Heterogeneous Graph Learning for NBA Game Outcome Prediction

Joseph Egan

5/11/2023

Abstract

Predicting the outcome of National Basketball Association (NBA) games is a challenging task that has significant implications for the sports industry and fans alike. This paper presents a comprehensive exploration of the application of both traditional deep learning techniques and advanced graph-based models to predict NBA game outcomes. Utilizing a diverse set of model architectures including Multi-Layer Perceptrons (MLP), Graph Convolutional Networks (GCN), and Heterogeneous Graph Attention Networks (HAN), this project aimed to quantify the potential benefits of incorporating relational data inherent in team and player interactions. The results reveal that while all models exhibited robust predictive capabilities, the graph-based models showed a slight edge over non-graph models. These findings demonstrate the potential of graph-based models in handling complex relational data, offering a promising avenue for future research in sports analytics. This paper contributes to the growing body of work at the intersection of machine learning and sports prediction, providing valuable insights for academics, sports analysts, and enthusiasts.

1 Introduction

Sports, and basketball in particular, provide a rich terrain for machine learning exploration. With the increasing popularity of sports betting, powered by its recent legalization, there is a burgeoning interest in devising systematic methodologies for predicting game outcomes and discerning intricate patterns amongst players and teams. This surge of interest not only brings sports analytics to the forefront but also opens up avenues for unique and complex problems, such as basketball game prediction, to be addressed with innovative methods such as graph neural networks (GNNs).

Basketball, with its layered dynamics of individual player performances, team interactions, and historical game statistics, provides a compelling case for the utilization of machine learning. The combination of discrete events and continuous performance metrics makes it a particularly challenging and appealing problem. The complexity of relationships between players and teams, both within and across games, gives rise to an intricate data structure that cannot be effectively navigated using traditional deep learning techniques.

This is where graphs and, more specifically, graph deep learning, come into play. Graphs, ubiquitous data structures that signify systems where entities are interlinked through various dependencies or relations, offer a powerful means of representing complex data. Nodes connected by edges in a graph allow us to take into account not only individual features but also the network structure. This dual consideration of features and structure can lead to more insightful analyses.

Graph Neural Networks (GNNs) have emerged as an effective tool for addressing graph learning problems. By incorporating the structural information of a graph into learning algorithms, GNNs allow us to go beyond

simple node features and to tap into the rich, relational information that graphs offer. This approach has proven successful in a wide range of applications, leading to the development of a multitude of architectures and models to handle various graph types and problems.

In this project, I leverage the capabilities of GNNs to predict the outcome of NBA games, designating them as either a win or a loss for the home team. The dataset used comprises NBA game, team, and player statistics spanning seven seasons. By structuring this data into a heterogeneous graph, I aim to capture the multifaceted nature of basketball and demonstrate how graph deep learning can potentially enhance our predictive abilities in this domain.

The contributions of this project are as follows:

- The creation of a new basketball graph dataset, which is easily expandable to encompass additional seasons
- Evidence to suggest that the graph structure improves the predictive power of deep learning models for this learning problem, encouraging further research into these methods

1.1 Graph Formulation

The input data for the GNN models are heterogeneous graphs, with one graph representing one game consisting of a matchup between two teams and their players. A heterogeneous graph is a graph with different types of nodes and edges, each of which can have a different dimension of features and level of interactions with other nodes or edges. In this project there were two node types:

- **Team Nodes**: Two team nodes per game, one home team, one away team. Each team node has its own set of features consisting of team statistics up to that point in the season (statistics on a per game basis).
- **Player Nodes**: One node for each starting player in the game. Each player node has features consisting of individual stats up to that point in the season, also on a per game basis.

Additionally, there are 3 edge types:

- **Team-Team Edges**: One edge per graph connects the two teams, and contains match-up statistics from previous games during that season.
- **Opposing Player Edges**: Each player of the same position type is connected to any opposing team players of the same position. This allows the network to consider the importance of player node features when those players go head to head.
- **Player-Team edges**: Each player is connected to his own team's node, allowing the relationship between the quality of the player and his team to be considered.

