DeepClue: AI Powered Word Association Using NLP and Multi Agent Reinforcement Learning

EEE 486 Statistical Foundations of Natural Language Processing
Group 23
Buğra Kerem Özcan Selin Kasap Ayşe Selin Cin Mustafa Enes Erdem
20 May 2025

***Abstract*—Word association games like *Codenames* pose compelling problems in natural language processing (NLP) and multi agent reinforcement learning (MARL). In this work, we propose DeepClue, an advanced AI that is comprised of two collaborative agents that play an autonomous game of word association similar to *Codenames*. The Clue Giver produces short, one word hints meant to connect two or more target words, while the Guesser decodes these hints to correctly guess target words from a wide range of alternatives.**

In contrast to current methods that essentially build upon fixed semantic metrics or static heuristics, DeepClue combines strong semantic representation models (such as pretrained static embeddings like Word2Vec, GloVe, and transformer based contextual embeddings like BERT) with reinforcement learning methods, that is, Deep Q Networks (DQN) and Proximal Policy Optimization (PPO). In iterative self play, these agents develop communication strategies dynamically, adapting to gameplay results to increase clue effectiveness and interpretation accuracy.

We speculate that reinforcement learning self play encourages optimal, context dependent communication protocols, much like domain specific sublanguages optimally suited for gameplay. In this paper, we describe DeepClue's architecture and training process, situate it in the greater research context of NLP and MARL, and describe its potential contributions to more advanced and human like cooperative AI communication systems.

I. Introduction

Word association games serve as an experiment ground for artificial intelligence, as it combines linguistic information and strategic reasoning. One excellent example is *Codenames*, an engaging board game where one Spymaster provides one word clue to lead their partner to guess several associated words on the board while not picking up on "threatening" words that belong to one's competitor or an assassin [15].

Successfully playing Codenames demands creative yet accurate association within constraints, which tests natural language understanding (to understand semantic relations of words) as well as theory of mind reasoning (to predict what a partner is likely to interpret from a clue). Recent works on AI have started approaching this game as a standard of language based reasoning. Static measures of semantic similarity, e.g., basic cosine similarity of word embeddings, were used in early bot players of Codenames, to select clues [1]. Such approaches can identify related words, but lack flexibility: an optimal clue under one fixed similarity measure may mislead a human partner or break game rules subtly [4].

Later methods used more information (e.g., extensive lexical databases or co occurrence frequencies) and even transformer language models to enhance clue quality [2], [14]. But to this day, these systems still heavily depend on manually crafted scoring functions or one way analysis and do not acquire experience from gameplay. Unlike them, humans operating Codenames build up teamwork and mutual understandings gradually from repeated interactions, fine tuning their clue giving strategy to correlate it to that of their partner and, in fact, creating an ad hoc communication protocol.

This is the premise of our project: *Can two AI agents, using reinforcement learning, be taught to effectively communicate in a word association game and mimic human like clue giving behavior?* To investigate this, we propose an AI framework, DeepClue, in which a clue giver and guesser agent are co trained using multi agent reinforcement learning (MARL) to optimize performance in an environment based on Codenames. By casting clue creation and guessing into an interactive decision making problem, DeepClue enables the agents to create and establish their own protocol for semantic communication.

DeepClue is created by combining natural language processing methods like pre trained word embeddings and language models based on transformers and reinforcement learning algorithms. The clue giver, rather than using a pre defined formula, chooses clues based on an observed policy that makes decisions based on both current game state and predicted guesser behavior. The clue giver is thereafter reinforced using rewards based on guesser success in guessing correct target words. Concurrent to this process, the guesser learns to decode clues based on evolving clue giver strategy, which allows both agents to learn together. By integrating rich semantic representations into a reinforcement learning setting, DeepClue aims to close the gap between static semantic similarity approaches and emergent communication systems. On one end, we leverage strong pre trained language vectors, such as Word2Vec [7], GloVe [8], and BERT [9], to give agents an anchor grounding in word meanings. On the other end, under self play training, agents learn to adapt their use of language by optimizing for joint rewards, enabling novel and effective clue strategies to emerge that might differ from standard semantic similarity but prove extremely effective within task context.

The paper has four main contributions:

- **New AI Game Playing:** We propose DeepClue, an end to end completely self sufficient system composed of a guesser agent and clue giver to play an inspired game of Codenames. an end to end framework that learns

game strategies using self play reinforcement learning as opposed to fixed heuristics or data annotated by humans
- **NLP and RL integration:** We demonstrate that a reinforcement learning framework can be integrated using transformer based language models as well as distributional semantic models. Pre trained word embeddings are used as agents' semantic knowledge starting points, while reward driven learning allows them to refine clue selection abilities.
- **Emergent Communication Analysis:** We analyze communicative protocols that emerge among agents as they learn. Specifically, we look for any of these deviations, like instances of novel or unclear clues, and measure whether learned clues exhibit human like properties, like maximizing relevance to target words and not to nontargets. [12], [13].
- **Baselines and Benchmarking:** DeepClue is compared to currently used methods, for example, transformer based and embedding based clue generation approaches. While an extensive results presentation is beyond section scope, our framework is designed to be directly compared to pre established baselines, which allows for an even assessment of efficiency and success rate.

The rest of this paper is structured as follows: Section 2 provides an overview of related work in these four areas of interest to our work. In Section 3, DeepClue methodology is described, covering game formulation, model structure, and training process. In Section 4, experimental results and analysis are provided, and in Section 5, we conclude by summarizing implications to larger contexts and directions for future research.

II. Related Work

Our work draws on and intersects with multiple fields: (1) AI agents for linguistic games (and, in this case, especially Codenames); (2) clue generating methods based on semantic similarity; (3) word association using distributional semantic models and transformers; (4) reinforcement learning approaches to language problems; and (5) emergent language and multiagent communication. This section reviews representative work from each of these areas, identifying its strengths, weaknesses, as well as where it is complemented or challenged by DeepClue.

