League of Legends Match Outcome Prediction

Ethan Shapiro

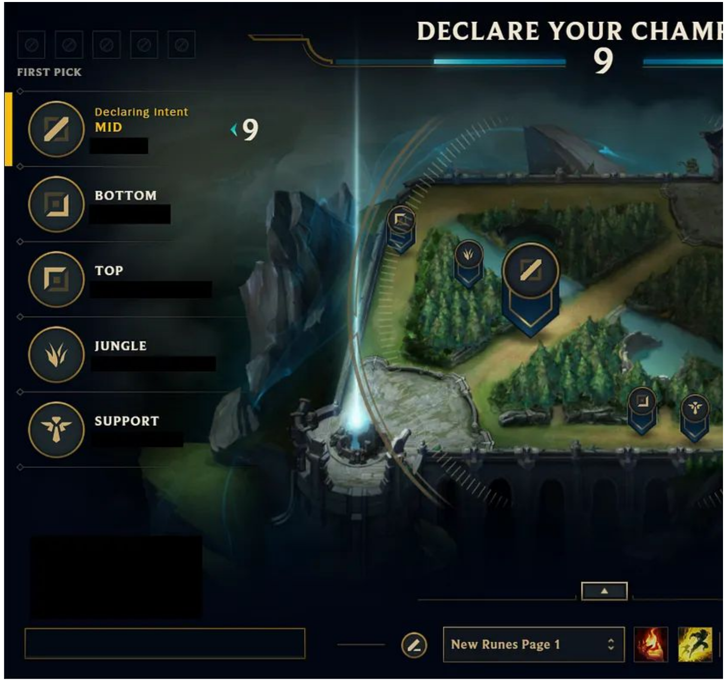
0. What is League of Legends?

League of Legends: A Team-Based Strategy Game

Similar to basketball or soccer, teams of 5 players compete with defined roles (e.g., defender, midfielder, striker).

Each player picks a unique "champion" (like choosing a specialized player) from over 160 options, each with unique skills and abilities.

Success relies on team coordination, individual performance, and strategic matchups—akin to setting up plays and countering opponents in traditional sports.



Champion Selection: What happens before the game

1. Problem Formulation

What is the goal?

We are trying to predict who will win the game before it starts.

In achieving our above goal, we also want to understand the factors for a team winning a league of legends game as well as beat other model's performance.

Why should we care?

Fair and balanced matches enhance player enjoyment and engagement.

If we can predict a match outcome before it begins, we are one step closer to building a more balanced matchmaking system.

2. The data

Raw Datasets

Match History

- Contains the information regarding all 10 player picks, bans, and roles for a match. It also contains when the match was played and who won.
- Over 400,000 totals matches

Player Performance History

- Contains the in-game match stats including kills, deaths, gold earned, etc.
- Over 12 million total matches
- The goal with this dataset is to provide some context to each player within the match

Match History Data

- Each row is one team in a specific match, so an entire match is two rows
- Over 400,000 totals matches
- Only 150,000 usable matches

match_id	game_creation	game_duration	…	team	pick1	lane1	won
NA1_5111521472	1726271773151	926	…	100	92	TOP	True
NA1_5111521472	1726271773151	926	…	200	122	TOP	False

Player History Data

- Contains the in-game performance statistics including kills, deaths, gold earned, etc.
- Over 12 million totals matches

match_id	game_creation	game_duration	…	user_id	assistMePings	goldPerMinute	win
NA1_5111521472	1707312577103	1422	…	IT7_x…0Vi	3	372.805906	True
NA1_4914990546	1726271773151	926	…	RT1zx…1ul	5	351.7041237	False

Data Prep General Process

1. **Raw Data Cleaning**
   a. Removed invalid matches or participants.
   b. Checked for missing or null values.
2. **Merging Datasets**
   a. Removed invalid matches or participants. Checked for missing or null values.
3. **Feature Engineering**
   a. Calculated aggregated stats (e.g., team total kills, gold difference).
   b. Created new features (details in next slide).
4. **Normalization**
   a. Normalized by game length except for binary statistics
   b. Applied z-score normalization for specific datasets.
5. **Encode Features**
   a. Encoded user ids, match ids, champion picks, etc. Only necessary for the transformer model
6. **Data Splitting**
   a. Divided data into train, validation, and test sets.

Features - Existing and Engineered

Original Features:

- Game Duration, Champion Played, Gold Earned, etc.

Engineered Features:

- Individual game performance metrics
- Aggregate stats across each team
- Z-scored features for balancing across games
- Role-based aggregates: Stats grouped by role (Ex. top, jungle, etc.)

Final Datasets

Individual Z-Scored Difference Dataset (IZ Diff)

1. Normalize player statistics based on their match history
2. Average performance over the last 50 games (max) for each player
3. Apply Z-scoring using the distribution of performance for their chosen champion
4. Subtract role-based performance differences to account for positional variations

Other Datasets Tested

Individual Z-Scored Dataset (IZ)

- Player-level stats normalized for comparisons across players

Team Z-Scored Dataset (TZ)

- Aggregated and normalized stats for each team in a match

Team Z-Scored Difference Dataset (TZ Diff)

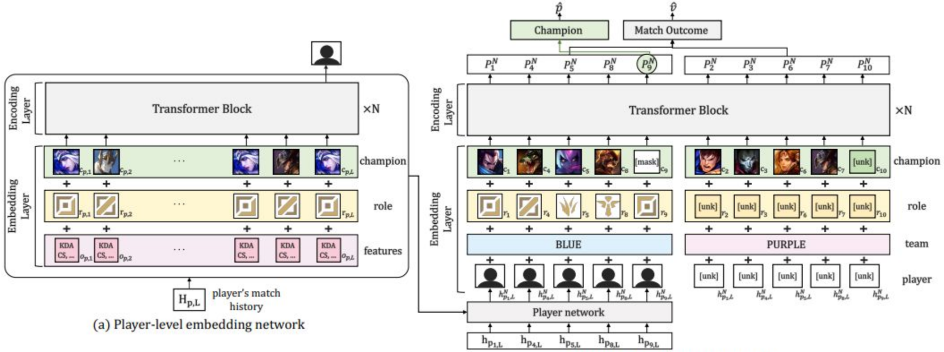
- Same as above but we take the difference between teams

Transformer Dataset

- Stats and matches are separated
- When training on matches, the player's history before that match is grabbed from the history dataset

3. Preliminary research

The most current model: DraftRec


Encoding Layer
Transformer Block
×N
Embedding Layer
champion
$c_{p,1}$ $c_{p,2}$ $c_{p,L}$
role
$r_{p,1}$ $r_{p,2}$ $r_{p,L}$
features
KDA CS, ...
$o_{p,1}$ $o_{p,2}$ $o_{p,L}$
$H_{p,L}$
player's match history
$\hat{p}$
$\hat{v}$
Champion
Match Outcome
P_1^N P_4^N P_5^N P_8^N P_9^N
P_2^N P_3^N P_6^N P_7^N P_{10}^N
Encoding Layer
Transformer Block
×N
Embedding Layer
[mask]
[unk]
c_1 c_4 c_5 c_8 c_9 c_2 c_3 c_6 c_7 c_{10}
champion
r_1 r_4 r_5 r_8 r_9 r_2 r_3 r_6 r_7 r_{10}
role
BLUE
PURPLE
team
player
$h_{p_1,L}^N$ $h_{p_4,L}^N$ $h_{p_5,L}^N$ $h_{p_8,L}^N$ $h_{p_9,L}^N$
Player network
$h_{p_1,L}$ $h_{p_4,L}$ $h_{p_5,L}$ $h_{p_8,L}$ $h_{p_9,L}$

