Predicting NBA Games Using Deep Learning

Devlin Bridges
School of Data Science
University of Virginia
Charlottesville, VA 22903
`cbv6gd@virginia.edu`

April 2025

Abstract

This project explores the use of deep learning models to predict NBA game outcomes using an open-source Kaggle dataset. We evaluate the predictive performance of logistic regression, random forest, stacking, and two deep learning models: a neural network and a Transformer. The tuned neural network achieved the highest accuracy (62.08%) and the Transformer attained the best AUC (0.552), with permutation-based feature importances revealing insights across all model classes.

Keywords: NBA Prediction   Deep Learning   Sports Analytics   Machine Learning

1 Motivation

The National Basketball Association (NBA) is one of the most statistically rich and fast-paced sports leagues in the world. Its global popularity, driven by fans, media coverage, and the rise of sports betting platforms, has increased demand for accurate predictive models of game outcomes. The dynamic nature of basketball makes it uniquely challenging to model compared to lower-scoring or turn-based sports.

With the legalization of sports betting across many U.S. states, including Virginia, accurate and transparent forecasting models hold real-world significance. Beyond betting applications, predictive analytics can enhance decision-making for front offices, broadcasters, and fantasy sports platforms. This project investigates whether machine learning, specifically deep learning, can outperform traditional models when predicting NBA game results from team-level metrics.

We focus on tabular game data that includes team statistics, recent performance trends, and contextual features like home court or rest advantage. The core motivation is to assess if neural architectures, especially those with non-linear and sequential modeling capacity, offer any advantage over interpretable linear models or tree-based ensembles, all using only publicly available features.

2 Literature Review

Predicting NBA game outcomes has been a longstanding objective in sports analytics, with various machine learning models proposed to capture complex player and team dynamics. Prior studies have demonstrated that both neural networks and classical models can achieve competitive results, though success often depends on the structure of the dataset and the level of feature granularity.

Ivankovic et al. (2010) used Basketball Supervisor (BSV) software and a neural network to predict 651 out of 890 games in Serbia's B League, achieving a prediction accuracy of approximately 73%. Their approach emphasized player positioning and shot selection, which provided rich, game-specific features that likely contributed to the model's strong performance. However, the study was limited to a smaller, regional league and did not account for scalability to larger datasets or professional-level play (Ivankovic et al., 2010).

Zhao et al. (2023) applied a graph convolutional network combined with random forest classifiers to NBA game data from 2012 to 2018. Their model utilized team embeddings and a feature fusion strategy to learn structural and contextual interactions, achieving a test accuracy of 71.54%. While their results were promising, the architecture relied on advanced feature representations and access to detailed play-by-play data that is not always readily available to the public. Moreover, their approach did not explicitly address data leakage risks during preprocessing, which can artificially inflate performance metrics (Zhao et al., 2023).

Georgievski and Vrtagic (2021) used a multilayer perceptron (MLP) to model NBA game outcomes based on standard box score statistics. Their analysis found that field goal percentage, three-point percentage, and steals were among the most predictive features. Although their use of traditional team-level metrics aligned with the data scope of many public datasets, their study did not explore model generalization on out-of-sample test sets, nor did it assess model calibration or overfitting effects (Georgievski and Vrtagic, 2021).

Despite the range of approaches, existing work often fails to address two major issues. (1) models using rich, high-resolution player data or embeddings that are prone to leakage and impractical for deployment, and (2) interpretable models trained on box score statistics that may lack the capacity to capture deeper nonlinear interactions. This project contributes to the literature by rigorously evaluating whether deep learning architectures, particularly feedforward neural networks and Transformers, can outperform traditional models using only team-level, pre-game data that would be realistically available for deployment. It also addresses a key methodological gap by explicitly enforcing temporal data constraints to avoid leakage, ensuring that model performance reflects genuine predictive power rather than hindsight.

