The Thirty-Ninth AAAI Conference on Artificial Intelligence (AAAI-25)

# Improving Cooperation in Language Games with Bayesian Inference and the Cognitive Hierarchy

**Joseph Bills, Christopher Archibald, Diego Blaylock**
Computer Science Department, Brigham Young University, Provo, UT, USA
jotbills@byu.edu, archibald@cs.byu.edu, diegonb@byu.edu

## Abstract

In two-player cooperative games, agents can play together effectively when they have accurate assumptions about how their teammate will behave, but may perform poorly when these assumptions are inaccurate. In language games, failure may be due to disagreement in the understanding of either the semantics or pragmatics of an utterance. We model coarse uncertainty in semantics using a prior distribution of language models and uncertainty in pragmatics using the cognitive hierarchy, combining the two aspects into a single prior distribution over possible partner types. Fine-grained uncertainty in semantics is modeled using noise that is added to the embeddings of words in the language. To handle all forms of uncertainty we construct agents that learn the behavior of their partner using Bayesian inference and use this information to maximize the expected value of a heuristic function. We test this approach by constructing Bayesian agents for the game of Codenames, and show that they perform better in experiments where semantics is uncertain.

## 1 Introduction

Many real-world problems require cooperation with AI or human agents for success. Cooperative games provide an important setting for the development and evaluation of cooperative AI techniques. Agents generally utilize some form of communication to cooperate. Some settings define the semantics of the communication through actions which exist only within the game. However, agents can differ in their use of these actions in practice to communicate information. One challenge faced by some reinforcement learning approaches to these problems is that agents develop their own way of using actions to communicate during training. When these agents interact with other agents who do not use the actions in the same way, performance can suffer greatly.

In this paper, we focus on a specific type of cooperative game: *cooperative language games*. These games require agents to communicate using natural language. One might expect this solves the difficulty agents have communicating, but this is not necessarily the case. Because AI agents might use different language models to represent relationships between words, they could use language differently in the game. Even when using the same language model, their policy may respond differently to an utterance, especially if the meaning of an utterance is ambiguous due to restrictions on the amount of information which can be communicated. This illustrates the distinction between the semantic and pragmatic understandings of an utterance, and differences in either understanding can impede communication between agents.

Our goal is to design an agent for a cooperative language game able to cooperate successfully with any teammate agent, regardless of the semantics of the teammate's language model or the pragmatics of their use of language. We approach this using a Bayesian framework, where instead of being restricted to a single language model, our agent is given a set of possible language models over which it can reason probabilistically. Bayes' rule is used to update the agent's beliefs as it observes its teammate acting. This is, to our knowledge, the first example of an agent having a probability distribution over a set of language models, updating this distribution given observations, and adapting in this manner to any individual teammate.

We explore this idea in the cooperative language game of Codenames. This game, described in Section 2.1, involves using single words as clues to convey information about the true state of the game. The linguistic concepts of *semantics* and *pragmatics* have specific interpretations within the Codenames AI framework. In Codenames, semantics corresponds to the language model an agent uses to determine the relationship of words in the game, while pragmatics refers to the strategy an agent uses to determine which action to take, based on their understanding of semantics, their partner, and any signals received. An example of a difference in semantics would be using word2vec (Mikolov et al. 2013b) instead of GloVe (Pennington, Socher, and Manning 2014) as a word embedding. A difference in pragmatics might be guessing as many words as the clue gave in the order of increasing distance versus stopping once there are no more words within a set distance of the clue. Most previous Codenames agent models assume a common understanding of pragmatics where a guesser is expected to guess words based on similarity, so their behavior is parameterized only by differences in semantics, but previous work shows alternative pragmatics can improve performance even with teammates using the same language model (Bills and Archibald 2023).

Our proposed Bayesian framework will model and reason

---

about uncertainty in both semantics and pragmatics within Codenames. We will demonstrate that the resulting Bayesian agents have improved performance with a variety of other agents in Codenames. The remainder of the paper will provide required background and details of the game of Codenames. The Bayesian approach will then be described, followed by the details of our experiments evaluating it.

## 2 Background

Much previous research has focused on the task of agents who can cooperate with arbitrary teammates, which has been called *ad hoc teamwork* (Mirsky et al. 2022; Stone et al. 2010). The different approaches proposed include some from a reinforcement learning perspective where the goal is to determine a representative set of potential teammates strategies that can be used during training (Canaan et al. 2019; Rahman, Cui, and Stone 2024), while others instead focus on modifying behavior during play with a specific teammate (Bard et al. 2013; Pedersen and Crandall 2023). An example of a game that has been widely used as an ad hoc domain is Hanabi. Hanabi is a cooperative card game where players communicate using game-defined actions to gain information about their own cards, so that they can successfully play their cards and benefit the group (Bard et al. 2020; Canaan et al. 2019, 2023; Foerster et al. 2019). This is an example where the semantics of the communication are defined by the rules of the game, and so agents can be assumed to have the same understanding of them. The major uncertainty with an unknown teammate comes in the pragmatics of how that agent will use the actions. In contrast, Codenames involves natural language used outside the game. This means that agents might differ in both their semantic interpretation and representation of words as well as in the pragmatics of how they use the words in the game.

Coming to a common understanding of pragmatics represents a specific aspect of ad hoc teamwork. (Barrett et al. 2014) introduces a specific problem where agents communicate with defined semantics where an optimal solution can be computed. (Mirsky et al. 2020) develops a method for leveraging a commonly understood communication protocol for improving teamwork. (Macke, Mirsky, and Stone 2021) introduces an environment called CAT where communication is costly, motivating complex ad hoc pragmatic reasoning. (Cope and McBurney 2024) uses an environment where a communication protocol must be learned while playing.

