Glicko-2 [9] rating for the puzzle. The drawback is that it's quite time-consuming to determine the puzzle rating accurately using this process.

Fig. 1: An example chess puzzle, from https://lichess.org/XlOLZAcC/black#17. The opponent, playing as black, has just made the move `Qd8xd4`, capturing a pawn (red circle). This leaves the black queen undefended, so the puzzle solver, playing as white, can make the move `Bd3xb5`, capturing the black pawn (green arrow) and checking the black king (green circle). After the opponent captures the bishop (red arrow), the white queen can capture the black queen with the move `Qd1xd4` (blue arrow). The result is white having won a queen while only losing a bishop, giving white a decisive material advantage.

The IEEE Big Data Cup 2024 Challenge on Predicting Chess Puzzle Difficulty [10] considers the problem of predicting the difficulty of chess puzzles using only the initial board state and the solution moves. Having accurate predictions for puzzle ratings could reduce the need to rely on human puzzle solvers to determine the difficulty. The solving experience when introducing new puzzles would also improve since their initial rating would better reflect the difficulty of the puzzle. Our method was applied to this challenge, achieving third place.

II. Related work

Chess engines like Stockfish [11] and Leela Chess Zero [12] are commonly used tools for analyzing chess games and positions. The purpose of the engines is to play chess at the highest level possible, and computer chess competitions such as TCEC [13] exist to determine which engine is the strongest. Chess engines have been stronger than even the best human players since the mid-2000s [14], and the difference in playing strength has since grown significantly with the improvement of chess engines. Engines that play at full strength are thus not very useful opponents for humans due to them being too strong. Instead, chess engines are usually used to analyze chess positions to find moves that were missed by the human players during the game and are also used for opening analysis when preparing for future games.

Modern chess engines may, due to their immense calculation power, make moves that would be very unlikely to be spotted by humans, and thus the choice of moves by chess engines may not accurately reflect which moves humans find difficult. There also exist engines like Maia [15], Maia 2 [16], and Allie [17] that do not aim to play chess at the highest possible level, but instead try to predict which move a human player would make in a certain position. Such engines can be used as a more human-like computer opponent for players compared to traditional chess engines, and they also give insight into which positions are hard for humans to play correctly.

Research has also been conducted on predicting the rating of human chess players using different inputs such as game annotations [18], the moves from a single game [19], or the moves and the times used to make the moves [20]. Chess has additionally been analyzed from the perspective of creativity by predicting which moves humans view as brilliant [21] and by modeling the aesthetics of chess puzzles [22].

III. Dataset

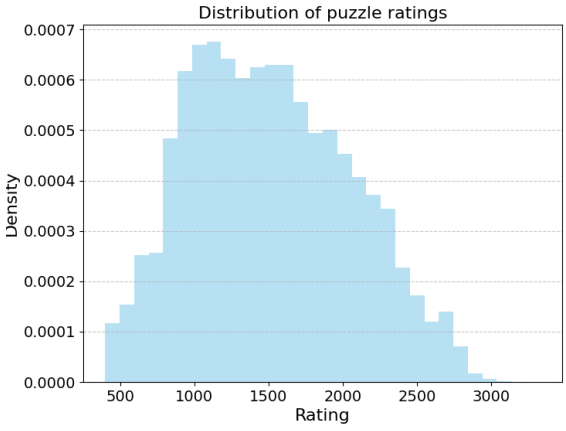

Fig. 2: The distribution of the puzzle ratings in the training set.

The training dataset used in the IEEE Big Data Cup 2024 Challenge on Predicting Chess Puzzle Difficulty [23] consists of 3,888,765 unique chess puzzles extracted from the Lichess open database [24]. An example puzzle is shown in Figure 1. The initial position of the puzzle is given in the FEN notation [25] and the solution moves are provided in the long algebraic notation [26]. The target variable is the rating of the puzzle. The training dataset also contains additional metadata, such as the rating deviation, themes (like tactical motifs and the length of the solution) describing the puzzle, and information about the opening played in the game from which the puzzle was extracted.

Additionally, a public test set of 2,282 puzzles is provided. The public test set only contains the initial position and the solution moves — the rating and the additional metadata are missing. The goal of the challenge is to predict the rating for the puzzles in the public test set.

