The `bread emoji` Team's Submission to the IEEE BigData 2024 Cup: Predicting Chess Puzzle Difficulty Challenge

Tyler Woodruff
Independent Researcher
hhhtylerw@protonmail.com

Oleg Filatov
Google
me@olegwritescode.fyi

Marco Cognetta
Google + Tokyo Institute of Technology
cognetta.marco@gmail.com

***Abstract*—We detail the `bread emoji` team's submission to the IEEE BigData 2024 Predicting Chess Puzzle Difficulty Challenge. Our solution revolved around the use of ensembled, pretrained, neural chessboard embedders (specifically, truncated Maia and Leela models) and an empirically-guided distribution rescaling postprocessing step.**

Our approach was the outright winner of the competition with a >16.4% reduction in mean squared error (MSE) over second place in the preliminary evaluation and a >13.3% reduction in MSE over second place in the final evaluation.[1]

I. Introduction

A chess puzzle is a position that has a sequence of moves which leads to some material gain for one side. Chess puzzles have a variety of themes and difficulties, and the IEEE BigData Predicting Chess Puzzle Difficulty Challenge's goal is to predict the rating (a measurement of difficulty) of a puzzle given its description [1].[2]

In this paper, we give an overview of our winning submission to this challenge. In particular, we discuss the backbone neural chess models that we used as board embedders and the simulation-based rescaling postprocessing function that was used to fit the noisy competition dataset distribution. We additionally discuss several other approaches that we took to this challenge (though all were based on neural chess position embeddings) that did not lead to an improvement.

II. Lichess Dataset

The Lichess puzzle dataset[3] contains (at the time of writing) 4.2 million chess puzzles drawn from real games on Lichess. The puzzles are generated by using Stockfish to search for positions for which there is a large swing in evaluation due to some forced line. Other than the puzzle solution, a number of metadata entries are available for each puzzle, detailed in Table I. Puzzles are given a rating using the Glicko-2 rating system [2].

[1]Our code can be found here: https://github.com/mcognetta/ieee-chess.
[2]https://knowledgepit.ai/predicting-chess-puzzle-difficulty/
[3]https://database.lichess.org/#puzzles
[4]https://lichess.org/training/B4iZi. The full solution is `..♟f5?? ♔h4 ♚h6 ♖c5! ♜xg4+ ♙xg4`.

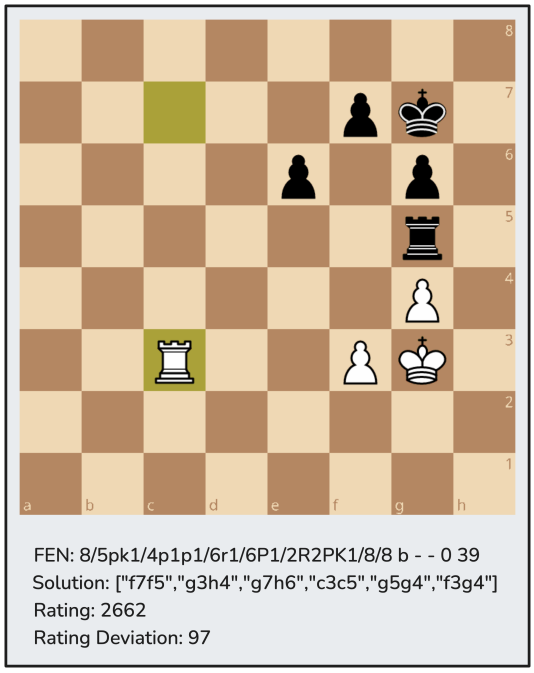


Fig. 1: An example of a puzzle and its relevant metadata (Table I). The initial position and the solution (a sequence of moves resulting in a material gain) is given, and the goal of the challenge is to predict the puzzle's difficulty, measured by a Glicko-2 rating. The goal of this puzzle is to win the black rook after black plays ♟`f5` (a blunder).[4]

A. IEEE Dataset

The competition dataset (we often use "IEEE Dataset" interchangeably here) is a specially curated puzzle dataset of just over 2000 puzzles. The same process that Lichess uses to generate puzzles was used to collect a list of puzzles, which were then loaded onto a custom puzzle server. However, there are several important differences between the IEEE dataset and the Lichess dataset, both in the included data and the manner in which the puzzle server worked.

