AI Applications for Solutions to Popular Games: A Comparative Analysis of Heuristic, Search Algorithms, and Machine Learning

Sanay Nesargi

Abstract

Artificial Intelligence (AI) encompasses a broad spectrum of techniques, ranging from classical search algorithms and heuristics to more advanced machine learning and deep learning models. To find the solution for popular game problems, the choice of the appropriate AI method depends on the nature of the problem being addressed. For well-defined, rule-based problems such as turn-based games like Tic-Tac-Toe or Connect 4, classical algorithms like Minimax and Monte Carlo Tree Search (MCTS) are highly effective due to their ability to explore finite state spaces and make optimal decisions. However, for more dynamic and complex problems, such as those encountered in sports analytics or real-time decision-making, advanced machine learning techniques such as Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and Long Short-Term Memory (LSTM) networks are more effective to capture intricate patterns and handle unstructured data. Despite the growing capabilities of deep learning, simpler methods like heuristic-based optimization and statistical techniques such as curve fitting and K-nearest neighbors (KNN) can still be highly effective in certain domains. This paper explores the strengths, weaknesses, and effectiveness of various AI techniques and develops guidelines on how to select the right approach based on problem complexity and data characteristics. Through case studies of different popular games, we demonstrate how different AI methods can be applied to real-world problems, and how hybrid approaches can often provide the best solution for complex tasks.

Introduction

Artificial intelligence (AI) has revolutionized numerous fields by providing computational methods to solve increasingly complex problems. These AI techniques span a diverse range of approaches, each suited to specific types of challenges. For popular games with well-defined structures, such as traditional board games like Chess and RISK, classical search algorithms and heuristics are often the most effective solutions. These problems are characterized by finite state spaces in which the goal is to optimize decisions based on clear rules and strategies. Algorithms like Minimax and Monte Carlo Tree Search (MCTS) are particularly adept at navigating these scenarios [1], [2].

However, as problems become more dynamic and data-driven, such as those encountered in sports analytics or real-time decision-making scenarios, traditional algorithms begin to fall short. In these more complex, unstructured environments, machine learning (ML) and deep learning techniques, including models like Long Short-Term Memory (LSTM) networks, provide the flexibility and power needed to

analyze large datasets, learn patterns over time, and make predictions under uncertainty [3], [4]. These techniques excel in domains where data is less structured and involves temporal dependencies, such as predicting player performance in sports [5].

In this paper, we explore how various AI techniques—from classical algorithms like Minimax and MCTS to modern machine learning models—can be effectively applied to solve different kinds of problems. By examining case studies from RISK (a classical strategy board game) and CourtSightML (a machine learning model for sports analytics), we highlight the strengths and weaknesses of both traditional and modern AI approaches. We also discuss how these methods can be combined to address more complex, real-world challenges, illustrating the versatility of AI techniques across a range of domains.

AI Techniques for Problem Solving: A Comparative Overview

Classical AI Techniques: Search Algorithms and Heuristics

Classical AI techniques are foundational to the development of problem-solving methods, particularly in domains that involve structured decision-making or well-defined rules. These methods include search algorithms and heuristics, which are commonly applied in games and decision-making scenarios with finite state spaces and clear objectives. Despite the rise of more complex AI approaches, such as deep learning, these classical techniques remain highly effective for certain problem domains due to their simplicity, interpretability, and efficiency when applied in appropriate contexts.

Table 1. Strengths and limitations of selected AI algorithms

Algorithm	Description	Strengths	Limitations
Minimax	AI algorithm for two-player games; uses a game tree to evaluate and select optimal moves.	• Guarantees optimal play in deterministic environments. • Simple to implement.	• Computationally expensive for large state spaces. • Struggles with incomplete information.
Monte Carlo Tree Search (MCTS)	Probabilistic algorithm for large state spaces; uses random simulations to refine decisions.	• Efficient in large decision spaces. • Handles uncertainty well.	• Computationally demanding for real-time decisions. • Slower in dynamic environments.
K-Nearest Neighbors	Classification algorithm	• Simple and easy	• Computationally

(KNN)	using similarity to prior examples for decision-making.	to implement. ● Adaptable to various problems.	expensive with large datasets. ● Sensitive to noisy data.
Classification Algorithms	Techniques like decision trees, SVM, and logistic regression for pattern recognition.	● Transparent decision-making (e.g., decision trees). ● Flexible for various data types.	● Prone to overfitting. ● Struggles with high-dimensional data without proper regularization.

Machine Learning: A Shift to Data-Driven Models

While classical AI techniques such as search algorithms and heuristics are highly effective for structured, well-defined problems, modern applications increasingly involve vast amounts of data and complex, dynamic systems. These scenarios require more adaptable and powerful models that can learn from data rather than relying on predefined rules. Machine learning (ML) provides such an approach by leveraging data-driven models to identify patterns, make predictions, and optimize decision-making [6]. Among the various ML techniques, neural networks and genetic algorithms stand out as particularly versatile tools for solving real-world problems across different domains [9]. Neural networks, especially deep learning models, are widely used in tasks such as speech recognition, computer vision, and natural language processing, while genetic algorithms are applied to problems involving optimization and search, such as engineering design and resource allocation [6], [9]. These methods are particularly useful in environments where traditional AI struggles with unstructured data and evolving contexts.