The rating for each puzzle has been determined by repeated solving of the puzzle in the Lichess trainer [6]. Each puzzle and user that solves puzzles have an initial rating of 1,500. If the user solves the puzzle correctly the puzzle rating decreases and the user rating increases, and vice versa if the solution is incorrect. The changes in ratings are calculated using the Glicko-2 method. The density of the distribution of the ratings in the train set is highest around the initial rating of 1,500, as is seen in Figure 2. Most of the puzzles in the training set have short solutions: only about 2.3% of the puzzles in the training set have solutions containing five or more moves.

IV. Method

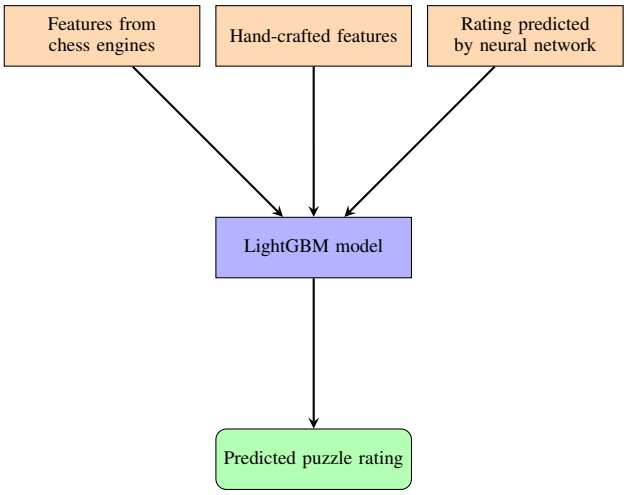

Fig. 3: Flowchart showing the basic structure of the method to predict the Glicko-2 rating for chess puzzles. A variety of features are generated, including features extracted from chess engines and hand-crafted features. Additionally, a neural network model is trained to predict the rating using the initial board state and the solution moves as input. All of these features are provided as input to a LightGBM model that then performs the final rating prediction. The features are described in more detail in Section IV.

Our method for predicting the difficulty of chess puzzles consists of a gradient boosting decision tree model trained us-

ing the *LightGBM* [27] library. The input features of the model can be grouped into three categories, which are described in more detail in the next sections:

A. Features extracted from chess engines
B. Hand-crafted features created from the initial position and the solution moves
C. Puzzle rating predicted by a residual feed-forward neural network trained on the rating dataset

We use *pandas* [28] to manage the training data and *python-chess* [29] to analyze the chess positions.

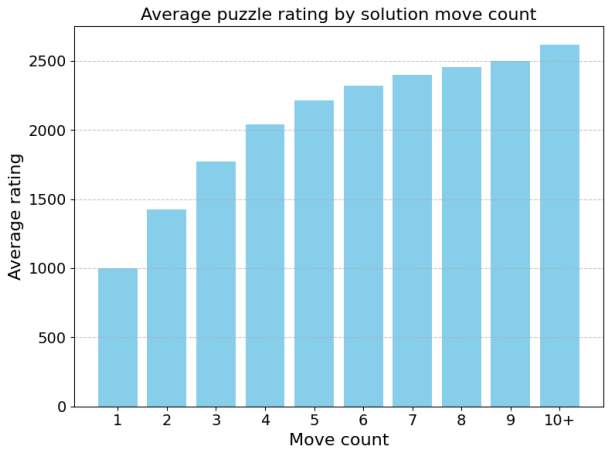

Fig. 4: Average puzzle rating by solution move count. Puzzles with long solutions are, on average, clearly harder than puzzles with short solutions.

A. Features extracted from chess engines

The board position before the first to the fifth player (i.e. puzzle solver) and opponent moves are shown to the Maia 1100, 1500, 1900 [15], BadGyal-8 [30], and Leela-T1-256x10 [31] models. All of these models use the Leela Chess Zero [12] engine code, the difference between them being in the learned network weights. Further moves are omitted since only about 0.8% percent of puzzles have a solution containing more than five moves. The feature extracted is the probability of the solution move being selected by the engine evaluation function without any search being performed. This is achieved by setting the node search depth to 1 using the command `go nodes 1` in the UCI interface [32]. In case a solution has less than five moves, the move probability is set to one. The reasoning here is that since there is no move to be made the correct move is selected automatically.

