Utilizing Graph Neural Networks to Model Major League Baseball At-Bat Outcomes

Selah Dean

ABSTRACT

Baseball is a game of numbers and in recent years the use of data analytics has grown tremendously as teams look for any advantage they can get. Baseball analytics has traditionally relied on basic statistical models and machine learning approaches that tend to treat players in isolation. This approach overlooks the complex dependencies between players especially when considering the pitcher versus hitter match up that is central to the game. This project proposes modeling pitcher-hitter matchups as a bipartite graph, where nodes represent players and edges capture detailed at-bat interactions. It will utilize graph neural networks (GNNs) to predict future at-bat outcomes leveraging the relational structures in the data. The research will explore different GNN architectures and assess the importance of different features in the predictions. Results show GNNs achieving 38% accuracy for four-class outcomes and 31% for eight-class outcomes. Graph calibration loss improved probabilistic predictions, while temporal encodings enhanced strikeout prediction specifically. Despite AUC scores ranging from 0.51 to 0.68 across outcome classes, the inherent randomness in baseball presents a natural ceiling on predictive accuracy. This work contributes to both baseball analytics and the broader field of sports analytics by introducing a novel application of GNNs to individual player interactions.

1 INTRODUCTION

Baseball is built from the hitter versus pitcher matchup. The hitter is trying to reach base while the pitcher is trying to get an out. Managers are also trying to find the matchup where their hitter or pitcher is most likely to succeed. Many factors can contribute to the potential outcome of the at-bat including simple attributes such as handedness, but there are also more complex attributes such as the pitcher's arsenal or the hitter's tendencies.

Traditional baseball analytical methods rely on statistical models and machine learning approaches that often treat players in isolation or use aggregate statistics. However, these methods may overlook the intricate relationships between pitchers and hitters. To address this limitation, this project will model the pitcher-hitter matchups as a bipartite graph, where nodes represent pitchers and hitters with their relevant attributes, and an edge exists between two nodes when a pitcher and hitter have previously faced each other. The edge contains features representing their previous at-bat including when the at-bat occurred, the game situation of the at-bat, and the ultimate outcome of the at-bat (strikeout, walk, groundball, etc.). Multiple edges can exist between the same pitcher and hitter if they have faced each other more than once over the season or their careers.

This project explores the use of graph neural networks (GNNs) to predict at-bat outcomes by leveraging the structural relationships within the pitcher-hitter network. It will examine if a GNN can effectively model pitcher-hitter interactions to improve at-bat outcome predictions. It will also assess what features contribute the most to predictive performance and how historical interactions influence future matchups. To answer these questions, this study will train a GNN on historical Major League Baseball (MLB) data using datasets containing individual pitcher and hitter statistics as well as detailed at-bat records. By capturing the complex dependencies between players, this approach aims to offer further insight into the pitcher-hitter matchup.

The results demonstrate that GNNs can effectively capture relational information in baseball data, though predictive accuracy remains constrained by the inherent variability in the sport. A Graph Attention Network model using graph calibration loss achieved 38% accuracy when classifying outcomes into four categories (in-play air, in-play ground, strikeout, walk/HBP) and 31% accuracy for eight detailed outcome categories. The addition of temporal encodings to edge features improved strikeout prediction specifically, suggesting time-dependent patterns in pitcher-batter interactions. Our models produced well-calibrated probabilities, particularly for high-frequency outcomes, though they struggled with rare events like bunts and hit-by-pitches. This research contributes a novel graph-based framework for baseball analytics that captures the complex interdependencies between players, providing a foundation for more sophisticated modeling of matchup dynamics in baseball.

2 BACKGROUND

2.1 Related Work

The use of data analytics in sports especially in baseball has increased greatly since the early 2000s. In recent years machine learning applications in sports analytics have grown as well. Koseler & Stephan (2017) [1] did a systematic review of machine learning applications in baseball looking into how machine learning algorithms can be applied to find valuable insights into player and team performance. The literature they found primarily used traditional machine learning algorithms and found that support vector machines and k-nearest neighbors are some of the most common methods.

More recent research has demonstrated the effectiveness of GNNs in sports analytics. Tracy et al. (2023) [2] explored the use of graph-based encodings in volleyball. They modeled player interactions and game sequences using graph convolutional networks (GCNs), graph attention networks (GATs), and Graph Transformers. They found that GNN-based models significantly outperformed traditional machine learning models by preserving the inter-player dependencies and sequential interactions. Similarly, Xenopoulos & Silva (2021) [3] developed a general graph representation of game states that can be applied to a variety of sports. They used their model to predict yards gained in American football and estimate win probability in esports. They also found improved performance and they were able to answer the "what if" questions that occur in sports through their modeling of player interactions. These studies demonstrate the generalizability of GNNs in sports analytics and supports the idea that relational modeling improves predictive accuracy.

It appears that GNNs have yet to applied to the problem of predicting player matchup outcomes in baseball. Silver (2020) [4] developed a neural network model named Singlearity-PA that is designed to predict the outcome of plate appearances in MLB. Singlearity-PA provides more accurate predictions across various scenarios when compared to traditional methods that have been used for this problem such as log5. The model includes various predictive variables and handles specific game situations. The research proposed in this project builds on use of deep learning but exploits the network structure of the data. In a flat deep learning model, at-bats are treated as independent events, so the evolving relationship between a pitcher and hitter is not captured. A graph-based model is also able to leverage the contextual information from other similar matchups.