A. AI Agents for Word Association Games (Codenames)

While games like Go and Chess have long been used as AI milestones, word games present linguistic and everyday reasoning difficulties [5]. One such game of interest among these is Codenames as a cooperative AI task. The first paper to be published is that of Kim et al. [1], where an AI tournament for Codenames was organized and one of the earliest agents was designed for an abridged game variant thereof. In Kim et al.'s approach, both clue giver and guesser agents employed distributional word representations (Word2Vec and GloVe) to represent words and measure clue quality [1].

They created a baseline algorithm that requires that a chosen clue be closer in cosine similarity to all of the target words of the spymaster than to any other word on the board. By imposing this simple requirement, Kim et al. learned that when both agents used the same embedding model and strategy, the team could often solve for every target correctly, but when there was another strategy (or embedding) used, performance greatly degraded. This revealed an important issue: agents that optimize for the semantic measure of their own model struggle to work well together with agents that "think" differently. In other words, an fixed similarity based method may overfit to its creator's assumptions and be less robust to communication stylization changes [1].

Taking these observations into consideration, Jaramillo et al. [2] created Word Autobots, which was a pioneer using up to date deep learning to play Codenames. Their work employed six variants of agents to be classified into Codemaster (the clue giver) and Guesser roles. In addition to replicating standard agents based on NLP (like a TF–IDF weighted one and Naïve Bayes), it introduced a transformer based Codemaster that makes use of GPT 2 language model embeddings to generate and evaluate clues.

The transformer Guesser paired with an agent that used transformer Codemaster attained close to the strong Word2Vec+GloVe ensemble baseline's win rates using round robin testing. Surprisingly, even when compared to the static embedding baseline, the transformer agent's clues generally resulted in faster games (fewer turns to victory). With that agent's bad guessing, it turned out that, based on a small user study in their paper, participants slightly preferred to play using the transformer generated clues (seeing them as more "natural"). Based on these results, transformer language models can encode subtlety that imbues guesses with more of a humanlike feel, although it is still not possible to match the guesser using these guesses [2].

More recently, computational creativity has also been used to study Codenames. For example, Spendlove and Ventura [5], [6] analyze competitive word games like Codenames from a computational creativity perspective and consider clue creation to be an exercise in constrained creativity. They describe that clue givers utilize such strategies as using an analogy, metaphor, or wordplay to connect dissimilar sounding words, which indicate semantic similarity that is not merely direct semantic association.

These studies shed light on the cognitive principles of clue giving that can lead to more productive strategy spaces for AI, despite not creating a game playing AI in and of itself. Broadly speaking, research to date on playing agents for Codenames reveals that an elementary level of play is possible using purely semantic similarity, but also highlights just how central flexibility and mutual understanding become to clue based communication. In an attempt to simulate the gradual cooperative learning that teammates develop, DeepClue differs from previous approaches by training two agents explicitly together using reinforcement learning. We move from fixed, one size fits all clue metrics to an interactive, dynamic process

where the clue giver explicitly models what they think the guesser is likely to interpret (and vice versa) by enabling agents to learn from every game result.

B. Clue Generation with Semantic Similarity Methods

A central technical challenge in Codenames (and DeepClue) is automatic clue generation: how can an AI find a single word that meaningfully links multiple target words while avoiding others? Early approaches treated this as a search and ranking problem over a large vocabulary, guided by measures of semantic relatedness. As mentioned, Kim et al. [1] used simple yet intuitive heuristics based on cosine similarity between word vectors. In their criterion, a candidate clue c was considered valid if $\min_{t \in T} s(c,t) > \max_{b \in B} s(c,b)$, where T is the set of target words for the clue, B is the set of "bad" words (non targets) on the board, and $s(x,y)$ is a similarity score (typically the cosine similarity between the embedding vectors of words x and y). In other words, the clue had to be closer to every intended word than it was to any unintended word. If this condition was met, clues were then ranked by $\min_{t \in T} s(c,t)$ (preferring clues that have a strong connection even to the least similar target). This approach, using pretrained embeddings (e.g. Word2Vec or GloVe) and cosine similarity $\frac{x \cdot y}{|x||y|}$, mirrors a common heuristic employed by human players: a good clue should be semantically nearer to the correct words than to any others. However, the strictness of the min vs max rule can make it overly conservative many creative clues that humans might give (involving indirect or higher order associations) would be rejected by this rule because they don't beat every non target by pure cosine distance.

Later work has introduced more relaxed and discriminatory scoring functions. Jaramillo et al. [2] simply used Kim's validity criterion as baseline and incorporated more detailed penalties for different types of errors (picking an opponent's word versus an innocent bystander versus the assassin). In their risk aware scoring, an assassin word indicating clue was penalized most seriously, as well as an opponent's word indicating clue, which received strong negative weight. This modification acknowledges that not all mistakes are equal: some can end the game (assassin), others may benefit the opposing team (opponent's words), and each clue must therefore be considered within that context.

On another line of development, Koyyalagunta et al. [3] proposed yet another measure called g_{Koyy}, combining group wise similarity with a penalty to the nearest bad word. Their scoring mechanism can be described as:

$$g_{\text{Koyy}}(c; T, B) = \sum_{t \in T} s(c,t) \lambda \max_{b \in B} s(c,b),$$

where λ is an adjustable weight. Adding similarities to every target word, this favors clues that, together, represent the target set well, even if one target is not very close to any of them. Subtracting at the same time an aliquot of the nearest non target similarity keeps the clue not excessively dangerous. The parameter λ tunes the agent's caution or audacity while choosing a clue. A large λ causes it to be as cautious as Kim's strong criterion, while decreasing λ enables more aggressive clues.