Most of the metadata from Table I is not included in the IEEE dataset. In fact, only the `PuzzleID`, `FEN`, and `Moves` metadata fields are provided. Crucially, this means that there

Field	Description	Available in IEEE?
PuzzleID	(String) A unique puzzle identifier.	Yes
Fen	(String) The initial position of the puzzle.	Yes
Moves	(List) The accepted solution to the puzzle.	Yes
Rating	(Integer) Glicko-2 rating of the puzzle.	No
RatingDeviation	(Integer) Rating deviation for Glicko-2.	No
Popularity	(Integer) User-driven popularity score.	No
NbPlays	(Integer) How many times the puzzle has been played.	No
Themes	(Categorical) A list of themes present in the puzzle (e.g., mate-in-3).	No
GameUrl	(URL) The game that the puzzle is taken from.	No
OpeningTags	(Categorical) The opening of the game from which the puzzle was derived.	No

Table I: Metadata information for the Lichess and IEEE datasets.

was not even a validation set with ground-truth ratings that could be used for tuning a model.

The competition puzzle server had two major differences with the Lichess puzzle server: the players were not matched against puzzles that were near their skill level but rather were just matched with puzzles at random, and the puzzle ratings were hidden from the user, so they had no knowledge *a priori* of how difficult a puzzle might be.

III. Neural Chess Models

In contrast to classical chess computers which use handcrafted evaluation features and high performance search functions to play chess at a high level, neural chess models learn to play chess solely from game data. A variety of neural models exist, and an overview of neural chess techniques is given in [3]. Stockfish, one of the strongest engines in history, originally used a fully handcrafted evaluation function but switched to an efficient neural evaluator based on the NNUE architecture for Shogi [4], [5]. AlphaZero is another strong engine, but it is a fully neural engine that was trained purely on self-play without any human intervention [6]. Leela is an open-source re-implementation and continuation of this engine [7]. Maia is another open-source, fully neural engine, but is trained to mimic human players rather than for raw strength [8]. However, all of them share a similarity in that they use a neural evaluation function in conjunction with tree search. More recently, there has been progress on strong neural engines which require only a single evaluation without search [9]. Another work treated chess games as a language modeling task and trained a generic GPT-style transformer for it [10], and a recent work used a basic transformer with a chess-specific positional encoding for modeling [11]. We utilized only Leela and Maia for our submissions.

A. Maia

Maia [8] is family of neural chess engines intended to play as human as possible given a specific rating. All models in this family share the same architecture, but each is trained on different data depending on the desired strength of the model. Open-source weights for ratings 1100 to 1900, in increments of 100, are available.

Maia's architecture is a set of 6 convolutional blocks that feed into independent policy and value heads (a common architectural setup for reinforcement learning).

B. Leela

Leela [7] is another family of neural chess engines, based on AlphaZero [6]. There are three different architectures, `small`, `medium`, and `large`. Each model shares roughly the same structure: the three models have 9, 14, and 14 convolutional blocks, respectively, with increasingly large internal hidden dimensions. The convolutional blocks feed into three, independent heads (value, policy, and moves-left-ahead).

IV. Our Approach

The core of our approach was using Leela and Maia models as board embedders for a regression model. Here, we detail each part of the architecture, as shown in Figure 2.

A. Maia/Leela Embeddings

We utilized Leela and Maia models by removing the policy heads and using the flattened output of the last hidden layer as the output. For Maia models, the flattened output has dimension 4096, while for Leela, it has dimension 16384, 32768, 49152 for Leela-`small`, `medium`, and `large`, respectively. These representations are then projected to a 1024-dimensional space with a dense layer.

B. Puzzle Embedder

Using Maia and Leela, we extracted representations for each board in a puzzle but require a single representation of the puzzle for the downstream model. We used a simple RNN that combines the embeddings of each board-state into a single vector representation. Dropout is applied to the final output of the RNN with $p = 0.2$. We capped the number of moves processed by an RNN for any individual puzzle at 10 and pad the length of each puzzle in a batch. After this layer, we had a single 1024-dimensional-vector representation for each puzzle.

To account for outlier puzzle lengths, we also included a categorical puzzle-length embedding. For each puzzle length (up to a maximum of 16), we constructed a learnable, 128-dimensional embedding that is concatenated to the output of the RNN.