Outside of the area of sports, there has been research done to explore the use of GNNs for modeling dynamic relationships and predicting future interactions. Kim et al. (2019) [5] propose an edge-labeling GNN (EGNN) which improves on few-shot leaning by updating edge-labels and considering both intra-cluster similarity and inter-cluster dissimilarity. This model also performs well for predicting multiple classes. The concepts from EGNN are relevant to the setting of modeling at-bat outcomes since the focus is on predicting edge labels. Additionally, Rossi et al. (2020) [6] introduce a temporal graph network (TGN) which is used for learning on continuously evolving graphs. The experiments in the study focus on link prediction rather than edge labeling and find that their results significantly outperform the baseline models. While the task in this study is future edge prediction rather than edge label prediction, the concept of a dynamic graph is relevant to modeling at-bat outcomes. Both pitchers and hitters make adjustments over time, so time can be an important element when considering the pitcher-hitter relationship. The proposed project of modeling MLB at-bat outcomes will build upon concepts from the EGNN and TGN models that have been introduced in existing literature.

3 METHODS

3.1 Data

The data for this project came from two different sources Baseball Savant and Retrosheet.

The pitcher and hitter features were obtained from Baseball Savant [7, 8]. When selecting features, the focus was on process statistics such as fastball velocity for pitchers or exit velocity for hitters instead of outcome statistics such as earned run average for pitchers and batting average for hitters. The reasoning for this was to isolate what pitchers and hitters have the most control over to improve the finding of similar pitchers and hitters when modeling. There were 17 features selected for pitchers: handedness, percent of each pitch type thrown, average fastball velocity, walk and strikeout rates and 10 features selected for hitters: handedness, average exit velocity, average launch angle, swing percent, in-zone and out-zone contact percentage, whiff rate, strikeout and walk rates.

The information to create the graphs and the edge features was obtained from Retrosheet which has detailed play-by-play data for all MLB games [9]. For the modeling, data used was from the 2022, 2023, and 2024 regular seasons. From this dataset, only rows which resulted in an end of a pitcher-hitter matchup were retained (eliminates stolen bases, caught stealing, etc.). Each row has a column with a detailed event string that contains information about the end of the matchup. Retrosheet has a key for these strings to given a simplified version of the events made by the batter at the plate. These simplified events resulted in 25 different outcomes. These outcomes were simplified further into eight different outcomes to isolate the pitcher versus hitter matchup and eliminate any defensive plays that could have affected the outcome. These outcomes and their distribution can be seen in Figure 1.

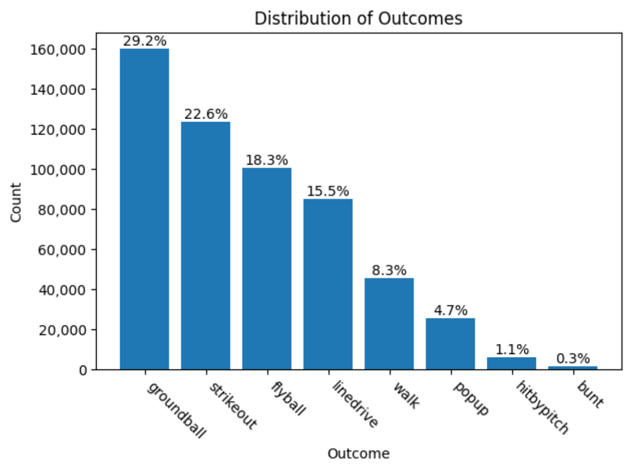


Figure 1: Model Data Outcome Distribution

Some of the models tested also further simplify these results on four classes combining groundball and bunt into a class called in play ground, combining flyball, pop-up, and linedrive into a class called in play air, combining walk and hit-by-pitch into one class, and leaving strikeouts as the final class. The distribution of the four classes can be seen in Figure 2.

Edge features were also selected from the Retrosheet dataset. These features focused on the situation when the at-bat occurred. There were five features included: inning, number of outs prior to the at-bat, and an indicator for whether there was a runner on each base. In addition to these features, there is a date attribute for when the matchup took place. To connect the node features to the edge features, Chadwick Register was used to map the player id's from the different sources. The features were also connected by year since the node features were season-based and daily pitcher/hitter data is unavailable. Additionally any edges for a pitcher or hitter that did not appear in the node features data set were dropped. These corresponded to players who have fewer than ten plate appearances in the season since the Baseball Savant data only includes such players.

Once all of the features were complied, all continuous numeric values were standardized (grouped by year to adjust for seasonal variation) and categorical features were encoded.

For training and validation the 2022 and 2023 season data was used. When creating the validation set the edges were kept in order

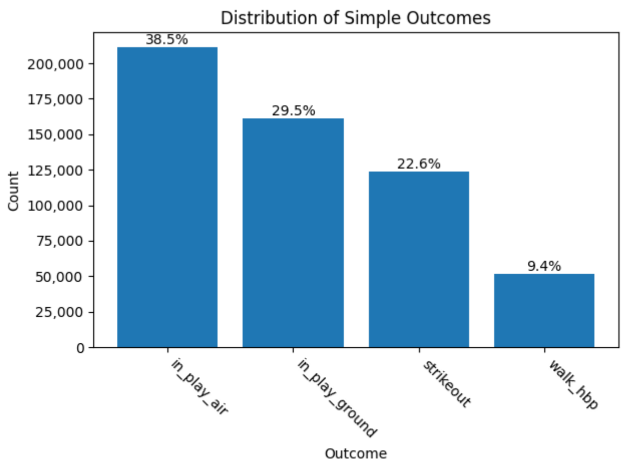


Figure 2: Model Data Simple Outcome Distribution

to prevent any future matchups from influencing the predicted result of past matchups. A 80/20 split was used for training and validation. For testing, edges from the 2024 season that represented the first time the specific pitcher and batter faced each other were used.