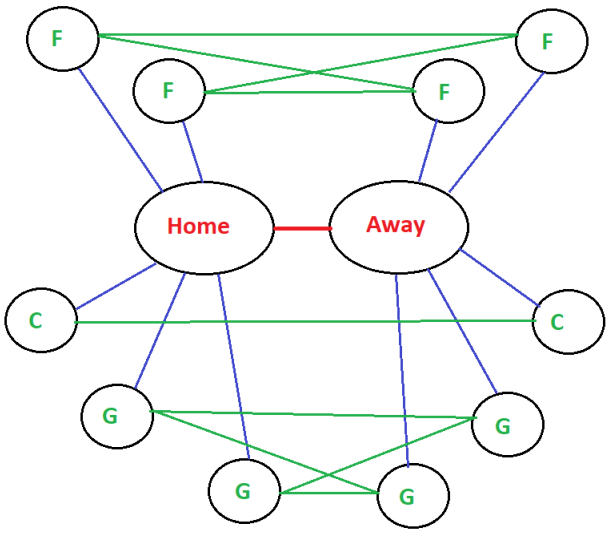


Figure 1. Color-coded depiction of a single game graph with two node types and three edge types. Green letters represent position ϵ {Forward (F), Center (C), Guard (G)}

This formulation of a sports prediction learning problem for the NBA has not yet been addressed in the literature. One unpublished researcher has sampled a wide variety of GNN architectures' ability to predict NBA game point differentials, but with limited features and not using networks that allow for heterogeneous graphs. Therefore this is the first exploration of whether this structure is advantageous to solving this learning problem.

1.2 Dataset and Feature Engineering

The data for this project came from the nba_api, a python client library for accessing current and historical data related to the NBA. It provides easy-to-use interfaces for retrieving player, team, game, and statistical data, among other types of data, directly from the NBA's official API. The data pulled from this API are single game statistics for players and teams, along with some personal data on players (although the quality of the latter data is not great). In order to create usable features for each team, player, and team-team edge, for each game in each season, single-game statistics for teams and players during the same season as the game were averaged up until, but not including that game. This resulted in per-game statistics that gave a good measure of a team or player's quality up to that point in the season. Our features were therefore all numeric except for the player's position which was a one-hot encoded categorical, and a IS_HOME_TEAM indicator for the team nodes:

All as Avg %s

- **Team Nodes**: Field Goals Made, Attempted; 3-Pointers Made, Attempted; Free-throws Made, Attempted; Offensive, Defensive, Total Rebounds; Assists; Steals; Blocks; Turnovers; Fouls; Points
  - Not averaged but included: Winning Percentage; Winning Percentage last 10 games; Home team indicator
- **Player Nodes**: Minutes; Field Goals Made, Attempted; 3-Pointers Made, Attempted; Free-throws

Made, Attempted; Offensive, Defensive, Total Rebounds; Assists; Steals; Blocks; Turnovers; Fouls; Points; Plus/Minus; Possessions; and advanced metrics such as Offensive/Defensive/Net rating; True-Shooting percent; Usage; Player-Impact-Estimate

- **Team-Team Edge**: The Team Node features were used, but the away team's features were subtracted from the home team's features. Additionally, the same features were generated, but only considering previous match-ups between those teams during that season, and a similar subtraction was done and the result concatenated to the other team-team edge features.

Each graph contains a binary label, where if the home team won, the label is 1, and if the home team lost, the label is 0. Each feature is then Min-Max scaled using sklearn's MinMaxScaler, such that all features across node and edge types are $\epsilon[0, 1]$. Finally, the first 10 games for each team during each season were dropped to give players and teams time to 'warm up'. See Appendix A for a full description of each feature.

2 Related Work

2.1 Heterogeneous Graph Analysis with GNNs

A significant development in analyzing heterogeneous graphs using GNNs is the Heterogeneous Graph Attention Network (HAN) proposed by Shi et al. [1]. The HAN model captures semantic-level and node-level information in heterogeneous graphs by employing a two-level attention mechanism. The first level, node-level attention, focuses on the features of individual nodes within each meta-path-based neighborhood. The second level, semantic-level attention, aggregates information from different meta-path-based neighborhoods. HAN has demonstrated its effectiveness in various tasks, such as node classification and link prediction, on heterogeneous graphs. This work provides valuable insights into leveraging GNNs for heterogeneous graph analysis, which can be applied to the NBA game prediction problem.

