Proceedings of the 20th Conference on Computer Science and Intelligence Systems (FedCSIS) pp. 801–806
DOI: 10.15439/2025F6497
ISSN 2300-5963 ACSIS, Vol. 43

Estimating the Difficulty of Chess Puzzles by Combining Fine-Tuned Maia-2 with Hand-Crafted and Engine Features

Sebastian Björkqvist
0009-0006-9039-8623
IPRally Technologies Oy
Helsinki, Finland
Email: sebastian@iprally.com

***Abstract*—A common way for chess players to practice tactical awareness is to solve chess puzzles, consisting of an initial position and a sequence of moves to achieve a winning position. This practice is more effective when puzzles are matched to the player's skill level. In this work, we present an approach for estimating the difficulty of a chess puzzle using only the initial position and the sequence of correct moves. Our approach uses a fine-tuned modification of the Maia-2 model combined with a set of hand-crafted features and features extracted from chess engines such as Leela Chess Zero and Stockfish. All of these features are then used as input to a gradient boosted decision tree model that predicts the final rating of the puzzle. We applied our approach to the *FedCSIS 2025 Challenge on Predicting Chess Puzzle Difficulty Part 2*, where it achieved first place.**

***Index Terms*—chess, chess puzzle, machine learning, transformer, neural network, gradient boosted decision tree**

I. Introduction

SOLVING chess puzzles, where a player is presented a specific chess position and needs to find a sequence of moves leading to a winning advantage, is an important part of any chess player's training routine. Solving puzzles allows the player to learn important tactical patterns much more quickly compared to just playing chess games, since a puzzle always contains a critical position, and similar patterns can thus be practiced far more frequently than they would appear in actual games.

Chess puzzles are widely available on online platforms [1], [2], [3] and may be automatically created by analyzing online games and finding critical positions where a specific sequence of moves leads to a winning position for one side [4]. While the creation of puzzles can be automated, the process of determining the difficulty remains mostly dependent on human input. The puzzle difficulty is usually estimated by having a number of different people attempt the puzzle and measuring the success rate. This information can then be used to calculate the puzzle rating, using for instance the Elo [5] or Glicko-2 [6] algorithm. The drawback of this process is that determining the puzzle rating accurately is quite time-consuming, requiring the input of multiple human solvers. Automating the process of determining the difficulty would allow newly included puzzles to be immediately shown to players of suitable strength, which would improve the puzzle solving experience.

Fig. 1. An example of a chess puzzle. The opponent (black pieces), just made the move `Qd8xd4`, capturing a pawn (red circle), leaving the black queen undefended. Thus the puzzle solver (white pieces) 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) to escape check, the white queen captures the black queen with the move `Qd1xd4` (blue arrow). In the final position black is without a queen while white has only lost a bishop, leaving white with a decisive material advantage.

The FedCSIS 2025 Challenge on Predicting Chess Puzzle Difficulty Part 2 [7] is centered around automating the task of predicting the difficulty of chess puzzles using only the initial position and the sequence of moves, without the need for repeated human solving. The competition is a follow-up to the IEEE Big Data Cup 2024 Challenge on Predicting Chess Puzzle Difficulty [8]. The second edition of the competition introduces more chess puzzles in the training dataset. Additionally, the training data features also include success probabilities computed using the Maia-2 [9] model.

In this work, we introduce an approach for estimating the

difficulty of chess puzzles which adapts and fine-tunes Maia-2, originally trained to predict the probability of humans making certain moves in a specific position. We also utilize ideas from the approaches of the first [10] and second place [11] teams from the first edition of the competition, as well as extend our own previous method [12] which placed third. Our new approach was applied to the second edition of the challenge, achieving first place.

II. Related work

While the estimation of chess puzzle difficulty had been analyzed on a small scale previously [13], [14], the main body of work related to the problem of predicting the difficulty of chess puzzles is due to the IEEE Big Data Cup 2024 Challenge on Predicting Chess Puzzle Difficulty [8], which we later refer to as the *first edition* of the competition. The top performing teams of the first edition used the following methods:

1) The winning team [10] used multiple Maia-1 [15] and Leela Chess Zero [16] models to encode the chess positions in a puzzle into vectors and then used an RNN to combine these position embeddings into one puzzle embedding, which was used to predict the puzzle rating.
2) The second place team [11] used a CNN architecture similar to Leela Chess Zero to encode each position in the puzzle and trained the model from scratch to predict the rating. The model also predicted the next move in the position and used the loss of that prediction as an additional feature when predicting the rating.
3) The third place team [12] (our submission to the first edition) used hand-crafted features and features extracted from chess engines, as well as a rating predicted by a residual neural network as input to a LightGBM [17] model to predict the puzzle rating.