3.2 Model

The model for this project builds off of existing PyTorch Geometric and leverages recent advances in graph neural networks. At its core, the architecture is based on GATv2Conv, an improved version of the original Graph Attention Network (GAT). A basic GAT leverages attention mechanisms to learn the importance of neighboring nodes in a graph, enabling more flexible and context-aware node representations. In the traditional GAT PyTorch model (GATConv), a linear transformation is applied to the features first then it computes the attention scores between the transformed features [10]. However in improved PyTorch model (GATv2Conv) the attention is computed on the raw features directly (before any linear transformations) which allows the model to learn attention in a more flexible and dynamic way [11].

Additionally, the HeteroConv wrapper was used to address the heterogeneous graph structure [12]. It allows for different layers to be applied to different edge types in the graph. HeteroConv address the asymmetric nature of pitcher-batter relationship. The graph contains two directed edge types: ("pitcher", "faces", "batter") and ("batter", "rev_faces", "pitcher"). This dual-edge structure allows the model to treat the directionality of matchups explicitly since facing a batter is not the same as being faced by a pitcher.

The graph encoder consists of two stacked HeteroConv layers, each applying GATv2Conv to both edge types. The GATv2Conv layers handle message passing across the entire graph to update node embeddings by learning which neighbors provide the most valuable information. The first layer uses concatenation but the second layer does not. This allows the model to initially expand the feature space to capture diverse aspects of relationships before condensing this information into a more compact, focused representation. The model also uses a multi-head attention mechanism to allow the network to jointly attend to information from different representation subspaces. The outputs from different edge types are aggregated using a sum operation, ensuring that each node's representation is updated using all relevant relationship types. The two-layer architecture enables the model to learn more complex, non-linear patterns in player interactions. The first layer enables each player node to aggregate information from its direction neighbors (e.g., batters a pitcher has faced). The second layer allows for multi-hop information flow which enables a player to indirectly learn from similar opponents (e.g., a pitcher can incorporate knowledge from other pitchers who have faced the same batters).

After message passing, the model uses a custom AttentionEdgeClassifier to make predictions for each pitcher-batter edge. The AttentionEdgeClassifier uses the contextually-enhanced pitcher and batter embeddings from the GATv2Conv layers to make specific predictions about individual pitcher-batter encounters. It transforms the pitcher and batter embeddings through separate linear projections, computes attention weights to determine each player's relative influence on the outcome, and combines this weight representation with edge features before passing through a two-layer multi-layer perceptron to produce final predictions. The attention-based approach allows the model to dynamically focus on the model relevant aspects of each player's characteristics depending on the specific matchup context. Together, the GATv2Conv layers and AttentionEdgeClassifier form a two-stage attention mechanism. GATv2Conv determines how players should incorporate information from their past opponents, while the AttentionEdgeClassifier determines how to weight each player's contribution to the final prediction for a specific matchup.

The second model builds off the first model by incorporating temporal information into the edge features. It uses a TimeEncoding module based on the method of Xu et al. (2020) [13], where scalar timestamps are transformed into high-dimensional vectors via sinusoidal functions. Unlike fixed sinusoidal encodings, the model uses learnable frequency parameters so it is able to adaptively discover periodic patterns in the data. The timestamp is calculated as the number of days from the earliest date in the edge set. Each timestamp is transformed using sine and cosine functions across the learned frequencies, resulting in a 2xd-dimensional embedding that captures both the relative ordering and seasonal variation of matchups.

These temporal encodings are concatenated with the original edge features before passing into to the GATv2Conv layers and the edge classifier. This allows the model to weight recent matchups differently from older ones and detect changes in player performance over time. The edge classifier is slightly modified to accommodate the expanded edge feature dimension.

Together, these models form a powerful, attention-driven framework for predicting the outcome of MLB pitcher-batter matchups, while accounting fro both structural relationships and temporal dynamics.

3.3 Model Evaluation

The performance of the models were evaluated using both classification accuracy and probability calibration metrics.

Classification performance was assessed using precision, recall, F1-score, and accuracy. These metrics provide insight into the model's ability to correctly identify each outcome class. A confusion matrix was also computed and visualized to examine patterns of misclassification and to identify which outcome categories were most frequently confused.

To evaluate the quality of the predicted probabilities, the Brier score and Expected Calibration Error (ECE) were used. The Brier score measures the mean squared difference between predicted probabilities and actual outcomes, while the ECE summarizes the deviation between predicted confidence and observed accuracy across probability bins.

Reliability diagrams were generated to visualize calibration performance. These diagrams compare predicted probabilities to empirical accuracy across bins, with perfectly calibrated predictions lying on the diagonal. Both an overall reliability diagram and one-vs-rest reliability diagrams for individual classes were created to identify class-specific calibration behavior.

To evaluate discriminative performance, one-vs-rest ROC curves were plotted and the corresponding AUC (Area Under the Curve) scores were computed for each class. These curves highlight the model's ability to distinguish between the positive class and all other classes, providing a measure of class-specific separability.

Finally, the distribution of predicted probabilities was plotted for each outcome class. These distributions help reveal whether the model tends to produce well-calibrated, overconfident, or under-confident predictions, and offer additional context for interpreting calibration metrics.

