Final Project Report: March Madness Prediction Network

Sheraz Hassan
Georgia Institute of Technology
Atlanta, USA
shassan74@gatech.edu

Kiran Nazarali
Georgia Institute of Technology
Atlanta, USA
knazarali@gatech.edu

Henry Raman
Georgia Institute of Technology
Atlanta, USA
hraman8@gatech.edu

Sai-Aksharah Sriraman
Georgia Institute of Technology
Atlanta, USA
ssriraman3@gatech.edu

Abstract

March Madness, the annual National Collegiate Athletic Association (NCAA) basketball tournament featuring the top 68 college teams, presents a very hard challenge in sports prediction due to its inherent unpredictability and high-stakes nature. In 2014, Warren Buffett highlighted this challenge by offering a billion dollars to anyone who could perfectly predict the outcomes of all 67 games. Our project aims to introduce a novel predictive model using a deep learning approach to predict the results of March Madness. We looked at leveraging historical game statistics and team performances, including elements like hot and cold streaks, to get to our desired results, which will not only enhance fan engagement but also provide valuable information to coaches and analysts. From the results of our approaches, we found two key insights about how deep learning models make connections for this type of data.

1. Introduction

Predicting all 67 March Madness games accurately has remained a challenge, as the odds of predicting every single one of these results correctly are about 1 in 147 quintillion. The difficulty of this challenge stems from not just the number of games but the inherent unpredictability of the tournament, where upsets are common. Given that US bettors wagered over 2.7 billion dollars on March Madness this year and around 30-40% of the public closely follow the tournament, an improved predictive model could have significant implications. It could enhance fan engagement, inform sports betting, and provide insights to coaches and analysts. Even if a perfect bracket remains improbable, this project seeks to uncover trends and patterns that influence team performance, offering valuable insights into the dynamics of March Madness.

This project proposes a novel approach to predicting March Madness outcomes using a time series Long Short-Term Memory (LSTM) deep learning model. Unlike traditional statistical models, random forests, or past approaches using deep neural networks, our method leverages historical game statistics and team performance throughout the tournament. By subtly incorporating key factors like hot and cold streaks, the model aims to capture the ”human” element that often drives unexpected victories.

2. Related Works

The study of predicting outcomes in the NCAA March Madness basketball tournament is a point of interest for researchers who utilize various statistical and machine learning methods. Shen et al. [4] evaluated support vector machines and two other machine learning models, including random forests and a Bayesian approach,h to forecast the results of two tournament years. Gumm et al. on the other hand used a more statistical approach by creating an ensemble model of regression functions and leveraged the Kaggle Machine Learning March Mania Competition, successfully placing in the top 15 percentile by comparing their techniques against other entries [1]. Further research highlighted the impact of each player on the team performance while showing that teams with previous postseason experience have a significant advantage in terms of victory margins [3]. This reflects on the idea that previous data of performances effect the results and can be a strong predictor of the results. Kim et al. expanded on these concepts by using machine learning methods such as random forests. They explored the importance of non-box score statistics, using metrics like classification accuracy and the area under the curve to assess model performance [2]. Building on this, our approach aims to incorporate LSTM models to track player performance across matches, aiming to optimize predictions and achieve superior outcomes. More details about our dataset and methodology will follow in the next section.

3. Dataset

For the dataset, we required game-by-game box score information for multiple NCAA Division I Men's College Basketball seasons. This data was provided by Bart Torvik [5], which allows the game information from any season to be downloaded to a spreadsheet, providing a dataset with multiple years of information. For every game, each team's major statistical categories are tracked, including points, assists, rebounds, steals, blocks, and field goal percentages. With this data, we put together rolling windows of game history containing 10-12 statistical categories per team. Using the process described later in the report, we created around 60,000 training samples, covering years 2011-2025.

4. Method / Approach

As discussed in the background, most of the current approaches to solving this type of problem have not investigated the time series aspect. In basketball, momentum is an important concept, whether it is during a game or over the course of a few games. A particularly poor showing in one game could lead to a reinvigorated effort in the next game, and the opposite can also be true. A small win streak could continue to snowball into a team's performance peaking at the right time. Thus, our methodology looked at leveraging the potential information in a time series approach.

