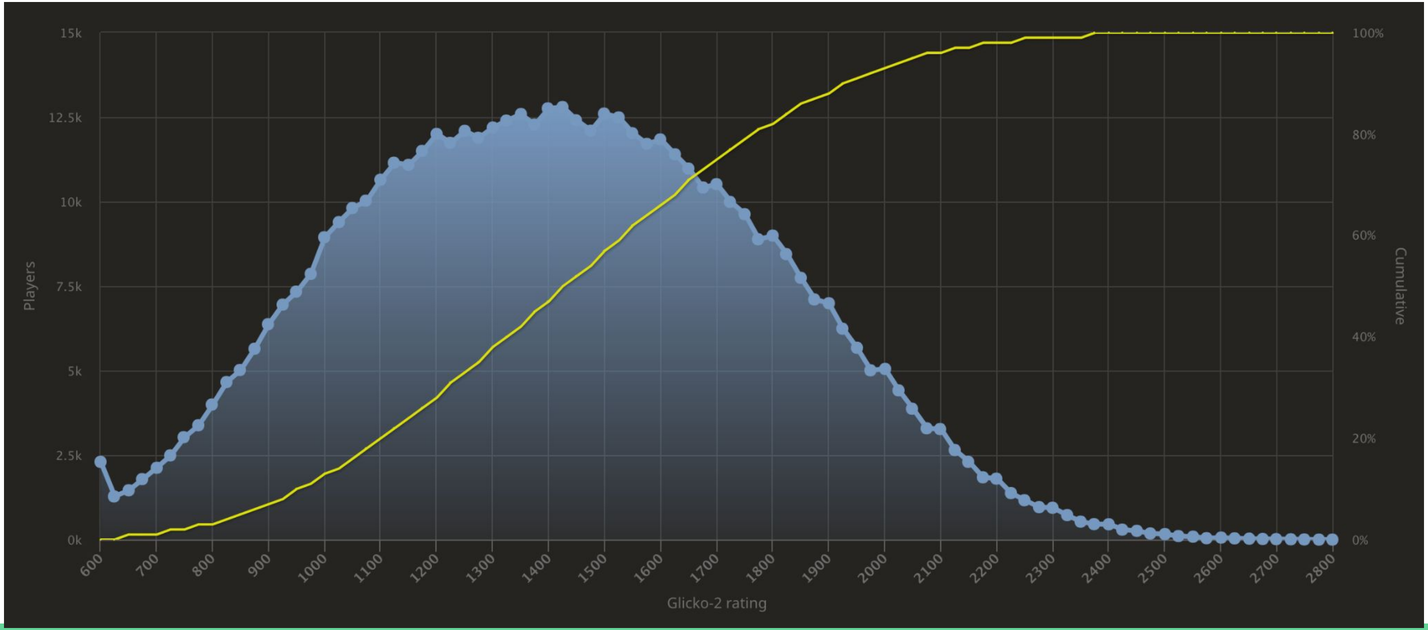
Better Chess Matchmaking with Artificial Intelligence

Simon Lazarus

Background

Background

- Lichess.org: Large chess platform with over 100M games played in Jan 2023
  - Maintains open database of over 4B games played
- Matchups are made based on players' *ratings*
  - Ratings based on who beats whom
  - Similar-rated players are generally matched up



Weekly "Rapid" ratings distribution, Feb 28 2023

Data Science Problem: Better Matchmaking for New Players

- Ratings are fairly unreliable measures of skill when players are *new to Lichess*
- Problem: Come up with a better way to find good matches for *new players*.
- Equivalently: **Find a model that makes "better" predictions of new players' early game outcomes** than Lichess.org's model does.
- Lichess uses "Glicko-2 Boost" model (details later)

“Better Predictions”: Metrics for Evaluation

- Good matchmaking requires good ***probabilistic predictions***.
  - Search for matches where the *probability* that P1 beats P2 is ≒ 50%

- Probabilistic predictions evaluated using ***Binary Cross Entropy*** *(BCE)*
  - Penalizes farther-off predictions at increasing rates
  - Equivalent to main metric used in Deloitte/FIDE Chess Rating Challenge Kaggle competition

- Also will measure:
  - **Accuracy** (among non-draw games)
  - **Mean Absolute Error** (MAE) of prob. predictions from the truth (win=1, lose=0, draw=.5)
    - MAE does not penalize farther-off predictions at increasing rates

Data

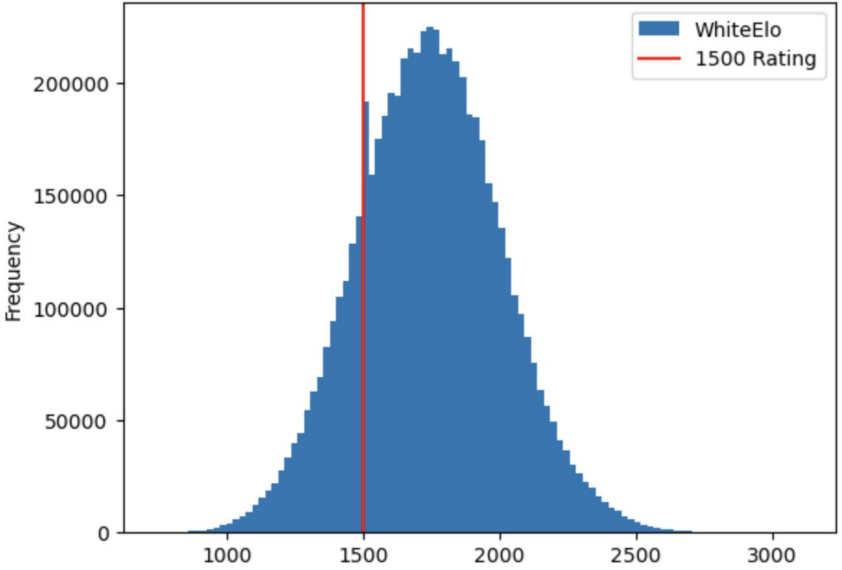
Data

- Start with all **6.25M games** played on Lichess in July 2016

- Find "new" players: ***1500 rating*** during their ***1st game of the month***
  - (**Starting rating** for a new player)

- For each new player, get 3 games:
  - New player's **1st-ever game**
  - New player's **2nd-ever game**
  - New player's **2nd-game opponent's most recent previous game** (if it exists)