4 RESULTS

4.1 Experimental Setup

The two different models (MLBMatchupPredictor and MLBMatchupPredictorTemporal) were trained for predicting two different matchup outcome variables (the more detailed outcome with eight classes and the simplified outcome with only four classes). Each model employed 128 hidden units and two attention heads. During training, two different loss functions were utilized: Cross Entropy Loss with class weights [14] and Graph Calibration Loss (GCL) [15]. The weighted cross entropy loss helped address class imbalance by emphasizing underrepresented classes. The graph calibration loss (GCL) modifies the standard cross-entropy to improve the calibration of GNNs by assigning higher loss to confident but incorrect predictions and reinforcing confident correct ones. The loss is defined as:

$$\mathcal{L}_{\text{GCL}} = -\sum_{y=1}^{K} \left(1 + \gamma \hat{p}_y\right) \, p_y \log \hat{p}_y$$

$\hat{p}_y$ is the model's predicted probability for class y, p_y is the true distribution (usually one-hot), and γ is a tunable hyperparameter controlling the strength of the calibration effect [15]. By penalizing low-confidence correct predictions more than confident ones, GCL reduces the entropy of the predicted distribution and improves the alignment between predicted probabilities and actual correctness—resulting in a more trustworthy GNN. Both models were trained for 200 epochs using the Adam optimizer with a learning rate of 0.005. For the models trained with the graph calibration loss a gamma value of 0.01 was used.

4.2 Comparing Loss Functions

First, the model without time encodings (MLBMatchupPredictor) was trained on the data to predict the outcome into one of four classes (in_play_air, in_play_ground, strikeout, and walk_hbp) with the two different loss functions (Cross Entropy Loss and graph calibration loss).

4.2.1 Classification Metrics. Table 1 presents the classification performance MLBMatchupPredictor trained using both cross entropy loss and graph calibration loss, evaluated across four outcome classes: in_play_air, in_play_ground, strikeout, and walk_hbp. Graph calibration loss enhances performance on high-frequency outcomes like in_play_air, improving overall accuracy and weighted scores, but at the cost of degraded performance on less frequent or harder-to-predict outcomes. cross entropy loss yields more balanced results across all outcome types.

Table 1: Loss Function Model Accuracy Comparison

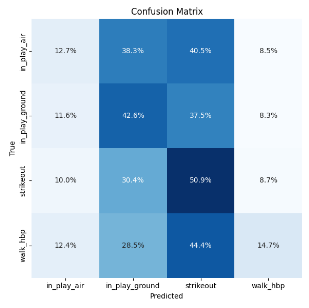
	Cross Entropy Loss				**Graph Calibration Loss**			
Label	Prec	Rec	F1	Sup	Prec	Rec	F1	Sup
in_play_air	0.40	0.13	0.19	20128	0.39	0.81	0.52	20128
in_play_ground	0.33	0.43	0.37	15634	0.38	0.16	0.22	15634
strikeout	0.29	0.51	0.37	13322	0.36	0.14	0.20	13322
walk_hbp	0.16	0.15	0.15	5246	0.21	0.00	0.00	5246
accuracy			0.31	54330			0.38	54300
macro avg	0.30	0.30	0.27	54330	0.33	0.28	0.24	54330
weighted avg	0.33	0.31	0.29	54330	0.36	0.38	0.31	54330

Figure 3 compares the confusion matrices for the MLBMatchupPredictor trained with cross entropy loss (a) and graph calibration loss (b). The model trained with cross entropy loss shows high confusion between similar outcome types with a majority of each true classes were predicted as either in_play_ground or strikeout. The model trained with graph calibration loss heavily favors in_play_air and has very few instances predicted as walk_hpb. While this model achieves higher overall accuracy (0.38 vs. 0.31), it lacks balanced predictive performance across outcome categories.

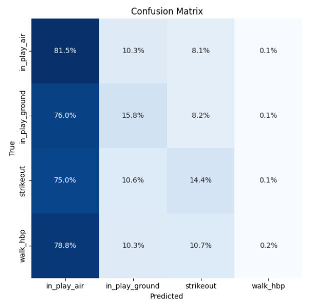
4.2.2 Calibration. Table 2 compares the calibration performance of models trained using cross entropy loss and graph calibration loss. The Brier score is slightly lower for the graph calibration model (0.699 vs. 0.729), indicating improved overall probabilistic accuracy. However, the Expected Calibration Error (ECE) is higher for the graph calibration model (0.097 vs. 0.042), suggesting a greater mismatch between predicted confidence and observed accuracy.

This trade-off is visually depicted in Figure 4, which presents reliability diagrams for both models. Panel (a) shows that the cross entropy model tends to be underconfident across the mid-confidence range but avoids extreme overconfidence. Panel (b), in contrast, illustrates that the graph calibration model more closely tracks the ideal calibration line, although it introduces greater fluctuation across bins, leading to a higher ECE.

The per-class reliability diagrams in Figure 5 further reveal how calibration varies by outcome. For example, the in_play_air and

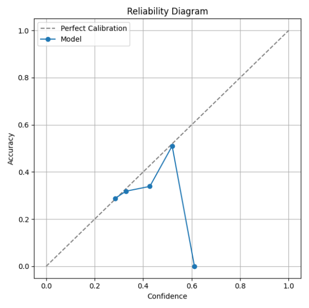


(a) Cross Entropy Loss

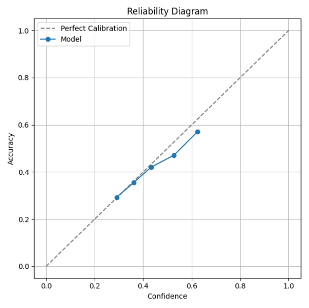


(b) Graph Calibration Loss

Figure 3: Loss Function Model Confusion Matrices Comparison



(a) Cross Entropy Loss



(b) Graph Calibration Loss

Figure 4: Loss Function Model Reliability Diagram Comparison

in_play_ground outcomes are more closely aligned with perfect calibration under the graph calibration model, while classes like strikeout and walk_hbp exhibit erratic bin-level performance. These inconsistencies explain the increase in ECE despite the improvement in Brier score.

To further understand the source of these differences, Figure 6 displays the predicted probability distributions for each class under both losses. In the cross entropy model, predicted probabilities are largely clustered around a common value across all outcomes. This suggests that the model treats classes with similar levels of uncertainty, regardless of their empirical frequency. In contrast, the graph calibration model produces predicted probabilities that are centered more closely around the true frequencies of each class. This alignment implies that the graph calibration loss encourages the model to adjust its predictions in proportion to the underlying distribution of outcomes.

