Decoding the Chameleon Game

Anonymous submission

Abstract

The Chameleon game is a challenging word association activity where players are given a secret word and must respond with words relevant to that secret word. It requires strategic thinking and deduction. The Chameleon must cleverly guess the secret keyword in this game while avoiding suspicion. Our research aims to create an advanced AI model that can play the Chameleon game from both perspectives: as the Chameleon and as a Human. This AI is designed to guess secret keywords based on the information provided by the players, choose the best strategies to avoid detection as the Chameleon, identify and vote for the most suspicious players when acting as a Human, and learn from previous games to improve its performance.

1 Introduction

Word association games, where players are given a word (the cue) and must provide the first word that comes to mind (De Deyne et al. 2019) have gained significant traction due to their engaging and strategic nature, offering both entertainment and cognitive benefits. These games, which involve players generating words in response to given cues, enhance various cognitive functions such as attentional control, working memory, and problem-solving (Ornaghi, Brockmeier, and Gavazzi 2011). Researchs shows that games (i.e. chess and word association) can improve strategic thinking, memory, and creativity (Martinez, Gimenes, and Lambert 2023)(Mercier and Lubart 2021). While traditional word association games provide valuable cognitive and social benefits, they often rely on in-person interactions, which can be hindered by social isolation or geographical distance. Although wireless technology can partially address these challenges by facilitating remote play, finding a sufficient number of players to participate simultaneously remains a significant obstacle.

To address this, we developed an artificial intelligence (AI) model for the Chameleon game, a variation of word association where players must deduce a secret keyword while the Chameleon blends in by providing clues. By integrating techniques (e.g. Word2Vec, GloVe embedding, Naive Bayes, and Feed Forward Neural Network), the AI will learn from past gameplays and adapt to players' strategies over time. This approach not only replicates the interactive challenge of the original game but also creates a dynamic and evolving system that offers increasingly sophisticated gameplays, bridging social gaps and enhancing cognitive engagement.

The paper is organized into following sections. Section 2 reviews precedent works in applying AI to board games. Section 3 provides an overview of the game. Section 4 describes the database structure. Section 5 outlines the attributes assigned to the Chameleon, while Section 6 discusses the voting process. Section 7 explains how the Chameleon guesses the keywords. Section 8 covers the training process, and Section 9 focuses on testing. Finally, Section 10 presents the conclusions and suggests directions for future work.

2 Related Works

In board games, AI has demonstrated remarkable advancements through mastering complex games like Go, Chess, Poker, and Shogi. In "Mastering the game of Go without human knowledge", Silver et al. demonstrated how deep reinforcement learning can achieve superhuman performance through self-play (a method where an AI trains by competing against itself, learning and improving through repeated games), and highlighted AI's potential in strategic environments (Silver et al. 2017). Similarly, in different research, Silver and Hubert demonstrate that general reinforcement learning algorithms using self-play and extensive training enable AI to challenge humans in Chess and Shogi (Silver et al. 2018). Additionally, Brown and Sandholm developed Pluribus, an AI model that achieved superhuman performance in six-player no-limit Texas hold 'em poker, as detailed in their research (Brown and Sandholm 2019). These studies highlight AI's transformative impact on strategic board games, demonstrating how self-learning algorithms achieve unmatched proficiency through rigorous training.

In word association games, AI applications are increasingly proving their values. Shi et al. used a black-box model to predict the difficulty of word games like Wordle, showing how AI can enhance players' experience and game design by analyzing challenge complexity (Shi et al. 2024). Additionally, research by Richards and Amir focuses on opponent modeling in Scrabble (Richards and Amir 2007). Their work investigates how AI can strategically predict and respond to opponents' moves, providing deeper insights into game dynamics and improving competitive play. In another research, Hasan and Gan developed an unsupervised

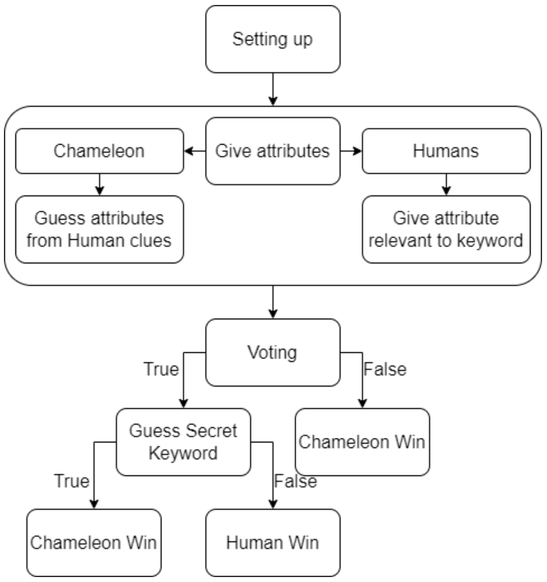

Figure 1: Game Component Diagram

adaptive brain-computer interface that improves gameplay in Hangman by learning from user brain signals (Hasan and Gan 2012). Their system demonstrated enhanced accuracy and efficiency compared to non-adaptive methods. Together, these studies exemplify the growing role of AI in understanding and mastering word association games, from predicting game difficulty to modeling opponents' strategies.