Distribution of ratings of White player, in games from July 2016



Data (continued)

- We look only at new players who played ≥ 2 games in July 2016.
- We drop games with < 10 moves in total.
- Train / Test / Validate Split:

Data Set	**Number of New Players** (3 games of data per player)
Train	20,420
Test	1500
Val	1500

The Specific Problem We'll Solve

- Try to **predict outcome of new player's 2nd-ever game**

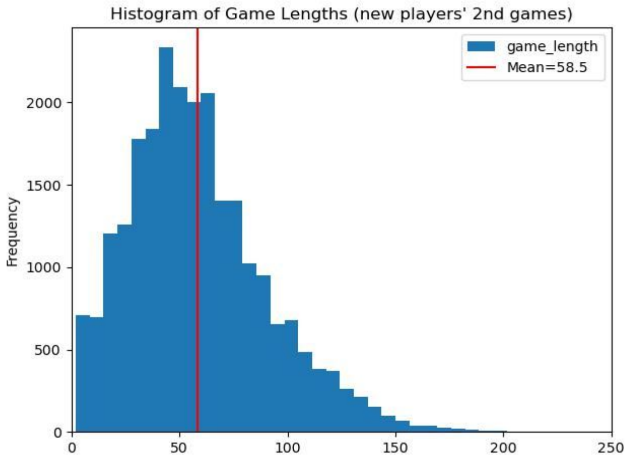
- Use ***move data*** from new player's 1st game

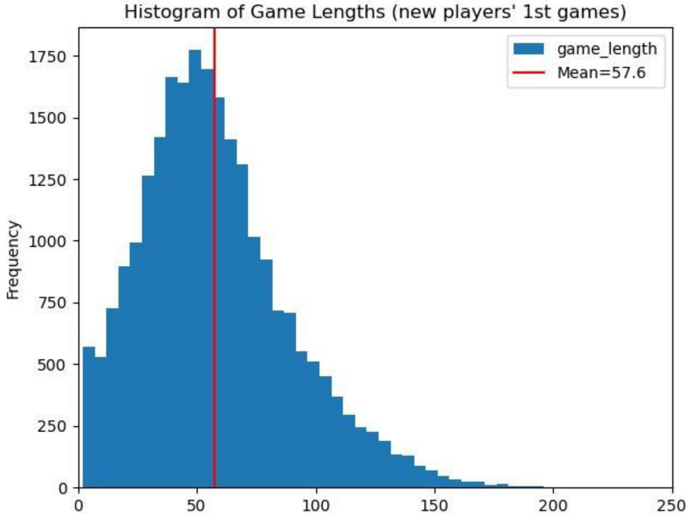
- Use "**metadata**" about upcoming 2nd game:
  - Players' ratings
  - Time limits on the game
  - Are both players new?

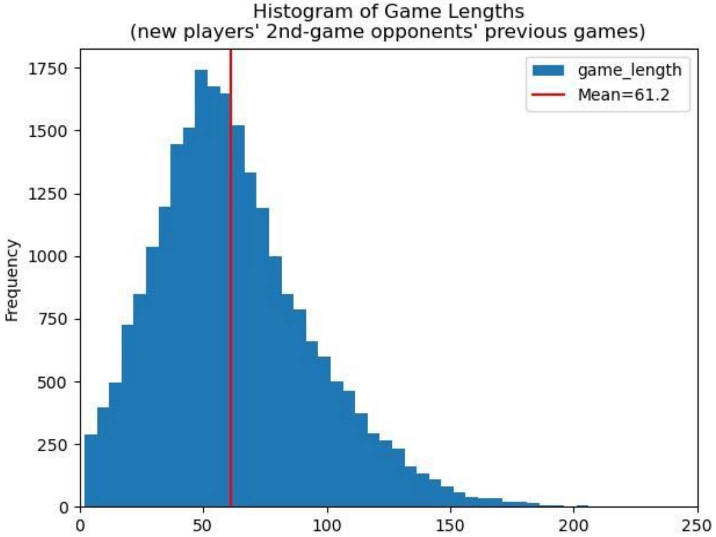
- Use above metadata & other metadata from both players' previous game:

  - Game outcome
  - Game length
  - Color played
  - Rating points gained/lost by each player
  - Did game end due to running out of time?
  - Did the loser concede before the end?

Game Lengths







Existing Models

Existing Models: Elo and Glicko

- The International Chess Federation (FIDE) uses “Elo”

- Many online chess platforms (including Lichess) use “Glicko”

- Both models give players a **rating score** and predict the outcome of a game using a **logistic function of the ratings difference** (P1’s rating) - (P2’s rating)
    - For Elo, P[P1 beats P2] is modeled as

