Journal of Artificial Intelligence Research 71 (2021) 319-346 Submitted 01/2021; published 06/2021

Playing Codenames with Language Graphs and Word Embeddings

Divya Koyyalagunta DIVYAKOYY@GMAIL.COM
Anna Sun ANNA.SUN@ALUMNI.DUKE.EDU
Rachel Lea Draelos RLB61@DUKE.EDU
Cynthia Rudin CYNTHIA@CS.DUKE.EDU
Department of Computer Science
Duke University
Durham, NC 27708, USA

Abstract

Although board games and video games have been studied for decades in artificial intelligence research, challenging word games remain relatively unexplored. Word games are not as constrained as games like chess or poker. Instead, word game strategy is defined by the players' understanding of the way words relate to each other. The word game Codenames provides a unique opportunity to investigate common sense understanding of relationships between words, an important open challenge. We propose an algorithm that can generate Codenames clues from the language graph BabelNet or from any of several embedding methods – word2vec, GloVe, fastText or BERT. We introduce a new scoring function that measures the quality of clues, and we propose a weighting term called DETECT that incorporates dictionary-based word representations and document frequency to improve clue selection. We develop BabelNet-Word Selection Framework (BabelNet-WSF) to improve BabelNet clue quality and overcome the computational barriers that previously prevented leveraging language graphs for Codenames. Extensive experiments with human evaluators demonstrate that our proposed innovations yield state-of-the-art performance, with up to 102.8% improvement in precision@2 in some cases. Overall, this work advances the formal study of word games and approaches for common sense language understanding.

1. Introduction

If you wanted to cue the words "piano" and "mouse," but not "bison" or "tree" would you have immediately thought of the clue "keyboard"? If so, perhaps you are an expert at the game Codenames. This clue, however, was not provided by a human expert – rather, it was provided by an automated Codenames player that we introduce in this work.

For decades, games have served as a valuable testbed for research in artificial intelligence (Yannakakis & Togelius, 2018). Deep neural networks have outperformed human experts in chess (Campbell, Hoane Jr, & Hsu, 2002), Go (Silver, Huang, Maddison, Guez, Sifre, et al., 2016), and StarCraft (Vinyals, Babuschkin, Czarnecki, Mathieu, Dudzik, et al., 2019). In comparison, progress in AI for word games is more limited. The study of word games has immense potential to facilitate deeper insight into how well models represent language, particularly common sense relationships between words.

word representation	word2vec	GloVe	fastText	BERT	BabelNet-WSF
clue	**animals**	**salmon**	**salmon**	**harbour**	**rock**
intended word 1	bison	fish	fish	pirate	diamond
intended word 2	cat	bison	bison	port	cliff

Figure 1: Example of a simplified version of a Codenames board. Blue words belong to the blue team and red words to the red team. Only the clue-givers can see which words belong to which teams. A clue-giver on the blue team generates clues to induce the blue team guessers to pick blue words. The team that identifies all of their own words first wins. In this figure, the table shows clues chosen for the blue team by applying our DETECT algorithm, using various word representations. "Animals" is an example of a clue that is too generic, as it also applies to the red words "calf" and "mammoth." "Salmon" is a good clue for the intended word "fish" but less likely to induce a guesser to choose the intended word "bison" – however, "pirate" and "port" are the only other related words, which are both blue and therefore will not result in a penalty. "Harbour" successfully indicates "pirate" and "port" without applying to any red words, meaning it is a high-quality clue. Similarly, "rock" connects to "diamond" and "cliff" without any close relationships to red words.

Figure 2: Clues chosen for board words by one of our algorithms using various word representations.

Codenames is a word game in which a clue-giver must analyze 25 board words and choose a clue word that connects to as many of their own team's words as possible, while avoiding the opposing team's words (**Figure 1**). Only the clue-giver knows which words belong to their team, so it is critical that the clue-giver avoid selecting a clue that will cause their team members to guess the opposing team's words. Choosing quality clues requires understanding complex semantic relationships between words, including linguistic relationships such as syno-, anto-, hyper-, hypo- and meronymy, references to history and

popular culture, and polysemy (the multiple meanings associated with one word). For example in **Figure 2**, the clue "keyboard" connecting the words "piano" and "mouse" uses polysemy (a keyboard can refer to a musical keyboard or a computer keyboard), hyponymy (a keyboard IS-A piano), and context (a computer keyboard is generally accompanied by a mouse). Thus, Codenames is distinct from other natural language processing tasks such as word sense disambiguation (e.g., Iacobacci et al., 2016) which focuses solely on polysemy, machine translation (e.g., Devlin et al., 2014) which focuses on cross-lingual understanding, or part-of-speech tagging (e.g., Collobert et al., 2011) which focuses on the grammatical role of words in a sentence.

Previous work on Codenames has leveraged word embedding models as both clue-givers and guessers, in order to evaluate performance based on how many times a model wins when paired with another model (Kim, Ruzmaykin, Truong, & Summerville, 2019; Jaramillo, Charity, Canaan, & Togelius, 2020). This performance evaluation measures how closely the embedding spaces of two methods align, and does not indicate the actual clue quality as judged by a human player. For example, the clue "duke" chosen for the board word "slug" from word2vec results in a correct guess by a word2vec guesser, but would likely result in a miss by a human guesser. Thus, when one word embedding method judges another, it is possible for the clue-giver embedding method to obtain "high performance" in spite of producing nonsensical clues. Instead of using computer simulations for our evaluations, we conduct extensive experiments with Amazon Mechanical Turk to validate the performance of our algorithms through human evaluation.

Previous work has been unable to leverage language graphs for Codenames due to computational barriers. We introduce a new framework, BabelNet-WSF, that improves computational performance by *caching subgraphs* and *introducing constraints on the types of graph traversals that are allowed*. The constraints also improve clue quality by permitting traversals only along paths for which the starting and ending node remain conceptually connected to each other. Additionally, we introduce a method that *extracts semantically relevant single-word clues from BabelNet synsets*, which are groupings of synonymous words associated with a node in the graph. On the whole, BabelNet-WSF enables competitive Codenames performance while remaining broadly applicable to other downstream tasks, including word sense disambiguation (e.g., Navigli, Jurgens, & Vannella, 2013) and semantic relatedness tasks (e.g., Navigli & Ponzetto, 2012).

In addition to our proposed methods for knowledge graphs, we introduce techniques that improve Codenames performance for both word embedding-based and knowledge-based methods. Most embedding methods rely on a word's context to generate its vector representation. There are two main limitations to this context-based approach for the Codenames task: (1) the resulting embeddings are capable of placing rare words (typically bad clues) close to common words, and (2) important relationships such as meronymy and hypernymy are difficult to capture in embedding methods. In the case of meronymy for example, parts of things (e.g. "finger") do not necessarily have the same context as the whole thing (e.g. "hand"). To address these limitations, *we propose* DETECT, (DocumEnT frEquency + diCT2vec), *a scoring approach that combines document frequency, a weighting term that favors common words over rare words, with an embedding of dictionary definitions.* Dictionary definitions more effectively capture meronymy, synonymy, and fundamental semantic relationships. For the embedding of dictionary definitions, we use Dict2Vec (Tissier,

Gravier, & Habrard, 2017), though our method can be used with other dictionary-based embeddings. DETECT significantly improves clue quality, with an increase of up to 102.8% precision@2 above baseline algorithms when evaluated by human players. Furthermore, DETECT leads to universal improvement on Codenames across all four word embedding methods (word2vec, GloVe, fastText, BERT) as well as an improvement on BabelNet-WSF.

The fact that our methods lead to improvement across all word representation methods is significant, especially in the context of recent work that suggests different word representation methods may be best suited to different sub-tasks. For example, CBOW outperforms GloVe on a Categorization task (clustering words into categories), but GloVe outperforms CBOW on Selectional Preference (determining how typical it is for a noun to be the subject or object of a verb) (Schnabel et al., 2015). DETECT is a promising metric to re-weight word similarities in embedding space and in knowledge graphs anywhere that word representations are used, such as comment analysis or recommendation engines.

This work shows promising results on a difficult word game task that involves many layers of language understanding and human creativity. Because we have focused on human evaluation, we identified problem areas in which word representations fail to perform well, and propose solutions that improve performance dramatically. Overall, our proposed methods advance the formal study of word games and the evaluation of word embeddings and language graphs in their ability to represent common sense language understanding.

2. Related Work

The closest related work to ours is that of Kim et al. (2019), who proposed an approach for Codenames clue-giving that relies on word embeddings to select clues that are related to board words. They evaluated the performance of word2vec and GloVe Codenames clue-giver bots by pairing them with word2vec and GloVe guesser bots. Although this evaluation approach is easy to run repeatedly over many trials, as it is purely simulation-based, the evaluation is limited to how well the clue-givers and guessers "cooperate" with one another. "Cooperation" measures how well GloVe and word2vec embedding representations of words are aligned on similarity or dissimilarity of given words. To be more explicit, two methods with different embeddings "cooperate well" if word embeddings that are relatively close based on the clue-giver's embeddings are also close based on the guesser's embeddings, and vice versa. As a result, it is clear that perfect performance (100% win percentage) comes from pairing a clue-giver and a guesser who share the same embedding method. However, this "cooperation" metric does not evaluate whether a clue given by a clue-giver is actually a good clue – that is, a clue that would make sense to a human.

To address this limitation, in this paper, we evaluate Codenames clue-givers based on human performance on the task of guessing correct words given a clue generated by an algorithm.