While AI models are commonly applied to many board games, their uses in the Chameleon game, particularly in developing strategies for both the Chameleon and the Humans, is unique. Our research addresses these challenges by creating AI models that focus on the most crucial phases for both roles. The most critical phases for the Chameleon are giving attributes and guessing the keyword. The most important phase for the Humans is voting on who they believe the Chameleon is.

3 About the Chameleon game

3.1 Game Rules

The Chameleon game is designed for 4-8 players. In this study, we studied a six-player version under the food theme. A six-player setup balances the game's complexity, providing enough interaction to challenge the Chameleon while keeping the deduction process engaging for Humans. The food theme works well because it draws from everyday experiences, making it easy for players to offer specific, recognizable attributes like flavors, textures, or common dishes (Meigs 1987). Other themes, such as animals, movies, colors, geography, or sports, can also be engaging options; however, their accessibility may vary based on players' knowledge and interests, particularly for younger players. Players are randomly assigned as either Humans or the Chameleon. All players receive a Code Card, except one who gets the Chameleon Card. A Topic Card with sixteen food-related words is placed face-up. Humans use a grid reference from the Code Card to find their word, while the Chameleon, who does not know the word, must infer it from context.

Players take turns giving one-word clues related to the secret word. After all clues are given, Humans vote to identify the Chameleon. If the Chameleon is identified, they get one chance to guess the secret word. Correct guesses mean a win for the Chameleon; incorrect guesses mean a win for the Humans. If the Chameleon is not identified, they win the round. The game is played over multiple rounds, with points tallied to determine the overall winner.

3.2 Versions of Chameleon Games

The Chameleon game, a social deduction board game, has multiple versions, each offering unique gameplay elements. The simple version is more straightforward, allowing players to quickly choose and declare attributes without concern for order or duplication. Voting in this version is based on identifying which attributes seem less relevant to the keyword.

The complex version introduces more strategic depth. Players have time to think carefully before declaring unique attributes. Voting considers various aspects, like player behavior and the relevance of attributes across multiple keywords. In this research, we chose to focus on the harder version of the game, as mastering the more challenging aspects implies proficiency in simpler versions.

3.3 Setting Up the Game

In the setup phase, the game runner randomly selects a keyword from a predefined list, which becomes the game's central theme. Players are then assigned roles as either Humans or the Chameleon. Humans are aware of the keyword, while the Chameleon knows only the list of possible keywords and must deduce the chosen one from the clues given by the Humans. A topic card with 16 keywords in a 4x4 grid (Table 2) is used for this purpose. The game runner selects one keyword and assigns roles to six players: one Chameleon and five Humans. Humans provide clues related to the keyword, while the Chameleon tries to guess the keyword without revealing their identity.

4 Database Structure

The database for the Chameleon game is structured to store attributes, which are properties of things and ways humans might describe them. Their relevance rate to each keyword is built on the idea of the Word2Vec model, which represents words in vector form (Jatnika, Bijaksana, and Suryani 2019). A vector represents attributes in a 16-dimensional space, where each element indicates the relevance of that attribute to the corresponding keyword. When adding an attribute to the database, the default vector will be initialized with zeros, indicating no relevance to any keyword recorded. After matches, as the model learns from gameplay data, these vectors can be updated to reflect the true relevance of each attribute to the corresponding keywords.

For example, the "Attribute 1" is initially added with a zero vector, indicating no relevance to any of the keywords. However, if "Attribute 1" is found to be relevant to "Pizza" and "Cake," the relevance value for "Pizza" and "Cake" in the vector will be adjusted accordingly. As the game progresses and more data is gathered, these relevance values will continuously refine, allowing the model to better understand the relationships between different attributes and keywords. This helps the technique make more informed decisions during gameplay, improving its ability to identify the keyword accurately or blend in as the Chameleon.

5 Giving attribute as Chameleon

5.1 Getting and Handling Input Attributes

Starting the attributes phase, each human will give an attribute. When it is the Chameleon's turn, the model will take the record of attributes provided by previous players as input to the model and, at the same time, use the database as a reference system to make predictions. When the AI model encounters a new attribute not present in the database, it will create a new entry for this attribute. This approach ensures that the model can handle unknown attributes and start with a neutral baseline. On the other hand, if the attributes already exist in the database, the AI model will retrieve the pre-existing vectors for these attributes.