While there are other types of Neural Networks and Machine Learning algorithms, for the purposes of this paper, we focus on the following:

Table 2. Different Types of Neural Networks

Algorithm/Model	**Description**	**Strengths**	**Limitations**	**Applicable Games**
Recurrent Neural Networks (RNNs)	Models designed to handle sequential data by maintaining memory of previous inputs.	● Excels in time-series and sequential prediction tasks. ● LSTMs are ideal for modeling	● Requires large datasets. ● Computationally expensive. ● Struggles with rare or extreme	Sports Analysis, RPGs, Dynamic Simulations

	LSTMs (Long Short-Term Memory networks) are a specialized type of RNN that captures long-term dependencies effectively.	long-term dependencies and complex temporal patterns.	events.	
Convolutional Neural Networks (CNNs)	Primarily for image recognition and processing spatial data.	• Automatically learns hierarchical spatial features. • Effective for structured data like images or grid-based games.	• Requires large, labeled datasets. • Limited to structured data; less useful for non-spatial input.	Go, Chess, Checkers, Visual Board Games
Feedforward Neural Networks (FNNs)	Processes data from input to output without recurrence or convolutional layers.	• Simple to implement. • Effective for non-sequential, non-spatial tasks.	• Limited for sequential or spatial data. • Performance is often lower compared to specialized models like CNNs or RNNs.	Strategy games, Classification tasks

Case Study I: RISK and the Application of Search Algorithms

In this paper, we offer a case study of creating an AI agent to play near-optimal moves in the board game of **RISK**, demonstrating how classical AI techniques can be applied to increasingly complex problems. RISK is a strategy game that requires players to take turns attempting to capture territories on a world map by using military forces to engage in battles and conquer opposing territories. The game's rules and objectives are relatively simple, yet the complexity emerges from the strategic planning, negotiation, and probabilistic outcomes that occur throughout the game.

Overview of RISK: A Strategic Board Game

Similar to the popular board game Monopoly, RISK is a turn-based game. The primary goal in RISK is to control territories on a map and gradually eliminate opposing players by capturing their territories. Each player starts with a set number of armies placed on various territories, and during each turn, players must decide how to allocate their forces and attack or fortify territories.

The game involves several key components:

- **Territories**: The board is divided into regions (territories) grouped into continents. Each player controls a set of territories and must use them to launch attacks, defend against other players, and acquire more territories.
- **Armies**: Each player's military forces are represented by a number of armies. These armies can be deployed on any controlled territory to attack or defend.
- **Battles**: When players engage in attacks, the outcome is based on dice rolls, introducing an element of chance into the game. The attacker rolls dice to determine the strength of the attack, while the defender rolls to resist the attack. The dice rolls introduce an inherent uncertainty, making strategy and tactics crucial to success.
- **Reinforcements**: At the beginning of each turn, players receive additional armies based on the number of territories they control, which further complicates decision-making by providing opportunities to reinforce defenses or prepare for attacks.
- **Goal**: The ultimate goal of RISK is to eliminate all other players and take control of the entire world map, but this requires not just careful planning but also adapting to the constantly shifting state of the game.

Challenges in Creating an AI for RISK

At first glance, RISK may appear to be a relatively straightforward game, but its complexity arises from the need to balance several strategic and tactical objectives:

1. **Territory Control and Expansion**: Players must manage their territory and determine which areas to expand into. Expanding too quickly can overextend a player's forces, while too little expansion can leave a player vulnerable.
2. **Resource Management**: Reinforcements are earned based on the number of controlled territories, but this also means that spreading too thinly across the board can prevent players from maximizing reinforcements.
3. **Battle Strategy**: The random nature of dice rolls means that players must carefully decide which battles to engage in and how to distribute their forces. Additionally, knowing when to fight and when to fortify defenses requires deep strategic thinking.

4. **Opponent Behavior**: Predicting the actions of opponents is one of the most difficult aspects of RISK, as players must try to predict how others will react to shifts in territory control, battle outcomes, and strategic moves.

These factors make RISK a more complex game than it initially seems, requiring advanced strategic decision-making. The inherent randomness of the dice rolls complicates things further, requiring the AI to consider not only the best possible outcomes based on available data but also to factor in the probabilistic nature of battles.

Applying Classical AI to RISK

To handle the strategic depth of RISK, we turn to classical AI techniques. One of the most effective algorithms for such strategy games is Minimax, which uses a decision-tree structure to evaluate the potential outcomes of different moves. Minimax works by assuming that both players will act rationally and will try to maximize their own benefit while minimizing their opponent's. It evaluates all possible moves in a tree-like structure and selects the optimal one based on these evaluations. However, Minimax relies on a fully (or mostly fully) defined game state space, and with the complexity of RISK, such an approach becomes computationally expensive, especially due to the large number of possible configurations of the game state [7].