2.2 Predicting NBA Game Outcomes

Several studies have attempted to predict NBA game outcomes using various machine learning approaches. One such study by Loeffelholz et al. [2] employed a variety of neural network related architectures including feed-forward, radial basis, probabilistic and generalized regression neural networks to predict game outcomes using just 620 games. Their results showed that neural networks outperformed human experts by almost 6 percent in terms of prediction accuracy.

Additionally, Cheng et al. [3] in their paper "Predicting the Outcome of NBA Playoffs Based on the Maximum Entropy Principle," presented a novel approach to predicting NBA game results by formulating it as a classification problem. They develop an NBA Maximum Entropy (NBAME) model that fit discrete statistics for NBA games. The model demonstrated a 74.4% accuracy in predicting the winning team in playoff games, outperforming other classical machine learning algorithms with a maximum prediction accuracy of 70.6% in their experiments. Playoff games are considered to be easier to predict than regular season games due to their lower upset-rate (22% vs 32% in the regular season) [4], and an unpublished study on NBA game prediction surveyed regular season game predicting models and found accuracy between 66-72% to be the upper bound [5].

While these studies did not use GNNs specifically, they demonstrated the potential of using advanced machine learning models for predicting NBA game outcomes, paving the way for this investigation of GNNs tailored

for NBA game prediction. This project focused on regular season games due to the availability and quality of data for the regular season, with the side benefit of being a tougher challenge.

3 Background

In this section, I provide a brief introduction to the Heterogeneous Graph Attention Network (HAN) architecture and its underlying mathematics, highlighting the differences between HAN and the Graph Attention Network (GAT) architecture which we studied in class.

3.1 Graph Attention Network (GAT)

The GAT architecture, proposed by Veličković et al. [6], is designed to work with homogeneous graphs. GAT uses an attention mechanism to compute the weights of edges between nodes, allowing it to learn and incorporate the importance of neighboring nodes. The attention coefficients for an edge between nodes i and j are computed as follows:

$$\alpha_{ij} = softmax_j(LeakyReLU(a^T[Wh_i||Wh_j]))$$

where $\boldsymbol{h}_i$ and $\boldsymbol{h}_j$ are the feature vectors of nodes i and j, respectively, $\boldsymbol{W}$ is a shared weight matrix, $\boldsymbol{a}$ is an attention vector, and $||$ denotes concatenation. The softmax function is applied to ensure the weights sum up to 1. Finally, the new node features are computed as a weighted sum of the neighbors' features:

$$h_i^{'} = \sum_{j \epsilon \mathcal{N}(i)} \alpha_{ij} W h_j$$

3.2 Heterogeneous Graph Attention Network (HAN)

The HAN architecture, proposed by Shi et al. [1], extends the GAT architecture to handle heterogeneous graphs with different node and edge types. HAN employs a two-level attention mechanism: node-level attention and semantic-level attention.

3.2.1 Node-Level Attention

For each node type, node-level attention calculates the importance of neighboring nodes within the same meta-path-based neighborhood. Given a meta-path instance $P_r = (v_1, v_2, \ldots, v_m)$, the attention mechanism computes the edge weight between nodes v_i and v_j:

$$\alpha_{ij}^r = softmax_j(LeakyReLU(a_r^T[W_r h_{v_i}||W_r h_{v_j}]))$$

where $\boldsymbol{h}_{v_i}$ and $\boldsymbol{h}_{v_j}$ are the feature vectors of nodes v_i and v_j, respectively, and $\boldsymbol{W}_r$ and $\boldsymbol{a}_r$ are weight matrices specific to the meta-path r. The output features for node v_i are computed as a weighted sum of its neighbors' features:

$$h_i^{'r} = \sum_{j \epsilon \mathcal{N}_r(v_i)} \alpha_{ij}^r W_r h_{v_j}$$

3.2.2 Semantic-Level Attention

Semantic-level attention aggregates information from different meta-path-based neighborhoods. For each node, it computes the importance of different meta-paths. Given a set of meta-paths R, the attention coefficients for a meta-path r and node v_i are calculated as:

$$\beta_i^r = softmax_r(LeakyReLU(qTtanh(W_q h_{v_i}^{'r})))$$

where $\boldsymbol{q}$ and $\boldsymbol{W}_q$ are the shared attention vector and weight matrix, respectively, and $\boldsymbol{h'}^r_{v_i}$ is the node feature for node v_i generated from the node-level attention. The softmax function is applied to normalize the attention coefficients across different meta-paths. The final node representation is computed as a weighted sum of the node features generated from different meta-paths:

$$h_{v_i}^{''} = \sum_{r \epsilon R} \beta_i^r h_{v_i}^{'r}$$

In summary, HAN first computes attention coefficients for node pairs within meta-path-based neighborhoods, and then aggregates information across different meta-path-based neighborhoods using semantic-level attention. This design enables HAN to effectively capture the complex relationships present in heterogeneous graphs, making it a suitable choice for this NBA game prediction problem.

4 Methodology

4.1 Model Types

While GNNs are capable of capturing complex graph structures and relationships, it's important to assess whether these capabilities actually contribute to performance improvement. Therefore, in order to make a fair comparison between the graph structure and regular neural network models, I evaluated non-GNN models as well. By comparing GNNs to traditional models like MLPs, we can evaluate the degree to which the additional complexity of GNNs is warranted. If a simple MLP model can perform comparably to a GNN on the same task, it may suggest that the graph structure is not as informative for the prediction task as hypothesized. Conversely, if GNNs outperform MLPs, it suggests that leveraging the graph structure provides meaningful benefits. Therefore, evaluating non-GNN models serves as a baseline comparison to assess the true value of incorporating graph-based learning in our approach.

Therefore, four model types were tested on this dataset:

- **Team-Team** feed-forward neural network, which utilized only team node features and team-team edge features
- **Team-Team-Player** feed-forward neural network, which utilized all available features including the players'
- **HeteroConv** graph neural network, a simpler graph neural network which allows for message passing between nodes of the same type, in this case a standard graph convolutional network (GCN) layer was used for both team and player nodes, with pooling and a feed-forward neural network after message passing
- **HANConv** graph neural network, as described earlier, allowing for attention weights to be calculated between nodes of the same and different types across all edge types, also utilizing pooling and a feed-forward neural network after message passing

4.2 Architecture Decisions

Handling Edge Features: One of the most critical decisions in designing the models was determining how best to handle the edge features. In this case, given that there is only one edge with edge features, it was impossible to message pass between similar edges. Further, calculating an edge weight using the features of

the edge was found to potentially discard a significant amount of information stored in the feature vector. After careful deliberation, I decided to concatenate this feature vector to the other features before the final linear layer. This approach ensured that the model retained as much of the edge feature information as possible, thereby maximizing the potential predictive power of these features.

Pooling: Pooling is a critical step in graph classification models, as it allows us to effectively aggregate information from multiple nodes into a form suitable for prediction. In this context, I faced two main decisions regarding pooling: the type of pooling to use and what exactly to pool. After evaluating various options, including SAG pooling and mean pooling, I found mean pooling to be just as effective as SAG pooling, but with fewer parameters. This resulted in a more efficient and streamlined model. The second decision pertained to how the model should pool player features. I found that pooling all players from both teams together led to a loss of predictive power. Instead, I developed split pooling, where the model pools players of the same team together (after message passing in the GNNs), and then concatenates the two separate player feature vectors to the penultimate vector. This approach led to models that performed best in later evaluations, indicating that maintaining some level of separation between player features was beneficial.

Number of Layers: Deciding on the number of layers to use in the models was another important aspect of our architectural decisions. There is often a trade-off between complexity and generalizability in model design - more complex models may capture more intricate relationships in the data, but they also risk overfitting and may not generalize well to unseen data. In this case, I found that the simpler models, which did not project and use activation functions on every feature, were able to generalize much better than their more complex counterparts. This insight reaffirmed the importance of careful model design and the value of simplicity in ensuring models that not only fit the training data well but also perform well on unseen data.

4.3 Training and Fine-Tuning

The training procedures encompassed a series of strategic steps to ensure optimal model performance.