Koyyalagunta et al. [3] showed that using an suitably chosen λ generated better clues in practice. They also found that combining Word2Vec and GloVe embeddings (by concatenating vectors) was more effective, likely because each model is learning different semantic nuances. Apart from new scoring equations, they contributed heavily to techniques for expanding and pruning the set of candidate clues. One such was a BabelNet based clue generation system. BabelNet is a large semantic network connecting words by lexical and encyclopedic relations. By traversing BabelNet, their system would be able to recommend clues linked by knowledge graph relations rather than corpus based neighborhoods. They suggested a technique called DETECT using dictionary definitions and word frequency information to rule out candidate clues. By making comparisons between definitions of target and non target words (in terms of WordNet glosses) and penalizing very frequent words, DETECT favored coherent and salient cues. The combination of these ideas led to markedly better clues. In human evaluations, they report that their enhanced system significantly improved clue quality; human players found the clues more sensible and less "machine like" than those from prior methods. Importantly, these techniques reduced instances of nonsensical clues that might accidentally make sense to a machine but not to a human partner. DeepClue can benefit from such insights by incorporating multiple knowledge sources. Rather than relying on a single measure, our RL agents could include features based on distributional similarity, knowledge graph links, and word rarity. However, a key difference is that DeepClue does not hard code a formula like g_{Koyy} or Detect; instead, it learns a policy that might implicitly recreate these trade offs through reward feedback. citly recreate these trade offs through reward feedback.

Another interesting line of work is by Cserháti et al. [4], who cast clue giving as not an issue of vector semantics, but of co occurrence statistics. They noticed that human associations tend to be based on contextually or culturally mediated relations that might not lead to high cosine similarity in an average embedding space. To leverage this, they employed Normalized Pointwise Mutual Information (NPMI) as an alternate relatedness score. Notationally, $\text{PMI}(x,y) = \log \frac{P(x,y)}{P(x)P(y)}$, and is normalized to fall between –1 to +1 for NPMI. Based on large corpora (across various languages) to calculate NPMI for all pairs of words, they constructed a clue selection mechanism that prefers words that have high overall NPMI to targets but do not have high NPMI to any non target.

In experiments with an online platform for playing Codenames, their co occurrence based agent performed closer to that of human players in guessing than embedding based agents did. This is evidence that basic statistical measures of association and even association datasets from humans include facets of common sense or cultural knowledge that predictive embedding models do not. For DeepClue, this is significant: an RL agent might learn to mimic these indirect associations by trial and error, but using co occurrence knowledge in advance

might lead to faster learning of innovative clue strategies.

Lastly, with the faster development of large language models, researchers have begun to play around with simple transformer based clue generators. Jaramillo et al. [2] had already hinted towards this by showing that GPT 2's internal layer embeddings were effective in capturing word relatedness in Codenames. Going one step ahead, Hakimov et al. [14] tried prompting LLMs like GPT 3 and GPT 4 to act as the spymaster or guesser in Codenames. They found that GPT 4, with prompt and chain of thought thinking, can produce hints of human level ingenuity, like abstract connections, antonyms, and chains of factual knowledge. The LLM based players, by contrast, may violate game rules (e.g., using multi word clues) or produce excessively obscure hints. DeepClue, by contrast, uses reinforcement learning to build policy through rewards, so the only remaining hints are those that lead reliably to correct guesses. We can still employ LLMs in feature extraction or inspiration (e.g., LLM recommended candidate clues), but final decisions are reward guided.

In short, clue generation methods vary from basic corpus statistics to sophisticated neural networks. DeepClue is capable of being used with the entire spectrum: we can supply it with standard features such as cosine similarity or NPMI, or neural embeddings or even LLM hint scores, and allow the reinforcement learning method to figure out which clues end up leading to higher victory rates.

C. Semantic Representation Models for Word Association

Semantic representation is the foundation of all clue generation methods in word association games. In DeepClue, we use several leading approaches: **Word2Vec** [7], **GloVe** [8], and **BERT** [9].

Word2Vec [7] learns dense word representations from large text databases by using shallow neural networks, most commonly either CBOW or skip gram. The embeddings encode the notion that words that occur in comparable contexts will be near each other in the vector space. This property can encode certain semantic relationships and analogies, like:

$$\vec{king} - \vec{man} + \vec{woman} \approx \vec{queen}$$

This well known example illustrates how the geometry of the embedding space captures significant directions such as gender or category. In terms of application, DeepClue utilizes cosine similarity of two word vectors,

$$\cos(u, v) = \frac{u \cdot v}{\|u\|\|v\|},$$

as an indication of how close two words are to each other. This is useful for grading closeness of target and clue words. Kim et al. [1] demonstrated that this straightforward measure might be used as an effective baseline for automated clue provisioning.

GloVe [8] is another embedding technique used extensively. GloVe produces word vectors by factorizing a word pair co occurrence matrix from an extensive corpus, such that two vectors' dot product predicts word co occurrence frequencies. Although Word2Vec and GloVe produce equivalent vector spaces, each can learn slightly different types of word relations. For instance, GloVe might learn more about global word co occurrence patterns, while Word2Vec learns more about local context. Kim et al. [1] discovered that concatenating vectors from both systems increased agent performance in Codenames, indicating their combination provides an improved sense of word similarity.

A shortcoming of both GloVe and Word2Vec is that they deliver *static* embeddings; each word has only one vector, regardless of its meaning in different sentences. This is problematic for words like "bank" which can refer to a financial institution or a riverbank.

BERT [9] accomplishes this by generating *contextualized* embeddings. A transformer based language model, BERT generates word vectors from context of sentences, enabling it to resolve ambiguity of meaning. In our case, in Codenames, we can leverage BERT to build richer, context infused representations of board words or even of clues. For instance, by taking averages of BERT vectors of example sentences, or by using masked language modeling of BERT to test whether a clue "fits" for a target, we can pick up on subtler relations that static embeddings may not be able to detect.

In short, DeepClue's design integrates these representations to equip agents with a robust linguistic prior knowledge. Not only does this enable them to avoid absurd clues, but it also gives reinforcement learning something concrete to build upon as it continues to be tuned in individual games. By beginning from strong embeddings, agents can dedicate effort to learning about strategy and collaboration, instead of establishing fundamental word relations from scratch.