Bayesian statistics provides a framework for reasoning about uncertainty (Bolstad and Curran 2016), which has been widely applied to multi-agent settings enabling AI agents to reason about teammates and opponents. As some examples, Bayesian reasoning has been used within the game of Hanabi (Foerster et al. 2019; Canaan et al. 2023), $n$-player competitive games (Sturtevant, Zinkevich, and Bowling 2006) and auctions (Baarslag et al. 2013), and to determine which of an agent's strategies will work best against the current opponent in poker (Bard et al. 2013). In each case the set of objects over which probabilistic beliefs are held, the way those beliefs are updated upon observations of events in the game, and how the current beliefs are utilized to determine the best action must be designed specifically for that domain. We explore the application of similar ideas to Codenames, which to our knowledge involves the first case of having a probability distribution over a set of language models in a cooperative language game.

In addition to utilizing Bayesian reasoning to reason about pragmatics we design our proposed agents to fit within a *hierarchy*. Our model is similar to the cognitive hierarchy introduced by Stahl (Stahl 1993; Stahl and Wilson 1995). Our model differs in that best-responses are only approximate and that in addition to a distribution of lower level agents, the belief model also includes alternate language models. The cognitive hierarchy itself is an extension of $k$-level reasoning, which has been applied to cooperative games including Hanabi (Cui et al. 2021), and the Keynesian beauty contest (Shapiro, Shi, and Zillante 2014). In the context of ad-hoc cooperaton, (Hu et al. 2021) provides a cognitive-hierarchy-based learning algorithm that is guaranteed to converge to a unique solution, making it ideal for zero-shot coordination.

Semantics and pragmatics are two foundational linguistic concepts. Semantics refers to the literal meaning of words and phrases and how the meanings of different words are related, while pragmatics refers to how context and the goals of speech acts change the meaning of utterances (Grice 1957) Most computational models of semantics are based on the distribution hypothesis – the meaning of words can be inferred from the meaning of other words used in the same context (Harris 1954). Note that while context is used to train these models, many of these models do not model context itself and thus only model semantics. A popular form of semantic model is the vector space *word embeddings*, which maps words to points in a normed vector space. In addition to capturing the similarity between words through similarity metrics in the vector space, the vector space can also capture semantic relationships between words through vector operations (Mikolov et al. 2013a). All language models we utilize in this work are vector-space word embeddings. (Lee et al. 2021) used noise to perturb word embeddings and change words, showing the potential of using random noise to simulate slightly different language models. The Rational Speech Act framework (Goodman and Frank 2016) models pragmatics as a Bayesian game (Harsanyi 1967), and this is the understanding of pragmatics that guides our model.

### 2.1 Codenames

Codenames is a board game which explicitly involves language and concepts of similarity between words (Chyátvil 2015). It has 25 cards with a single word on them laid out on the board. Each board card belongs to a hidden category. The possible categories are red, blue, bystander, and assassin. Two teams (red and blue) of at least two players each, compete to be the first to identify all of their team's board cards. The players take one of two roles on each team: *spymaster* and *guesser*. The spymaster is aware of the hidden category of each word on the board, but only the guesser can pick a board card and reveal its category assignment. Each turn, the spymaster gives a clue to the guesser, consisting of a single word and a number. The clue word is related in meaning to some of the cards on the board, while the clue number is generally the number of board cards the spymaster intends to connect to the clue word. The guesser must guess at least one card, revealing its hidden category, and guesses cards until either 1) a card is guessed that doesn't belong to their team, 2) they have guessed one card more than the clue number, or 3) they choose to end their turn. A team wins when all of their cards have been revealed, ending the game. A team loses instantly if they guess the assassin card.

A single-team variant of Codenames, where the goal is to guess all the team's cards in as few turns as possible, was used in a Codenames AI competition, and has been used in subsequent AI research. on Codenames (Summerville et al. 2019; Kim et al. 2019). The best AI Codenames agents to date have used *word embeddings*. While alternatives have been explored, such as large language models (Costarelli et al. 2024), they have not yet been as effective as the embedding approaches. Embedding approaches have a modular design, making it easy to separate and interchange the semantic and pragmatic elements of an agent. An agent's word embedding defines the relationships between words, and basic guessers essentially guess board cards in the order of distance from the clue word, according to their word embedding. Spymasters give the clue that maximizes the number of board cards correctly identified by a basic guesser using the same word embedding. These basic strategies result in agents that play very well together when using the same word embedding, but performance is typically very poor when embeddings differ (Kim et al. 2019). Subsequent work generally expanded the explored set of language models and began evaluating with a small set of humans (Jaramillo et al. 2020). One agent explored in that work used a näive Bayes filter to classify words, but it did not use Bayesian inference to adapt to the behavior of its teammate. Another focused on improving clues for play with humans, but didn't use the actual game of Codenames (Koyyalagunta et al. 2021). All this previous work focuses on identifying the single language model and/or strategy that will work best with a given population of teammate agents, oftentimes humans. However, all of these proposed agents are static, as they do not adjust or adapt their behavior based on the interaction with their current teammate. The baseline agents used in our experiments are derived from these works.

Conceptually, these agents have an internal model of their partner, and to play well this model needs to be accurate. Another paper explored new strategies or pragmatics for an abstraction of Codenames, using a deductive hierarchy where pairs of agents higher in the hierarchy perform better than those lower in the hierarchy. On the other hand, agents who are not adjacent in the hierarchy perform poorly with each other due to their inaccurate beliefs about the other (Bills and Archibald 2023). A major remaining challenge from all prior Codenames AI research is the creation of agents that can play well despite uncertainty about their partner. (Archibald and Blaylock 2024) learned noise levels to best adapt to their partner, but did not adapt the underlying model of their partner. (Sidji and Stephenson 2024) investigated using LLMs instead of word embeddings to play in, but found that while they were pragmatically different, they performed worse. (White, Pandey, and Pan 2024) looked at using cultural priors to play Codenames Duet to enchance pragmatic reasoning.