$$\frac{1}{1 + 10^{-(r_1 - r_2)/400}}.$$

Glicko System

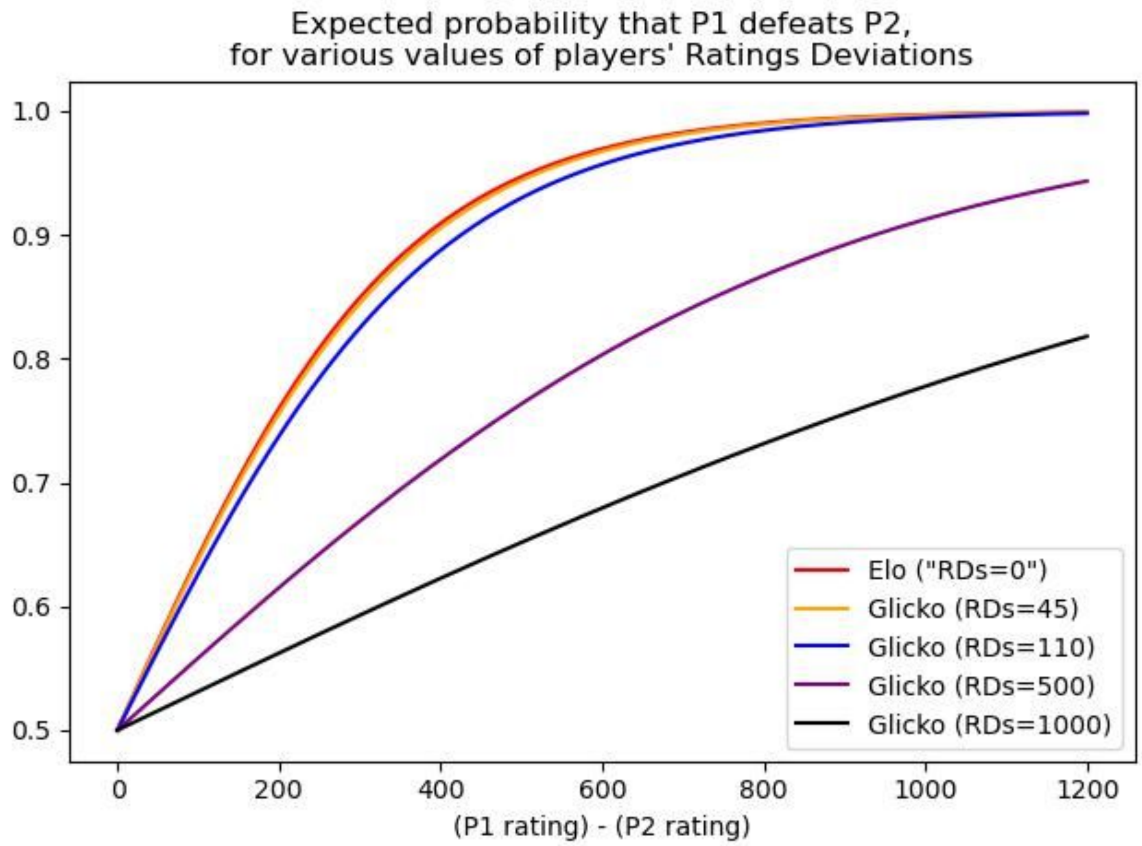
- Has several variants

- Each player has a **rating** r and a "**rating deviation**" RD measuring our uncertainty about her true skill level
  - Ratings reported as a 95% CI: **r ± RD**
  - New-player rating on Lichess: **1500 ± 1000**

- RD goes down as you play more games
  - Minimum possible RD on Lichess: **45**
  - RD needed to have a "certain" rating on Lichess: **110**

- Predicts P[P1 beats P2] = $\dfrac{1}{1 + 10^{-g\left(\sqrt{RD_1^2+RD_2^2}\right)\cdot(r_1-r_2)/400}}$

Glicko and Elo’s Probabilistic Predictions



Our Baseline Models

Our Version of “Glicko”

- We ***can’t*** **replicate Glicko’s *probabilistic* predictions** *for each match* without data on *individual players’ RDs before each game* (not provided by Lichess)
    - Would need other data too, to replicate Lichess’s “Glicko-2 Boost” prob. predictions

- Instead, we assume all players have the ***same RD*** and hyperparameter search for what this RD value should be (to minimize BCE of predictions)
    - Best RD to use is **408**

- We ***can*** **replicate Glicko’s *binary* predictions**!
    - These are the same as Elo: **“The higher-rated player wins”**

Our Baseline Models

- **Null Model**: Always predicts "New player loses with 100% probability"
  - Among non-draw games, the new player loses her 2nd game **53.7%** of the times
- **Uninformed Model**: Always predicts "New player wins with 50% probability"
- **Elo Model**
- **"Glicko" Model** (assuming everyone's RD=408)

Baseline Results

	bce_train	bce_test	bce_val	mae_train	mae_test	mae_val	acc_train	acc_test	acc_val
Null Model	Infinite	Infinite	Infinite	0.46403	0.464	0.456667	0.537332	0.537396	0.545014
Uninformed Model	0.693147	0.693147	0.693147	0.481758	0.481333	0.481333	0.5	0.5	0.5
Elo Model	0.687638	0.673591	0.686807	0.419215	0.411154	0.421461	0.621499	0.635734	0.628116
"Glicko" Model	0.65095	0.643291	0.652622	0.445177	0.440259	0.446432	0.621499	0.635734	0.628116

Our Modeling Techniques

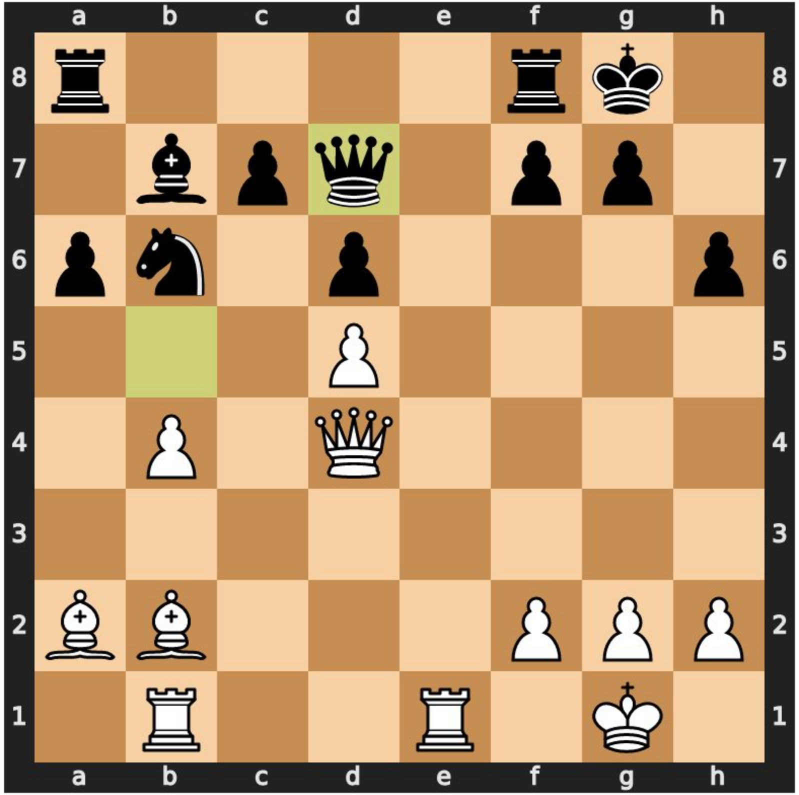
Our Main Innovation: Processing Move Data with Stockfish

- Use the ***move data from new player's 1st game***
  - Elo, Glicko only look at "metadata" of games

- Use chess engine **Stockfish to evaluate positions and potential moves**
  - Powerful and open-source
  - Uses intelligent tree search based on "Efficiently Updateable Neural Network" model
  - We use **search depth** = **15**; this equates to **Rating** ≈ **2563** (not exactly human, though)