The purpose of extracting the probabilities for the player moves is that especially the Maia and BadGyal-8 neural networks are trained to mimic human moves, so if a solution move is less likely to be played by a human, the puzzle is more likely to be difficult. While the Leela Chess Zero engine is not trained to mimic human moves, the evaluation function by itself without any search still gives information about which moves are easily findable without any deep calculation, which correlates with the difficulty of the puzzle.

Including the probabilities for the opponent's moves also turns out to be useful, though not as useful as the probabilities for the player's moves. The usefulness of the opponent's moves might stem from the observation that if the response from the opponent is surprising then this may make the puzzle harder to solve.

Additionally, the material, positional, and total evaluations from the Stockfish 16 [11] evaluation function are extracted. These evaluations are obtained using the Stockfish 16 UCI interface by calling the `eval` command. The Stockfish 16 centipawn evaluation before and after the first and second player moves are also used as features. These features give information about the general difficulty of the puzzle from a chess engine standpoint.

B. Hand-crafted features

The rating of a puzzle depends quite strongly on some fairly simple features. For instance, as seen in Figure 4 puzzles with short solutions are on average much easier than puzzles with long solutions. A complex model like a LightGBM model is also less likely to overfit when using simple features like these compared to more complicated features, so including these basic features makes the model more robust.

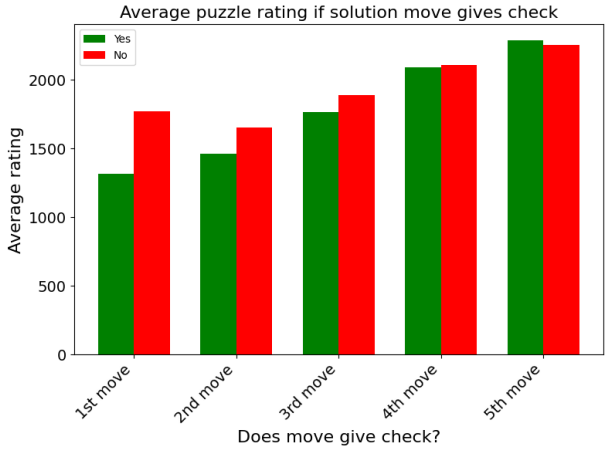

Fig. 5: Average puzzle rating depending on whether a solution move gives check. Puzzles are on average easier if moves give check, especially for early moves.

The following basic features are generated directly from the board state and the solution moves by analyzing the puzzle using the python-chess library:

- The number of moves in the solution (integer), see Figure 4
- Does the solution move give check? (boolean), see Figure 5)
- Is the solution move a capture? (boolean), see Figure 6)
- The type of the piece being moved (categorical)
- Is the player in check? (boolean)
- Is the opponent in check? (boolean)
- The number of legal moves in the position (integer)

- The material count for the player and opponent, and the material difference (integer)
- The piece at each square before the first player move (categorical)
- The start and end rows and columns for each player move (categorical)

Additionally, the following features are generated by analyzing the puzzle in more detail using hard-coded rules and analysis by Stockfish 16:

- Puzzle themes generated by the Lichess puzzle tagger [5] (boolean)
  - Themes include tactical motifs (pin, skewer, fork, etc.), information about the type of endgame (queen, rook, pawn, etc.) for endgame puzzles, and the type of mate (back rank, smothered, etc.) for puzzles ending in checkmate.
  - The generated themes are very similar to the themes provided in the training set since the initial themes for the training set are created by the same process. They are later refined by the puzzle solvers.
- Does the solution move recapture a piece that previously did a capture? (boolean)
- Does the solution move create a mate threat? (boolean)
  - This is determined by checking whether the player could mate the opponent if the opponent passes their turn. The analysis of whether checkmate is possible is done using Stockfish 16.

The move-specific features are generated for each of the first five solution moves, with a default value (false if boolean, N/A if categorical) being used if the solution has less than five moves.