For a given matchup of Team A vs. Team B, we included relevant statistics from each of Team A's and Team B's previous games. If the current game is labeled in time as T, stats from games T-1, T-2...T-X were used. An example with just points and assists for Team A would be: [Points(T-X), Assists(T-X)...Points(T-2), Assists(T-2), Points(T-1), Assists(T-1)].

We had two separate outputs - one was a simple binary classification with a binary cross-entropy loss, denoting which team is predicted to win the game. The other was a two-element vector predicting the final score of the game, with an MSE loss.

Given the time-series nature of this approach, the main models were two variants of an LSTM and CNN-based deep architecture for our neural network. Feeding it a sequence of games would allow for connections to be made between the games, potentially improving performances of the prediction. We also trained a transformer as an alternative for comparison to see if the time-series architecture provided any benefits. The basis for comparison was the accuracy in terms of correct and incorrect predictions for the winner of each game in the 2025 March Madness tournament, and the results will be discussed later in the report.

4.1. Data Pre-processing

In order to take the raw basketball game data and format it into rolling windows of game history, a custom preprocessing methodology was created. The data from Bart Torvik was loaded and the statistical categories that we want to use as features (in-game statistics and/or seeding and categorical features) were identified. This raw data lacked column headers, thus requiring manual header assignments to ensure the necessary data was recognized correctly by the models.

The process using this data is as follows:

1. The data is split into Pandas dataframes for each team, where each team's dataframe contains all of the games that the team has played during that season.

2. With a user-specified window size, we iterate through each row of the original dataset (containing all the games that occurred in the season)

3. For a given game, we identify the date the game occurred and the two teams involved, and check in the team-specific dataframes, if these teams have played enough prior games to create a full window of game history. If each team has played enough games, their stats from each of those games are concatenated to create a training sample with a size of (window size, number of statistical categories * 2). The teams are randomly assigned as either "Team1" or "Team2" to reduce the possibility of overfitting in the models.

4. This training sample is then linked to a target of the final score between the two teams in the game that they play vs. each other (i.e., the game that we are at in the original dataset as we iterate through it in Step 2).

With this process, we created a training dataset.

To test, the process is as follows:

1. Prompt the dataset creation function with the desired date and teams, and create the appropriate sample

2. Use the trained model to predict the final score using this sample

4.2. Exploratory Data and PCA Analysis

We began by examining the correlation matrix of the game-level features to identify potential multicollinearity. As expected, free throw attempts (FTA) and free throws made (FTM) were found to be highly correlated, reflecting the inherent dependency between attempts and conversions.

To reduce dimensionality and identify dominant patterns in the data, we applied Principal Component Analysis (PCA). The first principal component (PC1) captured the largest variance and was most strongly associated with field goals made (FGM), field goals attempted (FGA), assists (AST), three-point attempts (3PA), and three-point makes (3PM), suggesting it represents overall offensive productivity.

The second component (PC2) revealed a contrasting structure, where the same offensive metrics, when attributed solely to Team 2, were negatively correlated. This indicates that PC2 may differentiate relative offensive performance between competing teams.

PC3 showed comparatively higher loadings for offensive rebounds (OR) and fouls, pointing to a component that reflects physical or aggressive styles of play.

To determine the appropriate number of principal components to include as model inputs, we analyzed the cumulative explained variance. We selected the top components that collectively explained approximately 90% of the total variance, allowing for a more compact and informative feature set for downstream modeling.

Figure 1 shows the contribution of each statistic from the original data towards the principal components that were then used for training.

4.3. Baseline Models

We tested on three models as discussed above. All of them have the same window size of three games, batch size of 16/32 and learning rate of 0.001 for consistency, and each model was trained for 50/100 epochs. We saved the best-performing model for each category to see which architecture performed the best.

4.3.1 Multi-Layer Perceptron (MLP)

The MLP is a fully-connected feed-forward model architecture designed to model non-linear relationships, which is useful for recognizing complex patterns in data. The layer architecture is as follows:

Fully Connected Layers: A series of dense layers (512, 256, 128, 64, 32, and 16 neurons)

Activation Functions: LeakyReLU for hidden layers, Sigmoid for the final output

The MLP model processes structured game data, applies non-linear transformations, and reduces dimensionality across hidden layers before making a final prediction using a sigmoid activation function.