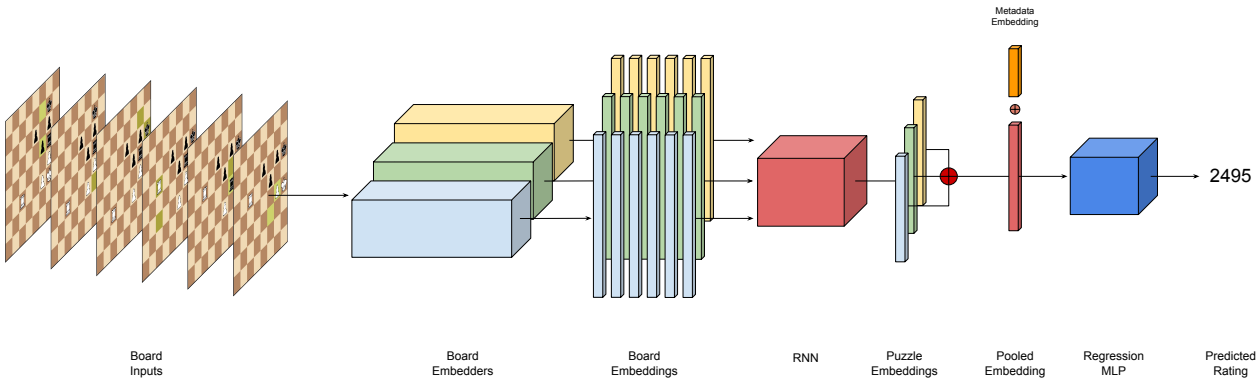


Fig. 2: The general layout of our model architecture. Note that the board embedders do not have to be the same model (we can mix Maia and Leela models, and models within each family).

C. Regression MLP

We used a simple MLP to predict the rating of a puzzle from a board representation. The puzzle representation is passed to a 4 layer MLP[5] with `relu` activations. Dropout is applied to a subset of the intermediate layers with $p = 0.15$.

The final layer has a scalar output representing the predicted rating of the puzzle.

D. Embedding Ensembling

We experimented with ensembled embeddings, where multiple Maia and/or Leela models were used to produce embeddings for a board. The embedding ensembling took place immediately after the RNN layer. The puzzle embeddings from each board embedder were averaged and then passed to the regression MLP to proceed as normal. Intuitively, different embedders (and even different instantiations of the same embedder, as they have different random initializations for the 1024-dimensional projection) will learn different representations of the board which may be complementary. In particular, as Leela and Maia have fundamentally different training criteria (i.e., raw skill vs human imitation), they should have complementary representations of a board.

E. Model Ensembling

We performed simple model ensembling by training multiple models, running them in parallel on a given input, and averaging their results. This greatly improved model robustness and performance.

F. Dataset Modifications

For our full model training runs, we discarded the IEEE training dataset in favor of a newer snapshot of the Lichess dataset (retrieved in early August, 2024), as the Lichess dataset is periodically updated with new puzzles. Additionally, puzzle ratings are updated and, to the best of our knowledge, puzzles are not removed. Thus, the August Lichess dataset provided a strictly larger number of converged puzzles to train on.

To further clean the data, we removed any puzzle that had rating deviation > 90. This resulted in a final dataset of 3.2M puzzles, from which we extracted a 10k validation and 10k test set and used the rest for training.

G. Implementation and Training

We extracted Leela and Maia models to ONNX[6] representations before importing them to PyTorch [12]. We used an ADAM optimizer with an inverse square root scheduler and batches of 128 puzzles. We used Mean Squared Error (MSE) as our loss function, to match the competition's evaluation metric. The model was validated every 2000 batches, and we stopped training when the model failed to improve after 8 consecutive validation steps. The best performing checkpoint was used as our final model.

V. Results

Due to the differences between the Lichess and the IEEE datasets and the lack of IEEE validation data, we considered the competition to have two stages. First, we tried to perform as well as possible on the Lichess dataset, and then we searched for ways to fit that to the IEEE distribution.

A. Lichess Dataset Results

Using the cleaned, up-to-date Lichess dataset (Section IV-F), we experimented on various model types and ensembles. A representative subset is listed in Table II.

Notably, Maia models vastly outperformed Leela models, and Leela models had worse performance as their size (and therefore rating) increased. Ensembling board embeddings improved model performance, but in an inconsistent way (for example, a combined Maia-1300 and Leela-`small` model was worse than either by themselves, but Maia-1300 and Leela-`med` was significantly better than either).

Ensembling outputs also greatly improved performance. Simply averaging the outputs of a set of Maia-1300, 1500, and 1700 models improved the MSE by $\sim$8% compared to any of them individually.

[5]We experimented with various configurations but settled on this for our final model, though potentially other variations appeared within our ensembles.