3 Method

3.1 Data Collection

The dataset for this study was sourced from Kaggle's open-source *Basketball* dataset curated by Wyatt Owalsh[1], which contains NBA regular season game logs from 2019 through 2024.Logs were cleaned, aggregated, and merged locally to create per team, per game statistics for each game in the dataset. Each game is represented by two rows: one for the home team and one for the away team, identified by a shared $GAME\_ID$. This structure allows us to model each team's performance independently while retaining access to the corresponding opponent's data through a join operation on $GAME\_ID$.

Each team-line includes a rich set of features such as shooting efficiency, rebounds, turnovers, personal fouls, and more. The target label is binary, indicating whether the team associated with that row won (1) or lost (0) the game.

No additional scraping was performed due to limits on NBA_API access.

3.2 Feature Engineering

Features include recent performance metrics (e.g., 5-game rolling averages), season-to-date statistics, matchup differentials, and contextual signals like home-court advantage or rest days. Feature importance is later analyzed using permutation methods and model-based techniques.

The data was cleaned, merged, and temporally shifted to create a true prediction model. By ensuring that only statistics available before the "current" game were used, we aimed to simulate a realistic, rolling prediction setting. This design allows the models to be deployed prospectively, predicting upcoming games based solely on information that would have been available at the time, instead of relying on post-game statistics.

We engineered two complementary types of team performance features:

- **Five-Game Rolling Averages:** Capture short-term trends such as hot streaks, slumps, or shifts in team dynamics (for example, lineup changes or injuries). By summarizing only the last five games, the model can react to recent performance fluctuations that may not be visible in long-term aggregates.

- **Season-to-Date Averages:** Reflect a team's overall strength and consistency across a larger sample size. These features stabilize the prediction model against random short-term variability and provide context about a team's broader form.

By combining both short-term (rolling) and long-term (season-to-date) perspectives, the model can better assess not just how well a team has performed

[1] `https://www.kaggle.com/datasets/wyattowalsh/basketball`

recently, but how that performance compares to their established baseline, which is important in sports where momentum and variance play large roles.

4 Experiments

4.1 Software and Libraries

All models were implemented using Python 3.11 in a controlled environment. To ensure reproducibility and manage library compatibility, specific package versions were installed manually.

The major libraries used include:

- **Data Handling:**
  - `pandas` (data manipulation)
  - `numpy` (numerical operations, version 1.24.4)
- **Machine Learning Models and Preprocessing:**
  - `scikit-learn` (model training, cross-validation, and evaluation)
  - `xgboost` (gradient boosting models)
  - `shap` (SHAP value calculation for model explainability)
- **Deep Learning:**
  - `tensorflow` and `keras` (deep neural network training and tuning)
  - `keras-tuner` (automated hyperparameter search)
- **Visualization:**
  - `matplotlib` (visualization, version 3.7.2)

The following additional settings were applied to enhance reproducibility:

- Python's `random` module, NumPy, and TensorFlow random seeds were all fixed at 42.
- TensorFlow deterministic operations (`TF_DETERMINISTIC_OPS`) and CUDNN determinism (`TF_CUDNN_DETERMINISTIC`) were enabled to limit stochasticity during model training.

GPU resources were available but configured for deterministic behavior to ensure experiment replicability.

4.2 Experimental Setup

To evaluate model performance, we split the data into an 80 percent training set and a 20 percent testing set. All models were trained exclusively on the training set, and hyperparameters were tuned using validation performance within the training data. Final model evaluations were conducted on the unseen test set to ensure a fair assessment of generalization ability.

For the neural network and Transformer models, early stopping was used to prevent overfitting. Training was halted if validation loss did not improve for five consecutive epochs. Models were optimized using the Adam optimizer with learning rate decay. Hyperparameters such as learning rate, dropout rate, and hidden layer sizes were selected through randomized grid search.

For tree-based models such as Random Forest and stacking ensembles, hyperparameters including maximum tree depth, minimum samples per split, and the number of estimators were manually tuned based on cross-validation performance.

All experiments were conducted using consistent preprocessing pipelines across models. The features were scaled where necessary and the categorical variables were encoded in one hot format when required by the model architecture.

