Using intuitive behavior models to rapidly adapt to and work with human teammates in Hanabi

Arnav Mahajan

CMU-CS-22-119

May 2022

Computer Science Department
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Reid Simmons (Advisor), Pat Virtue

Submitted in partial fulfillment of the requirements for the Master of Science degree in Computer Science.

Supported by the Multidisciplinary University Research Initiative (MURI)

Keywords: Human-Computer Interaction, Artificial Intelligence, Behavioral Models

For my family and friends who supported me through everything.

Abstract

An agent that can rapidly and accurately model its teammate is a powerful tool in the field of Collaborative AI. Furthermore, if an approximation for this goal was possible in the field of Human-AI Collaboration, teams of people and machines could be more efficient and effective immediately after starting to work together. Using the cooperative card game Hanabi as a testbed, we developed the **Chief** agent, which models teammates using a pool of intuitive behavioral models. To achieve the goal of rapid learning, it uses Bayesian inference to quickly evaluate the different models relative to each other. To generate an accurate model, it uses historical data augmented by up-to-date knowledge and sampling methods to handle environmental noise and unknowns. We demonstrate that the **Chief**'s mechanisms for modeling and understanding the teammate show promise, but the overall performance still can use improvement to reliably outperform a solution which skips inferring a best strategy and assumes all strategies in the pool are equally likely for the teammate.

Acknowledgments

Thank you to Professor Reid Simmons for mentoring, guiding, and teaching me valuable research and life lessons. Thank you to Pallavi Koppol for being an additional source of amazing guidance. Finally, thank you to Gavin Zhu and Jeremy Chiu for their overwhelming commitment and hard work.

Contents

List of Figures

Chapter 1

Introduction

Motivation

The card game Hanabi has recently become a hot spot for AI research, both in AI self-play and in Human-AI collaboration. This is due to the environment of the game being partially observable, and inter-player communication being limited to game actions. The partial observability is a common challenge that AI research likes to solve, because it is so often a key obstacle in real world applications. In most tasks, agents would not have access to all relevant information, and so reasoning based on the beliefs they do have access to is a vital skill. Having communication limited to in-game actions is interesting, because it requires agents to somehow infer meaning from each others' actions. This is natural for humans, but very complicated for AI. In this thesis, I discuss the **Chief** agent: a collaborative agent that our team designed to rapidly model and accordingly react to human teammates. The agent was designed with Hanabi in mind, but the ideas can very easily be generalized to any Human-AI Collaborative problem.

Rules of Hanabi

Hanabi is a fully collaborative card game. Each player has 4-5 cards in their hand, which all other players can see other than them. The cards are marked by a color (5 different colors) and a number (1-5). When cards are played successfully, they're added to the board. When discarded or played unsuccessfully, they're put into the trash. For each round, the players take turns in a fixed order choosing from the available actions. The categories of actions are play, discard, hint number, and hint color. A card is playable if it is the next number on the board for its respective color. For example, if the board has Red 1 and Red 2 already played, then a player can play Red 3 successfully, but Red 4 and Red 5 being played would count as a miss. For hint number/color, the hint will take the form of "these are the cards in your hand that are the number ___" or "these are the cards in your hand that are the color ___".

The hinting player must say all the cards that match the hint (and not just only reveal the one they want to focus on). The group's score at the end of the game is the number of cards that were successfully played on the board, and, after three unsuccessful plays (misses), the game is over. It is important to note that there is a known and fixed quantity of each card number/color combo. For example, there are 3 1s for each color. Thus, players can reason about the likelihood of having certain cards in their hand by combining hints with that info. In the context of AI research, the players in the game don't have access to the extra out-of-game communication that human players do, so communication is done purely through the actions in the game. This presents an interesting challenge for collaborative AI research, where agents have to infer meaning from a very limited vocabulary of actions which impact the game state. The next section will go more into the interesting nature of Hanabi in the context of AI research.

Main Challenges

One interesting challenge that arises in Human-AI Collaboration in general, with this domain being no exception, is that the suboptimality of people is to the agent's detriment. This is because a teammate's error is harmful to the common goal and, even if the error is not that harmful, acting outside of the expected optimal behavior gives the agent more challenge in understanding its teammate's behavior. In other Human-AI game research for competitive environments, people making mistakes actually benefits the agent even if the agent doesn't expect it, so the agent can continue to assume optimal actions from its opponent regardless.

Another challenge we wanted to focus on was the goal of modeling human teammates quickly enough in the first game while still being accurate with that model. Presumably, our agent would always be dealing with a new, never-before-seen teammate. In a game like Hanabi, with maybe 20-30 moves total to observe from the teammate, it is impossible to achieve this from scratch. Therefore, there is the need for using as much prior domain knowledge as possible without over-generalizing over the spectrum of teammates we might face.

One final challenge we have to handle when modeling a teammate in a partially observable environment is knowing how to handle modeling someone with a different perspective. When the **Chief** is playing against a teammate, the teammate can see the **Chief**'s hand but **Chief** can't see that same information. Therefore, either our models or the **Chief** agent itself need to reason over the beliefs about what the teammate is seeing.

Overview of our Solution

When trying to handle the challenge of being able to rapidly and accurately model human teammates, we accessed our own intuition and common sense, as well as domain knowledge about the game. Our agent, the **Chief**, is powered by our own intuition and knowledge of the game. Using behavioral models to represent different intuitive strategies for approaching the game and inference based on observations of the teammate, the **Chief** is able to quickly evaluate the relative likelihood of each of the models in the pool being the one the teammate is using, and use that distribution as a quick model of the teammate.

To increase its accuracy of simulated models in a partial observable environment, the **Chief** uses sampling to handle any stochastic beliefs and propagates the most current knowledge into any computations done in the past to reduce the entropy in the beliefs used for those computations. The sampling and use of knowledge are solutions specifically for the challenge of reasoning about the teammate's strategy when the teammate is a different perspective than the **Chief**.

Our results show that the **Chief**'s mechanisms for modeling and understanding the teammate show promise, but the overall performance still can use improvement to be clearly better than more general solutions.

Chapter 2

Related Work

Related to this project, there is previous work domain-specific to Hanabi, as well as domain-general within the subject of Human-AI Cooperation. A major publication in the rise of Hanabi as an AI research topic was [1], which proposed the environment as a good field regarding the "beliefs and intentions" of external agents as well as a good field for testing collaboration with humans. The two main prongs of this proposed effort were ad-hoc cooperation and pre-coordinated self-play.

One hot topic in Hanabi is AI-AI cooperation (AKA self-play). Here, papers were focusing more on the partial-observability and restriction of inter-agent communication to actions. The best example would be the highest performing agent, created by [13], where they applied a multi-agent search technique on top of already developed policies that all the agents agree on. Facebook AI created this state of the art, which built on a chain of two previous papers, [9] which uses Bayesian updates to reason about public beliefs in a way related to human's theory of mind reasoning and [10] which improves on that by simplifying complexities related to the epsilon-greedy exploration method used during training. In our focus, we cannot assume any prior agreement, since our agent is meant to be cooperating with humans it has never seen before. However, there is a similarity when it comes to handling the partial-observability and reliance on actions as communication, which motivates our usage of their LSTM structure in our "human-like behavior clones" (see 4). [3]'s team used a genetic algorithm built on top of a set of rules to create an effective system for creating collaborative pre-trained agents in Hanabi. [17] is a paper that showed the benefits of using models of teammates in self-play Hanabi agents, an idea that will see much more usage in the ad-hoc area as well.

The other main focus in Hanabi AI research was the same one that our project tackles, which is human-AI cooperation in Hanabi, a branch of the ad-hoc cooperation prong of the Hanabi AI literature. [8] approached this problem by following the idea that agents can communicate their intentions through actions, and materializing the idea by creating a protocol based on

H.P. Grice's maxims of communication. Our work also works off of ideas about human behavior within restricted communication channels. In contrast, our strategy relaxes the restriction on models from following rules of human communication to just producing intuitive behavior. Additionally, we forked our code base off of theirs due to their interface for testing agents against other agents as well as with people (with some tweaks to the interface and game engine). Another approach to the human-AI cooperation problem was [11], which introduced the technique "Other-play" which generalizes learning algorithms over environmental symmetries (like colors in Hanabi, where red and blue are not inherently meaningful other than the fact that they're different categories). An agent created with this technique will be strongly resistant to learning non-intuitive strategies. We took inspiration from their goal of avoiding non-intuitive actions, but, instead of creating a learning algorithm that avoids generating non-intuitive strategies, we created one that builds off of definitionally intuitive strategies. [4] actually brings up very similar ideas of pools of models meant to model ad-hoc cooperators, but they focused more on the angle of generating a pool of models to evaluate the effectiveness of an approach as opposed to our usage of models in the approach itself.