# 3 Foundations

In this section we introduce the foundational concepts and a general overview of our proposed Bayesian approach for cooperative language agents. In particular we will detail how uncertainty regarding both semantics and pragmatics will be represented within the framework. Codenames is used throughout as a concrete example of a cooperative language game, but the same concepts should be applicable to similar settings, albeit with some necessary adaptation.

## 3.1 The Deductive Hierarchy

The deductive hierarchy is an organization of agents for playing Codenames initially proposed in (Bills and Archibald 2023). The foundation of the hierarchy (level 0) is a *static guesser*, or a guesser that only considers the current clue and unrevealed board cards in determining its guess. The hierarchy then consists of alternating spymasters and guessers, designated so that a level $k$ spymaster ($S^k$) approximates a best response to a level $k$ guesser ($G^k$), while a level $k$+1 guesser ($G^{k+1}$) similarly approximates a best response to $S^k$. Roughly speaking, $S^k$ assumes it is playing with $G^k$, simulating $G^k$'s response to any clue it could give, and choosing the clue that results in the maximum revealed team cards. $G^k$ assumes that the clues it receives are generated by $S^{k-1}$. $G^k$ maintains beliefs over all possible states of the world (board card assignments), removing those that are inconsistent with given clues and revealed information. Board card identities are deduced when they are true in every remaining possible state. When $G^k$ has deduced that a board card is not on its team, it will skip over that card when guessing, and when it deduces a card is on its team it will use the extra guess to guess it. This behavior allows hierarchical agents on the same level to gain the most information possible from each clue and win the game in fewer turns. From the perspective of this hierarchy, the basic Codenames AI agent framework initially described in (Kim et al. 2019) consists of a level 0 guesser and a level 0 spymaster. Agents from higher levels in the hierarchy are dynamic, as the clues and guesses they produce will depend upon the entire history of the game to that point. The deductive hierarchy is fragile and hierarchical agents perform poorly when assumptions about teammates are incorrect. Our proposed Bayesian framework can be viewed as an extension of the hierarchy to reason probabilistically and be more robust with all teammates.

## 3.2 Modeling Uncertainty: Pragmatics and Semantics

One of the core ideas of any Bayesian framework is to explicitly model and account for sources of uncertainty. The main sources of uncertainty in cooperative language games like Codenames are semantics and pragmatics. We represent uncertain semantics by a probability distribution over a set of different word embeddings. Uncertain pragmatics can be represented by a probability distribution over different levels of the deductive hierarchy. Both of these sources of uncertainty can be captured by having a set of possible teammates, each with a word embedding and level in the deductive hierarchy. In order to account for the possibility of partnering with an unknown agent, we also add a noise model to the word embeddings. This means that any clue or guess has some non-zero probability of being generated by any agent model. Based on prior work involving perturbed word embeddings (Lee et al. 2021), we use Gaussian noise to simulate uncertainty in the implied communication channel.

### 3.3 Bayesian Approach Overview

We now provide an overview of the proposed Bayesian approach for Codenames agents. Each agent will have a set of possible teammate models $M$, and each $m \in M$ should differ by the word embedding it is using (semantics) or its position in the hierarchy (pragmatics). Beliefs over these teammate models will be maintained in the form of a probability distribution $P(m)$. When a teammate action $a$ is observed, the agent will update its beliefs using Bayes rule as $P(m \mid a) \propto P(a \mid m)P(m)$, where the posterior $P(m \mid a)$ will be used as the prior $P(m)$ for the next update.

The distribution $P(a \mid m)$ corresponds to how teammate model $m$ acts in the game, as determined by its strategy and word embedding. The Bayesian approach will fail if $P(a \mid m) = 0$ for all teammate models when action $a$ is observed since it will result in the posterior $P(m)$ being set to zero for all models, preventing any future inference. To avoid this problem and ensure all models may continue to be used for inference despite faults in the approximation, each teammate model should have a non-zero probability of generating any action. All previous Codenames agents of which we are aware have been deterministic: for the same game state, they would generate the same guess or clue. Thus, previous agent designs must be modified to be stochastic before they can be used effectively as teammate models in our proposed Bayesian approach. This can be done by adding a random perturbation to the embeddings of words, and details of this process will be provided in later sections.

### 3.4 Heuristic Utility Function

When making a decision in the game, Bayesian agents will use their beliefs to select the action that maximizes expected utility. The Bayesian agents will utilize the following heuristic utility function which will provide a utility for a sequence of cards to be guessed on one turn, based on the identity of revealed cards. The spymaster, with knowledge of the true world state, can use the actual card identities, while the guesser will instead use a possible world state. The utility of a turn is the sum of the values of any cards revealed that turn, minus 1. The value for the card types are as follows, assuming the agents are on the blue team: $u(\text{blue}) = 1$, $u(\text{red}) = -1$, $u(\text{bystander}) = 0$, $u(\text{assassin}) = -|B|$, where $|B|$ is the total number of blue team cards. This heuristic calculates the marginal contribution to the score at the end of the game from the current turn assuming a particular variant of the solitaire Codenames where an opponent card must be revealed each turn. Further explanation can be found in the full version of the paper (Bills, Archibald, and Blaylock 2024). The heuristic utility function could easily be replaced by another, where motivated.