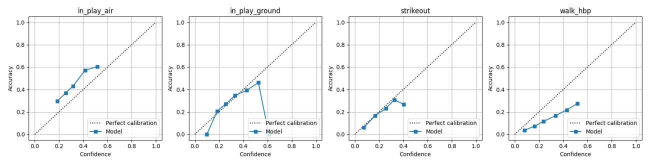
This behavior explains why the graph calibration model achieves a lower Brier score. Its predictions are more probabilistically accurate in aggregate, but it has a higher ECE, as this global accuracy comes with greater per-bin variability. Thus, the predicted probability distributions in Figure 6 provide critical context: while cross entropy promotes uniform confidence, graph calibration better reflects class imbalance, but at a cost to bin-level confidence calibration.

Table 2: Loss Function Model Calibration Comparison

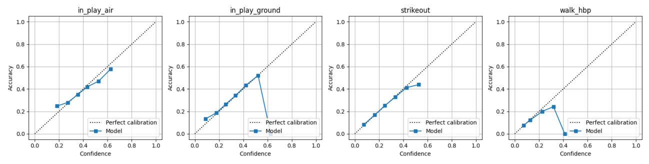
Metric	Cross Entropy Loss	Graph Calibration Loss
Brier Score	0.729	0.699
Expected Calibration Error (ECE)	0.042	0.097

4.2.3 ROC-AUC. To assess the discriminative performance of each model for individual outcome classes, Figure 7 for the four predicted classes: in_play_air, in_play_ground, strikeout, and walk_hbp. These metrics reflect the ability of the model to distinguish each outcome from all others.

For both loss functions, the AUC values are relatively low with performance marginally better than random guessing (AUC = 0.50). When trained with graph calibration loss, the all class-specific AUCs have a slight improvement. These improvements suggest that the graph calibration loss not only enhances probabilistic calibration (as seen in the Brier score and reliability diagrams), but also contributes to better class separability. In particular, aligning predictions with class frequencies (highlighted in the predicted probability distributions) appears to support more distinct decision boundaries, particularly for underrepresented classes.

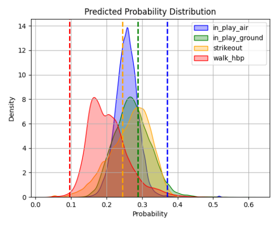


(a) Cross Entropy Loss

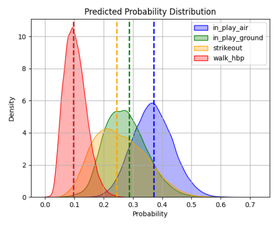


(b) Graph Calibration Loss

Figure 5: Loss Function Model Single Class Reliability Diagram Comparison

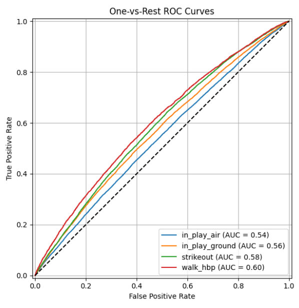


(a) Cross Entropy Loss

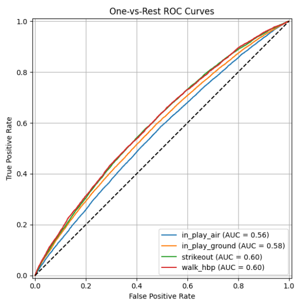


(b) Graph Calibration Loss

Figure 6: Loss Function Model Probability Distribution Comparison



(a) Cross Entropy Loss



(b) Graph Calibration Loss

Figure 7: Loss Function Model Single Class ROC-AUC Comparison

4.3 Four Classes Outcome Variable

Both models (MLBMatchupPredictor and MLBMatchupPredictorTemporal) were run on the graph data to predict the outcome into one of four classes (in_play_air, in_play_ground, strikeout, and walk_hbp). The graph calibration loss was used during training.

4.3.1 Classification Metrics. Table 3 presents the classification performance of two models, MLBMatchupPredictor and MLBMatchupPredictorTemporal, evaluated across four outcome classes: in_play_air, in_play_ground, strikeout, and walk_hbp. There is a high degree of randomness in individual at-bat outcomes which makes strict accuracy-based evaluation difficult. However, there are a few key notes from accuracy metrics. The precision scores are similar across most classes except for the walk_hbp class where the temporal model does not correctly identify any walks or hit-by-pitches. For both models the in_play_air class has the highest recall suggesting the a better ability to correctly identify the outcome. Additionally MLBMatchupPredictorTemporal demonstrates a notable improvement in recall for the strikeout class which suggests that the time patterns help when predicting strikeouts. The macro and weighted averages suggest that MLBMatchupPredictor slightly outperforms MLBMatchupPredictorTemporal in terms of overall precision, but both models have comparable accuracy.