Outside the domain of Hanabi, there are many angles from which Human-AI Collaboration is studied and thought about. [5] used human models as training partners for more traditional Deep Reinforcement-Learning approaches in a simplified version of the cooperative game Overcooked, and found that the methods that were using general human models as training partners performed much better in practice. An important takeaway from this was that learning with the suboptimality of humans as part of the training will transfer better to real cooperation. Our methods use intuitive strategy models to both model our teammates and respond to them, rather than using black-box learning with a training partner. Another difference is that we do not use data-driven models for the human side, because we want to define and follow explicit strategic conventions for each model. Focusing more on tackling the individual differences between human teammates, in [12], the researchers used imitation learning with less data on top of multi-agent reinforcement learning to effectively be able to handle different social conventions between people. The aspect of rapidly adapting to the conventions between people is similar to our goal of rapidly adapting to the set of strategies that we recognize in our teammate, but our approach uses fixed models and learning on top rather than updating pretrained models during execution. [14] tested the power of human-human theory in human-robot situations and showed that applying the human teaming practice of cross-training, where teammates switch positions while training, to human-robot collaboration actually produced effective results. Looking more at handling possible errors extremely specific to human-AI problems, [15] demonstrated a method of effectively calibrating trust levels in times when a human team-

mate may be over-trusting the autonomous system. The issue here is the safety concerns that arise from a human teammate taking the uncertainties of an autonomous agent for granted.

Another important premise in our work is agent modeling, which, as seen earlier, has been a common technique in many Hanabi-related papers. In an article talking about agent modeling [7], Crandall proposes Builders as an agent-archetype in addition to the usual Leaders and Followers, where Leaders find an outcome that they think the partner will like and try to enforce that outcome as a common goal and Followers model their partner and try to respond as well as possible given that model. A Builder would be an agent that is proposing common goals but continually evaluating if that common goal is agreeable for the partner. While Builders seem to be a powerful concept in Human-AI cooperation, we decided to push the traditional Follower archetype, due to the complexity of evaluating and proposing goals when the only form of communication is actions within the game.

A core part of our project is motivated by the idea that having a model of our human collaborators is a plus. This isn't a trivial conclusion, as modeling human teammates could just as easily over-complicate and slow down a cooperative agent. [6] compared Theory-of-Mind based learning algorithms which model the human to black box approaches and model-free approaches under these conditions in the context of Human-Robot Interaction. With limited data and small errors in the Theory-of-Mind assumptions, using the ToM models proved to be more effective than the black box model-based approaches. Understandably, model-free methods required much more data than a ToM model-based method with correct assumptions. These results motivated some design decisions of our algorithm, because our target situation is working with new teammates, which implies less interaction data, and partial observability, which implies possible errors in model assumptions. Thus, our algorithm is model-based and uses "human-like" models which are essentially just designed to play using intuitive strategies and reasoning, without trying to define strict Theory-of-Mind principles.

Chapter 3

Chief Agent

The codebase for all relevant components is available at https://github.com/ArnavM1499/Hanabi-HumanAI.

Motivation

The highest-level goal for the **Chief** agent is to produce an agent that can achieve high team scores when playing with a human teammate. This teammate will be someone that the agent has never played with before, so we break this goal down into two main sub-goals. We want the **Chief** 1. to be able to rapidly adapt to the teammate's strategy and 2. to be able to execute a strategy that works well with the teammate. It is important to note that the teammate, a human, will also be adapting to the **Chief** in the final intended environment for the **Chief**, and so certain assumptions made in the design may be inaccurate in practice. Specifically, we avoid some complexity in the design by assuming that the teammate's strategy will not change. In practice, future user studies and iteration on the design might require removing this assumption.

Figure 3.1 shows at a very high-level how the **Chief** is handling the complexities of the game environment while making use of the teammate's observed behavior and a pretrained pool of models.

Collaborating with a Person

In any environment, the **Chief** agent has the environment's observable state $\mathbf{G}$, hidden features of the state $\mathbf{H}$ and so the full state is $\mathbf{H} \times \mathbf{G} = \mathbf{S}$. We can denote the teammate as **Teammate**. Additionally, to infer the teammate's strategy and execute moves of our own, we give the **Chief** a pool of pretrained dual-purpose models that can be used as a model of the teammate

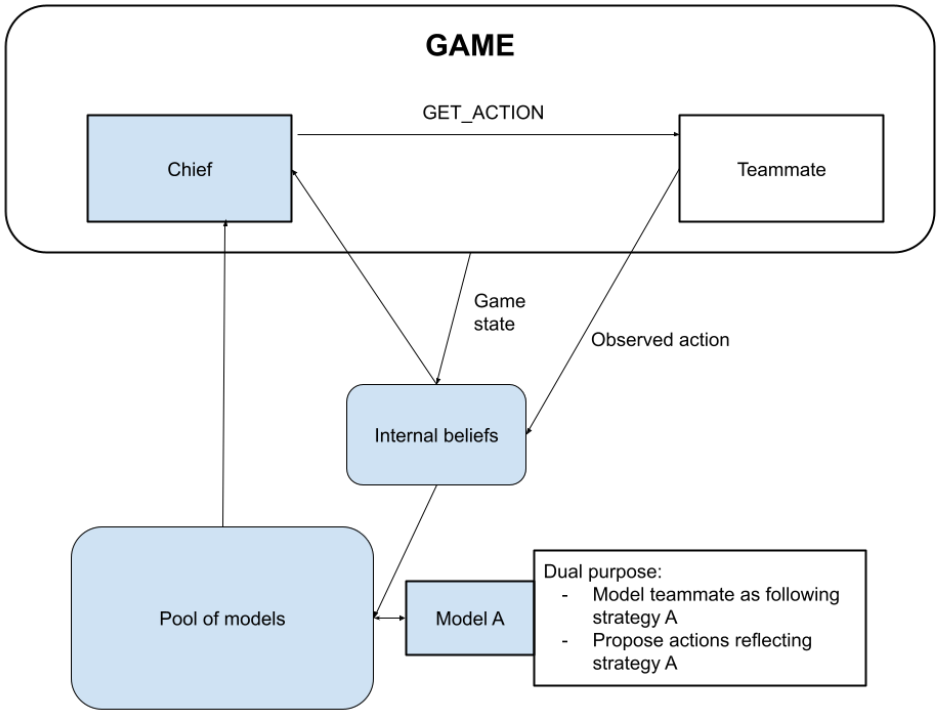


Figure 3.1: A brief overview of the **Chief** in the context of a game

and/or be used to execute actions reflecting a specific strategy. Each of these models is meant to represent a specific intuitive strategy in both purposes.

Chief has two high-level functions:

1. $UPDATE\_SYSTEM(\mathbf{S}, \mathbf{A}, \mathbf{S}')$ where $\mathbf{A}$ is an action observed from the teammate and $\mathbf{S}'$ is the resulting state [UPDATE_SYSTEM is *inform* in the code].
2. $GET\_ACTION(\mathbf{S})$.

Connecting back to the motivation listed above, $UPDATE\_SYSTEM$ handles the goal of rapidly adapting to the teammate's strategy, by inferring a strategy based on the observable behavior of the teammate. $UPDATE\_SYSTEM$ has a secondary purpose of handling the uncertainty in the game environment. This second purpose is useful for better inference of the teammate's strategy, since it makes assumptions about the teammate's knowledge more accurate. It is also helpful for GET_ACTION which handles the sub-goal "to be able to execute a strategy that works well with the teammate". In the current design, we assume that the best response to the teammate is to mirror their strategy. If this assumption was to change, it would be simple to update GET_ACTION accordingly by replacing the model proposing each action to be whatever the best response is instead of a mirrored strategy. The

reason we are working with the mirror assumption currently is that mirroring the teammate's strategy would be the most obvious way to generate our own strategy that the human teammate can easily understand and work with. A possible drawback of this assumption, however, is when we are playing against a very sub-optimal player and are trying to replicate this sub-optimal strategy rather than trying to bring the team score above what the player would normally be able to achieve.

Before explaining the internals of these two high-level functions, we can go over the general process. Here are the steps that the **Chief** will encounter on each turn:

1. **Chief** observes **Teammate** execute action $\mathbf{A}$ at state $\mathbf{S}$ and also records the resulting state $\mathbf{S}'$
2. **Chief** runs $UPDATE\_SYSTEM(\mathbf{S}, \mathbf{A}, \mathbf{S}')$ to use **Teammate**'s observed behavior to update its beliefs about **Teammate**'s strategy and to use the resulting state $\mathbf{S}' = \mathbf{H}' \times \mathbf{G}'$. In context of Hanabi, $\mathbf{H}'$ would be all cards in the **Chief**'s hand at that move and $\mathbf{G}'$ would be the observable game state after the move, like "what hints have been given", "what cards have been played", etc. If any update to $\mathbf{G}'$ is relevant to the current $\mathbf{H}'$ or any previous instance of $\mathbf{H}$ in the game, these updates are propagated back in time till the present to recompute its beliefs about the teammate's strategy.
3. After updating its beliefs, $\mathbf{S}$ gets set to $\mathbf{S}'$, and **Chief** runs $GET\_ACTION(\mathbf{S})$.
4. $GET\_ACTION(\mathbf{S})$ takes a weighted sum for each possible action over the pool models representing **Chief**'s perspective, with each weight being the model's confidence in the actions multiplied by the **Chief**'s confidence that the teammate is using the same model. If we were not assuming that mirroring the teammate's strategy was best, we would multiply the actions by the confidence this is the best strategy to respond to our teammate with.
5. **Chief** executes the action that $GET\_ACTION(\mathbf{S})$ returns.
6. Repeat from step 1