We now describe the Bayesian spymaster, followed by the Bayesian guesser. Due to space restrictions, the full details of each agent will not be given here, but details are included in the full version of the paper. Experimental evaluation of the agents will be given in Section 6.

## 4 The Bayesian Spymaster

The Bayesian spymaster aims to deduce which guesser it is playing with, so that it can give more effective clues. The Bayesian spymaster maintains beliefs over a set of guesser models $M$, represented by a probability distribution that is updated after each turn and thus incorporates all information obtained from previous guesses. Initially, these beliefs are set to be the uniform distribution over $M$. Given observation of a guess $g$, the update is $P(m \mid g) \propto P(g \mid m)P(m)$. The conditional probability $P(g \mid m)$ cannot be computed exactly for arbitrary guessers, so it is estimated with a multinomial distribution, using Monte Carlo sampling to count guess occurrences for each model with Laplace smoothing, initializing all the counts to 1. Each guesser model is sampled multiple times, where each time noise is added to the guesser's embedding of the clue word. Distances to board words are computed from the perturbed embedding and the guesser then guesses according to its strategy. Since all guesses are included in the support for all guessers, a Bayesian spymaster is never completely certain which guesser it is paired with.

To choose a clue, the Bayesian spymaster computes the expected utility of each potential clue with each model guesser. A similar sampling process is used where each model guesser is queried multiple times, and its response generated using the perturbed embedding of the clue word. The total expected utility of a clue is then calculated for each guesser across these samples and across all models using the current beliefs. Clue words are omitted from consideration if, without noise, each guesser model in $M$ would guess an incorrect card, given that clue. While this optimization sacrifices theoretical optimality, it greatly increases the computational speed of the agent and performs well in practice. Further details are in the full version of the paper. A Bayesian spymaster that does not add noise to guesser embeddings and has only a single level-$k$ guesser model is equivalent to a level-$k$ spymaster from (Bills and Archibald 2023).

## 5 The Bayesian Guesser

The Bayesian guesser will have a set of spymaster models, $M$, over which it will maintain beliefs. It also maintains a history of previous clues and has two threshold parameters. After receiving a clue, the guesser will first sample possible world states consistent with the current observed state of the board using the method described in (Bills and Archibald 2023). For each clue $l_t$ in the history, it will calculate the likelihood the clue was given for each combination of spymaster $m$ and world state $w$ in the sample. This likelihood is given exactly by $\int_{\nu(l_t)} \mathcal{N}(m_t(w), \sigma, x)dx$ where $\nu(l_t)$ is the Voronoi region centered around $l_t$, $m_t(w)$ is the clue model $m \in M$ would give at turn $t$ assuming $w$ was the true state of the world, and $\mathcal{N}(m_t(w), \sigma, x)$ is the multivari-

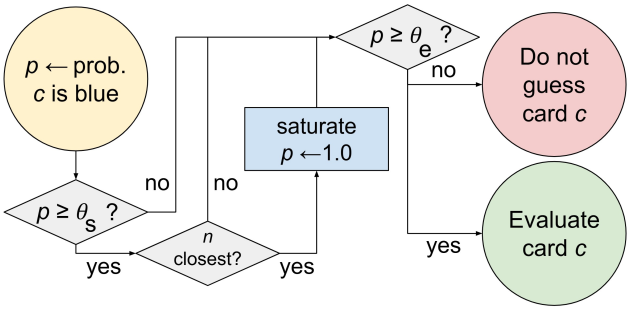

Figure 1: Bayesian Guesser evaluation of board card $c$

ate symmetric Gaussian centered at $m_t(w)$ with covariance $\sigma$, where $x$ is an arbitrary point in the embedding space. We use the Gaussian distribution since it is used for our experiments, but it could be replaced by another distribution. The entire history's likelihood is calculated as the product of the likelihoods for each turn's clue. The likelihoods of the clue history for each model and world state can be separately marginalized out. If there is a nonzero likelihood of the current clue occurring then the beliefs about model probabilities are updated according to the posterior distribution. Otherwise this step is skipped, mirroring the behavior of the level-$k$ guesser (Bills and Archibald 2023). If beliefs are not updated then the clue is not added to the history.

Next, the posterior beliefs for each world state are calculated assuming they were equally likely a priori. The probability of each unknown card on the board belonging to any category is then estimated using a näive Bayesian filter. To decide on a guess, the Bayesian guesser evaluates each board card $c$ using the process shown in Figure 1, which begins with the posterior probability $p$ that card belongs to the guesser's team. A *saturation threshold* $\theta_s$ is first used to adjust $p$. If $p > \theta_s$ and $c$ is one of the $n$ closest board cards to the clue word, then $p$ is *saturated*, or increased to 1.0, where $n$ is the numerical component of the clue and distance is measured according to the word embedding of the model spymaster with the highest posterior probability.

A *evaluation threshold* $\theta_e$ is used next. Only board cards with $p > \theta_e$ are evaluated using the expected value of heuristic function described in Section 3.4 over all possible assignments. The card with the highest evaluation is guessed. If no cards have a positive evaluation, then the card with the highest evaluation among all cards is selected, since the rules require at least one card to be guessed. The entire process is repeated for subsequent guesses, taking into account the information revealed from previous successful guesses. The guesser stops when all possible guesses have been used or when there are no cards with a positive evaluation, in which case the guesser ends its turn. Finally, if the guesser ever observes a card assignment which it believed had a zero probability of occurring then it will reset, again mirroring the behavior of the level-$k$ guesser.