4.3.2 Convolutional Neural Network (CNN)

CNNs are designed to extract spatial features from sequential data and are particularly proficient in long-term pattern recognition.

Convolutional Layers: Three 1D convolutional layers (64, 128, and 256 filters, respectively)

Pooling Layers: Max pooling after convolutional layers

Batch Normalization: Applied to each convolutional layer

Fully Connected Layers: A series of dense layers (512, 256, 128, 64, and 1 neuron)

Activation Functions: LeakyReLU for hidden layers, Sigmoid for the final output

The CNN architecture extracts hierarchical features using convolutional and pooling layers. The flattened feature representation is passed through fully connected layers to produce a final probability score for game predictions.

4.3.3 Long Short-Term Memory (LSTM)

One architecture used for basketball game outcome prediction is an LSTM network. LSTMs are designed to process sequential data and capture long-term dependency patterns.

Epochs: Defined dynamically at runtime

Hidden Dimension: Configurable

Number of LSTM Layers: Configurable

The LSTM layer processes input sequences with a hidden dimension and multiple layers. The final output is passed through a fully connected layer, adjusting dimensions for prediction.

4.4. Proposed Models

On top of the above mentioned baseline models, we propose three models that are designed to perform better by combining different architectures. These models aim to capture both spatial patterns and sequential dependencies more effectively. Specifically, we explore two variations of CNN-LSTM hybrids and one transformer to improve predictive performance.

4.4.1 Transformer

Transformers are used for processing sequential data through attention mechanisms, allowing the model to hone in on the most important portions of the input sequence.

Number of Attention Heads: 8

Number of Transformer Encoder Layers: 4

Dropout: 0.1 (between encoder layers)

The Transformer encoder layers process input sequences through stacked layers of multihead attention and feed forward networks. This allows the model to capture complex dependencies between the elements of the input sequence. The final prediction is produced by passing the output of the encoder layers through one fully-connected layer.