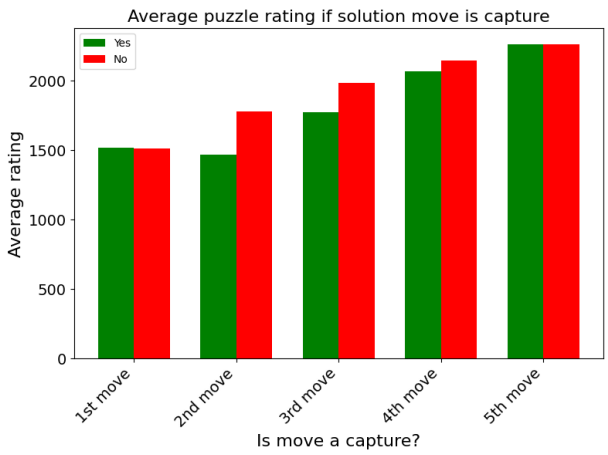

Fig. 6: Average puzzle rating depending on whether a solution move is a capture. Puzzles are on average easier if solution moves are a capture.

C. Rating predicted by neural network

A residual feed-forward neural network is trained to directly predict the rating of the puzzle. A residual network was chosen instead of a regular feed-forward network due to the residual network converging much faster. The network is provided the initial board state before the first player move as a 768-dimensional bitboard representation, where each of the 64 squares is represented by a 12-dimensional vector. The vector representing a square is a one-hot vector showing the piece and its color if there is a piece in that square and a zero vector otherwise.

The first three player moves and the first two opponent moves of the solution are also used as input. Further moves are omitted since only about 8% percent of puzzle solutions have more than three player moves, and including more moves might increase the risk of overfitting to very specific puzzle solutions. When encoding the moves, both the start and end squares of the moves are one-hot encoded. In case a solution has fewer moves than three then the later moves are zero vectors. A single move thus is a 128-dimensional vector, which results in a total input dimension of $768 + 5 \times 128 = 1408$.

Listing 1: PyTorch code for the neural network architecture trained to predict puzzle ratings.

```
class ResidualFFN(nn.Module):
    def __init__(self, input_size, hidden_size,
        num_hidden_layers, activation=nn.SELU,
        normalization=nn.BatchNorm1d):
        super(ResidualFFN, self).__init__(

        )
        self.in_proj = nn.Sequential(normalization(
            input_size), nn.Linear(input_size,
            hidden_size), activation())

        hidden_blocks = []
        for _ in range(num_hidden_layers):
            block = nn.Sequential(normalization(
                hidden_size), nn.Linear(hidden_size,
                 hidden_size), activation())
            hidden_blocks.append(block)
        self.hidden_blocks = nn.ModuleList(
            hidden_blocks)

        self.final = nn.Sequential(normalization(
            hidden_size), nn.Linear(hidden_size, 1))

    def forward(self, x):
        out = self.in_proj(x)
        for block in self.hidden_blocks:
            out = block(out) + out
        out = self.final(out)
        return out

num_moves = 5
model = ResidualFFN(input_size=768+num_moves*128,
    hidden_size=256, num_hidden_layers=2)
optimizer = torch.optim.AdamW(model.parameters(), lr
    =1e-2, weight_decay=1e-3)
loss_fn = nn.MSELoss()
```

Before encoding the puzzle position, we perform the following normalization: In case the player has the black pieces, the board is mirrored so the player has the white pieces. The moves are mirrored as well. This removes the need for the model to learn to predict ratings for both white and black

pieces and effectively doubles the size of the training data set.

The neural network is trained using *PyTorch* [33]. The code for the neural network model is shown in Listing 1. The network has two hidden layers of width 256 and uses batch normalization [34] before each layer and the SELU [35] activation function after each layer except the final layer.

The training process uses the AdamW [36] optimizer with an initial learning rate of 10^{-2} and weight decay of 10^{-3}. The loss function is the mean squared error loss and the batch size is set to 1,000. A decaying learning rate schedule is used where the learning rate is halved if the evaluation score hasn't improved in the last three evaluations, where the evaluation is performed every 500th batch. The training process is stopped when the learning rate is below 10^{-3} and the evaluation score hasn't improved in the last three evaluations.

The training set used to train the downstream LightGBM model is the same as the training set for the neural network. This is done so that both the neural network and the LightGBM model can use a large amount of data for training. To avoid providing as input for the LightGBM model ratings predicted by data the neural network has seen during training, we split the training data into five folds when training the neural network and train the neural network five times, each fold being left out once. The predicted ratings given as input to the LightGBM model are predicted with the model where the fold was not used for training. For validation and test samples, the predictions are ensembled using the mean predicted rating of all five training models. In case out-of-sample ratings are not used, the LightGBM model overfits quite heavily on the neural network rating predictions, which makes the final predicted ratings less accurate on the validation and test sets.