Kim et al. (2019) also explored the use of knowledge graphs for Codenames clue-giving, but ultimately did not consider knowledge graph-based clue-givers in their final evaluation due to poor qualitative performance and computational expense. In contrast, we propose a method for an interpretable knowledge graph-based clue-giver that performs competitively with embedding-based approaches. The knowledge graph-based method has a clear advantage in interpretability over the embedding-based approaches.

In an extension of Kim et al. (2019), Jaramillo et al. (2020) compared the baseline word2vec + GloVe word representations with versions using TF-IDF values, classes from a naive Bayes classifier, or nearest neighbors of a GPT-2 Transformer representation of the concatenated board words. Similar to Kim et al. (2019), they evaluated their methods primarily by pairing clue-giver bots with guesser bots. They included an initial human evaluation, where 10 games were played for both the baseline (word2vec + GloVe) and the Transformer representations as clue-giver and guesser, but the human evaluation is limited to only 40 games. Again, since evaluations from bots may not represent human judgments, our human evaluation is more realistic and extensive, conducted through Amazon Mechanical Turk with 1,440 total samples.

Another method (Zunjani & Olteteanu, 2019) proposes a formalization of the Codenames task using a knowledge graph, but does not provide an implementation of their proposed recursive traversal algorithm. We found that recursive traversal does not scale to the computation required to run repeated evaluations of Codenames - for each blue word b, we must find each associated word w in the knowledge base that has similarity $s(w, b)$ greater than some threshold t, and repeat this process every trial. In BabelNet, because each word may be connected to tens or hundreds of other words, this becomes unscalable when traversing more than one or two levels of edges. We propose an approach that scales significantly better than naive recursive traversal by limiting paths through the graph to those that yield high-quality clues.

Shen et al. (2018) also propose a simpler version of the Codenames task with human evaluation. The experimental setup focuses on comparing different semantic association metrics, including a knowledge graph-based metric. Their task differs from ours in two key ways. First, each of their trials considers three candidate clues drawn from a vocabulary of 100 words, whereas we consider candidate clues drawn from the larger vocabulary of all English words. Second, their usage of ConceptNet is different from our usage of BabelNet because they use vector representations derived from an ensemble of word2vec, GloVe, and ConceptNet using retrofitting (Speer, Chin, & Havasi, 2017) whereas we leverage the graph structure of BabelNet.

2.1 Other Language Games

Ashktorab et al. (2021) propose three different AI approaches for a word game similar to Taboo, including a supervised model trained on Taboo card words, a reinforcement learning model, and count-based model using the Small World of Words (De Deyne et al., 2019), a word evocation dataset. Their task, in which an agent gives clues until a user guesses the secret word, is different from Codenames. However, the word evocation dataset used in their count-based model could be leveraged for the Codenames task since it represents word relatedness, and presents interesting directions for future work. Another related task is the Taboo Challenge competition (Rovatsos, Gromann, & Bella, 2018), where AI systems must guess a city based on clues crowdsourced from humans. In this task, the AI system acts as the guesser, rather than the clue giver.

Other work in language games that are similar to Codenames include the Text-Based Adventure competition (Atkinson et al., 2019), which evaluates agents in text-based adventure games, Xu and Kemp (2010), which models a task in which a speaker gives a one-word

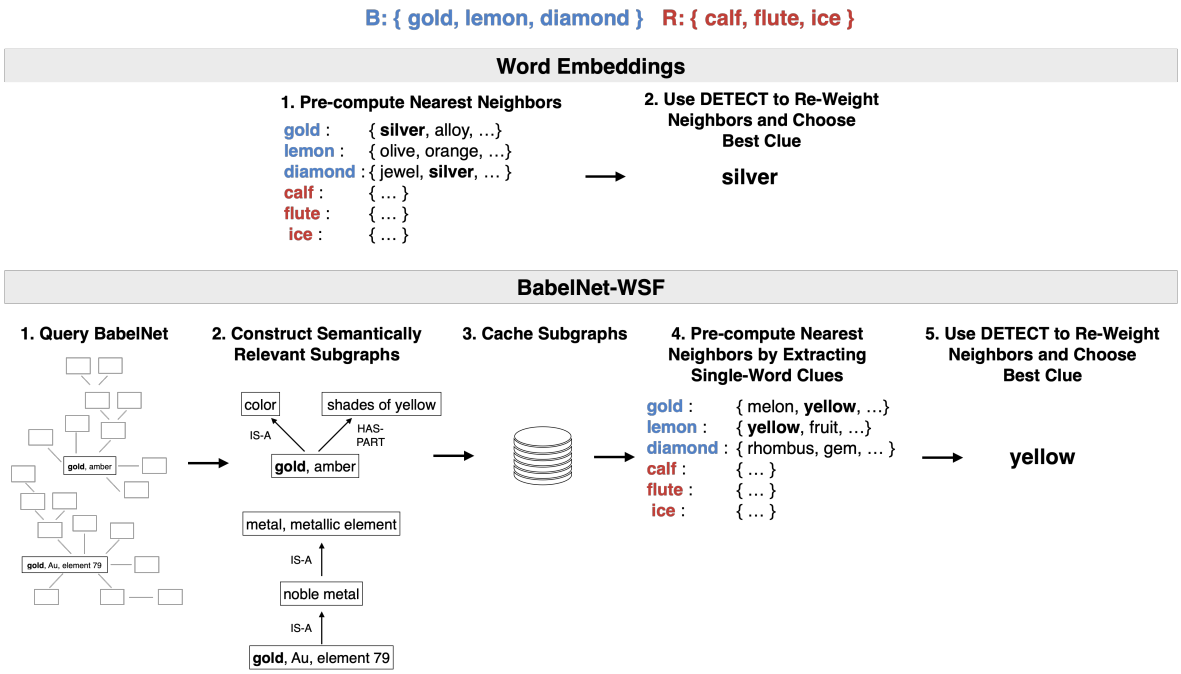

Figure 3: An overview of our proposed methods for improving Codenames performance. For word embedding approaches, this involves pre-computing nearest neighbors for all board words (e.g., gold, lemon, diamond, calf, flute and ice as shown above). To pre-compute nearest neighbors using BabelNet-WSF, we query BabelNet for every board word, and then construct the semantically relevant subgraph connected to that board word (described in **Section 3.2.2**). We then cache the subgraphs for later use (see **Section 3.2.1**), and extract single-word clues from BabelNet synsets (see **Section 3.2.3**). In **Section 3.4**, we describe how we get single word embeddings from a contextual embedding-based method such as BERT. Once candidate clues are produced by either a word embedding or BabelNet-WSF, we use the ClueGiver algorithm described in **Section 3.1** and apply our DETECT weighting term to choose the best clue for that board (see **Section 3.3**).

clue to a listener with the goal of guessing a target word, and Thawani, Srivastava, and Singh (2019), which proposes a task to evaluate embeddings based on human associations.

2.2 Quantifying Semantic Relatedness from WordNet

There is a body of previous work that proposes methods to quantify semantic relatedness in a knowledge graph beyond simply counting the number of edges between two nodes. Hirst, St-Onge, et al. (1998) developed a ranking of relationships into three groups, extra-strong, strong, and medium-strong, where only medium-strong includes a numeric score based on path length and path type; in our work we use numeric scores for all word comparisons and do not separate into three categories. Budanitsky and Hirst (2006) evaluated different techniques for the quantification of semantic relatedness for WordNet, but most techniques used noun-only versions of WordNet, whereas we consider all parts of speech including nouns, verbs, and adjectives. Furthermore, four of their five techniques restrict to hyponym relationships whereas we consider multiple relationship types. The proposed PageRank technique of Agirre et al. (2009) could be applied to BabelNet and is an interesting direction for future work.

3. Methods

Codenames is a word-based, multiplayer board game illustrated in **Figure 1**. The board consists of a total of $2N$ words, divided equally into the blue team's words $B = \{b_n\}_{n=1}^{N}$ and the red team's words $R = \{r_n\}_{n=1}^{N}$. Only the clue-giver knows which words are assigned to which teams. A clue-giver provides a clue, and a guesser on their team selects a subset of board words related to that clue. The team that guesses all of their own words first wins.

In this paper we focus on the task of clue-giving. The clue-giving task is the most interesting, since the clue-giver needs to search the space of all possible words to identify suitable clues, and then rank the clues to select the best one. A blue clue-giver must generate a clue c that is conceptually closest to one subset (here, a pair) of blue words from the set of all possible blue word pairs $I \in B^2$. The clue c must also be sufficiently far from all red words R. Note that in the original Codenames game, a clue-giver can produce a clue corresponding to an arbitrary number of blue words from $I \in B^m$ for any $m \leq N$, but for more consistent evaluation we calculate performance on the clue-giving task considering $m = 2$ only. Our proposed innovations can be applied for arbitrary m.

Figure 3 provides an overview of our proposed methods. In **Section 3.1**, we describe the baseline algorithm for choosing and ranking clues. **Section 3.2** details BabelNet-WSF, our method for querying the very large BabelNet graph and multiple techniques to improve the quality of the returned clues. In **Section 3.3** we propose the DETECT algorithm which leverages document frequency and dictionary embeddings to universally improve Codenames performance across embedding and graph-based clue-givers. Finally, **Section 3.5** explains the extensive human evaluation experiments conducted through Amazon Mechanical Turk.