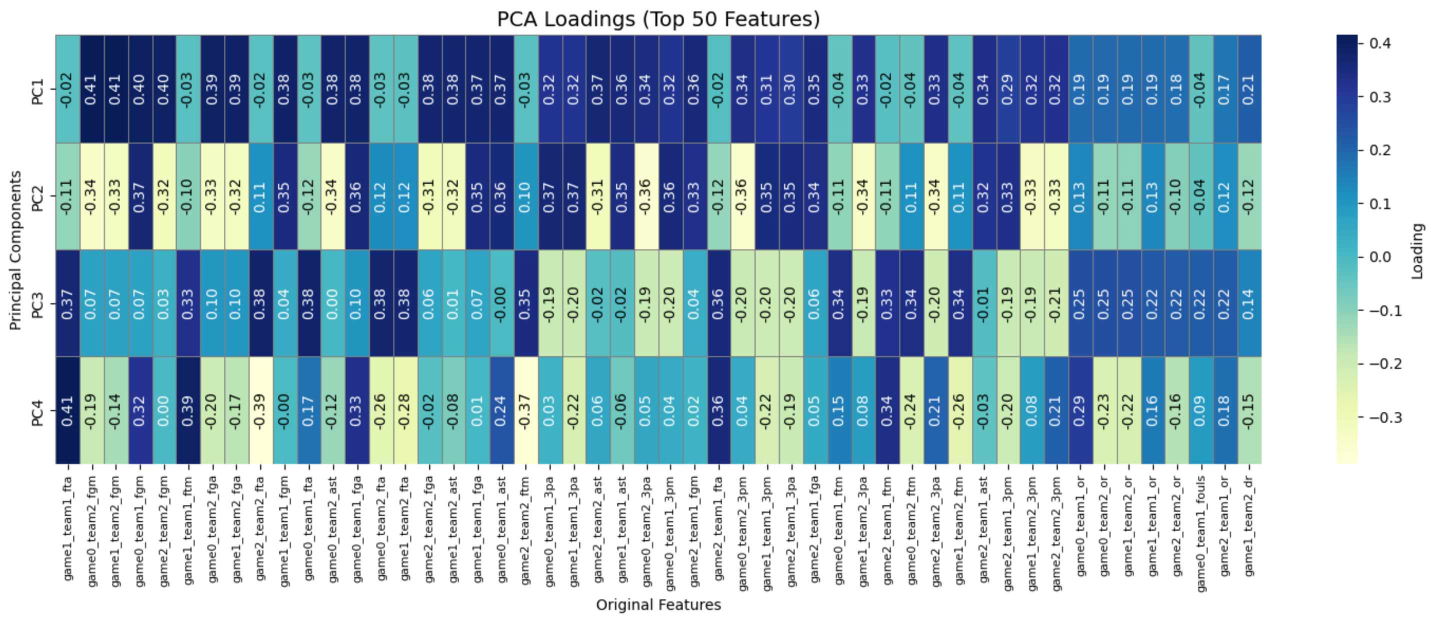

Figure 1. PCA loadings capture correlations between original features and Principal Components

4.4.2 Sequential CNN-LSTM

We implemented a Sequential CNN-LSTM model where the input data is first processed through a 1D convolutional layer. The extracted features are then passed into an LSTM layer to capture sequential dependencies. The final output is produced through fully connected layers with a sigmoid activation. This model was trained with a batch size of 16, a learning rate of 0.001, and for 20 epochs using AdamW optimizer and binary cross-entropy loss.

4.4.3 Parallel CNN-LSTM

In the Parallel CNN-LSTM model, the CNN and LSTM branches process the input data independently before their outputs are concatenated. The CNN branch extracts spatial features through convolution and pooling, while the LSTM branch models sequential patterns. The fused features are passed through dense layers to make the final prediction. The model was trained with hyperparameters similar to those of the sequential model, using AdamW optimization and binary cross-entropy loss.

5. Experiments and Results

We evaluate our models based on accuracy, precision, F1-score and recall, which are standard metrics when working with binary classification. The F1-score is the weighted average of Precision and Recall and is particularly useful in datasets like ours where class distribution is uneven.

During training, binary cross-entropy (BCE) and mean-squared error (MSE) are used as the loss functions. The BCE loss is used to capture how well the network is predicting a winner. MSE is being utilized as an auxiliary function that tracks the error between the model prediction of final score and the actual final score of a game.

5.1. Preliminary Results - Baseline Models

	Accuracy	**Precision**	**Recall**	**F1-Score**
MLP	0.60084	0.66734	0.62592	0.62134
CNN	**0.63445**	**0.74630**	0.62754	**0.66983**
LSTM	0.61660	0.61157	**0.66013**	0.61212

Table 1. Performance Metrics

From Figures 2 and 3, we see that the CNN performs the best in terms of Accuracy, Precision and F1-Score. However, the LSTM scores the best in recall. This means that the CNN correctly identified the most true positives and negatives over the total pool of observations. Additionally, the positive predictions made by the CNN have the highest chance of being correct. With the best F1-Score, the CNN also has the best balance of precision and recall across the board. However, the LSTM was able to identify the most amount of positive cases present in the observations having scored the highest recall. Thus, overall, the CNN performed the best during the baseline experiments.

These results were based solely on the results of the 2019 season, to give an idea of how well a deep model may actually perform when given more data. Based on these results, we decided to explore an LSTM/CNN model as well