In contrast, Monte Carlo Tree Search (MCTS) is more suited to handle the dynamic and uncertain aspects of RISK. MCTS doesn't require full knowledge of the game's state and excels at exploring large decision spaces by simulating possible future game states. It is particularly effective in games like RISK, where uncertainty, such as the randomness of dice rolls, plays a significant role. MCTS works by running simulations (or rollouts) to predict the outcomes of different actions, refining its evaluation of potential moves over time [4]. This ability to simulate and adapt to uncertainty makes MCTS a powerful tool for games like RISK, where probabilistic elements heavily influence gameplay outcomes.

In our case study, we designed an AI agent that uses MCTS to evaluate possible moves in RISK by simulating a series of games, iteratively improving its decision-making through the exploration of various game states. This agent evaluates potential moves by considering:

- The game state (current territories controlled, number of armies, etc.).
- The probable outcome of attacking certain territories.
- Potential strategies for fortifying defenses or preparing for the opponent's next moves.

Using MCTS for Strategic Decision-Making in RISK

MCTS is particularly suited for handling the uncertainty and large state space in RISK because it focuses on simulation-based exploration rather than exhaustive tree searches. The basic idea behind MCTS is to randomly simulate several possible game scenarios (called rollouts) from a given state, evaluating the

likelihood of winning from those scenarios. These simulations help the AI agent prioritize moves based on the expected outcome over multiple simulations.

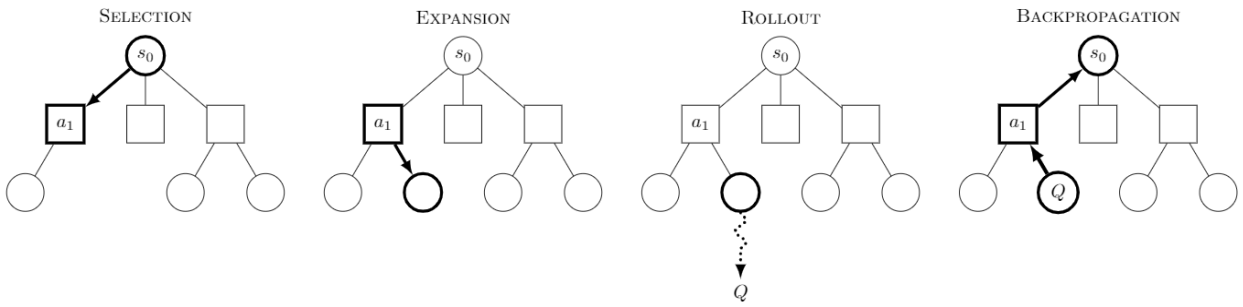

Figure 1: Overview of the MCTS Algorithm. By Robert Moss. *Own work, CC BY-SA 4.0*, https://commons.wikimedia.org/w/index.php?curid=111182752

Through repeated iterations, the AI becomes better at selecting moves that maximize its chances of success in RISK, even when faced with complex, dynamic decision-making.

Implementation

The goal of this implementation of MCTS was to create an AI agent that chooses well-known RISK strategies based on the current board state, opportunities to attack, and the relative safety of the current position, among other factors. In this approach, MCTS, or more specifically, DL-MCTS (Depth-Limited MCTS), is used for a heuristic-based analysis of the current position. This involves utilizing a UCT (Upper Confidence Bound for Trees) strategy, as well as incorporating it into the final position evaluation calculation, which ultimately determines which strategy to employ [7]. The use of DL-MCTS allows the agent to simulate possible future states, considering both short-term and long-term consequences, and to make decisions based on the most advantageous outcomes [8].

As part of the analysis, we pitted different strategies against each other to observe which resulted in the best average outcomes. These strategies included a mix of aggressive and defensive tactics, where the agent decided when to build troops, when to attack, and when to fortify its position. However, we found that the deviations in results were statistically insignificant, most likely due to near-perfect play, where the AI's decision-making approached the optimal strategies given the constraints of the game. The statistical insignificance of the deviations suggests that the model, through the use of DL-MCTS, consistently made near-optimal decisions, leaving little room for variation in outcomes when strategies were tested against each other.

Creating the Game Environment

The game space was represented by a hash table, with keys as the territory names and values representing the agent who owned the territory, as well as the number of troops stationed there (stored as an ordered pair). Graphically, the user was presented with a game map, where icons for each agent and indicators for troop counts were displayed and updated dynamically as the game state hash table changed.

Additionally, troop bonuses were tracked in a separate hash table and applied during the main game loop. These bonuses were added after each turn, in accordance with the rules of the game. Battles were simulated using random number generation to mimic dice rolls, with the standard attack-defense rules of RISK applied.

Running the DL-MCTS Algorithm

The implementation of the DL-MCTS algorithm in our system allows us to model and simulate various strategic decisions within the RISK game by traversing the search tree and evaluating different potential game states. The core idea behind DL-MCTS is to efficiently balance exploration (trying new moves) and exploitation (capitalizing on the most successful strategies discovered so far). The UCT (Upper Confidence Bound for Trees) plays a crucial role in this, helping us determine which branches of the search tree to explore based on the trade-off between exploration and exploitation.