5.2 Finding Attributes Intersection

After compiling the list of attributes given by the players, the AI model uses the data in the database to determine relevance. In this database, a value of 0 indicates no relevance between the meanings of attributes and keywords, while a value greater than 0 indicates at least some relevance between them. The model then searches for keywords relevant to all the attributes other players provide. Suppose the AI model cannot find any keywords that are relevant to all attributes (possibly due to missing data or because the combination of relevant keywords and attributes has never appeared in previous games). In that case, the model will adapt its approach. It will progressively relax the requirement by reducing the number of attributes it considers, checking keywords relevant to the number of previous players $n-1$ attributes, then $n-2$ attributes, and so on. This process continues until the AI model identifies at least one keyword that satisfies the requirement. By using this adaptive strategy, the AI ensures that it can always find at least a suitable keyword, even when the data is incomplete or when encountering novel combinations of attributes and keywords. This approach helps the AI maintain robust performance and effective gameplay.

To illustrate how the Chameleon identifies the intersection of attributes, consider the scenario where the Chameleon is in position 4, requiring it to analyze the previous three attributes. These attributes are represented as vectors, with each position corresponding to a specific keyword and the value indicating the relevance between the attribute and the keyword.

For example, assume the following attribute vectors:

- **Attribute 1:** [2, 1, 0, 3, 0, 2, 3, 0, 1, 2, 0, 3, 0, 1, 0, 3]
- **Attribute 2:** [1, 2, 0, 3, 0, 1, 2, 0, 0, 0, 3, 2, 3, 0, 1, 2]
- **Attribute 3:** [0, 3, 2, 1, 1, 3, 2, 3, 0, 1, 2, 3, 2, 3, 1, 0]

The model compares the vectors at each position (or keyword ID) to find the intersection. The intersection occurs where all three attributes share a common keyword, indicated by non-zero values across the vectors at that specific position. In this example, the keywords with index 2, 4, 6, 7, and 12 are identified as the intersection, where the Chameleon recognizes shared relevance among the previous attributes. This process helps the Chameleon narrow down potential keywords that align with the collective input from previous rounds.

5.3 Calculating Attributes Probability

After obtaining the intersection of the attributes from previous humans, we remove any data not present in this intersection column. This step helps avoid data distractions that might come from attributes not in the intersection column but with high weight. Such data can significantly reduce the accuracy of the intersection column, which affects both the probability calculation and the subsequent spreading step. This process involves creating a new table where values irrelevant to the maximum number of available attributes are set to zero. By doing this, the model focuses solely on the relevant data, reducing potential distractions from unrelated attributes. This refined dataset enhances the technique's ability to make precise predictions and blend in effectively as the Chameleon.

In the previous example, Attribute 1 is represented by the vector [2, 1, 0, 3, 0, 2, 3, 0, 1, 2, 0, 3, 0, 1, 0, 3]. The intersection was identified at index 2, 4, 6, 7, and 12. In the new table, only the values at these positions are retained, while all other values are set to zero. Consequently, the updated attribute vector would be [0, 1, 0, 3, 0, 2, 3, 0, 0, 0, 0, 3, 0, 0, 0, 0]. This process is applied to all attributes in the database, ensuring that only the relevant values corresponding to the identified intersections are kept.

We calculate the probability of relevant attributes for each keyword to account for the relevance of attributes to keywords and avoid interference from irrelevant data. This helps ensure that only significant data is considered in the spreading calculation. The probability of relevance of each keyword is determined using the following formula:

$$\text{Probability} = \frac{\text{Favorable Cases}}{\text{Possible Cases}} \tag{1}$$

(Blitzstein and Hwang 2019)

where:

- **Favorable Cases** = Number of cells (representing the relevance value between attribute and keyword).
- **Possible Cases** = Total number of rows (the total number of attributes considered).

After calculating the probability for each cell, we obtain a table showing the relevance of each attribute to each keyword as a percentage. This table helps to understand which keywords are more likely to be associated with the given attributes based on these probabilities. The percentage reflects the proportion of attributes relevant to each keyword compared to the total number of attributes available. Keywords with a relevance percentage greater than 0.01 are included in the spreading calculation. This threshold ensures that only keywords with a meaningful number of relevant attributes are considered, thus reducing noise and improving the accuracy of keyword selection.

For example, after applying the probability formula to Attribute 1, the resulting vector is Attribute 1 probabilities: [0, 0.08, 0, 0.25, 0, 0.17, 0.25, 0, 0, 0, 0, 0.25, 0, 0, 0, 0]. Following the rule above that considers an attribute to be relevant to a keyword if its percentage exceeds 0.01, we observe that there are five values in indices 2, 4, 6, 7, and 12 that meet this criteria. Therefore, the spreading of attribute 1 corresponds to five keywords. After calculating the keyword spreading for all attributes, the table is then sorted by the number of keywords each attribute spreads, highlighting the significance of each attribute in the selection process.

To avoid predictable patterns for other players, the AI randomly selects between attributes with equal relevance within the range of intersection (spreading). This randomness ensures that the AI's choices are not easily anticipated by human players, maintaining the element of surprise and making the game more challenging for Humans to guess who is Chameleon.