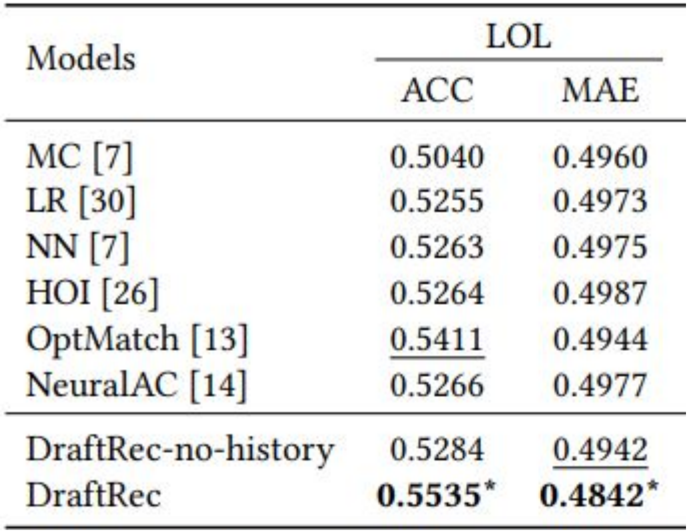
(a) Player-level embedding network

(b) Match-level embedding network

DraftRec

- *Draft Recommendation:* Suggests champions based on role strengths, player stats, and champion matchups.
- *Two Objectives:* Predict champion picks for better team synergy and game outcomes to evaluate draft strength.
- *Balanced Approach:* Combines complex matchup modeling with simple player stat analysis for accurate recommendations.

Previous Model Performances



Models	LOL	
	ACC	MAE
MC [7]	0.5040	0.4960
LR [30]	0.5255	0.4973
NN [7]	0.5263	0.4975
HOI [26]	0.5264	0.4987
OptMatch [13]	0.5411	0.4944
NeuralAC [14]	0.5266	0.4977
DraftRec-no-history	0.5284	0.4942
DraftRec	**0.5535***	**0.4842***

1. The best previous model is DraftRec with 55.35% accuracy
2. We are **essentially “fighting” the matchmaking algorithm** by trying to extract who will win before the game is played, so **~55% accuracy is quite good.**
3. **My goal** is just to see if I can achieve any **better than Draft Rec’s 55.35%.**

My First Challenge: Recreating DraftRec

Challenges…

Data Preparation Issues: DraftRec's dataset preparation was complex and since the documentation (already very minimal) is in Korean and didn't directly translate well, it was hard to debug.

Resource Constraints: Model training consumed over 70GB of memory, making it impractical to run efficiently.

Over-Complexity: The transformer-based model was too complex for the amount of data I had, so it wasn't able to learn well.

What I learned…

Start Simple: It's always important to get a baseline. This way, I know what the minimum performance is so I know what is an improvement.

Fit Model to Data: Even state-of-the-art models won't work if they're not suited to the dataset or the problem. DraftRec had a different overall objective and much more data, making it not as well suited for my objective.

Iterative Approach Matters: Slowly building up a model based on insights is more effective for understanding than starting out big. Not only that, the resources needed can stay within the bounds of my machine.

4. Basic Models and Learnings from them

Basic Model Performance

Model and (% accuracy)	Individual Z-Score Dataset	Individual Z-Score Difference Dataset (Final Dataset)	Team Z-Score Dataset	Team Z-Score Difference Dataset
Logistic Regression	55.73	**56.04 (Best)**	55.77	55.77
Random Forest	52.66	52.75	53.60	54.27
XGBoost	53.72	53.49	54.62	54.62
K-Nearest Neighbors	50.71	50.53	51.13	51.13

The Best Model: Logistic Regression

	Precision	Recall	F1-Score	Support
Win	0.56	0.55	0.56	14732
Lose	0.56	0.57	0.56	14767
Accuracy			0.56	29499

Things to Note

- Relatively well balanced between precision and recall, so we aren’t just guessing
- Losses were slightly easier to predict than wins

5. Findings from our Basic Models

What did we learn from the models?

Linear relationships are relevant and should be included.

None of the basic non-linear models performed as well as Logistic Regression. This suggests that at a minimum there is some linear relationship between win and the features.

What did we learn from trying different datasets?

Feature scaling matters.

Testing the z-score vs non-zscored datasets showed that the z-score data sets performed better.

What did we learn from trying different datasets?

The skill difference between players in each particular role is important.

The datasets with the difference between players performed better than the datasets compressed down to team metrics. This suggests that we lose too much information when only considering the overall team performance.

What did we learn from recreating DraftRec?

More complexity isn’t always better (especially with less data).

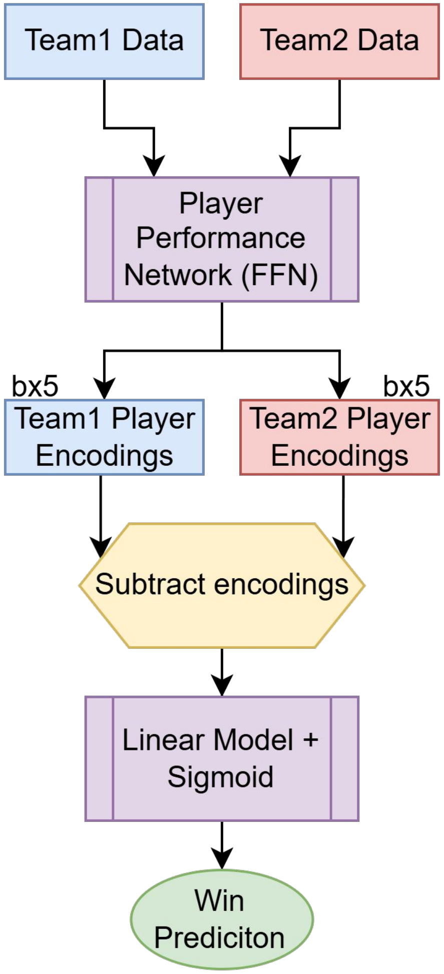
The most complex transformer model performed worse than even our worst basic model like XGBoost. Logistic Regression is the simplest but performed the best. This doesn’t mean more complexity won’t ever work, but for my dataset size, it may not be optimal.