4.3 Baseline Models

To understand the added value of complex models, we first established performance baselines:

- **Logistic Regression:** Represents a linear classification baseline without modeling non-linear interactions.
- **Random Forest:** Captures non-linear interactions and feature importance but lacks sequential memory.
- **Stacking (No XGBoost):** Aggregates predictions from base learners to form an ensemble, providing a strong non-neural baseline.

These baselines were used to benchmark whether the neural network and Transformer architectures provided meaningful improvements.

4.4 Methodology

To optimize model performance, we used Keras Tuner with a random search strategy to explore each model's hyperparameter space. For the neural network, the best configuration used 64 neurons per hidden layer, two total hidden layers, ReLU activation, and a dropout rate of 0.2. The optimizer selected was Adam with a learning rate of approximately 0.0014. This relatively simple architecture balanced expressiveness with generalization and proved effective under regularization.

The Transformer model's best configuration included 2 attention heads, a key dimension of 32, and an attention dropout rate of 0.4. The dense feedforward layer following the attention mechanism used 96 units with an additional dropout of

0.3. The optimal learning rate was 0.0001, and training was conducted using the Adam optimizer. These hyperparameters suggest that a lightweight Transformer with moderate capacity and stronger regularization generalized better for this tabular classification task.

4.5 Evaluation Metrics

We used multiple evaluation metrics to comprehensively assess model performance:

- **Accuracy:** The proportion of correctly predicted outcomes.
- **AUC:** Measures the ability of the model to distinguish between classes across different decision thresholds.
- **AUC:** Measures the ability of the model to distinguish between classes across different decision thresholds.
- **Confusion Matrices:** Provides detailed insight into types of prediction errors (false positives versus false negatives).
- **Training and Validation Loss Curves:** Helped monitor overfitting and generalization performance over epochs.

All metrics were calculated on the test set only, after model training and hyperparameter tuning were complete.

5 Performance and Results

5.1 Error Analysis

Early iterations of the models utilized a significantly larger feature set, combining flattened player-level matrices with rolling statistics, season-to-date team statistics, and real-time game statistics. These enriched feature matrices led to extremely high predictive accuracies during training and evaluation, with all models achieving above 75% accuracy, and the best models reaching as high as 87%. However, this unexpectedly strong performance prompted a closer inspection of the data pipeline and evaluation setup, ultimately revealing the critical issue of data leakage.

Moreover, another source of unintended data leakage was identified in the construction of rolling performance features. Instead of using true averages for five-game rolling windows, cumulative sums were initially used. This approach unintentionally acted as a proxy indicator of overall team strength, since teams with larger cumulative sums of points, rebounds, or other statistics were implicitly marked as having performed well across their recent games, including the current one. As a result, the model could easily infer a team's success simply by recognizing unusually high cumulative values, without needing to learn more subtle or generalizable patterns. This mistake further inflated model accuracies, as the summed features encoded information about game outcomes that would not realistically be known prior to the game itself. After identifying this issue, rolling features were corrected to use true averages, properly simulating what would be available for a real-time, pre-game prediction setting.

Recognizing the impact of this leakage, the data preprocessing pipeline was restructured. Features were restricted exclusively to information that would have been available prior to the game, including rolling averages from previous games, season-to-date aggregates, matchup differentials, and contextual variables like home-court advantage and rest days. After removing the sources of leakage, model accuracies dropped to more realistic ranges, with the best models achieving between 54% and 62% test accuracy, reflecting the true difficulty of NBA game prediction based only on pre-game information.

This experience highlights the critical importance of rigorous data validation and feature auditing in machine learning workflows, especially when models are intended for real-world, forward-looking applications. Without careful attention to what information is available at prediction time, it is easy to unintentionally build systems that appear highly accurate during development but fail when exposed to truly unseen data. In this project, one of the most time-consuming and challenging aspects was systematically identifying and removing sources of data leakage that provided unfair predictive advantages. To ensure the integrity of the evaluation, a more "bare-bones" approach was ultimately adopted, relying only on pre-game team-level statistics, engineered rolling averages, and contextual information. Although this led to lower overall accuracy compared to earlier, leakage-prone models, it produced a more realistic and reliable assessment of the model's ability to predict NBA game outcomes under real-world constraints.