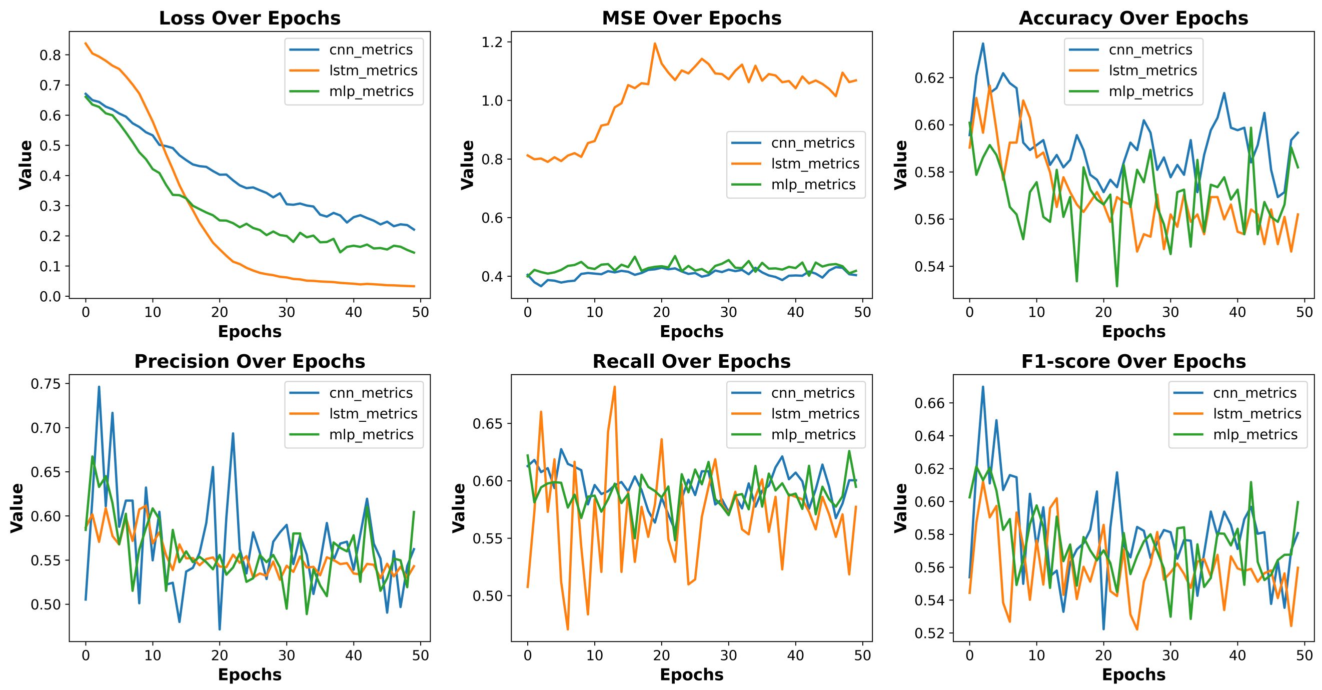

Figure 2. Training Results for each Architecture

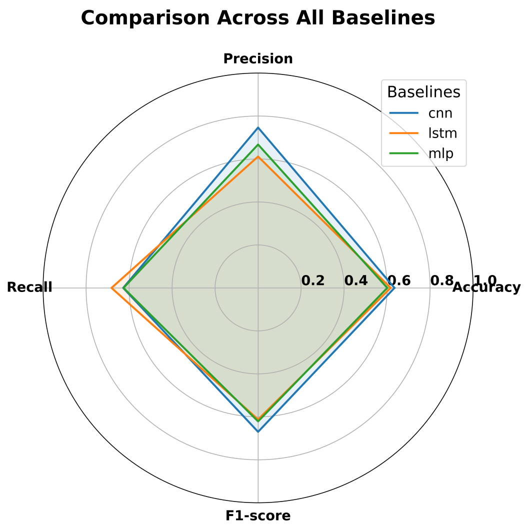

Figure 3. Spider plot comparing performance across all baselines.

as a transformer. We also wanted to get a better understanding of which features to include for training, which led to the implementation of PCA, as described previously. The resulting components were then used for training the transformer and LSTM models.

5.2. Final Results

In Table 2 we present the results of retraining all of the baseline models along with the new models on all of the data, besides 2025's data, using PCA to determine the most important statistics (S and P represent the Sequential and Parallel implementations of the LSTM/CNN respectively). From our results, we can see that CNN maintains the highest accuracy, precision and F1-score during the training phase. However, the Transformer receives the maximum possible recall, meaning that all true positives were identified and no false negatives were raised. This could mean that the transformer adopts a message in which it raises every result as "positive" meaning that the accuracy is lower due to the fact that even negative true labels were predicted as positives (false positives). These results can be better visualized by the spider plot [4]