5.4 Updating the Database

After the match, the model records the attributes given by human players and updates the database accordingly. For each attribute provided by the players that is relevant to the keyword used in the match, the corresponding value in the database increases by 1. This increment represents an increase in the relevance of that attribute to the keyword, indicating that real players are more likely to choose that attribute when considering the keyword. This continual learning process allows the model to refine its understanding of the relationships between attributes and keywords. By incrementally updating the database after each match, the technique becomes better at making accurate guesses and effectively blending in as the Chameleon. This adaptive learning enhances the model's performance and its ability to participate in the game in a more human-like manner.

6 Voting

By the Voting phase, each player is presented with key information that includes the six attributes provided by all six players, one of whom is the Chameleon. Five Humans will know what the secret keyword is. These attributes are revealed in a specific order, corresponding to the order of the players. Each human player must assess these attributes' relation to the keyword to determine who might be the Chameleon. The Chameleon, however, operates under different conditions. In the "Guessing Keyword as Chameleon" phase, the Chameleon can deduce the secret keyword based on the attributes provided by the other players. This deduction occurs before the Voting phase and does not rely on information from the Voting process itself. During this phase, the Chameleon uses the attributes given by other players to infer the keyword and strategically craft responses to blend in with the other players.

To enable humans to identify the Chameleon and allow the Chameleon to use the network during voting to decrease suspicion and shift focus away from themselves, we employ a Neural Network (Guresen and Kayakutlu 2011). This network is a massively parallel combination of simple processing units that can acquire knowledge from the environment through a learning process and store that knowledge in its connections. The network structure is as follows:

6.1 Input Layer

The Neural Network's input layer is designed to process the information available to each player. Specifically, the player is given six attributes—one from each of the six players, including the Chameleon—and a secret keyword. The first step in this process is to calculate the distance between each of the six attributes and the keyword by using cosine similarity:

$$\text{cosine similarity} = \frac{\vec{A} \cdot \vec{B}}{\|\vec{A}\|\|\vec{B}\|} \tag{2}$$

(Stewart 2012)

These distances are essential as they measure how closely related each attribute is to the keyword, reflecting the likelihood that the attribute was chosen by someone who knows the keyword or by the Chameleon attempting to blend in. The resulting six distances are fed into the Neural Network as input features.

6.2 Hidden Layer

The Neural Network is designed with multiple layers, each playing a critical role in processing the input data. The network starts by receiving six inputs from the input layer, which represent the calculated distances between the attributes and the keyword. The hidden layers consist of 1024 neurons each. As the data passes through these layers, it is progressively refined, allowing the network to capture

Neural Network	Training Minimum Loss	Training Maximum Accuracy	Testing Accuracy
6-1-6+DO(0.2)+BN	1.70	29.58	13.30
6-2-6+DO(0.2)+BN	1.64	36.77	15.14
6-1024-6+DO(0.2)+BN	1.46	57.83	16.22
6-1024-512-6+DO(0.2)+BN	1.39	64.83	18.34
6-1024-512-256-6+DO(0.2)+BN	1.30	75.29	25.32
6-1024-512-256-128-6+DO(0.2)+BN	**1.24**	**80.84**	32.75
6-1024-1024-1024-1024-6+DO(0.2)+BN	1.39	64.43	**64.25**
6-1024-512-256-512-1024-6+DO(0.2)+BN	1.39	64.43	**64.25**

Table 1: Comparison between different Neural Network

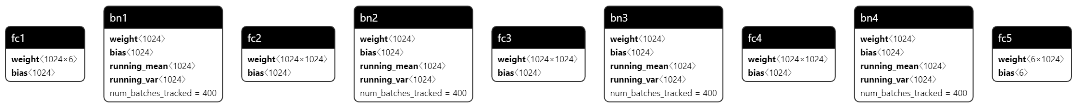

Figure 2: Illustration of Neural Network

increasingly nuanced patterns and relationships within the data (Uzair and Jamil 2020). ReLU (Rectified Linear Unit) is used as the activation function between these layers (Arora et al. 2016). The ReLU function is defined as:

$$f(x) = x^{+} = \max(0, x) = \frac{x + |x|}{2} = \begin{cases} x & \text{if } x > 0, \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

(Nair and Hinton 2010)

ReLU effectively introduces non-linearity into the model, which is essential for solving complex classification problems. It allows the network to pass only positive values to the next layer while setting negative values to zero, enabling the network to learn diverse patterns without the issue of vanishing gradients that can occur with other activation functions, such as the sigmoid function. Unlike the sigmoid function, which compresses outputs between 0 and 1 and is typically used in binary classification, ReLU allows the network to learn a broader range of patterns by not restricting the output values (Pratiwi et al. 2020). To further enhance the network's stability and performance, we incorporated Batch Normalization (BN) between the hidden layers (Bjorck et al. 2018). BN normalizes the input to each layer within a mini-batch, reducing internal covariate shift and speeding up the training process. This normalization enables the use of higher learning rates, leading to faster convergence and improved overall accuracy. Additionally, BN acts as a form of regularization, helping to mitigate overfitting by ensuring that the network's learning process is more robust.