### 5.1 Saturation and Evaluation Thresholds

To ensure the Bayesian Guesser is a strict generalization of the level-$k$ guesser, it is parameterized using aforementioned thresholds $\theta_s$ and $\theta_e$. In the deductive hierarchy, clue proximity signals which cards are on a player's team, but clues which have been deduced to not be on the player's team are skipped when updating beliefs using proximity. This behavior is not directly modeled in the Bayesian ideal and is missing in some edges cases, so it is explicitly modeled in the Bayesian Guesser by having the probability of close cards be saturated. As such, the saturation threshold $\theta_s$ describes the highest posterior probability for which nearby cards are considered to not have been signaled because they are likely not on the player's team. The evaluation threshold describes how confident the guesser must be that the card is on the team before it could be guessed. These parameters are designed so that when $\theta_s = 1$ and $\theta_e = 0$ the model behaves uses the ideal Bayesian reasoning described, but as $\theta_s \to 0$ and $\theta_e \to 1$ it relies less on the heuristic and Bayesian reasoning and more on an approximation of logical deduction. When $\theta_s = 0$, $\theta_e = 1$, no noise is used, and the set of modeled spymasters is a singleton of the level-$k - 1$ spymaster, then it behaves identically to the level-$k$ guesser. Details for how these parameters are integrated into the model are included in the full version of the paper.

### 5.2 Cognitive Hierarchy

A cognitive hierarchy can be constructed by assuming Bayesian agents have prior distributions over other Bayesian agents. To define this behavior, whenever a reference is made to the word embedding of a Bayesian agent, the word embedding of its highest posterior probability agent becomes the assumed word embedding, with a defined ordering of agents to break ties. This implies the word embedding of a level $k$ agent is the same as the level 0 guesser it is founded on. This allows arbitrary cognitive hierarchy models to be defined using Bayesian agents.

## 6 Experimental Evaluation in Codenames

We now describe the details of the experiments used to evaluate the described Bayesian Codenames agents at level $k = 1$ in the hierarchy.

### 6.1 Varied Semantics: Word Embeddings

The word embeddings used in the experiments include:

- Word2Vec (w2v) – trained using a word context windows (Mikolov et al. 2013b).
- Dict2Vec (d2v) – similar to w2v but trained on cleaned dictionary entries with an improvement on semantic similarity tasks (Tissier, Gravier, and Habrard 2017).
- FastText (ftxt) – Uses bags of character $n$-grams with weighting by position (Mikolov et al. 2018).
- GloVe (g1, g3) – trained on pre-computed statistical co-occurrence probabilities for words in a corpus (Pennington, Socher, and Manning 2014). We used embeddings with dimensions 100 and 300.

| Agent | Role | Noise | $\theta_s$ | $\theta_e$ |
|---|---|---|---|---|
| $S$ | Spymaster | None | None | None |
| $\widetilde{S}$ | Spymaster | 1.0 | None | None |
| $D$ | Guesser | None | 0 | 1 |
| $\widetilde{D}$ | Guesser | 1.0 | 0 | 1 |
| $B$ | Guesser | None | 1 | 0 |
| $\widetilde{B}$ | Guesser | 1.0 | 1 | 0 |
| $X$ | Guesser | None | 0.5 | 0.5 |
| $\widetilde{X}$ | Guesser | 1.0 | 0.5 | 0.5 |

Table 1: Bayesian agents for experiments

- ConceptNet Numberbatch (cnnb) – uses retrofitting to incorporate the ConceptNet Knowledge graph into an embedding. (Speer, Chin, and Havasi 2017).
- Word2Vec-Glove (wg) – concatenation of w2v and 50-dimensional GloVe that previous work found effective (Kim et al. 2019).
- ELMo (elmo) – a 1024-dimensional de-contextualized embedding derived from 3 layers of a trained contextual model (Peters et al. 2018). The context-free embedding was created by pooling contextual embeddings across many contexts (Bommasani, Davis, and Cardie 2020).

### 6.2 Experiment Setup

The Bayesian agents used in the experiments are detailed in Table 1. For each agent type, one doesn't utilize internal perturbation noise, while the other uses a noise value of 1.0. 6 different Bayesian guessers were used, where $D$ is used to indicate a *deductive* guesser, $B$ a purely *Bayesian* guesser, and $X$ a *mixed* guesser. All the Bayesian agents included the same set of word embeddings (w2v, g3, cnnb, and d2v) in the model set $M$. These are called the *internal* word embeddings. A set of static non-Bayesian agents using these word embeddings was included in each experiment as a baseline. In addition, the following word embeddings were also used in the experiments, although they were not included in the Bayesian agent model set: g1, ftxt, wg, and elmo. These will be referred to as *external* word embeddings.

To more efficiently calculate the set of possible clues, the 300 nearest neighbors of each word were precomputed. The probability that a perturbed vector would fall in the Voronoi region for any clue was precomputed using 1000 samples at each noise level. This was done by perturbing a model's word embedding using a normal distribution with mean at the embedding and then finding the closest word among the 500 closest neighbors. The Bayesian spymasters used 10 samples, and the Bayesian guessers used 1000 or 10,000 samples.

Each of the Bayesian spymasters – as well as the static spymasters for both internal and external word embeddings – were evaluated in environments both with and without the addition of stochastic embedding perturbations to all communicated clue words. These are called *stochastic* and *deterministic* environments respectively. Each spymaster played against all of the guessers. The guessers consisted of all the Bayesian guessers, as well as static guessers using both internal and external word embeddings, in both stochastic and deterministic environments. In stochastic environments noise was added to the clue embedding for the guesser in the Bayesian spymaster experiments, while for the Bayesian guesser experiments noise was added to the spymaster's word embedding and then transformed to the closest clue in the vocabulary before being passed to the guesser. Each pairing played 500 games. The results report the *win rate* for each pair, which is the fraction of solitaire games the pairing is able to successfully win.