[6]https://github.com/onnx/onnx

Model Type	Model Configuration	MSE
Single Model	(1) Maia-1300	$56.5k$
	(2) Maia-1500	$57.5k$
	(3) Maia-1700	$57.9k$
	(4) Leela-`small`	$65.5k$
	(5) Leela-`med`	$66.1k$
	(6) Leela-`large`	$71.6k$
Ensembled Embeddings	(7) [Maia-1300 + Maia-1500 + Maia-1700]	$56.0k$
	(8) [Maia-1300 + Leela-`small`]	$65.4k$
	(9) [Maia-1300 + Leela-`med`]	$50.8k$
Ensembled Outputs	(1) + (2) + (3)	$52.4k$
	(1) + (2) + (3) + (4) + (5) + (6)	$50.3k$
	(1) + (2) + (3) + (7)	$51.6k$
	(7) + (8) + (9)	$\mathbf{46.7k}$
	(1) + (2) + (3) + (4) + (5) + (6) + (7) + (8) + (9)	$48.6k$

Table II: A representative subset of modeling results on the Lichess dataset. Models wrapped in brackets (e.g., [Model-A + Model-B]) are embedding-ensembled models, while + denotes output ensembling.

Model Configuration	MSE
Maia-1300$^{(x3)}$	$77.9k$
Leela-`small`$^{(x3)}$	$64.8k$
Maia-1300$^{(x3)}$ + Maia-1500$^{(x3)}$ + Maia-1700$^{(x3)}$ + Leela-`small`$^{(x3)}$ + Leela-`med` + Leela-`large`	$66.6k$
[Multiple Maia + Leela] (rescaled with $L = H = 400$)	$52.2k$
[Multiple Maia + Leela] (rescaled with $L = 250$, $H = 200$)	$\mathbf{49.1k}$

Table III: A representative subset of modeling results on the preliminary IEEE Dataset.

Ensembling a set of ensembled-embedding models led to our best model with an MSE of $46.7k$.

B. IEEE Dataset

We found that the IEEE competition dataset was substantially different from the Lichess dataset in ways that made direct improvements on the Lichess dataset meaningless for the competition. In particular, the ground truth ratings were very noisy, due to the low number of plays-per-puzzle and resulting high rating deviation of the curated IEEE dataset. This means that making more accurate rating predictors does not necessarily translate to better performance on the competition dataset.

Since there was no ground truth competition data, it was not possible to train models to directly match the competition dataset distribution. This motivated us to search for systematic differences between the Lichess and IEEE datasets that could be exploited in order to convert the ratings from a ground-truth predictor to one matching the competition dataset.

We experimented with two approaches. The first attempted to model the competition dataset directly by including rating deviation as an input feature. Then, training on noisy data (i.e., the uncleaned Lichess dataset, which contains puzzles with high rating deviation) should allow us to properly model the competition dataset, if we are able to approximate the ground truth rating deviations. This approach failed, and is detailed in Section VI-B.

Our second, more fruitful approach, was distribution rescaling, where we predicted ground-truth ratings but then rescaled them so that the distribution matched what we expected the IEEE dataset distribution to be.

C. Simulation-Based Rescaling

Since the details of the IEEE dataset curation server were available, we attempted to simulate how puzzle ratings would evolve under the dataset-collection framework. Our simulation framework revolved around using puzzles for which we had the ground-truth rating information (i.e., converged puzzles from the Lichess dataset) and *player* information which we extracted from the Lichess player database.[7] Puzzles were also assigned a "mock" rating, initialized to rating = 1500 and rating deviation 500 (the default initialization used by the Lichess and IEEE puzzle servers).

By using the ground-truth rating of a puzzle, we have a way to estimate how hard the puzzle truly is relative to the player's strength via the expected score from the Glicko-2 rating-change calculation [2], and by using a mock rating, we can simulate how a puzzle of a given difficulty's rating changes when matched at random with different players. This simulation is a close approximation to the IEEE puzzle server.[8]

The average number of plays for each puzzle on the competition puzzle server was given to be between 25 and 50. Figure 4 gives the rating-mismatch distribution for a set of 50k puzzles for a number of plays, $N \in \{10, 25, 50, 100, 500, 1000\}$. The results, especially for the low-number-of-plays simulations, show that as the puzzle's true rating gets farther from the mean rating, the deviation tends to grow. In particular, the simulated ratings tend to be closer to the mean (over-estimating the rating of low-rated puzzles and under-estimating the rating of high-rated puzzles). Intuitively this makes sense as, even if a puzzle with true rating 2700 was matched against 15 players with each of them failing to solve it, it may still not be enough to gain 1200 rating points simply due to how ratings are updated. This is especially true when the puzzles are not matched against players of the appropriate skill, as in the competition framework.

These findings suggest a rating-rescaling that scales ratings more the farther they are from the mean. We used:

$$R(r) = \begin{cases} \max\left(1510, r - H \times \max(1, \left(\frac{r-1510}{D}\right)^4\right) & \text{if } r \geq 1510 \\ \min\left(1510, r + L \times \min(1, \left(\frac{1510-r}{D}\right)^4\right) & \text{if } r < 1510, \end{cases}$$

where H and L are tunable parameters that allow us to limit the maximum rating correction done by the rescaler and D is the distance from the mean ($\mu \approx 1510$ in the Lichess dataset) at which we should max out the rescaling.[9] Our best model used $H = 200$, $L = 250$, and $D = 1000$, which roughly aligns with the values from Figure 4, with low play counts.