All of our code for these proposed methods is available for public use.[1]

1. https://github.com/divyakoyy/codenames

3.1 ClueGiver: a Baseline Algorithm that Produces Clues

We propose ClueGiver, an algorithm that gives clues on the basis of a measurement of word similarity $s(w_1, w_2)$.

First, we define $s(w_1, w_2)$ for word embedding-based and knowledge graph-based approaches. For word embeddings, the word similarity $s(w_1, w_2)$ is defined as 1 minus the cosine distance between the word embeddings, *i.e.* $1 - \cos(f(w_1), f(w_2))$, where f is the embedding function. When measuring word similarity with a knowledge graph method such as BabelNet-WSF, the word similarity $s(w_1, w_2)$ is defined as the inverse of the number of edges along the path between the graph node for word w_1 and the graph node for word w_2, *i.e.* $\frac{1}{h(w_1,w_2)+1}$, where h is a function that gives the number of edges along the shortest path between w_1 and w_2.

ClueGiver has two steps. To choose a clue for the blue team, we first calculate the T nearest neighbors of each blue word in $B = \{b_n\}_{n=1}^{N}$. We go through each subset $I \in B^m$ and add the union of the nearest neighbors for every blue word $b \in I$ to a set of candidate clues $\tilde{C} = \{\tilde{c}_{tn}\}_{t=1..T;n=1..N}$. The subset I represents the set of intended words that are meant to match the candidate clue. Every subset of B of size m is therefore considered as a candidate set of intended words. Next, we score each candidate clue $\tilde{c}$ using the following scoring function $g(\cdot)$ which produces a large positive value for good clues and a lower value for bad clues (and thus should be maximized):

$$g(\tilde{c}, I) = \quad \lambda_B \left(\sum_{b \in I} s(\tilde{c}, b) \right) - \lambda_R \left(\max_{r \in R} s(\tilde{c}, r) \right) \tag{1}$$

The final chosen clue c is the candidate clue with the highest score. A candidate clue $\tilde{c}$ will have a high score if it is closest to the subset I (thus making $\lambda_B \big(\sum_{b \in I} s(\tilde{c}, b) \big)$ as large as possible), while remaining as far away as possible from all the red words. The expression $\max_{r \in R} s(\tilde{c}, r)$ means that we calculate the highest similarity between the red words and our candidate clue $\tilde{c}$, which corresponds to the red word closest to $\tilde{c}$. If $\tilde{c}$ is a good clue for the blue team, then even this closest red word is far away from the candidate clue, with a small positive value of $s(\tilde{c}, r)$. In the case that there is no overlap between the nearest neighbors of the blue words, the algorithm will choose a clue for one word. This happened extremely rarely when computing 500 nearest neighbors for each board word. The choices of λ_B and λ_R determine the relative importance of each part of the scoring function, *i.e.*, whether we should prioritize clues that are close to blue words or prioritize clues that are far from red words. We found $\lambda_B = 1$ and $\lambda_R = 0.5$ to be effective values empirically across all word representations. None of the human experiments used the same data that was used to set these parameters. The Codenames boards used to tune the parameters were randomly sampled from the total of $\binom{208}{20} = 3.68\text{e}{+27}$ Codenames boards (208 being the possible board words, and 20 being the board size). The Codenames boards used for human evaluation on AMT were different boards, randomly sampled from all possible boards with each having probability $\frac{1}{3.68e+27}$.

Comparison to the scoring function of Kim et al. (2019) In our experiments, we compare our scoring function $g(\cdot)$ with the following scoring function proposed by Kim et al. (2019). The term λ_T is configurable to limit how aggressive the clue-giver is:

$$g_{kim}(\tilde{c}, I) = \begin{cases} \min_{b \in I} s(\tilde{c}, b), & \text{if } \min_{b \in I} s(\tilde{c}, b) > \lambda_T \text{ and } \min_{b \in I} s(\tilde{c}, b) > \max_{r \in R} s(\tilde{c}, r) \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

The main difference between g_{kim} and our proposed scoring function g is that our scoring function g incorporates a penalty to the score based on the similarity of the closest red word, while g_{kim} enforces the constraint that the similarity between the clue and the furthest blue word must be greater than the similarity to the closest red word. In addition, g_{kim} enforces that the similarity between the furthest blue word and the clue is greater than the threshold λ_T. Notably, g_{kim} would give equal scores (assuming that blue word distances are equal) to clue words that do not violate the constraint, even if one was much closer to a red word than the other, whereas g applies a soft penalization to red words.

3.2 BabelNet-WSF: Solving Codenames Clue-Giving with the BabelNet Knowledge Graph

As discussed previously, knowledge graphs have not been successfully used to play Codenames in prior work. In this section we propose BabelNet-WSF, which includes three innovations to enable high-performance use of the BabelNet knowledge graph for the Codenames clue-giving task, including (1) a method for constructing a nearest neighbor BabelNet subgraph relevant to a particular Codenames board, (2) constraints that filter the nearest neighbors to improve candidate clue quality, and (3) an approach to select a good single word clue from the set of synonymous phrases associated with a particular node.

3.2.1 Constructing Subgraphs of BabelNet to Identify Nearest Neighbors

In order to choose a clue using the scoring function defined in the previous section, it is necessary to identify the nearest neighbors of the board words. When solving Codenames with a knowledge graph like BabelNet, the graph connectivity defines the nearest neighbors. We define the "nearest neighbors" of an origin node as all nodes within 3 edges of the origin node. The challenge arises because the full BabelNet 4.0.1 graph is 29 gigabytes, so recursively traversing it to identify nearest neighbors is computationally prohibitive as noted by Kim et al. (2019).

We propose three steps to enable fast nearest neighbors identification from BabelNet for a particular Codenames board. First, we restrict the relationship edges that are added to the subgraph. Every edge in the BabelNet graph indicates a particular type of relationship (*e.g.* IS-A). For each node in the graph representing a board word, we obtain all outgoing edges for the first edge, but for edges beyond that, we only recursively traverse edges that are in the HYPERNYM relationship group, as discussed in **Section 3.2.2**.

Second, we exclude edges that were automatically generated, because we found these to be of poor quality for the Codenames task. Finally, we cache the nearest neighbor results for each board word, in order to reuse these results for future boards. This caching step is important due to a daily limit in querying the BabelNet API. Appendix **Algorithm 2** details the overall process for obtaining nearest neighbors from BabelNet.

3.2.2 Filtering Nearest Neighbors for BabelNet

Once a nearest neighbor graph is constructed, we apply two filtering steps on the edges in order to retain only the highest quality nearest neighbors, and thereby improve the selected clues.

Hypernym edge constraint beyond the first edge. The first filtering step is to only allow hypernym relationships (IS-A or SUBCLASS-OF) beyond the first edge. The motivation behind this constraint is that traversing the graph using randomly chosen relationship types leads to "nearest neighbors" which are not very related to the origin node. However, using only hypernym relationship types is too restrictive, since exploiting different kinds of relationships is critical in order to perform well at Codenames clue-giving. We found that allowing the first edge to be any relationship type, but restricting all subsequent edges to be a hypernym relationship type maintained diversity of clues while usually preserving conceptual relatedness of the origin node and (possibly distant) neighbor node. Table 1 illustrates the improvement in nearest neighbor quality that this constraint yields.

passes constraint	needle $\xrightarrow{\text{HAS-PART}}$ point
fails constraint	needle $\xrightarrow{\text{HAS-PART}}$ point $\xrightarrow{\text{HAS-KIND}}$ spearhead
passes constraint	litter $\xrightarrow{\text{IS-A}}$ trash $\xrightarrow{\text{IS-A}}$ waste
fails constraint	litter $\xrightarrow{\text{IS-A}}$ trash $\xrightarrow{\text{HAS-KIND}}$ scrap metal
passes constraint	mouse $\xrightarrow{\text{IS-A}}$ pointing device $\xrightarrow{\text{IS-A}}$ input device
fails constraint	mouse $\xrightarrow{\text{IS-A}}$ pointing device $\xrightarrow{\text{HAS-KIND}}$ light gun

Table 1: Examples of hypernym edge constraint beyond the first edge. Hypernym relationships are IS-A or SUBCLASS-OF. Words in red failed the constraint and are filtered out of the graph.

Same-type edge constraint. The second constraint restricts all edges after the first edge to be the same relationship type. In combination with the hypernym constraint, that means a traversal after the first edge of IS-A/IS-A/IS-A is allowed, a traversal after the first edge of SUBCLASS-OF/SUBCLASS-OF/SUBCLASS-OF is allowed, but any traversal after the first edge randomly combining IS-A and SUBCLASS-OF is forbidden. We found that this additional stringency improved the relevance of retrieved nearest neighbors. Examples are shown in Table 2.