6. Neural Networks

1. Role-Based Strength Model (Ignores Champions)

Concept:

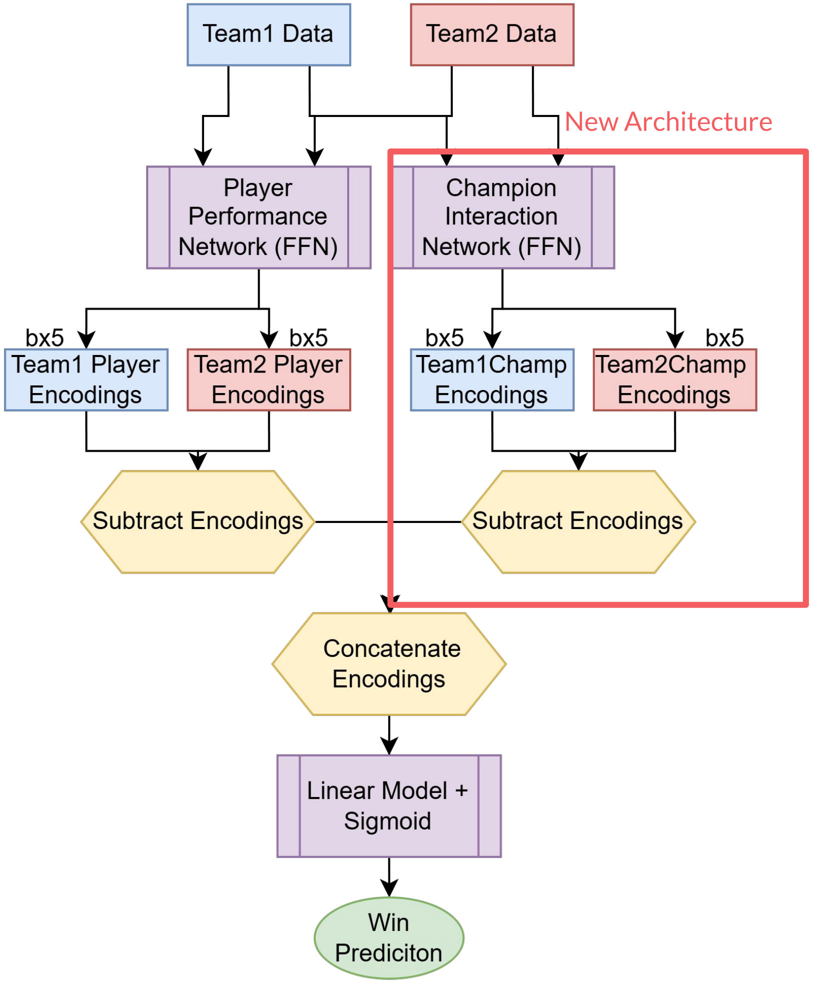
- Predicts win probability based only on player stats
- Each player's performance in a role is encoded into a scalar strength score
- The difference in role strength between teams is aggregated into a team advantage score



2. Role-Based Strength Model With Champion Info

Concept:

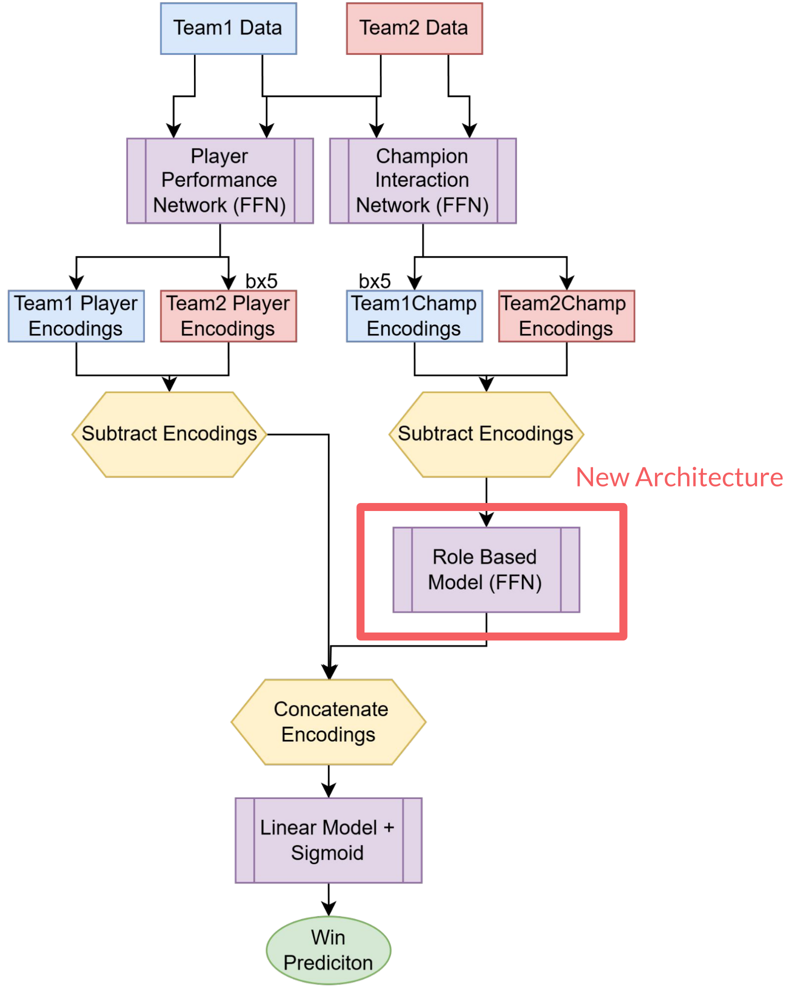
- Extends the baseline model by incorporating champion matchups
- Introduces champion embeddings to learn interactions between opposing champions



3. Role-Based Strength Model With Champion Network

Concept:

- Extends the Champion Info Model by adding a role-weighting mechanism, **learning role importance dynamically.**



4. Role-Based Strength Model With Champion Network and Linear Component

Deep Learning Component (Role-Based Strength Model)

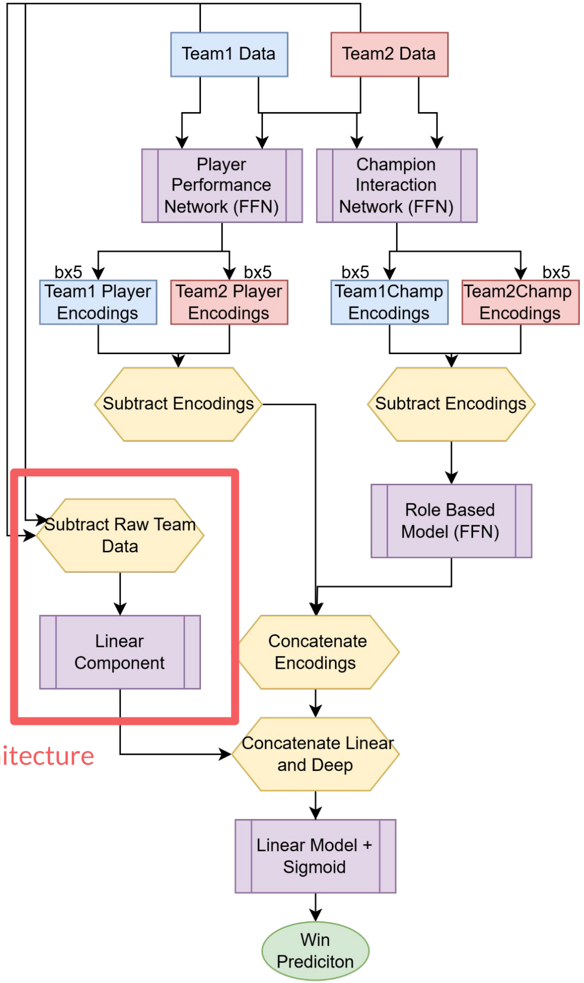
- Encodes player stats into role strength scores.
- Computes champion matchups using embeddings.
- Uses neural networks to process the information and generate role-based advantages

Linear Component (Raw Stats Difference)

- Directly processes team stat differences using a single linear layer.
- Produces a single numeric score per game.