Other approaches used in the first edition of the competition include a RankNet-based deep neural network model [18], a Transformer-based approach using a modified version of ChessFormer [19] and other CNN-based solutions [20], [21].

III. Chess Puzzle Challenge Dataset

The dataset of chess puzzles available for training provided by the FedCSIS 2025 Challenge on Predicting Chess Puzzle Difficulty Part 2 [7] consists of approximately 4.55 million chess puzzles obtained from the Lichess open database [22]. An example puzzle can be seen in Figure 1. Each puzzle in the training dataset contains an initial position given in FEN notation [23] and the sequence of the correct moves of the puzzle in long algebraic notation [24]. Additionally, the training dataset contains metadata such as rating deviation, attempt count, popularity, tactical themes, and opening info. A new addition to the second edition of the competition is the inclusion of 22 success probability predictions, describing how likely humans of varying playing strength are to correctly solve the puzzle. The target variable is the rating of the puzzle, determined by repeated solving of the puzzle in the Lichess trainer [1]. The distribution of ratings is shown in Figure 2.

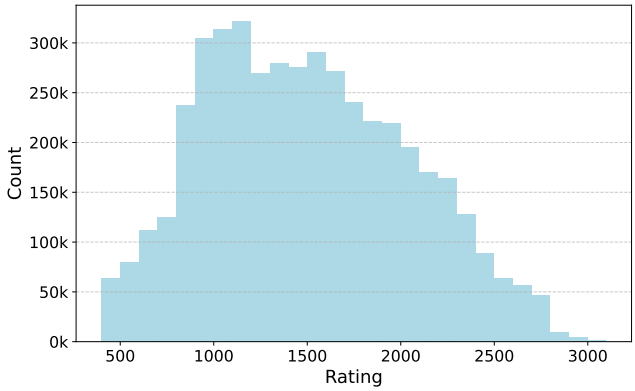

Fig. 2. The distribution of the puzzle ratings in the training set. A large number of puzzles have a rating near the initial puzzle rating of 1500, with another visible concentration of puzzles around 1000.

The competition also provides a public test set containing 2,235 puzzles. The public test set does not contain any of the metadata that is available for the training set — it only has the initial position, the solution moves as well as the predicted human success probabilities. The goal of the competition is to predict accurate ratings for the public test set. The public test set is further split into a holdout set used to compute the preliminary score, and a final test set, used for the final competition ranking.

IV. Approach

Our approach is based mainly on the following observation: The first place solution [10] in the first edition used pre-trained chess engine models to extract position embeddings but did not fine-tune the engine model weights. The second place solution [11], on the other hand, used the model architecture of a chess engine model and trained it end-to-end, but did not initialize the model weights with any pre-trained model. Thus, we wanted to analyze whether these two methods could be combined by both training a model end-to-end and initializing the model with pre-trained weights. The reason why no team did this in the first edition of the competition is likely related to the difficulty of extracting the trained weights from Maia-1 and Leela Chess Zero models for use in a commonly used deep learning library like PyTorch.

Shortly after the first edition of the competition finished, the Maia-2 [9] model was released. This model turned out to be a great candidate to both initialize a model with and to fine-tune, since the model code and weights are freely available, and the model is written in PyTorch, making it easy to modify and extend. Thus, we decided to use the Maia-2 model as the backbone of our approach.

We also wanted to verify whether the use of hand-crafted features and features extracted from other chess engines is still useful even when fine-tuning a deep learning model to predict the ratings, so we decided also to include and extend the features used in our third place solution for the first edition [12].