### 6.3 Experimental Results

Table 2 shows the win-rate performance of the two Bayesian spymasters against different groups of guessers. In each case, we compare to the performance of the best static spymaster, using one of the internal word embeddings. The left half of Table 2 shows performance against guessers that are *in-distribution*, meaning that each guesser is using an internal language model, or one in the spymaster's set of models. The right half of the table shows the performance when the guessers are out-of-distribution – meaning they used external models that the Bayesian spymaster does not have. Table 3 shows the same type of results, but for the Bayesian guessers across different spymasters.

## 7 Discussion

In general, the Bayesian spymaster that assumed noise performed better than any other spymaster whenever they were partnered with out-of-distribution models or in stochastic environments. We believe this is because the spymaster rapidly learns which of its model guessers is the best fit for the guesser it is playing with, and then performs better than the corresponding spymaster for that word embedding because it chooses clues robust to uncertainty from noise or unmmodeled behavior.

Surprisingly, the Bayesian spymaster performed better with the guesser using d2v than the corresponding spymaster even in the deterministic environment, which is the condition the baseline spymaster was designed to be approximately optimal in. We believe that this is because the Bayesian spymaster can discriminate between different cards types not on the team, allowing it to avoid the assassin in cases where it is forced to give a bad clue, while the baseline spymaster cannot discriminate in this manner. The average column summarizes these results, showing the overall performance of each spymaster across all guessers. For both in-distribution and out-of-distribution guessers, in both deterministic and stochastic environments, the Bayesian spymasters have the best average performance. With out-of-distribution guessers, $\widetilde{S}$, the Bayesian spymaster with noise, performs the best by a wide margin. The improvement over the best model in these cases represents a huge improvement over previously published Codenames agents in the same cross-language model setting. In particular, it is noteworthy that the Bayesian framework allows the spymaster to do better than *any* of its constituent models would do on its own.

In contrast to the Bayesian spymasters, the Bayesian

|  |  | In-distribution guessers |  |  |  |  | Out-of distribution guessers |  |  |  |  |
|---|---|---|---|---|---|---|---|---|---|---|---|
|  |  | **w2v** | **g3** | **cn** | **d2v** | **Avg** | **g1** | **ftxt** | **wg** | **elmo** | **Avg** |
| **Det. Env.** | Best Model | 1.000 | 1.000 | 1.000 | 0.702 | **0.926** | 0.828 | 0.61 | 0.894 | 0.538 | **0.718** |
|  | $S$ | 1.000 | 1.000 | 1.000 | 0.858 | **0.960** | 0.840 | 0.656 | 0.956 | 0.592 | **0.761** |
|  | $\widetilde{S}$ | 0.830 | 0.994 | 0.954 | 0.650 | **0.860** | 0.974 | 0.802 | 0.990 | 0.720 | **0.872** |
| **Stoch. Env.** | Best Model | 0.364 | 0.568 | 0.184 | 0.364 | **0.370** | 0.592 | 0.208 | 0.606 | 0.464 | **0.468** |
|  | $S$ | 0.398 | 0.614 | 0.140 | 0.324 | **0.370** | 0.568 | 0.224 | 0.650 | 0.464 | **0.477** |
|  | $\widetilde{S}$ | 0.438 | 0.846 | 0.140 | 0.380 | **0.450** | 0.810 | 0.232 | 0.850 | 0.608 | **0.625** |

Table 2: Experimental Win Rate Results for Bayesian Spymasters

|  |  | In-distribution spymasters |  |  |  |  | Out-of distribution spymasters |  |  |  |  |
|---|---|---|---|---|---|---|---|---|---|---|---|
|  |  | **w2v** | **g3** | **cn** | **d2v** | **Avg** | **g1** | **ftxt** | **wg** | **elmo** | **Avg** |
| **Deterministic Env.** | Best Model | 1.000 | 1.000 | 1.000 | 0.702 | **0.926** | 0.752 | 0.788 | 0.702 | 0.580 | **0.706** |
|  | $D$ | 0.994 | 0.990 | 0.954 | 0.960 | **0.975** | 0.447 | 0.644 | 0.564 | 0.479 | **0.534** |
|  | $\widetilde{D}$ | 0.984 | 0.994 | 0.940 | 0.928 | **0.962** | 0.458 | 0.609 | 0.542 | 0.479 | **0.522** |
|  | $B$ | 0.788 | 0.794 | 0.794 | 0.756 | **0.783** | 0.356 | 0.471 | 0.432 | 0.374 | **0.408** |
|  | $\widetilde{B}$ | 0.792 | 0.786 | 0.826 | 0.758 | **0.791** | 0.423 | 0.518 | 0.419 | 0.352 | **0.428** |
|  | $X$ | 0.904 | 0.884 | 0.854 | 0.878 | **0.880** | 0.425 | 0.607 | 0.538 | 0.484 | **0.514** |
|  | $\widetilde{X}$ | 0.898 | 0.910 | 0.824 | 0.868 | **0.875** | 0.458 | 0.587 | 0.479 | 0.451 | **0.494** |
| **Stochastic Enviroment** | Best Model | 0.720 | 0.990 | 0.482 | 0.640 | **0.708** | 0.712 | 0.450 | 0.688 | 0.580 | **0.608** |
|  | $D$ | 0.712 | 0.972 | 0.262 | 0.734 | **0.670** | 0.397 | 0.300 | 0.514 | 0.479 | **0.423** |
|  | $\widetilde{D}$ | 0.704 | 0.976 | 0.508 | 0.742 | **0.733** | 0.417 | 0.283 | 0.521 | 0.477 | **0.425** |
|  | $B$ | 604 | 0.772 | 0.216 | 0.602 | **0.549** | 0.313 | 0.227 | 0.419 | 0.374 | **0.333** |
|  | $\widetilde{B}$ | 0.638 | 0.776 | 0.530 | 0.648 | **0.648** | 0.363 | 0.222 | 0.389 | 0.348 | **0.331** |
|  | $X$ | 0.696 | 0.872 | 0.252 | 0.682 | **0.626** | 0.389 | 0.296 | 0.499 | 0.484 | **0.417** |
|  | $\widetilde{X}$ | 0.770 | 0.894 | 0.556 | 0.738 | **0.740** | 0.406 | 0.274 | 0.497 | 0.451 | **0.407** |