D. Training the LightGBM model

The final rating prediction for the puzzles is obtained by providing all the aforementioned features as input to a LightGBM model, the target being the Glicko-2 rating for the puzzles. Table I lists the values of the hyperparameters used when training the model. Other hyperparameters are set to the default values. The training process uses a validation set containing 30,000 samples (extracted from the original training set) to perform early stopping. The training is stopped if the validation loss doesn't improve in 50 iterations.

TABLE I: LightGBM hyperparameters

Hyperparameter	Value
learning_rate	0.1
num_leaves	63
max_bin	63
num_boost_round	5,000
seed	1
objective	regression
metric	rmse

V. Evaluation

The model is evaluated using the mean squared error loss, comparing the predicted ratings to the ground truth ratings. The choice of loss function means that large errors in predictions for some puzzles can affect the mean loss drastically. This is one of the reasons why hand-crafted features were included as inputs for the LightGBM model. Such simple features are less likely to cause the model to overfit, compared to more complex methods like predicting the rating directly using a deep neural network.

A. Weighting of the validation and test sets

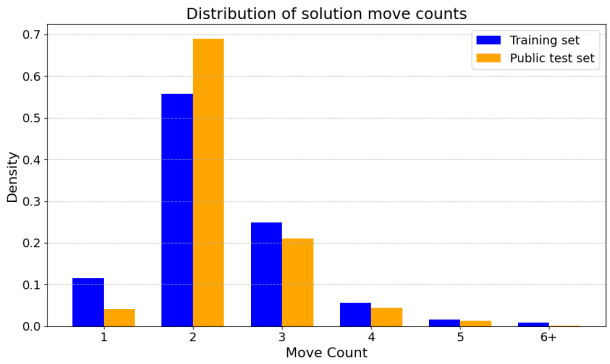

Fig. 7: The distribution of the number of solution moves in the training set and the public test set.

As seen in Figure 7, the distribution of the puzzles in the training set differs quite significantly from the puzzle distribution in the public test set. To counteract this, the validation set of size 30,000 used for early stopping, and the test set of size 70,000 used to evaluate the performance of the LightGBM model, which are extracted from the training set, are weighted to match the distribution of the public test set. This is done by assigning a weight to each sample in the validation and test sets. The weights set so that the distributions of the features listed below approximately match the distribution of the unweighted public test set:

- The number of moves in the solution
- Does the solution end in checkmate?
- Does the first and second user moves give check?
- The probability of the second user move being selected according to the Maia 1100 and Leela-T1-256x10 models.

These features are chosen since they capture the main differences in the distributions of the sets. The assigned weights are then used when calculating the mean squared error loss for the validation and test sets.

VI. Results

Our method achieved the MSE loss of 67,016 on the weighted test set and a preliminary score of 69,891 on the leaderboard [23]. Our final score on the public leaderboard was 123,103, which was the third highest score. The best final score on the leaderboard was 104,541, while the second-best team achieved a score of 120,682.

Our final score on the leaderboard was thus 76% higher than the preliminary score. This indicates that, despite the effort to weigh the validation and test sets to match the public

test set, the model is significantly overfitted to the available training set. The discrepancy between the scores was however not unique to our solution. All other teams in the top 10 on the leaderboard had a discrepancy between the preliminary and final scores of over 85%, and for all but one of them the difference was over 100%.

The reason for the difference between the preliminary and final scores might be that the weighting of the validation and test sets does not fully capture the differences in distributions. Another cause may be that the pool of puzzle solvers that determined the ratings in the public test set is not the same as the pool of users for the training set, which might skew the ratings of the puzzles.

TABLE II: Results when using different subsets of features

Features used	Weighted test set MSE	Relative diff.
All features	67,016	-
Engine + hand-crafted	68,989	+2.9%
Engine + neural net	79,330	+18%
Hand-crafted + neural net	94,769	+41%
Engine only	91,413	+36%
Hand-crafted only	98,411	+47%
Neural net only	122,877	+83%

We performed an ablation study to evaluate the importance of different feature types on the final result. The results are evaluated on the weighted test set, which was originally extracted from the training set, due to the ground-truth ratings for the public test set not being available. For the study, the features are grouped into three separate groups, following the grouping in Section IV:

- Features extracted from chess engines (*Engine* for short)
- Hand-crafted features (*Hand-crafted*)
- Rating predicted by neural network (*Neural net*)

The results of the ablation study are shown in Table II. The best results are achieved when all feature groups are used, but there are clear differences in the importance of the different groups. The features extracted from chess engines are the most useful for the task, followed by the hand-crafted features. The ratings predicted by the neural network contribute the least, and leaving them out increases the mean squared error by only about 2.9%.