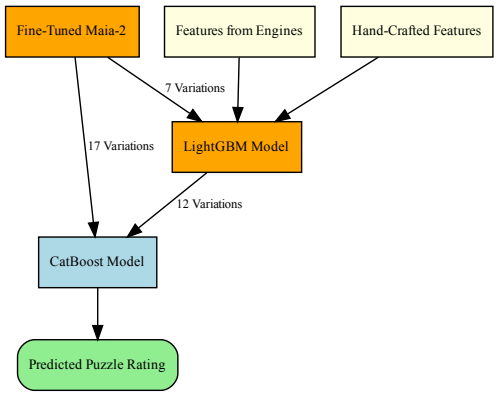

Fig. 3. Flowchart showing the basic structure of the method to predict the Glicko-2 rating for chess puzzles. Ratings are first predicted by a fine-tuned Maia-2 model as well as a LightGBM model. Multiple variations of both models are trained, and the predicted ratings are given as input to a final CatBoost model that ensembles the ratings into a final prediction. The different models and features are described in detail in Section IV.

An overview of our approach is shown in Figure 3. We first train a fine-tuned Maia-2 model to predict the rating of puzzles, and use this rating as well as hand-crafted and chess engine features as input to a LightGBM model to predict the puzzle rating. Finally, we use ratings predicted by multiple variants of the fine-tuned Maia-2 and the LightGBM model as input to a final CatBoost ensemble, which then predicts the final puzzle rating. For the predictions submitted to the competition, we additionally perform a linear scaling to account for the differences in the rating distribution between the training set and the public test set.

A. Fine-Tuned Maia-2

1) Architecture: Maia-2 [9] is a model trained to predict the probability of a human making a specific move in a certain chess position. Maia-2 consists of a ResNet backbone that embeds a position to a set of patch embeddings, and of a Transformer model that takes the patch embeddings as input and outputs a *position embedding* encoding relevant information about the board state. The Transformer layers incorporate skill-aware attention which allows the model to see information about the skill level of the players when predicting the next move. The position embedding is used as input to three different heads: The *move head* predicting the probability of each possible next move, the *auxiliary head* predicting, among others, the legal moves and move-specific information such as whether the move is a check or capture, and the *value head* predicting the result of the game.

An overview of how we use the Maia-2 model is shown in Figure 4. We utilize the model for predicting the puzzle rating as follows: First, for each position in the puzzle (up to and including the fifth move), we compute the position embedding using the ResNet backbone and the Transformer model. We inject a constant skill value to the Transformer model since puzzle solvers are unknown.

Additionally, for each position in the puzzle, we compute the move and auxiliary predictions by passing the position embedding to the heads. Then, we compute the cross-entropy loss of these using the ground truth (which is known because all the puzzle moves are available), following how move predictions were used in [11]. Finally, we embed the correct move in the position using a two-layer MLP. The final embedding of the position is a concatenation of the Maia-2 position embedding and the move and auxiliary prediction losses, as well as the correct move embedding. We also experimented with using the Maia-2 value head here, but it did not improve the rating predictions, so we ultimately left it out.

To obtain a single embedding for an entire puzzle, we pass the final position embedding sequence into a bi-directional GRU model (with two layers and a hidden size of 512) to obtain a single 1024-dimensional embedding for the entire puzzle, similarly as in [10]. The main difference from their solution is that we use the Maia-2 model instead of Maia-1 and Leela, and we also fine-tune the entire model instead of keeping the position embedder frozen.

The final predicted rating is obtained by passing the puzzle embedding given by the GRU to a final linear layer. This layer predicts the rating of the puzzle, normalized to zero mean and unit variance.

As an additional variant, we also include a set of other features as input to the final linear layer. These features are a subset of the ones used as input to the LightGBM model (described in section IV-B). These features are embedded using a residual neural network, and the output embedding is concatenated with the puzzle embedding and given as input to the final linear layer.

2) Training: The Maia-2 model is implemented using *PyTorch* [25], so we also implement our additional layers using PyTorch and use it to train the model. We initialize the Maia-2 model with the pre-trained rapid or blitz weights. The entire model is trained end-to-end, meaning that also the pre-trained Maia-2 layers change during training. We use mean squared error (MSE) loss for the predicted rating, and we additionally also compute the cross-entropy loss for the move and auxiliary predictions from Maia-2 to keep them from diverging, with the cross-entropy losses having significantly lower weight than the rating MSE loss. We also weigh the loss of each puzzle by the inverse of its rating deviation to prevent the model from paying too much attention to puzzles that have a very uncertain rating.