- Get **evaluation of *every* board position** in new player's 1st game
- Get **"top 10 moves" on each of the *new player's* turns**

Example Stockfish Evaluation



Evaluation:

```
{'type': 'mate', 'value': 1}
```

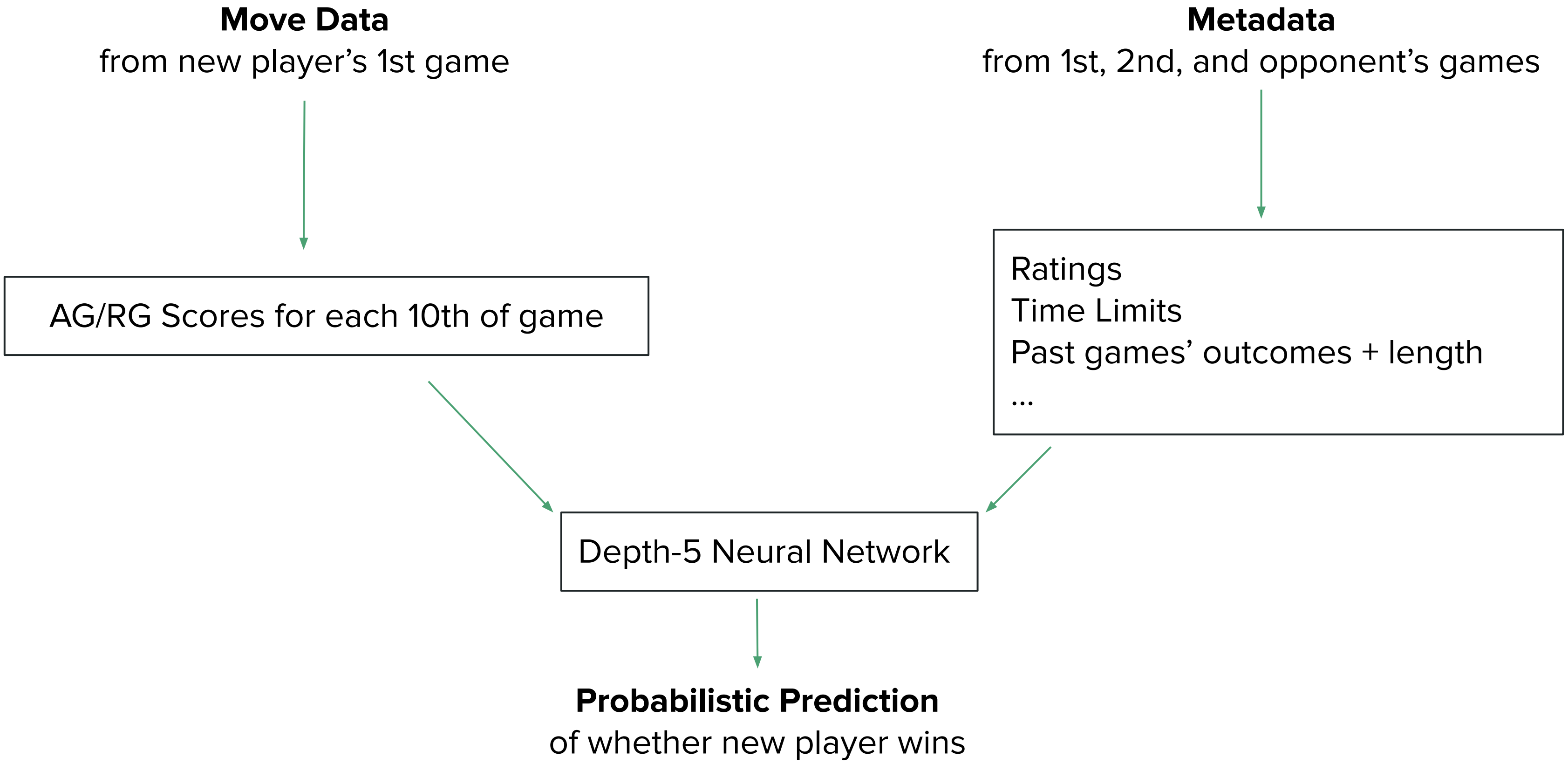
Top 10 Moves:

```
[{'Move': 'd4g7', 'Centipawn': None, 'Mate': 1},
 {'Move': 'e1e7', 'Centipawn': 488, 'Mate': None},
 {'Move': 'b1c1', 'Centipawn': 97, 'Mate': None},
 {'Move': 'h2h3', 'Centipawn': 83, 'Mate': None},
 {'Move': 'b1d1', 'Centipawn': 65, 'Mate': None},
 {'Move': 'e1e3', 'Centipawn': 48, 'Mate': None},
 {'Move': 'h2h4', 'Centipawn': 35, 'Mate': None},
 {'Move': 'b2a1', 'Centipawn': 24, 'Mate': None},
 {'Move': 'b1a1', 'Centipawn': 19, 'Mate': None},
 {'Move': 'f2f3', 'Centipawn': 13, 'Mate': None}]
```

Our Production Model: Outline

- From each new player's 1st game, get 2 sets of scores of how well she played:
  - **Absolute Goodness Score**
  - **Relative Goodness Score**
- Average these scores across each tenth of the game
  - Get a **2 x 10 matrix of AG & RG scores** for each 10th of game
- Feed **metadata + averaged scores** to a **neural network**
  - 29 metadata features + 20 scores
  - Depth-5 NN feeding into a sigmoid function
- Get **probabilistic prediction** of whether the new player wins

Our Production Model: Diagram



Learn an “Embedding” of Mate Scores into Centipawns

“= 1500 Centipawns” (say)

```
[{'Move': 'd4g7', 'Centipawn': None, 'Mate': 1},
 {'Move': 'e1e7', 'Centipawn': 488, 'Mate': None},
 {'Move': 'b1c1', 'Centipawn': 97, 'Mate': None},
 {'Move': 'h2h3', 'Centipawn': 83, 'Mate': None},
 {'Move': 'b1d1', 'Centipawn': 65, 'Mate': None},
 {'Move': 'e1e3', 'Centipawn': 48, 'Mate': None},
 {'Move': 'h2h4', 'Centipawn': 35, 'Mate': None},
 {'Move': 'b2a1', 'Centipawn': 24, 'Mate': None},
 {'Move': 'b1a1', 'Centipawn': 19, 'Mate': None},
 {'Move': 'f2f3', 'Centipawn': 13, 'Mate': None}]
```