	Accuracy	Precision	Recall	F1-Score
MLP	0.60084	0.66734	0.62592	0.62134
CNN	**0.63445**	**0.74630**	0.62754	**0.66983**
LSTM	0.61660	0.61157	0.66013	0.61212
Transformer	0.62407	0.61196	**1.0**	0.65251
S-LSTM/CNN	0.63340	0.64048	0.75281	0.65311
P-LSTM/CNN	0.62969	0.65035	0.69317	0.64919

Table 2. Performance Metrics

Overall, the CNN most likely performs the best across the board due to CNN's proficiency in long-term pattern recognition. Additionally, the deeper models, such as the Transformer, S-LSTM/CNN and P-LSTM/CNN, suffer under the condition of minimal data. Even with the data from 2011-2024, these models struggle to define robust patterns in the data. Therefore, using a larger dataset would improve the accuracy of the deeper models and possibly lead to better results in predictions for an entire bracket. However, we do see that the three new architectures explored in this section perform better in terms of accuracy than the MLP and LSTM baseline models. This is due to these architectures having more advanced mechanisms for long-term pattern recognition than that of our baseline models, despite not being given enough data to fully learn the dataset.

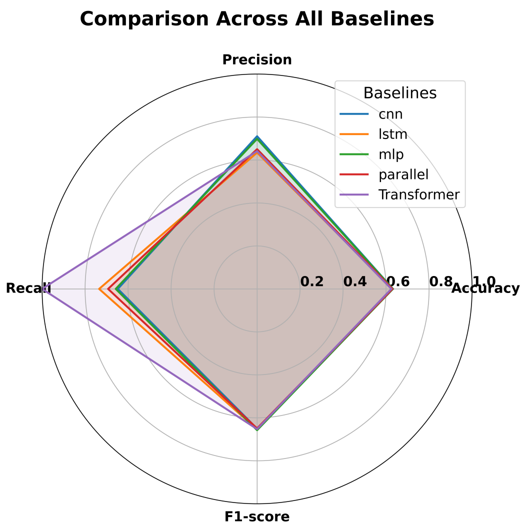

Figure 4. Spider plot comparing performance across all models

5.3. Testing on 2025 Bracket

The ultimate test for any of these models was the 2025 March Madness bracket. We took the trained transformer and the LSTM models and predicted the outcomes of the games to build our own brackets. Figure 5 shows the result of using the transformer.

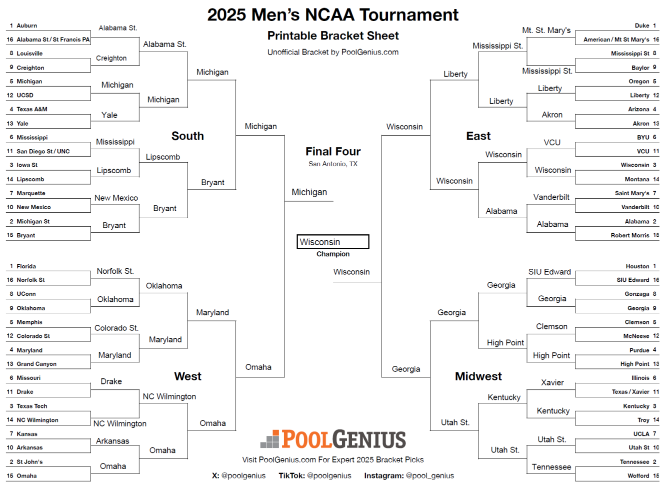

Figure 5. Predicted Bracket from Transformer

Overall, this was a poorer showing than expected. Many of the highly-seeded teams were knocked out by teams that were not in the same tier of skill. The network predicted 10 games correctly out of the 32 first-round games. Over the course of filling out this bracket, we noticed a few reasons that this result did not meet expectations.

When creating the dataset and while predicting games, one team is denoted as Team 1, and the other is denoted as Team 2. During dataset creation, these denotions were randomly assigned, so that the team number would not affect the results. When we started predicting games however, we noticed that Team 2 won most of the matchups, but also that when we swapped which team was 1 and which team was 2, the winner of the matchup changed, sometimes by a large margin (10 or more points). For example, for Auburn vs. Alabama St., if Alabama was Team 2, it won by around 10 points. But if Auburn was Team 2, then it won by a similar margin. Despite attempts to randomize this assignment and to normalize the final scores during training and un-normalize the predictions, this discrepancy still occurred.