Despite these improvements, overfitting remained a concern, particularly as indicated by the significant difference between training accuracy and test accuracy (as shown in Table 1). To address this issue, we employed Dropout with a rate of 0.2. Dropout randomly deactivates 20% of neurons during each training iteration, forcing the network to generalize better by preventing it from relying too heavily on specific neurons (Baldi and Sadowski 2013). This technique effectively reduces overfitting, leading to improved performance on unseen data. In parallel with Dropout, we also applied L2 Regularization across the entire model. L2 regularization adds a penalty proportional to the square of the magnitude of the model parameters, encouraging the network to maintain smaller weights and avoid overfitting (Shi et al. 2019). While L2 regularization provides a baseline level of regularization for all models, the introduction of Dropout further enhances the model's ability to avoid overfitting by promoting diversity in the learned representations.

After extensive testing and tuning, this specific network configuration, consisting of 6 input neurons, 4 hidden layers with 1024 neurons each, and 6 output neurons, using ReLU as the activation function, L2 regularization, dropout (rate 0.2), and batch normalization, was found to provide the best performance for the task, as shown in Table1. This setup enables the network to effectively analyze the input data and make accurate classifications, whether for identifying the Chameleon or helping the Chameleon reduce suspicion during gameplay.

6.3 Output Layer

The output layer of the Neural Network is designed as a multi-class classification layer. It generates a prediction that identifies which player is most likely to be the Chameleon. The prediction is represented as one of six possible integers corresponding to the six players in the game. Each integer in the list [1, 2, 3, 4, 5, 6] represents a specific player, with the model's objective being to accurately predict the player most likely to be the Chameleon.

In this model, we use Cross Entropy Loss as the loss function because it is the most effective and commonly used method for multi-class classification problems (Hinton, Osindero, and Teh 2006). Cross Entropy Loss measures the performance of the classification model whose output is a probability value between 0 and 1. It calculates the difference between the actual label and the predicted probability distribution, penalizing predictions that are further from the true label. This loss function is particularly suitable for multi-class classification because it considers the probability distribution over all classes, ensuring that the model not only predicts the correct class but also assigns low probabilities to incorrect classes. By minimizing Cross Entropy Loss, the model is trained to provide more accurate and confident predictions across multiple classes, making it an essential component in developing a reliable AI model for tasks like identifying the Chameleon in the game.

7 Guessing keyword as Chameleon

After completing the giving attributes step, the Chameleon must guess the keyword based on the 5 attributes provided by the 5 humans. The process of finding the intersection of these 5 attributes is similar to how the attribute intersection is determined in the "Giving Attributes as the Chameleon" part. This process results in a table showing how many attributes intersect at each keyword.

After that, we calculate attribute probabilities, particularly in the context of the "Giving Attributes as Chameleon" step. The database is refined by setting any data that is below the highest number of intersecting attributes to zero. Subsequently, the refined dataset is utilized to compute the probability table for the 5 attributes. After determining the probability of each attribute given a relevant keyword, the Naive Bayes classifier—a simple learning algorithm that utilizes Bayes' rule along with the strong assumption that the attributes are conditionally independent given the class—is employed to estimate the probability of each keyword being the secret one (Webb, Keogh, and Miikkulainen 2010). Initially, each keyword $P(k)$ is set to $\frac{1}{16}$. For each keyword k, the model calculates the conditional probability of each attribute A_i given the keyword: $P(A_i|k)$. Using the Naive Bayes formula, the combined probability of all attributes given the keyword is determined. Specifically, the model computes:

$$P(k|A_1, A_2, \ldots, A_n) = P(k) \cdot \prod_{i=1}^{n} P(A_i|k) \tag{4}$$

(Alpaydin 2020)

The initial probability will be $P(k) = \frac{1}{n}$, for n keywords. In this case, we substitute $P(k) = \frac{1}{16}$, since we have 16 keywords and $A_n = A_5$ since we only have 5 attributes. The formula then becomes:

$$P(k|A_1, A_2, \ldots, A_5) = P(k) \cdot \prod_{i=1}^{5} P(A_i|k) \tag{5}$$

To avoid underflow, which occurs when a calculation gets so close to 0 that the computer treats it as 0, we take the natural log of both sides of the formula (note: the logarithm can be of any base):

$$\ln(P(k|A_1, A_2, \ldots, A_5)) = \ln\left(\frac{1}{16} \cdot \prod_{i=1}^{5} P(A_i|k)\right) \tag{6}$$

Using the logarithm product rule ($\log_b(x \cdot y) = \log_b(x) + \log_b(y)$,), the formula simplifies to:

$$\ln(P(k|A_1, A_2, \ldots, A_5)) = \ln\left(\frac{1}{16}\right) + \sum_{i=1}^{5} \ln(P(A_i|k)) \tag{7}$$

We use the natural logarithm instead of the exponential function because the natural logarithm helps manage numerical stability when dealing with very large or small values (Haarhoff, Kok, and Wilke 2013). Specifically, the natural logarithm allows us to avoid underflow issues that can occur with the exponential function. While the exponential function can produce extremely large numbers for large inputs, the natural logarithm provides a way to work with these values in a more controlled manner, preventing the numerical problems associated with extremely large or small values.