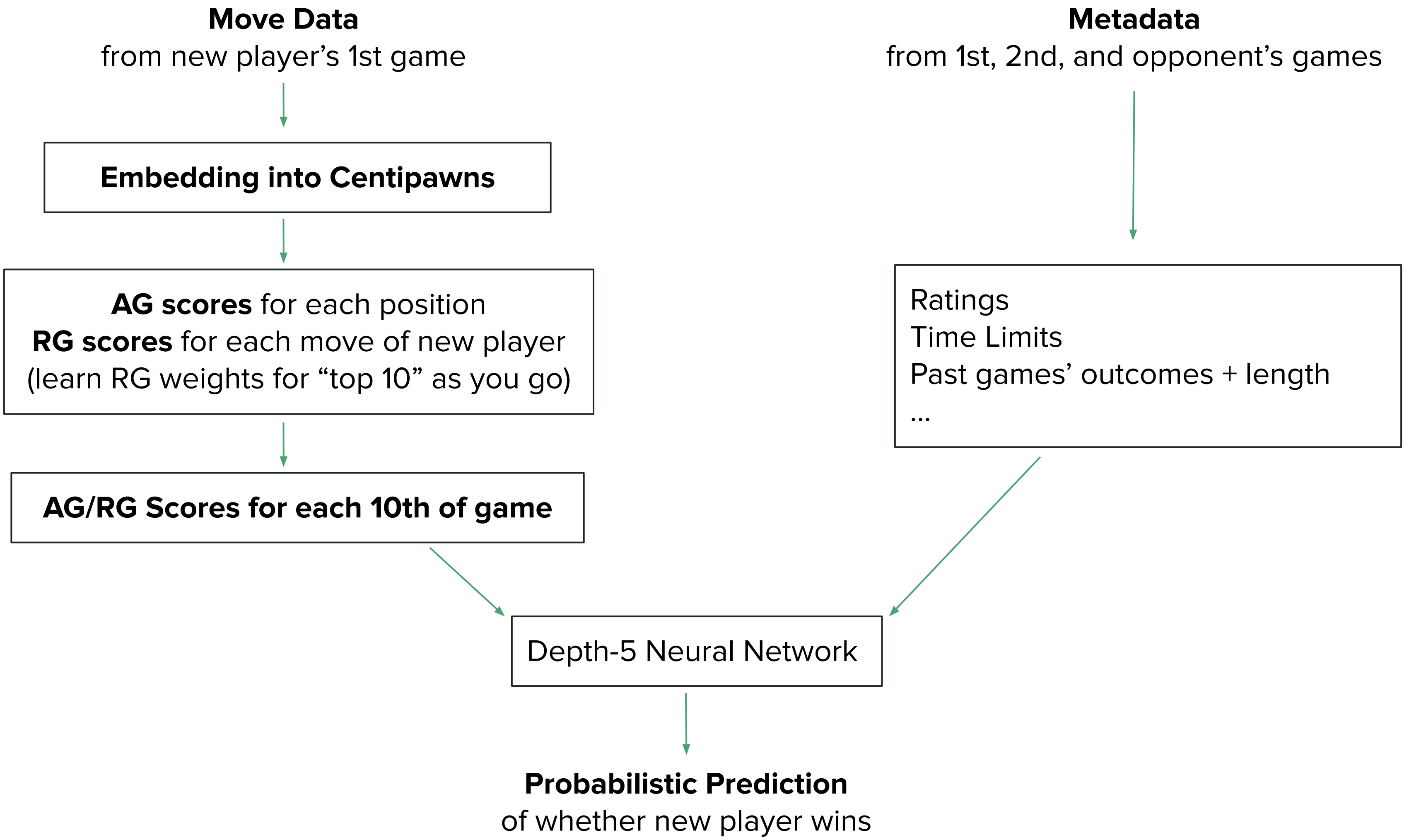
Absolute Goodness vs. Relative Goodness Scores

- **AG scores capture how strong the new players' positions are** over the course of the game
  - For each board position, AG score is just the Stockfish evaluation of that position
  - AG is **sensitive to strength of opponent** (RG is less so)

- **RG scores capture how well the new player selected moves out of those available to her** on each of her turns
  - If Stockfish centipawn scores of "top 10 moves" are $S_1, \ldots, S_{10}$ and the score of the actual move the new player made is S_{actual} then the RG score for this move is

$$\frac{S_{actual}}{w_{i,1}S_1 + w_{i,2}S_2 + \ \ldots \ + w_{i,10}S_{10}} - 1$$

where $w_{i,1}, \ldots, w_{i,10}$ are how the model learns to "weight" each of the top 10 moves during the *i*-th tenth of the game


Move Data
from new player's 1st game
Embedding into Centipawns
AG scores for each position
RG scores for each move of new player
(learn RG weights for "top 10" as you go)
AG/RG Scores for each 10th of game
Metadata
from 1st, 2nd, and opponent's games
Ratings
Time Limits
Past games' outcomes + length
...
Depth-5 Neural Network
Probabilistic Prediction
of whether new player wins

Additional Baseline: NN without Move Data

Move Data
from new player's 1st game

Metadata
from 1st, 2nd, and opponent's games

Ratings
Time Limits
Past games' outcomes + length
...