Weighting Network

- Deep model output (role-based advantages)
- Linear model output (raw team strength differences)



New Architecture

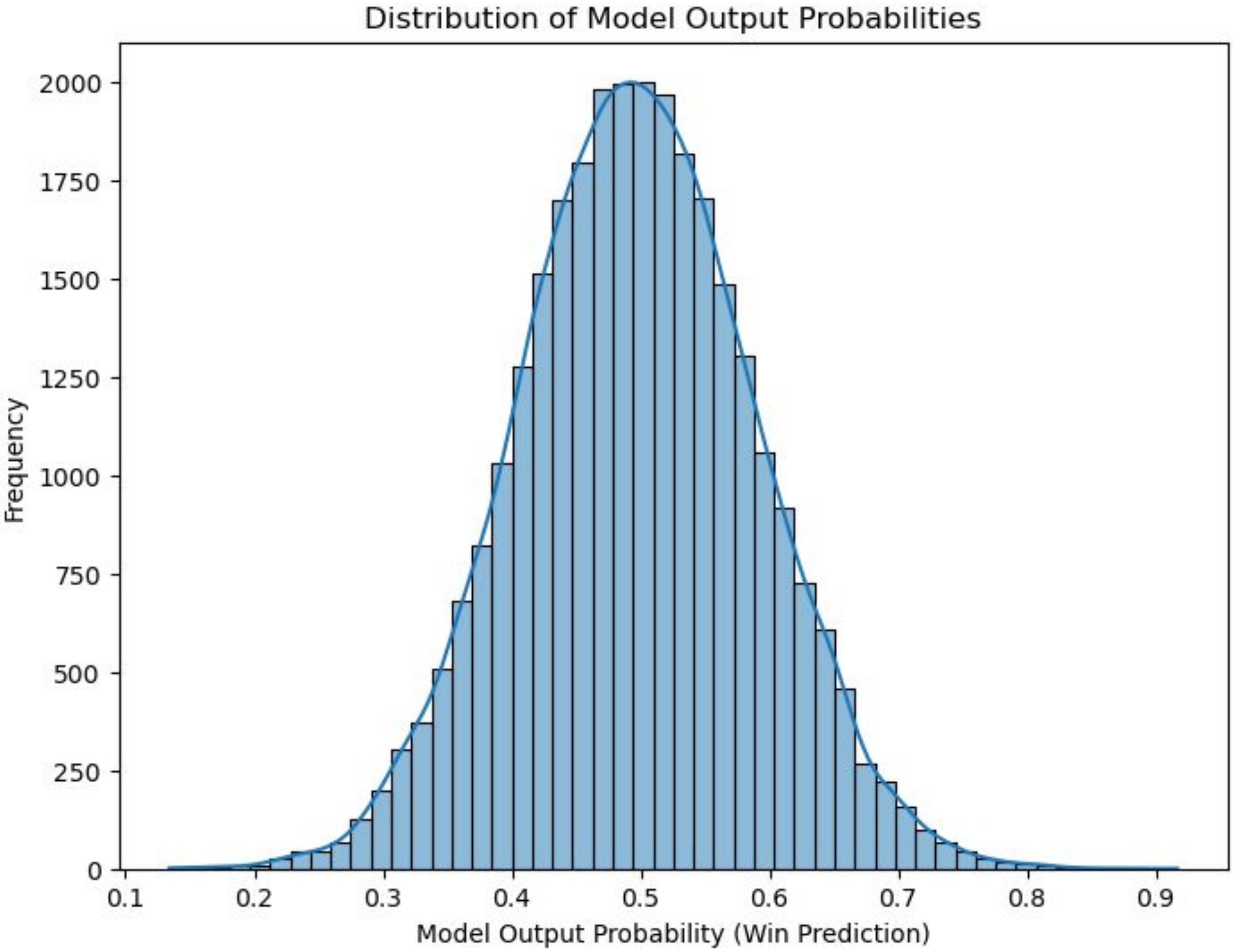
7. Performance and Comparison

Performance

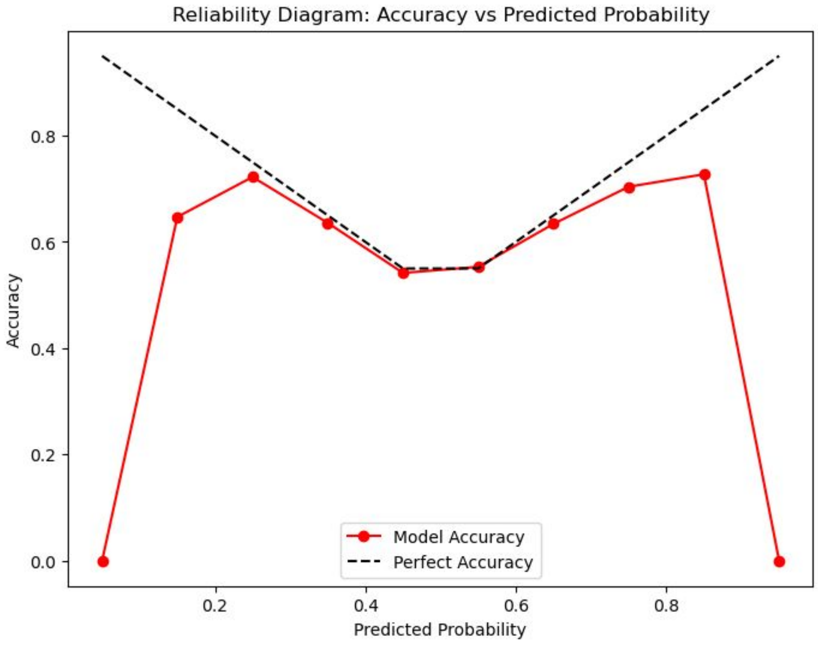
MODEL	% Accuracy	% Precision	% Recall	% F1-Score
Model 1	56.66	55.21	57.10	56.14
Model 2	56.81	55.50	57.35	56.41
Model 3	57.09	55.92	57.80	56.85
Final Model	**57.34**	**56.40**	**58.20**	**57.28**

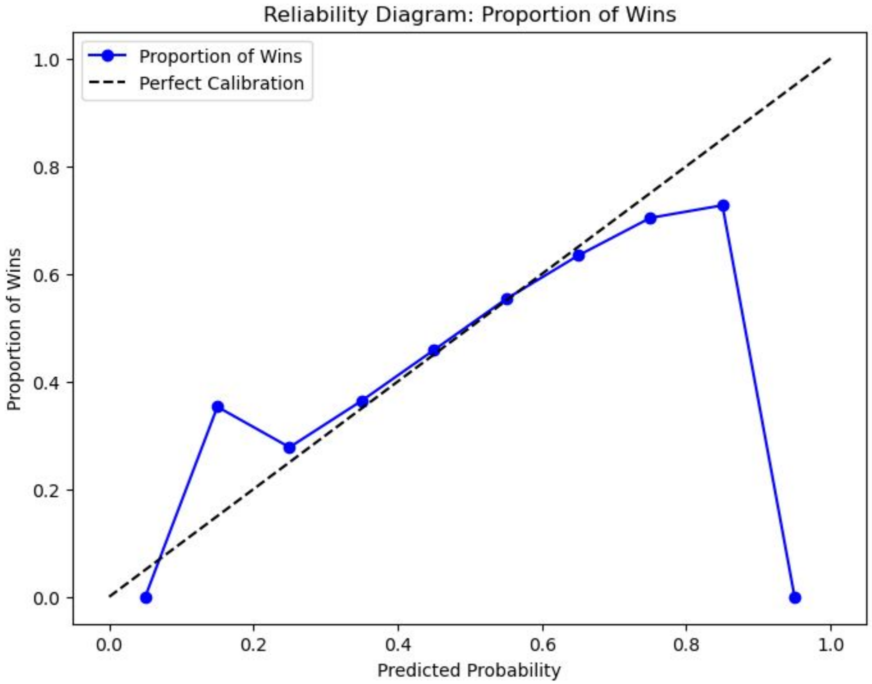
Since our output **probabilities are normal**, we know **our model learned** about the underlying data.