How $UPDATE\_SYSTEM(\mathbf{S}, \mathbf{A}, \mathbf{S}')$ works

Let $\mathbf{S}' = \mathbf{H}' \times \mathbf{G}'$ and assume that **Chief** stores $\mathbf{S_i}$ for $i \in 1, 2, ..., t$ for t time-steps so far, where $\mathbf{S_i}$ was the value for $\mathbf{S}$ when $UPDATE\_SYSTEM$ was called at time-step i. If any part of $\mathbf{G}'$ contains new knowledge about a feature of any $\mathbf{S_i}$'s corresponding $\mathbf{H_i}$, **Chief** will update the knowledge as described in Chapter 5. Once updating the knowledge, all previous computations for inferring the teammate's strategy will be redone with current knowledge. Some examples of this in the context of Hanabi would be receiving a hint about a

card in our hand that impacts our beliefs about the card's value or playing a card and realizing what value the card had at all times that we were holding it. Even if it isn't that useful to know the value of an already-played card, this knowledge would have made previous inferences about the teammate's behavior more accurate, and so we go back in time and re-sample and re-compute those beliefs. Additionally, **Chief** will use the combination of {$\mathbf{S_i}$ for $i \in 1, 2, ..., t$}, A, and any of the knowledge processed throughout execution about the various $\mathbf{H_i}$ to update its beliefs about **Teammate**'s strategy as described in Chapter 6.

How $GET\_ACTION(\mathbf{S})$ works

If **Chief** is currently at $\bar{\mathbf{S}}$, then it does the following process for each action A^j in the action space. Take each pool model and face it from the **Chief**'s perspective. Retrieve its confidence in executing A^j when processing state $\bar{\mathbf{S}}$. Multiply that value by the **Chief**'s confidence that its teammate is using this same model according to the model likelihood in Chapter 6. Each A^j has a sum of these weights over the models in our pool. Thus, the action with the largest sum is played. As mentioned before, if we changed the mirror assumption, we could update the weight factor of each model to reflect the new assumption while still using our beliefs about the teammate.

Chapter 4

Pool of Models

Note: This portion of the project was mainly implemented by undergraduate researchers Jeremy Chiu and Gavin Zhu, under the guidance of myself and Professor Reid Simmons.

Since the **Chief** agent is meant to quickly adapt to human teammates in complex and partially observable environments, having it model a human from scratch would be too slow. To give the **Chief** a chance of being able to rapidly model a person, we create a pool of possible models for the human teammate ahead of time and use inference rather than modeling to determine the best model from our pool for our teammate as quickly as possible. The goal here is to have pretrained models, each of which represents a different intuitive strategy, that are then selected from as the best model for the teammate. The reason for this choice of method was to avoid having to perform any sort of extra deep learning to adapt to the human teammate, since the game environment is too complex to perform meaningful and accurate deep learning updates from the first game with a person.

We first generate a list of intuitive conventions a human player might use (excluding any special non-intuitive techniques that experts might use). Some examples of these would be:

- Is the player counting cards when interpreting hints
- Does the player prefer to play/discard older or newer cards or just randomly choose
- The importance the player puts on recent hints
- How much risk is the player likely to take when playing/discarding
- How important is it to the player to protect cards that might be useful later on in the game
- How will the player prioritize the possible actions

Based on these intuitive conventions, we implement mostly deterministic agents that are parameterized by these. Therefore, each "intuitive strategy" is actually some selection of these conventions. In total, there were three structures implemented: ExperimentalPlayer, ValuePlayer, and HardcodePlayer. The first and third are based on logical reasoning and the second is based on creating values for the valid actions and choosing the maximizing one. The implementations for these players are in the “Agents” folder in the **Chief** codebase. The purpose of these agents were to use intuitive reasoning inspired by how a human player might reason about the game, and make decisions using the parameters that reflect the generated strategies. The agents all output one action.

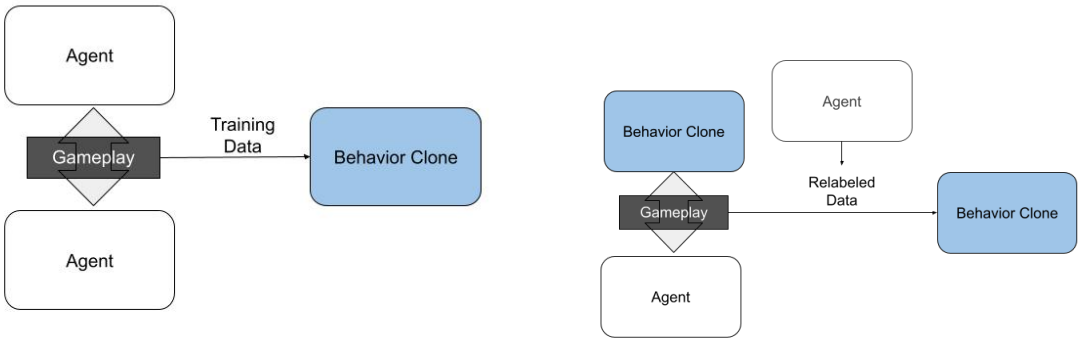
The purpose of the agent pool is to generate possible models in a Bayesian inference based system, which requires each model to have an associated probability for the observed action. For the agents that use logical reasoning, it is hard to infer a probability of them picking a specific action given a game state. Additionally, we empirically found difficulty inferring probability even with the value-based players.

To transform the agents into stochastic models that could give us these probabilities, we trained respective behavior clones. A good example of applying imitation learning with complex deep networks can be found in [2], where they were able to learning effective self-driving behaviors just using the human demonstrator’s steering angles as the label data. Therefore, using imitation learning with neural networks to capture intuitive decision-making in the complicated game environment of Hanabi is a supported approach.

Behavior Cloning the Pool’s Models

The desired goal of the behavior cloning process is to produce stochastic models that reflect the intuitive strategies they were cloned from, provide probabilities usable for Bayesian inference, and can be used to play in games themselves.

Figure 4.1 illustrates the overall process of generating behavior clones of the intuitive agents. In the first round of training, the behavior clone is getting labeled data based on games between agents. For example, a behavior clone of Agent A would be trained on a sequence of game states labeled with the respective action taken by Agent A in that moment. The idea of this process is open-ended, in the sense that Agent A could be playing with an agent based on a different strategy or with a mirror of itself. While we only



(a) First Round of Training a Behavior Clone on an Agent

(b) Second Round Using ideas from DAGGER Algorithm [16]

Figure 4.1: Creating Stochastic Behavior Clones of Deterministic Agents

trained the behavior clones on the mirrored case, relabeling data for games with other agents could be an effective way to make the behavior clones better generalized representations of the original intuitive strategies. Even if each strategy is designed to be played with another version of itself, its logical reasoning would still hold in games with other strategies, so giving the behavior clone that same capability would be a possible improvement for future work.

In the second round of training, the behavior clone is deployed in games with an agent (either the agent it's being cloned from or another). This is based on the ideas from the DAGGer algorithm [16]. One way to view this is that the environment it is being deployed in is a game with this specific agent. The second round is to help with avoiding rolling errors over time that push the behavior clone into more and more unknown territory.

Table 4.1 shows the accuracy of the Behavior Clones both while observing a game between agents (like in phase 1 of the cloning process) and playing with its source agent (like in phase 2). Unfortunately, there are not equivalent results for the behavior clones before DAGGer, but it is clear here that the overall process has created clones that, for the most part, maintain the quality of imitation while deployed that they had in the initial phase of training. Every accuracy value was computed by running 50 games with either the behavior clone or source agent playing with a teammate using the source agent and averaging the accuracy evenly over all moves total. Some of these values are slightly lower than what was recorded during the actual training of the network, but these give a good approximation of how accurate the clones are expected to be in context of the Chief. In terms of the actual accuracy values, it is important to note that these are discrete accuracies using only the maximizing action for the clone and not the probability of it choosing

Source agent id	Observing accuracy	Deployed accuracy
00001	0.778	0.772
00002	0.747	0.745
00003	0.612	0.591
00004	0.719	0.617
00005	0.706	0.673
10001	0.823	0.761
10002	0.806	0.772
10003	0.845	0.733
10004	0.849	0.791
10005	0.680	0.643

Table 4.1: Behavior Clone accuracy while observing agent and while choosing actions itself. This accuracy is the fraction of times that the clone's most likely action was the same one the deterministic source would have picked in the same situation

the same action as the agent, so some models may be penalized even if the correct action had a likelihood of 0.4.

To give some context for the meaning of the agent id strings that are being used to identify each strategy in the **Chief**'s pool, 00001-00003 are versions of the ExperimentalPlayer class with different settings, 00004-00005 are versions of the ValuePlayer class with different settings, and 10001-10005 are versions of the HardcodePlayer class with different settings. It's perhaps helpful to note that the parametrizations of each of these agents can be found at: https://github.com/ArnavM1499/Hanabi-HumanAI/blob/master/Agents/configs/players.json

Figure 4.2 shows some scores for match ups between the deterministic agents. This gives some insight into the pros and cons of our assumption that mirroring the teammate's strategy is best. It will likely always generally achieve good performance from the standards of the teammate's strategy, but it may not be the best way to respond to that specific strategy.