3.2.3 Selecting a Good Single-Word Clue from a Multi-Word Synset

Once the nearest neighbors have been filtered, a final clue must be selected. In the Codenames task, the clue must consist only of a single word. However, each node of the BabelNet graph is a synset, which is a group of related concepts organized as a "main sense label" with "other sense labels." To add to the complexity, each label can be one or more words. For example from Table 3, the synset with the definition "material used to provide

passes constraint	litter $\xrightarrow{\text{IS-A}}$ animal group $\xrightarrow{\text{IS-A}}$ biological group $\xrightarrow{\text{IS-A}}$ group
passes constraint	litter $\xrightarrow{\text{IS-A}}$ animal group $\xrightarrow{\text{SUBCLASS-OF}}$ fauna
fails constraint	litter $\xrightarrow{\text{IS-A}}$ animal group $\xrightarrow{\text{SUBCLASS-OF}}$ fauna $\xrightarrow{\text{IS-A}}$ aggregation
passes constraint	moon $\xrightarrow{\text{GLOSS-RELATED}}$ planet $\xrightarrow{\text{IS-A}}$ celestial body $\xrightarrow{\text{IS-A}}$ natural object
fails constraint	moon $\xrightarrow{\text{GLOSS-RELATED}}$ planet $\xrightarrow{\text{SUBCLASS-OF}}$ planemo $\xrightarrow{\text{IS-A}}$ object
passes constraint	figure $\xrightarrow{\text{GLOSS-RELATED}}$ diagram $\xrightarrow{\text{IS-A}}$ drawing $\xrightarrow{\text{IS-A}}$ representation
fails constraint	figure $\xrightarrow{\text{GLOSS-RELATED}}$ diagram $\xrightarrow{\text{SUBCLASS-OF}}$ graphics $\xrightarrow{\text{IS-A}}$ visual communication

Table 2: Examples of same-type edge constraint on edges after the first edge.

a bed for animals" has a main sense label of "bedding," and other sense labels "litter" and "bedding material." The synset with the main sense label "creative work" has other sense labels "artwork," "work," and "work of art." It is not immediately obvious how to extract a good-quality single word clue from a synset.

Main sense label	Definition	Other labels
bedding	Material used to provide a bed for animals	litter, bedding material
bedding	Coverings that are used on a bed	bedclothes, bed clothing
stringed instrument	A musical instrument in which taut strings provide the source of sound	string instrument, chordophone
creative work	A creative work is a manifestation of creative effort including fine artwork, writing, filmmaking, and musical composition	creative work, artwork, work, work of art

Table 3: Examples of synsets with multi-word labels

Selecting a single word at random from a synset is not effective. For example "work" on its own is not an ideal choice for "creative work" and "material" is not an ideal choice for "bedding material." We develop a scoring system to select the best possible single word clue from a synset. First, we keep only single words that belong to the intersection of the nearest neighbors of two board words. Taking a simplified example from Figure 3 (bottom), for the board words "gold," having neighbor synset with labels {"shades of yellow," "variations of yellow"}, and "lemon," having neighbor synset with labels {"yellow," "yellowness," "color yellow"}, the word "yellow" is kept.

We also apply weights (which are parameters of our method) on each of the words comprising the synset labels. The weights on these words are denoted by w_1, w_2, w_3, and

w_4 where $w_1 \leq w_2 \leq w_3 \leq w_4$, and they are assigned based on the type of synset label and whether the label has multiple words, as shown in **Table 4**. The number of edges $h(\tilde{c}, w)$ is multiplied by the weight; thus a label type with a lower weight, such as a main sense single word, is desirable. Empirically, we found the values $w_1 = 1, w_2 = 1.1, w_3 = 1.1, w_4 = 1.2$ to be effective. Appendix **Algorithm 1** details the process for obtaining single-word clues and their corresponding weights.

label type	single- or multi- word	weight
main sense	single	w_1
main sense	multi	w_2
other sense	single	w_3
other sense	multi	w_4

Table 4: Weights applied to labels taken from BabelNet depend on whether the label is a main sense or other sense label (this distinction is provided by BabelNet), and whether that label is composed of one or more words.

3.2.4 Example BabelNet-WSF Clue

The combination of the three aforementioned innovations enables selection of high-quality clues from BabelNet, as exemplified in Figure 4.

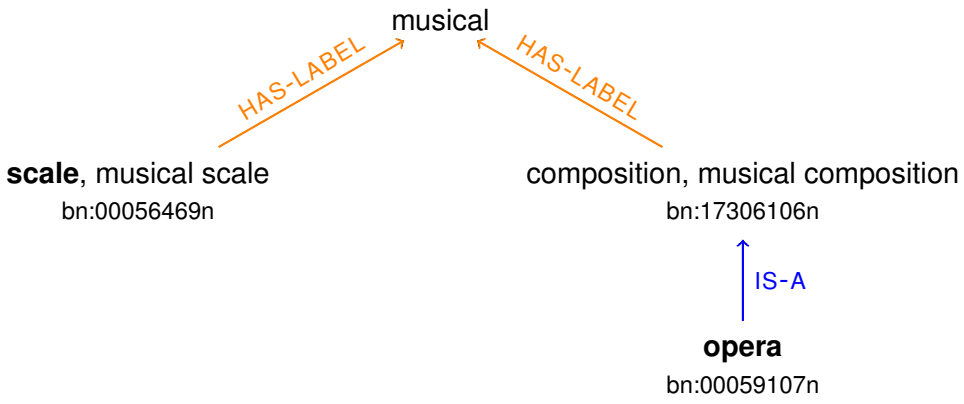

Figure 4: Sub-graph of BabelNet showing how the clue "musical" was chosen for the board words "scale" and "opera" using BabelNet-WSF. The HAS-LABEL edges are not real edges in BabelNet, but rather single word labels that we extract using the single-word clue approach described in Section 3.2.3. Synsets from BabelNet are annotated with their ID from BabelNet 4.0.1 as bn:synset-id.

3.3 DETECT for Improving Clue Quality

The previous sections focused on obtaining good clues from BabelNet. In this section, we describe a method called DETECT that improves the quality of clues for both BabelNet-

WSF and word embedding-based methods. We originally developed DETECT to address deficiencies in clues produced by word embedding-based methods, and then discovered that DETECT also improved BabelNet-WSF clues. In our experiments, we show results of word embedding and BabelNet-WSF methods with and without DETECT.

We identified three deficiencies in the clues chosen via word embeddings: (1) obscure clues, (2) overly generic clues, and (3) lack of clues exploiting many common sense word relationships. **Table 5** provides real examples of clues that suffer from deficiencies (1) and (2). Obscure clues are not necessarily "incorrect" in the way they connect two words: for example the word "djedkare" in **Table 5** refers to the name of the ruler of Egypt in the 25th century B.C., and therefore correctly connects the words "egypt" and "king." However this clue does not reflect the average person's knowledge of the English language and is likely to yield random guesses if presented to a human player. Overly generic clues, in contrast, are more likely to match too many board words. For example "artifact" in **Table 5** connects "key" and "pipe," but also matches other red words on the board for that trial, such as "crown" and "racket."

Clue	Intended Words to Match Clue
aether	jupiter, vacuum
djedkare	egypt, king
machine	key, crown
artifact	key, pipe

Table 5: Examples of obscure and generic clues. Obscure clues include "aether," which in ancient and medieval science is a material that fills the region of the universe above the terrestrial sphere, and "djedkare" which is the name of the ruler of Egypt in the 25th century B.C. Overly generic clues include "machine" and "artifact", which often apply to many board words.

To address all three issues in clue quality, we added a scoring metric, DETECT, to our scoring function $g(\cdot)$. DETECT includes two parts, a function $FREQ(w)$ that uses document frequency to exclude too-rare and too-common words, and a function $DICT(w_1, w_2)$ that encourages clues relying on common sense word relationships.

Leveraging document frequency with FREQ. In order to penalize overly rare as well as overly generic tokens, we leverage the document frequency f_w of a word, which indicates the count of documents in which word w is found in a cleaned subset of Wikipedia. We calculate $FREQ(w)$ as:

$$FREQ(w) = -\begin{cases} \frac{1}{df_w} & \text{when } \frac{1}{df_w} \geq \alpha \\ 1 & \text{when } \frac{1}{df_w} < \alpha. \end{cases} \tag{3}$$

$FREQ(w)$ penalizes rare words more and common words less, unless a word is so common that the inverse function of its document frequency is lower than a value α, which is an algorithm parameter. α was chosen empirically based on the distribution of document frequencies in a cleaned subset of the Wikipedia corpus as shown in **Figure 5**, together with the clues produced across all algorithms. α represents the upper bound document

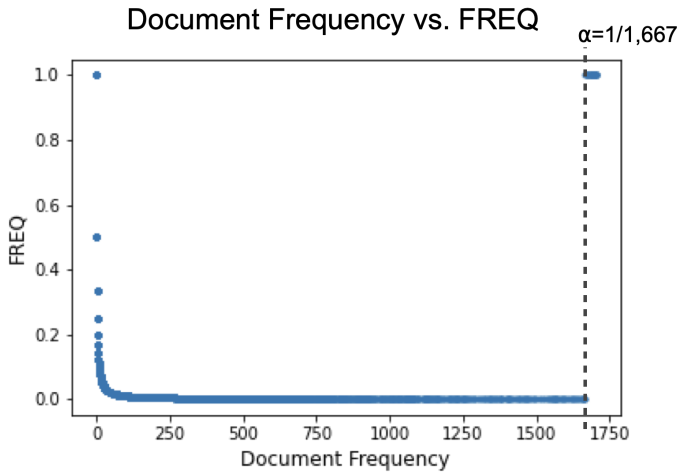

Figure 5: FREQ is a function of document frequency that is used to penalize rare words more and common words less unless the word is so common that it is not useful as a clue word. α was chosen by picking an upper bound document frequency and validating empirically across clue-giving algorithms. Note that $\alpha = 1/1,667$ was calculated on a sample of 1,701 cleaned documents from (Mahoney, 2020).

frequency at which point a word is considered too common to be useful as a clue word. Jaramillo et al. (2020) used TF-IDF as a standalone baseline method, whereas in this work $FREQ$ is a term which is part of the full scoring function.