All models utilized the Adam optimizer, recognized for its efficacy in dealing with sparse gradients and noisy data. I incorporated an exponential learning rate decay scheduler, which progressively reduced the learning rate, aiding in achieving a more precise model fit. Additionally, the models utilized weight decay as a form of regularization to control overfitting, which helped with generalization on the validation and test sets.

To further enhance performance, each model was run through an extensive grid search to optimize key hyperparameters. This included the hidden dimensions, which determined the capacity of the model, the dropout rate to control overfitting, and the batch sizes, influencing computational efficiency, gradient estimation, and also generalization. For the HANConv model, I also grid searched for the optimal number of attention heads, a crucial parameter that affects the model's ability to capture different types of relationships in the data.

Lastly, I utilized an early stopping mechanism based on validation accuracy. This approach helped identify the best performing model of each type, preventing unnecessary computational cost and overfitting. By monitoring the validation accuracy during training, the training loop was halted once validation accuracy stopped improving, thereby obtaining the most effective model.

4.4 Objective

For the non-GNN models, the input to the feed-forward network is a single vector containing team and/or player features. For the GNN models, the input is the heterogeneous graph described earlier. The outputs of the GNN layers (either GCN or HAN) are used with pooling to form a single vector, which is then processed through a feed-forward neural network to finally produce a vector of logits: **output** $\epsilon \mathbb{R}^2$. The predicted label is then calculated as $\hat{y} =$ **argmax output**.

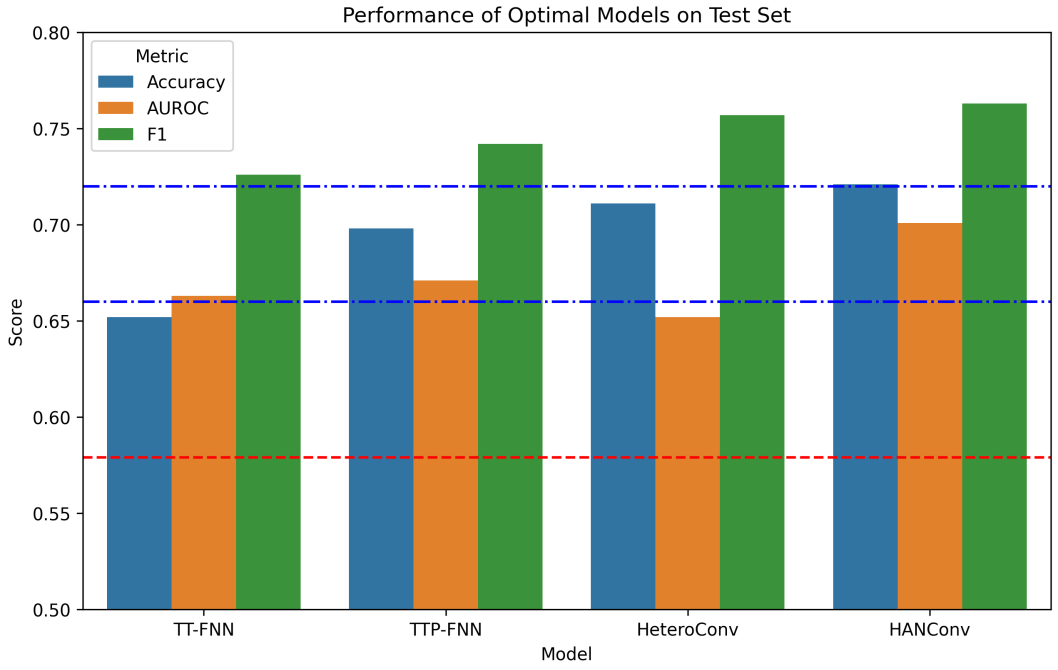
Binary cross entropy loss is then calculated as per below for each graph in each batch, and the average loss is the objective to be minimized.