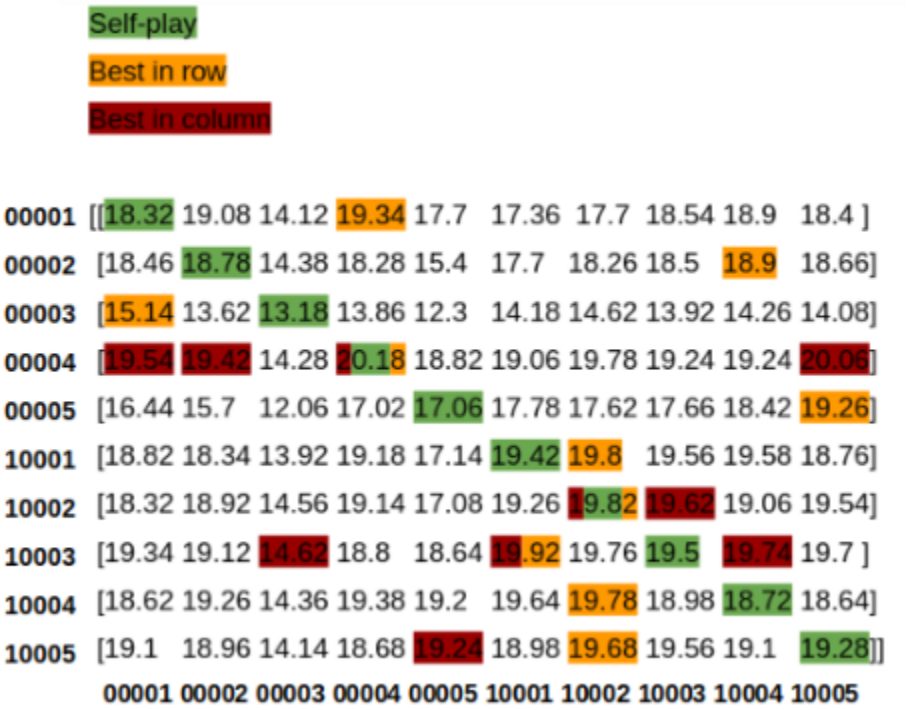


Figure 4.2: Scores of match ups between agents in each position. These are averaged over 50 games for each value.

Chapter 5

Environment Knowledge Management

In Figure 3.1, the abstract component labeled "internal beliefs" has the tall task of providing knowledge to the models in the pool and the **Chief**. To understand what types of knowledge are specifically needed, we can go back to the goals of the **Chief** agent. In order to rapidly adapt to the teammate's strategy, the **Chief** must use the observed actions of the teammate in combination with the observed game state to update its beliefs about the teammate's strategy. In order to execute strategies that work well with the teammate, the **Chief** must use these same two observation sets to choose actions that are likely to perform well in the team setting. Therefore, the "internal beliefs" component needs to handle the observations from the teammate and game state and propagate the relevant knowledge to the other components in the **Chief**. For this chapter, we will focus on how it handles and uses the observations from the game state.

For the unknown values in the game state, the main one in our Hanabi environment was the **Chief**'s own hand. However, this functionality would be easily generalizable to other partially-observable environments, and likely would work just as well. The motivation for the environment knowledge management's implementation is a desire for making the most of all knowledge we have access to.

In Hanabi, the **Chief** analyzes the game as a sequential process, which means that the historical observations are used alongside the current observation to make inferences. At every point in time, we will have access to certain knowledge about the partially observable state of the environment. However, at a time-step t', we might gain knowledge about a hidden feature that was relevant at time-step t. For example, take an example game where the teammate hinted that a card in my hand was a red card earlier in the game. Later on, I play the card and it is a miss, because the red card was a 1

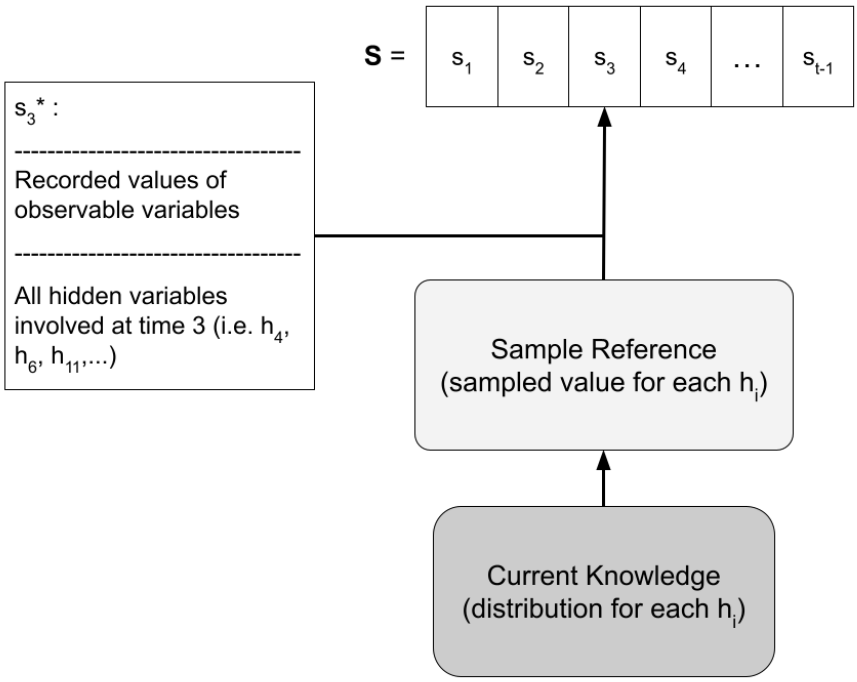


Figure 5.1: A step in the creation of one sample input for the LSTM behavior clone networks at time t

and the red card on the board was already a 3. Using these two moments, I look back in time and realize that my teammate will sometimes hint at cards that are useless with the color instead of the number, or make any other type of inference about that previous action. However, without the knowledge I just gained in the present about the card's value, inferring from that action would not be as informative. Events like this example happen constantly in Hanabi, and so we propagate this knowledge backwards to help us make these informative inferences more often.

If the **Chief** fills in the t-th part of a sequential input with only the knowledge available at time-step t, it isn't using easily accessible knowledge it has gained since t. To take advantage of this, we use **knowledge rollbacks**. Informally, knowledge rollbacks update a global set of knowledge with new findings and then use this global set to roll back the most up-to-date knowledge when generating sequential inputs. Formally, we can take any hidden feature h_k to represent the k-th variable we've encountered that we don't know the value of (i.e. a card in our own hand for Hanabi). Every time the **Chief** observes anything that can improve our knowledge of h_k (which means lowering the entropy of values that can be assigned to h_k), the knowledge model for h_k is updated in the global set. Finally, when generating any sequential in-

put of form $s_1, s_2, \ldots, s_T$, we can produce sampled values for each h_k observed so far (i.e. $[h_1 = H_1, h_2 = H_2, ..., h_n = H_n]$) and reference those samples for all existing h_k at each s_i. Figure 5.1 demonstrates this system generating one row of one sample sequential input, with "Current Knowledge" representing the global set and "Sample Reference" representing the sampled values generated for this input.

We don't have any theoretical justification for the accuracy of analysis being guaranteed to not decrease over time, since that would depend on the environment. In Hanabi, we do not have any added noise or uncertainty for a specific hidden feature as time goes on, so our accuracy will be non-decreasing.

With the above protocol for choosing what knowledge to use when handling hidden features, we can now use a simple sampling method for setting the feature values. Thus, **Chief** will use the knowledge for each h_k and assign a value from a weighted sample. This weighted sample will give us a value that we can then assume for h_k when doing the rest of the analysis on this sample. This allows our deep network models to act as if they have the level of full knowledge they were trained on so that they don't have to be robust to extra uncertainty. The uncertainty is handled by the sampling, and every sample sequential input is consistent between the steps in the sequence, meaning that the value for a card sampled for one time step will be the same value sampled at a different time step if the card was in our hand at either or both of those times.

Thus, with these steps in place, we have a sequential input with the most informed values for all hidden features at every time-step, and consistency of each hidden feature across different time-steps. This gives us effective robustness to environmental uncertainty by taking advantage of the sequential nature of the agent's experience in the environment and sampling at a high-level so that it does not need models that are trained under the same uncertainty. It is important to note that this process assumes that the domain is not Markovian. A justification for using this assumption is that we are designing for the human-AI space, and so we shape the domain around human reasoning. More specifically, a person playing Hanabi would likely use information from more states than the current one to make decisions. For example, if I got a hint two rounds ago, the meaning of my teammate giving that hint might still be important to me, even if I decided not to use it the previous round. This in itself shows that the environment cannot be Markovian if we want to fit our agent in settings with human players.

In the experiments used to generate the figures in this section, the **Chief** was run under different settings against different agents for 50 games each.

For each of these, the behavior clone of the agent that was the teammate for that game was monitored to see how likely it would be to choose the same action given some level of knowledge. When we say "the knowledge obtained by the last round", we are taking a snapshot of the **Chief**'s internal system at the end of the game, and looking at each model's last computed likelihood for the observed action. When we say "the knowledge obtained by that round", we are looking at a snapshot of the **Chief**'s internal system at the time it observed that action from the teammate, and taking the model likelihoods it had computed at that time. The values on the graphs are these likelihoods for the model trained on the teammate's deterministic source agent.