[7] Players, like puzzles, are assigned a Glicko-2 rating and rating deviation for their puzzle profile on Lichess.

[8] On the competition puzzle server, players were not able to see the (current) rating of the puzzle and players were matched with puzzles at random rather than being matched with puzzles that were close to their puzzle rating.

[9] We arrived at this specific formulation by manual tinkering. Even a constant rescaler (where ratings above a certain distance from the mean were scaled by an additive constant) performed well.

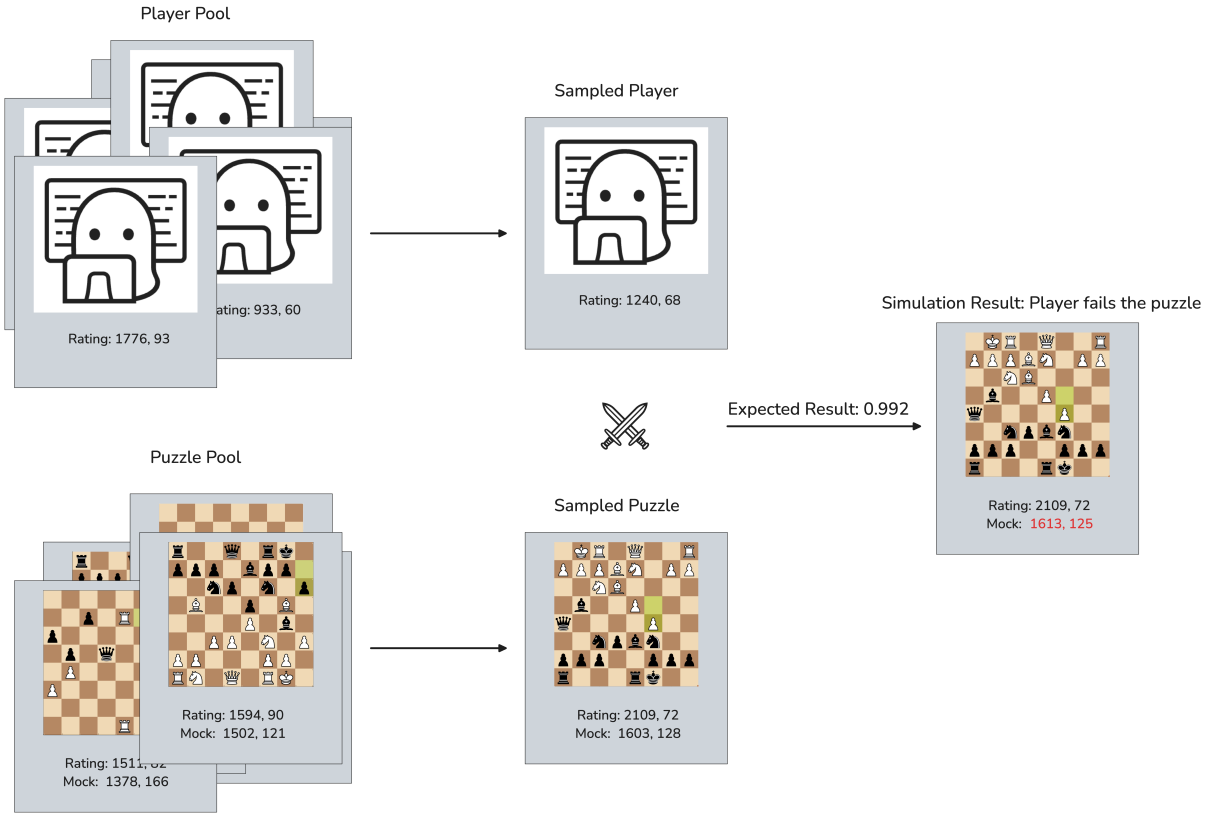


Fig. 3: An example iteration of our competition simulation framework. Players and puzzles are randomly matched and the result of the matching is simulated via a coin flip based on the expected result of the match-up according to Glicko-2 and the *true* rating of the puzzle and the player rating information. The result is used to update the *mock* rating of the puzzle.

Note that we are not trying to perfectly rescale the data, but rather to mitigate the effects of the low number of plays on the convergence of the ratings of puzzles with very high or very low true ratings.