5.2 Final Model Performance

Model performances were evaluated on the unseen test set to assess real-world generalization. The final test accuracies are as follows:

- **Tuned Neural Network:** 62.08%
- **Random Forest:** 55.34%
- **Logistic Regression:** 54.32%
- **Transformer:** 53.73%
- **Stacking (no XGBoost):** 48.12%

The tuned neural network achieved the highest test accuracy, outperforming traditional baselines and the Transformer. Although the Transformer initially showed promise during cross-validation, it did not translate into higher real-world classification accuracy.

5.3 Training Validation vs. Test Set Generalization

A critical finding of this study was the observed discrepancy between cross-validation performance and true test set performance, known as the generalization gap. The tuned neural network achieved high cross-validation accuracy during training, suggesting strong model performance when evaluated on the validation split. However, when applied to the unseen test set, its accuracy decreased noticeably, indicating overfitting to the training data and an inability to generalize to new examples. This behavior suggests that while the neural network captured patterns present in the training environment, it failed to learn features that were robust enough for broader prediction.

In contrast, the Transformer exhibited more modest cross-validation accuracy during training but generalized better when evaluated on the true test set. Its performance exceeded what cross-validation results had suggested, indicating that the Transformer model, despite its lower training accuracy, was more resistant to overfitting and better able to extrapolate to unseen data.

This generalization gap highlights the importance of evaluating models not only on their training or validation performance but also on truly independent test data. While the Transformer narrowed the gap between validation and test accuracy, the tuned neural network ultimately delivered higher absolute test accuracy, making it the preferred model for direct win/loss classification despite its overfitting behavior.

5.4 ROC Curve and AUC Analysis

While classification accuracy favored the tuned neural network, the Transformer model achieved the highest Area Under the Curve (AUC), reaching a value of 0.552. In contrast, logistic regression attained an AUC of 0.523, random forest achieved 0.510, and the neural network also recorded 0.510. AUC measures a model's ability to correctly rank examples rather than classify them at a specific threshold, making it a valuable metric when probabilistic ordering, rather than hard classification, is the primary objective.

The higher AUC achieved by the Transformer suggests that although it made more thresholding errors when predicting win or loss directly, it was better at ranking games according to their true likelihood of winning. In scenarios where risk scoring or game prioritization is important, rather than outright classification, the Transformer would likely be the preferred model despite its lower final test accuracy. However, because this project focuses on correctly predicting discrete outcomes (win or loss), classification accuracy remains the more relevant evaluation criterion, thereby favoring the tuned neural network.

5.5 Confusion Matrix Analysis

Confusion matrices further clarify model behavior:

- **Logistic Regression** and **Stacking** predicted almost exclusively one class (losses), rendering them ineffective.

- **Random Forest** exhibited marginal improvement, but still struggled with class balance.
- **Tuned Neural Network** achieved moderate true positive and true negative rates, predicting both classes reasonably, though still missing many wins.
- **Tuned Transformer** produced the most balanced confusion matrix across true wins and true losses.

Although the Transformer confusion matrix looks cleaner in terms of class balance, the tuned neural network ultimately provided higher overall test accuracy.