In principle, **Equation 1** should filter out overly generic clues that match red words via the penalty we apply to red words. However we found that in practice, performance improved by removing common words with the $FREQ$ component of DETECT. In BabelNet, the limitation is that the number of edges connecting a common word to all of its children is highly variable (e.g. the number of edges between "apple" and "object" is 4, vs. the number of edges between "cash" and "object" is 1).

Leveraging dictionary embeddings with DICT. Since most embedding methods rely on a word's context to generate its vector representation, they do not always encode important relationships such as meronymy and hypernymy. Dict2Vec (Tissier et al., 2017) addresses this issue by computing vector representations of words using dictionary definitions. Dict2Vec also identifies strong pairs of words as words that appear in each other's dictionary definition – for example, "car" might appear in the definition of "vehicle" and vice versa, making "car"/"vehicle" a strong pair. Synonymy, hypernymy, hyponymy and meronymy are all relationships captured in dictionary definitions and relationships that contribute to high quality clues, so incorporating Dict2Vec into our scoring function allowed us to more heavily weight clues that are semantically related in ways that context alone cannot capture.

We define $DICT(w_1, w_2)$ as the cosine distance between the Dict2Vec word embeddings for two words w_1 and w_2.

The DETECT score. Both term relevance $FREQ(w)$ and dictionary relevance $DICT(w_1, w_2)$ are incorporated into a new weighting term, DETECT:

$$DETECT(\tilde{c}) = \lambda_F FREQ(\tilde{c}) + \lambda_D \Bigg(\sum_{b \in I} 1 - DICT(\tilde{c}, b) - \max_{r \in R} \big(1 - DICT(\tilde{c}, r)\big) \Bigg). \quad (4)$$

A candidate clue $\tilde{c}$ will have a high value for $DETECT(\tilde{c})$ if it is a more common word (without being so common that it falls above α in **Equation 3**), and is close to as many of the intended blue words as possible in the Dict2Vec embedding space (therefore making $\sum_{b \in I} 1 - DICT(\tilde{c}, b)$ as large as possible) while remaining as far as possible from red words. $DETECT(\tilde{c})$ is added to the scoring function $g(\cdot)$ of **Equation 1** and $g_{kim}(\cdot)$ of **Equation 2** to re-weight candidate clues. We found $\lambda_F = 2$, $\lambda_D = 1$ for GloVe filtered on the top 10k English words, and $\lambda_D = 2$ for all other representations, to be most effective empirically.

The word embeddings (word2vec, GloVe, fastText, BERT) were obtained from publicly available collections of pre-trained vectors based on large corpora such as Wikipedia or Google News. The word2vec, GloVe, and fastText vectors were obtained from the gensim library (Řehůřek & Sojka, 2010), and BERT contextualized embeddings were obtained from a pre-trained BERT model (bert_12_768_12, book_corpus_wiki_en_uncased) made available in the GluonNLP package (Guo et al., 2020). DETECT leverages additional data sources summarized in different ways to improve clue choosing. The Dict2Vec component of DETECT uses dictionary definitions from Cambridge, Oxford Collins, and dictionary.com, and an embedding method to summarize this data. The $FREQ(w)$ component of DETECT uses a cleaned subset of Wikipedia (Mahoney, 2020).

3.4 Using Contextual Embeddings

This section describes our final methodological contribution to improve Codenames clue-giving: how to produce a clue using a contextual embedding method. In "classical" word embedding methods such as GloVe (Pennington, Socher, & Manning, 2014) and word2vec (Mikolov, Sutskever, Chen, Corrado, & Dean, 2013), each word is associated with only one vector representation, so the Codenames scoring function in **Equation 1** can be computed directly. However, contextual embedding methods like BERT (Devlin, Chang, Lee, & Toutanova, 2019) only produce a representation for a word in context. Therefore the word "running" in the following sentences, "She was running to catch the bus," and "She was running for president," are captured in their respective contexts. In order to compute the Codenames scoring function using BERT embeddings, we averaged over different contexts to produce a single embedding for each word. Specifically, we extracted contextualized embeddings from a pre-trained BERT model (bert_12_768_12, book_corpus_wiki_en_uncased), made available by the GluonNLP package (Guo et al., 2020), using a cleaned subset of English Wikipedia 2006 (Mahoney, 2020).

Then we defined a word's BERT embedding as the average over that word's contextual embeddings. An approximate nearest neighbor graph was produced from the final embeddings using the Annoy library.[2] Averaging allowed us to reduce noise caused by outlier contextual embeddings of a word, but in the future we could also experiment with clustering of contextual embeddings and using those clusters to construct a nearest neighbor graph.

2. https://github.com/spotify/annoy

3.5 Human Evaluation of Clue-Giving Performance

We use Amazon Mechanical Turk (AMT) for human evaluation of Codenames clue-giving algorithms.

3.5.1 General Performance Comparison and Efficacy of DETECT

We compare five Codenames clue-giving algorithms, based on our proposed scoring function $g(\cdot)$ (**Equation 1**): (1) BabelNet-WSF, (2) word2vec (Mikolov et al., 2013), (3) GloVe (Pennington et al., 2014), (4) GloVe-10k (filtering for the 10k most common words), (5) fastText (Bojanowski et al., 2017), and (6) BERT (Devlin et al., 2019). **Table 6** summarizes all methods we used in our experiments, including baseline methods from Kim et al. (2019), along with all of our new proposed methods and variations on those methods. The symbol "✓" indicates something novel to the paper, whether it is a new use of a method (first column), a new representation of words (second column), or uses our new ranking method for clues (third column). This table includes the 3 baselines used in Kim et al. (2019), as well as 9 new methods proposed in the paper. In our experiments, we evaluate each algorithm with and without DETECT (**Section 3.3**), and with our scoring function and the Kim et al. (2019) scoring function, for a total of 24 configurations. BabelNet-WSF includes the innovations described in **Section 3.2**. BERT includes the innovations of **Section 3.4**.

Word Representation	new use in Codenames $g_{kim}(\cdot)$	new use in Codenames $g(\cdot)$	new knowledge graph method in Codenames	new ranking method
BabelNet-WSF	✓	✓	✓	no
BabelNet-WSF+DETECT	✓	✓	✓	✓
BERT	✓	✓	no	no
BERT+DETECT	✓	✓	no	✓
fastText	✓	✓	no	no
fastText+DETECT	✓	✓	no	✓
GloVe	✓	✓	no	no
GloVe+DETECT	✓	✓	no	✓
GloVe-10k	no	✓	no	no
GloVe-10k+DETECT	✓	✓	no	✓
word2vec	✓	✓	no	no
word2vec+DETECT	✓	✓	no	✓

Table 6: A summary of all methods from our experiments, both baseline methods from Kim et al. (2019) as well as new proposed methods and the variations on those methods. The first column indicates whether an algorithm is a new method for Codenames using $g_{kim}(\cdot)$ scoring function, the second column indicates whether it is a new method for Codenames using $g(\cdot)$ scoring function, the third column indicates whether a new knowledge graph method is used, and the fourth column indicates the use of a new method for ranking clues.

To compare the algorithms, 60 unique Codenames boards of 20 words each were randomly generated from a list of 208 words obtained from the official Codenames cards. We used words from the official Codenames cards because these words were carefully selected by the game designers to have interesting, inter-related multiple meanings, and as such these words are an integral part of the game definition.

For a given Codenames board, each algorithm was asked to output the best clue and the two words intended to match the clue. These two words are the blue words that the algorithm based a particular clue on, which a human guesser is intended to select.

For human evaluation, United-States-based AMT workers with a high approval rate ($\geq 98\%$) on previous AMT tasks were asked to look at a full board of 20 words and 1 clue word, and rank the top four board words matching the given clue. The user was required to fill in Ranks 1 and 2, because the algorithm always intended 2 words to be selected. Ranks 3 and 4 were optional for the user to fill in; the AMT worker could specify "no more related words" for these ranks. Ranks 3 and 4 were included to distinguish between situations where the algorithm produced a wholly irrelevant clue (such that the worker could not guess the intended words even if given 4 slots) and a sub-optimal clue (such that the worker could guess intended words in Ranks 3 or 4, but not Ranks 1 or 2). The AMT workers had no knowledge of the underlying algorithm and no knowledge of the algorithm's "intended words." A high-performing algorithm chooses such good quality clues that the AMT workers correctly guess the intended words and place them in Ranks 1 and 2.

3.5.2 Comparison with Kim et al. (2019)

In order to compare with the work of Kim et al. (2019), we also evaluated the performance of each algorithm using their scoring function $g_{kim}(\cdot)$ (**Equation 2**), using the same set of Codenames boards. In the original formulation of the Kim et al. (2019) scoring function shown in **Equation 2**, the constraints $\min_{b \in I} s(\tilde{c}, b) > \lambda_T$ and $\min_{b \in I} s(\tilde{c}, b) > \max_{r \in R} s(\tilde{c}, r)$ were used. Since we evaluate performance on the clue-giving task considering $m = 2$ for consistency of evaluation across all algorithms, this constraint was relaxed if there were no clue words passing those constraints for $m = 2$ blue words.

In addition, Kim et al. (2019) restricts to the top 10k common English words. We tested this using the GloVe model filtered on the top 10k common words, and refer to this as GloVe-10k in our reported results.