After calculating the naive probabilities of the 16 keywords, the Chameleon will guess the keyword with the highest probability as the secret word for the game. If more than one keyword has the same Naive Bayes probability value, the Chameleon will randomly select one among them. This approach ensures that the Chameleon makes an informed guess based on the calculated probabilities, while also introducing an element of unpredictability when probabilities are tied.

8 Training Model

8.1 Training Database

In the training phase, our model will interact with these simulators five human simulators that guess attributes for the keywords using the method outlined below. We first used the GloVe (Global Vectors for Word Representation) model to generate an attribute list (GloVe list) (Pennington, Socher, and Manning 2014). This unsupervised learning algorithm, developed at Stanford University, generates word embedding by analyzing statistical co-occurrence patterns in large text corpora. The GloVe model captures semantic relationships between words by embedding them in a continuous vector space, where the distance between vectors reflects their semantic similarity. Words with similar meanings or contextual usage are positioned closely, enabling the identification and quantification of relationships between terms.

Using the GloVe model, we calculate the cosine similarity between each attribute in the GloVe list and each of the 16 keywords (Mikolov 2013). In the GloVe model, cosine similarity values close to 1 indicate high similarity, while values near 0 represent minimal relevance. Starting at 0.50 ensures that attributes have at least moderate semantic similarity to the keywords, avoiding those that are too dissimilar. This threshold balances diversity and relevance, ensuring attributes remain distinct without drifting too far from the keywords' meaning. We then divided these attributes into six pools based on the cosine similarity value. Table 2 represents keywords with associated cosine similarity values.

In addition, we eliminate all attributes that are either too similar to a keyword or have a singular form already present. This ensures that the remaining attributes have a distinct semantic distance from the keywords and other attributes, al-

Pool	Cosine Similarity	$\cos^{-1}(\mathbf{x})$	Weight	Pool Percentage
1	0.50	60.0°	18.90	13.90%
2	0.55	56.3°	20.14	14.86%
3	0.60	53.1°	21.36	15.76%
4	0.65	49.5°	22.91	16.90%
5	0.70	45.6°	24.87	18.34%
6	0.75	41.4°	27.39	20.20%

Table 2: Weight of Pool

lowing them to provide meaningful and diverse information. The higher the pool number, the more relevant the attribute is to the keyword, indicating a stronger semantic similarity. However, classification stops at Pool 6 because cosine similarity scores above 0.80 are rare for some attribute-keyword pairs, making it impractical to create additional pools beyond this threshold.

To determine which attributes the five human simulators provide in the database, we apply a random sampling method that uses weights based on cosine similarity scores. Words with higher cosine similarity scores will be assigned greater weights, increasing their probability of being selected from the pool. This ensures that data points are appropriately chosen according to the similarity metrics.

The weights for each pool are determined using the inverse cosine function ($\cos^{-1}$) of the cosine similarity scores, given in degrees as can be seen in Table 2:

To calculate the weight, we use the equation:

$$\text{Weight of pool } 1 \times 60.0^\circ \approx \ldots \approx \text{Weight of pool } 6 \times 41.4^\circ \tag{8}$$

The corresponding weights for each pool are shown in Table 2.

This weighting ensures that the most semantically relevant words (those in higher pools) are more likely to be chosen, reflecting their closer relationship to the keywords. By employing this method, the analysis effectively leverages the GloVe model's ability to capture nuanced semantic relationships, enabling a more precise and contextually aware interpretation of textual data.

To determine which attribute to use, we employ a two-layer random selection process.

Layer 1: Distribute Words into Pools
Attributes are first distributed into different pools. Within each pool, attributes are chosen randomly, with each having an equal probability of appearing.

Layer 2: Randomly Select Pool
A pool is then randomly selected based on the percentage as in Table 2:

$$\text{Pool 1: } \frac{18.90}{135.57} = 13.90\% \text{ chance of being selected}$$

$$\vdots$$

$$\text{Pool 6: } \frac{27.39}{135.57} = 20.20\% \text{ chance of being selected}$$

The numerator is the weight of a pool, and the denominator is the total weight of all pools.

If two attributes are identified as similar, the process is repeated to generate a new list of six random keywords. This revised list is then distributed to five human simulators, ensuring that the final selection benefits from a combination of algorithmic randomness and human judgment.

By using the algorithm described above, we enhance the performance and reliability of the model by conducting a random sampling process 30,000 times to generate a diverse set of samples. This extensive sampling is designed to capture various data dimensions, ensuring that the model is exposed to a wide range of scenarios and variations. Figure 3 shows how the accuracy of guessing the secret word as the Chameleon changes over time.