Another reason for this poor performance could be that the stats from the games alone were not enough. Some of the lower seeded teams (such as 14, 15, or 16) had much easier schedules, and thus their statistics from past games in their regular season may have looked better than those of higher-seeded teams who had tougher schedules. The stats from every team's games were looked at in a vacuum; that is to say they didn't incorporate any knowledge about team conference or seeding, which was our intent. So there is a good chance that because the network did not have access to this type of "program" information about each team (i.e. quality of their program), it simply gave the nod to teams with better numbers in their previous games, which weren't adjusted in any way for the level of team they were playing against. This makes sense when the network was trained on all teams' data and couldn't retain information about individual teams.

Figure 6 shows the predictions made by the LSTM model. It made 13 correct predictions out of the first round of 32 games. This was slightly better than the transformer model's predictions, but here too the bracket spiraled quickly out of control. Most noticeably, the Final Four was composed of two 16-seeded teams and a 13 seed, which is highly unlikely to ever happen. Comparing the first round results of this bracket to the transformer bracket, the two models made 15 different choices over the 32-game slate. This indicates a sense of randomness, which can be expected from the training accuracies of both models. However, there are some similar trends, in how the most lowest-seeded teams (13-16) are consistently beating higher seeds, which points to the easier schedules

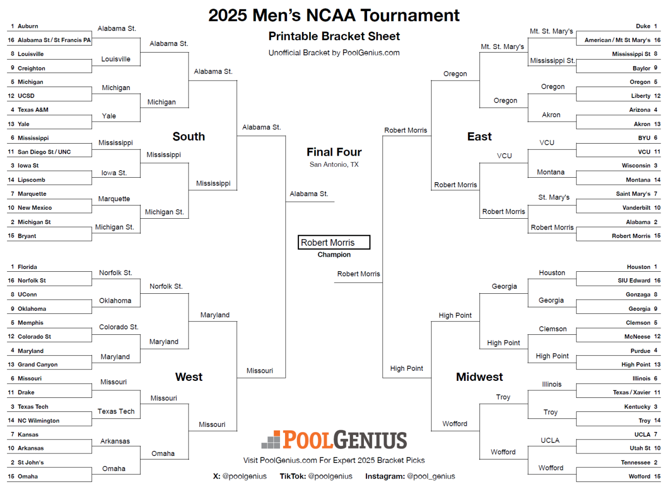

Figure 6. Predicted Bracket from LSTM

and better numbers reasoning mentioned previously. Certain teams like Michigan and Georgia were favored by both to some extent, and middle-of-the-pack teams had a possibility of knocking off one of the lower-seeded statistical powerhouses. The flip-flopping of wins based on the assignment of each team to 1 or 2 was much less prevalent with this model, on the positive side.

The comparison of the choices made by both models seems to indicate that randomness is a factor, or in other words, features that weren't accounted for. Also, the common preferences for certain teams indicates that there are some elements of this that the network understands and those can be investigated further in future work.

6. Conclusion

Overall, the networks did not seem to learn enough to do better than picking winners nearly at random, even with the inclusion of time-series data to capture momentum. However, we still gained insights from a machine learning perspective on the knowledge that the networks likely required and the representation of that knowledge.

The first insight was that data augmentation can be critical for success - finding a way to improve our assignment of teams to Team 1 and Team 2 would be a suggestion for future work. Ideally, we would want the predicted result of the game to stay the same, regardless of who is assigned which team number.

The second insight was that we needed a way to capture team "goodness" in the features. The statlines from previous games wasn't enough to explain to the network that a team like Alabama St. would likely lose nine out of ten games to a team like Auburn, even if Alabama St. had put up great numbers against their competition. For future work, information about current records and conference membership could boost the network's ability to discern this. In general, this is the difference between the regular season and the NCAA tournament, because many lower-seeded teams do not get the opportunity to play top-end talent during the season, so adjusting for that in the tournament predictions is difficult, outside of committee seed assignments.

This type of approach may be more useful at the NBA level, where the competition is much tighter and every team plays each other for the most part during the regular season.