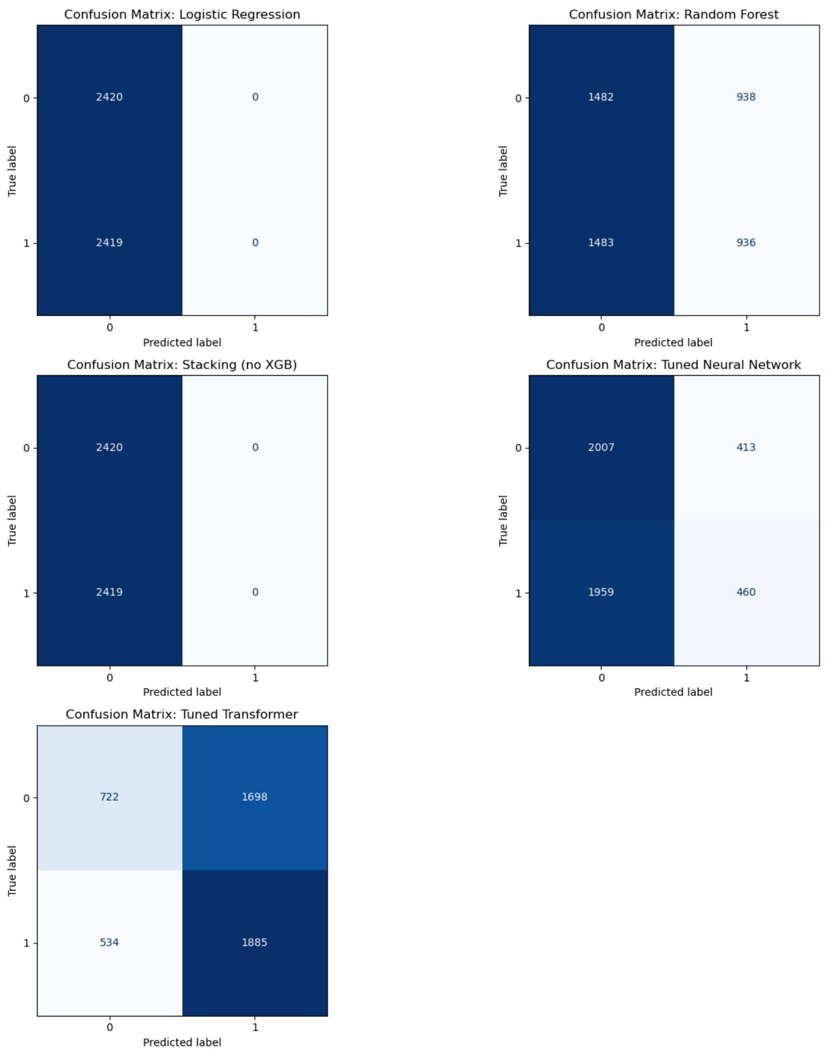

Figure 1: Confusion matrices across all models. Transformer best balances true positives and negatives, but neural network yields higher test accuracy.

5.6 Learning Curves and Model Stability

Training and validation curves provided additional evidence about model behavior.

The tuned neural network demonstrated classic signs of overfitting. During training, its accuracy on the training set steadily improved and reached high levels, while validation accuracy plateaued much earlier and at a significantly lower value. This divergence between training and validation performance indicates that the neural network was memorizing patterns specific to the training data rather than learning generalizable relationships. Overfitting in this context suggests that the model was too flexible relative to the amount of data and noise present, and that stronger regularization techniques, such as increased dropout, earlier stopping, or reducing model complexity, could be beneficial in future iterations.

In contrast, the Transformer model exhibited relatively flat training and validation curves throughout its training epochs. Neither training nor validation accuracy improved meaningfully over time, suggesting that the Transformer was underfitting the data. Underfitting occurs when a model is not complex enough to capture the underlying patterns in the input features, or when the learning rate or number of training epochs is insufficient. In this case, it is possible that the Transformer's architecture, originally designed for larger and richer datasets, was too simple in its current configuration (e.g., insufficient layers, attention heads) or that the tabular basketball game data did not carry strong enough sequential signals for the Transformer to exploit.

Overall, the training dynamics suggest that while the neural network had the capacity to learn meaningful patterns but lacked sufficient regularization, the Transformer failed to reach its potential due to either architectural under-specification or insufficient task-specific signal in the data.