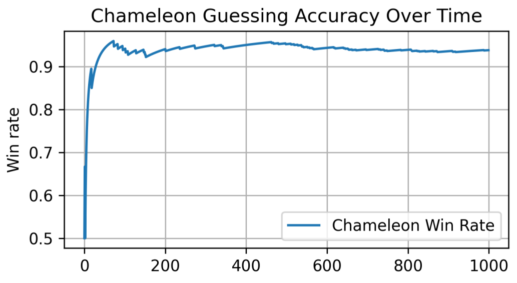

Figure 3: Chameleon Guessing Accuracy Overtime

This approach improves the model's robustness by allowing it to learn from a comprehensive dataset rather than being limited to a narrow or restricted subset. By mitigating overfitting and enhancing the model's generalizability, the Chameleon model achieves a high guessing accuracy rate of over 95% with the training dataset, demonstrating its effectiveness in predicting keywords. Initially, the model's win rate starts low but rapidly climbs to over 90% as it gains more experience through the training process. Even after 1,000 games, the win rate keeps fluctuating slightly around this high value, indicating consistent performance. The large-scale sampling process further validates this performance, ensuring consistent and reliable accuracy across diverse data conditions.

8.2 Training Neural Network

Although we generated data from 30,000 games in the training database process, we used only the last 10,000 as training data for the neural network. The first 20,000 games were primarily used for populating the database, providing a robust foundation for the model's training. The goal was to ensure that the database captured a wide range of scenarios, player interactions, and possible outcomes, thereby enhancing the model's ability to generalize effectively.

Following this initialization, the model was trained across ten sessions, with each session consisting of 400 epochs. The figure 4 corresponds to the tenth training session, reflecting the model's performance after it had undergone significant refinement through the preceding nine sessions. This approach allowed the model to be improved progressively, minimizing loss and maximizing accuracy with each subsequent session. The rigorous training process, supported by a well-initialized and diverse dataset, contributed to the model's robust performance in predicting the secret word as the Chameleon, achieving high accuracy.

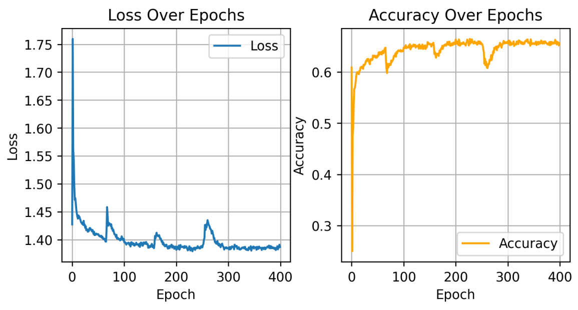

Figure 4: Loss graph and Accuracy graph

9 Testing Model

After the training phase, we created an additional 2,000 games for testing. In these test games, the Chameleon achieved an impressive win rate of 93% and was voted out in 64% of the games.

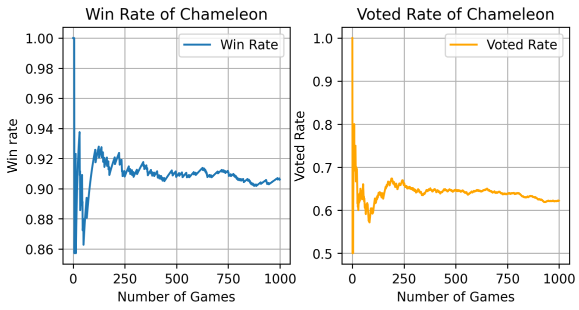

Figure 5: Chameleon's Win Rate and Voted Rate after Training

The bot selects a keyword based on the rule outlined in the training section. The Chameleon's high win rate demonstrates its effectiveness in a controlled environment where the bots, due to their predictable and informative cues, allow the Chameleon to blend in and avoid detection more easily. However, this success is expected to drop significantly in a different configuration—specifically, when there are five human players and one bot. Human players, with their less predictable and more varied hints, introduce greater complexity and additional variables, making it more challenging for the Chameleon to remain undetected.

The moderate voting rate of 64% likely results from the stochastic nature of attribute selection among the bots, introducing a degree of randomness that may assist the Chameleon in evading detection. The interplay of these factors is depicted in the figure 5, illustrating how the Chameleon's win rate and the effectiveness of the voting process evolve under different conditions.

10 Conclusion

Developing the AI model for Chameleon Games has revealed several strengths and areas for improvement. A significant challenge was the lack of readily available, real-world datasets recorded after each game. This limitation highlights the need for future efforts to expand the dataset by collecting more real game data, enhancing the model's ability to simulate and predict player behavior more accurately.

Despite these challenges, the model demonstrates promising capabilities. The model is designed to learn and adapt after each match, allowing it to refine its accuracy and effectiveness over time continuously. With these strengths and a commitment to addressing the identified limitations, the AI model has the potential to become a robust and versatile tool for simulating and analyzing gameplay in Chameleon Games and beyond.