Figures 5.2, 5.3, and 5.4 aim to demonstrate the effects of knowledge rollback in the case of 50 games each with three different agents in the pool. The results for each line ends when at least 10 games have ended. The error shading is based on the variance of the results at that round. Surprisingly, none of the graphs show any strong support for the benefit of knowledge rollbacks.

Figures 5.5, 5.6, and 5.7 aim to demonstrate the impact of more or less samples taken when estimating the likelihood of the correct behavior clone model choosing the same action that the teammate did. Again, against expectations, the results do not show any strong support for the benefits of more samples other than the 1 samples results having most of the noise at the lower end of likelihoods.

For both of these techniques, they definitely have solid theoretical foundation, but in practice there doesn't seem to be strong improvements coming from applying either of them. One possible reason is that the different sampled values for the cards in our hand may have a small relative impact on the LSTM's overall processing of the full information in the game. The input includes the hints that have been given, the cards on the board, the cards in the trash, the most recent action, and the conditional knowledge for each hint the teammate could give, so potentially the perturbations from the sampled values don't impact the output as much as what intuitively would be expected. This reason would explain the lack of improvement from both techniques, since, in practice, they are both only used for those sampled values.

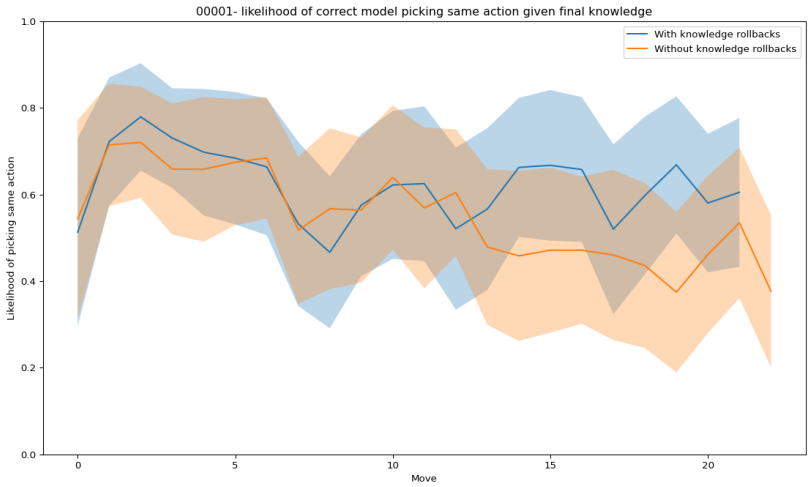


Figure 5.2: From 50 games with agent "00001", how did the knowledge rollback impact the likelihood of the behavior clone of "00001" in **Chief**'s pool picking the same action given the knowledge obtained by the last round. The shading is the variance at each time-step matched by color to the respective line

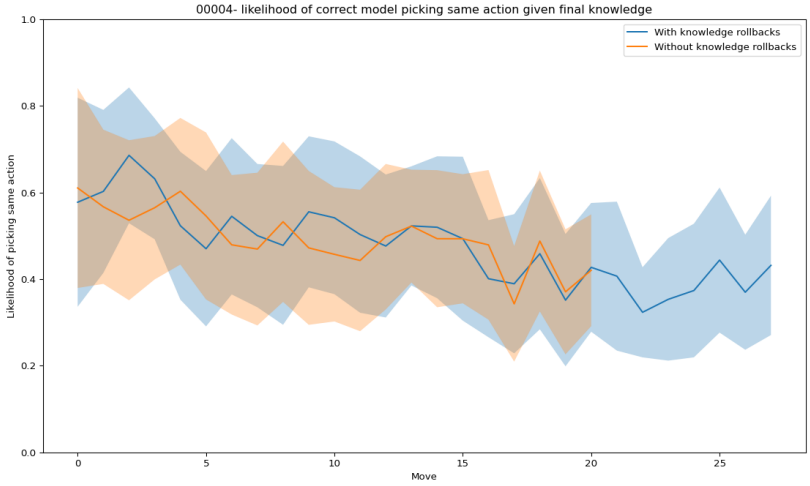


Figure 5.3: From 50 games with agent "00004", how did the knowledge rollback impact the likelihood of the behavior clone of "00004" in **Chief**'s pool picking the same action given the knowledge obtained by the last round. The shading is the variance at each time-step matched by color to the respective line

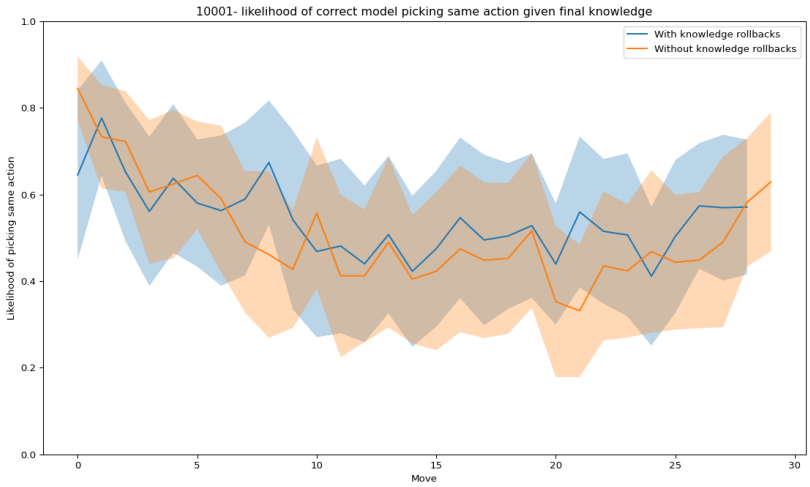


Figure 5.4: From 50 games with agent "10001", how did the knowledge rollback impact the likelihood of the behavior clone of "10001" in **Chief**'s pool picking the same action given the knowledge obtained by the last round. The shading is the variance at each time-step matched by color to the respective line

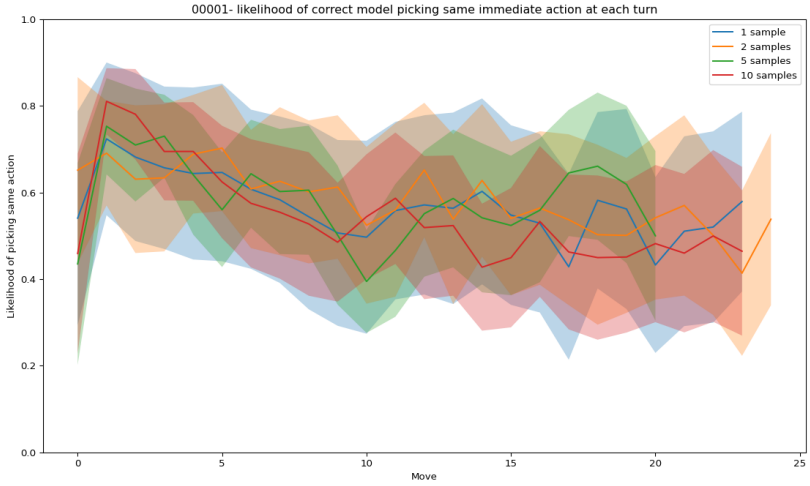


Figure 5.5: From 50 games with agent "00001", how did the number of samples impact the likelihood of the behavior clone of "00001" in **Chief**'s pool picking the same action given the knowledge obtained by that round. The shading is the variance at each time-step matched by color to the respective line

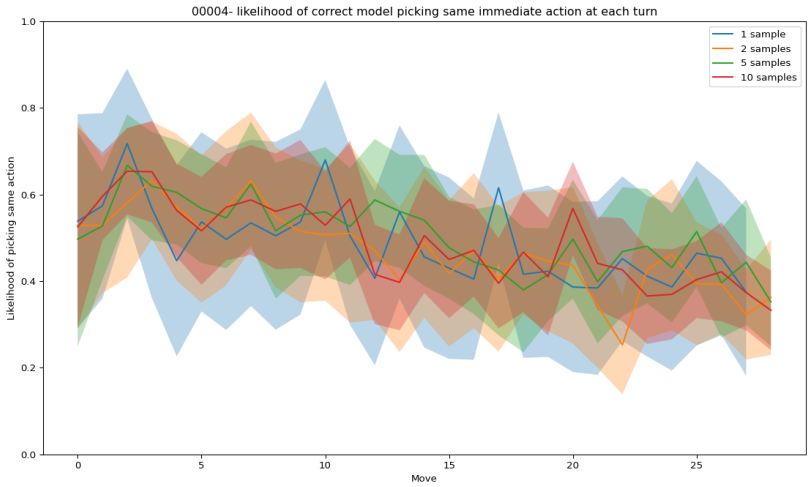


Figure 5.6: From 50 games with agent "00004", how did the number of samples impact the likelihood of the behavior clone of "00004" in **Chief**'s pool picking the same action given the knowledge obtained by that round. The shading is the variance at each time-step matched by color to the respective line