We use the Schedule-Free AdamW optimizer [26] to train our model, which allows us to use a fixed learning rate for the entire training. The model is trained using approximately 4.1 million puzzles. We use a batch size of 128 puzzles and evaluate the model every 5,000 iterations. The training is stopped when the validation loss hasn't improved for three consecutive evaluations.

A single NVIDIA RTX 4070 GPU is used to train the model, and we run the training using bfloat16 automatic mixed

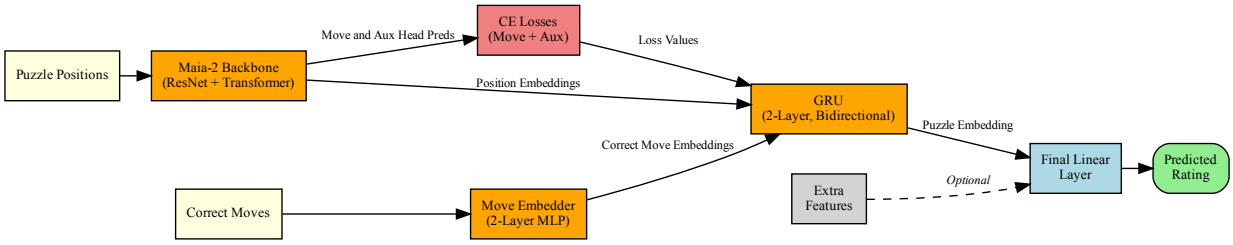

Fig. 4. Diagram describing how the fine-tuned and customized Maia-2 model for predicting puzzle ratings. First, each position in the puzzle is encoded using the Maia-2 backbone. Then, the predictions of the move and auxiliary heads are used to compute a loss using the ground-truth values. Additionally the correct move in the position is embedded using an MLP. The position embeddings, loss values and move embeddings of a puzzle are then combined into a single puzzle embedding using a GRU model. The predicted rating is then computed using a final linear layer, which also may obtain as input additional hand-crafted or engine features. The model is trained end-to-end with all layers being trained or fine-tuned simultaneously. For more details, see Section IV-A.

precision to speed up the training and reduce memory usage. A single training run takes about two hours. Multiple variants of the model are trained with different hyperparameters (dropout, weight decay, learning rate, loss weights) and slightly different model configurations and weights (Maia-2 rapid or blitz).

B. LightGBM model

The next step in our rating prediction pipeline is training a gradient boosted decision tree model using *LightGBM* [17] that receives as inputs three different types of features:

1) Ratings predicted by the fine-tuned Maia-2 model, as described above in section IV-A
2) Features extracted from various chess engines such as Leela, Maia-1 and Stockfish
3) Hand-crafted features created from the initial position and the solution moves

The LightGBM model is trained with approximately 4.4 million puzzles using MSE loss. The training is stopped if the validation loss doesn't improve in 50 iterations. Multiple different versions of the model are trained using slightly different feature sets and hyperparameters (like different dropout and bagging factor). We chose to use LightGBM due to its fast training speed when using a large number of features and puzzles. In the final ensemble we instead used CatBoost [27], which is slower but results in a slightly more accurate model.

1) Ratings predicted by fine-tuned Maia-2: The LightGBM model uses as input the rating predicted by seven different versions of the fine-tuned Maia-2 model. For six of these versions we trained the Maia-2 model multiple times using different folds, and then predicted the rating for the training set with the model where the fold was not used for training, following our previous approach [12]. We also included the rating of one fine-tuned Maia-2 model that was trained only once due to time constraints, meaning the LightGBM model retrieved ratings predicted on the train set of the Maia-2 model. We added Gaussian noise to these ratings to make sure the model didn't overfit to this specific feature.