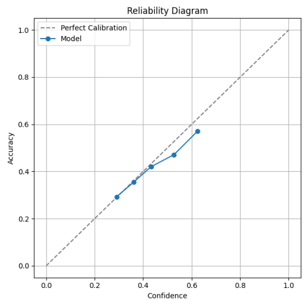
Table 3: Four Outcome Prediction Model Accuracy Comparison

	MLBMatchupPredictor				MLBMatchupPredictorTemporal			
Label	Prec	Rec	F1	Sup	Prec	Rec	F1	Sup
in_play_air	0.39	0.81	0.52	20128	0.39	0.71	0.51	20128
in_play_ground	0.38	0.16	0.22	15634	0.38	0.16	0.23	15634
strikeout	0.36	0.14	0.20	13322	0.35	0.31	0.33	13322
walk_hbp	0.21	0.00	0.00	5246	0.00	0.00	0.00	5246
accuracy			0.38	54330			0.38	54300
macro avg	0.33	0.28	0.24	54330	0.28	0.29	0.27	54330
weighted avg	0.36	0.38	0.31	54330	0.34	0.38	0.33	54330

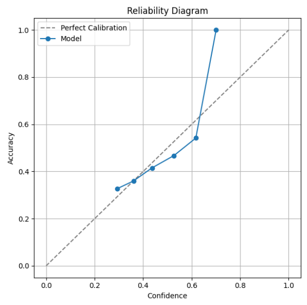
4.3.2 Calibration. Table 3 presents the calibration metrics. The calibration of the two models is comparable with identical Brier scores and almost identical expected calibration errors. Figure 8 shows that the temporal model has slightly higher confidence with similarly calibrated accuracy. There is unstable results with the highest bin have 100% accuracy.

Table 4: Four Outcome Prediction Model Calibration Comparison

Metric	MLBMatchupPredictor	MLBMatchupPredictorTemporal
Brier Score	0.699	0.699
Expected Calibration Error (ECE)	0.096	0.097



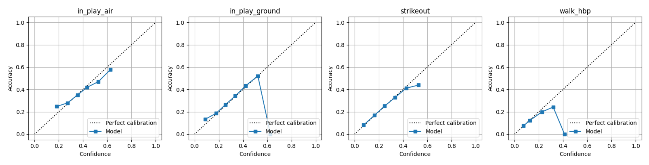
(a) MLBMatchupPredictor



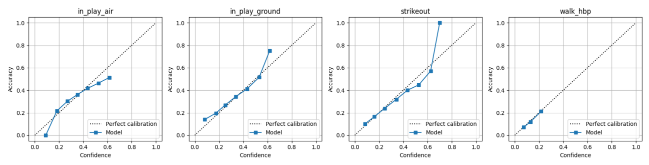
(b) MLBMatchupPredictorTemporal

Figure 8: Four Outcome Prediction Model Reliability Diagram Comparison

The calibration can be further explained by the single class reliability diagrams in Figure 9. Each of the four classes are close to perfect calibration, but with more variability with the lowest and highest confidence bins. The temporal model finds higher confidence for the strikeout class suggesting that temporal trends have a larger impact on strikeouts.



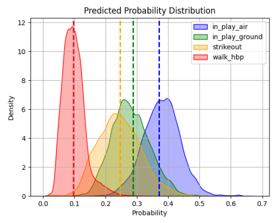
(a) MLBMatchupPredictor



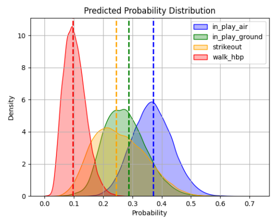
(b) MLBMatchupPredictorTemporal

Figure 9: Four Outcome Prediction Model Single Class Reliability Diagram Comparison

Figure 10 shows that the distribution of the predicted probabilities are very similar for both models.



(a) MLBMatchupPredictor

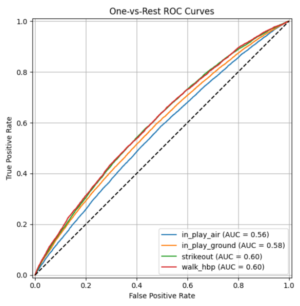


(b) MLBMatchupPredictorTemporal

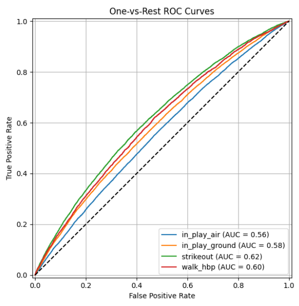
Figure 10: Four Outcome Prediction Model Probability Distribution Comparison

4.3.3 *ROC-AUC.*

Figure 11 shows identical AUC values for all classes except a slight improvement for the strikeout class (0.02) between the two models. Overall, adding the time encodings to the edge attributes produces very similar results to the model without the encodings. The encodings appear to affect the predictions of strikeouts the most suggesting a trends in strikeouts more than other outcome classes.



(a) MLBMatchupPredictor



(b) MLBMatchupPredictorTemporal

Figure 11: Four Outcome Prediction Model Single Class ROC-AUC Comparison

4.4 Eight Classes Outcome Variable

Both models (MLBMatchupPredictor and MLBMatchupPredictorTemporal) were run on the graph data to predict the outcome into one of eight classes (groundball, strikeout, flyball, linedrive, walk, popup, hit-by-pitch, and bunt). The graph calibration loss was used during training.

4.4.1 *Classification Metrics.*

Table 5 presents the classification performance of two models, MLBMatchupPredictor and MLBMatchupPredictorTemporal, evaluated across the eight outcome classes. Compared to the four class outcome there is even greater class imbalance.

Both models perform best on high-frequency classes like groundball and strikeout, but struggle on rare classes such as bunt, hit-by-pitch, and popup, where precision and recall are effectively zero. The temporal model slightly improves recall and F1 for strikeout but shows minimal gains on most other classes. Macro-averaged and weighted F1-scores remain low, indicating difficulty handling the extreme class imbalance inherent to the dataset.

Despite the temporal model's richer edge information, overall classification performance across classes remains largely unchanged, with both models achieving an accuracy of 31%. This highlights the complexity of predicting detailed outcome types in a highly skewed multi-class setting.