Loss = (target $*$ log(sigmoid(output[1])) + (1 − target) $*$ log(1-sigmoid(output[1])))

5 Results

After finding the optimal structure for each model, they were evaluated on the test set to achieve these results:

Model	Accuracy	AUROC	F1
TT-FNN	0.652	0.663	0.726
TTP-FNN	0.698	0.671	0.742
HeteroConv	0.711	0.652	0.757
HANConv	0.721	0.701	0.763



Figures 2 & 3. Barplot depicting the results of the table above. The blue horizontal lines indicate the upper-bound range of prediction accuracy from similar studies. The red horizontal line represents the home team winning percentage across the dataset (just under 58%), which is what we would expect to achieve in accuracy through random prediction

6 Discussion

These results highlight the potential of leveraging graph structures in modeling complex interactions within basketball games and support the notion that the inherent connectivity and relationships between teams and players can be harnessed to make more accurate predictions.

The Heterogeneous Graph Attention Network (HANConv) model emerged as the best-performing model (see Appendix B for parameterization), demonstrating the potential of utilizing attention mechanisms in graph-based models for this task. It's worth noting that while the difference in performance metrics between the Team-Team-Player Feedforward Neural Network (TTP-FNN), Heterogeneous Graph Convolutional Network (HeteroConv), and HANConv models was not large, it was significant enough to warrant further investigation into optimizing the graph-based models and dataset.

Interestingly, it is observed that the HeteroConv model achieved higher accuracy and F1 scores than the non-graph models, but a lower AUROC. This suggests that the model may have been better at making correct predictions, but less consistent in ranking the probability of positive classes. This discrepancy underscores the importance of considering multiple performance metrics when evaluating models.

In conclusion, the findings underscore the promising potential of graph-based models in predicting NBA game outcomes. Further studies are warranted to refine these models and explore their applicability to other sports or complex systems where interactions play a pivotal role in determining outcomes.

References

[1] Wang, X., Shi, C., Wang, Y., Philip, S. Y., & Wu, B. (2019). Heterogeneous Graph Attention Network. In *Proceedings of the 2019 World Wide Web Conference (WWW)* (pp. 2022-2032).

[2] Loeffelholz, B., Bednar, E., & Pospisil, A. (2009). Predicting NBA Games Using Neural Networks. *Journal of Prediction Markets*, 3(1), 29-44.

[3] Cheng, G., Zhang, Z., Kyebambe, M. N., & Kimbugwe, N. (2016). Predicting the Outcome of NBA Playoffs Based on the Maximum Entropy Principle. Department of Information Engineering, Xiangtan University. Published: December 16, 2016.

[4] Farmer, T., & Staff, O. S. (2020, March 30). In which sport do underdogs win most often?. Odds Shark. https://www.oddsshark.com/sports-betting/which-sport-do-betting-underdogs-win-most-often

[5] Weiner, J. (2022, July 15). Predicting the outcome of NBA games with Machine Learning. Medium. https://towardsdatascience.com/predicting-the-outcome-of-nba-games-with-machine-learning-a810bb768f20

[6] Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., & Bengio, Y. (2018). Graph Attention Networks. In *International Conference on Learning Representations (ICLR)*.

7 Appendix

7.1 A

Team Features

Feature	Description	Dtype (original -> transformed)	Domain
AVG_FGM	Field Goals Made	Int -> float	[0,1]
AVG_FGA	Field Goals Attempted	Int -> float	[0,1]
AVG_FG_PCT	Field Goal Percent	float -> float	[0,1]
AVG_FG3M	Three-Pointer Made	Int -> float	[0,1]
AVG_FG3A	Three-Pointer Attempted	Int -> float	[0,1]
AVG_FG3_PCT	Three-Pointer Percent	float -> float	[0,1]
AVG_FTM	Free-Throws Made	Int -> float	[0,1]
AVG_FTA	Free-Throws Attempted	Int -> float	[0,1]
AVG_FT_PCT	Free-Throws Percent	float -> float	[0,1]
AVG_OREB	Offensive Rebounds	Int -> float	[0,1]
AVG_DREB	Defensive Rebounds	Int -> float	[0,1]
AVG_REB	Total Rebounds	Int -> float	[0,1]
AVG_AST	Assists	Int -> float	[0,1]
AVG_STL	Steals	Int -> float	[0,1]
AVG_BLK	Blocks	Int -> float	[0,1]
AVG_TOV	Turnovers	Int -> float	[0,1]
AVG_PF	Personal Fouls	Int -> float	[0,1]
AVG_PTS	Points	Int -> float	[0,1]
W_PCT	Winning Percent	float -> float	[0,1]
W_PCT_PREV_10	Winning Percent Last 10 Games	float -> float	[0,1]
IS_HOME_TEAM	Home Team Indicator	Int -> Int	{0, 1}