2) Features extracted from chess engines: The main feature extracted from chess engines is the probability that the correct puzzle move is made in different positions during the puzzle, as in [12]. The probability is computed for the first five player and opponent moves, with a default value of 1.0 being used if the puzzle is shorter than five moves. We compute the probability using the engine evaluation function only, without any search being performed. This feature is obtained using the following models:

- All Maia-1 models (1100, 1200, ..., 1900) [15]
- Leela "best nets" T1-256x10 and T1-512x15x8h [28]
- BadGyal-8, GoodGyal-5, GoodGyal-7 and TinyGyal-8 [29]
- Additional Leela networks LD2, J104.1-30, T73_RL.2-10000 and J64-180 [30]

We also use Stockfish 17 [31] to extract the material, positional, and total evaluations according to its evaluation function. Stockfish is also used to compute the final centipawn value after the last move of the puzzle, as well as to compute whether the correct move is found on a certain depth. For this purpose we used depths 1 through 100,000 in powers of ten.

Additionally we use the success probabilities provided by the competition organizers in the original dataset.

3) Hand-crafted features: We extract three feature families using *python-chess* [32] and *pandas* [33]: puzzle-level basic features, move/position features, and tactical features.

a) Puzzle-level basic features: These features are computed once per puzzle: solution length; is the player white?; Lichess themes from the puzzle tagger [4]; initial piece placement (board array) before the first player move.

b) Move/position features: These features are computed for the first five moves by each side, and are listed in Table I.

c) Tactical features: Features indicating tactical motifs, described in Table II. Unless noted, computed for the first five moves by each side.

C. Final CatBoost ensemble

To obtain the final predicted rating, we train a gradient boosted tree model using *CatBoost* [27] that receives as input

TABLE I
MOVE- AND POSITION-SPECIFIC FEATURES

Category	Features
Position state	# legal moves; in-check flag
Forcing moves	# checking moves / pieces; # capturing moves / pieces; captures winning material; materially safe captures
Correct move	Check flag; moving piece mobility; captures undefended piece; material gain; piece type; from/to column and row
Material and structure	Side material and material diff.; mobility by piece type; undefended pieces (# and value); attackers/defenders near each king
Pawn structure	Pawn islands; doubled / isolated / passed pawns
Other	Over-/under-defended pieces; castling status (both sides); castling/en passant flags

TABLE II
TACTICAL FEATURES

Feature	Notes
Accepted sacrifice	Correct move accepts a sacrifice
Interposition defence	Correct move blocks a check
Mate threat	First five *player* moves only; correct move poses a mate threat, detected using Stockfish assuming opponent skips a turn
Recapture	First five *player* moves only; correct move recaptures previously lost piece

the predictions of 17 different versions of the fine-tuned Maia-2 model, as well as 12 different versions of the LightGBM model. Additionally, the model uses as input the number of moves in the puzzle solution and the probability of making the correct puzzle move according to the following Leela nets: T3-512x15x16h [30], T82-768x15x24h and BT3-768x15x24h [28]. We include these probabilities in this late stage instead of in the larger LightGBM model due to the computation of the probabilities with these models being quite slow.

The CatBoost model is trained using 100,000 puzzles, with MSE loss. We note that using a gradient boosted tree model for ensembling gives slightly better results compared to using just a simple average or learned weighted average.

D. Scaling of predicted ratings

Since the public test set puzzles have on average fewer attempts than the ones in the training set [8], they are also likely to have fewer extremely high and low ratings, due to puzzles with such extreme true ratings requiring many solving attempts to converge. To account for this difference, we follow the method used in [11] and apply linear scaling to align the predicted ratings more closely with the estimated mean $c = 1600$ of the test set distribution using the formula

$$\hat{y}_{scaled} = c + g * (\hat{y} - c),$$

where g is the scaling factor. The mean value was found by weighing our own test set (extracted from the training data) to better match the distribution of the public test set (similarly to how we weighed our own validation set in the first edition [12]). The raw predicted ratings we used for the final submissions in the FedCSIS 2025 competition all came from the same final CatBoost ensemble. We then applied the scaling using factors g of 0.7, 0.8 and 0.9 to obtain three different variations of the ratings.