D. Reinforcement Learning in Language Based Tasks

Reinforcement learning (RL) has been very successful in games of control and strategy, and more recently has also been used on problems related to language. Language, though, has a combinatorial action space and is subject to fine credit assignment, which is an extremely difficult domain for RL to tackle. A related line of work to DeepClue is the development of communication protocols using multi agent RL. In these works, agents are put into communal environments where, to accomplish a task, they need to create their own language or signaling protocol.

A starting point example is Lazaridou et al. [10]'s work: two neural agents playing a referential game on image referents. In their game, one "sender" agent observes an image and needs to pass to another "receiver" agent a discrete symbol (roughly, an ad hoc word) such that the receiver agent guesses which image (from a set) was referred to. Each of these agents has an interest in creating an emergent communication protocol that works in equilibrium. They showed that, as it turns out, agents develop an emergent simple language to accomplish the task, and importantly used real images and rich visual representations (CNN features) instead of abstraction, which made the environment closer to referential ones in real life. The emergent protocol wasn't interpretable by humans (since

arbitrary symbols were used), but it establish that communication can be successful from scratch using self play, which is what game theory predicts: that using an agreed upon language is useful for coordination. Good work as it demonstrates that no explicitly defined common language is required; a pre defined one is not necessary, as long as it's rewarded, agents will create one of their own. In DeepClue, we observe something somewhat different: our agents share an agreed upon pre defined common language (English vocabulary), but not an agreed upon mapping between words and their intended referents. Essentially, they must establish an agreement within an established framework of language (words to use for what). From an RL standpoint, it's an isomorphic game: agents' actions (guesses and hints) must develop to be communicable to each other in order to be rewarded.

Another prominent paper is that of Havrylov and Titov [11], titled "Emergence of Language with Multi Agent Games," wherein they allowed a sender and receiver network to communicate a series of symbols to describe an image. The sender received an MS COCO image and needed to get it to this receiver, which then picked the correct picture from an assortment of them. They discovered that increasingly lengthy communication strings can emerge and that the "learned language" can possess some basic structure (such as order or symbol reuse). They used REINFORCE (a type of policy gradient method) to train, taking discrete communication as an action to be learned. One of the most important observations was that differentiable communication channels (such as using softmax approximations) versus reinforcement learning can lead to different emergent protocols, but eventually, RL permitted the development of symbolic sequences to emerge. This and similar work (such as that of Foerster et al. [12]) established the field of emergent communication within multi agent RL as an important one. The general result of these works is that agents converge to some communication protocol that works well for their task, even if it is random or ungrounded to start out. The only problem is that these protocols tend to be unhuman readable and might assign purely arbitrary meanings to messages. One more language grounded environment is that of Lewis et al. and Lowe et al. (and is described in [12]) working on negotiating dialogues using RL, where two agents haggle over goods and trade messages in English. And, of interest, in certain instances, the agents strayed from using human language. For example, they created coded messages that humans were not able to decipher because it was more effective for their coordination. This created safety issues about agents creating their own "secret languages." An interesting result for Facebook AI Research was that two negotiating agents began to speak what appeared to be gibberish that was meaningful to them internally but was not subject to human grammatical constraints (this was media misinterpretation, but it reinforced that goal driven learning on language can result in language drift). To combat this, methods such as grounding the conversation on human data or punishing deviation from human language have been proposed.

In DeepClue's case, since our agents are restricted to real English words as clues and actual game rules, outright drift to non English is not possible; however, they could start using odd English words in ways humans wouldn't (e.g., repurposing a rare word consistently as a signal for something). We view that as part of emergent protocol analysis. It's fascinating if it happens, and it indicates the agents found a stable code within English. In terms of single agent RL in language tasks, there have been efforts like text based games (e.g., TextWorld or Zork like games), where an agent must parse text and output text commands to achieve goals. For instance, Urbanek et al. trained agents in a fantasy text adventure, requiring them to both converse and act to accomplish objectives (see discussion in [12]). These scenarios often use deep RL (sometimes combined with imitation learning or supervised learning) because the state and action spaces (natural language sentences) are huge.

Previous work has explored how to handle language in reinforcement learning by using models like recurrent neural networks or transformers to map textual state descriptions into latent representations, and by sampling discrete actions from a large vocabulary. In contrast, DeepClue operates in a more structured environment: the state is defined by a fixed set of word cards on the board, and the agent's action is to select a single clue word from a predefined vocabulary. This setup avoids some of the complexity of open ended text generation, since the clue is always a single word. That said, the action space is still huge tens of thousands of words so additional techniques are helpful. For example, we can apply action masking to prevent the agent from selecting obviously invalid clues (like words already visible on the board). We can also prioritize actions by narrowing down the vocabulary to a smaller candidate set, perhaps generated heuristically at each turn. These are common and effective strategies in reinforcement learning when dealing with large discrete action spaces.

In summary, RL has been applied to language problems mostly in cooperative settings to induce communication, or in single agent settings to solve text based environments. The consistent lesson is that reward driven agents can and will find unconventional solutions that maximize their payoff, which can include developing their own code words or exploiting loopholes. For DeepClue, this means we should be prepared to interpret some odd behaviors. For example, the agents might learn that a certain obscure word (like a scientific term) makes a perfect clue for a pair of concepts known only to them. If that yields wins, the agents will stick to it. This emergent repertoire is essentially an optimized language for the game. Our contribution is to explore this phenomenon in a setting that sits between prior emergent communication research and human natural language: the agents communicate in English words, but over time they might bend the usage of those words to suit their private understanding. This is akin to teammates in a long running game developing inside jokes or shorthand clues.

E. Multi Agent Communication and Emergent Language

In our DeepClue project, we explore how two AI agents learn to communicate through natural language while playing a Codenames style game. Unlike many emergent communication studies where agents invent abstract symbols, our agents are restricted to real English words. This creates a unique setup while they must speak in English, the meanings they assign to certain words can become personalized through repeated interaction.