Table 3: Experimental Win Rate Results for Bayesian Guessers

guessers almost never performed as well as the best model. We believe this is because the guesser inherently learns slower than the spymaster due to having a weaker signal. Because the clue signals both information about the state of the board and about the identity of the spymaster to the guesser, there is less information to specify the particular spymaster.

Despite their weaknesses, the Bayesian guessers often performed decently, and for both in-distribution cases one of the Bayesian guessers had the best average performance. This confirms that while they are not optimal, they are still robust as they were designed to be. $D$, the guesser that did not assume noise and had the lowest $\theta_s$ and highest $\theta_e$ did the best of all models on average in the deterministic environment. We believe this is because this model learns the fastest due to assuming a stronger signal from the lack of modeled noise, and from the fact that its deduction behavior discretely changes once a model raises to having the highest posterior distribution. Even though these deductions may be less accurate than the other models, the speed with which it acted proved more important in this particular game.

## 8 Conclusions

Bayesian inference and cognitive hierarchies can improve cooperation in language games. We demonstrated the first use of Bayesian reasoning to adapt to teammates in Codenames. The Bayesian spymaster was shown to be especially successful with out-of-distribution teammates, which has been a key difficulty with previous Codenames AI approaches. Effective implementation requires game-specific optimizations. While the theoretical description allows for arbitrarily large cognitive hierarchies, practical difficulties in implementing higher levels in the hierarchy made experimentation beyond the first level out of scope for this study. We hope to overcome that limitation in the future. Additionally, all experiments to date have involved only simulated language models. We would like to see how agents perform with human subjects, who display both complex semantic and pragmatic reasoning. We hope work with multiple semantic and pragmatic models in simulated environments will lead to creating agents that can better communicate with people.

# References

Archibald, C.; and Blaylock, D. 2024. Noisy Communication Modeling for Improved Cooperation in Codenames. In *2024 IEEE Conference on Games (CoG)*, 1–8. IEEE.

Baarslag, T.; Hendrikx, M.; Hindriks, K.; and Jonker, C. 2013. Predicting the performance of opponent models in automated negotiation. In *2013 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)*, volume 2, 59–66. IEEE.

Bard, N.; Foerster, J. N.; Chandar, S.; Burch, N.; Lanctot, M.; Song, H. F.; Parisotto, E.; Dumoulin, V.; Moitra, S.; Hughes, E.; et al. 2020. The hanabi challenge: A new frontier for ai research. *Artificial Intelligence*, 280: 103216.

Bard, N.; Johanson, M.; Burch, N.; and Bowling, M. 2013. Online implicit agent modelling. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, 255–262.

Barrett, S.; Agmon, N.; Hazon, N.; Kraus, S.; and Stone, P. 2014. Communicating with unknown teammates. In *ECAI 2014*, 45–50. IOS Press.

Bills, J.; and Archibald, C. 2023. A Deductive Agent Hierarchy: Strategic Reasoning in Codenames. In *2023 IEEE Conference on Games (CoG)*. IEEE.

Bills, J.; Archibald, C.; and Blaylock, D. 2024. Improving Cooperation in Language Games with Bayesian Inference and the Cognitive Hierarchy. arXiv:2412.12409.

Bolstad, W. M.; and Curran, J. M. 2016. *Introduction to Bayesian statistics*. John Wiley & Sons.

Bommasani, R.; Davis, K.; and Cardie, C. 2020. Interpreting Pretrained Contextualized Representations via Reductions to Static Embeddings. In Jurafsky, D.; Chai, J.; Schluter, N.; and Tetreault, J., eds., *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 4758–4781. Online: Association for Computational Linguistics.

Canaan, R.; Gao, X.; Togelius, J.; Nealen, A.; and Menzel, S. 2023. Generating and Adapting to Diverse Ad Hoc Partners in Hanabi. *IEEE Transactions on Games*, 15(2): 228–241.

Canaan, R.; Togelius, J.; Nealen, A.; and Menzel, S. 2019. Diverse Agents for Ad-Hoc Cooperation in Hanabi. In *2019 IEEE Conference on Games (CoG)*.

Chyátvil, V. 2015. Codenames.

Cope, D.; and McBurney, P. 2024. Learning Translations: Emergent Communication Pretraining for Cooperative Language Acquisition. *arXiv preprint arXiv:2402.16247*.

Costarelli, A.; Allen, M.; Hauksson, R.; Sodunke, G.; Hariharan, S.; Cheng, C.; Li, W.; and Yadav, A. 2024. GameBench: Evaluating Strategic Reasoning Abilities of LLM Agents. *arXiv preprint arXiv:2406.06613*.

Cui, B.; Hu, H.; Pineda, L.; and Foerster, J. 2021. K-level Reasoning for Zero-Shot Coordination in Hanabi. In Ranzato, M.; Beygelzimer, A.; Dauphin, Y.; Liang, P.; and Vaughan, J. W., eds., *Advances in Neural Information Processing Systems*, volume 34, 8215–8228. Curran Associates, Inc.