For DL-MCTS, we used a search depth of 12 to balance performance with accuracy. Going deeper could potentially improve the quality of the decisions made, but was impractical due to computational time constraints. To optimize memory usage and make the representation more efficient, we encoded the game board as binary values, akin to the bitboards used in Chess, allowing fast access to the state of the game and efficient calculations.

In this context, the DL-MCTS algorithm allows each strategy to be encapsulated as a distinct function, which modifies the game state during an attack scenario, troop reinforcement, or fortification. Here's a breakdown of the specific strategies implemented in our system, integrated into the DL-MCTS framework:

Table 3. Implemented RISK strategies

Category	**Strategy**	**Description**
Reinforcement Strategies	Reinforce Weak Territories	Focuses on defending territories with low troop counts by optimal troop distribution.

Reinforcement Strategies	Reinforce Owned Key Territories	Prioritizes reinforcing key territories essential for controlling large areas.
Reinforcement Strategies	Reinforce Attacked Territories	Reinforces territories under attack to make them harder to conquer.
Attack Strategies	Aggressive Attack	Prioritizes attacks with a high probability of success using heavily fortified territories.
Attack Strategies	Guerilla Style Attack	Targets weaker enemy territories to weaken the opponent's position strategically.
Attack Strategies	Blitz Attack	Attacks key but weakly defended enemy territories to gain strategic advantage.
Troop Transfer and Fortification Strategies	Troop Transfer	Reallocates troops to newly conquered territories to secure them against counterattacks.
Troop Transfer and Fortification Strategies	Fortify Weakest Territories	Strengthens the weakest player-owned territories to prevent vulnerabilities.

A specific example of optimality in RISK using DL-MCTS occurs when an AI agent decides whether to attack a fortified territory or strengthen its position. The agent, with a balanced number of troops, faces an opponent with a heavily fortified region. Without DL-MCTS, the agent might rush into an attack, potentially losing troops. However, with DL-MCTS, the agent simulates multiple future outcomes, considering the effects of both attacking immediately or waiting to reinforce its troops. The UCT strategy helps the agent explore these paths, and it identifies that waiting for reinforcements before attacking increases the chances of success. This strategy maximizes the agent’s long-term control by avoiding unnecessary losses and ensuring a stronger offensive.

In contrast, a suboptimal decision arises when the agent, overly focused on short-term gains, decides to attack immediately. Despite the risk, it may believe the attack will catch the opponent off guard. However, this premature assault could fail due to insufficient troop strength, resulting in significant losses and leaving the agent vulnerable to counterattacks. This example shows that while DL-MCTS generally aids in making optimal decisions, an overemphasis on short-term rewards can sometimes lead to suboptimal outcomes by neglecting long-term consequences.

Case Study 2: LTSM for NBA Player Statistic Prediction

Predicting NBA player statistics is a challenging task due to the complexity of basketball as a dynamic and team-oriented sport. Long Short-Term Memory (LSTM) networks, a type of recurrent neural network (RNN), are well-suited for this problem as they can model temporal dependencies and capture patterns in sequential data [8]. These models excel at handling sequences where the order of events is important, such as predicting player performance over time based on previous games. This case study explores the application of LSTM models for predicting individual player statistics in NBA games, leveraging their ability to analyze trends and temporal relationships between game events [9].

Problem Definition

The predicted player statistics can greatly enhance fantasy basketball strategy by enabling managers to make data-driven decisions when selecting and benching players. For instance, if the model forecasts a strong performance for a player like Nikola Jokić against a weaker defensive team, fantasy managers can prioritize starting him, optimizing their team's potential for points, rebounds, and assists. Additionally, the model can help identify players in a slump or on the verge of a breakout, allowing managers to make timely roster adjustments. Overall, the solution helps fantasy managers maximize their team's performance by aligning player selections with expected game outcomes.

Data Collection and Preprocessing

A custom dataset was created using the NBA Stats API, encompassing historical game data for NBA players. This dataset includes the following key categories:

- Player-specific data
    - This includes statistics from previous games such as points, rebounds, assists, playing time, player position, and injury information.
- Team data
    - Information on the team's performance, including win/loss record, pace, and both offensive and defensive ratings.
- Opponent data
    - Data related to the opponent's performance, including defensive metrics, matchup statistics, and specific player matchups.
- Temporal data
    - Sequence of games, details about back-to-back games, and rest days, helping to account for game fatigue and recovery periods.

This custom dataset, built specifically for the predictive modeling of NBA player statistics, integrates a variety of factors that could influence player performance on any given game day.

Handling Missing Values (e.g., Injury-Related Absences)

Challenges:

- Players may miss games due to injuries, rest, or other factors, leading to gaps in their statistical records.
- Missing data could hinder sequential modeling, especially for LSTMs, which rely on complete time-series inputs.