It's not guessing!



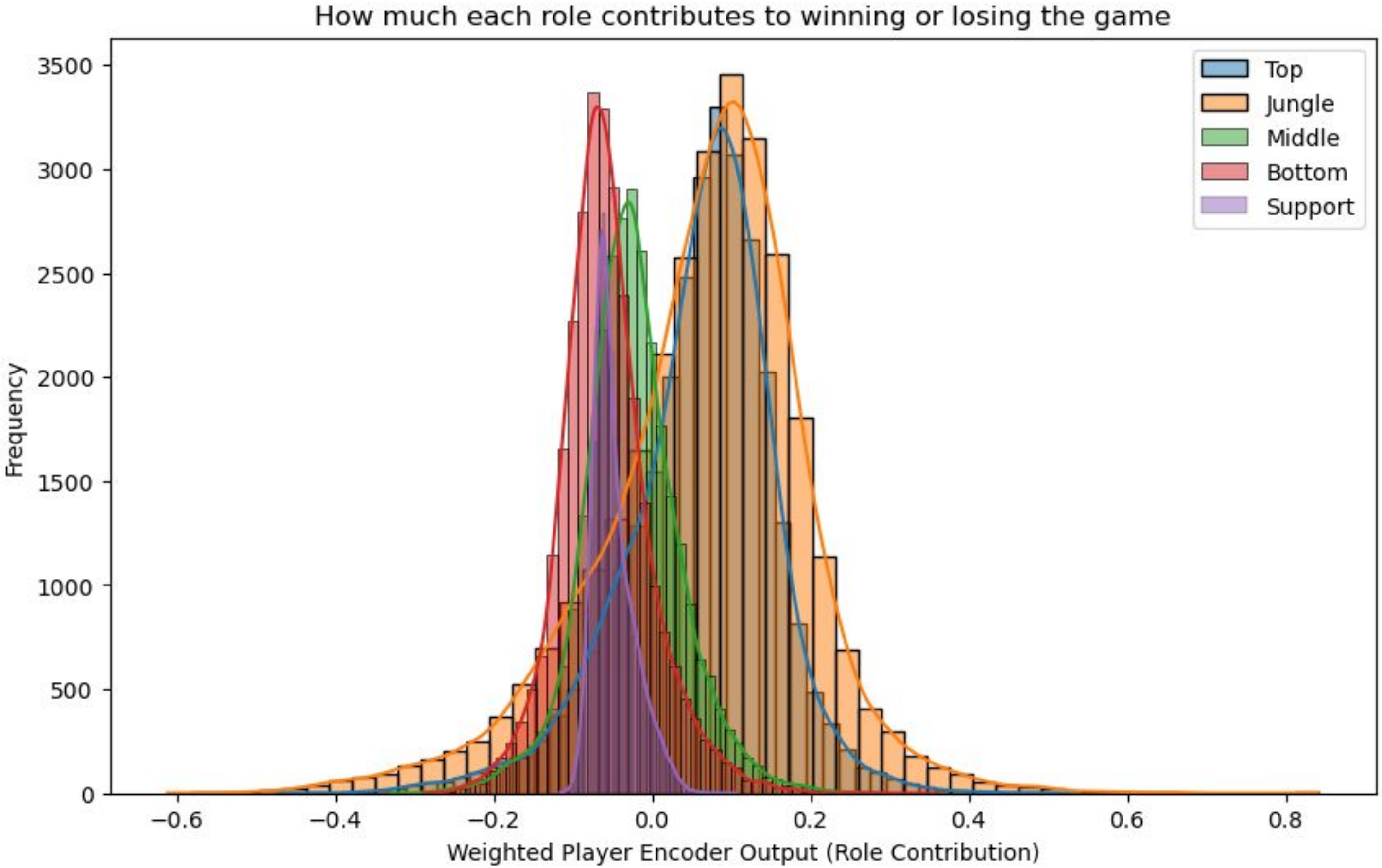
The accuracy line below shows that as we get further from 0.5 probability we get more accurate. This makes sense, as we are more certain about our prediction.



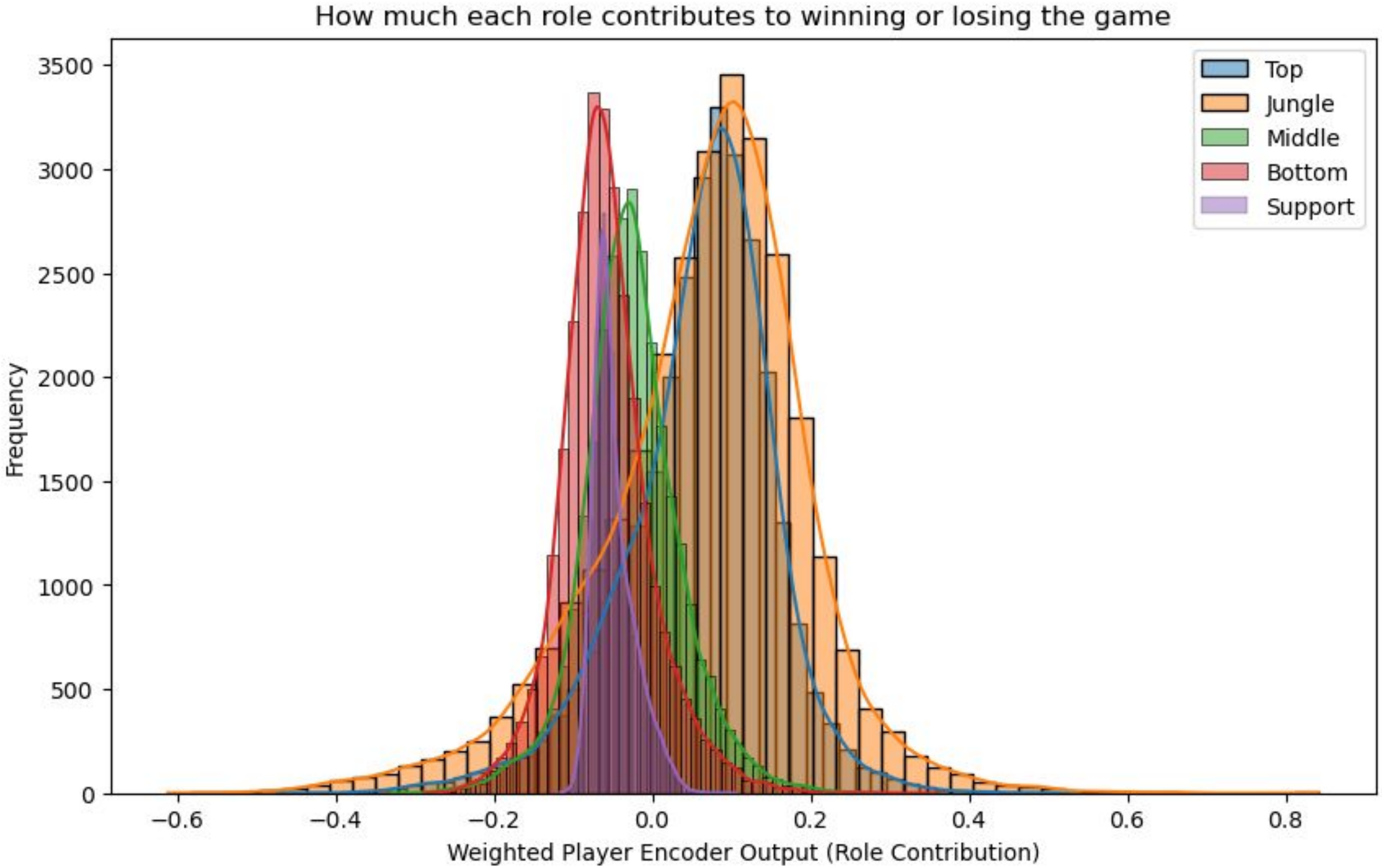


Although our proportion of wins is slightly higher than expected at the lower end of probability (meaning we can likely improve those predictions) the trend shows that we are mostly proportional to the win probability. This means we learned well!