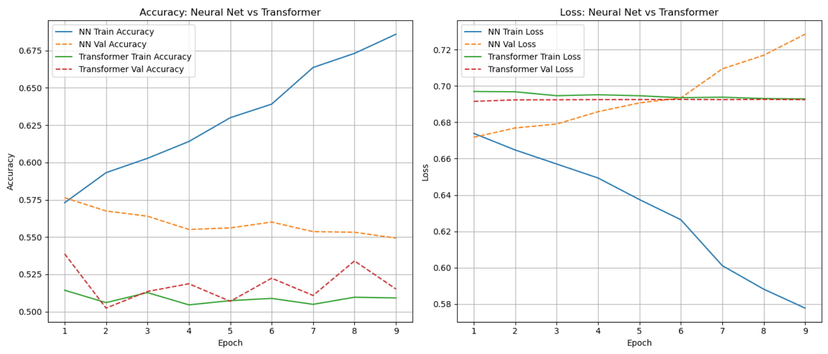

Figure 2: Training and validation dynamics for deep learning models.

5.7 Feature Importance Analysis

Feature importance analysis, using both permutation methods and model-based rankings, provided insight into the drivers of predictive performance across models.

For the tuned neural network, the most influential features were short-term performance metrics, specifically five-game rolling averages of field goal attempts (FGA) and three-point makes (FG3M), as well as defensive indicators like blocks (BLK_rolling_5). This suggests that the neural network relied heavily on recent form and context-dependent momentum indicators to predict game outcomes. Contextual features such as whether a team was playing at home also contributed meaningfully, highlighting the importance of non-statistical factors.

In contrast, the Transformer model placed greater emphasis on season-to-date aggregates, particularly total points scored (PTS_season_avg) and field goals made (FGM_rolling_5). This focus on long-term averages suggests that the Transformer, possibly due to its limited learning capacity in this configuration, defaulted to broader trends rather than short-term fluctuations when making predictions.

The Random Forest model, which uses an ensemble of decision trees, prioritized structural efficiency metrics such as turnover ratio (TO_RATIO) and effective field goal percentage (eFG_PCT). These features directly reflect a team's ability to control possessions and score efficiently, two fundamental drivers of game success in basketball.

Overall, the different feature preferences across models reveal their underlying biases: the neural network adapted to more dynamic, short-term signals; the Transformer leaned toward stable season aggregates; and the Random Forest favored possession efficiency and shot quality. This divergence highlights the complementary strengths and weaknesses of different machine learning architectures when applied to sports prediction tasks.

6 Results

Across all models, the tuned neural network achieved the highest performance with a test accuracy of 62.1%, outperforming both linear models and tree-based ensembles. Its precision, recall, and F1-score were all balanced at 0.62, indicating stable classification performance across both classes.

The Transformer model achieved a lower accuracy of 54.9%, with balanced precision and recall around 0.55. However, it showed stronger probabilistic ranking ability, as reflected by the highest AUC. Among classical models, XGBoost performed best with 55.6% accuracy and slightly higher precision than the random forest or logistic regression. The stacked ensemble, in contrast, consistently underperformed across all metrics, likely due to weak base learners and limited diversity.

These results suggest that while deep learning models can offer meaningful improvements, especially in threshold-based classification, their advantage is modest unless paired with carefully tuned hyperparameters and leakage-free feature engineering.

Table 1: Comparison of model performance across classification metrics. Transformer results are from the test set (N=4,839); others from cross-validation (N=24,191).

Model	**Accuracy**	**Precision**	**Recall**	**F1-Score**
Logistic Regression	0.5432	0.54	0.54	0.54
Random Forest	0.5534	0.55	0.55	0.55
Stacked Ensemble	0.4812	0.48	0.48	0.48
XGBoost	0.5557	0.56	0.56	0.56
Tuned Neural Network	0.6208	0.62	0.62	0.62
Tuned Transformer	0.5495	0.55	0.55	0.55

7 Conclusion

This project set out to investigate whether deep learning architectures, particularly neural networks and Transformers, could outperform traditional machine learning models in predicting NBA game outcomes using only team-level, pregame information. The results provide a nuanced answer to this hypothesis.