Approach:

- Imputation Techniques:
  - Forward Filling: Missing values were replaced with the player's most recent available game statistics. This assumes consistency in performance trends.
  - Backward Filling: Used in rare cases when data was unavailable for the last recorded games but available earlier in the season.
  - Season Average Imputation: When no prior data was available (e.g., a rookie's first game after an absence), the player's season average statistics were used.
- Injury Indicator Feature:
  - A binary feature Injury Flag was added to mark whether a player was injured or unavailable for a given game.
  - This feature helped the model account for context, such as a player's return from an absence or reduced performance after injury.
- Complete Sequence Imputation:
  - If an entire sequence (e.g., the last 5 games) had missing values for a player, the sequence was excluded unless interpolated based on team or position-specific averages.

Normalizing Numerical Features for Consistent Scaling

Challenges:

- Player statistics (e.g., points, assists) vary widely in scale, potentially leading to biased model predictions.

Approach:

- Min-Max Scaling:

- All numerical features were scaled to a range of [0,1] using:
  $$[X_{\text{scaled}} = \frac{X - \min(X)}{\max(X) - \min(X)}]$$
- This preserved the relative differences between features and ensured compatibility with LSTM requirements.

- Z-Score Normalization:
  - For features with outliers (e.g., anomalously high-scoring games), Z-score normalization was applied:
  - $[Z = \frac{X - \mu}{\sigma}]$ where μ is the mean, and σ is the standard deviation of the feature.
- Feature-Specific Scaling:
  - Separate normalization was applied to season-long averages, short-term trends (e.g., P5 stats), and opponent-specific data to maintain interpretability.

Accounting for Temporality in Sequential Data

Challenges:

- LSTMs require sequences that reflect temporal relationships (e.g., recent games should weigh more than older games).
- Temporal trends like hot streaks or slumps must be captured accurately.

Approach:

- Sequence Creation:
  - Overlapping input sequences of length TT (e.g., 5 games) were generated for each player. Each sequence included:
    - Game-by-game statistics for the last TT games.
    - Contextual features like opponent data, team performance, and game date.
  - For example, for a sequence length of 5 games:
    - $\text{Game } t : [X_{t-4}, X_{t-3}, X_{t-2}, X_{t-1}, X_t]$
- Weighted Emphasis on Recent Games:
  - Recent games were assigned higher weights using exponential decay or similar methods during feature aggregation:
    $$[\text{Weighted Feature} = \sum_{i=1}^{T} w_i \cdot \text{Feature}_i] where \quad w_i = \exp(-\lambda \cdot i), \quad \lambda \text{ controlling the decay rate.}$$
- Opponent-Specific Trends:
  - Features like P5 Season PPG vs OPP were calculated dynamically by aggregating statistics from recent games against the current opponent.

- Sliding Window Technique:
  - A sliding window approach ensured that every game contributed to multiple sequences, maintaining data richness.
- Game Context:
  - Additional features such as rest days, back-to-back games, and travel distance were incorporated to provide a richer temporal context.

Model Architecture

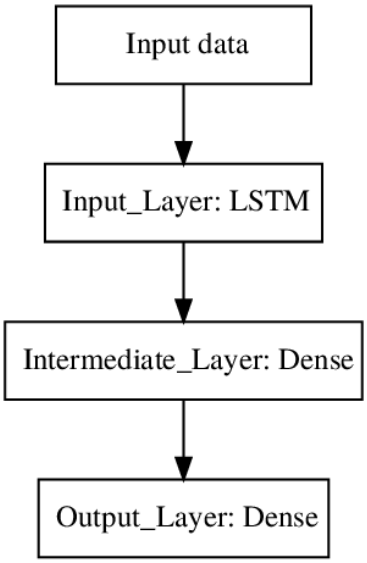

Figure 2: Diagram of the proposed neural network

Input Layer

The input layer accepts a sequence of player statistics and contextual game data, which are used to predict the game outcomes. The inputs are represented as a 2D or 3D tensor depending on how the data is structured (e.g., for a sequence of games). These inputs include:

Table 4. Input layer for the described Neural Network

Feature Name	Description	Relevance
Opponent Def. Ranking	The opponent's defensive ranking for the season.	Indicates the difficulty of the opponent's defense, impacting expected player performance.

Season PPG	The player's average points per game (PPG) for the season.	Reflects the player's overall scoring consistency throughout the season.
Season APG	The player's average assists per game (APG) for the season.	Represents the player's playmaking ability and contribution to team offense.
Season RPG	The player's average rebounds per game (RPG) for the season.	Indicates the player's rebounding effectiveness, both offensive and defensive.
P5 Season PPG	The player's average points per game (PPG) over the past 5 games.	Captures short-term trends in scoring, useful for identifying momentum or slumps.
P5 Season APG	The player's average assists per game (APG) over the past 5 games.	Highlights recent playmaking trends, indicating hot streaks or reduced effectiveness.
P5 Season RPG	The player's average rebounds per game (RPG) over the past 5 games.	Tracks recent rebounding activity, accounting for short-term performance fluctuations.
P5 Season PPG vs OPP	The player's average points per game (PPG) over the past 5 games specifically against the current opponent.	Focuses on matchup-specific scoring trends, highlighting performance against similar defenses or players.
P5 Season APG vs OPP	The player's average assists per game (APG) over the past 5 games specifically against the current opponent.	Identifies the player's recent playmaking ability in comparable matchups.
P5 Season RPG vs OPP	The player's average rebounds per game (RPG) over the past 5 games specifically against the current opponent.	Emphasizes rebounding effectiveness in similar matchups, useful for defensive or offensive rebounding analysis.