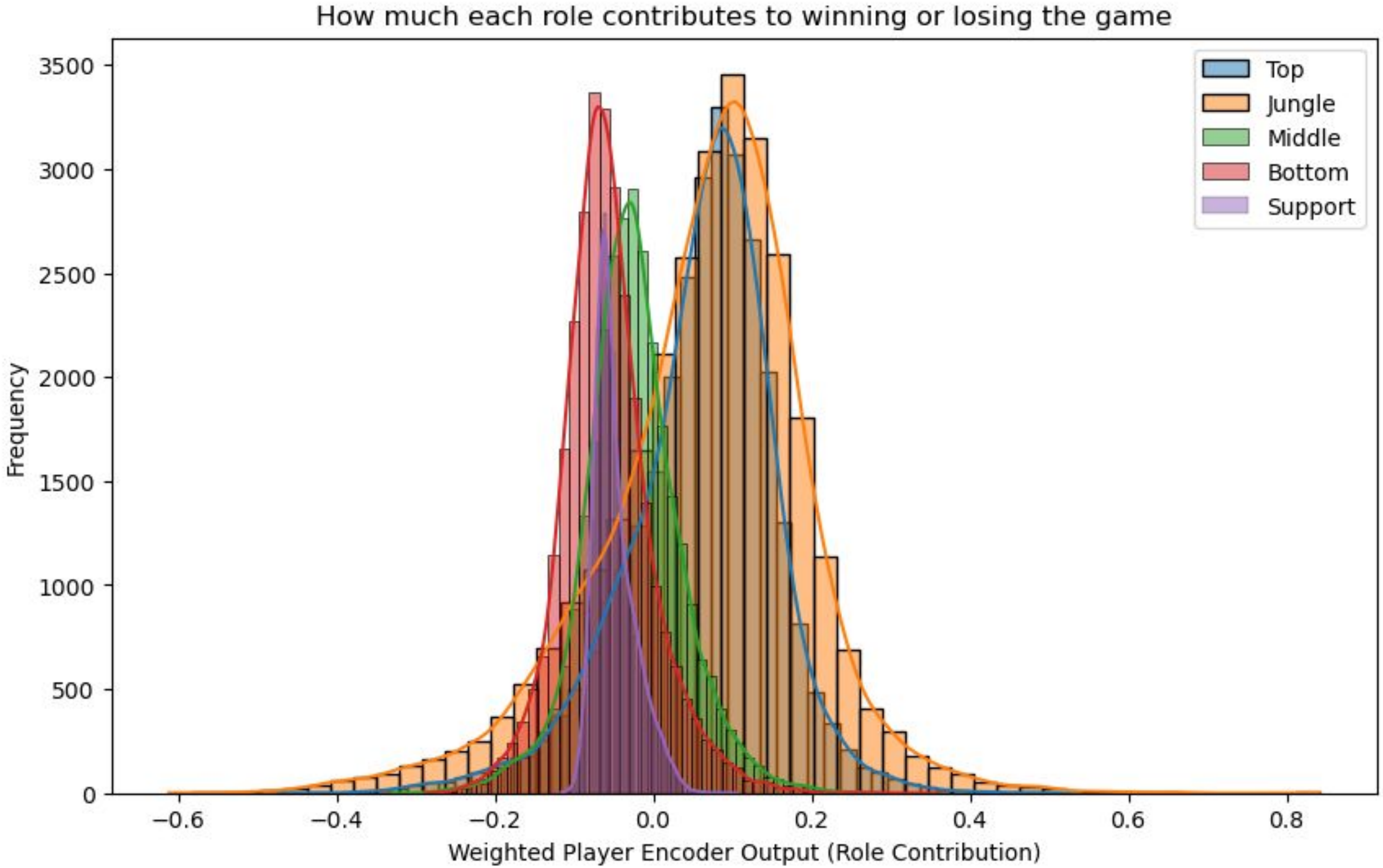
We can see that **Top and Jungle** have the **most impact** on winning or losing a game.



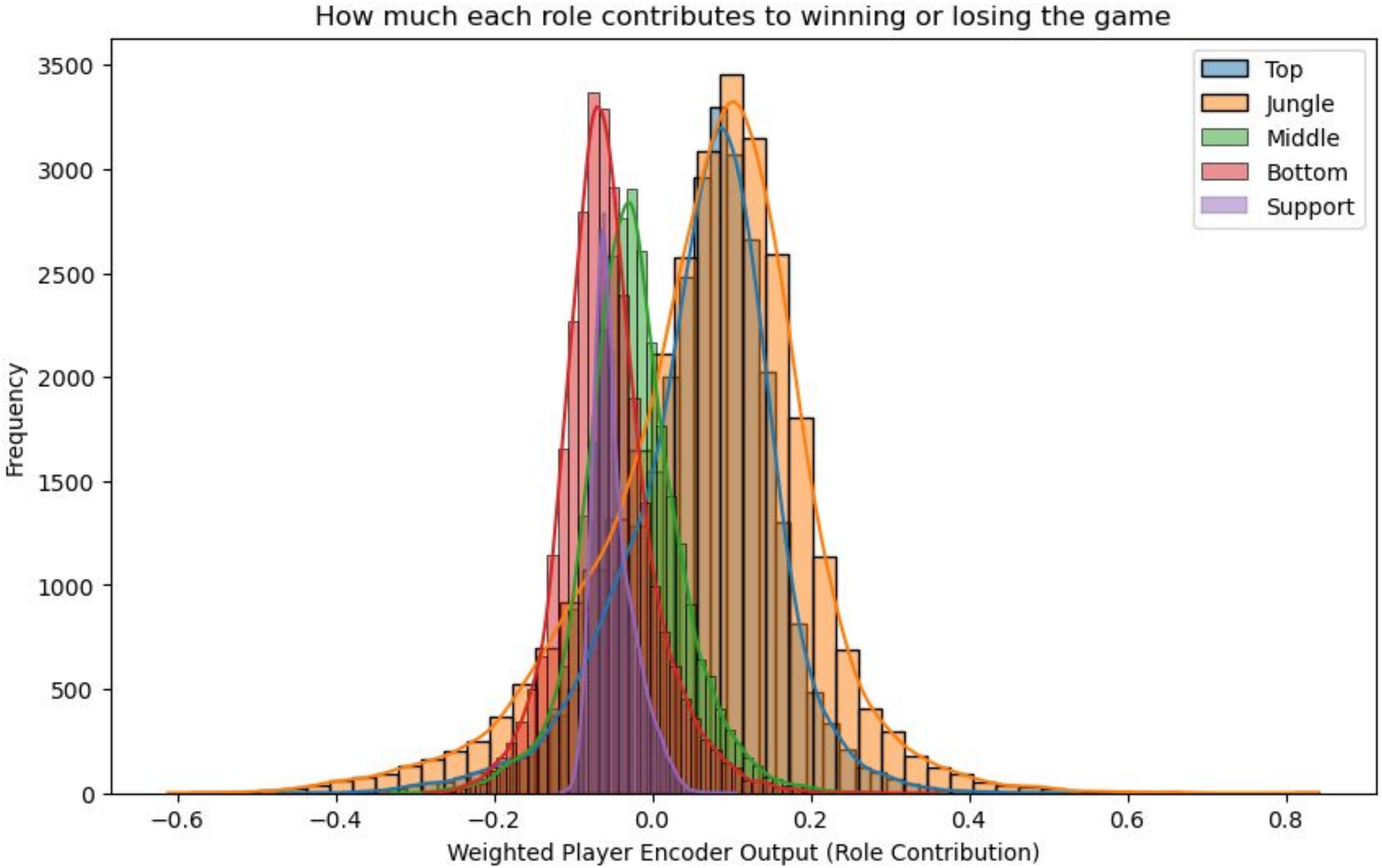
Middle player’s performance typically only **minorly affects the game** in either way.