Table 5: Eight Outcome Prediction Model Accuracy Comparison

	MLBMatchupPredictor				**MLBMatchupPredictorTemporal**			
Label	Prec	Rec	F1	Sup	Prec	Rec	F1	Sup
bunt	0.00	0.00	0.00	196	0.00	0.00	0.00	196
flyball	0.23	0.01	0.02	9525	0.20	0.01	0.02	9525
groundball	0.31	0.73	0.44	15438	0.32	0.63	0.42	15438
hitbypitch	0.00	0.00	0.00	650	0.00	0.00	0.00	650
linedrive	0.24	0.00	0.01	8118	0.11	0.00	0.00	8118
popup	0.00	0.00	0.00	2485	0.00	0.00	0.00	2485
strikeout	0.31	0.41	0.35	13322	0.30	0.52	0.38	13322
walk	0.45	0.00	0.00	4596	0.19	0.00	0.01	4596
accuracy			0.31	54330			0.31	54300
macro avg	0.19	0.14	0.10	54330	0.14	0.15	0.10	54330
weighted avg	0.28	0.31	0.22	54330	0.23	0.31	0.22	54330

4.4.2 *Calibration.*

Table 6 compares the probabilistic calibration performance of the two models using Brier score and Expected Calibration Error (ECE).

Both models show similar Brier scores and ECE values, with the temporal model performing slightly better. The modest reduction in ECE from 0.160 to 0.159 and Brier score from 0.791 to 0.789 suggests a marginal improvement in probabilistic calibration when incorporating temporal information.

Although the gain is small, the inclusion of time-aware features helps produce slightly better-calibrated predictions, which is valuable in decision-making scenarios that rely on probability estimates rather than hard classifications.

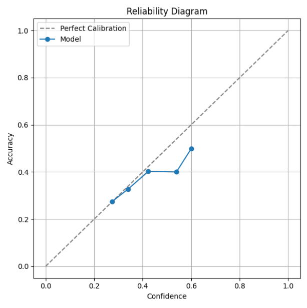
Additionally, the reliability diagrams in Figure 12 shows general over-confidence in the predictions and relatively low confidence in the predicted class (the highest confidence for both models is approximately 60%).

Table 6: Eight Outcome Prediction Model Calibration Comparison

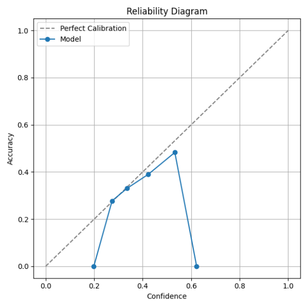
Metric	**MLBMatchupPredictor**	**MLBMatchupPredictorTemporal**
Brier Score	0.791	0.789
Expected Calibration Error (ECE)	0.160	0.159

4.4.3 *ROC-AUC.*

Figure 13 very similar AUC values from both models for each classes. The temporal model has slightly higher values for the walk, pop-up, and hit-by-pitch values with the most improvement coming from the walk class (0.03 increase). The bunt class had the highest AUC at 0.68 for both models, but this may be due to the extremely limited occurrences of bunts in the data. The hit-by-pitch class had the lowest AUC at 0.51 and 0.52, just marginally better than random guessing.

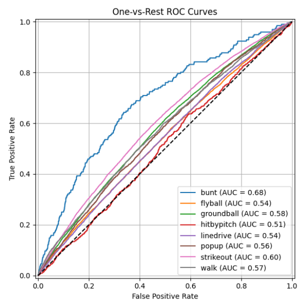


(a) MLBMatchupPredictor

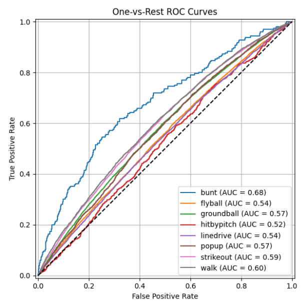


(b) MLBMatchupPredictorTemporal

Figure 12: Eight Outcome Prediction Model Reliability Diagram Comparison



(a) MLBMatchupPredictor



(b) MLBMatchupPredictorTemporal

Figure 13: Eight Outcome Prediction Model Single Class ROC-AUC Comparison

5 CONCLUSION

This project explored the application of Graph Neural Networks (GNNs) to model pitcher-hitter matchups in Major League Baseball. By representing these interactions as a bipartite graph with pitchers and hitters as nodes and at-bats as edges, the models were able to leverage the relational structure of baseball data in ways that traditional statistical and machine learning approaches cannot.

The project compared two GNN architectures, a base model using GATv2Conv layers and an enhanced model incorporating temporal information, across two prediction tasks: classifying at-bat outcomes into four broad categories and eight more specific categories. Additionally, impact of different loss functions (cross entropy loss and graph calibration loss) on model performance and probability calibration were evaluated.

The graph calibration loss resulted in better calibrated models with lower Brier scores. Both models produced reasonably well-calibrated probabilities, particularly for high-frequency outcomes. The reliability diagrams showed that predicted probabilities generally tracked observed frequencies, though with some overconfidence in certain probability ranges. However, there is still improvements to be made in the calibration since the best Brier Score was only 0.699. Additionally, for most of the models there is also relativity low confidence for the predicted classes with the highest confidence being approximately 0.75.

Adding temporal encodings to edge features produced marginal improvements, particularly for strikeout predictions, which saw increases in both recall and F1-score in the four-class model. This suggests that recent performance trends may be more predictive for certain outcome types than others.

While overall predictive accuracy remains relatively modest (38% for four classes, 31% for eight classes), this reflects the inherently high variability in baseball outcomes. The models demonstrated better-than-random class separation, with AUC scores ranging from 0.51 to 0.68 across different outcome classes.

Both models struggled with rare outcome classes (e.g., bunts, hit-by-pitches), highlighting the challenges of modeling highly imbalanced baseball data. Even with graph-based approaches that can leverage similar matchups, predicting uncommon events remains difficult.