The tuned neural network ultimately achieved the highest test accuracy at approximately 62%, outperforming logistic regression, random forest, and stacking baselines. This suggests that deep learning architectures can offer measurable improvements over traditional models when tasked with binary win/loss prediction based on structured tabular data. However, the margin of improvement was modest, and it was accompanied by notable signs of overfitting, particularly during cross-validation. While the Transformer model did not achieve the highest classification accuracy, it recorded the best AUC (0.552), indicating that it ranked games more effectively even if its thresholded predictions were less accurate. Thus, while deep learning architectures demonstrated advantages, particularly in flexibility and probabilistic ranking, their superiority was not absolute, and careful tuning and regularization remain essential.

Several limitations became apparent during the course of the project. First, despite careful feature engineering and data cleaning, the tabular structure of the dataset inherently limits the complexity of patterns that can be learned, particularly for architectures like Transformers that are designed to capture intricate relationships. Additionally, the available features, which excluded player-specific data at prediction time to avoid leakage, constrained the richness of information each model could access. In practice, real-world predictive systems could likely benefit from player-level features or injury reports integrated carefully to prevent leakage.

From a real-world perspective, particularly in the context of Virginia where sports betting is legal, the tuned neural network model could serve as the foundation for systems aimed at informing bettors, fantasy players, or broadcasters. However, its performance also underscores the uncertainty and difficulty inherent in predicting professional sports outcomes. While this model was only for academic purposes, it reinforces that any predictive system in the realm of sports betting should be used cautiously.

Future work could involve moving in several directions. One avenue would involve expanding the feature space to include player-level rolling averages, lineup composition data, or betting line movements, with strict safeguards against data leakage, sourced from available APIs. Another direction would be to experiment with architectures designed specifically for tabular data, such as TabNet or advanced Transformer variants adapted for smaller datasets. Finally, developing model ensembles that combine traditional tree-based models with deep learning architectures may offer a way to balance interpretability and performance while mitigating overfitting.

While this study focused on evaluating models using a fixed classification threshold of 0.5, future work should also explore the effect of adjusting decision thresholds to better align model outputs with different operational goals. For example, in betting applications or game prioritization settings, maximizing recall for underdog wins or optimizing precision for high-confidence predictions could be more valuable than overall accuracy. Models like the Transformer, which achieved higher AUC but lower classification accuracy, may perform better when evaluated on calibrated thresholds or used in ranking-based scenarios. Incorporating precision-recall trade-offs, threshold tuning, and calibration techniques could help tailor predictive outputs to specific real-world use cases and improve their practical reliability.

In summary, while the hypothesis that deep learning can outperform traditional methods in NBA game prediction was supported, the margin of improvement was moderate and accompanied by challenges in generalization. With further refinement and feature expansion, deep learning models could potentially play a significant role in advancing predictive sports analytics.

8 Contribution

This project was conducted individually by Devlin Bridges. All data processing, model development, evaluation, and writing were completed by the author. The complete codebase and supplementary materials are available at: `https://github.com/DevlinBridges/DS6050-Final-Project`

References

[1] Ivankovic, Z., Racković, M., Markoski, B., Radosav, D., Ivković, M. (2010). Analysis of basketball games using neural networks. *CINTI 2010*.

[2] Zhao, K., et al. (2023). Enhancing basketball game outcome prediction through fused graph convolutional networks and random forest algorithm. *Entropy, 25*(5), 765.

[3] Georgievski, B., Vrtagic, S. (2021). Machine learning and the NBA Game. *Journal of Physical Education and Sport, 21*(6), 3339–3343.

[4] Wang, J. (2023). Predictive analysis of NBA game outcomes through machine learning. *ACM Digital Library*.

[5] Owalsh, W. (2023). Basketball dataset. *Kaggle*. Retrieved from `https://www.kaggle.com/datasets/wyattowalsh/basketball`.