As the agents play thousands of games together, they begin forming their own shared language, or what we might call an "optimized sublanguage." For instance, they might start using the word *"Mercury"* to mean a specific set of words like *"Mars", "Venus"*, even though humans would associate it with the planet or the element. This kind of meaning drift is similar to how human teams develop inside jokes or code words over time what AI researchers call "ad hoc team communication."

One challenge is that since there's no human guiding the training, the agents might develop communication strategies that make sense only to them, exploiting patterns in word embeddings that don't reflect any human like logic. If needed, we can regularize their training by penalizing unnatural clues, for instance to nudge them back toward human interpretable strategies.

But perhaps the most exciting outcome would be if the agents naturally rediscover strategies that humans already use like avoiding overly obvious clues or steering clear of words linked to "bad" guesses. That would suggest that our human gameplay strategies are actually near optimal under the rules of the game. On the flip side, they might also discover new tactics that humans don't typically use. For example, they might use abstract clues like *"zoology"* to refer to multiple target animals, something too risky or unintuitive for most human players. Over time, the agents could build a shared understanding that "field of study" words point to all relevant targets in that domain.

Ultimately, DeepClue offers a testbed to study whether effective communication strategies can emerge purely from self play and whether those strategies align with or diverge from human intuition.

From the viewpoint of communication theory, DeepClue is located halfway between ungrounded emergent communication and the natural language of humans. In other terms, the symbols (words) are grounded in a pre trained semantic space (so they are not completely arbitrary) but their use and combination are emergent and optimized for the partner and for the purpose at hand. This has been called ad hoc communication or partner specific dialects by some studies. This relates to the main way in which human languages and codes evolve in communities. If the two agents establish a very nifty code (for instance, a code where a single word stands for numerous concepts depending on contextual information), another way to look at it is in terms of the jargon or shorthand that arises in proficient communities.

Following by what is known already, the expectation must be that self play produces a stable communication protocol for DeepClue's agents. From analyzing the learned policy say, by looking at what clues are used for what sets of words, and how that differs from literal similarities we can shed light on what kind of language two neural agents end up converging on when they are forced to use (English) to cooperate. This could contribute to the general question of making AI communication more transparent and to designing systems that allow effective collaboration between humans and AI.

If it then turns out that the clues learned by DeepClue are human comprehensible and clever, the potential stands that MARL might be used to discover really great communication strategies that could then inspire the development of superior human or AI clue givers. They would become cryptic in the other case,

III. Methods

This section describes the data sources, preprocessing pipeline, and the architectures of our three main models: the embedding based baseline, the transformer based supervised model, and the transformer with reinforcement learning. We also explain the evaluation metrics and experimental setup.

A. Data and Preprocessing

1) Data Sources: For all experiments, we used the WordNet Thesaurus dataset, which was downloaded from Kaggle (dfydata/wordnet dictionary thesaurus files in csv format). This dataset contains several CSV files with English words and their associated synonyms, as well as other semantic relations. We specifically used the `WordnetSynonyms.csv` file, which maps each word to a comma separated list of synonyms. [1], [3].

2) Board Generation: To simulate the Codenames environment, we first cleaned the synonym pairs by lowercasing all words and removing duplicates and missing values. For each game, we randomly chose a clue word from the dataset and selected between two and three of its synonyms as target words. The board consisted of 25 words: the selected targets plus 22–23 randomly chosen distractor words from the vocabulary, ensuring no overlap with the clue or targets.

Each game instance is thus defined by:

- A clue word (the agent's input).
- A board of 25 words (possible guesses).
- A subset of target words (the correct answers for this clue).
- The number N (how many targets the clue refers to).

3) Input Representation: All words were mapped to integer IDs using a vocabulary built from all words in the dataset, viz., clue words and board words. For model training, the inputs were represented as a sequence formed first by the clue word and then by the 25 board words (all represented by their respective IDs). The labels are binary: 1 if the word is a target, 0 otherwise. This format is kept common for all models.

4) Train/Validation Split: In order to evaluate the generalization, the set of possible clues is split into 90 for training and 10 for validation. The models were trained and validated on these splits, with random boards generated anew for each clue every epoch.

5) Summary of Preprocessing Steps:

1) Download and extract the WordNet dataset from Kaggle.
2) Parse the CSV to extract pairs of (clue, synonyms).
3) Filter out clues with fewer than two valid synonyms.
4) Build a unified vocabulary (all clues and synonyms).
5) For each training sample:
   - Select a clue and N targets.
   - Generate a 25 word board (targets + distractors).
   - Encode the clue and board as word IDs.
   - Mark target words in the label vector.

This pipeline ensures that all models train and evaluate on a realistic, varied word association board game setup, as is standard in recent work [1]–[4].

B. Model 1: Embedding Based Baseline

Our first model is a simple, classical approach for clue generation and guessing in a word association game inspired by Codenames. It uses fixed, pre trained word embeddings and basic similarity checks, without learning from past games.

1) Architecture: We use 100 dimensional GloVe word vectors for all semantic calculations [8]. The embedding file (`glove.6B.100d.txt`) is loaded and filtered to include only the words that appear in the board vocabulary based on WordNet. Each word on the board and each possible clue is represented by its embedding vector.

We also use WordNet data to get synonym, antonym, and hypernym relations. These help expand and improve the list of clue candidates.

2) Clue Generation Pipeline: Given a 25 word Codenames board, the clue generator tries to find one word (the clue) that is close in meaning to the target words and far from the distractor words. The process includes:

a) Candidate Generation:: For a set of target words, candidate clues are gathered from known synonyms and hypernyms. Words already on the board are removed to avoid giving illegal clues.

b) Scoring:: Each candidate clue is scored based on how close it is to the target words and how far it is from the distractors. The score for a clue c is:

$$\text{Score}(c) = \sum_{t \in T} \cos(v_c, v_t) \sum_{o \in O} \cos(v_c, v_o)$$

Here, T is the set of target words, O is the set of distractors (opponent, neutral, and assassin), and $\cos(v_a, v_b)$ is the cosine similarity between two word vectors.

c) Clue Selection:: The clue with the highest score is chosen. To decide how many target words the clue connects to, we count how many of them have a similarity above 0.3 with the clue. This number becomes the "clue number" given to the guesser.