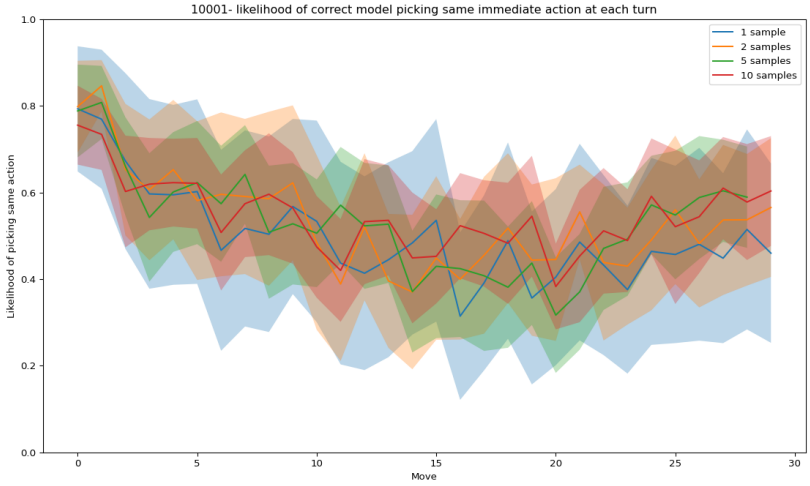


Figure 5.7: From 50 games with agent "10001", how did the number of samples impact the likelihood of the behavior clone of "10001" in **Chief**'s pool picking the same action given the knowledge obtained by that round. The shading is the variance at each time-step matched by color to the respective line

Chapter 6

Teammate Belief Processing

The second layer of uncertainty that **Chief** has to handle is uncertainty over the teammate's strategy. With the knowledge management in place and the model pool filled with a good selection of behavior clones, we now design a system working under those assumptions to reason about the teammate's strategy. Figure 6.1 shows an overview of this system.

Conditional Likelihood of Observed Action

The first feature we need is to be able to calculate how likely each of our models would be to choose the same action as our teammate for each move. Formally, if A_t represents the observed action of the teammate at move t, $O_{1,...,t}$ represents the observations of the game from the start till move t, and M_i is an indicator random variable of whether the model i in our pool is the one we're evaluating, we want to know $Pr[A_t = a | O_{1,...,t} = (o_1, o_2, ..., o_t), M_i = 1]$. This value is complicated to compute if we take in the uncertainty over our observations into account, so we use our knowledge management described in the previous section to approximate it. To make the connection between these two components more clear, we can look at the actual Hanabi environment to see how this comes into play. As mentioned before, when the **Chief** is playing against a teammate, the teammate can see the **Chief**'s hand but **Chief** can't see that same information. However, if we want to try to find the best model for our teammate, and our models work best with full information, we make use of all of our knowledge and sample (as seen in the previous chapter) values for the cards in our hand. This gives our models trained on logical agents full information to reason off of, and our top-level sampling is handling the uncertainty about the cards in our hand.

$$Pr[A_t = a | O_{1,...,t} = (o_1, o_2, ..., o_t), M_i = 1] \approx \frac{1}{N} \sum_{j=1}^{N} Pr[A_t = a | S = s_j, M_i = 1]$$

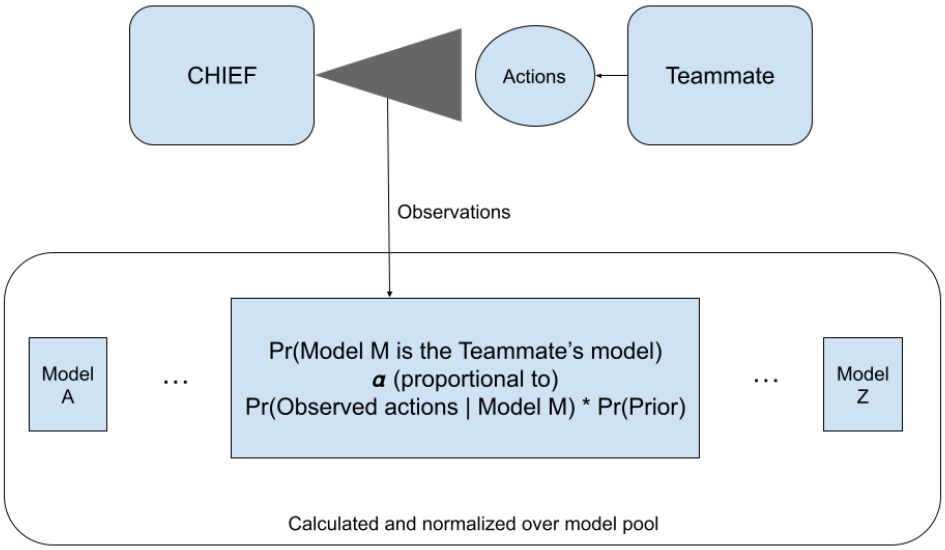


Figure 6.1: An overview of the process of incorporating observed actions from the teammate into beliefs about the teammate's strategy

Here s_j is a sampled sequence of game-states drawn from our knowledge up till this point: $[(h_{1,1}, ..., h_{1,t}), ..., (h_{K,1}, ..., h_{K,t})]$, where K is the total number of hidden features of the game states till now and N is the number of samples taken. As explained before, our knowledge management system makes sure that at time t, we're using the most up-to-date knowledge for all time-steps in the sequence, so our sample s_j is really drawn from $[(h_{1,t}), ..., (h_{K,t})]$. This is a mathematical view of how we estimate conditional likelihood probabilities for each model of producing the observed action. Practically, in the game environment of Hanabi, these hidden variables are just representing the different cards we've held in our hand throughout the game, and the values sampled for them are possible values those cards could have. Of course, if those cards have left our hand by now, we know the value, so the respective h variable can only be assigned that value now.

For our implementation, we achieve this with the following protocol:

Estimate conditional likelihood
(Given model from pool M_i and observed action A)
1. For N runs: compute sample-level likelihood and add to a running sum
2. Return the average over the N runs as our estimated likelihood of model M_i choosing the observed action

Sample-level likelihood (Given M_i, new random seed)
1. Using most recent knowledge for all hidden features encountered till

now, generate sequential sample S

2. For each time-step a, S_a will be a game-state with the relevant hidden features $h_{k,a}$ set with a value sampled from the knowledge for h_k at time T (T is our most recent current time-step)
3. Give this sequence S to M_i which is trained to run on sequential inputs (after formatting the input as needed)
4. Return the probability of A outputted by M_i on S

Model Likelihood for Teammate Behavior

Within our pool of models, we work under the assumption that the model generating the data is in the pool. Obviously, this will mostly never be the case, especially once applied for human collaboration. However, we can work with this assumption because we are just comparing the pool models to each other. The intuition is that the model that is considered the most likely out of our pool to be the teammate, and will be one of the best models to make predictions with. While we cannot give guarantees that it will be the best predictive model, we can use empirical results to convince ourselves that it generally does a good job.

To obtain this probability, we use Bayesian inference. Formally, we want the probability of a model from our pool being the one that produced all the observed actions until now. Because this is inference over our pool, this probability for each model should sum up to one if summed over all the models in the pool. Thus, we can use Bayes rule as so:

$$\begin{aligned} &Pr[M_i = 1 | A_1, ..., A_t, O_1, ..., O_t] \\ &\propto Pr[A_t | M_i, O_1, ..., O_t, A_1, ..., A_{t-1}] * Pr[M_i | O_1, ..., O_t, A_1, ..., A_{t-1}] \end{aligned}$$

These observations are just the visible portions of the game-states and the knowledge about the hidden features. Our active knowledge management work handles all of this, and sets up the problem with all of that as part of the environment, and so we're really just interested in

$$\begin{aligned} &Pr[M_i = 1 | A_1, ..., A_t] \\ &\propto Pr[A_t | M_i, A_1, ..., A_{t-1}] * Pr[M_i | A_1, ..., A_{t-1}] \end{aligned}$$

Therefore, we can set this up with a Bayesian inference system of processing all recorded data. Let's assume that the knowledge is all consistent and available. Then, we can do the following procedure to get this model likelihood at every step.

Compute model likelihoods (Given observed actions and model pool)

1. Repeat for each action: if we are at A_i, estimate the conditional likelihood for each M_x in our pool of producing A_i given our current knowledge and record of game-states using the "Estimate conditional likelihood" function above
2. For each M_x multiply the conditional for the current A_i by the prior probability for M_x from the previous iteration (use uniform distribution if we are at iteration 1)
3. Normalize these products over the models, so that $\sum_x Pr[M_x|A_1, ...A_i] = 1$, and make these the new priors for each M_x

This is the formal procedure, but we can save time by not recomputing likelihoods for rounds where the practical knowledge hasn't actually changed. For example, an update about a card that was drawn in round 3 wouldn't ever change the knowledge **Chief** had about round 1. Therefore, there would be no reason to redo the sampling and simulation of models at that round.