7. Discussion on Deep Learning Aspects

In our project, we tackled March Madness outcome prediction as both a binary classification and a regression problem, aiming to predict winners and final scores using historical game data. Given the importance of momentum and performance trends in basketball, we focused on time series analysis, primarily leveraging LSTM models to capture sequential dependencies. We also experimented with CNNs, MLPs, and Transformers using a fixed window of past games for comparison.

Our models learned all parameters in convolutional, recurrent, and dense layers, since we were training from scratch, while post-processing steps like interpreting probabilities into decisions remained non-learned. Inputs were normalized and structured as windows of game history between two teams, and outputs were either win probabilities or score vectors, depending on the task. Binary cross-entropy and mean squared error were used as loss functions. To combat overfitting and encourage generalization, we randomized team ordering and applied batch normalization and dropout. We tuned hyperparameters such as learning rate, batch size, and window size, with Adam serving as the optimizer. We used PyTorch, starting with simple architectures and gradually making them more complex based on performance.

Overall, our deep learning approach provided insights into how past performance impacts tournament outcomes and demonstrated the strengths and challenges of time-series modeling in sports prediction.

References

[1] Jordan Gumm, Andrew Barrett, and Gongzhu Hu. A machine learning strategy for predicting march madness winners. In *2015 IEEE/ACIS 16th International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, pages 1–6, 2015. 1

[2] Jun Woo Kim, Mar Magnusen, and Seunghoon Jeong. March madness prediction: Different machine learning approaches with non-box score statistics. *Managerial and Decision Economics*, 44(4):2223–2236, Jan. 2023. 1

[3] N. David Pifer, Timothy D. DeSchriver, Thomas A. Baker, and James J. Zhang. The advantage of experience: Analyzing the effects of player experience on the performances of march madness teams. *Journal of Sports Analytics*, 5(2):137–152, Oct. 2018. 1

[4] Gang Shen, Di Gao, Qian Wen, and Rhonda Magel. Predicting results of march madness using three different methods. *Journal of Sports Research*, 3(1):10–17, Jan. 2016. 1

[5] Bart Torvik. 2025 team stats – customizable college basketball tempo free stats. Web, 2025. 2

8. Team Contributions

Student Name	**Contributed Aspects**	**Details**
Kiran Nazarali	MLP and CNN Implementation and Training; Exploratory Data Analysis and Feature Selection; Report Writing	During the midterm phase, I implemented and trained MLP and CNN models with a fixed window method using the 2019 dataset, developing the models from scratch. For the final phase, I conducted exploratory data analysis focusing on feature correlations, PCA, and PCA loadings to identify key contributing features; the extracted PCA components were then used by other team members in their models. I also helped with report writing.
Henry Raman	Transformer Implementation and Training; Implementation of a pipeline for using all datasets; report writing	During the midterm phase of the project, I wrote the project check in report and helped analyze the results of training and testing on the various models. During the final phase, I implemented the transformer model and trained it on the 2019 dataset. After all of the models had been trained on just the 2019 dataset, I added a pipeline that allows us to train each of the models on all of the datasets that we had collected, providing more robust training and better results. I also aided in the writing of the final report.
Sheraz Hassan	Completely handled Plotting script, LSTM, 2 combined models of LSTM and CNN; Handled CNN, MLP and Random forest to run over all data; Report writing	I wrote the script that reads all the results from all models and plots them to keep everything consistent. To make sure all models give the same type of output, I integrated the needed changes into the CNN, MLP, and Random Forest pipelines. I also rewrote the CNN, MLP, and Random Forest code so they can run on PCA-transformed data. I fully wrote the LSTM model and also worked on two different combined models using LSTM and CNN. Besides that, I also helped with writing the report.
Sai Sriraman	Time-series idea for technical direction, dataset retrieval and pipeline creation, bracket testing and resulting analysis, report writing	I led the technical direction for the project in terms of investigating the time-series approach for March Madness games. I found the data sources and created the pre-processing pipeline for loading the data, cleaning it, and making the rolling windows. I took the completed models and developed the testing pipeline for the 2025 bracket and analyzed the results of these predictions. Finally, I helped with writing the report.

Table 3. Contributions of team members.