D. IEEE Dataset Results

A sample of our modeling results on the *preliminary* IEEE dataset is given in Table III. The preliminary results were calculated on a subset of the total test data while the final results used the entire test set. However, information about model-specific results on the final test set was not released, so we only consider the preliminary test set here.

A major issue is that improvements on the Lichess dataset did not necessarily lead to improvements on the IEEE dataset. For example, Maia models always outperformed Leela models on the Lichess dataset, but Leela did far better on the IEEE dataset. Additionally, a large ensemble model, which was shown to generally improve performance on the Lichess dataset fails to do so on the IEEE dataset.

Following our experimental findings in Section V-C, using a rescaling postprocessing step to correct for the differences in the Lichess and IEEE dataset distributions lead to far better modeling performance. In particular, the rescaling model with $L = 250$ and $H = 200$, which closely aligns with the distribution of differences in Figure 4 for low play-counts performed the best, at $49.1k$ MSE. This score corresponds to a $\sim 16.4\%$ improvement in MSE over the second-place entry in the competition.

VI. What Didn't Work

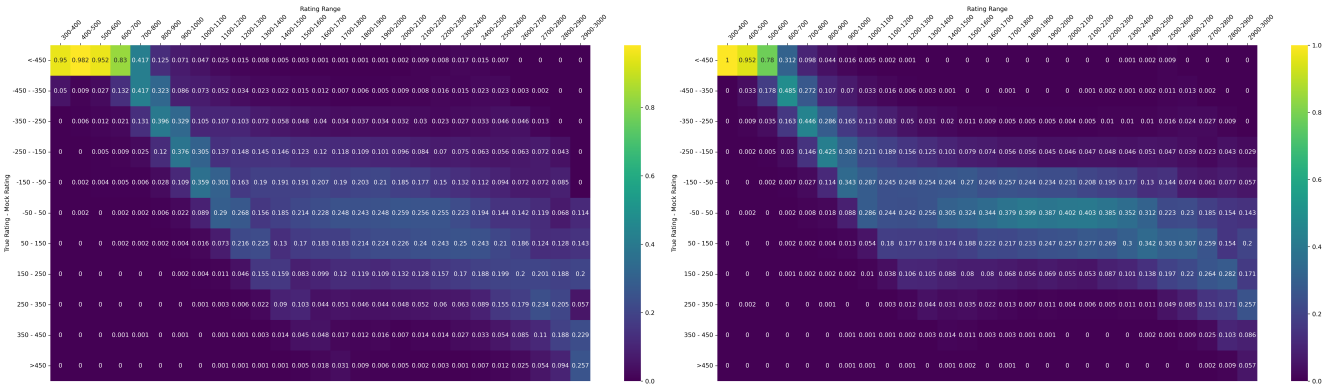
Here, we cover a subset of the alternative approaches and extensions to our main approach that we considered. None of these meaningfully improved the model performance, and some greatly harmed performance, despite being seemingly reasonable ideas.

A. Data Augmentation and Regularization

Despite the provided dataset being fairly large, we experimented with several forms of data augmentation to increase the dataset size. The most straightforward were rating regularization and board flipping. For rating regularization, we randomly perturbed the puzzle's true rating by ± 20 points to help avoid over-fitting. However, we found that this did not substantially impact model quality or robustness.

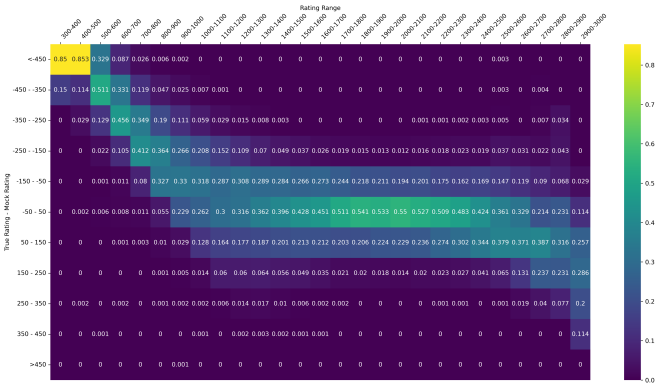
For board flipping, we noted that the mirror of a puzzle is a puzzle of (presumably) identical difficulty. Thus, given a puzzle, we constructed a new puzzle by swapping the active player and mirroring the board.[10] While the positional evaluations