Depth-4 Neural Network

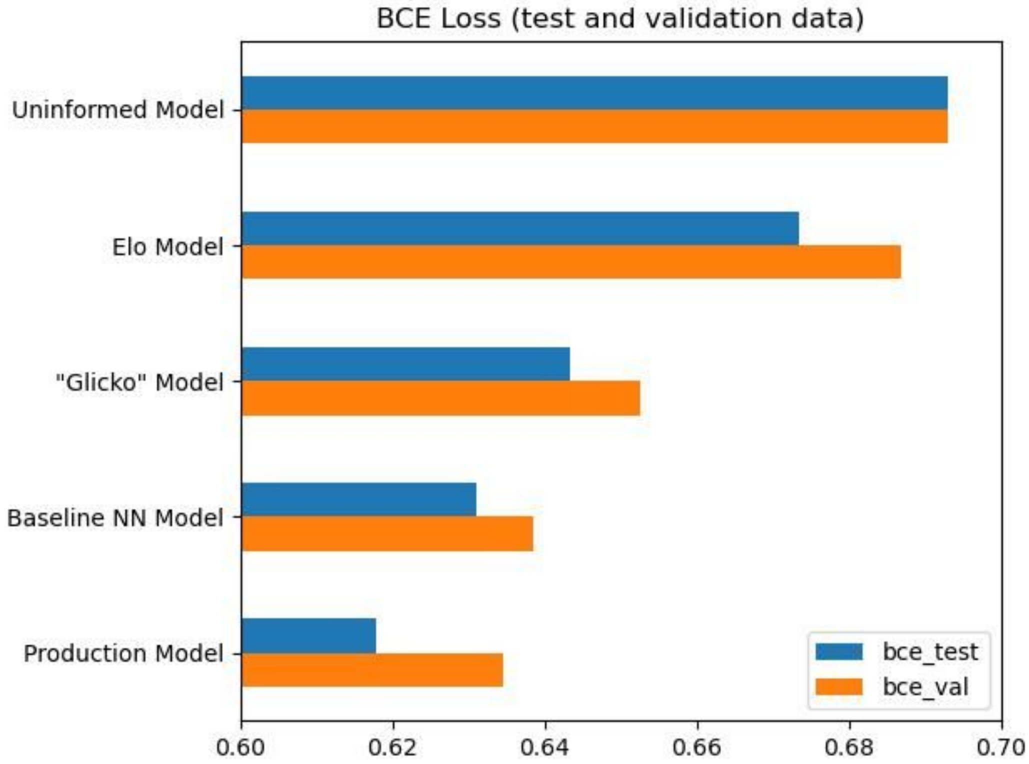
Probabilistic Prediction
of whether new player wins

Results

Table of Results

	bce_train	bce_test	bce_val	mae_train	mae_test	mae_val	acc_train	acc_test	acc_val
Null Model	Infinite	Infinite	Infinite	0.464030	0.464000	0.456667	0.537332	0.537396	0.545014
Uninformed Model	0.693147	0.693147	0.693147	0.481758	0.481333	0.481333	0.500000	0.500000	0.500000
Elo Model	0.687638	0.673591	0.686807	0.419215	0.411154	0.421461	0.621499	0.635734	0.628116
"Glicko" Model	0.65095	0.643291	0.652622	0.445177	0.440259	0.446432	0.621499	0.635734	0.628116
Baseline NN Model	0.638268	0.631029	0.638584	0.433393	0.429935	0.433968	0.646404	0.643352	0.649584
Production Model	0.624087	0.617946	0.634561	0.419645	0.418611	0.425845	0.665769	0.659972	0.653740

BCE Loss



Production Model Improvement over "Glicko" Model

(measured as a percentage of "Glicko's" improvement over Uninformed, on val data)

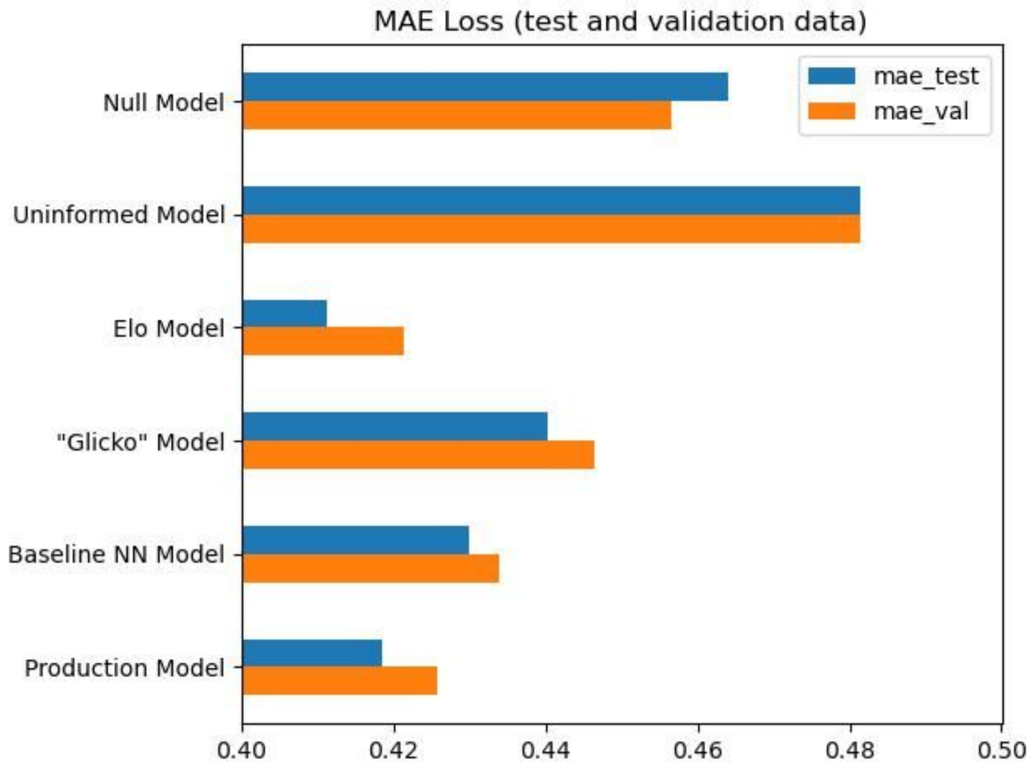
44.6%

Baseline NN Model Improvement over "Glicko" Model

(measured as a percentage of "Glicko's" improvement over Uninformed, on val data)

34.6%

MAE Loss



Production Model Improvement over “Glicko” Model

(measured as a percentage of “Glicko’s” improvement over Uninformed, on val data)