VII. Discussion

The neural network's small contribution to the final results might be partially due to its small size (roughly 500,000 learnable parameters). We also experimented with larger residual feed-forward neural networks and simple CNN architectures, but their performance was approximately the same as that of the smaller network.

Large Transformer networks or more complex CNN architectures were not used due to resource limitations, and using such could increase the impact of the neural network predictions. Such an approach may on the other hand risk increasing the difference in the performance between the training set and the public test set. Another option that may improve the neural network's performance is to utilize transfer learning from networks used in chess engines like Stockfish, Leela, or Maia.

The usefulness of the features extracted from chess engines indicates that there could be value in extracting even more features from existing engines. The recently published Maia 2 [16] and Allie [17] chess engines improve the accuracy of human move prediction compared to the Maia model used in our method. It is thus likely that extracting features from these newer engines would improve the accuracy of the puzzle rating predictions since puzzle ratings are strongly correlated with how likely human players are to play the solution moves.

VIII. Conclusion

In this work, we presented a method for predicting the difficulty of a chess puzzle using only the initial board state and the solution moves as input using a LightGBM model with features extracted from chess engines, hand-crafted features, and a rating predicted by a neural network. The combination of the different features resulted in better rating predictions than using just any single group of features.

References

[1] J. Silman, *How to Reassess Your Chess: Chess Mastery Through Imbalances*, 4th ed. Siles Press, 2010, 658 pages, paperback.

[2] M. Dvoretsky, *Dvoretsky's Endgame Manual*, 5th ed. Russell Enterprises, Inc., 2020, 440 pages, paperback.

[3] F. Reinfeld, *1001 Brilliant Ways to Checkmate*, reissue ed. Wilshire Book Co, 1971, 224 pages, paperback.

[4] J. Nunn, *John Nunn's Chess Puzzle Book*, paperback ed. Gambit Publications, 1999, 208 pages.

[5] T. Duplessis, "lichess puzzler," https://github.com/ornicar/lichess-puzzler. [Online]. Available: https://github.com/ornicar/lichess-puzzler

[6] Lichess.org, "Lichess training," https://lichess.org/training, 2024, accessed: 2024-10-12.

[7] Chess.com, "Chess.com puzzles," https://www.chess.com/puzzles, 2024, accessed: 2024-10-12.

[8] ChessTempo, "Chesstempo," https://chesstempo.com/, 2024, accessed: 2024-10-12.

[9] M. E. Glickman, "Example of the glicko-2 system," http://www.glicko.net/glicko/glicko2.pdf, 2022, published: 2022-03-22, accessed: 2024-10-12.

[10] J. Zyśko, M. Świechowski, S. Stawicki, K. Jagieła, A. Janusz, and D. Ślęzak, "Ieee big data cup 2024 report: Predicting chess puzzle difficulty at knowledgepit.ai," in *IEEE International Conference on Big Data, Big Data 2024, Washington DC, USA, December 15-18, 2024*. IEEE, 2024.

[11] The Stockfish developers, T. Romstad, M. Costalba, J. Kiiski, G. Linscott, Y. Nasu, M. Isozaki, and H. Noda, "Stockfish." [Online]. Available: https://stockfishchess.org/

[12] The LCZero Authors, "Leela chess zero." [Online]. Available: https://lczero.org/

[13] Top Chess Engine Championship, "Top chess engine championship (tcec)," https://tcec-chess.com/, 2024, accessed: 2024-10-12.

[14] Wikipedia contributors, "Human–computer chess matches — Wikipedia," https://en.wikipedia.org/wiki/Human%E2%80%93computer_chess_matches, 2024, accessed: 2024-10-12.