Season PPG vs OPP	The player's average points per game (PPG) against the current opponent for the entire season.	Tracks season-long scoring trends against the specific opponent.
Season APG vs OPP	The player's average assists per game (APG) against the current opponent for the entire season.	Indicates playmaking consistency in past encounters with the specific opponent.
Season RPG vs OPP	The player's average rebounds per game (RPG) against the current opponent for the entire season.	Provides insights into rebounding trends against a recurring opponent.

These features represent both the player's season performance and recent trends against specific opponents.

LSTM Layers

The LSTM layers learn temporal dependencies by processing sequences of the input data, such as how performance in past games impacts future game statistics. LSTM neurons include memory cells, each with input, forget, and output gates:

- Input Gate
  - Controls how new information is added to the memory.
- Forget Gate
  - Decides which information to discard.
- Output Gate
  - Determines what information is passed forward.

These memory cells enable the LSTM to retain relevant statistical trends over time, such as whether a player's performance improves or declines against certain opponents or in recent games. This is crucial for capturing time-based patterns, such as fluctuations in performance due to fatigue or improved form [10]. By remembering these long-term dependencies, LSTM models can make more accurate predictions based on the historical context of a player's performance [11]. Moreover, LSTMs are particularly effective

in handling the complex interactions between game events, enabling them to capture nuances such as the impact of previous games on future performance, which traditional models might miss [12].

Dense Intermediate Layer

The Dense intermediate layer enables the model to learn complex patterns by establishing connections between all neurons in the previous layer. This fully connected structure allows the model to extract meaningful features, transforming the input data through a weighted sum followed by an activation function. For example, a Dense layer with 128 neurons can capture intricate relationships in the data, contributing to the model's ability to learn and generalize better across different tasks.

Dense Output Layer

The Dense output layer predicts the three statistics for the game:

- Points (pts)
- Rebounds (reb)
- Assists (ast)

Each neuron corresponds to one of these output statistics. The Dense layer uses a fully connected network to combine the features learned by the LSTM and Dropout layers, producing predictions for each statistic. The activation function is typically linear for regression tasks, allowing the model to output continuous values for each predicted statistic (e.g., predicted points, rebounds, and assists).

Training and Evaluation

- Dataset Split:
    - The historical game data was divided into three sets: training, validation, and testing.
    - Training Set: 70% of the data
        - This set was used to train the model and update its weights. It contained the majority of the historical game data.
    - Validation Set: 15% of the data
        - This set was used to validate the model during training, helping to tune hyperparameters and prevent overfitting.
    - Testing Set: 15% of the data
        - After the model was trained, it was evaluated on this set to assess its performance and generalization to unseen data.
- Temporal Split:
    - Given the sequential nature of the data (time-series), the split was performed chronologically. For instance:
        - The first 70% of the games in the dataset (based on the date) were used for training.
        - The next 15% of games (chronologically) were used for validation.

- The final 15% of the games were used for testing.

Evaluation Metrics Computation

- Root Mean Squared Error (RMSE):
  - RMSE was used to measure the average magnitude of the prediction error. A lower RMSE value indicates better prediction accuracy.
- R-squared (R^2):
  - R^2 was used to assess how well the model explained the variance in the data. It ranges from 0 to 1, with a higher value indicating better model performance.
- Mean Absolute Error (MAE):
  - MAE was used to quantify the average absolute error between the predicted and actual values. Smaller MAE values indicate better accuracy.

Training Process

- Optimizer
  - Adam optimizer was used for training, combining the benefits of both RMSProp and SGD with momentum.
- Batch Size
  - A batch size of 64 was chosen for efficiency and model convergence.
- Epochs
  - The model was trained for 50 epochs, with early stopping based on validation loss to prevent overfitting.

Error Metric Calculation with TensorFlow

- TensorFlow Handling
  - TensorFlow's built-in functions were used to calculate the error metrics during both training and evaluation. The framework automatically computes RMSE, MAE, and R^2 as part of its model evaluation process.
  - For example, the tf.keras.metrics.MeanSquaredError and tf.keras.metrics.MeanAbsoluteError were used to calculate RMSE and MAE. The R^2 metric could be computed using a custom function or adapted from available TensorFlow utilities.

- This automated handling allows for real-time performance monitoring as the model trains and validates, providing efficient feedback and ensuring consistency across the evaluation process.