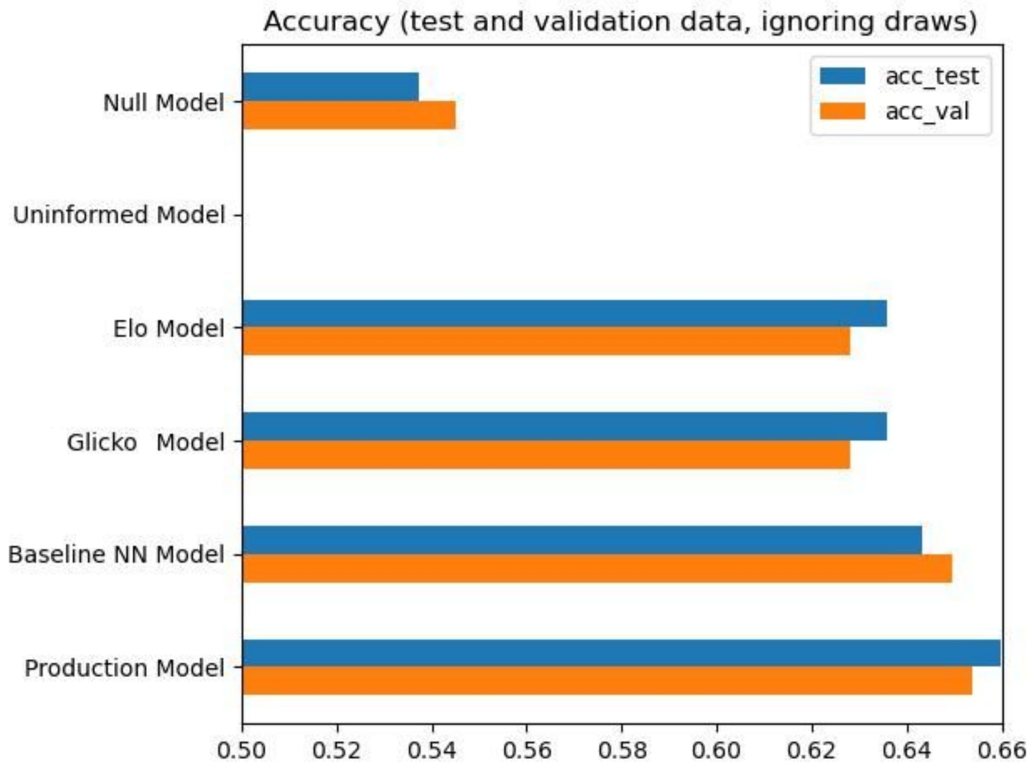
59.0%

Baseline NN Model Improvement over “Glicko” Model

(measured as a percentage of “Glicko’s” improvement over Uninformed, on val data)

35.7%

Accuracy (this time compared to *actual* Glicko!)



Production Model Improvement over Glicko Model

(measured as a percentage of Glicko's improvement over Uninformed, on val data)

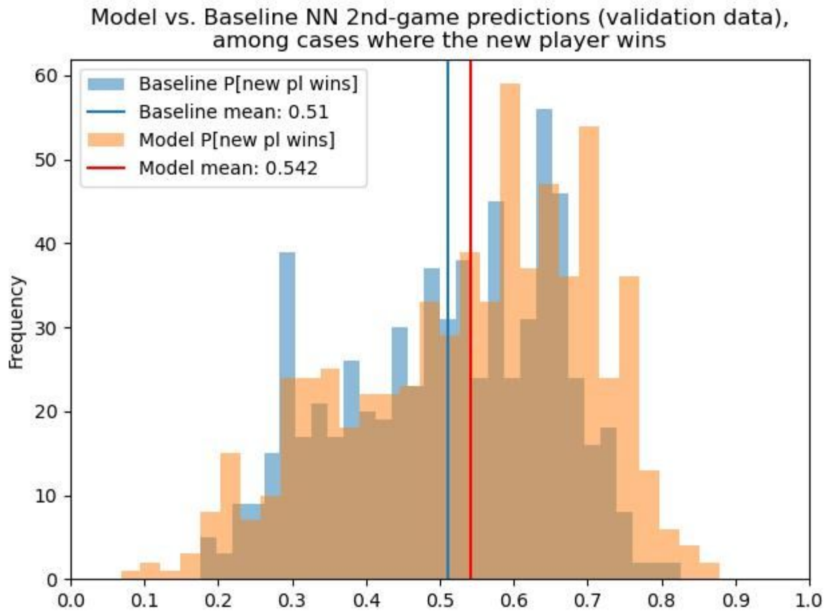
20.0%

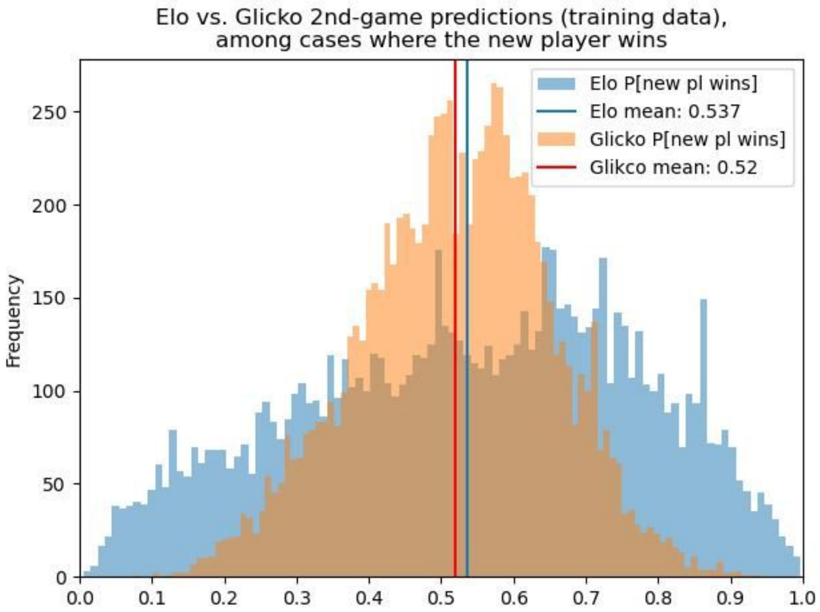
Baseline NN Model Improvement over Glicko Model

(measured as a percentage of Glicko's improvement over Uninformed, on val data)