Additionally, the current implementation uses a simple neural network architecture. Future work should focus on exploring more modern and sophisticated neural network models to improve the model's performance and adaptability.

References

Alpaydin, E. 2020. *Introduction to machine learning*. MIT press.

Arora, R.; Basu, A.; Mianjy, P.; and Mukherjee, A. 2016. Understanding deep neural networks with rectified linear units. *arXiv preprint arXiv:1611.01491*.

Baldi, P.; and Sadowski, P. J. 2013. Understanding dropout. *Advances in neural information processing systems*, 26.

Bjorck, N.; Gomes, C. P.; Selman, B.; and Weinberger, K. Q. 2018. Understanding batch normalization. *Advances in neural information processing systems*, 31.

Blitzstein, J. K.; and Hwang, J. 2019. *Introduction to probability*. Chapman and Hall/CRC.

Brown, N.; and Sandholm, T. 2019. Superhuman AI for multiplayer poker. *Science*, 365(6456): 885–890.

De Deyne, S.; Navarro, D. J.; Perfors, A.; Brysbaert, M.; and Storms, G. 2019. The "Small World of Words" English word association norms for over 12,000 cue words. *Behavior research methods*, 51: 987–1006.

Guresen, E.; and Kayakutlu, G. 2011. Definition of artificial neural networks with comparison to other networks. *Procedia Computer Science*, 3: 426–433.

Haarhoff, L. J.; Kok, S.; and Wilke, D. N. 2013. Numerical strategies to reduce the effect of ill-conditioned correlation matrices and underflow errors in Kriging. *Journal of Mechanical Design*, 135(4): 044502.

Hasan, B. A. S.; and Gan, J. Q. 2012. Hangman BCI: An unsupervised adaptive self-paced brain–computer interface for playing games. *Computers in biology and medicine*, 42(5): 598–606.

Hinton, G. E.; Osindero, S.; and Teh, Y.-W. 2006. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7): 1527–1554.

Jatnika, D.; Bijaksana, M. A.; and Suryani, A. A. 2019. Word2vec model analysis for semantic similarities in english words. *Procedia Computer Science*, 157: 160–167.

Martinez, L.; Gimenes, M.; and Lambert, E. 2023. Video games and board games: Effects of playing practice on cognition. *PLoS One*, 18(3): e0283654.

Meigs, A. 1987. Food as a cultural construction. *Food and foodways*, 2(1): 341–357.

Mercier, M.; and Lubart, T. 2021. The effects of board games on creative potential. *The Journal of Creative Behavior*, 55(3): 875–885.

Mikolov, T. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Nair, V.; and Hinton, G. E. 2010. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, 807–814.

Ornaghi, V.; Brockmeier, J.; and Gavazzi, I. G. 2011. The role of language games in children's understanding of mental states: A training study. *Journal of cognition and development*, 12(2): 239–259.

Pennington, J.; Socher, R.; and Manning, C. D. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 1532–1543.

Pratiwi, H.; Windarto, A. P.; Susliansyah, S.; Aria, R. R.; Susilowati, S.; Rahayu, L. K.; Fitriani, Y.; Merdekawati, A.; and Rahadjeng, I. R. 2020. Sigmoid activation function in selecting the best model of artificial neural networks. In *Journal of Physics: Conference Series*, volume 1471, 012010. IOP Publishing.

Richards, M.; and Amir, E. 2007. Opponent Modeling in Scrabble. In *IJCAI*, 1482–1487.

Shi, G.; Zhang, J.; Li, H.; and Wang, C. 2019. Enhance the performance of deep neural networks via L2 regularization on the input of activations. *Neural Processing Letters*, 50: 57–75.

Shi, L.; Chen, Y.; Lin, J.; Chen, X.; and Dai, G. 2024. A black-box model for predicting difficulty of word puzzle games: a case study of Wordle. *Knowledge and Information Systems*, 66(3): 1729–1750.

Silver, D.; Hubert, T.; Schrittwieser, J.; Antonoglou, I.; Lai, M.; Guez, A.; Lanctot, M.; Sifre, L.; Kumaran, D.; Graepel, T.; et al. 2018. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362(6419): 1140–1144.

Silver, D.; Schrittwieser, J.; Simonyan, K.; Antonoglou, I.; Huang, A.; Guez, A.; Hubert, T.; Baker, L.; Lai, M.; Bolton, A.; et al. 2017. Mastering the game of go without human knowledge. *nature*, 550(7676): 354–359.

Stewart, J. 2012. *Calculus: early transcendentals*. Cengage Learning.

Uzair, M.; and Jamil, N. 2020. Effects of hidden layers on the efficiency of neural networks. In *2020 IEEE 23rd international multitopic conference (INMIC)*, 1–6. IEEE.

Webb, G. I.; Keogh, E.; and Miikkulainen, R. 2010. Naïve Bayes. *Encyclopedia of machine learning*, 15(1): 713–714.