[10]https://www.chessprogramming.org/Color_Flipping

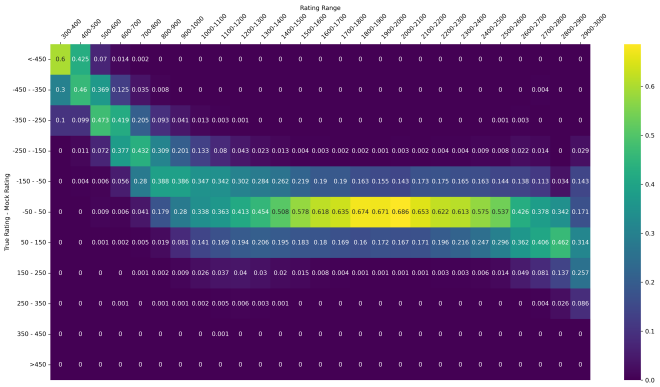


(a) Number of Plays $= 10$

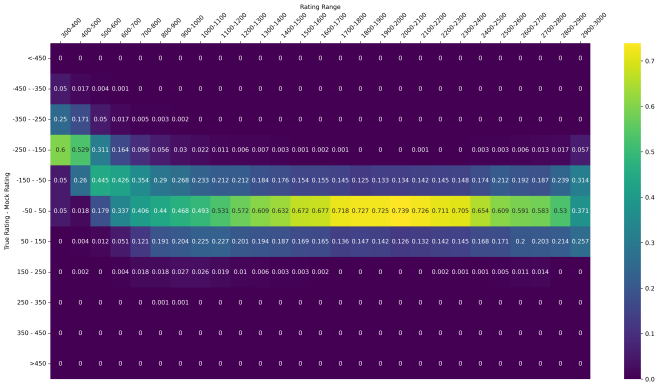
(b) Number of Plays $= 25$



(c) Number of Plays $= 50$



(d) Number of Plays $= 100$



(e) Number of Plays $= 500$

(f) Number of Plays $= 1000$

Fig. 4: The distribution of differences between the true and simulated ratings across different numbers of plays-per-puzzle. A column represents a rating range of the true ratings of puzzles and a row represents a range of differences between the true puzzle rating and the simulated puzzle rating after a given number of plays. A cell in a column shows what percentage of the puzzles in that rating range had a specific Δrating. As the plays-per-puzzle increases, the simulated ratings converge to the true ratings, but for low play counts, the simulated ratings tend to be closer to the mean for very high and very low true ratings—for very low rated puzzles, the simulated ratings are higher than the true ratings, and vice-versa for very high rated puzzles.

for Maia and Leela were extremely close under mirroring, we hypothesized that the underlying latent representations of a position and its mirror would be sufficiently different that training on both would improve modeling performance. Additionally, predicting the rating of the original puzzle and its mirror at inference time would act as an additional layer of ensembling.

However, we found that this technique did not improve modeling quality, and the rating predictions from a puzzle and its mirror were very similar. This suggests that the underlying embedding models were robust to board symmetries, which is supported by the Leela and Maia board representations (both models use the same representation format). Leela and Maia's board representation is color-agnostic and uses a single bit to represent which color is the active player.

We also considered artificially generating new puzzles by taking an existing puzzle from the dataset and slightly perturbing it in a way that preserved the sharpness and general structure of the problem (i.e., the acceptable set of solutions was the same). However, since other forms of data-augmentation and regularization were not effective, and it is not possible to perfectly validate that a puzzle under such a perturbation would have the exact same difficulty, we did not implement this augmentation.

B. Using Rating Deviation as Features

While most of the Lichess metadata is missing from the IEEE dataset (Table I), the organizers provided some information about the IEEE test set's `RatingDev` and `NbPlays` features. In particular, they stated that the IEEE set had average rating deviation of $\sim$130 (compared to $\sim$90 for the Lichess dataset) and between 25-50 plays.

In light of this, we experimented with including rating deviation as a feature during training, then estimating the rating deviation of the puzzles in the IEEE dataset and using that as a constructed feature.

To validate this idea, we simulated the high-rating-deviation setting by using the *uncleaned* Lichess dataset. In particular, we used the puzzles from the Lichess dataset that we removed due to having too large of a rating deviation as an additional test set with ground-truth, large rating deviations (in conjunction with the original, clean test set).

Using this additional feature, we were able to achieve slightly better MSE on the cleaned Lichess validation and test sets as well as better performance on the high-rating-deviation test set compared to models which did not use rating deviation as a feature. However, we also found that the model was extremely sensitive to the value of rating deviation. For example, hard coding the rating deviation feature to the mean rating deviation of the noisy dataset resulted in a substantial increase in MSE (for example, an increase of 50k MSE to more than 100k). Since we had only vague estimates of the rating deviation of the IEEE dataset puzzles and the gain in MSE was marginal when using ground-truth rating deviation features, we abandoned this approach.