[15] R. McIlroy-Young, S. Sen, J. Kleinberg, and A. Anderson, "Aligning superhuman ai with human behavior: Chess as a model system," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ser. KDD '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 1677–1687. [Online]. Available: https://doi.org/10.1145/3394486.3403219

[16] Z. Tang, D. Jiao, R. McIlroy-Young, J. Kleinberg, S. Sen, and A. Anderson, "Maia-2: A unified model for human-ai alignment in chess," 2024. [Online]. Available: https://arxiv.org/abs/2409.20553

[17] Y. Zhang, A. P. Jacob, V. Lai, D. Fried, and D. Ippolito, "Human-aligned chess with a bit of search," 2024. [Online]. Available: https://arxiv.org/abs/2410.03893

[18] C. Scheible and H. Schütze, "Picking the amateur's mind - predicting chess player strength from game annotations," in *Proceedings of Coling 2014*, 2014.

[19] T. Tijhuis, P. Mavromoustakos Blom, and P. Spronck, "Predicting chess player rating based on a single game," in *2023 IEEE Conference on Games (CoG)*. IEEE, 2023.

[20] M. Omori and P. Tadepalli, "Chess rating estimation from moves and clock times using a cnn-lstm," 2024. [Online]. Available: https://arxiv.org/abs/2409.11506

[21] K. Zaidi and M. Guerzhoy, "Predicting user perception of move brilliance in chess," 2024. [Online]. Available: https://arxiv.org/abs/2406.11895

[22] A. Iqbal, "Computing the aesthetics of chess," in *AAAI Workshop on Computational Aesthetics*. Boston, MA: AAAI Press, 2006.

[23] J. Zyśko, K. Jagieła, M. Świechowski, S. Stawicki, A. Janusz, D. Ślezak, and Z. Pakleza, "Ieee bigdata 2024 cup: Predicting chess puzzle difficulty," https://knowledgepit.ml/predicting-chess-puzzle-difficulty/, 2024, accessed: 2024-10-12.

[24] Lichess.org, "Lichess open database: Chess puzzles," https://database.lichess.org/#puzzles, 2024, accessed: 2024-10-12.

[25] Wikipedia contributors, "Forsyth–edwards notation — Wikipedia," https://en.wikipedia.org/wiki/Forsyth%E2%80%93Edwards_Notation, 2024, accessed: 2024-10-12.

[26] ——, "Algebraic Notation (Chess) — Wikipedia," https://en.wikipedia.org/wiki/Algebraic_notation_(chess), 2024, accessed: 2024-10-12.

[27] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: a highly efficient gradient boosting decision tree," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17. Red Hook, NY, USA: Curran Associates Inc., 2017, p. 3149–3157.

[28] W. McKinney, "Data structures for statistical computing in python," in *Proceedings of the 9th Python in Science Conference*, S. van der Walt and J. Millman, Eds., 2010, pp. 51 – 56.

[29] N. Fiekas, "python-chess: a chess library for python," https://github.com/niklasf/python-chess, 2024. [Online]. Available: https://github.com/niklasf/python-chess

[30] dkappe, "Leela chess weights: Bad gyal 8," https://github.com/dkappe/leela-chess-weights/releases/tag/bad-gyal-8, 2019, GitHub release, accessed 2024-10-12. [Online]. Available: https://github.com/dkappe/leela-chess-weights/releases/tag/bad-gyal-8

[31] Leela Chess Zero contributors, "Best networks for lc0," https://lczero.org/dev/wiki/best-nets-for-lc0/, 2024, accessed: 2024-10-12. Specific weights downloaded from: https://storage.lczero.org/files/networks-contrib/t1-256x10-distilled-swa-2432500.pb.gz. [Online]. Available: https://lczero.org/dev/wiki/best-nets-for-lc0/

[32] Wikipedia contributors, "Universal chess interface — Wikipedia," https://en.wikipedia.org/wiki/Universal_Chess_Interface, 2024, accessed: 2024-10-12.

[33] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, *PyTorch: an imperative style, high-performance deep learning library*. Red Hook, NY, USA: Curran Associates Inc., 2019.

[34] S. Ioffe and C. Szegedy, "Batch normalization: accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ser. ICML'15. JMLR.org, 2015, p. 448–456.

[35] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter, "Self-normalizing neural networks," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17. Red Hook, NY, USA: Curran Associates Inc., 2017, p. 972–981.

[36] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. [Online]. Available: https://openreview.net/forum?id=Bkg6RiCqY7