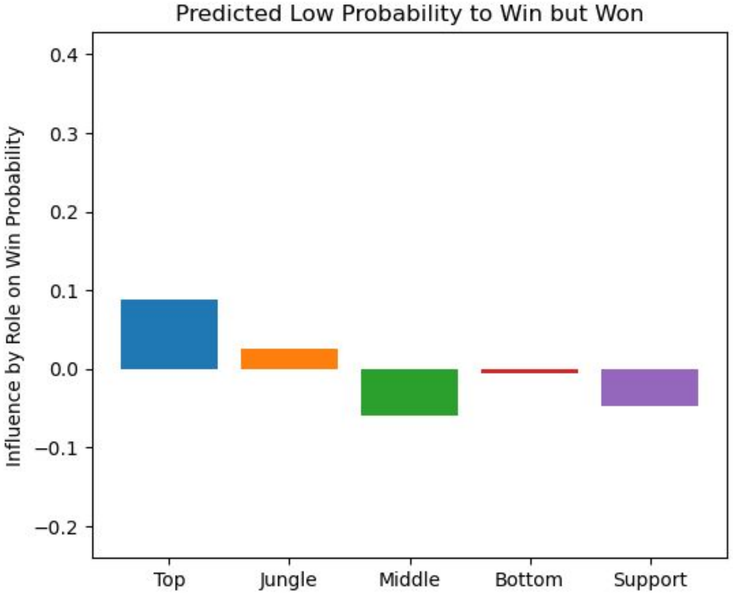


A **Support player's** performance **barely has an impact on the outcome** shown by the low variance.

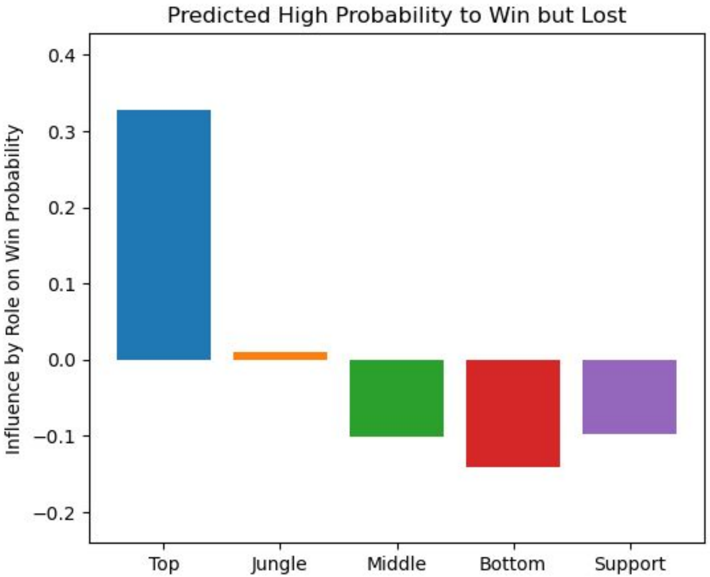


A **Bottom player's** performance **must be above average to positively affect** the outcome of a game.

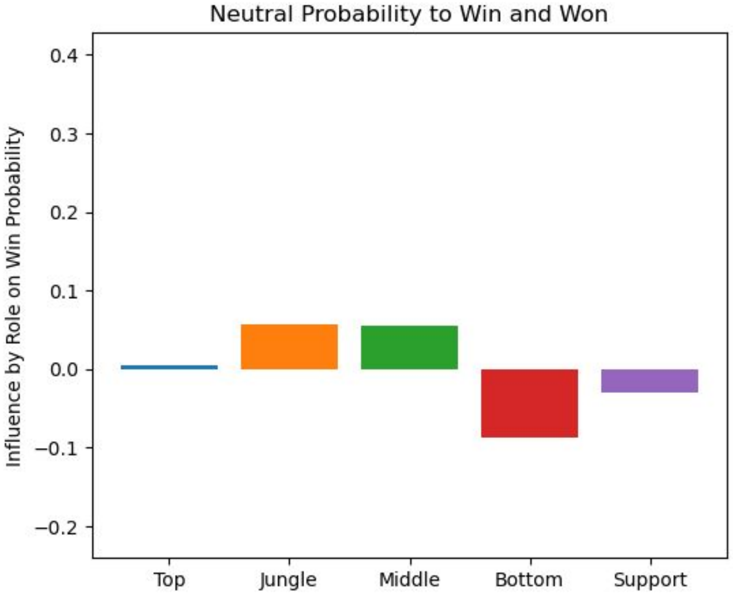




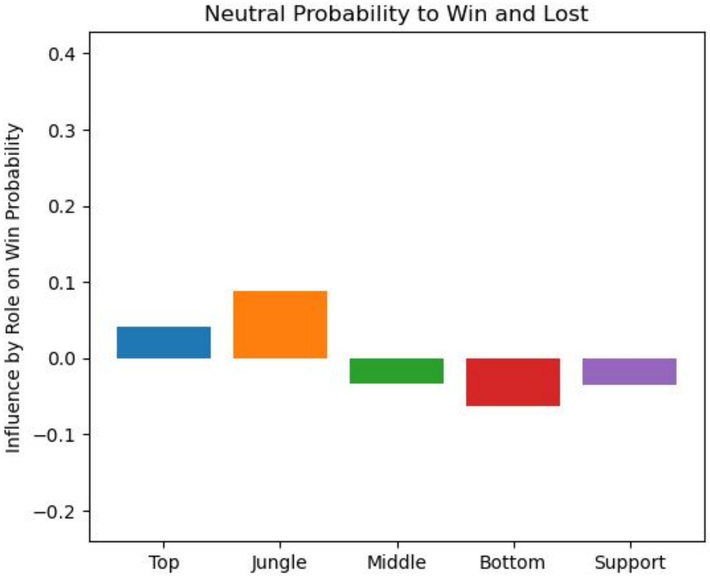
If top performs marginally better than the opponent and other teammates underperform, we think the team will lose