C. Classification vs Regression

Our submission models were all simple regression models — we predicted the rating of the puzzle directly from the puzzle representation. However, the MSE of the top teams' submissions during the competition was very high, with most teams being above 60k MSE on the competition dataset. Additionally, even on our clean dataset, our best performing regression models were only in the mid-40k MSE range.

To beat a 40k MSE model, one needs only to get within ± 200 of the true rating, which seems quite easy. Thus, we experimented with chunking the rating ranges into buckets of width at most 200 (since the distribution is bell-shaped, we used smaller buckets around the mean in order to fit the distribution more closely) and transformed the problem into a classification problem with cross-entropy loss as our loss function. Since the values are now categorical, and the semantics of adjacent buckets being more similar than non-adjacent buckets is not inherently captured by the loss function, we added an additional inductive bias by using a smoothed cross-entropy loss where probability mass 0.1 was allocated to the buckets immediately to the left and right of the true bucket.[11] For inference, we used the mean rating of the highest probability bucket as the predicted rating for the puzzle.

However, we were not able to get top-1 accuracy high enough to get competitive scores. Our best models barely broke 100k MSE on the validation set, so we abandoned this approach.

D. Bucketed Regression

Since pure categorical modeling failed, we also experimented with a mixed categorical-regression modeling framework. In this setting, we used a linear combination of the bucket means weighted by their predicted probability. This considerably improved performance over the pure-categorical baseline, reaching 80k MSE, but again was not competitive with pure regression.

We experimented with several other variations of this. For example, we used a linear combination of the top-3 buckets' means if they were adjacent[12] and the top-1 prediction otherwise. These also failed to come close to the regression baselines.

VII. Conclusion

Our approach leveraged large, pretrained neural chess models and transfer learning to predict chess puzzle difficulty. Among the many different approaches that we tried, ensembled-embedding models which were then ensembled together for regression ended up performing, by far, the best. However, the results did not transfer to the competition dataset until we employed an empirically-grounded rescaling postprocessing step. This resulted in the winning submission to the competition.

Interestingly, the Maia models, which are both smaller and weaker than Leela models vastly outperformed the Leela models

[11] That is, the true bucket was assigned weight 0.9 and the buckets to the left and right were each assigned weight 0.05. For boundary buckets, we assigned 0.95 to the true bucket and 0.05 to the sole neighboring bucket.

[12] Top-2 for boundary buckets.

on this task. This is perhaps because of their differing training goals. Whereas Leela is designed to be as strong as possible, Maia is designed to mimic human play, which may allow it to better understand what styles of puzzles are challenging for humans.

For future work, it would be interesting to compare other neural models, especially NNUE [4], the newer Maia-2 family [13], the Chessformer model [11], and the search-less neural models described in [9].

ACKNOWLEDGMENT

We thank the organizers of the competition for their hard work and the Lichess staff for providing the open puzzle dataset.

REFERENCES

[1] 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.

[2] M. E. Glickman, "Example of the glicko-2 system," http://www.glicko.net/glicko/glicko2.pdf, 2022.

[3] D. Klein, "Neural networks for chess," 2022. [Online]. Available: https://arxiv.org/abs/2209.01506

[4] T. S. Contributors, "Stockfish chess engine," https://github.com/official-stockfish/Stockfish.

[5] Y. Nasu, "NNUE: Efficiently updatable neural-network-based evaluation functions for computer shogi," 2018.

[6] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," 2017. [Online]. Available: https://arxiv.org/abs/1712.01815

[7] T. L. Authors, "Leela chess zero," https://lczero.org/.

[8] R. McIlroy-Young, S. Sen, J. M. Kleinberg, and A. Anderson, "Aligning superhuman AI with human behavior: Chess as a model system," in *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020*, R. Gupta, Y. Liu, J. Tang, and B. A. Prakash, Eds. ACM, 2020, pp. 1677–1687. [Online]. Available: https://doi.org/10.1145/3394486.3403219

[9] A. Ruoss, G. Delétang, S. Medapati, J. Grau-Moya, L. K. Wenliang, E. Catt, J. Reid, and T. Genewein, "Grandmaster-level chess without search," 2024. [Online]. Available: https://arxiv.org/abs/2402.04494

[10] D. Noever, M. Ciolino, and J. Kalin, "The chess transformer: Mastering play using generative language models," *CoRR*, vol. abs/2008.04057, 2020. [Online]. Available: https://arxiv.org/abs/2008.04057

[11] D. Monroe and P. A. Chalmers, "Mastering chess with a transformer model," 2024. [Online]. Available: https://arxiv.org/abs/2409.12272

[12] 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. Z. 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," in *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, Eds., 2019, pp. 8024–8035. [Online]. Available: https://proceedings.neurips.cc/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html

[13] 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