3.5.3 Performance Metrics

Each trial from Amazon Mechanical Turk provided the 2 to 4 board words (ranked) that the AMT worker selected as most related to the given clue word. To quantify the performance of different algorithms, we calculated precision@2 and recall@4. Precision@2 measures the number of correct words that an AMT worker guessed in the first 2 ranks, where correct words are the intended words that the algorithm meant the worker to select for a given clue. Recall@4 measures the number of correct words chosen in the first 4 ranks.

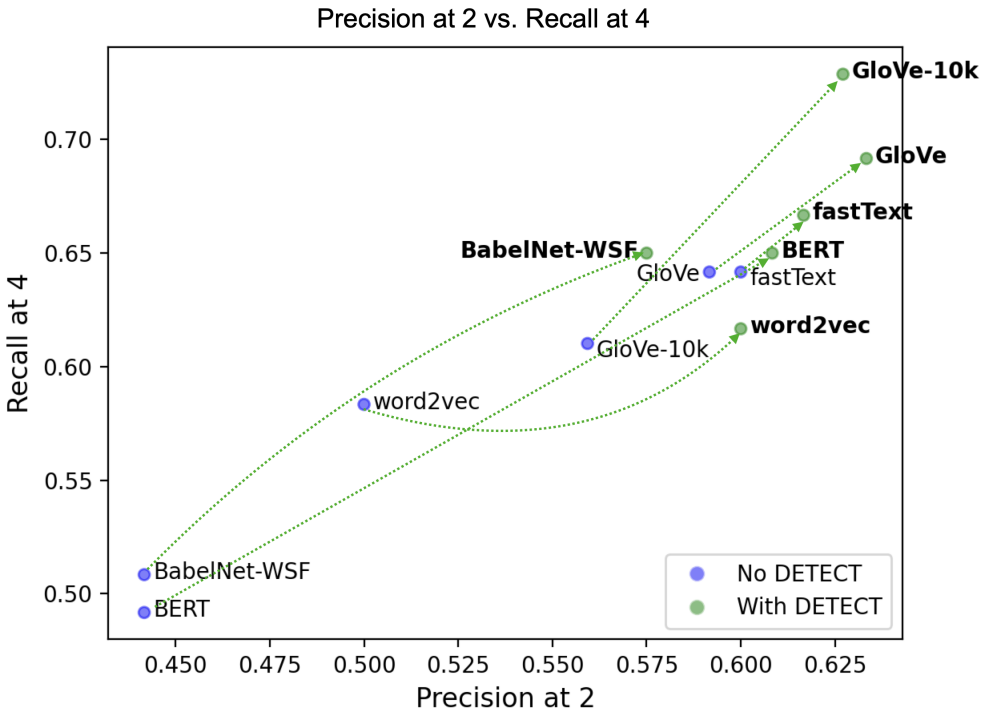

Figure 6: Precision@2 (for the 2 intended words chosen by the algorithm for a clue) vs. Recall@4 (for the 4 words that guessers could answer) from Amazon Mechanical Turk results using our proposed scoring function $g(\cdot)$.

4. Results

Results from the different algorithms with our scoring function are shown in **Figure 6**, while results for the different algorithms with the Kim et al. (2019) scoring function are shown in **Figure 7**. Additional results from the AMT evaluation are reported in **Appendix C**. The results are summarized as follows.

- Some of our methods surpass state-of-the-art performance for the Codenames clue-giving task. The best performing algorithm for precision@2 across both scoring functions is fastText+DETECT, with 66.67% precision@2. **fastText+DETECT outperforms the prior state-of-the-art by Kim et al. (2019) using GloVe-10k, which has precision 55.93% as shown in Figure 7.**

- **Our proposed DETECT algorithm leads to universal improvement across all word representations** with a median percent improvement of 18.0% for precision@2. DETECT also leads to improvement across both scoring functions, with a median 16.1% improvement for our scoring function and even higher median improvement of 20.3% for the Kim et al. (2019) scoring function. For BERT, the advantage of using DETECT is the most substantial, with a 102.8% improvement in precision@2.

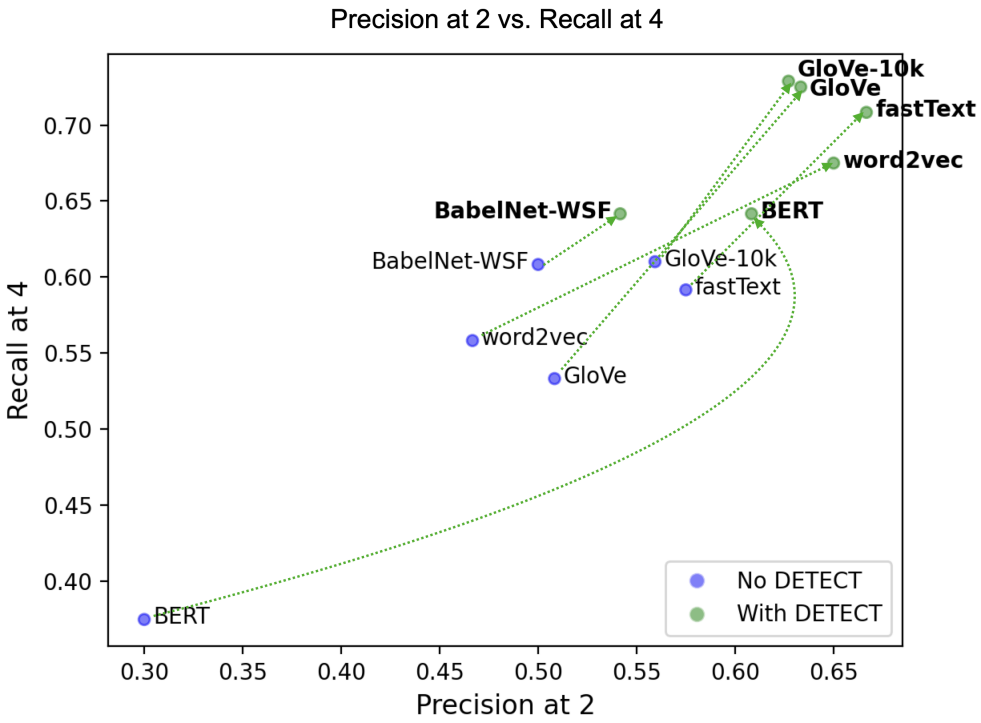

Figure 7: Precision@2 (for the 2 intended words chosen by the algorithm for a clue) vs. Recall@4 (for the 4 words that guessers could answer) from Amazon Mechanical Turk results using the Kim et al. (2019) scoring function.

- When DETECT is used, our scoring function leads to better performance for BabelNet-WSF, while the Kim et al. (2019) scoring function leads to better performance for word2vec and fastText, and both performed equally on GloVe, GloVe-10k, and BERT (precision@2). Thus, neither scoring function is definitively superior. However, since BabelNet-WSF is more interpretable and easier to troubleshoot than the word embedding methods, we suspect that our scoring function (which performs better on BabelNet-WSF) might be more useful in settings where one might want to debug and improve performance for the system.

- **We report the first successful use of a knowledge graph to solve the Codenames task**, with 57.5% precision@2 for BabelNet-WSF with our scoring function, comparable to the performance of word embedding-based methods.

5. Discussion

Codenames is a task that is difficult even for humans, who sometimes struggle to generate clues for particular boards. As with all games, the element of difficulty is necessary in order to produce an exciting challenge. Codenames relies on a deep understanding of language. Traditional language tasks often focus on one axis of language understanding

such as analogies or part-of-speech tagging, while Codenames requires leveraging many different axes of language, including common sense relationships between words.

In this work, we propose several innovations that improve performance on the Codenames clue-giving task. First, we enable successful clue-giving from a knowledge graph, BabelNet-WSF, via new techniques for sub-graph construction, nearest neighbor filtering, and single word clue selection from multi-word synset phrases. Our innovations enable BabelNet-WSF performance that is comparable to word embedding-based methods, while retaining the advantage of full interpretability due to the underlying graph structure. Next, we propose DETECT, a score that combines document frequency and Dict2Vec embeddings to eliminate too-rare or too-common potential clues while incorporating more diverse word relationships. DETECT improves performance across all algorithms and scoring functions. Finally, we complete the first large-scale human evaluation of Codenames algorithms on Amazon Mechanical Turk, to accurately evaluate the real-world performance of our algorithms. Overall, our proposed methods yield state-of-the-art performance on Codenames and advance the formal study of word games.

Acknowledgments

Divya Koyyalagunta and Anna Sun contributed equally to this work.

Appendix A. Examples

Figures 8-11 show examples of clues chosen by all the experimental configurations for four Codenames boards.