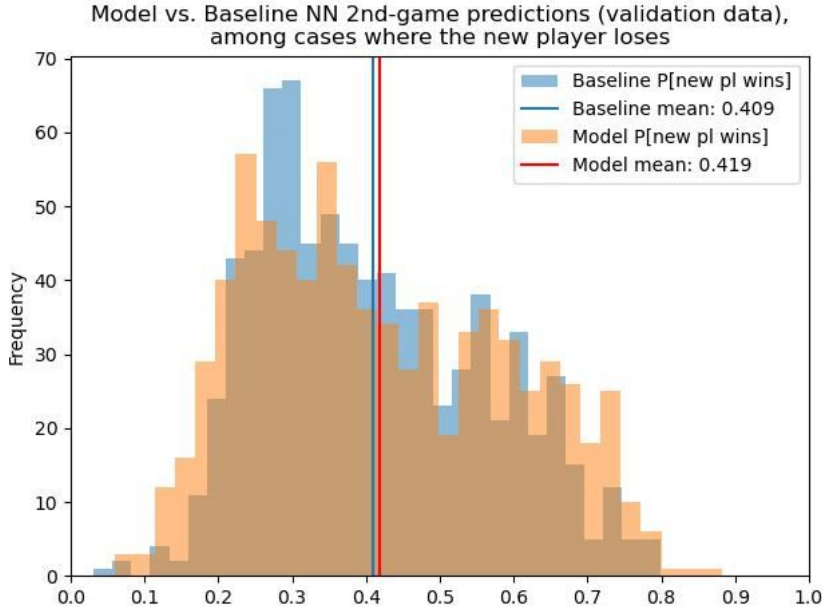
16.8%

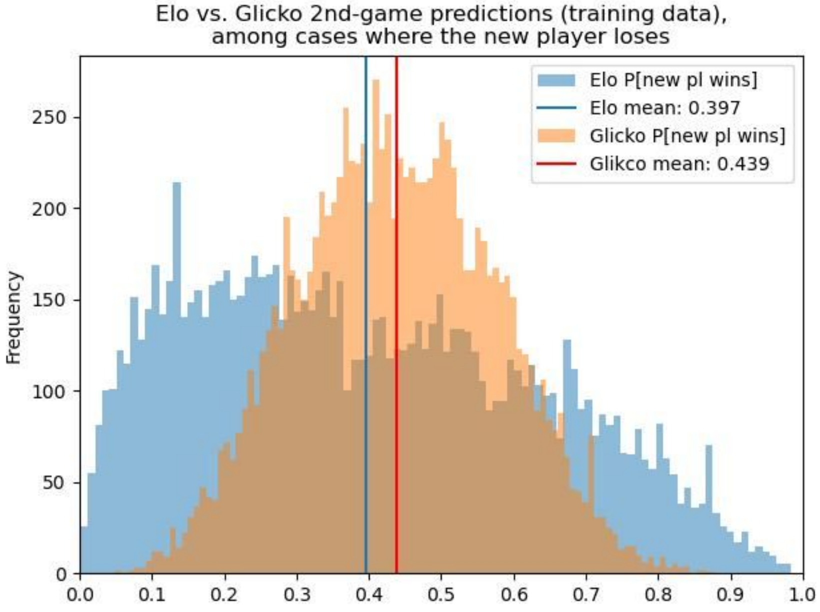
Predictions Among Games where New Player Wins



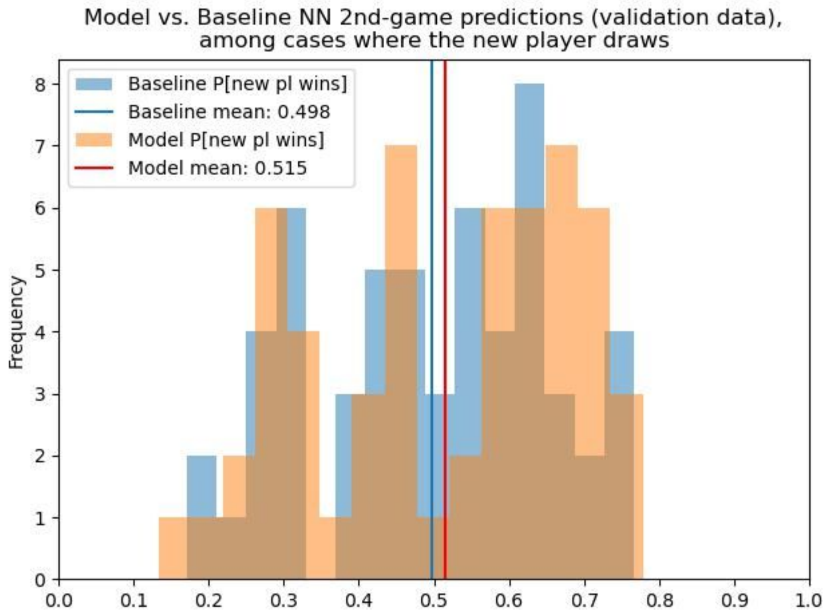


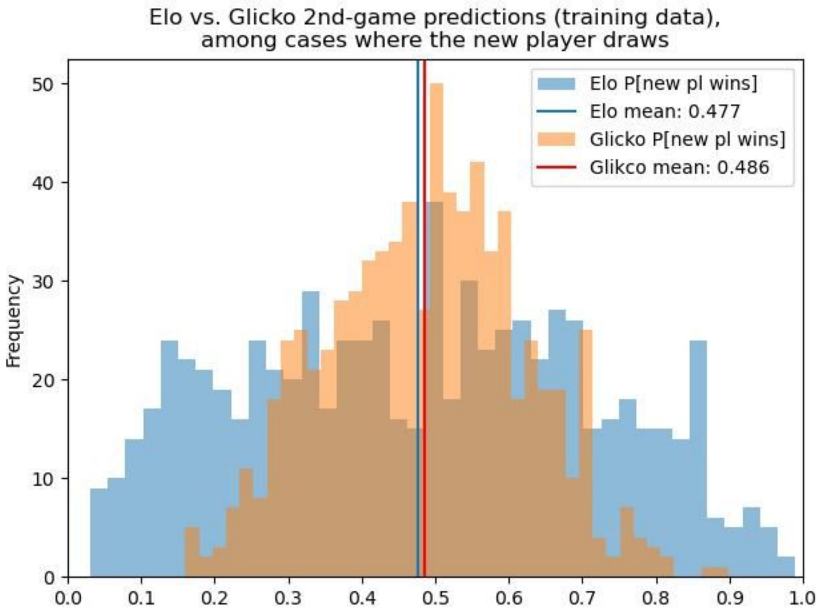
Predictions Among Games where New Player Loses





Predictions Among Games where New Player Draws





Conclusions

Conclusions

- **AI techniques** can significantly improve our ability to predict the outcomes of chess games among new players, leading to **better matchmaking**

- **Most of the improvement** of our model over existing models comes from using a **neural network**, not from looking at move data
  - This may change if we use **more games' move data**

- Using move data primarily improves model's ability to detect ***high skill*** of new player
  - Predictions were about as good as baseline NN among games where new player didn't win

- Plenty of room for tinkering and improvement!