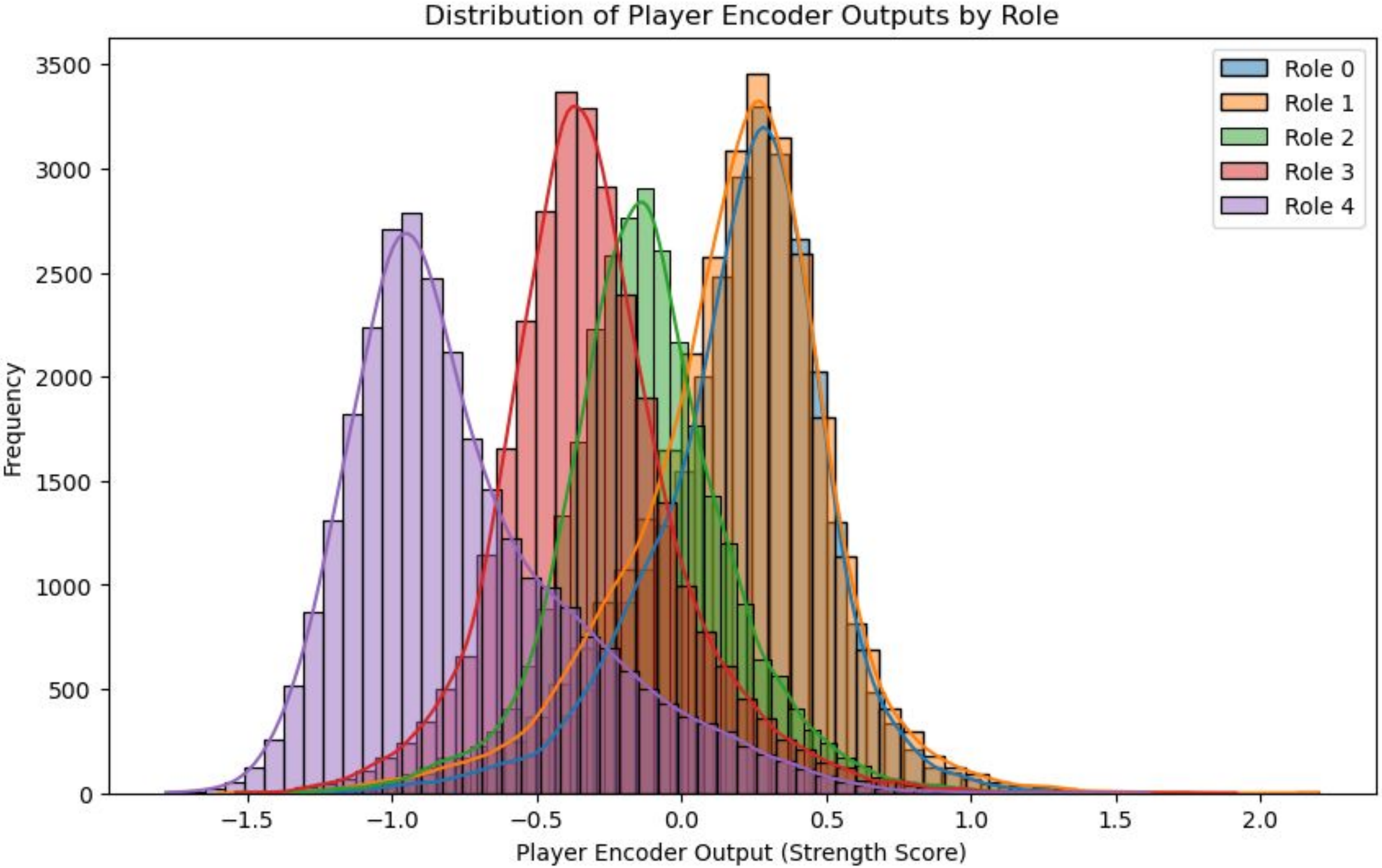
If top overperforms we think the team will win, despite our teammates underperforming.



Since top is performing average the model had a hard time telling.



Top also didn’t perform here so we had a hard time telling, despite jungle performing above average.


Distribution of Player Encoder Outputs by Role
Role 0
Role 1
Role 2
Role 3
Role 4
Frequency
3500
3000
2500
2000
1500
1000
500
0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
Player Encoder Output (Strength Score)

8. Challenges and what I learned from them

What was the problem? (Data Collection)

Collecting useful data scaled exponentially and became unsustainable (over 6 months).

The key mistake was not **prototyping and testing on smaller data subsets first** before committing to long-term data collection.

Fetching match data required retrieving full match histories, leading to an exponentially growing dataset. For every usable match, I needed match history for all 10 players, which expanded unpredictably with each new match. Additionally, older matches became inaccessible over time due to API limitations.

What did I learn?

Breaking the problem into smaller steps would have revealed this earlier.

Instead of focusing only on the final dataset I wanted, I should have:

1. **Estimated API limits and growth rates early** to anticipate scaling issues.
2. **Collected and analyzed a small sample first** to refine my approach.
3. **Designed a more efficient data retrieval strategy**, focusing on getting available data now before moving to a completely new set of matches.

What was the problem? (Model Creation)

The complexity of the model wasn’t catered to available resources and task.

Starting with a complex model made it difficult to debug and track improvements. Without a simple baseline, it was unclear whether changes were actual improvements or just random fluctuations.

What did I learn?

Always consider the resources available and data you have to solve your task.

Instead of trying the state of the art model, I will:

1. **Choose a model that fits the available data and computing resources** to ensure feasibility.
2. Avoid complexity for the sake of it, models should be designed to fit the actual problem being solved.

What was the problem? (Model Creation)

Starting with a complex model made it hard to debug and understand what progression was.

Starting with a complex model made it difficult to debug and track improvements. Without a simple baseline, it was unclear whether changes were actual improvements or just random fluctuations.

What did I learn?

Start simple. Baselines can guide assumptions and clearly define improvements.

Instead of immediately jumping to the complex model, I will:

1. **Start with a simple model** to establish a baseline before adding complexity
2. **Measure improvements iteratively** rather than assuming a complex model will perform better.

What was the problem? (Model Training)

Small performance changes made it difficult to evaluate improvements.

With low and near-random model performance, it was difficult to determine whether small improvements were meaningful or just noise. Balancing precision, recall, and accuracy was challenging because an increase in one could come at the expense of another.

Even traditional metrics like F1-score proved too sensitive to small fluctuations, sometimes misrepresenting overall model quality.

What did I learn?

Averaging metrics wasn’t enough, we had to analyze raw values.

Instead of relying solely on aggregated metrics, I:

1. Analyze raw precision, recall, and accuracy for each class individually.
2. **Optimize for a more balanced distribution rather than just maximizing a single metric.**
3. Save models based on a **combination of accuracy and balance** rather than just peak scores.

9. Final thoughts and potential future areas of interest

Final Thoughts

The combination Linear and Deep Neural Network performed the best, showing that we can improve our baseline Linear model with cross-feature interactions.

It was interesting that the Logistic Regression model performed the best at first, but thinking about how performance of player's is likely normally distributed and likelihood of winning is also likely normally distributed, it makes sense.

It also makes sense that to the difference between individual roles is important, as almost half of the game is simply against your opponent of the same role.

Future Work

While my model performed well, I think incorporation of champion matchups could be done better. Combining a graph neural network to represent champion matchups with detailed information about each champions strengths, weakness, etc. I believe could improve the model further.

For all of my models I used the average of a player's historical performance to predict whether a player will win their match. The hardest matches are ones where players are statistically close in performance. It's more important we can accurately predict matches that aren't likely to be even so we don't match those teams/players together.