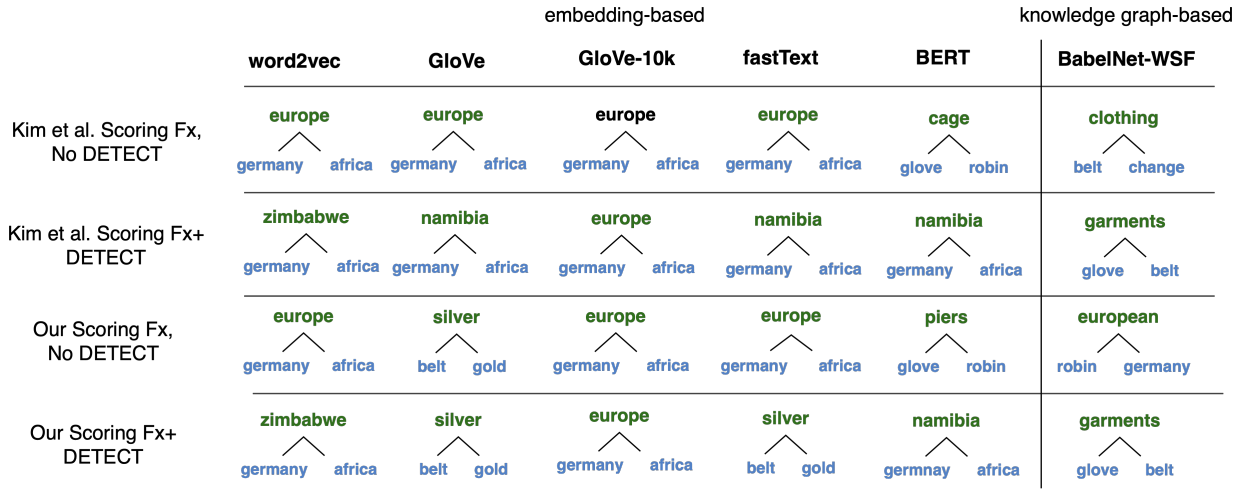

Figure 8: The clues chosen (in black for baselines and green for our methods) and the intended board words chosen for that clue (in blue) for all experimental methods. This includes the 6 word representations (word2Vec, GloVe, GloVe-10k, fastText, BERT, BabelNet-WSF), 2 scoring functions (ours and Kim et al.'s), and with and without DETECT applied. The board words for this trial were blue = germany, car, change glove, needle, robin, belt, board, africa, gold; red = pipe, kid, key, boom, satellite, tap, nurse, pyramid, rock, bark.

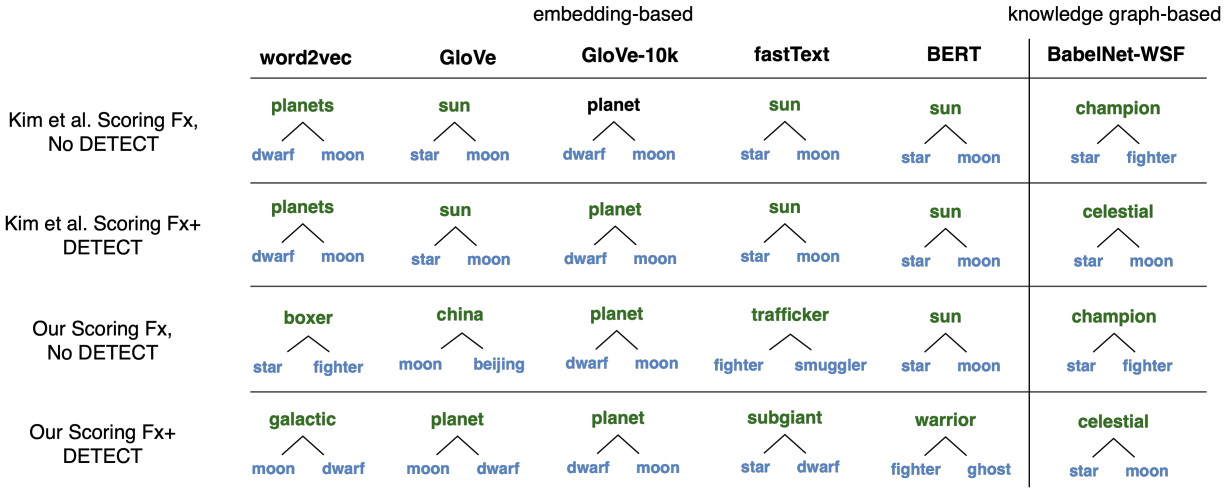

Figure 9: The clues chosen (in black for baselines and green for our methods) and the intended board words chosen for that clue (in blue) for all experimental methods. The board words for this trial were blue = dwarf, foot, moon, star, ghost, beijing, fighter, roulette, alps; red = club, superhero, mount, bomb, knife, belt, robot, rock, bar, lab.

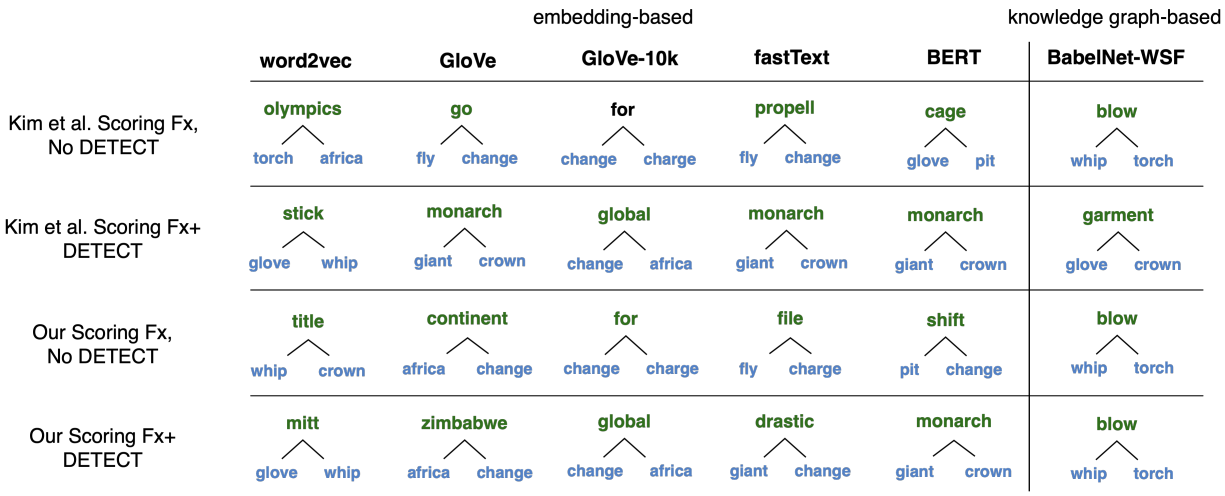

Figure 10: The clues chosen (in black for baselines and green for our methods) and the intended board words chosen for that clue (in blue) for all experimental methods. The board words for this trial were blue = crown, pit, change, glove, charge, torch, whip, fly, africa, giant; red = amazon, hole, shark, ground, shop, cast, nurse, server, vacuum, rock.

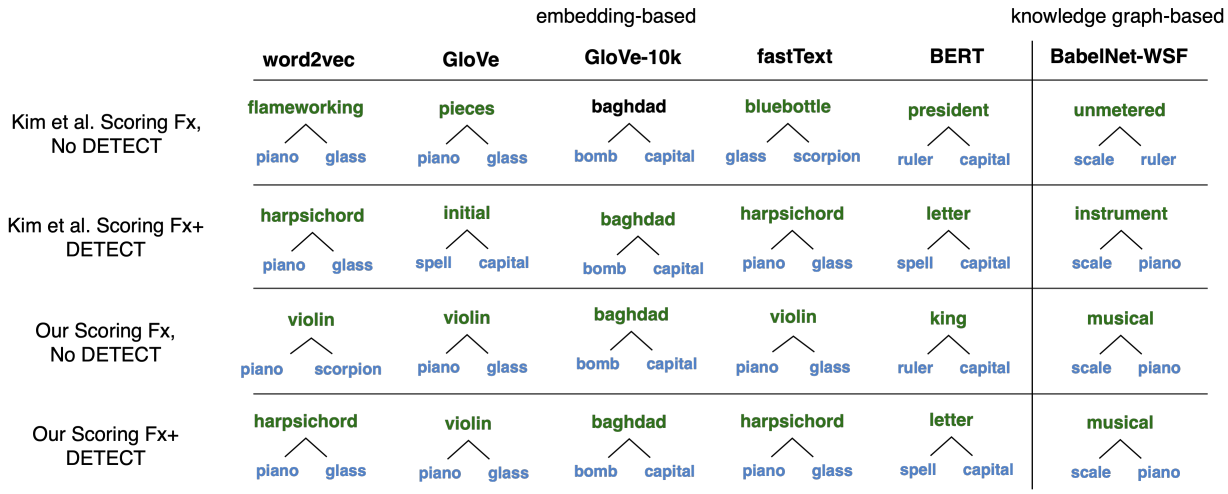

Figure 11: The clues chosen (in black for baselines and green for our methods) and the intended board words chosen for that clue (in blue) for all experimental methods. The board words for this trial were blue = amazon, spell, ruler, scale, round, bomb, piano, glass, capital, scorpion; red = paste, air, ground, cold, lemon, belt, torch, point, saturn, game.

Appendix B. Algorithms

Algorithm 1 details the algorithm for getting single word clues from BabelNet, as described in **Section 3.2.3**. **Algorithm 2** details how nearest neighbors are queried for and cached from BabelNet, as described in **Section 3.2.1**.