For specificity, the **Chief** stores information for each round of the game. In a single round, there will be M sets of sampled values used for generating the sequential samples for the LSTM models, where M is a preset number of samples. These M sample-sets are unique from those generated for different rounds. This is so that each sequential input to the LSTMs will have exactly one sampled value for any single card, but the sequential inputs will be independent from the other M-1 inputs for this round, as well as the sampled inputs for any other round. Finally, the M sequential inputs for a single round are used to compute the Bayesian update for that round (combining the conditional likelihoods of the observed action from the samples for each model). Then, the model likelihood is computed using the most recent Bayesian updates from each round to do a single (new) run of Bayesian inference.

Figures 6.2, 6.3, 6.4 demonstrates the effectiveness of this belief processing protocol. These results were taken from the same runs as the 5 sample runs in figures 5.5-5.7. Therefore, we are using 5 samples and knowledge rollbacks for this. Here we can see that for agents "00001" and "00004", they are clearly recognized by the **Chief**'s belief processing system very early on in the game. This is a great sign for the **Chief**'s ability to recognize a teammate's deterministic if it has access to a similar stochastic representation. For "10001", we can see that its behavior cloned model in the pool commonly gets mixed up with two other models in the pool, when observing the deterministic "10001" agent. It is important to point out that, just because the **Chief** chooses the wrong model for the teammate does not imply that it has chosen

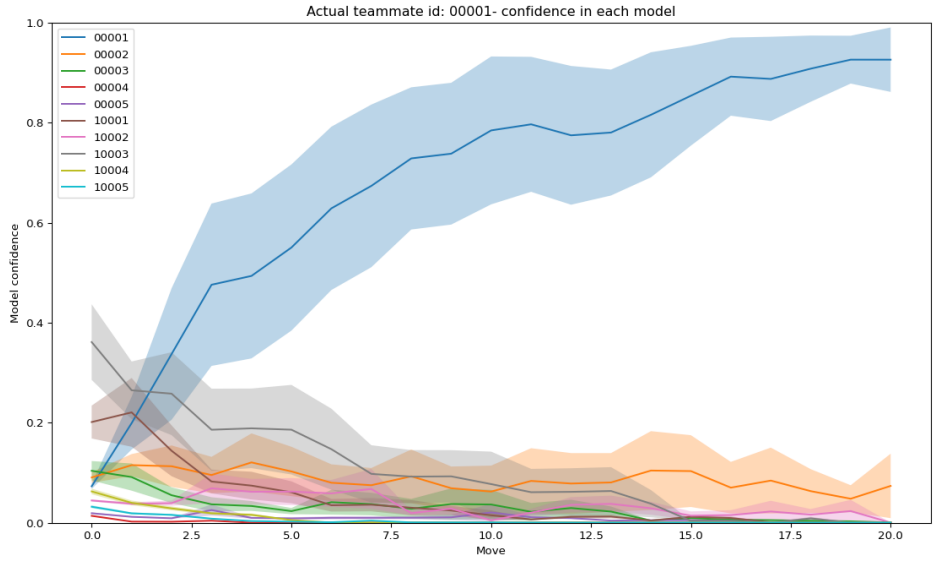


Figure 6.2: The normalized confidence in all ten models at each turn when playing with teammate "00001"

a bad model. However, on the other side of the same coin, choosing the best model from the pool might not guarantee a good model in the real-world deployment of Chief.

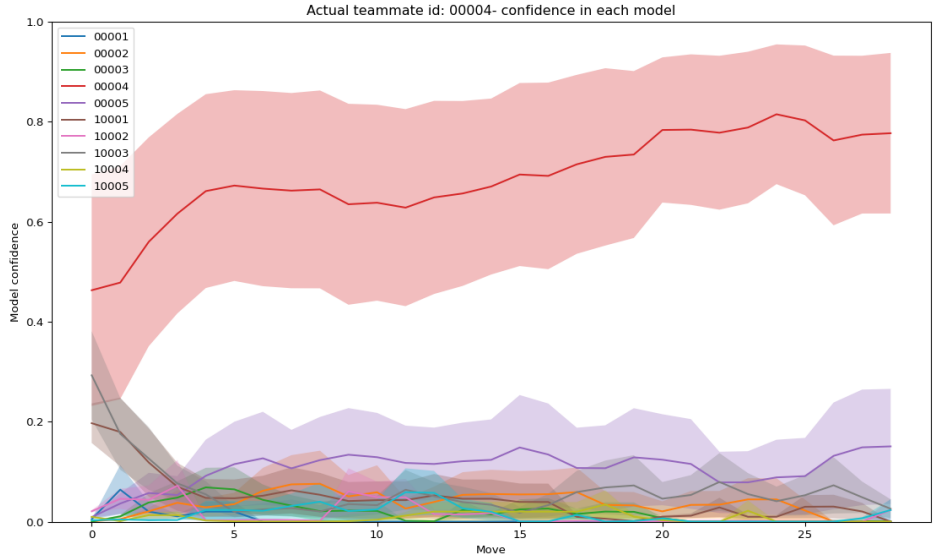


Figure 6.3: The normalized confidence in all ten models at each turn when playing with teammate "00004"

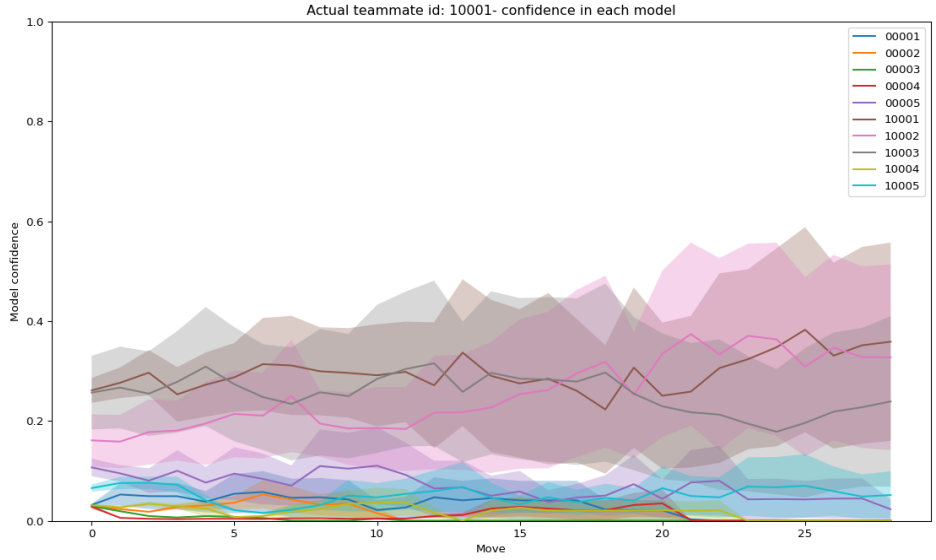


Figure 6.4: The normalized confidence in all ten models at each turn when playing with teammate "10001"

Chapter 7

Main Results

For these results, three setups were run for 50 games each. The first setup was the **Chief** agent being run against a randomly selected agent for a game. All of the agents it could be paired up against had a model in the pool cloned from them. Within each game, we keep track of the **Chief**'s confidence in each model being the model of the teammate over time, and of course the score of each game. The second setup is this, but with a General intuitive model, which is an approximation for a general model created by having the **Chief** always be uniformly confident in all of the models in the pool. Therefore, the action selection is based on an even average of action probabilities over all the models. The final setup was to run the agents with themselves as teammates, representing the idealized self-play scenario. One important note is that the randomized set of 50 choices for a teammate for each game were the same between the three setups.

The overall results were surprising and not as hypothesized. In Figure 7.1, we see that the general intuitive model and the **Chief** essentially perform equally in terms of final score. This is surprising, because it implies that all the processes for developing the model of the teammate were not actually improving the final score. From the error bars, we can see that the **Chief**'s scores seem to be noisier than the other two setups, hinting to one possible area for improvement, which is consistency. Also, when running Welch's t-test on these scores, using scipy.stats.ttest_ind from the scipy python library, we get that the scores for the mirror case are significantly better than the scores for the **Chief** and than the scores for the general case (both comparisons had p-values of around 0.004), but the scores for the **Chief** and general case were not significantly different (p-value of around 0.71). Potential solutions/future investigations for these areas for improvement will be discussed in the next chapter.

One gut check result, which is certainly promising, is that most games have the **Chief** correctly and confidently detecting the model of the teammate. It is important to note that there were 10 models in the pool, so the **Chief** is able to detect which out of the 10 it is playing against frequently.

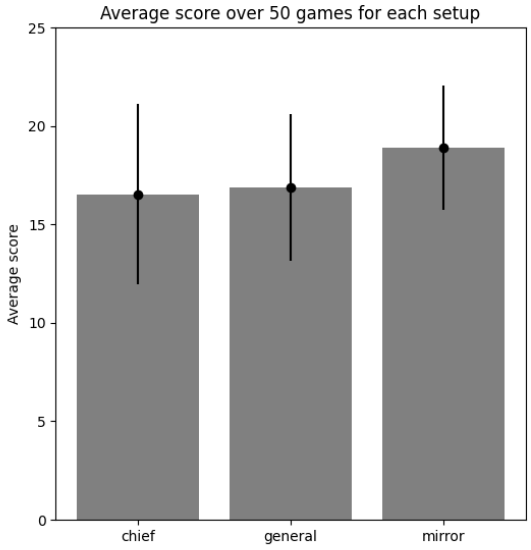


Figure 7.1: The average scores over 50 games for the Chief, a general intuitive model, and idealized self-play

This is visible in 7.2.