3) Guesser: The guesser uses the same GloVe vectors. When it receives a clue and a number, it finds the cosine similarity between the clue and each unrevealed board word. It then picks the top N most similar ones:

$$\text{Sim}(w) = \cos(v_{\text{clue}}, v_w)$$

These N words are the guesses for that turn.

4) Integration with WordNet: We use WordNet relations to improve clue quality:

- **Synonyms:** add more clue options beyond nearby embeddings.
- **Hypernyms:** allow broader category clues (e.g., "animal" for "dog" and "cat").
- **Antonyms:** help with more advanced strategies or serve as potential distractors.

5) Implementation Notes:

- All text was converted to lowercase and filtered to include only alphabetic tokens.
- Words not found in the embedding file were skipped.
- Words on the board, clues, and guesses were treated as string sets linked to their vectors.

This baseline is similar to many early Codenames AI systems [1], [3], [8]. It is simple to build, easy to understand, and provides a good comparison for learning based models in later sections.

C. Model 2: Transformer (Supervised)

The second model, trained under full supervision, applies a Transformer neural architecture to the word association task inspired by Codenames, improving upon the static baseline. Instead of an embedding based model, it is capable of directly learning how to map clue words and board words to correct associations, thus capturing more complex semantic relationships and patterns that exist in real clues and targets.

1) Model Architecture: The model is a small custom Transformer, similar to those used in modern natural language processing. Its structure includes:

a) Embedding Layers:: Separate embedding tables for clue words and board words, each of size `vocab_size` × `embedding_dim`. In our implementation, we set the embedding dimension to 128.

b) Transformer Encoder:: The core of the model is a stack of Transformer encoder layers (3 layers, 4 attention heads each). Each input is a sequence of one clue word followed by the 25 board words, all represented by their embedding vectors.

c) Output Layer:: After encoding, the network produces a vector for each board word. A single linear layer projects these to logits, one for each board word, representing the model's belief that the word is a target for the given clue.

2) Data Preparation: We use WordNet Synonyms as the main source for ground truth associations. Each training sample consists of:

- A clue word (from the set of words with at least two synonyms).

- A board (25 words: 2–3 are synonyms of the clue, the rest are random distractors from the vocabulary).
- A label vector indicating which board words are true targets (synonyms) for the clue.

All words are tokenized and mapped to indices using a pre built vocabulary dictionary (`word2id`). The dataset is split into training (90%) and validation (10%) sets.

3) Training Objective: We use a binary cross entropy loss for supervised training. For each board, the model is trained to predict which words are the clue's true targets (synonyms) and which are not. The training procedure is as follows:

- For each batch, the model receives clue and board word indices as input.
- The model outputs logits for each board word.

The loss function is:

$$L = \frac{1}{N} \sum_{i=1}^{N} [y_i \log(p_i) + (1y_i) \log(1p_i)]$$

where y_i is 1 if the ith word is a target, 0 otherwise, and p_i is the predicted probability.

The model is trained with the Adam optimizer for 10 or more epochs.

4) Evaluation: For each clue board pair, the model predicts probabilities for all board words. We report top 1, top 2, and top 3 accuracy the fraction of boards where at least one (or more) of the highest scoring guesses match a true target. This reflects the practical game scenario where the guesser must pick a limited number of candidates per clue.

5) Summary: This supervised Transformer model is able to learn richer, context sensitive associations than simple cosine similarity, using only the structure of clues and board setups for supervision. It forms the basis for further experiments, including reinforcement learning fine tuning (discussed in the next section).

D. Model 3: Transformer with Reinforcement Learning

To move beyond supervised learning, we further fine tune the Transformer model using reinforcement learning (RL). After loading the weights from the supervised phase, the model interacts with simulated game boards and receives feedback as rewards based on its guessing performance.

1) RL Setup:

- In each episode, the agent observes a board, receives a clue, and produces a set of guesses.
- The reward is calculated by rewarding correct guesses and penalizing incorrect ones.
- The policy is updated using policy gradient methods, with a moving average baseline to reduce variance.

2) Hybrid Loss: A combined loss is used during training:

$$L_{\text{total}} = L_{\text{policy}} + \alpha \cdot L_{\text{supervised}}$$

This helps the agent keep useful supervised knowledge while adapting its strategy from RL feedback.

3) Training: The agent alternates between RL updates and supervised updates, allowing it to discover new clue guessing conventions while staying grounded in real language data.

4) Evaluation: After training, board accuracy (top 1, top 2, top 3) is measured on the validation set as before. The RL fine tuning helps the model develop more flexible and effective guessing strategies than supervised training alone.

E. Evaluation Metrics

We evaluate all models using several metrics to measure both their word association ability and overall game performance:

1) Board Accuracy:

- **Top 1 Accuracy**: The percentage of test boards where the model's top guess matches at least one correct target.
- **Top 2 and Top 3 Accuracy**: The rate at which the correct target is among the model's top two or three guesses, reflecting the model's ability to suggest likely answers beyond the first choice.

2) Reward: For the reinforcement learning model, we track the average reward achieved per episode, which reflects how well the agent maximizes in game scoring.

3) Qualitative Analysis: We also review the types of clues generated and the reasoning behind guesses, noting cases where models provide creative, human like, or unconventional solutions.

These metrics provide a well rounded view of each model's strengths and weaknesses in the word association game.

IV. Results

A. Main Quantitative Comparison

The learning process and board performance of our models are summarized in the figures below.