Despite the novel application of GNNs to baseball matchups, several limitations suggest directions for future research. First, would be a focus on feature engineering by expanding the node and edge features to better the embeddings used in prediction. Additionally, while the temporal model incorporated time encodings, more sophisticated temporal graph architectures could better capture how player relationships evolve throughout a season. Finally, methods can be developed to interpret what the GNN models learn about player similarities and matchup dynamics to provide valuable insights for baseball analysts and team decision-makers.

This project demonstrates that graph neural networks offer a promising approach for modeling the complex interdependencies in baseball matchups. While predictive accuracy remains bounded by the inherent randomness in baseball, the GNN framework effectively captures relational information between players, providing probabilistic predictions that reflect the true distribution of outcomes. As both GNN architectures and baseball data collection continue to advance, these models have potential to become valuable tools for baseball analytics and decision-making.

REFERENCES

[1] K. Koseler and M. Stephan, “Machine learning applications in baseball: A systematic literature review,” *Applied Artificial Intelligence*, vol. 31, no. 9–10, pp. 745–763, Nov. 2017.

[2] R. Tracy, H. Xia, A. Rasla, Y.-F. Wang, and A. Singh, “Graph encoding and neural network approaches for volleyball analytics: From game outcome to individual play predictions,” 2023.

[3] P. Xenopoulos and C. Silva, “Graph neural networks to predict sports outcomes,” in *2021 IEEE International Conference on Big Data (Big Data)*. Orlando, FL, USA: IEEE, Dec. 2021, pp. 1757–1763.

[4] J. Silver. (2020) Singlearity: Using a neural network to predict the outcome of plate appearances. Accessed: Mar. 10, 2025. [Online]. Available: https://www.baseballprospectus.com/news/article/59993/singlearity-using-a-neural-network-to-predict-the-outcome-of-plate-appearances/

[5] J. Kim, T. Kim, S. Kim, and C. D. Yoo, “Edge-labeling graph neural network for few-shot learning,” in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Long Beach, CA, USA: IEEE, Jun. 2019, pp. 11–20.

[6] E. Rossi, B. Chamberlain, F. Frasca, D. Eynard, F. Monti, and M. Bronstein, “Temporal graph networks for deep learning on dynamic graphs,” 2020.

[7] MLB Advanced Media, “Custom Leaderboard: Pitchers (2015–2024),” https://baseballsavant.mlb.com/leaderboard/custom?year=2024%2C2023%2C2022%2C2021%2C2020%2C2019%2C2018%2C2017%2C2016%2C2015&type=pitcher&filter=&min=10&selections=player_age%2Ck_percent%2Cbb_percent%2Cbatting_avg%2Cslg_percent%2Con_base_percent%2Cisolated_power%2Carm_angle%2Cn_ff_formatted%2Cn_sl_formatted%2Cn_ch_formatted%2Cn_cu_formatted%2Cn_si_formatted%2Cn_fc_formatted%2Cn_fs_formatted%2Cn_kn_formatted%2Cn_st_formatted%2Cn_sv_formatted%2Cn_fo_formatted%2Cn_sc_formatted%2Cn_fastball_formatted%2Cfastball_avg_speed&chart=false&x=player_age&y=player_age&r=no&chartType=beeswarm&sort=xwoba&sortDir=asc, 2025, accessed: Mar. 20, 2025.

[8] ——, “Custom Leaderboard: Batters (2015–2024),” https://baseballsavant.mlb.com/leaderboard/custom?year=2024%2C2023%2C2022%2C2021%2C2020%2C2019%2C2018%2C2017%2C2016%2C2015&type=batter&filter=&min=10&selections=player_age%2Ck_percent%2Cbb_percent%2Cbatting_avg%2Cslg_percent%2Con_base_percent%2Cisolated_power%2Cexit_velocity_avg%2Claunch_angle_avg&chart=false&x=player_age&y=player_age&r=no&chartType=beeswarm&sort=xwoba&sortDir=desc, 2025, accessed: Mar. 20, 2025.

[9] Retrosheet, “Game Sets by Season,” https://www.retrosheet.org/downloads/othercsvs.html, 2025, accessed: Mar. 20, 2025.

[10] PyTorch Geometric, “torch_geometric.nn.conv.gatconv — pytorch geometric documentation,” https://pytorch-geometric.readthedocs.io/en/latest/generated/torch_geometric.nn.conv.GATConv.html, 2025.

[11] ——, “torch_geometric.nn.conv.gatv2conv — pytorch geometric documentation,” https://pytorch-geometric.readthedocs.io/en/latest/generated/torch_geometric.nn.conv.GATv2Conv.html, 2025.

[12] ——, “torch_geometric.nn.conv.heteroconv — pytorch geometric documentation,” https://pytorch-geometric.readthedocs.io/en/latest/generated/torch_geometric.nn.conv.HeteroConv.html, 2025.

[13] D. Xu, C. Ruan, E. Korpeoglu, S. Kumar, and K. Achan, “Inductive representation learning on temporal graphs,” *arXiv preprint arXiv:2002.07962*, 2020. [Online]. Available: https://arxiv.org/abs/2002.07962

[14] PyTorch, “torch.nn.conv.crossentropyloss — pytorch documentation,” https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html, 2025.

[15] M. Wang, H. Yang, and Q. Cheng, “Gcl: Graph calibration loss for trustworthy graph neural network,” in *Proceedings of the 30th ACM International Conference on Multimedia*, ser. MM ’22. New York, NY, USA: Association for Computing Machinery, 2022, p. 988–996. [Online]. Available: https://doi.org/10.1145/3503161.3548423

APPENDIX: CODE AVAILABILITY

The complete source code for this project is available at the following GitHub repository: https://github.com/selah-dean/ms-thesis/tree/main/csds446_project