Foerster, J.; Song, F.; Hughes, E.; Burch, N.; Dunning, I.; Whiteson, S.; Botvinick, M.; and Bowling, M. 2019. Bayesian action decoder for deep multi-agent reinforcement learning. In *International Conference on Machine Learning*, 1942–1951. PMLR.

Goodman, N. D.; and Frank, M. C. 2016. Pragmatic Language Interpretation as Probabilistic Inference. *Trends in Cognitive Sciences*, 20(11): 818–829.

Grice, H. P. 1957. Meaning. *The philosophical review*, 66(3): 377–388.

Harris, Z. S. 1954. Distributional structure. *Word*, 10(2-3): 146–162.

Harsanyi, J. C. 1967. Games with incomplete information played by “Bayesian” players, I–III Part I. The basic model. *Management science*, 14(3): 159–182.

Hu, H.; Lerer, A.; Cui, B.; Pineda, L.; Brown, N.; and Foerster, J. 2021. Off-belief learning. In *International Conference on Machine Learning*, 4369–4379. PMLR.

Jaramillo, C.; Charity, M.; Canaan, R.; and 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*, volume 16, 231–237.

Kim, A.; Ruzmaykin, M.; Truong, A.; and 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*, volume 15, 160–166.

Koyyalagunta, D.; Sun, A.; Draelos, R. L.; and Rudin, C. 2021. Playing codenames with language graphs and word embeddings. *Journal of Artificial Intelligence Research*, 71: 319–346.

Lee, S.; Kang, M.; Lee, J.; and Hwang, S. J. 2021. Learning to perturb word embeddings for out-of-distribution QA. *arXiv preprint arXiv:2105.02692*.

Macke, W.; Mirsky, R.; and Stone, P. 2021. Expected value of communication for planning in ad hoc teamwork. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, 11290–11298.

Mikolov, T.; Chen, K.; Corrado, G.; and Dean, J. 2013a. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Mikolov, T.; Grave, E.; Bojanowski, P.; Puhrsch, C.; and Joulin, A. 2018. Advances in Pre-Training Distributed Word Representations. In *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*.

Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G. S.; and Dean, J. 2013b. Distributed Representations of Words and Phrases and their Compositionality. In Burges, C.; Bottou, L.; Welling, M.; Ghahramani, Z.; and Weinberger, K., eds., *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc.

Mirsky, R.; Carlucho, I.; Rahman, A.; Fosong, E.; Macke, W.; Sridharan, M.; Stone, P.; and Albrecht, S. 2022. A Survey of Ad Hoc Teamwork Research. In Baumeister, D.;

and Rothe, J., eds., *Multi-Agent Systems*, 275–93. Cham: Springer International Publishing.

Mirsky, R.; Macke, W.; Wang, A.; Yedidsion, H.; and Stone, P. 2020. A penny for your thoughts: The value of communication in ad hoc teamwork. International Joint Conference on Artificial Intelligence.

Pedersen, E.; and Crandall, J. 2023. AlegAATr the Bandit. In *ECAI 2023*, 1867–1874. IOS Press.

Pennington, J.; Socher, R.; and Manning, C. 2014. GloVe: Global Vectors for Word Representation. In Moschitti, A.; Pang, B.; and Daelemans, W., eds., *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 1532–1543. Doha, Qatar: Association for Computational Linguistics.

Peters, M. E.; Neumann, M.; Iyyer, M.; Gardner, M.; Clark, C.; Lee, K.; and Zettlemoyer, L. 2018. Deep contextualized word representations. *CoRR*, abs/1802.05365.

Rahman, A.; Cui, J.; and Stone, P. 2024. Minimum Coverage Sets for Training Robust Ad Hoc Teamwork Agents. In *AAAI*.

Shapiro, D.; Shi, X.; and Zillante, A. 2014. Level-k reasoning in a generalized beauty contest. *Games and Economic Behavior*, 86: 308–329.

Sidji, M.; and Stephenson, M. 2024. Prompt engineering ChatGPT for codenames. In *2024 IEEE Conference on Games (CoG)*, 1–4. IEEE.

Speer, R.; Chin, J.; and Havasi, C. 2017. Conceptnet 5.5: An open multilingual graph of general knowledge. In *Proceedings of the AAAI conference on artificial intelligence*, volume 31.

Stahl, D. O. 1993. Evolution of smartn players. *Games and Economic Behavior*, 5(4): 604–617.

Stahl, D. O.; and Wilson, P. W. 1995. On players' models of other players: Theory and experimental evidence. *Games and Economic Behavior*, 10(1): 218–254.

Stone, P.; Kaminka, G. A.; Kraus, S.; and Rosenschein, J. S. 2010. Ad hoc autonomous agent teams: Collaboration without pre-coordination. In *Twenty-Fourth AAAI Conference on Artificial Intelligence*.

Sturtevant, N.; Zinkevich, M.; and Bowling, M. 2006. Prob-max$^n$: Playing N-player games with opponent models. In *AAAI*, volume 6, 1057–1063.

Summerville, A.; Kim, A.; Ruzmaykin, M.; and Truong, A. 2019. The codenames AI competition.

Tissier, J.; Gravier, C.; and Habrard, A. 2017. Dict2vec : Learning Word Embeddings using Lexical Dictionaries. In Palmer, M.; Hwa, R.; and Riedel, S., eds., *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, 254–263. Copenhagen, Denmark: Association for Computational Linguistics.

White, I.; Pandey, S.; and Pan, M. 2024. Communicate to Play: Pragmatic Reasoning for Efficient Cross-Cultural Communication. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, 12201–12216.