Player Features

Feature	Description	Dtype (original -> transformed)	Domain
AVG_MIN_x	Minutes Played	time -> float	[0,1]
AVG_FGM	Field Goals Made	Int -> float	[0,1]
AVG_FGA	Field Goals Attempted	Int -> float	[0,1]
AVG_FG_PCT	Field Goal Percent	float -> float	[0,1]
AVG_FG3M	Three-Pointer Made	Int -> float	[0,1]
AVG_FG3A	Three-Pointer Attempted	Int -> float	[0,1]
AVG_FG3_PCT	Three-Pointer Percent	float -> float	[0,1]
AVG_FTM	Free-Throws Made	Int -> float	[0,1]
AVG_FTA	Free-Throws Attempted	Int -> float	[0,1]
AVG_FT_PCT	Free-Throws Percent	float -> float	[0,1]
AVG_OREB	Offensive Rebounds	Int -> float	[0,1]
AVG_DREB	Defensive Rebounds	Int -> float	[0,1]
AVG_REB	Total Rebounds	Int -> float	[0,1]
AVG_AST	Assists	Int -> float	[0,1]
AVG_STL	Steals	Int -> float	[0,1]
AVG_BLK	Blocks	Int -> float	[0,1]
AVG_TO	Turnovers	Int -> float	[0,1]
AVG_PF	Personal Fouls	Int -> float	[0,1]
AVG_PTS	Points	Int -> float	[0,1]
AVG_PLUS_MINUS	Plus-Minus	Int -> float	[0,1]
AVG_E_OFF_RATING	Offensive Efficiency Rating	float -> float	[0,1]
AVG_OFF_RATING	Offensive rating	float -> float	[0,1]
AVG_E_DEF_RATING	Defensive Efficiency Rating	float -> float	[0,1]
AVG_DEF_RATING	Defensive Rating	float -> float	[0,1]
AVG_E_NET_RATING	Net Efficiency Rating	float -> float	[0,1]
AVG_NET_RATING	Net Rating	float -> float	[0,1]
AVG_AST_PCT	Assists per 100 Possessions	float -> float	[0,1]
AVG_AST_TOV	Assist to Turnover Ratio	float -> float	[0,1]
AVG_AST_RATIO	Assist Ratio	float -> float	[0,1]
AVG_OREB_PCT	Offensive Rebounds per 100 Possesssions	float -> float	[0,1]
AVG_DREB_PCT	Defensive Rebounds per 100 Possessions	float -> float	[0,1]
AVG_REB_PCT	Total Rebounds per 100 Possessions	float -> float	[0,1]
AVG_TM_TOV_PCT	Team Turnover per 100 possessions	float -> float	[0,1]
AVG_EFG_PCT	Effective Field Goals Percentage	float -> float	[0,1]
AVG_TS_PCT	True Shooting Percentage	float -> float	[0,1]
AVG_USG_PCT	Usage	float -> float	[0,1]
AVG_E_USG_PCT	Usage Efficiency	float -> float	[0,1]
AVG_E_PACE	Pace Efficiency	float -> float	[0,1]
AVG_PACE	Pace (advanced stat)	float -> float	[0,1]
AVG_PACE_PER40	Pace per 40 Mins	float -> float	[0,1]
AVG_POSS	Possessions	float -> float	[0,1]
AVG_PIE	Player Impact Estimate	float -> float	[0,1]
BIRTHDATE	Birthdate	time -> float	[0,1]
START_POSITION_C	Position Indicator	Int -> Int	{0, 1}
START_POSITION_F	Position Indicator	Int -> Int	{0, 1}
START_POSITION_G	Position Indicator	Int -> Int	{0, 1}

7.2 B

Best Model Hyperparameters:

Hyperparameter	Value
Hidden Dims	24
Attention Heads	4
Dropout Rate	0.2500
Learning Rate	0.0250
Gamma	0.9900
Weight Decay	0.0005
Batch Size	16
GNN Layers	1