V. RESULTS

TABLE III
FINAL FEDCSIS 2025 COMPETITION RESULTS (TOP 5 TEAMS OUT OF 31)

Rank	Prelim. MSE	*Final MSE*	Rel. diff.
1	55,382	*52,311*	-
2	58,186	*54,377*	+3.9%
3	58,892	*55,938*	+6.9%
4	61,524	*57,492*	+9.9%
5	67,062	*61,045*	+17%

In the FedCSIS 2025 Challenge, the predicted puzzle ratings are evaluated on the public test set by comparing the predicted puzzle ratings to the ground truth ratings and computing the MSE loss. Our method achieved a preliminary score of 55,382, and a final score of 52,311, resulting in first place in the final results (as well as on the preliminary leaderboard). The scores of the top five teams (out of a total of 31 teams with a final score) are shown in Table III. The final score of the second place team is 3.9% higher than our score, and all of the top four teams are within 10% of our score. This indicates that the second edition of the competition was much closer than the first edition, where the difference between the first and second place teams was around 15% [8].

TABLE IV
RESULTS OF DIFFERENT VARIATIONS OF OUR MODEL

Model	Test set MSE	Rel. diff.
Final CatBoost ensemble	35,400	-
LightGBM (all feats)	36,400	+2.9%
LightGBM w/o hand-crafted feats	36,800	+4.1%
LightGBM w/o engine feats	37,300	+5.4%
Fine-tuned Maia-2 (with extra feats)	38,900	+10%
Fine-tuned Maia-2 (w/o extra feats)	40,800	+15%
LightGBM w/o fine-tuned Maia-2	52,600	+49%

We also performed an ablation study to verify the impact of different parts of our solution, using a test set of 50,000 puzzles extracted randomly from competition training set. The results of the ablation study are shown in Table IV. From the results we can see that the fine-tuned Maia-2 model is critical to our solution, since the LightGBM model without these ratings has an MSE that is 49% higher than our final solution. We also see that the final CatBoost ensemble is useful, since without it the loss is 2.9% higher.

The ablation study also confirms that both the chess engine features and hand-crafted features are still useful. Additionally we see that even though the fine-tuned Maia-2 model is critical for our model, by itself it performs around 15% worse than

our final ensemble, which would place it close to the fifth place finisher, assuming the results transfer to the competition public test set.

VI. Conclusion

In this work, we described an approach for predicting the difficulty of a chess puzzle based on fine-tuning the Maia-2 model and combining the predicted rating from this model with hand-crafted features as well as features extracted from chess engines. We showed that fine-tuning the pre-trained Maia-2 model achieved strong performance for the puzzle rating prediction by itself, and that hand-crafted and chess engine features are still a useful addition to this deep learning-based method. We demonstrated the effectiveness of the method, achieving first place in the FedCSIS 2025 Challenge on Predicting Chess Puzzle Difficulty.

References

[1] Lichess.org, "Lichess training," https://lichess.org/training, 2025, accessed: 2025-08-16.

[2] Chess.com, "Chess.com puzzles," https://www.chess.com/puzzles, 2025, accessed: 2025-08-16.

[3] ChessTempo, "Chesstempo," https://chesstempo.com/, 2025, accessed: 2025-08-16.

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

[5] A. E. Elo, *The Rating of Chessplayers, Past and Present*. New York: Arco Pub., 1978. ISBN 0668047216 9780668047210. [Online]. Available: http://www.amazon.com/Rating-Chess-Players-Past-Present/dp/0668047216

[6] M. E. Glickman, "Example of the glicko-2 system," http://www.glicko.net/glicko/glicko2.pdf, 2022, published: 2022-03-22, accessed: 2025-08-16.

[7] J. Zyśko, M. Ślęzak, D. Ślęzak, and M. Świechowski, "FedCSIS 2025 knowledgepit.ai Competition: Predicting Chess Puzzle Difficulty Part 2 & A Step Toward Uncertainty Contests," in *Proceedings of the 20th Conference on Computer Science and Intelligence Systems*, ser. Annals of Computer Science and Information Systems, M. Bolanowski, M. Ganzha, L. Maciaszek, M. Paprzycki, and D. Ślęzak, Eds., vol. 43. Polish Information Processing Society, 2025. doi: 10.15439/2025F5937. [Online]. Available: http://dx.doi.org/10.15439/2025F5937

[8] 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.

[9] 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," *Advances in Neural Information Processing Systems*, vol. 37, pp. 20 919–20 944, 2024.