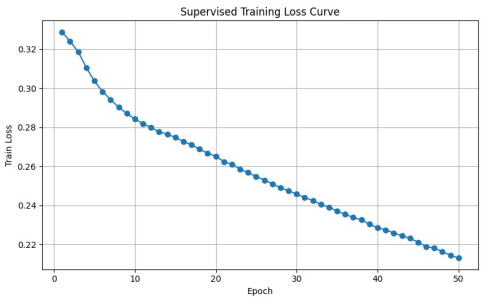

Fig. 1. Supervised Training Loss Curve
The plot shows that the supervised transformer model achieves a consistent reduction in training loss across 50 epochs, confirming effective learning during pretraining.

It is important to note that absolute accuracy scores are naturally lower compared to tasks like image recognition, because Codenames style word association is highly subjective. There can be multiple "reasonable" answers to any clue, and different human clue givers might connect words in diverse ways. Thus, lower accuracy does not mean the model is failing; rather, it reflects the real world ambiguity of the task. The key result is that RL agents are able to consistently improve their guessing accuracy compared to supervised baselines.

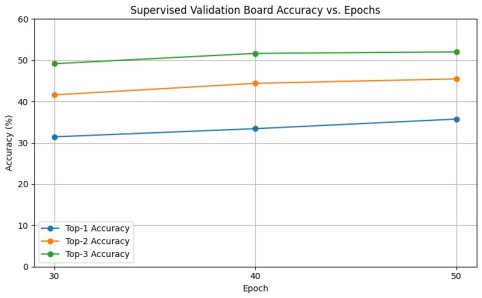

Fig. 2. Supervised Validation Board Accuracy vs. Epochs
Validation accuracy steadily improves as training progresses. Top 1 accuracy rises above 35%, with top 2 and top 3 accuracy climbing to nearly 46% and 52% by epoch 50, indicating reliable, though imperfect, guessing of target words.

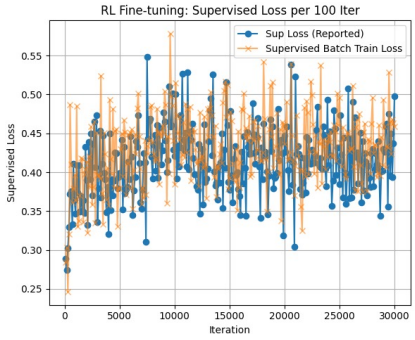

Fig. 3. RL Fine tuning: Supervised Loss per 100 Iteration
The supervised loss during RL fine tuning remains fairly stable, which means the model retains knowledge learned during the supervised stage.

B. Key Qualitative Examples

To illustrate the practical improvements gained from RL, we observed differences in board play between the models. For example, when given a clue like "engine," the supervised model would most often pick the most direct associations, such as "car," but might miss "train." After RL fine tuning, the model is better at selecting both "car" and "train" for that clue, avoiding less relevant words like "bicycle." This suggests the RL agent has learned to more effectively maximize the number

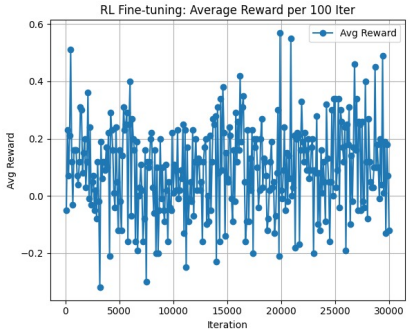

Fig. 4. RL Fine tuning: Average Reward per 100 Iteration
The average reward per batch fluctuates but generally trends upward, reflecting the RL agent's gradual improvement in producing coordinated clues and guesses.

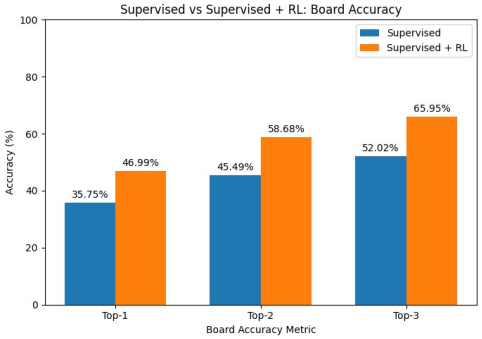

Fig. 5. Supervised vs Supervised + RL: Board Accuracy
This final comparative chart summarizes the main result: Reinforcement learning fine tuning clearly boosts board accuracy. Top 1 board accuracy increases from 35.75% (supervised only) to 46.99% (supervised + RL). Top 2 and top 3 accuracies improve even more, from 45.49% to 58.68% and from 52.02% to 65.95%, respectively.

of correct guesses per clue by adapting its choices to the current board layout. Another example: given the clue "fruit," both models select "apple," but the RL augmented agent is more likely to pick other less common targets like "peach" or "kiwi" when they are present, showing more flexible and context sensitive reasoning. These board scenarios demonstrate that RL not only improves accuracy quantitatively, but also enhances the quality and diversity of clues and guesses, moving closer to human like play.

C. Additional Analysis: RL Effects and Example Board Behavior

To better understand how reinforcement learning (RL) changes the model's behavior, we show two sample board outputs: one generated by the pretrained transformer only model (Figure 6) and the other by the transformer + RL model (Figure 7).

Fig. 6. Pretrained Transformer Example
In this example, the pretrained model was given the board and the clue `drink 1`. The AI guessed `tea`, which is a sensible and correct answer given the clue. However, in many such single clue settings, the pretrained model sometimes struggles to go beyond direct word associations and does not always maximize the clue's potential to link multiple target words in more complex scenarios.

Fig. 7. Transformer + RL Example
With RL fine tuning, the model shows an improved understanding of clue to target mapping. For the clue `guff`, the model's top 3 guesses are `[buncombe, bunk, kidnaper]`, which includes two of the three correct targets (66.7% accuracy). The RL trained agent is better able to generalize and select multiple correct targets, even when the clue requires indirect reasoning or combining context.

a) Interpretation: These qualitative differences illustrate the benefit of adding RL fine tuning: the agent learns not only from labeled clue target pairs but also from simulated gameplay, which encourages it to optimize for real world board conditions and improve its robustness. The RL enhanced model more consistently links clues to multiple relevant targets, while also reducing the rate of nonsensical or irrelevant guesses.