In Figures 7.3 and 7.4, we can see that the **Chief**'s confidence in the correct model seems to have no positive impact on the final score, even if it is highly confident halfway through the game. To get more insight into this, I ran the third setup again (with a newly generated sequence of 50 teammates), but had one of the teammates be the behavior cloned model of the other. This would be a secondary idealized case, but limited to the models the **Chief** can actually use to play. Here, the average score was **15.56**, which is around the same level of performance as the **Chief** and General intuitive model. Another interesting insight discovered informally was that certain pairs of different agents play better with each other than one of them does in the mirror match-up.

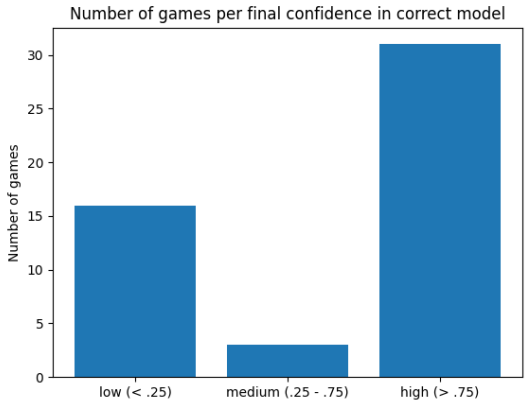


Figure 7.2: The number of games out of 50 for the **Chief** split by the confidence in the correct model the **Chief** had at the end of the game

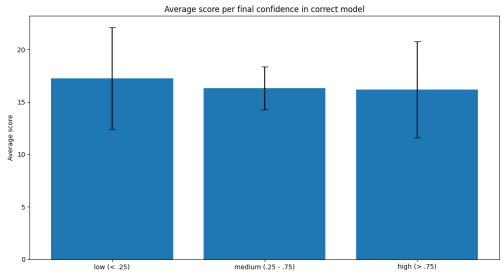


Figure 7.3: The average scores over 50 games for the **Chief** split by the confidence in the correct model the **Chief** had at the end of the game

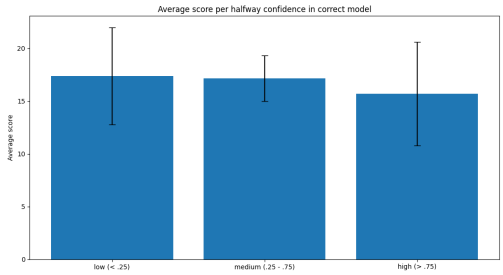


Figure 7.4: The average scores over 50 games for the **Chief** split by the confidence in the correct model the **Chief** had halfway through the game

Chapter 8

Possible Improvements and Future Investigation

While the techniques discussed in Chapter 5 and Chapter 6 seem to have positive results for detecting the correct model for the teammate, there are still some aspects of the **Chief** that need to be tuned and improved. One would be the further iterate the behavior cloned models to achieve high scores with their source counterpart. Another would be to train them to achieve higher scores with the agents that their source has high performing match-ups with. The corresponding change in the **Chief** would be to select the best match-up's action during GET_ACTION rather than the mirrored model's action. Support for this idea can be seen in Table 4.2, where certain match-ups perform better with each other than one of the agents did with a copy of itself.

A valuable future investigation would be to understand more about why the general intuitive model performed just as well as the Chief. The results give some hints to possible explanations. For example, maybe the noisy errors from a single behavior clone are more negative than the positives from high confidence in the correct model. Thus, generalizing over all the models leads to safer play. This could maybe be a possible strategy early in the game, to avoid early errors from noise and play with a general model and, after collecting more data, start using the confidence distribution if there is a model that the **Chief** is highly confident in.

Finally, a very important future investigation, which is already in process, is a user study to understand how this mechanism interacts and cooperates with real people.

Chapter 9

Conclusion

In this document, I detailed the mechanisms behind the **Chief** agent. Using behavior clones to generate stochastic models of different intuitive strategies, we gained strong tools for both modeling teammates and executing actions in the context of the source strategies. We used current knowledge to improve historical inferences, so that the current inference is more reliable over time. Finally, using Bayesian inference to determine model confidence, we have the **Chief** attempt to mirror the teammate it thinks it is playing with. Our results show that the **Chief** is able to perform well at correctly selecting the model of the teammate, indicating that it may be a good way to quickly model human teammates. However, the score does not currently trend upward as hoped with high performing models of the teammate. Thus, with improvements to the execution component of the Chief, this could be a valuable solution in the human-AI cooperation field.

Bibliography

[1] Nolan Bard, Jakob N. Foerster, Sarath Chandar, Neil Burch, Marc Lanctot, H. Francis Song, Emilio Parisotto, Vincent Dumoulin, Subhodeep Moitra, Edward Hughes, Iain Dunning, Shibl Mourad, Hugo Larochelle, Marc G. Bellemare, and Michael Bowling. The hanabi challenge: A new frontier for ai research. *Artificial Intelligence*, 280:103216, 2020. ISSN 0004-3702. doi: https://doi.org/10.1016/j.artint.2019.103216. URL https://www.sciencedirect.com/science/article/pii/S0004370219300116. 2

[2] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to end learning for self-driving cars, 2016. URL https://arxiv.org/abs/1604.07316. 4

[3] Rodrigo Canaan, Haotian Shen, Ruben Torrado, Julian Togelius, Andy Nealen, and Stefan Menzel. Evolving agents for the hanabi 2018 cig competition. In *2018 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 1–8, 2018. doi: 10.1109/CIG.2018.8490449. 2

[4] Rodrigo Canaan, Julian Togelius, Andy Nealen, and Stefan Menzel. Diverse agents for ad-hoc cooperation in hanabi. In *2019 IEEE Conference on Games (CoG)*, pages 1–8, 2019. doi: 10.1109/CIG.2019.8847944. 2

[5] Micah Carroll, Rohin Shah, Mark K. Ho, Thomas L. Griffiths, Sanjit A. Seshia, Pieter Abbeel, and Anca Dragan. On the utility of learning about humans for human-ai coordination, 2019. URL https://arxiv.org/abs/1910.05789. 2

[6] Rohan Choudhury, Gokul Swamy, Dylan Hadfield-Menell, and Anca D. Dragan. On the utility of model learning in hri. In *2019 14th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 317–325, 2019. doi: 10.1109/HRI.2019.8673256. 2

[7] Jacob W. Crandall. When autonomous agents model other agents: An appeal for altered judgment coupled with mouths, ears, and a little more tape. *Artificial Intelligence*, 280:103219, 2020. ISSN 0004-3702. doi: https://doi.org/10.1016/j.artint.2019.103219. URL https://www.sciencedirect.com/science/article/pii/S0004370219300396. 2

[8] Markus Eger, Chris Martens, Pablo Sauma Chacón, Marcela Alfaro Córdoba, and Jeisson Hidalgo-Cespedes. Operationalizing intentionality to play <italic>hanabi</italic> with human players. *IEEE Transactions on Games*, 13(4):388–397, 2021. doi: 10.1109/TG.2020.3009359. 2

[9] Jakob N. Foerster, Francis Song, Edward Hughes, Neil Burch, Iain Dunning, Shimon Whiteson, Matthew Botvinick, and Michael Bowling. Bayesian action decoder for deep multi-agent reinforcement learning, 2018. URL https://arxiv.org/abs/1811.01458. 2

[10] Hengyuan Hu and Jakob N Foerster. Simplified action decoder for deep multi-agent reinforcement learning, 2019. URL https://arxiv.org/abs/1912.02288. 2

[11] Hengyuan Hu, Adam Lerer, Alex Peysakhovich, and Jakob Foerster. "other-play" for zero-shot coordination, 2020. URL https://arxiv.org/abs/2003.02979. 2

[12] Adam Lerer and Alexander Peysakhovich. Learning existing social conventions via observationally augmented self-play, 2018. URL https://arxiv.org/abs/1806.10071. 2

[13] Adam Lerer, Hengyuan Hu, Jakob Foerster, and Noam Brown. Improving policies via search in cooperative partially observable games. 2019. doi: 10.48550/ARXIV.1912.02318. URL https://arxiv.org/abs/1912.02318. 2

[14] Stefanos Nikolaidis and Julie Shah. Human-robot cross-training: Computational formulation, modeling and evaluation of a human team training strategy. In *2013 8th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 33–40, 2013. doi: 10.1109/HRI.2013.6483499. 2

[15] Kazuo Okamura and Seiji Yamada. Adaptive trust calibration for human-ai collaboration. *PLOS ONE*, 15(2):1–20, 02 2020. doi: 10.1371/journal.pone.0229132. URL https://doi.org/10.1371/journal.pone.0229132. 2

[16] Stephane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning, 2011. 4.1b, 4

[17] Joseph Walton-Rivers, Piers R. Williams, Richard Bartle, Diego Perez-Liebana, and Simon M. Lucas. Evaluating and modelling hanabi-playing agents. In *2017 IEEE Congress on Evolutionary Computation (CEC)*, pages 1382–1389, 2017. doi: 10.1109/CEC.2017.7969465. 2