Results and Analysis

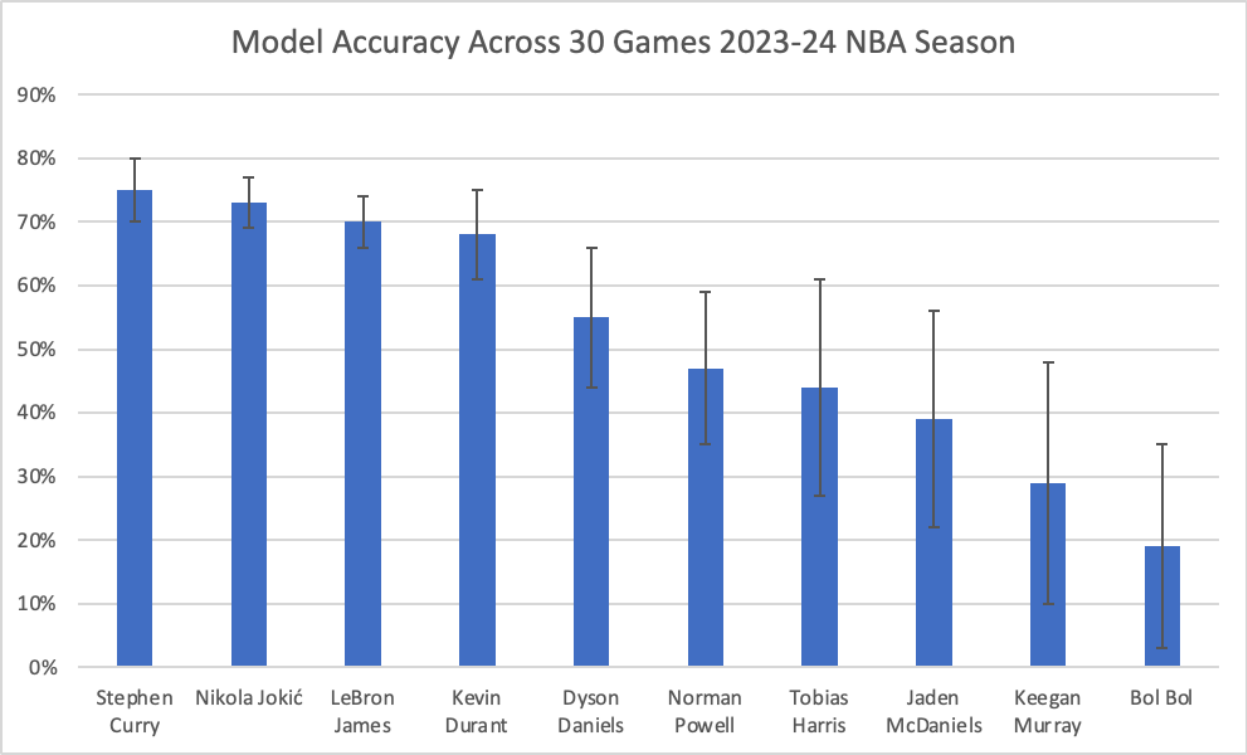

Figure 3: Model results across multiple NBA players of decreasing minutes (left to right)

The LSTM model demonstrated strong performance in predicting player statistics, achieving an overall accuracy of up to 75%. Accuracy was calculated using the following formula:

$$\text{Accuracy} = 100 - \left(\frac{\text{Average Error of All Stats}}{\text{Actual Stats}}\right)$$

Overall Accuracy

While the maximum accuracy observed was 75%, there were streaks where the model achieved accuracy levels exceeding 90%. These high-accuracy periods typically occurred when predicting the statistics of star players with consistent performance and playing time.

Trends Observed

- Consistency of Star Players
    - The LSTM model consistently demonstrated higher accuracy when predicting statistics for star players with stable roles and performance patterns, such as Nikola Jokić and Stephen Curry. These players tend to have well-established performance metrics, including points, rebounds, and assists, which are less prone to fluctuations.
    - For these players, the model's accuracy was particularly strong in predicting points and assists, given their consistent offensive roles. The accuracy for these players sometimes exceeded 90%, especially in games where their performance was not significantly impacted by external factors such as injuries or major changes in their role (e.g., due to opponent-specific strategies). The model's predictions were less influenced by noise and more aligned with their typical output, which is reflected in the high accuracy rates during these periods.
    - Moreover, these players' historical data provided a reliable training foundation for the model, allowing it to capture recurring patterns in their performance. This high accuracy is indicative of the LSTM model's capacity to leverage consistent, historical performance to generate robust predictions.
- Impact of Team and Opponent Data
    - Incorporating contextual data such as team statistics and opponent defense metrics significantly enhanced the prediction accuracy, particularly for points and assists. When external factors like the opponent's defensive strength or the team's offensive performance were integrated into the model, the accuracy of predictions improved notably. This highlights how the model can adjust for contextual influences that affect player output.
    - For example, when predicting points, the model was able to account for situations where players faced stronger or weaker defenses. It showed heightened accuracy when predicting performance against weaker defensive teams, where players like Jokić or Curry could be expected to perform at their peak. Conversely, the model performed slightly less accurately when these players were up against top-tier defenses, yet it still showed improved results compared to scenarios where such contextual data was not used.
    - The inclusion of opponent-specific data—such as defensive rankings, opposing team performance, and even recent trends—allowed the model to better align player performance with the broader game dynamics, pushing prediction accuracy closer to or above the 90% mark in certain matchups.
- Challenges with Inconsistent Players
    - One of the key challenges for the LSTM model arose with predicting statistics for players who exhibited inconsistent playing time or fluctuating performance, such as bench players or those returning from injury. These players often experience significant variability in their roles from game to game, making it more difficult for the model to predict their statistics with high accuracy.