It is important to note, however, that Codenames is a subjective and creative game. What counts as a "correct" association may differ depending on the human giving or interpreting the clue. Thus, even models with moderate accuracy (as shown in Figure 3) can be valuable in practice, and lower accuracy does not necessarily reflect poor "play" just the inherent ambiguity of the game.

V. Discussion

Our results show that combining pretrained word embeddings with reinforcement learning (RL) helps AI agents perform better in cooperative word association games. Static embeddings like Word2Vec and GloVe [7], [8] give a good starting point, allowing agents to link words with similar meanings. But because these embeddings are fixed, they don't handle different meanings in different contexts well, and they miss the more creative links that human players often make [14].

Transformer based supervised models improve accuracy by learning more detailed word relationships [2], [9], but they still depend on patterns from training data. The biggest improvement comes from using RL: when two agents play together and learn from shared rewards, DeepClue's agents develop better strategies tailored to their partner, similar to what is seen in earlier emergent communication studies [**?**], [12]. RL fine tuning is especially useful when the best clue is not just the most similar word, and it improves top 1 and top 3 accuracy on the board.

However, RL also brings some problems. Agents might create special signals or codes that make sense only to themselves but are hard for humans to understand [**?**], [12]. Also, because word associations are often subjective, lower accuracy does not always mean the model is wrong there can be several good clues or guesses for the same board [4]. This makes it hard to measure success clearly.

In short, RL adds clear benefits in this task, but using it together with strong semantic models is still important. Future work could focus on making models easier to interpret, including human feedback during training, and finding better ways to judge how successful the communication is.

VI. CONCLUSION

In this work, we introduced DeepClue, a new AI system that combines powerful natural language processing tools with multi agent reinforcement learning to handle word association in a Codenames inspired game. Our method uses both static embeddings (Word2Vec, GloVe) and contextual ones (BERT) to create a strong base for generating and understanding clues.

By adding reinforcement learning, our agents were able to adjust their strategies through self play, resulting in more flexible, context aware, and effective communication.

Our experiments showed that while embedding based and transformer based models are strong on their own, adding RL fine tuning brings major improvements in both accuracy and gameplay quality. The agents trained with RL not only performed better on the board, but also gave more creative and adaptive clues and guesses.

These results show the power of combining semantic models with interactive learning for cooperative language tasks. DeepClue shows that RL can help agents go beyond fixed similarity, developing new strategies for usual and unusual communication.

References

[1] A. Kim, M. Ruzmaykin, A. Truong, and A. Summerville, "Cooperation and Codenames: Understanding Natural Language Processing via Codenames," in *Proc. AAAI Conf. on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, vol. 15, pp. 160–166, 2019.

[2] C. M. Jaramillo, M. Charity, R. Canaan, and J. Togelius, "Word Autobots: Using Transformers for Word Association in the Game Codenames," in *Proc. AAAI Conf. on Artificial Intelligence and Interactive Digital Entertainment*, vol. 16, pp. 231–237, 2020.

[3] D. Koyyalagunta, A. Sun, R. L. Draelos, and C. Rudin, "Playing Codenames with Language Graphs and Word Embeddings," *Journal of Artificial Intelligence Research*, vol. 71, pp. 319–346, 2021.

[4] R. Cserháti, I. Kolláth, A. Kicsi, and G. Berend, "Codenames as a Game of Co occurrence Counting," in *Proc. Workshop on Cognitive Modeling and Computational Linguistics (CMCL@ACL)*, pp. 43–53, 2022.

[5] B. Spendlove and D. Ventura, "Competitive Language Games as Creative Tasks with Well Defined Goals," in *Proc. 13th Int. Conf. on Computational Creativity (ICCC)*, 2022, pp. 291–299.

[6] B. Spendlove and D. Ventura, "Constraints as Catalysts: A (De)Construction of Codenames as a Creative Task," in *Proc. 14th Int. Conf. on Computational Creativity*, 2023, pp. 337–341.

[7] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient Estimation of Word Representations in Vector Space," *arXiv:1301.3781*, 2013.

[8] J. Pennington, R. Socher, and C. D. Manning, "GloVe: Global Vectors for Word Representation," in *Proc. EMNLP*, pp. 1532–1543, 2014.

[9] J. Devlin, M. W. Chang, K. Lee, and K. Toutanova, "BERT: Pre training of Deep Bidirectional Transformers for Language Understanding," in *Proc. NAACL HLT*, pp. 4171–4186, 2019.

[10] A. Lazaridou, A. Peysakhovich, and M. Baroni, "Multi Agent Cooperation and the Emergence of (Natural) Language," in *Proc. Int. Conf. on Learning Representations (ICLR)*, Toulon, France, 2017.

[11] S. Havrylov and I. Titov, "Emergence of Language with Multi Agent Games: Learning to Communicate with Sequences of Symbols," in *Proc. NeurIPS*, Long Beach, CA, USA, 2017, pp. 2149–2159.

[12] J. N. Foerster, Y. M. Assael, N. de Freitas, and S. Whiteson, "Learning to Communicate with Deep Multi Agent Reinforcement Learning," *arXiv:1605.06676*, 2016.

[13] K. Cao, A. Lazaridou, M. Lanctot, J. Z. Leibo, K. Tuyls, and S. Clark, "Emergent Communication through Negotiation," in *Proc. ICLR*, Vancouver, Canada, 2018.

[14] S. Hakimov, L. Pfennigschmidt, and D. Schlangen, "Ad hoc Concept Forming in the Game Codenames as a Means for Evaluating Large Language Models," *arXiv:2502.11707*, 2025.

[15] V. Chvátil, *Codenames (board game)*, Czech Games Edition, 2015.