Algorithm 1: Extracting single-word clues for a synset

Input : *mainSense*, a string representing the main sense label of a synset. *otherSenses*, a list of strings representing the other sense labels of the synset. In both *mainSense* and *otherSenses*, multi-word clues delimited by the '_' character. w_1, w_2, w_3, w_4, corresponding to weights described in **Table 4**

Output: a dictionary of single word labels and scores corresponding to the configured weights for each label type

```
begin
    singleWordLabels ⟵ ∅ ;                          ▷ Initialize singleWordLabels as an empty set
    splitMainSense = SPLIT(mainSense,'_') ;          ▷ Split main sense label using delimiter _
    if len(splitMainSense) = 1 then                  ▷ Main sense label is a single word
        singleWordLabels[splitMainSense[0]] = w1;
    end
    else                                             ▷ Main sense label is multiple words
        for word ∈ splitMainSense do                 ▷ Set each word's weight to w2
            singleWordLabels[word] = w2;
        end
    end
    for sense ∈ otherSenses do                       ▷ Iterate over other sense labels
        splitOtherSense = SPLIT(sense,'_') ;         ▷ Split other sense label using delimiter _
        if len(splitOtherSense) = 1 then             ▷ Other sense label is a single word
            singleWordLabels[splitOtherSense[0]] = w3;
        end
        else                                         ▷ Other sense label is multiple words
            for word ∈ splitOtherSense do            ▷ Set each word's weight to w4
                singleWordLabels[word] = w4;
            end
        end
    end
    return singleWordLabels;
end
```

Algorithm 2: Querying BabelNet Edges

Input : *lemmaSynsets*, a dictionary of synsets for each board word. B and R, the set of our team's and the other team's board words, respectively. L, number of levels (the number of edges from source word) to query

Output: a dictionary of board words and synset edges at each level of edges to be cached

```
begin
    for word ∈ B ∪ R do        ▷ Iterate over the union of blue and red teams' board words
        synsets ⟵ lemmaSynsets[word] ;        ▷ Get the synsets mapped to board word
        for level ⟵ 1 to L do        ▷ Repeat for each level from 1 to L, where level is the
         number of edges from the board word
            nextLevelSynsets ⟵ ∅ ;        ▷ Initialize nextLevelSynsets to empty set
            for synset ∈ synsets do
                edges ⟵ getOutgoingEdges(synset) ;  ▷ Get the outgoing edges from
                 this synset in the BabelNet graph
                D[word][synset][level] ⟵ edges ;  ▷ Store the edges mapped to this board
                 word, synset, and level
                for edge ∈ edges do
                    if !isAutomatic(edge) and relationGroup(edge) =
                     HYPERNYM then
                        nextLevelSynsets ⟵ nextLevelSynsets+
                         getConnectedSynset(edge) ;  ▷ If the edge is non-automatic
                         and in relation group HYPERNYM, add the connected synset to the set
                         of synsets to query for the next level
                    end
                end
            end
            synsets ⟵ nextLevelSynsets;
        end
    end
    return D;
end
```

Appendix C. Amazon Mechanical Turk Results

Table 7 shows Amazon Mechanical Turk results for 1,440 trials, using our scoring function and Kim et al. (2019)'s scoring function, as well as with and without applying DETECT to the clue selection algorithm.

	$g(\cdot)$		$g_{kim}(\cdot)$	
Word Representation	Precision@2	Recall@4	Precision@2	Recall@4
BabelNet-WSF	0.442	0.508	0.500	0.608
BabelNet-WSF+DETECT	**0.575***	**0.650***	0.542	0.642
BERT	0.442	0.492	0.3	0.375
BERT+DETECT	**0.608***	**0.65***	**0.608***	**0.642***
fastText	0.6	0.642	0.575	0.592
fastText+DETECT	0.617	0.667	0.667	0.708
GloVe	0.592	0.642	0.508	0.533
GloVe+DETECT	0.633	0.692	**0.633***	**0.725***
GloVe-10k	0.560	0.61	0.56	0.61
GloVe-10k+DETECT	0.627	**0.729***	0.627	**0.729***
word2vec	0.5	0.583	0.467	0.558
word2vec+DETECT	0.6	0.617	**0.65***	0.675

Table 7: Amazon Mechanical Turk results for our proposed scoring function $g(\cdot)$ and Kim et al. (2019) scoring function $g_{kim}(\cdot)$ for intended word precision@2 (number of intended words chosen in the first 2 ranks/2) and intended word recall@4 (number of intended words chosen in the first 4 ranks/2). * indicates statistical significance ($p < 0.05$) between the word representation and word representation +DETECT using a z-test.

References

Agirre, E., Alfonseca, E., Hall, K., Kravalova, J., Pasca, M., & Soroa, A. (2009). A study on similarity and relatedness using distributional and WordNet-based approaches. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pp. 19–27, Boulder, Colorado. Association for Computational Linguistics.

Ashktorab, Z., Dugan, C., Johnson, J., Pan, Q., Zhang, W., Kumaravel, S., & Campbell, M. (2021). Effects of communication directionality and AI agent differences in human-AI interaction. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, CHI '21. Association for Computing Machinery.

Atkinson, T., Baier, H., Copplestone, T., Devlin, S., & Swan, J. (2019). The text-based adventure AI competition. *IEEE Transactions on Games*, *11*(3), 260–266.

Bojanowski, P., Grave, E., Joulin, A., & Mikolov, T. (2017). Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*,

5, 135–146.

Budanitsky, A., & Hirst, G. (2006). Evaluating WordNet-based measures of lexical semantic relatedness. *Computational linguistics*, *32*(1), 13–47.

Campbell, M., Hoane Jr, A. J., & Hsu, F.-h. (2002). Deep blue. *Artificial Intelligence*, *134*(1-2), 57–83.

Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., & Kuksa, P. (2011). Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, *12*, 2493–2537.

De Deyne, S., Navarro, D. J., Perfors, A., Brysbaert, M., & Storms, G. (2019). The "small world of words" english word association norms for over 12,000 cue words. *Behavior research methods*, *51*(3), 987–1006.

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT*, pp. 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Devlin, J., Zbib, R., Huang, Z., Lamar, T., Schwartz, R., & Makhoul, J. (2014). Fast and robust neural network joint models for statistical machine translation. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1370–1380.

Guo, J., He, H., He, T., Lausen, L., Li, M., Lin, H., Shi, X., Wang, C., Xie, J., Zha, S., Zhang, A., Zhang, H., Zhang, Z., Zhang, Z., Zheng, S., & Zhu, Y. (2020). Gluoncv and gluonnlp: Deep learning in computer vision and natural language processing. *Journal of Machine Learning Research*, *21*(23), 1–7.

Hirst, G., St-Onge, D., et al. (1998). Lexical chains as representations of context for the detection and correction of malapropisms. *WordNet: An electronic lexical database*, *305*, 305–332.

Iacobacci, I., Pilehvar, M. T., & Navigli, R. (2016). Embeddings for word sense disambiguation: An evaluation study. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 897–907.

Jaramillo, C., Charity, M., Canaan, R., & Togelius, J. (2020). Word Autobots: Using transformers for word association in the game codenames. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, 16(1)*, pp. 231–237.

Kim, A., Ruzmaykin, M., Truong, A., & Summerville, A. (2019). Cooperation and codenames: Understanding natural language processing via codenames. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, 15(1)*, pp. 160–166.

Mahoney, M. (2011 (accessed October 3, 2020)). About the test data.. http://mattmahoney.net/dc/textdata.html.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. *Advances in Neural Information Processing Systems*, *26*, 3111–3119.

Navigli, R., Jurgens, D., & Vannella, D. (2013). Semeval-2013 task 12: Multilingual word sense disambiguation. In *Proceedings of the 7th International Workshop on Semantic Evaluation (SemEval), in the Second Joint Conference on Lexical and Computational Semantics (*SEM 2013)*, pp. 222–231.

Navigli, R., & Ponzetto, S. P. (2012). Babelrelate! a joint multilingual approach to computing semantic relatedness. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence (AAAI 2012)*.

Pennington, J., Socher, R., & Manning, C. D. (2014). GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1532–1543.

Řehůřek, R., & Sojka, P. (2010). Software framework for topic modelling with large corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pp. 45–50, Valletta, Malta. ELRA.

Rovatsos, M., Gromann, D., & Bella, G. (2018). The Taboo challenge competition. *AI Magazine*, *39*(1), 84–87.

Schnabel, T., Labutov, I., Mimno, D., & Joachims, T. (2015). Evaluation methods for unsupervised word embeddings. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pp. 298–307.

Shen, J. H., Hofer, M., Felbo, B., & Levy, R. (2018). Comparing models of associative meaning: An empirical investigation of reference in simple language games. In *Proceedings of the 22nd Conference on Computational Natural Language Learning*, pp. 292–301, Brussels, Belgium. Association for Computational Linguistics.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., et al. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, *529*(7587), 484–489.

Speer, R., Chin, J., & Havasi, C. (2017). Conceptnet 5.5: An open multilingual graph of general knowledge. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, AAAI'17, p. 4444–4451. AAAI Press.

Thawani, A., Srivastava, B., & Singh, A. (2019). Swow-8500: Word association task for intrinsic evaluation of word embeddings. In *Proceedings of the 3rd Workshop on Evaluating Vector Space Representations for NLP*, pp. 43–51.

Tissier, J., Gravier, C., & Habrard, A. (2017). Dict2vec : Learning word embeddings using lexical dictionaries. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pp. 254–263.

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., et al. (2019). Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, *575*(7782), 350–354.

Xu, Y., & Kemp, C. (2010). Inference and communication in the game of password. In *Advances in Neural Information Processing Systems*, pp. 2514–2522.

Yannakakis, G. N., & Togelius, J. (2018). *Artificial Intelligence and Games*, Vol. 2. Springer.

Zunjani, F. H., & Olteteanu, A.-M. (2019). Towards reframing codenames for computational modelling and creativity support using associative creativity principles. In *Proceedings*

of the 2019 on Creativity and Cognition, p. 407–413, New York, NY, USA. Association for Computing Machinery.