[10] T. Woodruff, O. Filatov, and M. Cognetta, "The bread emoji team's submission to the ieee bigdata 2024 cup: Predicting chess puzzle difficulty challenge," in *2024 IEEE International Conference on Big Data (BigData)*. IEEE, 2024, pp. 8415–8422.

[11] A. Schütt, T. Huber, and E. André, "Estimating chess puzzle difficulty without past game records using a human problem-solving inspired neural network architecture," in *2024 IEEE International Conference on Big Data (BigData)*. IEEE, 2024, pp. 8396–8402.

[12] S. Björkqvist, "Estimating the puzzlingness of chess puzzles," in *2024 IEEE International Conference on Big Data (BigData)*. IEEE, 2024, pp. 8370–8376.

[13] S. Stoiljkovikj, I. Bratko, M. Guid, and F. UNI, "A computational model for estimating the difficulty of chess problems," in *Proceedings of the third annual conference on advances in cognitive systems ACS*, 2015, p. 7.

[14] D. Hristova, M. Guid, and I. Bratko, "Assessing the difficulty of chess tactical problems," *International journal on advances in intelligent systems*, vol. 7, no. 3, pp. 728–738, 2014.

[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. doi: 10.1145/3394486.3403219. ISBN 9781450379984 p. 1677–1687. [Online]. Available: https://doi.org/10.1145/3394486.3403219

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

[17] 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. ISBN 9781510860964 p. 3149–3157.

[18] A. Rafaralahy, "Pairwise learning to rank for chess puzzle difficulty prediction," in *2024 IEEE International Conference on Big Data (BigData)*. IEEE, 2024, pp. 8385–8389.

[19] S. Miłosz and P. Kapusta, "Predicting chess puzzle difficulty with transformers," in *2024 IEEE International Conference on Big Data (BigData)*. IEEE, 2024, pp. 8377–8384.

[20] D. Ruta, M. Liu, and L. Cen, "Moves based prediction of chess puzzle difficulty with convolutional neural networks," in *2024 IEEE International Conference on Big Data (BigData)*. IEEE, 2024, pp. 8390–8395.

[21] M. Omori and P. Tadepalli, "Chess rating estimation from moves and clock times using a cnn-lstm," in *International Conference on Computers and Games*. Springer, 2024, pp. 3–13.

[22] Lichess.org, "Lichess open database: Chess puzzles," https://database.lichess.org/#puzzles, 2025, accessed: 2025-08-16.

[23] Wikipedia contributors, "Forsyth–edwards notation — Wikipedia," https://en.wikipedia.org/wiki/Forsyth%E2%80%93Edwards_Notation, 2025, accessed: 2025-08-16.

[24] ——, "Algebraic Notation (Chess) — Wikipedia," https://en.wikipedia.org/wiki/Algebraic_notation_(chess), 2025, accessed: 2025-08-16.

[25] 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.

[26] A. Defazio, X. Yang, A. Khaled, K. Mishchenko, H. Mehta, and A. Cutkosky, "The road less scheduled," *Advances in Neural Information Processing Systems*, vol. 37, pp. 9974–10 007, 2024.

[27] A. V. Dorogush, V. Ershov, and A. Gulin, "Catboost: gradient boosting with categorical features support," *arXiv preprint arXiv:1810.11363*, 2018. [Online]. Available: https://arxiv.org/abs/1810.11363

[28] Leela Chess Zero contributors, "Best networks for lc0," https://lczero.org/dev/wiki/best-nets-for-lc0/, 2024, accessed: 2025-07-25. [Online]. Available: https://lczero.org/dev/wiki/best-nets-for-lc0/

[29] dkappe, "Leela chess weights: Bad gyal," https://github.com/dkappe/leela-chess-weights/wiki/Bad-Gyal, 2020, GitHub release, accessed 2025-07-25. [Online]. Available: https://github.com/dkappe/leela-chess-weights/wiki/Bad-Gyal

[30] Leela Chess Zero contributors, "Contributed networks for lc0," https://storage.lczero.org/files/networks-contrib/, 2024, accessed: 2025-07-26. [Online]. Available: https://storage.lczero.org/files/networks-contrib/

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

[32] N. Fiekas, "python-chess: a chess library for python," https://github.com/niklasf/python-chess, 2025. [Online]. Available: https://github.com/niklasf/python-chess

[33] 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.