- For example, bench players who received fewer minutes or had sporadic involvement in the offense were harder to predict. The model's predictions tended to deviate more from actual performance, leading to lower overall accuracy for these players. This issue was particularly pronounced when predicting statistics for players returning from injuries, as their performance could be impacted by factors like reduced minutes or a gradual return to form. These players often do not exhibit the stable trends that star players do, and their unpredictable roles added noise to the model's predictions.
- Despite these challenges, there were instances where the model achieved higher accuracy with these inconsistent players, particularly when their role in the game was clear or when they had a consistent number of minutes in recent games. For example, if a player had been consistently performing at a certain level in the past few games, the model was able to make more accurate predictions for their performance in subsequent games, and accuracy for such players sometimes approached 90%. However, overall, these inconsistencies contributed to the model's overall accuracy of 75%.

Conclusion

In conclusion, the best AI approach for solving a particular problem depends largely on the nature of that problem. For games like RISK, where the environment is relatively structured and deterministic, classical AI techniques such as Minimax and Monte Carlo Tree Search (MCTS) are effective. However, for dynamic and complex problems like sports analytics, where the data is large, unstructured, and temporal, deep learning models like LSTM networks provide significant advantages.

Moreover, hybrid models that combine the strengths of traditional algorithms and machine learning techniques can offer a more comprehensive solution to real-world problems. Understanding how and when to use these different AI methods is essential for developing systems that can solve a broad range of problems efficiently and accurately.

References

1. C. Browne et al., "Monte Carlo Tree Search: A Review of Recent Modifications and Applications," *Springer*, 2022. [Online]. Available: https://link.springer.com/article/10.1007/s10462-022-10228-y. [Accessed: Mar. 23, 2025].
2. Y. Lu et al., "Deep Neural Network and Monte Carlo Tree Search Applied to Fluid-Structure Topology Optimization," *Sci. Rep.*, vol. 9, no. 1, 2019. [Online]. Available: https://www.nature.com/articles/s41598-019-51111-1. [Accessed: Mar. 23, 2025].

3. D. Silver et al., "Thinking Fast and Slow with Deep Learning and Tree Search," *NeurIPS*, 2017. [Online]. Available: https://papers.neurips.cc/paper/7120-thinking-fast-and-slow-with-deep-learning-and-tree-search.pdf. [Accessed: Mar. 23, 2025].
4. F. Alamudun, "Monte Carlo Tree Search and Reinforcement Learning," *ResearchGate*, 2017. [Online]. Available: https://www.researchgate.net/post/Monte-Carlo-Tree-Search-and-Reinforcement-Learning. [Accessed: Mar. 23, 2025].
5. "Monte Carlo Tree Search-Based Deep Reinforcement Learning for Flexible Operation & Maintenance Optimization of a Nuclear Power Plant," *Sci. Direct*, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S294992672300001X. [Accessed: Mar. 23, 2025].
6. X. Guo et al., "Deep Learning for Reward Design to Improve Monte Carlo Tree Search in ATARI Games," *IEEE Trans. Comput. Intell. AI Games*, vol. 8, no. 1, 2016. [Online]. Available: https://web.eecs.umich.edu/~honglak/ijcai2016-rewardDesign.pdf. [Accessed: Mar. 23, 2025].
7. "Beyond Games: A Systematic Review of Neural Monte Carlo Tree Search Methods," *SpringerLink*, 2023. [Online]. Available: https://link.springer.com/article/10.1007/s10489-023-05240-w. [Accessed: Mar. 23, 2025].
8. S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997. [Online]. Available: https://deeplearning.cs.cmu.edu/S23/document/readings/LSTM.pdf. [Accessed: Mar. 23, 2025].
9. H. Sak et al., "Long Short-Term Memory Recurrent Neural Network Architectures for Large Scale Acoustic Modeling," *IEEE Trans. Audio, Speech, Lang. Process.*, vol. 22, no. 4, pp. 809–818, 2014. [Online]. Available: https://research.google.com/pubs/archive/43905.pdf. [Accessed: Mar. 23, 2025].
10. "A Tutorial into Long Short-Term Memory Recurrent Neural Networks," *arXiv*, 2019. [Online]. Available: https://arxiv.org/abs/1909.09586. [Accessed: Mar. 23, 2025].
11. H. Sak et al., "Long Short-Term Memory Recurrent Neural Network Architectures for Large Scale Acoustic Modeling," *IEEE Trans. Audio, Speech, Lang. Process.*, vol. 22, no. 4, pp. 809–818, 2014. [Online]. Available: https://research.google.com/pubs/archive/43905.pdf. [Accessed: Mar. 23, 2025].
12. "RNN-LSTM: From Applications to Modeling Techniques and Beyond," *Sci. Direct*, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1319157824001575. [Accessed: Mar. 23, 2025].